

GOVERNMENT OF TAMILNADU

HIGHER SECONDARY FIRST YEAR

COMPUTER SCIENCE VOLUME-II

A publication under Free Textbook Programme of Government of Tamil Nadu

Department of School Education

Untouchability is Inhuman and a Crime

Government of Tamil Nadu

First Edition - 2018

NOT FOR SALE

Content Creation



State Council of Educational Research and Training © SCERT 2018

Printing & Publishing



Tamil NaduTextbook and Educational Services Corporation

www.textbooksonline.tn.nic.in

Human civilization achieved the highest peak with the development of computer known as "Computer era". Literate are those who have the knowledge in using the computer whereas others are considered illiterate inspite of the other degrees obtained.

T h e growth of the nation at present lies in the hands of the youth, hence the content of this book is prepared in such a way so as to attain utmost knowledge considering the future needs of the youth.

 This book does not require prior knowledge in computer Technology

REFACE

- Each unit comprises of simple activities and demonstrations which can be done by the teacher and also students.
- Technical terminologies are listed in glossary for easy understanding
- The "Do you know?" boxes enrich the knowledge of reader with additional information
- Workshops are introduced to solve the exercises using software applications
- QR codes are used to link supporting additional
- materials in digital form

How to get connected to QR Code?

o Download the QR code scanner from the google play store/ apple app store into your smartphone



HOW

TO USE

THE BOOK

- o Open the QR code scanner application
- Once the scanner button in the application is clicked, camera opens and then bring it closer to the QR code in the textbook.
- o Once the camera detects the QR code, a URL appears in the screen. Click the URL and go to the content page.

CAREER GUIDANCE AFTER 12TH

COURSES	COLLEGES/ UNIVERSITIES	PROFESSION
B.E / B.Tech	All University and their affiliated Colleges and Self financing Colleges in India and Abroad.	Software Engineer, Hardware Engineer, Software Development, Healthcare Section, IT & ITEs
	Science and Humanities	
B.Sc (Computer Science) BCA B.Sc (Maths, Physics, Chemistry, Bio-Chemistry, Geography, journalism, Library Sciences, Political Science, Travel and Tourism)	All University and their affiliated Colleges and Self financing Colleges in India and Abroad.	Government Job and Private Company BPO, Geologist, Journalist
	LAW	
LLB B.A+LLB B.Com BBM+LLB BBA+LLB	All University and their affiliated Colleges and Self financing Colleges in India and Abroad.	Lawyer, Legal Officer, Govt Job
СА	The Institute of Chartered Accountant of India (ICAI)	CA Private and Govt.
Diploma	Government Polytechnic and Self- financing colleges	Junior Engineer (Government and Private)
	Commerce Courses	
B.com-Regular, B.com-Taxation & Tax Procedure, B.com-Travel &Tourism, B.com-Bank Management, B.com-Professional, BBA/BBM-Regular, BFM- Bachelors in Financial Markets, BMS-Bachelors in Management Studies, BAF- Bachelors in Accounting & Finance, Certified Stock Broker & Investment Analysis, Certified Financial Analyst, Certified Financial Planner, Certified Investment Banker	All University and their affiliated Colleges and Self financing Colleges in India and Abroad.	Private Organization , Government ,Banking sectors and prospects for self – employment.

COURSES	COLLEGES/ UNIVERSITIES	PROFESSION
	Management Courses	
Business Management Bank Management Event Management Hospital Management Human Resource Management Logistics Management	All University and their affiliated Colleges and Self financing Colleges in India and Abroad.	Private Organization , Government ,Banking sectors and prospects for self – employment.
	Science and Humanities	
B.Sc.Botany B.Sc.Zoology B.Sc.Dietician & Nutritionist B.Sc.Home Science B.Sc.Food Technology B.Sc.Dairy Technology B.Sc. Hotel Management B.Sc. Fashion Design B.Sc. Mass Communication B.Sc. Multimedia B.Sc3D Animation	All University and their affiliated Colleges and Self financing Colleges in India and Abroad	Government Job and Private Company BPO, Geologist, Journalist

Table of Contents

Chapter No.	Title			
	UNIT III – INTRODUCTION TO C++			
9	Introduction to C++	1		
10	Flow of Control	60		
11	Functions	101		
12	Arrays and Structures	137		
UNIT IV - OBJECT ORIENTED PROGRAMMING WITH C++				
13	Introducton to Object Oriented Programming Techniques	194		
14	Classes and objects	200		
15	Polymorphism	256		
16	Inheritance	280		
UNIT V - COMPUTER ETHICS AND CYBER SECURITY				
17	Computer Ethics And Cyber Security	327		
18	Tamil Computing	342		



E - book



Assessment



DIGI links

Unit III | Introduction to C++

Introduction to C++

CHAPTER

Learning Objectives

After the completion of this chapter, the student will be able to

- Understand the basic building blocks of C++ programming language
- Able to construct simple C++ programs
- Execute and debug C++programs

9.1 Introduction

C++ is one of the most popular programming language developed by Bjarne Stroustrup at AT & T Bell Lab during 1979. C++ supports both procedural and Object Oriented Programming paradigms. Thus, C++ is called as a **hybrid language**. C++ is a superset (extension) of its predecessor C language. Bjarne Stroustrup named his new language as "C with Classes". The name C++ was coined by Rick Mascitti where ++ is the C language increment operator.

> **Bjarne Stroustrup** Inventor of C++ Programming Language



Bjarne is a Danish Computer Scientist was born on 30th December 1950. He has a Master degree in Mathematics and Computer Science in 1975 from Aarhus University, Denmark and Ph.D in Computer Science in 1979 from the University of Cambridge, England.

History of C++

C++ was developed by Bjarne Stroustrup at AT & T Bell Laboratory during 1979. C++ is originally derived from C language and influenced by many languages like Simula, BCPL, Ada, ML, CLU and ALGOL 68. Till 1983, it was referred "New C" and "C with Classes". In 1983, the name was changed as C++ by Rick Mascitti.



standardized C++ is bv the International Organization for Standardization (ISO). The latest standard version published in December 2017 as ISO/IEC 14882:2017 which is informally known as C++17. The first standardized version was published in 1998 as ISO/IEC 14882:1998 which was then enhanced by the C++03 (ISO/IEC 14882:2003), C++11 (ISO/IEC 14882:2011) and C++14 (ISO/ IEC 14882:2014). The Next standard version will be C++20 in 2020.

C# (C-Sharp), D, Java and newer versions of C languages has been influenced by C++.

Benefits of learning C++

- C++ is a highly portable language and is often the language of choice for multi-device, multi-platform app development.
- C++ is an object-oriented programming language and includes classes, inheritance, polymorphism, data abstraction and encapsulation.
- C++ has a rich function library.
- C++ allows exception handling, inheritance and function overloading which are not possible in C.
- C++ is a powerful, efficient and fast language. It finds a wide range of applications – from GUI applications to 3D graphics for games to real-time mathematical simulations.

9.2 Character set

Character set is a set of characters which are allowed to write a C++ program. A character represents any alphabet, number or any other symbol (special characters) mostly available in the keyboard. C++ accepts the following characters.

Most of the Character set, Tokens and expressions are very common to C based programming languages like C, C++, Java, PHP etc.,

Alphabets	A Z, a z
Numeric	0 9
Special	+ - * / ~ ! @ # \$ % ^& [] (
Characters) { } = >< _ \ ? . , : ```;
White space	Blank space, Horizontal tab (\rightarrow), Carriage return (\downarrow), Newline, Form feed
Other characters	C++ can process any of the 256 ASCII characters as data.

9.3 Lexical Units (Tokens):

C++ program statements are constructed by many different small elements such as commands, variables, constants and many more symbols called as operators and punctuators. These individual elements are collectively called as **Lexical units** or **Lexical elements** or **Tokens**. C++ has the following tokens:

- Keywords
 - Identifiers
- Literals Operators
- Punctuators

TOKEN:

The smallest individual unit in a program is known as a Token or a Lexical unit

9.3.1 Keywords

Keywords are the reserved words which convey specific meaning to the C++ compiler. They are the essential elements to construct C++ programs. Most of the keywords are common to C, C++ and Java.

C++ is a case sensitive programming language so, all the keywords must be in lowercase.

asm	auto	break	case	catch
char	class	const	continue	default
delete	do	double	else	enum
extern	float	for	friend	goto
if	inline	int	long	new
operator	private	protected	public	register
return	short	signed	sizeof	static
struct	switch	template	this	throw
try	typedef	union	unsigned	virtual
void	volatile	while		

Table 9.1 C++ Keywords

• With revisions and additions, the recent list of keywords also includes:

using, namespace, bal, static_cast, const_cast, dynamic_cast, true, false

• Identifiers containing a double underscore are reserved for use by C++ implementations and standard libraries and should be avoided by users.

9.3.2 Identifiers

Identifiers are the user-defined names given to different parts of the C++ program viz. variables, functions, arrays, classes etc., These are the fundamental building blocks of a program. Every language has specific rules for naming the identifiers.

Rules for naming an identifier:

• The first character of an identifier must be an alphabet or an underscore (_).

- Only alphabets, digits and underscore are permitted. Other special characters are not allowed as part of an identifier.
- C++ is case sensitive as it treats upper and lower-case characters differently.
- Reserved words or keywords cannot be used as an identifier name.

As per ANSI standards, C++ places no limit on its length and therefore all the characters are significant.

Idontificano	Valid /	Reason for
Identifiers	Invalid	invalid
Num	Valid	
NUM	Valid	
_add	Valid	
total_sales	Valid	
tamilMark	Valid	
		Contains
num-add	Invalid	special
		character (-)
		This is one of
		the keyword.
this	Invalid	Keyword
		cannot be used
		as identifier
		names.
		Name must
2myfile		start begins
	Invalid	with an
		alphabet or an
		underscore

• You may use an underscore in variable names to separate different parts of the name (eg: *total_sales* is a valid identifier where as the variable called *total sales* is an invalid identifier). • You may use capital style notation, such as *tamilMark* ie. capitalizing the first letter of the second word.

9.3.3 Literals (Constants)

Literals are data items whose values do not change during the execution of a program. Therefore Literals are called as Constants. C++ has several kinds of literals:



Figure 9.1 Types of Constants

Numeric Constants:

As the name indicates, the numeric constants are numeric values, which are used as constants. Numeric constants are further classified as:

- 1. Integer Constants (or) Fixed point constants.
- 2. Real constants (or) Floating point constants.

(1) Integer Constants (or) Fixed point constants

Integers are whole numbers without any fractions. An integer constant must have at least one digit without a decimal point. It may be signed or unsigned. Signed integers are considered as negative, commas and blank spaces are not allowed as part of it. In C++, there are three types of integer constants: (i) Decimal (ii) Octal (iii) Hexadecimal

(i) Decimal

Any sequence of one or more digits (0 9)

Valid	Invalid
725	7,500 (Comma is not allowed)
-27	66 5(Blank space is not allowed)
4.56	9\$ (Special Character not allowed)

If you assign **4.56** as an integer decimal constant, the compiler will accept only the integer portion of **4.56** ie. **4**. It will simply ignore **.56**.



If a Decimal constant declared with fractions, then the compiler will take only the integer part of the value and it will ignore its fractional part. This is called as "Implicit Conversion". It will be discussed later.

(ii) Octal

Any sequence of one or more octal values $(0 \dots 7)$ that begins with 0 is considered as an Octal constant.

Valid	Invalid
012	05,600(Commas is not allowed)
027	04.56 (Decimal point is not
-027	allowed)**
+ 0.22.1	0158 (8 is not a permissible digit
+0231	in octal system)

** When you use a fractional number that begins with 0, C++ has consider the number as an integer not an Octal.

(iii) Hexadecimal

Any sequence of one or more Hexadecimal values (0 9, A F) that starts with 0x or 0Xis considered as an Hexadecimal constant.

Valid	Invalid
0x123	0x1,A5 (Commas is not allowed)
0X568	0x.14E (Decimal point is not allowed like this)

The suffix L or l and U or u added with any constant forces that to be represented as a long or unsigned constant respectively.

(2) Real Constants (or) Floating point constants

A real or floating point constant is a numeric constant having a fractional component. These constants may be written in fractional form or in exponent form.

Fractional form of a real constant is a signed or unsigned sequence of digits including a decimal point between the digits. It must have at least one digit before and after a decimal point. It may have prefix with + or - sign. A real constant without any sign will be considered as positive.

Exponent form of real constants consists of two parts: (1) Mantissa and (2) Exponent. The mantissa must be either an integer or a real constant. The mantissa followed by a letter E or e and the exponent. The exponent should also be an integer.

For example, 58000000.00 may be written as 0.58×10^8 or 0.58E8.

Mantissa	Exponent
(Before E)	(After E)
0.58	8

Example:

5.864 E-1	→	5.864×10^{1}	→	58.64
5864 E-2	→	5864×10^{-2}	→	58.64
0.5864 E-2	→	0.5864×10^{2}	→	58.64

Boolean Literals

Boolean literals are used to represent one of the Boolean values(True or false). Internally true has value 1 and false has value 0.

Character constant

A character constant is any valid single character enclosed within single quotes. A character constant in C++ must contain one character and must be enclosed within a single quote.

Valid character constants : 'A', '2', '\$'

Invalid character constants : "A"

The value of a single character constant has an equivalent ASCII value. For example, the value of 'A' is 65.

Escape sequences (or) Non-graphic characters

C++ allows certain non-printable characters represented as character constants. Non-printable characters are also called as non-graphical characters. Nonprintable characters are those characters that cannot be typed directly from a keyboard during the execution of a program in C++, for example: backspace, tabs etc. These nonprintable characters can be represented by using escape sequences. An escape sequence is represented by a backslash followed by one or two characters.

Table 9.2 Escape	Sequences
------------------	-----------

Escape sequence	Non-graphical character
\a	Audible or alert bell
\ b	Backspace
\f	Form feed
\n	Newline or linefeed
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
//	Backslash
\'	Single quote
\"	Double quote
/;	Question Mark
\On	Octal number
\xHn	Hexadecimal number
\0	Null

Even though an escape sequence contains two characters, they should be enclosed within single quotes because, C++ consider escape sequences as character constants and allocates one byte in ASCII representation. ASCII (American Standard Code for Information Interchange) was first developed and published in 1963 by the X3 committee, a part of the American Standards Association (ASA).

String Literals

Sequence of characters enclosed within double quotes are called as String literals. By default, string literals are automatically added with a special character '\0' (Null) at the end. Therefore, the string "welcome" will actually be represented as "welcome\0" in memory and the size of this string is not 7 but 8 characters i.e., inclusive of the last character \0.

Valid string Literals : "A", "Welcome" "1234"

Invalid String Literals : 'Welcome', '1234'

Evaluate Yourself

- What is meant by literals? How many types of integer literals available in C++?
- 2. What kind of constants are following?
 - i) 26 ii) 015 iii) 0xF iv) 014.9
- 3. What is character constant in C++?
- 4. How are non graphic characters represented in C++?
- 5. Write the following real constants into exponent form: i) 32.179ii) 8.124 iii) 0.00007
- 6. Write the following real constants into fractional form: i) 0.23E4ii) 0.517E-3 iii) 0.5E-5
- 7. What is the significance of null (\0) character in a string?

9.3.4 Operators

The symbols which are used to do some mathematical or logical operations are called as **"Operators"**. The data items or values that the operators act upon are called as **"Operands"**.



In C++, The operators are classified on the basis of the number of operands.

- (i) **Unary Operators** Require only one operand
- (ii) **Binary Operators** Require two operands
- (iii) **Ternary Operators** Require three operands
- C++ Operators are classified as:
 - (1) Arithmetic Operators
 - (2) Relational Operators
 - (3) Logical Operators
 - (4) Bitwise Operators
 - (5) Assignment Operators
 - (6) Conditional Operator
 - (7) Other Operators

(1) Arithmetic Operators

Arithmetic operators perform simple arithmetic operations like addition, subtraction, multiplication, division etc.,

Operator	Operation	Example
+	Addition	10 + 5 = 15
-	Subtraction	10 - 5 = 5
*	Multiplication	10 * 5 = 50
		10 / 5 = 2
/	Division	(Quotient of
		the division)
	Modulus	10 % 3 =
0/	(To find the	1(Remainder
/0	reminder of a	of the
	division)	division)

• The above mentioned arithmetic operators are binary operators which requires minimum of two operands.

Increment and Decrement Operators

++ (Plus, Plus) Increment operator

-- (Minus, Minus) Decrement operator

An increment or decrement operator acts upon a single operand and returns a new value. Thus, these operators are unary operators. The increment operator adds 1 to its operand and the decrement operator subtracts 1 from its operand. For example,

- x++ is the same as x = x+1;
 It adds 1 to the present value of x
- x-- is the same as to x = x-1;
 It subtracts 1 from the present value of x

The ++ or -- operators can be placed either as prefix (before) or as postfix (after) to a variable. With the prefix version, C++ performs the increment / decrement before using the operand.

For example: **N1=10**, **N2=20**;

$$S = ++N1 + ++N2;$$

The following Figure explains the working process of the above statement.



Figure 9.2 Prefix Increment Working process

In the above example, the value of num is first incremented by 1, then the incremented value is assigned to the respective operand.

With the postfix version, C++ uses the value of the operand in evaluating the expression before incrementing / decrementing its present value.

For example: **N1=10**, **N2=20**;

$$S = N1 + + + + N2;$$

The following Figure explains the working process of the above statement.



Figure 9.3 Postfix Increment Working process

In the above example, the value assigned to operand N1 is taken into consideration, first and then the value will be incremented by 1.

Operator	Operation	Example (Assume n=2; what will be value of a)					
		Prefix	Postfix				
	Increment	a=++n;	a=n++;				
++	merement	Value of $a = 3$	Value of $a = 2$				
	Docromont	a=n;	a=n;				
	Decrement	Value of a=1	Value of a=2				

(2) Relational Operators

Relational operators are used to determine the relationship between its operands. When the relational operators are applied on two operands, the result will be a Boolean value i.e 1 or 0 to represents True or False respectively. C++ provides six relational operators. They are,

Operator	Operation	Example
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b
==	Equal to	a == b
!=	Not equal	a != b

- In the above examples, the operand *a* is compared with *b* and depending on the relation, the result will be either 1 or 0. i.e., 1 for true, 0 for false.
- All six relational operators are binary operators.

(3)Logical Operators

A logical operator is used to evaluate logical and relational expressions. The logical operators act upon the operands that are themselves called as logical expressions. C++ provides three logical operators.

Operator	Operation	Description
		The logical AND combines two different relational
&&	AND	expressions in to one. It returns 1 (True), if both
		expression are true, otherwise it returns 0 (false).
		The logical OR combines two different relational
П	OP	expressions in to one. It returns 1 (True), if either
II	UK	one of the expression is true. It returns 0 (false), if
		both the expressions are false.
		NOT works on a single expression / operand. It
,	NOT	simply negates or inverts the truth value. i.e., if an
÷	NOT	operand / expression is 1 (true) then this operator
		returns 0 (false) and vice versa

• AND, OR both are binary operators where as NOT is an unary operator.

Example: a = 5, b = 6, c = 7;

Expression	Result
(a <b) &&="" (b<c)<="" td=""><td>1 (True)</td></b)>	1 (True)
(a>b) && (b <c)< td=""><td>0 (False)</td></c)<>	0 (False)
(a <b) (b="" ="">c)</b)>	1 (True)
!(a>b)	1 (True)

(4) Bitwise Operators

Bitwise operators work on each bit of data and perform bit-by-bit operation. In C++, there are three kinds of bitwise operators, which are:

- (i) Logical bitwise operators
- (ii) Bitwise shift operators
- (iii) One's compliment operators

(i) Logical bitwise operators:

&	Bitwise AND	(Binary AND)
	Bitwise OR	(Binary OR)
Λ	Bitwise Exclusive OR	(Binary XOR)

- Bitwise AND (&) will return 1 (True)if both the operands are having the value 1 (True); Otherwise, it will return 0 (False)
- Bitwise OR (|) will return 1 (True) if any one of the operands is having a value 1 (True); It returns 0 (False) if both the operands are having the value 0 (False)
- Bitwise XOR (^) will return 1 (True) if only one of the operand is having a value 1 (True). If both are True or both are False, it will return 0 (False).

Α	В	A & B	A B	A ^ B
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Truth table for bitwise operators (AND, OR, XOR)

Example:

If a = 65, b=15

Equivalent binary values of 65 = 0100 0001; 15 = 0000 1111

Operator	Operation		Result							
		а	0	1	0	0	0	0	0	1
	. 0 1	b	0	0	0	0	1	1	1	1
X		a&b	0	0	0	0	0	0	0	1
	$(a\&b) = 0000\ 0001_2 = 1_{10}$									
	a b	а	0	1	0	0	0	0	0	1
		b	0	0	0	0	1	1	1	1
		a b	0	1	0	0	1	1	1	1
	$(a b) = 01001111_2 = 79_{10}$									
~		а	0	1	0	0	0	0	0	1
	• • b	b	0	0	0	0	1	1	1	1
		a ^ b	0	1	0	0	1	1	1	0
			$(a^b) = 0100\ 1110_2 = 78_{10}$							

(ii) The Bitwise shift operators:

There are two bitwise shift operators in C++, **Shift left** (<<) and **Shift right** (>>).

Shift left (<<)– The value of the left operand is moved to left by the number of bits specified by the right operand. Right operand should be an unsigned integer.

Shift right (>>)– The value of the left operand is moved to right by the number of bits specified by the right operand. Right operand should be an unsigned integer.

Example:

Operator	Operation		Result							
	a	0	0	0	0	1	1	1	1	
	2.1.2		· · · · · · · · · · · · · · · · · · ·							
<<	a << 5	a << 3	0	1	1	1	1	0	0	0
		$(a << 3) = 01111000_2 = 120_{10}$								
		а	0	0	0	0	1	1	1	1
>>	a >> 2	a>> 2	0	0	0	0	1	1	0	0
				(a>> 2	2) = 00	00 00	$1_{2} = 3_{1}$	I	I	

If a =15; Equivalent binary value of a is 0000 1111

(iii) The Bitwise one's compliment operator:

The bitwise One's compliment operator ~(Tilde), inverts all the bits in a binary pattern, that is, all 1's become 0 and all 0's become 1. This is an unary operator.

Example:

If a =15; Equivalent binary values of a is 0000 1111

Operator	Operation		Result							
	(~2)	a	0	0	0	0	1	1	1	1
~			•							
	("	(~a)	1	1	1	1	0	0	0	0
				(~a) =	= 1111	00002	= -16 ₁₀)		

(5) Assignment Operator:

Assignment operator is used to assign a value to a variable which is on the left hand side of an assignment statement. = (equal to) is commonly used as the assignment operator in all computer programming languages. This operator copies the value at the right side of the operator to the left side variable. It is also a binary operator.



C++ uses different types of assignment operators. They are called as Shorthand assignment operators.

Operator	Name of Operator	Example
+=	Addition Assignment	a = 10; c = a += 5; (ie, $a = a + 5$) c = 15
-=	Subtraction Assignment	a = 10; c = a -= 5; (ie. $a = a - 5)c = 5$
*=	Multiplication Assignment	a = 10; c = a *= 5; (ie. a = a * 5) c = 50
/=	Division Assignment	a = 10; c = a /= 5; (ie. a = a / 5) c = 2
%=	Modulus Assignment	a = 10; c = a %= 5; (ie. a = a % 5) c = 0

Discuss the differences between = and == operators

(6) Conditional Operator:

In C++, there is only one conditional operator is used. **?:** is a conditional Operator. This is a Ternary Operator. This operator is used as analternate to if ... else control statement. We will learn more about this operator in later chapters along with if else structure.

(7) Other Operators:

The Comma operator	Comma (,) is an operator in C++ used to string together several expressions. The group of expression separated by comma is evaluated from left to right.		
Sizeof	This is called as compile time operator. It returns the size of a variable in bytes.		
Pointer	* Pointer to a variable& Address of		
Component selection	 Direct component selector -> Indirect component selector 		
Class member operators	::Scope access / resolution.*Dereference->*Dereference pointer to class member		

Precedence of Operators:

Operators are executed in the order of precedence. The operands and the operators are grouped in a specific logical way for evaluation. This logical grouping is called as an Association.

The order of precedence:

()[]	Operators within parenthesis are performed first	Higher
++,	Postfix increment / decrement	
++,	Prefix increment / decrement	
*, /, %	Multiplication, Division, Modulus	
+, -	Addition, Subtraction	
<, <=, >, >=	Less than, Less than or equal to, Greater than, Greater than or equal to	
==, !=	Equal to, Not equal to	
&&	Logical AND	
	Logical OR	
?:	Conditional Operator	
=	Simple Assignment	
+=, -=, *=, /=	Shorthand operators	•
>	Comma operator	Lower



In C++, one or two operators may be used in different places with different meaning. For example: Asterisk (*) is used for multiplication as well as for pointer to a variable.

9.3.5 Punctuators

Punctuators are symbols, which are used as delimiters, while constructing a C++ program. They are also called as "Separators". The following punctuators are used in C++; most of these symbols are very similar to C and Java.

Separator	Description	Example
Curly braces	Opening and closing curly braces indicate the start and the end of a block of code. A block of code containing more than one executable statement. These statements together are called as "compound statement"	<pre>int main () { int x=10, y=20, sum; sum = x + y; cout << sum; }</pre>
Parenthesis	Opening and closing parenthesis indicate function calls and function parameters.	clrscr(); int main ()
Square brackets []	It indicates single and multidimensional arrays.	int num[5]; char name[50];
Comma ,	It is used as a separator in an expression	int x=10, y=20, sum;
Semicolon ;	Every executable statement in C++ should terminate with a semicolon	<pre>int main () { int x=10, y=20, sum; sum = x + y; cout << sum; }</pre>
Colon :	It is used to label a statement.	private:
Comments // /* */	Any statement that begins with // are considered as comments. Comments are simply ignored by compilers. i.e., compiler does not execute any statement that begins with a // // Single line comment /**/ Multiline comment	<pre>/* This is written By myself to learn CPP */ int main () { int x=10, y=20, sum; // to sum x and y sum = x + y; cout << sum; }</pre>

Evaluate Yourself

- 1. What is use of operators?
- 2. What are binary operators? Give examples arithmetic binary operators.
- 3. What does the modulus operator % do?
- 4. What will be the result of 8.5 % 2?
- 5. Assume that R starts with value 35. What will be the value of S from the following expression? S=(R--)+(++R)
- 6. What will be the value of j = -k + 2k. if k is 20 initially ?
- 7. What will be the value of $p = p^* + j$ where j is 22 and p = 3 initially?
- 8. Give that i = 8, j = 10, k = 8, What will be result of the following expressions?
 (i) i < k (ii) i < j (iii) i > = k (iv) i = = j (v) j ! = k
- 9. What will be the order of evaluation for the following expressions?
 (i) i + 3 >= j 9
 (ii) a +10

10. Write an expression involving a logical operator to test, if marks are 75 and grade is 'A'.

9.4 I/O Operators

9.4.1 Input operator:

C++ provides the operator >> to get input. It extracts the value through the keyboard and assigns it to the variable on its right; hence, it is called as **"Stream extraction"** or **"get from"** operator.

It is a binary operator i.e., it requires two operands. The first operand is the pre-defined identifier cin (pronounced as C-In) that identifies keyboard as the input device. The second operand must be a variable.



Figure 9.4 Working process of cin

To receive or extract more than one value at a time, >> operator should be used for each variable. This is called cascading of operator.

Example:

cin >> num;	Pre-defined object cin extracts a value typed on keyboard and stores it in variable num.
cin >>x >> y;	This is used to extract two values. cin reads thefirst value and immediately assigns that to variable x; next, it reads the second value which is typed after a space and assigns that to y. Space is used as a separator for each input.

9.4.2 Output Operator:

C++ provides << operator to perform output operation. The operator << is called the **"Stream insertion"** or **"put to"** operator. It is used to send the strings or values of the variables on its right to the object on its left. << is a binary operator.

The first operand is the pre-defined identifier cout (pronounced as C-Out) that identifies monitor as the standard output object. The second operand may be a constant, variable or an expression.



Figure 9.5 Working process of cout

To send more than one value at a time, << operator should be used for each constant/ variable/expression. This is called **cascading of operator**.

Example:

cout << "Welcome";	Pre-defined object cout sends the given string "Welcome" to screen.		
cout << "The sum = " << sum;	First, cout sends the string "The Sum = " to the screen and then sends the value of the variable sum; Usually, cout sends everything specified within double quotes or single quotes i.e., string or character constants, except non-graphical characters.		
cout <<"\n The Area: " <<3.14*r*r;	First, cout sends everything specified within double quotes except \n to the screen, and then it evaluates the expression 3.14^*r^*r and sends the result to the screen. \n – is a non graphical character constant to feed a new line.		
cout << a + b ;	cout sends the sum of a and b to the output console (monitor)		

9.4.3. Cascading of I/O operators

The multiple use of input and output operators such as >> and << in a single statement is known as cascading of I/O operators.

Cascading cout:

int Num=20;

cout << "A=" << Num;

The Figure 9.6 is used to understand the working of Cascading cout statement



Figure 9.6 Cascading cout 18

Cascading cin - Example:

cout >> "Enter two number: ";

cin >> a >> b;

The Figure 9.7 is used to understanding the working of Cascading cin statement



Figure 9.7 Cascading cin

9.5 Sample program – A first look at C++ program

Let us start our first C++ program that prints a string "Welcome to Programming in C++" on the screen.



The above program produces, the following output:

Welcome to Programming in C++

This is very simple C++ program which includes the basic elements that every C++ program has. Let us have a look at these elements:

1 // C++ program to print a string

This is a comment statement. Any statement that begins with // are considered as comments. Compiler does not execute any comment as part of the program and it simply ignores. If we need to write multiple lines of comments, we can use /* */.

2 # include <iostream>

Usually all C++ programs begin with include statements starting with a # (hash / pound). The symbol # is a directive for the preprocessor. That means, these statements are processed before the compilation process begins.

#include <iostream> statement tells the compiler's preprocessor to include the header file "iostream" in the program.

The header file iostream should includ in every C++ program to implement input / output functionalities.

In simple words, iostream header file contains the definition of its member objects cin and cout. If you fail to include iostream in your program, an error message will occur on cin and cout; and we will not be able to get any input or send any output.

3 using namespace std;

The line using namespace std; tells the compiler to use standard namespace. Namespace collects identifiers used for class, object and variables. Namespaces provide a method of preventing name conflicts in large projects. It is a new concept introduced by the ANSI C++ standards committee.

4 int main ()

6

7 8 L

C++ program is a collection of functions. Every C++ program must have a main function. The main() function is the starting point where all C++ programs begin their execution. Therefore, the executable statements should be inside the main() function.

5 ⊟ { cout << "Welcome to Programming in C++";</pre> return 0; •

The statements between the curly braces (Line number 5 to 8) are executable statements. This is actually called as a block of code. In line 6, cout simply sends the string constant "Welcome to Programming in C++" to the screen. As we discussed already, every executable statement must terminate with a semicolon. In line 7, return is a keyword which is used to return the value what you specified to a function. In this case, it will return 0 to main() function.

```
9.6 Execution of C++ program:
```

For creating and executing a C++ program, one must follow four important steps.

(1) Creating Source code

Creating includes typing and editing the valid C++ code as per the rules followed by the C++ Compiler.

(2) Saving source code with extension .cpp

After typing, the source code should be saved with the extension .cpp

(3) Compilation

This is an important step in constructing a program. In compilation, compiler links the library files with the source code and verifies each and every line of code. If any mistake or error is found, it will inform you to make corrections. If there are no errors, it translates the source code into machine readable object file with an extension .obj

(4) execution

This is the final step of construction of a C++ Program. In this stage, the object file becomes an executable file with extension .exe. Once the program becomes an executable file, the program has an independent existence. This means, you can run your application without the help of any compiler or IDE.





9.7 C++ Development Environment

There are lot of IDE programs available for C++. IDE makes it easy to create, compile and execute a C++ program. Most of the IDEs are open source applications (ie.) that are available in free of cost.

9.7.1 Familiar C++ Compilers with IDE

Compiler	Availability
Dev C++	Open source
Geany	Open source
Code::blocks	Open source
Code Lite	Open source
Net Beans	Open source
Digital Mars	Open source
Sky IDE	Open source
Eclipse	Open source

Table 9.4 Open Source Compilers

9.7.2 Working with Dev C++

Among the dozens of IDEs, we take "Dev C++" compiler to create C++ programs. Programming techniques and illustrated programs of this book are based on "Dev C++" compiler.

Dev C++ is an open source, cross platform (alpha version available for Linux), full featured Integrated Development Environment (IDE) distributed with the GNU General Public License for programming in C and C++. It is written in Delphi. It can be downloaded from *http://www.bloodshed.net/dev/devcpp.html*

1. After installation **Dev C++** icon is available on the desktop. Double click to open IDE. Dev C++ IDE appears as given below.



Figure 9.9 Dev C++ opening Window

- 2. To create a source file, Select File \rightarrow New \rightarrow Source file or Press Ctrl + N.
- 3. On the screen that appears, type your C++ program, and save the file by clicking **File** → **Save** or Pressing **Ctrl** + **S**. It will add .cpp by default at the end of your source code file. No need to type .cpp along with your file name.



Figure 9.10 Dev C++ IDE with a program

4. After save, Click Execute \rightarrow Compile and Run or press F11 key.

If your program contains any error, it displays the errors under **compile log**. If your program is without any error, the display will appear as follows.



5. After successful compilation, output will appear in output console, as follows



Figure 9.12 Dev C++ Output Window

9.8 Types of Errors

Some common types of errors are given below:

Type of Error	Description		
Syntax Error	 Syntax is a set of grammatical rules to construct a program. Every programming language has unique rules for constructing the sourcecode. Syntax errors occur when grammatical rules of C++ are violated. Example: if you type as follows, C++ will throw an error. cout << "Welcome to Programming in C++" As per grammatical rules of C++, every executable statement should terminate with a semicolon. But, this statement does not end with a semicolon. 		
Semantic Error	• A Program has not produced expected result even though the program is grammatically correct. It may be happened by wrong use of variable / operator / order of execution etc. This means, program is grammatically correct, but it contains some logical error. So, Semantic error is also called as "Logic Error" .		
Run-time error	 A run time error occurs during the execution of a program. It is occurs because of some illegal operation that takes place. For example, if a program tries to open a file which does not exist, it results in a run-time error 		

Points to Remember:

- C++ was developed by Bjarne Stroustrup at AT & T Bell Labs during the year 1979.
- Character set is the set of characters which are allowed to write C++ programs.
- Individual elements are collectively called as Lexical units or Lexical elements or Tokens.
- Keywords are the reserved words that convey specific meaning to the C++ compiler.
- Identifiers are user-defined names given to different parts of the C++ program viz. variables, functions, arrays, classes etc.,

- Literals are data items whose values do not change during the execution of a program. Therefore, Literals are called as Constants.
- There are different kinds of literals used in C++ (Integer, Float, Character, String)
- The symbols which are used to do some mathematical, logical operations are called as "Operators".
- Punctuators are symbols, which are used as delimiters in constructing C++ programs. They are also called as "Separators".
- Extraction operator(>>) and Insertion operator (<<) are used to get input and send output in C++.



• Type the following C++ Programs in Dev C++ IDE and execute. if compiler shows any errors, try to rectify it and execute again and again till you get the expected result.

```
1. C++ Program to find the total marks of three subjects
#include <iostream>
using namespace std;
int main()
{
    int m1, m2, m3, sum;
    cout << "\n Enter Mark 1: ";
    cin >> m1;
    cout << "\n Enter Mark 2: ";
    cin >> m2;
    cout << "\n Enter Mark 3: ";
    cin >> m3;
    sum = m1 + m2 + m3;
    cout << "\n The sum = " << sum;
}</pre>
```

• Make changes in the above code to get the average of all the given marks.

2. C++ program to find the area of a circle

```
#include <iostream>
using namespace std;
int main()
{
    int radius;
    float area;
    cout << "\n Enter Radius: ";
    cin >> radius;
    area = 3.14 * radius * radius;
    cout << "\n The area of circle = " << area;</pre>
```

```
}
```

```
3. point out the errors in the following program:
```

```
Using namespace std;
int main()
{
cout << "Enter a value ";
```

cin << num1 >> num2
num+num2=sum;
cout >> "\n The Sum=" >> sum;

4. point out the type of error in the following program:

#include <iostream>
using namespace std;
int main()
{

}

int h=10; w=12; cout << "Area of rectangle " << h+w;</pre>

Evaluation	

PART – I

Choose the corret answer.

1.	Who	develo	ped	C++3	?
. .	1110	40,010	p c c	0.1.1	•

- (a) Charles Babbage (b) Bjarne Stroustrup
- (c) Bill Gates (d) Sundar Pichai
- 2. What was the original name given to C++?
 - (a) CPP (b) Advanced C
 - (c) C with Classes (d) Class with C
- 3. Who coined C++?
 - (a) Rick Mascitti (b) Rick Bjarne
 - (c) Bill Gates (d) Dennis Ritchie
- 4. The smallest individual unit in a program is:
 - (a) Program(b) Algorithm(c) Flowchart(d) Tokens

5. Which of the following operator is extraction operator of C++?

(a) >>	(b) <<	(c) <>	(d) ^^
--------	--------	--------	--------

- 6. Which of the following statements is not true?
 - (a) Keywords are the reserved words convey specific meaning to the C++ compiler.
 - (b) Reserved words or keywords can be used as an identifier name.
 - (c) An integer constant must have at least one digit without a decimal point.
 - (d) Exponent form of real constants consists of two parts
- 7. Which of the following is a valid string literal?
 - (a) 'A' (b) 'Welcome' (c) 1232 (d) "1232"
- 8. A program written in high level language is called as
 - (a) Object code (b) Source code
 - (c) Executable code (d) All the above
- 9. Assume a=5, b=6; what will be result of a&b?
 - (a) 4 (b) 5 (c) 1 (d) 0

10. Which of the following is called as compile time operators?

(a) sizeof (b) pointer (c) virtual (d) this

Part – II

Answer to all the questions

- 1. What is meant by a token? Name the token available in C++.
- 2. What are keywords? Can keywords be used as identifiers?
- 3. The following constants are of which type?(i) 39 (ii) 032 (iii) 0XCAFE (iv) 04.14
- 4. Write the following real constants into the exponent form:(i) 23.197 (ii) 7.214 (iii) 0.00005 (iv) 0.319
- 5. Assume n=10; what will be result of n>>2;?
- 6. Match the following

Α	В
(a) Modulus	(1) Tokens
(b) Separators	(2) Remainder of a division
(c) Stream extraction	(3) Punctuators
(d) Lexical Units	(4) get from

Part – III

Answer to all the questions

- 1. Describe the differences between keywords and identifiers?
- 2. Is C++ case sensitive? What is meant by the term "case sensitive"?
- 3. Differentiate "=" and "==".
- 4. Assume a=10, b=15; What will be the value of a^b ?
- 5. What is the difference between "Run time error" and "Syntax error"?
- 6. What are the differences between "Logical error" and "Syntax error"?
- 7. What is the use of a header file?
- 8. Why is main function special?
- 9. Write two advantages of using include compiler directive.
- 10. Write the following in real constants.

(i) 15.223 (ii) 211.05 (iii) 0.00025

Part – IV

Answer to all the questions

- 1. Write about Binary operators used in C++.
- 2. What are the types of Errors?
- 3. Assume a=15, b=20; What will be the result of the following operations?

(a) $a \otimes b$ (b) $a | b (c) a \wedge b$ (d) a >>3 (d) $(\sim b)$

References:

- (1) Object Oriented Programming with C++ (4th Edition), Dr. E. Balagurusamy, Mc.Graw Hills.
- (2) The Complete Reference C++ (Forth Edition), Herbert Schildt.Mc.Graw Hills.

(3) Computer Science with C++ (A text book of CBSE XI and XII), Sumita Arora, Dhanpat Rai & Co.

Data Types, Variables and Expressions

9.10 Introduction

Every programming language has two fundamental elements, viz., data types and variables. They are very essential elements to write even the most elementary programs. C++ provides a predefined set of data types for handling the data items. Such data types are known as fundamental or built-in data types. Apart from the built-in data types, a programmer can also create his own data types called as User-defined data types. In this chapter, we are going to learn about built-in data types.

9.11 Concept of Data types:

Let us look at the following example,

Name = Ram Age = 15 Average_Mark = 85.6

In the above example, Name, Age, Average_mark are the fields which hold the values such as Ram, 15, and 85.6 respectively.

In a programming language, **fields** are referred as **variables** and the **values** are referred to as **data**. Each data item in the above example is looking different. That is, "Ram" is a sequence of alphabets and the other two data items are numbers. The first value is a whole number and the second one is a fractional number. In real-world scenarios, there are lots of different kinds of data we handle in our day-to-day life. The nature or type of the data item varies, for example distance (from your home to school), ticket fare, cost of a pen, marks, temperature, etc.,

In C++ programming, before handling any data, it should be clearly specified to the language compiler, regarding what kind of data it is, with some predefined set of data types.

```
9.12 C++ Data types
```

In C++, the data types are classified as three main categories

(1) Fundamental data types

(2) User-defined data types and

(3) Derived data types.



Figure 9.13 Data types in C++

In this chapter, we are going to learn about only the Fundamental data types.

In order to understand the working of data types, we need to know about variables. The variables are the named memory locations to hold values of specific data types. In C++, the variables should be declared explicitly with their data types before they are actually used.

Syntax for declaring a variable:

<data type> <variable name>;

Example:

int num1;

To declare more than one variable which are of the same data type using a single statement, it can be declared by separating the variables using a comma.

Example:

int num1, num2, sum;

For example, to store your computer science marks; first you should declare a variable to hold your marks with a suitable data type. Choosing an appropriate data type needs more knowledge and experience. Usually, marks are represented as whole numbers. Thus, your variable for storing the computer science marks should be an integer data type.

Example:

int comp_science_marks;

Now, one variable named **comp_science_marks** is ready to store your marks.

We will learn more about variables later in this chapter.
9.12.1 Introduction to fundamental Data types:

Fundamental (atomic) data types are predefined data types available with C++. There are five fundamental data types in C++: **char, int, float, double** and **void**. Actually, these are the keywords for defining the data types.

(1) int data type:

Integers are whole numbers without any fraction. Integers can be positive or negative. Integer data type accepts and returns only integer numbers. If a variable is declared as an **int**, C++ compiler allows storing only integer values into it. If you try to store a fractional value in an int type variable it will accept only the integer portion of the fractional value..

Illustration 9.1: C++ Program to get 2 integer numbers and display their sum:

In the above program, there are three variables declared viz. **num1**, **num2** and **sum**. All these three variables are declared as integer types. So, three different integer values can be stored in these variables.

The variables **num1** and **num2** are used to store the values that are obtained from the user during the execution of the program, whereas the variable sum is used to store the processed (resultant) value.

During the execution of the above program, line numbers 7 and 9 prompt the user to Enter number 1 and number 2.

Input stream **cin** in line 8 and 10 will receive the values given by the user and store the same in variable **num1** and **num2**.

Line 11; both the values of **num1** and **num2** will be added and the result will be assigned to the variable **sum**.

(2) char data type:

Character data type accepts and returns all valid ASCII characters. Character data type is often said to be an integer type, since all the characters are represented in memory by their associated **ASCII Codes.** If a variable is declared as char, C++ allows storing either a character or an integer value.

Illustration 9.2: C++ Program to accept any character and display its next character

```
#include <iostream>
using namespace std;
int main()
{
     char ch;
     cout << "\n Enter a character: ";
     cin >> ch;
     ch = ch + 1;
cout << "\n The Next character: " << ch;
}
The output produced by the program will be
Enter a character: A
The Next character: B</pre>
```

In the above program, **ch** is declared as a char type variable to hold a character. When the user enters a character, it will be stored in the **ch** variable as an integer value (ie. Equivalent ASCII code).

ch = ch + 1;

In this statement, the value of **ch** is incremented by 1 and the new value is stored back in the same variable **ch**. (Remember that, arithmetic operations are carried out only on the numbers not with alphabets).

Another program illustrates how int and char data types are working together.

```
Illustration 9.3: C++ program to get an ASCII value and display the corresponding
character
#include <iostream>
using namespace std;
int main ()
{
       int n:
       char ch;
       cout << "\n Enter an ASCII code (0 to 255): ";
       cin >> n;
       ch = n:
       cout << "\n Equivalent Character: " << ch;</pre>
}
The output
Enter an ASCII code (0 to 255): 100
Equivalent Character: d
```

In the above program, variable **n** is declared as an int type and another variable **ch** as a char type. During execution, the program prompts the user to enter an ASCII value. If the user enters an ASCII value as an integer, it will be stored in the variable **n**. In the statement **ch** = **n**; the value of **n** is assigned into **ch**. Remember that, **ch** is a char type variable.

For example, if a user enters 100 as input; initially, 100 is stored in the variable **n**. In the next statement, the value of **n** i.e., 100 is assigned to **ch**. Since, **ch** is a char type; it shows the corresponding ASCII character as output. (Equivalent ASCII Character for 100 is d).

(3) float data type:

If a variable is declared as float, all values will be stored as floating point values.

There are two advantages of using float data types.

- (1) They can represent values between the integers.
- (2) They can represent a much greater range of values.

At the same time, floating point operations takes more time to execute compared to the integer type ie., floating point operations are slower than integer operations. This is a disadvantage of floating point operation.

```
Hlustration 9.4: C++Program to find the area of circle
#include <iostream>
using namespace std;
int main()
{
    float r, area;
    cout << "\n Enter Radius: ";
    cin >> r;
    area = 3.14 * r * r;
    cout << "\n The Area of the circle is " << area;
}
Output:
Enter Radius: 6.5
The Area of the circle is 132.665</pre>
```

In the above example, two variables **r** and **area** are declared as float type in a single statement. Variable **r** is used to hold radius which is a data provided by the user and **area** for holding the area of the circle which is obtained through the formula. For example, if the user inputs the radius value as 6.5, then the statement **area** = 3.14 * r * r; multiplies the value of **r** twice with 3.14 (value of pi) and the resultant value 132.665 will be kept in the variable **area**.

In this case, **r** and **area** both are same type of variables. So, these two variables are declared in a single statement. Declaring the same variables in a single statement reduces the processing time rather than multiple declarations.

(4) double data type:

This is for double precision floating point numbers. (precision means significant numbers after decimal point). The double is also used for handling floating point numbers. But, this type occupies double the space than float type. This means, more fractions can be accommodated in double than in float type. The double is larger and slower than type float. double is used in a similar way as that of float data type.

```
(5) void data type:
```

The literal meaning for void is 'empty space'. Here, in C++, the void data type specifies an empty set of values. It is used as a return type for functions that do not return any value.

Evaluate Yourself

- 1. What do you mean by fundemantal data types?
- 2. The data type char is used to represent characters. then why is it often termed as an integer type?
- 3. What is the advartage of floating point numbers over integers?
- 4. The data type double is another floating point type. Then why is it treated as a distinct data type?
- 5. What is the use of void data type?

9.12.2 Memory representation of Fundamental Data types:

One of the most important reason for declaring a variable as a particular data type is to allocate appropriate space in memory. As per the stored program concept, every data should be accommodated in the main memory before they are processed. So, C++ compiler allocates specific memory space for each and every data handled according to the compiler's standards.

The following Table 9.5 shows how much of memory space is allocated for each fundamental data type. Remember that, every data is stored inside the computer memory in the form of binary digits (See Unit I Chapter 2).

Data tuma	Space in :	memory	Danga of value	
Data type	in terms of bytes	in terms of bits	Kange of value	
char	1 byte	8 bits	-127 to 128	
int	2 bytes	16 bits	-32,768 to 32,767	
float	4 bytes	32 bits	3.4×10 ⁻³⁸ to 3.4×10 ³⁸ -1	
double	8 bytes	64 bits	1.7×10^{-308} to 1.7×10^{308} -1	

Table 9.5 Memory allocation for Fundamental data types

9.12.3 Data type modifiers:

Modifiers are used to modify the storing capacity of a fundamental data type except void type. Usually, every fundamental data type has a fixed range of values to store data items in memory. For example, int data type can store only two bytes of data. In reality, some integer data may have more length and may need more space in memory. In this situation, we should modify the memory space to accommodate large integer values. **Modifiers can be used to modify (expand or reduce) the memory allocation of any fundamental data type. They are also called as Qualifiers.**

There are four modifiers used in C++. They are:

(1) signed (2) unsigned (3) long (4) short

These four modifiers can be used with any fundamental data type. The following Table 9.6 shows the memory allocation for each data type with and without modifiers.

Integer type

	Space in	memory				
	in terms in terms		Range of value			
	of bytes	of bits				
short short is a short name for short int		2 bytes	16 bits	-32,768 to 32,767		
unsigned short	an integer number without minus sign.	2 bytes	16 bits	0 to 65535		
signed short An integer number with minus sign		4 bytes	32 bits	-32,768 to 32,767		
Both short and signed short are similar						
int	An integer may or may not be signed	2 bytes	16 bits	-32,768 to 32,767		
unsigned intAn integer without any sign (minus symbol)		2 bytes	16 bits	0 to 65535		
signed int An integer with sign		2 bytes	16 bits	-32,768 to 32,767		
	Both short and in	nt are simi	lar			
long long is short name for long int		4 bytes	32 bits	-2147483648 to 2147483647		
unsigned long	A double spaced integer without any sign	4 bytes	32 bits	0 to 4,294,967,295		
signed long	A double spaced integer with sign	4 bytes	32 bits	-2147483648 to 2147483647		

Table 9.6 Memory allocation for Data types

The above table clearly shows that an integer type accepts only 2 bytes of data whereas a long int accepts data that is double this size i.e., 4 bytes of data. So, we can store more digits in a long int. (long is modifier and int is a fundamental data type)

char type

		Space in	memory	
	in terms of bytes	in terms of bits	Range of value	
char	Signed ASCII character	1 byte	8 bits	-128 to 127
unsigned char	ASCII character without sign	1 byte	8 bits	0 to 255
signed char	ASCII character with sign	1 byte	8 bits	-128 to 127

Table 9.7 Memory allocation for char Data types

Floating point type

 Table 9.8 Memory allocation for floating point Data types

	Space in memory			
	Data type	in terms	in terms	Range of value
		of bytes	of bits	
float	signed fractional value	4 bytes	32 bits	3.4×10^{-38} to 3.4×10^{38} -1
double	signed more precision	8 bytes	61 bits	1.7×10^{-308} to
double	fractional value	obytes	04 0113	1.7×10^{308} -1
long double	signed more precision	10 butos	on hite	3.4×10^{-4932} to
	fractional value	10 Dytes	00 0113	1.1×10^{4932} -1

Memory allocation is subjected to vary based on the type of compiler that is being used. Here, the given values are as per the **Turbo C++** compiler. **Dev C++** provides some more space to **int** and **long** double types. Following Tables 9.9 shows the difference between Turbo C++ and Dev C++ allocation of memory.

Table 9.9 Memory allocation by Turbo C++ and Dev C++

Data typa	Memory size in bytes				
Data type	Turbo C++	Dev C++			
short	2	2			
unsigned short	2	2			
signed short	2	2			
int	2	4			
unsigned int	2	4			
signed int	2	4			
long	4	4			

unsigned long	4	4
signed long	4	4
char	1	1
unsigned char	1	1
signed char	1	1
float	4	4
double	8	8
long double	10	12

Since, Dev C++ provides 4 bytes to int and long, any one of these types can be used to handle bigger integer values while writing programs in Dev C++.

Note: sizeof() is an operator which gives the size of a data type.

```
Illustration 9.5: C++ Program to find the size of data types
#include <iostream>
using namespace std;
int main()
{
       short a;
       unsigned short b;
       signed short c;
       int d;
       unsigned int e;
       signed int f;
       long g;
       unsigned long h;
       signed long i;
       char j;
       unsigned char k;
       signed char l;
       float m;
       double n;
       long double p;
       cout << "\n Size of short = " << sizeof(a);</pre>
       cout << "\n Size of unsigned short = " << sizeof(b);</pre>
       cout << "\n Size of signed short = " << sizeof (c);</pre>
       cout << "\n Size of int = " << sizeof(d);</pre>
```

```
cout << "\n Size of unsigned int = " << sizeof(e);
cout << "\n Size of signed int = " << sizeof(f);
cout << "\n Size of long = " << sizeof(g);
cout << "\n Size of unsigned long = " << sizeof(h);
cout << "\n Size of signed long = " << sizeof(i);
cout << "\n Size of char = " << sizeof(j);
cout << "\n Size of unsigned char = " << sizeof(k);
cout << "\n Size of signed char = " << sizeof(l);
cout << "\n Size of float = " << sizeof(m);
cout << "\n Size of double = " << sizeof(n);
cout << "\n Size of long double = " << sizeof(p);</pre>
```

}

Output : (compiled and executed in Dev C++)

Size of short = 2 Size of unsigned short = 2 Size of signed short = 2 Size of int = 4 Size of unsigned int = 4 Size of signed int = 4 Size of long = 4 Size of unsigned long = 4 Size of signed long = 4 Size of char = 1 Size of unsigned char = 1 Size of signed char = 1 Size of float = 4 Size of double = 8 Size of long double = 12

Number Suffixes in C++

There are different suffixes for integer and floating point numbers. Suffix can be used to assign the same value as a different type. For example, if you want to store 45 in an int, long,

unsigned int and unsigned long int, you can use suffix letter L or U (either case) with 45 i.e. **45L** or **45U**. This type of declaration instructs the compiler to store the given values as long and unsigned. 'F' can be used for floating point values, example: 3.14F

9.13 Variables

Variables are user-defined names assigned to specific memory locations in which the values are stored. Variables are also identifiers; and hence, the rules for naming the identifiers should be followed while naming a variable. These are called as symbolic variables because these are named locations.

There are two values associated with a symbolic variable; they are **R**-value and **L**-value.

- R-value is data stored in a memory location
- L-value is the memory address in which the R-value is stored.



Figure 9.14 Memory allocation of a variable

Remember that, the memory addresses are in the form of Hexadecimal values

9.13.1 Declaration of Variables:

Every variable should be declared before they are actually used in a program. Declaration is a process to instruct the compiler to allocate memory as per the type that is specified along with the variable name. For example, if you declare a variable as int type, in Dev C++, the compiler allocates 4 bytes of memory. Thus, every variable should be declared along with the type of value to be stored.

Declaration of more than one variable:

More than one variable of the same type can be declared as a single statement using a comma separating the individual variables.

Syntax:

<data type> <var1>, <var2>, <var3> <var_n>;

Example:

int num1, num2, sum;

In the above statement, there are three variables declared as int type. Which means, in **num1, num2** and **sum**, you can store only integer values.

For the above declaration, the C++ compiler allocates 4 bytes of memory (i.e. 4 memory boxes) for each variable.



Figure 9.15 Memory allocation of int type variables

If you declare a variable without any initial value, the memory space allocated to that variable will be occupied with some unknown value. These unknown values are called as **"Junk"** or **"Garbage"** values.

```
#include <iostream>
using namespace std;
int main()
{
    int num1, num2, sum;
    cout << num1 << endl;
    cout << num2 << endl;
    cout << num1 + num2;
}</pre>
```

In the above program, some unknown values will be occupied in memory that is allocated for the variables **num1** and **num2**; and the statement c**out** << **num1** + **num2**; will display the sum of those unknown junk values.

9.13.2 Initialization of variables:

Assigning an initial value to a variable during its declaration is called as "Initialization".

Examples:

int num = 100;

float pi = 3.14;

double price = 231.45;

Here, the variables num, pi, and price have been initialized during the declaration. These initial values can be later changed during the program execution.

```
Illustration 9.6 C++ Program to find the Curved Surface Area of a cylinder (CSA) (CSA
= 2 pi r * h)
#include <iostream>
using namespace std;
int main()
{
        float pi = 3.14, radius, height, CSA;
        cout << "\n Curved Surface Area of a cylinder";</pre>
        cout << "\n Enter Radius (in cm): ";</pre>
        cin >> radius:
        cout << "\n Enter Height (in cm): ";</pre>
        cin >> height;
        CSA = (2*pi*radius)*height;
        system("cls");
        cout << "\n Radius: " << radius << "cm";</pre>
        cout << "\n Height: " << height << "cm";</pre>
        cout << "\n Curved Surface Area of a Cylinder is " << CSA <<" sq. cm.";
}
Output:
 Curved Surface Area of a cylinder
```

```
Enter Radius (in cm): 7
Enter Height (in cm): 20
Radius: 7cm
Height: 20cm
Curved Surface Area of a Cylinder is 879.2 sq. cm.
```

Variables that are of the same type can be initialized in a single statement.

Example:

int x1 = -1, x2 = 1, x3, n;

9.13.3 Dynamic Initialization:

A variable can be initialized during the execution of a program. It is known as "Dynamic initialization". For example,

int num1, num2, sum;

```
sum = num1 + num2;
```

The above two statements can be combined into a single one as follows:

```
int sum = num1+num2;
```

This initializes sum using the known values of num1 and num2 during the execution.

Illustration 9.7 C++ Program to illustrate dynamic initializetion

```
#include <iostream>
using namespace std;
int main()
{
      int num1, num2;
       cout << "\n Enter number 1: ";</pre>
       cin >> num1;
       cout << "\n Enter number 2: ";</pre>
       cin >> num2;
       int sum = num1 + num2; // Dynamic initialization
       cout << "\n Average: " << sum /2;</pre>
}
Output:
Enter number 1:78
Enter number 2:65
Average: 71
```

In the above program, after getting the values of num1 and num2, sum is declared and initialized with the addition of those two variables. After that, it is divided by 2.

```
Illustration 9.8: C++ program to find the perimeter and area of a semi circle
#include <iostream>
using namespace std;
int main()
{
       int radius:
       float pi = 3.14;
       cout << "\n Enter Radius (in cm): ";</pre>
       cin >> radius;
       float perimeter = (pi+2)*radius; // dynamic initialization
       float area = (pi*radius*radius)/2; // dynamic initialization
       cout << "\n Perimeter of the semicircle is " << perimeter << " cm";
       cout << "\n Area of the semicircle is " << area << " sq.cm";
}
Output:
Enter Radius (in cm): 14
Perimeter of the semicircle is 71.96 cm
Area of the semicircle is 307.72 sq.cm
```

9.13.4 The Access modifier const:

const is the keyword used to declare a constant. You already learnt about constant in the previous chapter. const keyword modifies / restricts the accessibility of a variable. So, it is known as Access modifier.

For example,

int num = 100;

The above declares a variable num with an initial value 100. However, the value of num can be changed during the execution. If you modify the above definition as **const int num** = **100**; the variable num becomes a constant and its value will remain as 100 throughout the program, and it can never be changed during the execution.

```
#include <iostream>
using namespace std;
int main()
{
     const int num=100;
     cout << "\n Value of num is = " << num;
     num = num + 1; // Trying to increment the constant
     cout << "\n Value of num after increment " << num;
}</pre>
```

For the above code, an error message will be displayed as **"Cannot modify the const object"** in Turbo compiler and **"assignment of read only memory num"** in Dev C++.

Evaluate Yourself

- 1. What is modifiers? What is the use of modifiers?
- What is wrong with the following C++ statement: long float x;
- 3. What is variable ? Why a varible called symblolic varible?
- 4. What do you mean by dynamic initialization of a variable? Give an exmple.
- 5. What is wrong with the following statement?

const int x;

9.13.5 References:

A reference provides an alias for a previously defined variable. Declaration of a reference consists of base type and an & (ampersand) symbol; reference variable name is assigned the value of a previously declared variable.

Syntax:

<type> <& reference_variable> = <original_variable>

```
Illustration 9.9: C++ program to declare reference variable
#include <iostream>
using namespace std;
int main()
{
    int num;
    int &temp = num; //declaration of a reference variable temp
    num = 100;
cout << "\n The value of num = " << num;
cout << "\n The value of temp = " << temp;
}
The output of the above program will be
The value of num = 100
The value of temp = 100</pre>
```

9.14 Formatting Output:

Formatting output is very important in the development of output screens for easy reading and understanding. Manipulators are used to format the output of any C++ program. Manipulators are functions specifically designed to use with the insertion (<<) and extraction(>>) operators.

C++ offers several input and output manipulators for formatting. Commonly used manipulators are: **endl, setw, setfill, setprecision and setf**. In order to use these manipulators, you should include the appropriate header file. **endl** manipulator is a member of iostream header file. **setw, setfill, setprecision and setf** manipulators are members of iomanip header file.

endl (End the Line)

endl is used as a line feeder in C++. It can be used as an alternate to '\n'. In other words, endl inserts a new line and then makes the cursor to point to the beginning of the next line. There is a difference between endl and '\n', even though they are performing similar tasks.

- endl Inserts a new line and flushes the buffer (Flush means clean)
- '\n' Inserts only a new line.

Example:

```
cout << "\n The value of num = " << num;
```

cout << "The value of num = " << num <<end;</pre>

Both these statements display the same output.

setw()

setw manipulator sets the **width of the field** assigned for the output. The field width determines the minimum number of characters to be written in output.

Syntax:

setw(number of characters)

Example:

```
Illustration 9.10: Program to Calculate Net Salary
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
       float basic, da, hra, gpf, tax, gross, np;
       char name[30];
       cout << "\n Enter Basic Pay: ";</pre>
       cin >> basic;
       cout \ll "\n Enter D.A : ";
       cin >> da:
       cout << "\n Enter H.R.A: ";
       cin >> hra;
       gross = basic+da+hra; // sum of basic, da nad hra
       gpf = (basic+da) * 0.10; // 10\% 0f basic and da
       tax = gross * 0.10; //10\% of gross pay
       np = gross - (gpf+tax); //netpay = earnings - deductions
      cout << setw(25) << "Basic Pay : " << setw(10) << basic << endl;
       cout << setw(25) << "Dearness Allowance : "<< setw(10)<<da<< endl;</pre>
       cout<<setw(25)<<"House Rent Allowance : "<<setw(10)<< hra<<endl;</pre>
       cout << setw(25) << "Gross Pay : " << setw(10) << gross << endl;
       cout << setw(25) << "G.P.F : " << setw(10) << gpf << endl;
       cout << setw(25) << "Income Tax : " << setw(10) << tax << endl;
       cout << setw(25) << "Net Pay : " << setw(10) << np << endl;
}
```

The output will be,					
Enter Basic Pay: 12000					
Enter D.A : 1250					
Enter H.R.A : 1450					
Basic Pay	:	12000			
Dearness Allowance	:	1250			
House Rent Allowance	:	1450			
Gross Pay	:	14700			
G.P.F	:	1325			
Income Tax	:	1470			
Net Pay	:	11905			
(HOT: Try to make multiple output statements as a single cout statement)					

In the above program, every output statement has two setw() manipulators; first setw (25) creates a filed with 25 spaces and second setw(10) creates another field with 10 spaces. When you represent a value to these fields, it will show the value within the field from right to left.



In field1 and field 2, the string "Basic Pay: " and the value of basic pay are shown as given in Figure 9.16 below.

Field 1 with 25 space width	Field 2 with 10 space w	vidth
Basic pay:	12000	



setfill ()

This manipulator is usually used after setw. If the presented value does not entirely fill the given width, then the specified character in the setfill argument is used for filling the empty fields.

Syntax:

setfill (character);

Example:

cout << "\n H.R.A : " << setw(10) << setfill (0) << hra;

In the above code, setw creates a field to show the presented value, setfill is used to fill un-occupied spaces with 0 (zero).

For example, if you assign 1200 to hra, setw accommodates 1200 in a field of width 10 from right to left and setfill fills 0 in the remaining 6 spaces that are in the beginning. The output will be, **0000001200**.

setprecision ()

This is used to display numbers with fractions in specific number of digits.

Syntax:

setprecision (number of digits);

Example:

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

```
float hra = 1200.123;
```

cout << setprecision (5) << hra;</pre>

}

In the above code, the given value 1200.123 will be displayed in 5 digits including fractions. So, the output will be **1200.1**

setprecision () prints the values from left to right. For the above code, first, it will take 4 digits and then prints one digit from fractional portion.

setprecision can also be used to **set the number of decimal places to be displayed**. In order to do this task, you will have to set an ios flag within **setf()** manipulator. This may be used in two forms: (i) **fixed** and (ii) **scientific**

These two forms are used when the keywords fixed or scientific are appropriately used before the setprecision manipulator.

Example:

#include <iostream>

#include <iomanip>

using namespace std;

int main()

```
{
```

cout.setf(ios::fixed);

```
cout << setprecision(2)<<0.1;</pre>
```

}

In the above program, ios flag is set to **fixed** type; it prints the floating point number in fixed notation. So, **the output will be, 0.10**

cout.setf(ios::scientific);

cout << setprecision(2) << 0.1;</pre>

In the above statements, ios flag is set to **scientific type;** it will print the floating point number in scientific notation. So, **the output will be, 1.00e-001**

9.15 Expression:

An expression is a combination of operators, constants and variables arranged as per the rules of C++. It may also include function calls which return values. (Functions will be learnt in upcoming chapters).

An expression may consist of one or more operands, and zero or more operators to produce a value. In C++, there are seven types of expressions, and they are:

- (i) Constant Expression
- (ii) Integer Expression
- (iii) Floating Expression
- (iv) Relational Expression
- (v) Logical Expression
- (vi) Bitwise Expression
- (vii) Pointer Expression

SN	Expression	Description	Example
1	Constant Expression	Constant expression consist only constant values	int num=100;
2	Integer Expression	The combination of integer and character values and/or variables with simple arithmetic operators to produce integer results.	sum=num1+num2; avg=sum/5;
3	Float Expression	The combination of floating point values and/or variables with simple arithmetic operators to produce floating point results.	Area=3.14*r*r;
4	Relational Expression	The combination of values and/or variables with relational operators to produce bool(true means 1 or false means 0) values as results.	x>y; a+b==c+d;
5	Logical Expression	The combination of values and/or variables with Logical operators to produce bool values as results.	(a>b)&& (c==10);
6	Bitwise Expression	The combination of values and/or variables with Bitwise operators.	x>>3; a<<2;
7	Pointer Expression	A Pointer is a variable that holds a memory address. Pointer declaration statements.	int *ptr;

Table 9.10 . Types of Expressions	Table 9.10	: Types	of Expi	ressions
-----------------------------------	------------	---------	---------	----------

9.16 Type Conversion

The process of converting one fundamental type into another is called as "Type Conversion". C++ provides two types of conversions.

(1) Implicit type conversion

(2) Explicit type conversion.

(1) Implicit type conversion:

An Implicit type conversion is a conversion performed by the compiler automatically. So, implicit conversion is also called as **"Automatic conversion"**.

This type of conversion is applied usually whenever different data types are intermixed in an expression. If the type of the operands differ, the compiler converts one of them to match with the other, using the rule that the "smaller" type is converted to the "wider" type, which is called as **"Type Promotion"**.

For example:

#include <iostream>

using namespace std;

int main()

{

int a=6;

float b=3.14;

cout << a+b;</pre>

}

In the above program, operand **a** is an int type and **b** is a float type. During the execution of the program, int is converted into a float, because a float is wider than int. Hence, the output of the above program will be: **9.14**

RHO LHO	char	short	int	long	float	double	long double
char	int	int	int	long	float	double	long double
short	int	int	int	long	float	double	long double
int	int	int	int	long	float	double	long double
long	long	long	long	long	float	double	long double
float	float	float	float	float	float	double	long double
double	long double						
long double							

The following Table 9.11 shows you the conversion pattern.

(RHO – Right Hand Operand; LHO – Left Hand Operand)

Table 9.11: Implicit conversion of mixed operands

(2) Explicit type conversion

C++ allows explicit conversion of variables or expressions from one data type to another specific data type by the programmer. It is called as **"type casting"**.

Syntax:

(type-name) expression;

Where type-name is a valid C++ data type to which the conversion is to be performed.

Example:

```
#include <iostream>
```

using namespace std;

int main()

{

```
float varf=78.685;
```

```
cout << (int) varf;</pre>
```

}

In the above program, variable **varf is** declared as a **float** with an initial value 78.685. The value of **varf** is explicitly converted to an **int** type in cout statement. Thus, the final output will be 78.

During explicit conversion, if you assign a value to a type with a greater range, it does not cause any problem. But, assigning a value of a larger type to a smaller type may result in loosing or loss of precision values.

SN	Explicit Conversion	Problem
1	double to float	Loss of precision. If the original value is out of range for the target type, the result becomes undefined
2	float to int	Loss of fractional part. If original value may be out of range for target type, the result becomes undefined
3	long to short	Loss of data

Table 9.12 – Explicit Conversion Problems

Example: #include <iostream> using namespace std; int main() { double varf=178.25255685; cout << (float) varf << endl; cout << (int) varf << endl; } Output: 178.253 178</pre>

Evaluate Yourself

- 1. What is meant by type conversion?
- 2. How implicit conversion different from explicit conversion?
- 3. What is difference between endl and n?
- 4. What is the use of references?
- 5. What is the use of setprecision ()?



Hands on practice:

- 1. Write C++ programs to interchange the values of two variables.
 - a. Using with third variable
 - b. Without using third variable
- 2. Write C++ programs to do the following:
 - a. To find the perimeter and area of a quadrant.
 - b. To find the area of triangle.
 - c. To convert the temperature from Celsius to Fahrenheit.

3. Write a C++ to find the total and percentage of marks you secured from 10th Standard Public Exam. Display all the marks one-by-one along with total and percentage. Apply formatting functions.

Points to Remember:

- Every programming language has two fundamental elements, viz., data types and variables.
- In C++, the data types are classified as three main categories (1) Built-in data types (2) User-defined data types (3) Derived data types.
- The variables are the named space to hold values of certain data type.
- There are five fundamental data types in C++: char, int, float, double and void.
- C++ compiler allocates specific memory space for each and every data handled according to the compiler's standards.
- Variables are user-defined names assigned to a memory location in which the values are stored.

- Declaration is a process to instruct the compiler to allocate memory as per the type specified along with the variable name.
- Manipulators are used to format output of any C++ program. Manipulators are functions specifically designed to use with the insertion (<<) and extraction(>>) operators.
- An expression is a combination of operators, constants and variables arranged as per the rules of C++.
- The process of converting one fundamental type into another is called as "Type Conversion". C++ provides two types of conversions (1) Implicit type conversion and (2) Explicit type conversion.



Part – I

Choose the correct answer.

- 1. How many categories of data types available in C++?
 - (a) 5 (b) 4 (c) 3 (d) 2
- 2. Which of the following data types is not a fundamental type?

(a) signed (b) int (c) float (d) char

3. What will be the result of following statement?

	char ch= 'B';				
	cout << (int) ch;				
	(a) B	(b) b	(c) 65	(d) 66	
4.	Which of the character is used as suffix to indicate a floating point value?				
	(a) F	(b) C	(c) L	(d) D	
5.	How many bytes o using Dev C++?	f memory allocate short int x;	es for the following va	riable declaration if you are	
	(a) 2	(b) 4	(c) 6		
6.	What is the output of the following snippet?				
	char ch = 'A';			F9H7JH	
	ch = ch + 1;				
	(a) B	(b) A1	(c) F	(d) 1A	
7.	Which of the following is not a data type modifier?				
	(a) signed	(b) int	(c) long	(d) short	
8.	Which of the following operator returns the size of the data type?			ta type?	
	(a) sizeof()	(b) int ()	(c) long ()	(d) double ()	
9.	Which operator to be used to access reference of a variable?				
	(a) \$	(b) #	(c) &	(d) !	
10.	This can be used as alternate to endl command:				
	(a) \t	(b) \b	(c) \0	(c) \n	
			Part – II		
Ans	swers to all the ques	stions (2 Marks):			
1.	Write a short note const keyword with an example.				
2.	What is the use of setw() format manipulator?				

3. Why is char often treated as integer data type?

- 4. What is a reference variable? What is its use?
- 5. Consider the following C++ statement. Are they equivalent?

char ch = 67;

char ch = 'C';

- 6. What is the difference between 56L and 56?
- 7. Determine which of the following are valid constant? And specify their type.

(i) 0.5 (ii) 'Name' (iii) '\t' (iv) 27,822

- 8. Suppose x and y are two double type variable that you want add as integer and assign to an integer variable. Construct a C++ statement for the doing so.
- 9. What will be the result of following if num=6 initially.

(a) cout << num;

- (b) cout << (num==5);
- 10. Which of the following two statements are valid? Why? Also write their result.

int a;

(i) a = 3,014; (ii) a=(3,014);

Part – III

Answers to all the questions (3 Marks):

- 1. What are arithmetic operators in C++? Differentiate unary and binary arithmetic operators. Give example for each of them.
- 2. Evaluate x+= x + ++x; Let x=5;
- 3. How relational operators and logical operators related to one another?

- 4. Evaluate the following C++ expressions where x, y, z are integers and m, n are floating point numbers. The value of x = 5, y = 4 and m=2.5;
 - (i) n = x + y / x;
 - (ii) z = m * x + y;
 - (iii) z = (x++) * m + x;

Reference:

- (1) Object Oriented Programming with C++ (4th Edition), Dr. E. Balagurusamy, Mc.Graw Hills.
- (2) The Complete Reference C++ (Forth Edition), Herbert Schildt. Mc.Graw Hills.
- (3) Computer Science with C++ (A text book of CBSE XI and XII), Sumita Arora, Dhanpat Rai & Co.

Unit III | Introduction to C++

O Learning Objectives

After learning this chapter, the students will be able to

• Understand the different kinds of statements.



• Construct different flow of control statements in C++.

10.1 Introduction

In the previous chapters you learnt the basic concepts of C++ programming such as variables, constants, operators, data types etc. Generally a program executes its statements sequentially from beginning to end. However, such a strict sequential ordering is restrictive and less useful. There are lot of situations where it is useful to decide the code block executed on the basis of a certain condition. In such situations, the flow of control jumps from one part of the code to another segment of code. Program statements that cause such jumps are called as "Control flow". This chapter deals with the basics of control structures such as "Selection", "Iteration" and "Jump" statement.

10.2 Statements

A computer program is a set of statements or instructions to perform a

Flow of Control

CHAPTER

specific task. These statements are intended to perform specific action. The action may be of variable declarations, expression evaluations, assignment operations, decision making, looping and so on.

There are two kinds of statements used in C++.

- (i) Null statement
- (ii) Compound statement
- 10.2.1 Null statement

The "**null** or **empty statement**" is a statement containing only a semicolon. It takes the flowing form:

; // it is a null statement

Null statements are commonly used as placeholders in iteration statements or as statements on which to place labels at the end of compound statements or functions.

10.2.2 Compound (Block) statement

C++ allows a group of statements enclosed by pair of braces {}. This group of statements is called as a compound statement or a block.

The general format of compound statement is:

In a program, statements may be executed sequentially, selectively or iteratively. Every programming languages provides statements to support sequence, selection (branching) and iteration.

If the Statements are executed sequentially, the flow is called as sequential flow. In some situations, if the statements alter the flow of execution like branching, iteration, jumping and function calls, this flow is called as control flow.

Sequence statement

Statement 1		
Statement 2		
Statement 3		

The **sequential statement** are the statements, that are executed one after another only once from top to bottom. These statements do not alter the flow of execution. These statements are called as sequential flow statements. They are always end with a semicolon (;).



{

statement1;

statement2;

statement3;

}

For example

{

int x, y;

x = 10;

y = x + 10;

}

The compound statement or block is a treated as a single unit and may be appear anywhere in the program.

10.3 Control Statements

Control statements are statements that alter the sequence of flow of instructions. **Selection statement**

The selection statement means the statement (s) are executed depends upon a condition. If a condition is true, a true block (a set of statements) is executed otherwise a false block is executed. This statement is also called **decision statement** or **selection statement** because it helps in making decision about which set of statements are to be executed.

Iteration statement



The **iteration statement** is a set of statement are repetitively executed depends upon a conditions. If a condition evaluates to true, the set of statements (true block) is executed again and again. As soon as the condition becomes false, the repetition stops. This is also known as **looping statement** or iteration statement.

The set of statements that are executed again and again is called the **body of the loop**. The condition on which the execution or exit from the loop is called **exit-condition** or **test-condition**.

Generally, all the programming languages supports this type of statements to write programs depends upon the problems. C++ also supports this type of statements. These statements will be discussed in coming sections.

In selection statements and iteration statements are executed depends upon the conditional expression. The conditional expression evaluates either true or false.



10.4 Selection statements

In a program a decision causes a one time jump to a different part of a program. Decisions in C++ are made in several ways, most importantly with if .. else ... statement which chooses between two alternatives. Another decision statement, switch creates branches for multiple alternatives sections of code, depending on the value of a single variable.

10.4.1 if statement

The if statement evaluates a condition, if the condition is true then a true-block (a statement or set of statements) is executed, otherwise the true-block is skipped. The general syntax of the if statement is:

if (expression)	
true-block;	
statement-x;	

In the above syntax, if is a keyword that should contain expression or condition which is enclosed within parentheses. If the expression is true (nonzero) then the true-block is executed and followed by statement-x are also executed, otherwise, the control passes to statement-x. The true-block may consists of a single statement, a compound statement or empty statement. The control flow of if statement and the corresponding flow chart is shown below.



Illustration 10.1 C++ program to calculate total expenses using if statement

```
#include<iostream>
using namespace std;
int main()
       int qty, dis=0;
       float rate, tot;
       cout<<"\nEnter the quantity ";
       cin>>qty;
```

{

```
cout<<"\nEnter the rate ";
       cin>>rate;
       if ( qty> 500)
       dis=10;
       tot = (qty * rate) - (qty * rate * dis / 100);
       cout<<"The total expenses is "<< tot;
       return 0;
}
Output
First Run
Enter the quantity 550
Enter the rate 10
The total expenses is 4950
Second Run
Enter the quantity 450
Enter the rate 10
The total expenses is 4500
```

In the first execution of the program, the test condition evaluates to true, since qty> 500. Therefore, the variable dis which is initialized to 0 at the time of declaration, now gets a new value 10. The total expenses is calculated using a new dis value.

In the second execution of the program, the test condition evaluates to false, since qty> 500. Thus, the variable dis which is initialized to 0 at the time of declaration, remains 0. Hence, the expression after the minus sign evaluates to 0. So, the total expenses is calculated without discount.



Output

Enter your age: 23 You are eligible for voting.... This statement is always executed.

```
Illustration 10.3 C++ program to calculate bonus using if statement
#include<iostream>
using namespace std;
int main()
{
int bonus,yr_of_ser;
cout<<"\nEnter your year of service ";</pre>
cin>>yr_of_ser;
       if ( yr_of_ser> 3 )
       {
                                                  The pair of braces must be required because the
       bonus=2000;
                                                  if condition followed by more then statement
       cout<<"\n Your bonus is " <<bonus;
       }
cout<<"\nCongratulations...";
return 0;
}
Output
Enter your year of service 5
Your bonus is 2000
Congratulations...
```

10.4.2 if-else statement

In the above examples of if, you have seen so for allow you to execute a set of statement is a condition evaluates to true. What if there is another course of action to be followed if the condition evaluates to false. There is another form of if that allows for this kind of either or condition by providing an else clause. The syntax of the if-else statement is given below:

```
if ( expression)
{
    True-block;
}
else
{
    False-block;
}
Statement-x
```

In if-else statement, first the expression or condition is evaluated either true of false. If the result is true, then the statements inside true-block is executed and false-block is skipped. If the result is false, then the statement inside the false-block is executed i.e., the true-block is skipped.



Illustration 10.4 C++ program to find whether the given number is even number or odd number using if-else statement

```
#include <iostream>
using namespace std;
int main()
{
       int num, rem;
       cout<< "\n Enter a number: ";</pre>
       cin>>num;
       rem = num \% 2;
       if (rem == 0)
              cout<< "\n The given number" <<num<< " is Even";</pre>
       else
              cout<< "\n The given number "<<num<< " is Odd";</pre>
       return 0;
}
Output
Enter number: 10
The given number 10 is Even
```

In the above program, the remainder of the given number is stored in rem. If the value of rem is zero, the given number is inferred as an even number otherwise, it is inferred as on odd number.
10.4.3 Nested if

An if statement contains another if statement is called nested if. The nested can have one of the following three forms.

- 1. If nested inside if part
- 2. If nested inside else part
- 3. If nested inside both if part and else part

The syntax of the nested if:

```
If nested inside if part
                                        If nested inside else part
                                      if (expression-1)
if (expression-1)
{
                                      {
                                              body of true part;
        if (expression)
                                      }
        {
         True_Part_Statements;
                                      else
        }
                                      {
        else
                                              if (expression)
        {
                                              {
         False_Part_Statements;
                                                True_Part_Statements;
                                              }
}
                                              else
else
                                               ł
                                               False_Part_Statements;
    body of else part;
                                              }
                                      }
```

```
If nested inside both if part
           and else part
if (expression)
{
        if (expression)
        {
         True_Part_Statements;
        }
        else
        {
         False_Part_Statements;
        }
}
else
{
        if (expression)
        {
         True_Part_Statements;
        }
        else
        {
         False_Part_Statements;
        }
}
```

In the first syntax of the nested if mentioned above the expression-1 is evaluated and the expression result is false then control passes to statement-m. Otherwise, expression-2 is evaluated, if the condition is true, then Nested-True-block is executed, next statement-n is also executed. Otherwise Nested-False-Block, statement-n and statement-m are executed.



The working procedure of the above said if..else structures are given as flowchart below:

Flowchart 10.2 If nested inside else part

Next Statement



Flowchart 10.3 If nested inside both if part and else part

```
Illustration 10.5 – C++ program to calculate commission according to grade using
nested if statement
 #include <iostream>
 using namespace std;
 int main()
 {
        int sales, commission;
        char grade;
        cout << "\n Enter Sales amount: ";</pre>
        cin >> sales;
        cout << "\n Enter Grade: ";</pre>
        cin >> grade;
        if (sales > 5000)
        {
                commission = sales * 0.10;
               cout << "\n Commission: " << commission;</pre>
        }
        else
         {
                commission = sales * 0.05;
               cout << "\n Commission: " << commission;</pre>
        cout << "\n Good Job ..... ";</pre>
        return 0;
 }
 Output:
 Enter Sales amount: 6000
 Enter Grade: A
 Commission: 600
 Good Job .....
```

10.4.4 if -else-if ladder

The if-else ladder is a multi-path decision making statement. In this type of statement 'if' is followed by one or more else if statements and finally end with an else statement.

The syntax of if-else ladder:

```
if (expression 1)
{
    Statemet-1
}
else
    if( expression 2)
    {
        Statemet-2
    }
    else
        if ( expression 3)
        {
            Statemet-3
        }
        else
        {
            Statemet-4
        }
```

When the respective expression becomes true, the statement associated with block is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.



Flowchart 10.4 if-else ladder flow chart

Illustration 10.6 C++ program to find your grade using if-else ladder.

```
#include <iostream>
using namespace std;
int main ()
{
int marks;
cout<<" Enter the Marks :";
cin>>marks;
if (marks \ge 60)
       cout<< "Your grade is 1st class !!" <<endl;</pre>
               else if( marks \geq 50 && marks < 60)
                       cout<< "your grade is 2nd class !!" << endl;
                               else if( marks >= 40 && marks < 50)
                                       cout<< "your grade is 3rd class !!" <<endl;
else
       cout<< "You are fail !!" <<endl;
return 0;
}
Output
Enter the Marks :60
Your grade is 1st class !!
```

When the marks are greater than or equal to 60, the message "Your grade is 1st class !!" is displayed and the rest of the ladder is bypassed. When the marks are between 50 and 59, the message "Your grade is 2nd class !!" is displayed, and the other ladder is bypassed. When the mark between 40 to 49, the message "Your grade is 3nd class !!" is displayed, otherwise, the message "You are fail !!" is displayed.

```
10.4.5 The ?: Alternative to if- else
```

The conditional operator (or Ternary operator) is an alternative for 'if else statement'. The conditional operator that consists of two symbols (?:). It takes three arguments. The control flow of conditional operator is shown below

The syntax of the conditional operator is:



In the above syntax, the expression 1 is a condition which is evaluated, if the condition is true (Non-zero), then the control is transferred to expression 2, otherwise, the control passes to expression 3.

```
Illustration 10.7 - C++ program to find greatest of two numbers using conditional
operator
#include <iostream>
using namespace std;
int main()
{
        int a, b, largest;
        cout << "\n Enter any two numbers: ";</pre>
        cin >> a >> b;
        largest = (a>b)? a : b;
        cout << "\n Largest number : " << largest;</pre>
        return 0;
}
Output:
Enter any two numbers: 12 98
Largest number : 98
```

10.4.6 Switch statement

The switch statement is a multi-way branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression. The switch statement replaces multiple if-else sequence.

The syntax of the switch statement is;

```
switch(expression)
{
    case constant 1:
        statement(s);
        break;
    case constant 2:
        statement(s);
        break;
    .
    .
    .
    default:
        statement(s);
}
```

In the above syntax, the expression is evaluated and if its value matches against the constant value specified in one of the case statements, that respective set of statementsare executed. Otherwise, the statements under the default option are executed. The workflow of switch statement and flow chart are shown below.



Flowchart10.5: workflow of switch and flow chart

Rules:

- 1. The expression provided in the switch should result in a constant value otherwise it would not be valid.
- 2. Duplicate case values are not allowed.
- 3. The default statement is optional.
- 4. The break statement is used inside the switch to terminate a statement sequence. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- 5. The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.
- 6. Nesting of switch statements is also allowed.

Illustration 10.8 – C++ program to demonstrate switch statement

```
#include <iostream>
using namespace std;
int main()
{
        int num:
        cout << "\n Enter week day number: ";</pre>
        cin >> num;
        switch (num)
                case 1 : cout << "\n Sunday"; break;</pre>
                case 2 : cout << "\n Monday"; break;</pre>
                case 3 : cout << "\n Tuesday"; break;</pre>
                case 4 : cout << "\n Wednessday"; break;</pre>
                case 5 : cout << "\n Thursday"; break;</pre>
                case 6 : cout << "\n Friday"; break;</pre>
                case 7 : cout << "\n Saturday"; break;</pre>
                default: cout << "\n Wrong input....";
        }
}
Output:
```

Enter week day number: 6 Friday

Illustration 10.9 - C++ program to demonstrate switch statement

```
#include <iostream>
using namespace std;
int main()
{
        char grade;
        cout << "\n Enter Grade: ";</pre>
        cin >> grade;
switch(grade)
                 case 'A' : cout << "\n Excellent...";</pre>
                 break:
                 case 'B':
                 case 'C' : cout << "\n Welldone ...";</pre>
                 break;
                 case 'D' : cout << "\n You passed ...";</pre>
                 break;
        case 'E' : cout << "\n Better try again ...";</pre>
                          break;
                          default : cout << "\n Invalid Grade ...";</pre>
                 }
```

```
cout << "\n Your grade is " << grade;
    return 0;
}
Output:
Enter Grade: C
Welldone ...
Your grade is C
```

10.4.7 Switch vs if-else

"if-else" and "switch" both are selection statements. The selection statements, transfer the flow of the program to the particular block of statements based upon whether the condition is "true" or "false". However, there are some differences in their operations. These are given below:

Key Differences Between if-else and switch

- 1. Expression inside if statement decide whether to execute the statements inside if block or under else block. On the other hand, expression inside switch statement decide which case to execute.
- 2. An if-else statement uses multiple statements for multiple choices. On other hand, switch statement uses single expression for multiple choices.
- 3. If-esle statement checks for equality as well as for logical expression. On the other hand, switch checks only for equality.
- 4. The if statement evaluates integer, character, pointer or floating-point type or Boolean type. On the other hand, switch statement evaluates only character or a integer data type.
- 5. Sequence of execution is like either statement under if block will execute or statements under else block statement will execute. On the other hand the expression in switch statement decide which case to execute and if do not apply a break statement after each case it will execute till the end of switch statement.
- 6. If expression inside if turn out to be false, statement inside else block will be executed. If expression inside switch statement turn out to be false then default statements are executed.
- 7. It is difficult to edit if-else statements as it is tedious to trace where the correction is required. On the other hand, it is easy to edit switch statements as they are easy to trace.

The if statement is more flexible than switch statement.

Some important things to know about switch

There are some important things to know about switch statement. They are

- 1. A switch statement can only work for quality of comparisons.
- 2. No two case labels in the same switch can have identical values.
- 3. If character constants are used in the switch statement, they are automatically converted to their equivalent ASCII codes.
- 4. The switch statement is more efficient choice than if in a situation that supports the nature of the switch operation.

Tips: The switch statement is more efficient than if-else statement.

```
10.4.8 Nested switch
```

When a switch is a part of the statement sequence of another switch, then it is called as nested switch statement. The inner switch and the outer switch constant may or may not be the same.

The syntax of the nested switch statement is;

```
switch (expression)
{
       case constant 1:
       statement(s);
       break;
              switch(expression)
              case constant 1:
                      statement(s);
                      break:
              case constant 2:
                      statement(s);
                     break;
              default :
              statement(s);
       }
case constant 2:
statement(s);
break;
default :
statement(s);
}
```

The below program illustrates nested switch statement example. The outer switch checks for zero or non-zero and the inner switch checks for odd or even.

Illustration 10.10 C++ program to check for zero or non-zero and odd or even using nested switch statement

```
#include <iostream>
using namespace std;
int main()
{
int a = 8;
cout<<"The Number is : " <<a <<endl;</pre>
switch (a)
       {
       case 0:
       cout<<"The number is zero" << endl;
       break:
       default:
       cout<<"The number is a non-zero integer" <<endl;</pre>
       int b = a \% 2;
       switch (b)
{
       case 0:
       cout<<"The number is even" <<endl;
       break:
       case 1:
       cout<<"The number is odd" <<endl;</pre>
       break;
       }
}
       return 0;
Output
The Number is : 8
The number is a non-zero integer
The number is even
```

10.5 Iteration statements

An iteration (or looping) is a sequence of one or more statements that are repeatedly executed until a condition is satisfied. These statements are also called as control flow statements. It is used to reduce the length of code, to reduce time, to execute program and takes less memory space. C++ supports three types of iteration statements;

- for statement
- while statement
- do-while statement

All looping statements repeat a set statements as long as a specified condition is remains true. The specified condition is referred as a loop control. For all three loop statements, a true condition is any nonzero value and a zero value shows a false condition.

10.5.1 Parts of a loop

Every loop has four elements that are used for different purposes. These elements are

- Initialization expression
- Test expression
- Update expression
- The body of the loop

Initialization expression(s): The control variable(s) must be initialized before the control enters into loop. The initialization of the control variable takes place under the initialization expressions. The initialization expression is executed only once in the beginning of the loop.

Test Expression: The test expression is an expression or condition whose value decides whether the loop-body will be execute or not. If the expression evaluates to true (i.e., 1), the body of the loop executed, otherwise the loop is terminated.

In an entry-controlled loop, the test-expression is evaluated before the entering into a loop whereas in an exit-controlled loop, the test-expression is evaluated before exit from the loop.

Update expression: It is used to change the value of the loop variable. This statement is executed at the end of the loop after the body of the loop is executed.

The body of the loop: A statement or set of statements forms a body of the loop that are executed repetitively. In an entry-controlled loop, first the test-expression is evaluated and if it is nonzero, the body of the loop is executed otherwise the loop is terminated. In an exit-controlled loop, the body of the loop is executed first then the test-expression is evaluated. If the test-expression is true the body of the loop is repeated otherwise loop is terminated

10.5.2 for loop

The for loop is the easiest looping statement which allows code to be executed repeatedly. It contains three different statements (initialization, condition or test-expression and update expression(s)) separated by semicolons.

The general syntax is:

```
for (initialization(s); test-expression; update expression(s))
{
    Statement 1;
    Statement 2;
    ......
}
Statement-x;
```

The initialization part is used to initialize variables or declare variable which are executed only once, then the control passes to test-expression. After evaluation of test-expression, if the result is false, the control transferred to statement-x. If the result is true, the body of the for loop is executed, next the control is transferred to update expression. After evaluation of update expression part, the control is transferred to the test-expression part. Next the steps 3 to 5 is repeated. The workflow of for loop and flow chart are shown below.





```
      Illustration 10.11 C++ program to display numbers from 0 to 9 using for loop

      #include <iostream>

      using namespace std;

      int main ()

      {
```

int i;
for(i = 0; i< 10; i ++)
cout<< "value of i : " < <i<<endl;< td=""></i<<endl;<>
return 0;
}
Output
value of i : 0
value of i : 1
value of i : 2
value of i : 3
value of i : 4
value of i : 5
value of i : 6
value of i : 7
value of i : 8
value of i : 9

The following lines describes the working of the above given for loop:



In the above program, first the variable i is initialized, next i is compared with 10, if i is less than ten, the value of i is incremented. In this way, the numbers 0 to 9 are displayed. Once i becomes 10, it is no longer < 10. So, the control comes out of the for loop.

Illustration 10.12 C++ program to sum the numbers from 1 to 10 using for loop

```
#include <iostream>
using namespace std;
int main ()
{
    int i,sum=0;
    for(i=1; i<=10;i++)
    {
        sum=sum+i;
    }
    cout<<"The sum of 1 to 10 is "<<sum;
    return 0;
}
Output
The sum of 1 to 10 is 55</pre>
```

Variations of for loop

The for is one of the most important looping statement in C++ because it allows a several variations. These variations increase the flexibility and applicability of for loop. These variations will be discussed below:

Multiple initialization and multiple update expressions

Multiple statements can be used in the initialization and update expressions of for loop. These multiple initialization and multiple update expressions are separated by commas. For example,



}
return 0;
}
Output
The value of i is 0 The value of j is 10
The value of i is 1 The value of j is 9
The value of i is 2 The value of j is 8
The value of i is 3 The value of j is 7
The value of i is 4 The value of j is 6

In the above example, the initialization part contains two variables i and j and update expression contains i++ and j++. These two variables are separated by commas which is executed in sequential order i.e., during initialization firstly i=0 followed by j=10. Similarly, in update expression, firstly i++ is evaluated followed by j++ is evaluated.

Prefer prefix operator over postfix

Generally, the update expression contains increment/decrement operator (++ or --). In this part, always prefer prefix increment/decrement operator over postfix when to be used alone. The reason behind this is that when used alone, prefix operators are executed faster than postfix.

Optional expressions

Generally, the for loop contains three parts, i.e., initialization expressions, test expressions and update expressions. These three expressions are optional in a for loop.

Case 1

```
Illustration 10.13 (a) C++ program to sum the numbers from 1 to 10
```

```
#include <iostream>
using namespace std;
int main ()
{
    int i, sum=0, n;
    cout<<"\n Enter The value of n";
    cin>>n;
    i =1;
    for ( ; i<=10;i++)
    {
        sum += i;
    }
}</pre>
```

```
cout<<"\n The sum of 1 to " <<n<<"is "<<sum;
return 0;
}
Output
Enter the value of n 5
The sum of 1 to 5 is 15
```

In the above example, the variable i is declared and sum is initialized at the time of variable declaration. The variable i is assigned to 0 before the for loop but still the semicolon is necessary before test expression. In a for loop, if the initialization expression is absent then the control is transferred to test expression/conditional part.

Case 2

```
Illustration 10.13 (b) C++ program to sum the numbers from 1 to 10
#include <iostream>
using namespace std;
int main ()
{
int i, sum=0, n;
cout<<"\n Enter The value of n";
cin>>n;
i =1;
for (; i<=10; )
 {
       sum += i;
       ++i;
 }
cout<<"\n The sum of 1 to " <<n<<"is "<<sum;
return 0;
}
Output
Enter the value of n 5
The sum of 1 to 5 is 15
```

In the above code, the update expression is not done, but a semicolon is necessary before the update expression.



In the above code, neither the initialization nor the update expression is done in the for loop. If both or any one of expressions are absent then the control is transferred to conditional part.

Case 3

An infinite loop will be formed if a test-expression is absent in a for loop. For example,



Similarly, the following for loop also forms an infinite loop.



Empty loop

Empty loop means a loop has no statement in its body is called an empty loop. Following for loop is an empty loop:



In the above code, the for loop contains a null statement, it is an empty loop.

Similarly, the following for loop also forms an empty loop.



In the above code, the body of a for loop enclosed by braces is not executed at all because a semicolon is ended after the for loop.

Declaration of variable in a for loop

In C++, the variables can also be declared within a for loop. For instance,



A variable declared inside the block of main() can be accessed anywhere inside main() i.e., the scope of variable in main()

```
10.5.3 While loop
```

A while loop is a control flow statement that allows the loop statements to be executed as long as the condition is true. The while loop is an entry-controlled loop because the testexpression is evaluated before the entering into a loop.

The while loop syntax is:

```
while ( Test expression )
{
    Body of the loop;
}
Statement-x;
```

The control flow and flow chart of the while loop is shown below.



Flowchart 10.7: while loop control flow and while loop flowchart

In while loop, the test expression is evaluated and if the test expression result is true, then the body of the loop is executed and again the control is transferred to the while loop. When the test expression result is false the control is transferred to statement-x.

```
Hustration 10.14 C++ program to sum numbers from 1 to 10 using while loop
#include <iostream>
using namespace std;
int main ()
{
    int i=1,sum=0;
    while(i<=10)
    {
        sum=sum+i;
        i++;
    }
    cout<<"The sum of 1 to 10 is "<<sum;
return 0;
    }
    Output
The sum of 1 to 10 is 55</pre>
```

In the above program, the integer variable i is initialized to 1 and the variable sum to 0. The while loop checks the condition, i < 10, if the condition is true, the value of i, which is added to sum and i is incremented by 1. Again, the condition i < 10 is checked. Since 2 < 10, 2 is added to the earlier value of sum. This continues until i becomes 11. At this point in time, 11 < 10 evaluates to false and the while loop terminates. After the loop termination, the value of sum is displayed.

```
Illustration 10.15 C++ program to sum numbers from 1 to 10 using while loop
```

```
#include <iostream>
using namespace std;
int main ()
{
int i=1,num,avg,sum=0;
while (i < =5)
{
       cout<<"Enter the number : ";
       cin>>num;
       sum=sum+num;
       i++;
}
avg=sum/5;
cout<<"The sum is "<<sum<<endl;
cout<<"The average is "<<avg;
return 0;
}
Output
Enter the number : 1
Enter the number : 2
Enter the number : 3
Enter the number : 4
Enter the number : 5
The sum is 15
The average is 3
```

In the above program, integer variables **num** and **avg** are declared and variable i is initialized to 1 and sum to 0. The while loop checks the condition, since $i \le 5$ the condition is true, a number is read from the user and this is added to sum and i is incremented by 1. Now, the condition is $i \le 5$ is again checked. Since $2 \le 5$, the second number is obtained from the user and it is added to sum. This continues, until i becomes 6, at which point the while loop terminates. After the loop termination, the avg is computed and both sum and avg are displayed.

While loop variation

A while loop may contain several variations. It can be an empty loop or an infinite loop. An empty while loop does not have any statement inside the body of the loop except null statement i.e., just a semicolon.

```
For example

int main()

{

int i=0;

while(++i < 10000)

This is an empty loop because the while

loop does not contain any statement

}

}
```

In the above code, the loop is a time delay loop. A time delay loop is useful for pausing the program for some time.

A while loop may be infinite loop when no update statement inside the body of the loop. For example,

```
int main()
{
```



}

Similarly, there is another variation of while is also shown below:

```
int main()
```

{

}

```
int i=1;
while( ++i < 10)
cout<< "The value of i is "<<i;
return 0;
```

In the above statement while (++i < 10), first increment the value of i, then the value of i is compared with 10.

```
int main()
{
    int i=1;
    while( i++ < 10)
    cout<< "The value of i is "<<i;</pre>
```

return 0;

}

In the above statement while (i++ < 10), first the value of i is compared with 10 and then the incrementation of i takes place. When the control reaches cout<< "The value of i is "<<i statement, i has already been incremented.

10.5.4 do-while loop

The do-while loop is an exit-controlled loop. In do-while loop, the condition is evaluated at the bottom of the loop after executing the body of the loop. This means that the body of the loop is executed at least once, even when the condition evaluates false during the first iteration.

The do-while loop syntax is:

do
{
 Body of the loop;
} while(condition);

The flow control and flow chart do-while loop is shown below



Flowchart 10.8 : do-while loop control flow and do-while loop flowchart

Illustration 10.16 C++ program to display number from 10 to 1 using do-while loop

```
#include <iostream>
using namespace std;
int main ()
{
    int n = 10;
    do
    {
        cout<<n<<", ";
        n--;
}while (n>0);
}
Output
10, 9, 8, 7, 6, 5, 4, 3, 2, 1
```

In the above program, the integer variable **n** is initialized to **10**. Next the value of **n** is displayed as **10** and **n** is decremented by **1**. Now, the condition is evaluated, since **9** > **0**, again **9** is displayed and **n** is decremented to **8**. This continues, until **n** becomes equal to **0**, at which point, the condition **n** > **0** will evaluate to false and the do-while loop terminates.

10.5.5 Nesting of loops

A loop which contains another loop is called as a nested loop.

The syntax is given below:

```
for (initialization(s); test-expression; update expression(s))
{
    for (initialization(s); test-expression; update expression(s)
        {
        statement(s);
        }
statement(s);
}
```

```
while(condition)
                           do
{
                           {
      while(condition)
                           statement(s);
      {
                                 do
      statement(s);
                                  {
      }
                                 statement(s);
statement(s);
                                 }while(condition);
}
                           } while( condition );
```

Illustration 10.17 C++ program to display matrix multiplication table using nested for loop

```
#include<iostream>
using namespace std;
int main(void)
{
    cout<< "A multiplication table:" <<endl <<" 1\t2\t3\t4\t5\t6\t7\t8\t9" <<endl<< "" <<endl;
    for(int c = 1; c < 10; c++)
    {
        cout<< c << "| ";
        for(int i = 1; i< 10; i++)
        {
        cout<<i * c << '\t';
        }
        cout<<endl;
    }
return 0;
}</pre>
```

Output										
A multiplication table:										
1	2	3	4	5	6	7	8	9		
1 1	2	3	4	5	6	7	8	9		
2 2	4	6	8	10	12	14	16	18		
3 3	6	9	12	15	18	21	24	27		
4 4	8	12	16	20	24	28	32	36		
5 5	10	15	20	25	30	35	40	45		
6 6	12	18	24	30	36	42	48	54		
7 7	14	21	28	35	42	49	56	63		
8 8	16	24	32	40	48	56	64	72		
9 9	18	27	36	45	54	63	72	81		

10.6 Jump statements

Jump statements are used to interrupt the normal flow of program. Types of Jump Statements are

- goto statement
- break statement
- continue statement

10.6.1 goto statement

The goto statement is a control statement which is used to transfer the control from one place to another place without any condition in a program.

The syntax of the goto statement is;

Syntax1	Syntax2
goto label;	label:
label:	goto label;



In the syntax above, label is an identifier. When goto label; is encountered, the control of program jumps to label: and executes the code below it.

Illustration 10.18 C++ program to calculate average of given numbers using goto statement

```
# include <iostream>
using namespace std;
int main()
{
float num, average, sum = 0.0;
int i, n;
cout<< "Maximum number of inputs: ";</pre>
cin >> n;
for(i = 1; i \le n; ++i)
{
cout<< "Enter n" <<i<< ": ";
cin>>num;
if(num< 0.0)
// Control of the program move to jump:
       goto jump;
}
sum += num;
}
jump:
average = sum / (i - 1);
cout<< "\nAverage = " << average;</pre>
return 0;
}
Output
Maximum number of inputs: 5
Enter n1: 10
Enter n2: 20
Enter n3: -2
Average = 15
```

In the above program the average of numbers entered by the user is calculated. If the user enters a negative number, it is ignored the average of numbers entered. Until that point is calculated.

10.6.2 break statement

A break statement is a jump statement which terminates the execution of loop and the control is transferred to resume normal execution after the body of the loop. The following Figure. shows the working of break statement with looping statements;



break statement in for, while and do-while loop

```
Illustration 10.19 C++ program to count N numbers using break statement
```

```
#include <iostream>
Using namespace std;
int main ()
{
    int count = 0;
    do
    {
        cout<< "Count : " << count <<endl;
        count++;
        if( count > 5)
        {
            break;
        }
}while( count < 20 );
return 0;
}</pre>
```

Output			
Count:0			
Count : 1			
Count : 2			
Count: 3			
Count:4			
Count : 5			

In the above example, while condition specified the loop will iterate 20 times but, as soon as the count reaches 5, the loop is terminated, because of the break statement.

10.6.3 continue statement

The continue statement works quite similar to the break statement. Instead of terminating the loop (break statement), continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and next iteration of the loop will begin.

The following Figure describes the working flow of the continue statement



The workflow of the continue statement

Illustraion 10.20 C++ program to display numbers from 1 to 10 except 6 using continue statement

```
#include <iostream>
using namespace std;
int main()
{
for (int i = 1; i<= 10; i++) {
if (i == 6)
continue;</pre>
```

```
else

cout<<i<<"";

}

return 0;

}

Output

1 2 3 4 5 7 8 9 10
```

In the above example, the loop will iterate 10 times but, if i reaches 6, then the control is transferred to for loop, because of the continue statement.

Difference between Break and Continue

Break	Continue
Break is used to terminate the execution of the loop.	Continue is not used to terminate the execution of loop.
It breaks the iteration.	It skips the iteration.
When this statement is executed, control will come out from the loop and executes the statement immediate after loop.	When this statement is executed, it will not come out of the loop but moves/jumps to the next iteration of loop.
Break is used with loops as well as switch case.	Continue is only used in loops, it is not used in switch case.



Hands on practice:

Write C++ program to slove the following problems :

- 1. Tempeature conversion program that gives the user the option of converting Fahrenheit to Celsius or Celsius to Fahrenheit and depending upon user's choice.
- 2. The program requires the user to enter two numbers and an operator. It then carries out the specified arithmetical operation: addition, subtraction, multiplication or division of the two numbers. Finally, it displays the result.
- 3. Program to input a character and to print whether a given character is an alphabet, digit or any other character.

4. Program to print whether a given character is an uppercase or a lowercase character or a digit or any other character. use ASCII codes for it. The ASCII codes are as given below:

Characters	ASCII Range
'0' - '9'	48 - 57
'A' - 'Z'	65 - 90
'a' - 'z'	97 - 122
other characters	0- 255 excluding the above mentioned codes.

- 5. Program to calculate the factorial of an integer.
- 6. Program that print 1 2 4 8 16 32 64 128.
- 7. Program to generate divisors of an interger.
- 8. Program to print fibonacci series i.e., 0 1 1 2 3 5 8.....
- 9. Programs to produces the following design using nested loops

(a)						(b)					(c)							
Α						5	4	3	2	1	#	#	#	#	#	#	#	
Α	В					5	4	3	2			#	#	#	#	#		
Α	В	С				5	4	3					#	#	#			
Α	В	С	D			5	4							#				
Α	В	С	D	E		5												
А	В	С	D	E	F													

10. Program to check whether square root of a number is prime or not.

Points to Remember:

- A computer program is a set of statements or instructions to perform a specific task.
- There are two kinds of statements used in C++, viz Null and Compound Statement.
- Control Statement are statements that alter the sequence of flow of instaructions.
- There are three kinds of control statement used in C++. (1) Sequence Statement (2) Selection Statement (3) Iteration Statement
- *If* and *Switch* are Selection Statements.

- The Conditional Operator is an alternative for 'if else Statement'.
- The Switch Statment is a multi-way branch statement.
- Iteration Statement (looping) is use to execute a set of statements repeatedly until a condition is satisfied.
- There are three kinds Iteration Statements supported. (1) *for* (2) *While* (3) *do-While*.
- In C++ three Jump Statment are used (1) goto (2) break (3) continue

Evaluation



01					MS24				
Cho	ose the correct answe	r							
1.	What is the alternate	name of null s	nt?						
	(A) No statement		(B) Er	npty statement					
	(C) Void statement		(D) Z	FHETRA					
2.	In C++, the group of	statements sho	ould en	closed within:					
	(A) { }	(B)[]		(C) ()	(D) < >				
3.	The set of statements	that are execu	ted aga	in and again in iterat	ion is called as:				
	(A) condition	(B) loop		(C) statement	(D) body of loop				
4.	The multi way branch	ning statement	:						
	(A) if	(B) if else		(C) switch	(D) for				
5.	How many types of iteration statements?								
	(A) 2	(B) 3		(C) 4	(D) 5				
6.	How many times the	following loop	will ex	xecute? for (int i=0; i	<10; i++)				
	(A) 0	(B) 10		(C) 9	(D) 11				
7.	Which of the followir	ng is the exit co	ontrol l	oop?					
	(A) for	(B) while		(C) dowhile	(D) ifelse				
8.	Identify the odd one	from the keyw	ords of	jump statements:					
	(A) break	(B) switch		(C) goto	(D) continue				
9.	Which of the following	ng is the exit co	ontrol l	oop?					
	(A) do-while	(B) for		(C) while	(D) if-else				
10.	A loop that contains a	another loop in	nside it	s body:					
	(A) Nested loop			(B) Inner loop					
	(C) Inline loop			(D) Nesting of loop					

Answers to all the questions (2 Marks):

- 1. What is a null statement and compound statement?
- 2. What is selection statement? write it's types?
- 3. Correct the following code sigment:

if (x=1)

p= 100;

else

p = 10;

4. What will be the output of the following code:

```
int year;
```

```
cin >> year;
```

```
if (year % 100 == 0)
```

```
if (year \% 400 == 0)
```

```
cout << "Leap";</pre>
```

else

cout << "Not Leap year";</pre>

If the input given is (i) 2000 (ii) 2003 (iii) 2010?

```
5. What is the output of the following code?
```

```
for (int i=2; i<=10; i+=2)
```

cout << i;

- 6. Write a for loop that displays the number from 21 to 30.
- 7. Write a while loop that displays numbers 2, 4, 6, 8......20.
- 8. Compare an if and a ? : operator.

Part – III

Answers to all the questions (3 Marks):

1. Convert the following if-else to a single conditional statement:

if (x >= 10)

a = m + 5;

else

a = m;

2. Rewrite the following code so that it is functional:

```
v = 5;
do;
{
 total += v;
 cout << total;
while v <= 10</pre>
```

- 3. Write a C++ program to print multiplication table of a given number.
- 4. Write the syntax and purpose of switch statement.
- 5. Write a short program to print following series:
 - (a) 1 4 7 10..... 40

Part – IV

Answers to all the questions (5 Marks):

- 1. Explain control statement with suitable example.
- 2. What entry control loop? Explain any one of the entry control loop with suitable example.
- 3. Write a program to find the LCM and GDC of two numbers.
- 4. Write programs to find the sum of the following series:

(a)
$$x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \frac{x^5}{5!} - \frac{x^6}{6!}$$

(b) $x + \frac{x^2}{2} + \frac{x^3}{3} + \dots + \frac{x^n}{n}$

5. Write a program to find sum of the series

$$S = 1 + x + x^2 + \dots + x^n$$

Reference:

- (1) Object Oriented Programming with C++ (4th Edition), Dr. E. Balagurusamy, Mc.Graw Hills.
- (2) The Complete Reference C++ (Forth Edition), Herbert Schildt. Mc.Graw Hills.
- (3) Computer Science with C++ (A text book of CBSE XI and XII), Sumita Arora, Dhanpat Rai & Co.

Unit III | Introduction to C++

CHAPTER

Functions

Learning Objectives

After learning this chapter, the students will be able to

- Understand the Definition of Functions and uses of Functions
- Understand the Types of Functions pre-defined and user-defined functions
- Apply mathematical functions for solving problems.
- Use String and Character functions for the manipulation of String and Character data
- Implement modular programming by creating functions
- Understand the role of arguments and compare different methods of the arguments
- Recognizes the scope of variables and functions in a program.

11.1 INTRODUCTION

A large program can typically be split into small sub-programs (blocks) called as functions where each sub-program can perform some specific functionality. Functions reduce the size and complexity of a program, makes it easier to understand, test, and check for errors. The functions which are available by default known as **"Built-in"** functions and user can create their own functions known as **"User-defined"** functions.

- Built-in functions Functions which are available in C++ language standard library.
- User-defined functions Functions created by users.

11.2 Need for Functions

To reduce size and complexity of the program we use Functions. The programmers can make use of sub programs either writing their own functions or calling them from standard library.

1. Divide and Conquer

- Complicated programs can be divided into manageable sub programs called functions.
- A programmer can focus on developing, debugging and testing individual functions.
- Many programmers can work on different functions simultaneously.



- 2. Reusability:
- Few lines of code may be repeatedly used in different contexts. Duplication of the same code can be eliminated by using functions which improves the maintenance and reduce program size.
- Some functions can be called multiple times with different inputs.

11.3 Types of Functions

Functions can be classified into two types,

- 1. Pre-defined or Built-in or Library Functions
- 2. User-defined Function.

C++ provides a rich collection of functions ready to be used for various tasks. The tasks to be performed by each of these are already written, debugged and compiled, their definitions alone are grouped and stored in files called **header files**. Such ready-to-use sub programs are called **pre-defined functions or built-in functions**.

C++ also provides the facility to create new functions for specific task as per user requirement. The name of the task and data required (arguments) are decided by the user and hence they are known as **User-defined functions**.

11.4 C++ Header Files and Built-in Functions

Header files provide function prototype and definitions for library functions. Data types and constants used with the library functions are also defined in them. A header file can be identified by their file extension **.h.** A single header file may contain multiple built-in functions.

For example: **stdio.h** is a header file contains pre-defined **"standard input/output"** functions.

11.4.1 Standard input/output (stdio.h)

This header file defines the standard I/O predefined functions getchar(), putchar(), gets(), puts() and etc.

11.4.1.1 getchar() and putchar() functions

The predefined function **getchar()** is used to get a single character from keyboard and **putchar()** function is used to display it.
```
Program 11.1 C++ code to accept a character and displays it
#include<iostream>
#include<stdio.h>
using namespace std;
int main()
{
    cout<<"\n Type a Character : ";
    char ch = getchar();
    cout << "\n The entered Character is: ";
    putchar(ch);
    return 0;
}
Output:
Type a Character : T
The entered Character is: T</pre>
```

```
11.4.1.2. gets() and puts() functions
```

Function gets() reads a string from standard input and stores it into the string pointed by the variable. Function **puts**() prints the string read by **gets**() function in a newline.

```
Program 11.2 C++ code to accepts and display a string
 #include<iostream>
 #include<stdio.h>
 using namespace std;
 int main()
 {
         char str[50];
         cout<<"Enter a string : ";</pre>
         gets(str);
         cout<<"You entered: "
         puts(str);
         return(0);
 }
 Output :
 Enter a string : Computer Science
 You entered: Computer Science
```

11.4.2 Character functions (ctype.h)

This header file defines various operations on characters. Following are the various character functions available in C++. The header file **ctype.h** is to be included to use these functions in a program.

11.4.2.1.isalnum()

This function is used to check whether a character is **alphanumeric or not**. This function returns non-zero value if c is a digit or a letter, else it returns 0.

Syntax:

```
int isalnum (char c)
```

Example :

```
int r = isalnum('5');
cout << isalnum('A') <<'\t'<<r;</pre>
```

But the statements given below assign 0 to the variable n, since the given character is neither an alphabet nor a digit.

```
char c = '$';
int n = isalnum(c);
cout<<c;</pre>
```

Output:

0

Program 11.3

Output-1:

Type a Character :A The Return Value of isalnum(ch) is :1 **Output-2:** Type a Character :? The Return Value of isalnum(ch) is :0

11.4.2.2. isalpha()

The isalpha() function is used to check whether the given character is an alphabet or not.

Syntax:

int isalpha(char c);

This function will return 1 if the given character is an alphabet, and 0 otherwise 0. The following statement assigns 0 to the variable n, since the given character is not an alphabet.

int n = isalpha('3');

But, the statement given below displays 1, since the given character is an alphabet.

```
cout << isalpha('a');</pre>
```

```
Output-1:
Enter a charater: A
The Return Value of isalpha(ch) is :1
Output-2:
Enter a charater: 7
The Return Value of isalpha(ch) is :0
```

11.4.2.3 isdigit()

This function is used to check whether a given character is a digit or not. This function will return 1 if the given character is a digit, and 0 otherwise.

Syntax:

int isdigit(char c);

When the following program is executed, the value of the variable n will be 1, since the given character is not a digit.

```
Program 11.5
using namespace std;
#include<iostream>
#include<ctype.h>
int main()
{
       char ch;
        cout << "\n Enter a Character: ";</pre>
       cin >> ch;
       cout<<"\n The Return Value of isdigit(ch) is :" << isdigit(ch) ;</pre>
}
Output-1
       Enter a Character: 3
       The Return Value of isdigit(ch) is :1
Output-2
       Enter a Character: A
       The Return Value of isdigit(ch) is :0
```

*Return 0; (Not Compulsory in latest compilers)

11.4.2.4. islower()

This function is used to check whether a character is in lower case (small letter) or not. This functions will return a non-zero value, if the given character is a lower case alphabet, and 0 otherwise.

Syntax:

int islower(char c);

After executing the following statements, the value of the variable **n** will be **1** since the given character is in lower case.

char ch = 'n';

int n = islower(ch);

But the statement given below will assign 0 to the variable n, since the given character is an uppercase alphabet.

int n = islower('P');

11.4.2.5. isupper()

This function is used to check the given character is uppercase. This function will return 1 if true otherwise 0. For the following examples value **1** will be assigned to **n** and **0** for m.

int n=isupper('A');

int m=isupper('a');

11.4.2.6. toupper()

This function is used to convert the given character into its uppercase. This function will return the upper case equivalent of the given character. If the given character itself is in upper case, the output will be the same.

Syntax:

char toupper(char c);

The following statement will assign the character constant 'K' to the variable c.

char c = toupper('k');

But, the output of the statement given below will be 'B' itself.

cout <<toupper('B');</pre>

11.4.2.7. tolower()

This function is used to convert the given character into its lowercase. This function will return the lower case equivalent of the given character. If the given character itself is in lower case, the output will be the same.

Syntax:

char tolower(char c);

The following statement will assign the character constant '**k**' to the variable c.

char c = tolower('K');

But, the output of the statement given below will be 'b' itself.

cout <<tolower('b');</pre>

11.4.3 String manipulation (string.h)

The library **string.h** (also referred as cstring) has several common functions for dealing with strings stored in arrays of characters. The string.h header file to be included before using any string function.

11.4.3.1 strcpy()

The **strcpy()** function takes two arguments: target and source. It copies the character string pointed by the source to the memory location pointed by the target. The null terminating character (**\0**) is also copied.



Output:

String in Source Before Copied :Computer Science String in Target Before Copied :target String in Target After strcpy function Executed :Computer Science

11.4.3.2 strlen()

The **strlen()** takes a null terminated byte string source as its argument and returns its length. The length does not include the null(\0) character.

11.4.3.3 strcmp()

The **strcmp()** function takes two arguments: string1 and string2. It compares the contents of string1 and string2 lexicographically.

The strcmp() function returns a:

- Positive value if the first differing character in string1 is greater than the corresponding character in string2. (ASCII values are compared)
- Negative value if the first differing character in string1 is less than the corresponding character in string2.
- 0 if string1 and string2 are equal.

```
Program 11.8
```

```
#include <string.h>
#include <iostream>
using namespace std;
int main()
{
       char string1[] = "Computer";
       char string2[] = "Science";
       int result;
       result = strcmp(string1,string2);
       if(result==0)
       {
       cout<<"String1 : "<<string1<<" and String2 : "<<string2 <<"Are Equal";</pre>
       }
       if (result<0)
       {
       cout<<"String1:"<<string1<<" and String2: "<<string2 <<" Are Not Equal";
       }
}
Output
       String1 : Computer and String2 : Science Are Not Equal
```

```
11.4.3.4 strcat()
```

The **strcat**() function takes two arguments: target and source. This function appends copy of the character string pointed by the source to the end of string pointed by the target.

Program 11.9

The **strupr()** function is used to convert the given string into Uppercase letters.

Converted the Source string computer science into Upper Case is COMPUTER SCIENCE

```
11.4.3.6 strlwr()
```

The **strlwr()** function is used to convert the given string into Lowercase letters.

11.4.4 Mathematical functions (math.h)

Most of the mathematical functions are defined in math.h header file which includes basic mathematical functions.

```
11.4.4.1 cos() function
```

The cos() function takes a single argument in radians. The cos() function returns the value in the range of [-1, 1]. The returned value is either in double, float, or long double.

```
Program 11.12
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
        double x = 0.5, result;
        result = cos(x);
        cout << "COS("<<x<<")= "<<result;
}
Output:
        COS(0.5)= 0.877583</pre>
```

11.4.4.2 sqrt() function

The **sqrt()** function returns the square root of the given value of the argument. The **sqrt()** function takes a single non-negative argument. If a **negative value** is passed as an argument to **sqrt()** function, a **domain error occurs**.

```
Program 11.13
```

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
        double x = 625, result;
        result = sqrt(x);
        cout << "sqrt("<<x<<") = "<<result;
        return 0;
}
Output:
        sqrt(625) = 25</pre>
```

11.4.4.3 sin() function

The **sin()** function takes a single argument in radians. The **sin()** function returns the value in the range of [-1, 1]. The returned value is either in double, float, or long double.

11.4.4 pow() function

The **pow()** function returns base raised to the power of exponent. If any argument passed to **pow()** is long double, the return type is promoted to long double. If not, the return type is double. The **pow()** function takes two arguments:

- **base** the base value
- exponent exponent of the base

```
Program 11.14
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
       double base, exponent, result;
       base = 5;
       exponent = 4;
       result = pow(base, exponent);
       cout << "pow("<<base << "^" << exponent << ") = " << result;
       double x = 25;
       result = sin(x);
       cout << "\nsin("<<x<<")= "<<result;
       return 0;
}
Output:
       pow(5^4) = 625
       sin(25) = -0.132352
```

11.4.5 Generating Random Numbers

The **srand()** function in C++ seeds the pseudo random number generator used by the **rand()** function. The seed for **rand()** function is 1 by default. It means that if no **srand()** is called before **rand()**, the **rand()** function behaves as if it was seeded with **srand(1)**. The **srand()** function takes an unsigned integer as its parameter which is used as seed by the **rand()** function. It is defined in<**cstdlib**>or <**stdlib.h**>header file.

Program 11.15

```
#include<iostream>
#include<cstdlib.h>
using namespace std;
int main()
{
       int random = rand(); /* No srand() calls before rand(), so seed = 1^*/
       cout << "\nSeed = 1, Random number = " << random;</pre>
       srand(10);
       /* Seed = 10 */
       random = rand();
       cout << "\n\nSeed = 10, Random number = " << random;</pre>
       return 0;
}
OUTPUT:
       Seed = 1, Random number = 41
       Seed = 10, Random number = 71
```

11.5 User-defined Functions

11.5.1 Introduction

We can also define new functions to perform a specific task. These are called as userdefined functions. User-defined functions are created by the user. A function can optionally define input parameters that enable callers to pass arguments into the function. A function can also optionally return a value as output. Functions are useful for encapsulating common operations in a single reusable block, ideally with a name that clearly describes what the function does.

11.5.2 Function Definition

In C++, a function must be defined before it is used anywhere in the program. The general syntax of a function definition is:

Return_Data_Type Function_name(parameter list)

{

Body of the function

}

Note:

- 1. The Return_Data_Type is any valid data type of C++.
- 2. The Function_name is a user-defined identifier.
- 3. The parameter list, which is optional, is a list of parameters, i.e. a list of variables preceded by data types and separated by commas.
- 4. The body of the function comprises C++ statements that are required to perform the intended task of this function.
- 11.5.3 Function Prototype

C++ program can contain any number of functions. But, it must always have **only one main() function** to begin the program execution. We can write the definitions of functions in any order as we wish. We can define the main() function first and all other functions after that or we can define all the needed functions prior to main(). Like a variable declaration, a function must be declared before it is used in the program. The declaration statement may be given outside the main() function.





The prototype above provides the following information to the compiler:

- The return value of the function is of type long.
- **fact** is the name of the function.
- the function is called with two arguments:

The first argument is of int **data** type.

The second argument is of **double** data type.

int display(int , int) // function prototype//

The above function prototype provides details about the return data type, name of the function and a list of formal parameters or arguments.

11.5.4 Use of void command

void type has two important purposes:

- To indicate the function does not return a value
- To declare a generic pointer.

void data type indicates the compiler that the function does not return a value, or in a larger context void indicates that it holds nothing.

Notes

For Example:

void fun(void)

The above function prototype tells compiler that the function **fun()** neither receives values from calling program nor return a value to the calling program.

11.5.5 Accessing a function

The user-defined function should be called explicitly using its name and the required arguments to be passed. The compiler refers to the function prototype to check whether the function has been called correctly. If the argument type does not match exactly with the data type defined in the prototype, the compiler will perform type conversion, if possible. If type conversion is impossible, the compiler generates an error message.

Example :

1	display()	calling the function without a return value and without any argument	
2	display (x, y)	calling the function without a return value and with arguments	
3	x = display() calling the function with a return value and withou any argument		
4	x = display(x, y)	calling the function with a return value and with arguments	

11.5.5.1 Formal Parameters and Actual Parameters or Arguments

Arguments or parameters are the means to pass values from the calling function to the called function. The variables used in the function definition as parameters are known as formal parameters. The constants, variables or expressions used in the function call are known as actual parameters.



Figure 11.2 Formal and Actual Parameters

11.5.5.2 Default arguments

In C++, one can assign default values to the formal parameters of a function prototype. The Default arguments allows to omit some arguments when calling the function.

When calling a function,

- For any missing arguments, complier uses the values in default arguments for the called function.
- The default value is given in the form of variable initialization.

Example : void defaultvalue(int n1=10, n2=100);

• The default arguments facilitate the function call statement with partial or no arguments.

Example : defaultvalue(x,y);

defaultvalue(200,150); defaultvalue(150); defaultvalue(x,150); • The default values can be included in the function prototype from right to left, i.e., we cannot have a default value for an argument in between the argument list.

Example : void defaultvalue(int n1=10, n2);//invalid prototype void defaultvalue(int n1, n2 = 10);//valid prototype

11.5.5.3 Constant Arguments

The constant variable can be declared using **const** keyword. The **const** keyword makes variable value stable. The constant variable should be initialized while declaring. The **const** modifier enables to assign an initial value to a variable that cannot be changed later inside the body of the function.

Syntax :

<returntype><functionname> (const <datatype variable=value>)

Example:

- int minimum(const int a=10);
- float area(const float pi=3.14, int r=5);

Program 11.16

```
#include <iostream>
using namespace std;
double area(const double r,const double pi=3.14)
{
        return(pi*r*r);
int main ()
        double rad, res;
       cout<<"\nEnter Radius :";</pre>
        cin>>rad;
        res=area(rad);
        cout << "\nThe Area of Circle ="<<res;</pre>
        return 0;
}
Output:
Enter Radius :5
The Area of Circle =78.5
```

If the variable value "r" is changed as **r=25**; inside the body of the function **"area"** then compiler will throw an error as **"assignment of read-only parameter 'r"**

```
double area(const double r,const double pi=3.14)
{
    r=25;
    return(pi*r*r);
```

}

11.6 Methods of calling functions

In C++, the arguments can be passed to a function in two ways. Based on the method of passing the arguments, the function calling methods can be classified as Call by Value method and Call by Reference or Address method.

11.6.1 Call by value Method

This method copies the value of an actual parameter into the formal parameter of the function. In this case, changes made to formal parameter within the function will have no effect on the actual parameter.

```
Program 11.17
#include<iostream>
using namespace std;
void display(int x)
{
       int a=x^*x;
       cout<<"\n\nThe Value inside display function (a * a):"<<a;</pre>
int main()
{
        int a;
       cout<<"\nExample : Function call by value:";</pre>
        cout<<"\n\nEnter the Value for A :";</pre>
        cin>>a;
       display(a);
        cout<<"\n\nThe Value inside main function "<<a;
       return(0);
}
Output :
Example : Function call by value
Enter the Value for A : 5
The Value inside display function (a * a) : 25
The Value inside main function 5
```

11.6.2 Call by reference or address Method

This method copies the address of the actual argument into the formal parameter. Since the address of the argument is passed ,any change made in the formal parameter will be reflected back in the actual parameter.

```
Program 11.18
#include<iostream>
using namespace std;
void display(int &x) //passing address of a//
{
        x=x^*x;
       cout<<"\n\nThe Value inside display function (n1 x n1) :"<<x ;
}
int main()
{
int n1;
cout<<"\nEnter the Value for N1 :";
cin>>n1;
cout<<"\nThe Value of N1 is inside main function Before passing : "<< n1;
display(n1);
cout << "\nThe Value of N1 is inside main function After passing (n1 \times n1) : "<< n1; return(0);
Output :
Enter the Value for N1:45
The Value of N1 is inside main function Before passing : 45
The Value inside display function (n1 x n1) :2025
The Value of N1 is inside main function After passing (n1 \times n1) : 2025
```

Note that the only change in the **display()** function is in the function header. The **&** symbol in the declaration of the parameter **x** means that the argument is a reference variable and hence the function will be called by passing reference. Hence when the argument **num1** is passed to the **display()** function, the variable **x** gets the address of **num1** so that the location will be shared. In other words, the variables **x** and **num1** refer to the same memory location. We use the name **num1** in the main() function, and the name **x** in the **display()** function to refer the same storage location. So, when we change the value of **x**, we are actually changing the value of **num1**.

11.6.3 Inline function

Normally the call statement to a function makes a compiler to jump to the functions (the definition of the functions are stored in STACKS) and also jump back to the instruction

following the call statement. This reduces the speed of program execution. Inline functions can be used to reduce the overheads like STACKS for small function definition.

An inline function looks like normal function in the source file but inserts the function's code directly into the calling program. To make a function inline, one has to insert the keyword **inline** in the function header.

Syntax :

inline returntype functionname(datatype parametername1, ... datatype parameternameN)

Advantages of inline functions:

- Inline functions execute faster but requires more memory space.
- Reduce the complexity of using STACKS.

```
Program 11.19
```

```
#include <iostream>
using namespace std;
inline float simpleinterest(float p1,float n1, float r1)
{
        float si1=(p1*n1*r1)/100;
        return(si1);
}
int main ()
{
        float si,p,n,r;
        cout<<"\nEnter the Principle Amount Rs. :";</pre>
        cin>>p;
        cout<<"\nEnter the Number of Years :";</pre>
        cin>>n;
        cout<<"\nEnter the Rate of Interest :";</pre>
        cin>>r:
        si=simpleinterest(p,n,r);
        cout << "\nThe Simple Interest = Rs."<<si;</pre>
        return 0;
}
Output:
Enter the Principle Amount Rs. :60000
Enter the Number of Years
                               :10
Enter the Rate of Interest
                            :5
The Simple Interest = Rs.30000
```

Though the above program is written in the normal function definition format during compilation the function code (**p1*n1*r1**)/100 will be directly inserted in the calling statement i.e.**si=simpleinterest(p,n,r)**; this makes the calling statement to change as **si=** (**p1*n1*r1**)/100;

11.7 Different forms of User-defined Function declarations

11.7.1 A Function without return value and without parameter

The following program is an example for a function with no return and no arguments passed .

The name of the function is **display()**, its return data type is void and it does not have any argument.

```
Program 11.20
#include<iostream>
using namespace std;
void display()
{
    cout<<"First C++ Program with Function";
}
int main()
{
    display(); // Function calling statement//
    return(0);
}
Output :
    First C++ Program with Function</pre>
```

11.7.2 A Function with return value and without parameter

The name of the function is **display()**, its return type is int and it does not have any argument. The **return** statement returns a value to the calling function and transfers the program control back to the calling statement.

```
Program 11.21
#include<iostream>
using namespace std;
int display()
       int a, b, s;
       cout<<"Enter 2 numbers: ";</pre>
       cin>>a>>b;
       s=a+b;
       return s;
       }
int main()
{
        int m=display();
       cout<<"\nThe Sum="<<m;</pre>
       return(0);
}
Output :
       Enter 2 numbers: 10 30
       The Sum=40
```

11.7.3 A Function without return value and with parameter

The name of the function is **display()**, its return type is void and it has two parameters or arguments **x** and **y** to receive two values. The **return** statement returns the control back to the calling statement.

Program 11.22

```
#include<iostream>
using namespace std;
void display(int x, int y)
{
     int s=x+y;
     cout<<"The Sum of Passed Values: "<<s;
}</pre>
```

```
int main()
{
    int a,b;
    cout<<"\nEnter the First Number :";
    cin>>a;
    cout<<"\nEnter the Second Number :";
    cin>>b;
    display(a,b);
    return(0);
}
Output :
Enter the First Number :50
Enter the Second Number :45
The Sum of Passed Values: 95
```

11.7.4 A Function with return value and with parameter

The name of the function is display(), its return type is int and it has two parameters or arguments \mathbf{x} and \mathbf{y} to receive two values. The return statement returns the control back to the calling statement.

```
Program 11.23
#include<iostream>
using namespace std;
int display(int x, int y)
{
       int s=x+y;
       return s;
}
int main()
{
        int a,b;
        cout<<"\nEnter the First Number :";</pre>
        cin>>a;
        cout<<"\nEnter the Second Number :";</pre>
        cin>>b;
        int s=display(a,b);
        cout<<"\nExample:Function with Return Value and with Arguments";
        cout<<"\nThe Sum of Passed Values: "<<s;</pre>
       return(0);
}
```

Output : Enter the First Number :45 Enter the Second Number :67 Example: Function with Return Value and with Arguments The Sum of Passed Values: 112

11.8 Returning from function

Returning from the function is done by using the **return** statement.

The **return** statement stops execution and returns to the calling function. When a **return** statement is executed, the function is terminated immediately at that point.

11.8.1 The return statement

The **return** statement is used to return from a function. It is categorized as a jump statement because it terminates the execution of the function and transfer the control to the called statement. A **return** may or may not have a value associated with it. If return has a value associated with it, that value becomes the return value for the calling statement. Even for void function return statement without parameter can be used to terminate the function.

Syntax:

return expression/variable;

Example : return(a+b); return(a);

return; // to terminate the function

11.8.2 The Returning values:

The functions that return no value is declared as void. The data type of a function is treated as **int**, if no data type is explicitly mentioned. For example,

For Example :

int add (int, int);

add (int, int);

In both prototypes, the return value is int, because by default the return value of a function in C++ is of type **int**. Look at the following examples:

Sl.No	Function Prototype	Return type
1	int sum(int, float)	int
2	float area(float, float)	float
3	char result()	char
4	double fact(int n)	double

Returning Non-integer values

A string can also be returned to a calling statement.

```
Program 11.24
#include<iostream>
#include<string.h>
using namespace std;
char *display()
{
       return ("chennai");
}
int main()
{
       char s[50];
       strcpy(s,display());
       cout<<"\nExample:Function with Non Integer Return"<<s;</pre>
       return(0);}
Output :
Example: Function with Non Integer Return Chennai
```

11.8.3 The Returning by reference

```
Program 11.25
#include<iostream>
using namespace std;
int main()
{
    int n1=150;
}
```

```
int &n1ref=n1;
cout<<"\nThe Value of N1 = "<<n1<<" and n1Reference = "<<n1ref;
n1ref++;
cout<<"\nAfter n1 increased the Value of N1 = "<<n1;
cout<<" and n1Reference = "<<n1ref;
return(0);
}
Output:
The Value of N1 = 150 and n1Reference = 150
After n1 increased the Value of N1 = 151 and n1Reference = 151
```

Notes

The variable n1ref is alias to n1. Hence when the value of n1ref is altered automatically the value of n1 is changed.

The two variables n1 and n1ref shares same memory or reference.

11.9 Recursive Function

A function that calls itself is known as recursive function. And, this technique is known as recursion.

Example 1: Factorial of a Number Using Recursion

```
Program 11.26
#include <iostream>
using namespace std;
int factorial(int); // Function prototype //
int main()
{
    int no;
    cout<<"\nEnter a number to find its factorial: ";
    cin >> no;
    cout << "\nFactorial of Number " << no <<" = " << factorial(no);
    return 0;
}</pre>
```

```
int factorial(int m)
{
    if (m > 1)
    {
        return m*factorial(m-1);
    }
    else
    {
        return 1;
}
}
Output :
Enter a number to find its factorial: 5
Factorial of Number 5 = 120
```

Note: Function prototype is mandatory since the function factorial() is given after the main() function.

```
Example 2: Finding HCF of any to number using Recursion
```

```
Program 11.27
#include <iostream>
using namespace std;
//Function to find HCF //
int hcf(int n1, int n2)
{
       if (n2!=0)
       return hcf(n2, n1 % n2);
       else
       return n1;
}
int main()
{
       int num1, num2;
       cout << "Enter two positive integers: ";</pre>
       cin >> num1 >> num2;
       cout << "Highest Common Factor (HCF) of " << num1;</pre>
       cout<< " & " << num2 << " is: " << hcf(num1, num2);
       return 0;
}
```

Output:

Enter two positive integers: 350 100 Highest Common Factor (HCF) of 350 & 100 is: 50

Notes

Function prototype is not necessary since the function hcf() is given before the main() function.

11.10 Scope Rules of Variables

Scope refers to the accessibility of a variable. There are four types of scopes in C++. They are: **Local scope, Function scope, File scope** and **Class scope**.

11.10.1 Introduction

A scope is a region or life of the variable and broadly speaking there are three places, where variables can be declared,

- Inside a block which is called local variables.
- Inside a function is called function variables.
- Outside of all functions which is called global variables.
- Inside a class is called class variable or data members.

11.10.2 Local Scope:

- A local variable is defined within a block. A block of code begins and ends with curly braces
 { }.
- The scope of a local variable is the block in which it is defined.
- A local variable cannot be accessed from outside the block of its declaration.

• A local variable is created upon entry into its block and destroyed upon exit.

```
Program 11.28 (a)
//Demo to test Local Scope//
#include<iostream>s
using namespace std;
int main ()
{
int a, b;
a = 10;
b = 20;
if (a > b)
{
int temp; //local to this if block//
temp = a;
a = b;
b = temp;
}
cout <<"\n Descending order .... \n";</pre>
cout <<a <<"\t"<<b;
return(0);
}
Output:
Descending order ....
10
     20
```

Program 11.28 (b)

```
//Demo to test Local Scope//
#include<iostream>s
using namespace std;
int main ()
{
int a, b;
a = 10;
b = 20;
if (a > b)
{
int temp; //local to this if block//
temp = a;
a = b;
b = temp;
}
cout <<temp;</pre>
return(0);
}
```

In function 'int main()':

[Error] 'temp' was not declared in this scope

On compilation the Program 11.28(b), the compiler prompts an error message: The variable **temp** is not accessible. Because the life time of a local variable is the life time of a block in its state of execution. Local variables die when its block execution is completed.

11.10.3 Function Scope:

- The scope of variables declared within a function is extended to the function block, and all sub-blocks therein.
- The life time of a function scope variable, is the life time of the function block. The scope of formal parameters is function scope.

```
Program 11.29 (a)
//Demo to test Function Scope//
#include<iostream>
using namespace std;
void add(int x, int y)
{
       int m=x+y; //'m' declared within function add()//
       cout << "\nThe Sum = "<< m;
}
int main ()
{
int a, b;
a = 10;
b = 20;
add(a,b);
return(0);
}
```

Program 11.29 (b)

//Demo to test Function Scope//
#include<iostream>
using namespace std;

```
void add(int x, int y)
{
     int m=x+y; //'m' declared within function add()//
     cout<<"\nThe Sum = "<<m;
}
int main ()
{
    int a, b;
    a = 10;
    b = 20;
    add(a,b);
    cout<<m; //'m' declared within function add()//
return(0);
}
Output:
The Sum = 30</pre>
```

```
Note : In function 'int main()':
```

[Error] 'm' was not declared in this scope

11.10.4 File Scope:

- A variable declared above all blocks and functions (including main ()) has the scope of a file. The life time of a file scope variable is the life time of a program.
- The file scope variable is also called as **global variable**.

```
Program 11.30
//Demo to test File or global Scope//
#include<iostream>
using namespace std;
int file_var=20; //Declared within File//
void add(int x, int y)
{
    int m=x+y+file_var;
    cout<<"\n The Sum = "<<m;
}</pre>
```

```
int main ( )
{
    int a, b;
    a = 10;
    b = 20;
    add(a,b);
    cout<<"\nThe File Variable = "<<file_var;
    return(0);
    }
Output:
The Sum = 50
The File Variable =20</pre>
```

11.10.5 Class Scope:

- A class is a new way of creating and implementing a user defined data type. Classes provide a method for packing together data of different types.
- Data members are the data variables that represent the features or properties of a class.

class student	The class student contains
{	mark1, mark2 and total are
private :	data variables. Its scope is
int mark1, mark2, total;	within the class student
};	only.

Note: The class scope will be discussed later in chapter "Classes and Object".

11.10.6 Scope resolution operator

- The scope operator reveals the hidden scope of a variable. The scope resolution operator (::) is used for the following purposes.
- To access a Global variable when there is a Local variable with same name. An example using Scope Resolution Operator.

Program 11.31

```
// Program to show that we can access a global variable
// using scope resolution operator :: when there is a local
// variable with same name //
#include<iostream>
using namespace std;
int x=45; // Global Variable x
int main()
{
 int x = 10; // Local Variable x
 cout << "\nValue of global x is " << ::x;</pre>
 cout << "\nValue of local x is " << x:
 return 0;
}
Output:
Value of global x is 45
Value of local x is 10
```

Points to Remember:

- A large program can typically be split into smaller sized blocks called as functions.
- Functions can be classified into Predefined or Built-in or Library Functions and User-defined Functions.
- User-defined functions are created by the user.
- The void function tells the compiler that the function returns nothing.
- The return statement returns a value to the calling function and transfers the program control back to the calling function.

- Default the data type of a function in C++ is of type int.
- A function that calls itself is known as recursive function.
- Scope refers to the accessibility of a variable.
- There are four types of Scopes. They are: Local scope, Function scope, File scope and Class scope.
- The scope operator (::) reveals the hidden scope of a variable.



Hands on practice:

Write C++ program to slove the following problems :

- 1. Program that reads two strings and appends the first string to the second. For example, if the first string is entered as Tamil and second string as nadu, the program should print Tamilnadu. Use string library header.
- 2. Program that reads a string and converts it to uppercase. Include required header files.
- 3. Program that checkos whether a given character is an alphabety or not. If it is an alphabet, whether it is lowercase character or uppercase character? Include required header files.
- 4. Program that checks whether the given character is alphanumeric or a digit. Add appropriate header file.
- 5. Write a function called zero_small () that has two integer arguments being passed by reference and sets smaller of the two numbers to 0. Write the main program to access this function.
- 6. Write definition for a function sumseries () in c++ with two arguments/ parameters double x and int n. The function should return a value of type double and it should perform sum of the following series:

x-x2 /3! + x3 / 5! - x4 / 7! + x5 / 9! -... upto n terms.

7. Program that invokes a function calc () which intakes two intergers and an arithmetic operator and prints the corresponding result.







Choose the best answer

- 1. Which of the following header file defines the standard I/O predefined functions ?
 - A) stdio.h B) math.h C) string.h D) ctype.h
- 2. Which function is used to check whether a character is alphanumeric or not.
 - A) isalpha()B) isdigit()C) isalnum()D) islower()
- 3. Which function begins the program execution ?A) isalpha() B) isdigit() C) main() D) islower()

- 4. Which of the following function is with a return value and without any argument ?
 - A) x=display(int, int) B) x=display() C) y=display(float) D) display(int)
- 5. Which is return data type of the function prototype of add(int, int); ?
 - A) int B) float C) char D) double
- 6. Which of the following is the scope operator ?
 - A) > B) & C) % D) ::

Part –II

Answer to all the questions (2 Marks):

- 1. Define Functions.
- 2. Write about strlen() function.
- 3. What are importance of void data type.
- 4. What is Parameter and list its types?
- 5. Write a note on Local Scope.

Part – III

Answer to all the questions (3 Marks):

- 1. What is Built-in functions ?
- 2. What is the difference between isuppr() and toupper() functions ?
- 3. Write about strcmp() function.
- 4. Write short note on pow() function in C++.
- 5. What are the information the prototype provides to the compiler ?
- 6. What is default arguments ? Give example.

Part –IV

Answer to all the questions (5 Marks):

- 1. Explain Call by value method with suitable example.
- 2. What is Recursion? Write a program to find GCD using recursion.
- 3. What are the different forms of function return? Explain with example.
- 4. Explain scope of variable with example.
- 5. Write a program to accept any integer number and reverse it.

Unit III | Introduction of C++



Arrays and Structures

Learning Objectives

After learning this chapter, the students will be able to

- Know the structured data type using arrays.
- Know the types of arrays.
- Writing programs to manuplates different types of arrays.

12.1 Introduction



The variables are used to store data. These variables are the one of the basic building blocks in C++. A single variable is used to store a single value that can be used anywhere in the memory. In some situations, we need to store multiple values of the same type. In that case, it needs multiple variables of the same data type. All the values are stored randomly anywhere in the memory.

For example, to store the roll numbers of the 100 students, it needs 100 variables named as roll1, roll2, roll3,.....roll100. It becomes very difficult to declare 100 variables and store all the roll numbers. In C++, the concept of Array helps to store multiple values in a single variable. Literally, the meaning of **Array is "More than one". In other words, array is an easy way of storing multiple values of the same type referenced by a common name". An array is also a derived data type in C++.**

"An array is a collection of variables of the same type that are referenced by a common name". In an array, the values are stored in a fixed number of elements of the same type sequentially in memory. Therefore, an integer array holds a sequence of integers; a character array holds a sequence of characters, and so on. The size of the array is referred to as its dimension.

12.2 Types of Arrays:

There are different types of arrays used in C++. They are:

• One-dimensional arrays

- Two-dimensional arrays
- Multi-dimensional arrays

12.2.1 One-dimensional array

This is the simplest form of an array. A one dimensional array represents values that are stored in a single row or in a single column.

Declaration

Syntax:

<data type><array_name>[<array_size>];

data_type declares the basic type of the array, which is the type of each element in the array.

array_name specifies the name with which the array will be referenced.

array_size defines how many elements the array will hold. Size should be specified with square brackets [].

Example:

int num[10];

In the above declaration, an array named "num" is declared with 10 elements (memory space to store 10 different values) as integer type.

To the above declaration, the compiler allocated 10 memory locations (boxes) in the common name "num" as given below



Each element (Memory box) has a unique index number starting from 0 which is known as "subscript". The subscript always starts with 0 and it should be an unsigned integer value. Each element of an array is referred by its name with subscript index within the square bracket. For example, num[3] refers to the 4th element in the array.

Some more array declarations with various data types:

char emp_name[25]; // character array named emp_name with size 25

float salary[20]; // floating-point array named salary with size 20

int a[5], b[10], c[15]; // multiple arrays are declared of type int

Memory representation of an one dimensional array

The amount of storage required to hold an array is directly related with type and size. The following figure shows the memory allocation of an array with five elements.
	int numb [5];																							
num [0] num [1] num [2] num [3] num [4]																								
	070	026	027	028	029	030	031	032	033	034	035	036	037	038	039	040	041	042	043	044	045	046	047	048
•	-	H	H	H	H	10	10	10	10	H	10	10	10	10	10	1	10	10	10	H	H	10	10	1(

The above figure clearly shows that, the array num is an integer array with 5 elements. As per the Dev-C++ compiler, 4 bytes are allocated for every int type variable. Here, there are totally 5 elements in the array, where for each element, 4 bytes will be allocated. Totally, 20 bytes will be allocated for this array.

Datatype	Turbo C++	Dev C++
char	1	1
	1	1
int	2	4
float	4	4
long	4	4
double	8	8
long double	10	10

The memory space allocated for an array can be calculated using the following formula:

Number of bytes allocated for type of array × Number of elements

Initialization

An array can be initialized at the time of its declaration. Unless an array is initialized, all the array elements contain garbage values.

Syntax:

```
<datatype> <array_name> [size] = {value-1,value-2,....,value-n};
```

Example

int age[5]={19,21,16,1,50};

In the above example, the array name is 'age' whose size is 5. In this case, the first element 19 is stored in age[0], the second element 21 is stored in age[1] and so on as shown in figure 12.1



Figure 12.1

While declaring and initializing values in an array, the values should be given within the curly braces ie. { }

The size of an array may be optional when the array is initialized during declaration.

Example:

```
int age[]={ 19,21,16,1,50};
```

In the above initialization, the size of the array is not specified directly in the declaration with initialization. So, the size is determined by compiler which depends on the total number of values. In this case, the size of the array is five.

More examples of array initialization:

float x[5] = {5.6, 5.7, 5.8, 5.9, 6.1};

char vowel[6] = {'a', 'e', 'i', 'o', 'u', '0'};

Accepting values to an array during run time :

Multiple assignment statements are required to insert values to the cells of the array during runtime. The for loop is ideally suited for iterating through the array elements.

```
// Input values while execution
#include <iostream>
using namespace std;
int main()
{
    int num[5];
    for(int i=0; i<5; i++)
    {
        cout<< "\n Enter value " << i+1 << "= ";
        cin>>num[i];
    }
}
```

In the above program, a for loop has been constructed to execute the statements within the loop for 5 times. During each iteration of the loop, *cout* statement prompts you to "Enter value" and *cin* gets the value and stores it in num[i];

The following table shows the execution of the above code block.

Iteration	i <5	cout << "\n Enter value " << i+1 << "= ";	cin>>num [i];	Recei value sto mem	ived ored in ory	i++ (i=i+1)
1	5 > 0 (T)	Enter value 1 =	num[0] = 5	num[0]	5	1
2	5 > 1 (T)	Enter value 2 =	num[1] = 10	num[1]	10	2
3	5 > 2 (T)	Enter value 3 =	num[2] = 15	num[2]	15	3
4	5 > 3 (T)	Enter value 4 =	num[3] = 20	num[3]	20	4
5	5 > 4 (T)	Enter value 4 =	num[25 = [4	num[4]	25	5
6	5 > 5 (F)	Exit from Loop				

Note

In for loop, the index *i* is declared with an initial value 0 (zero). Since in most of the cases, the initial value of the loop index will be used as the array subscript representation.

Accessing array elements

Array elements can be used anywhere in a program as we do in case of a normal variable. The elements of an array are accessed with the array name followed by the subscript index within the square bracket.

Example:

```
cout<<num[3];</pre>
```

In the above statement, num[3] refers to the 4th element of the array and *cout* statement displays the value of num[3].

Note

The subscript in bracket can be a variable, a constant or an expression that evaluates to an integer.

```
// Accessing array elements
#include <iostream>
using namespace std;
int main()
{
     int num[5] = {10, 20, 30, 40, 50};
     int t=2;
     cout<<num[2] <<endl; // S1
     cout<<num[3+1] <<endl; // S2
     cout<<num[t=t+1]; // S3
}
output:
30
50
40</pre>
```

In the above program, statement **S1** displays the value of the 3rd element (subscript index 2). **S2** will display the value of the 5th element (ie. Subscript value is 3+1 = 4). In the same way statement **S3** will display the value of the 4th element.

The following program illustrates the writing and reading of array elements.

```
//Program to read and write the values from an array
#include <iostream>
using namespace std;
int main()
{
    int age[4];//declaration of array
    cout<< "Enter the age of four persons:" <<endl;
    for(int i = 0; i < 4; i++)//loop to write array elements
        cin>> age[i];
    cout<<"The ages of four persons are:";
for(int j = 0; j<4; j++)
        cout<< age[j]<<endl;
}</pre>
```

The following table shows the execution of the above code lines

Iteration	i < 4	cin>> age [i];	Value received	age	i++ (i=i+1)
1	4 > 0 (T)	cin>> age[0];	18	age [0] = 18	1
2	4 > 1 (T)	cin>> age[1];	17	age [1] = 17	2
3	4 > 2 (T)	cin>> age[2];	21	age [2] = 21	3
4	4 > 3 (T)	cin>> age[3];	23	age [3] = 23	4
5	4 > 4 (F)		Exit fro	om loop	

After the successful execution of the above statements, the given values will be stored in memory like,



Second for loop:

for (int j = 0; j < 4; j++)

cout<< age[j]<<endl;</pre>

The above statements are used to read the values from the memory and display the values.

The following table shows the execution of the above code.

Iteration	j< 4	cout << age[j];	values read from memory	Output	j++		
1	4 > 0 (T)	cout << age[0];	18	18	1		
2	4 > 1 (T)	cout << age[1];	17	17	2		
3	4 > 2 (T)	cout << age[2];	21	21	3		
4	4 > 3 (T)	cout << age[3];	23	23	4		
5	4 > 4 (F)	Exit from loop					

So, the final output will be:

Enter the age of four persons:

18

17

21

23

The ages of four persons are:

18 17 21

23

Traversal:

Accessing each element of an array at least once to perform any operation is known as "Traversal". Displaying all the elements in an array is an example of "traversal".

```
// Traversal of an array
#include <iostream>
using namespace std;
int main()
{
       int num[5];
       for (int i=0; i<5; i++)
       {
              cout<< "\n Enter value " << i+1 <<"= ";
              cin>>num[i]; // Reading from keyboard
// Traversing the array elements sequentially and adding 1 to each element
              num[i] = num[i] + 1;
       }
       cout<< "\n After incrementing, the values in array num..." << endl;
       for (int j=0; j<5; j++)
// Traversing the array elements sequentially and printing each one of them
       cout<<num[j] <<endl;</pre>
}
Output:
Enter value 1 = 10
Enter value 2 = 20
Enter value 3 = 30
Enter value 4 = 40
Enter value 5 = 50
After incrementing, the values in array num...
11
21
31
41
51
```

Program to read the marks of 10 students and to find the average of all those marks.

```
#include <iostream>
using namespace std;
int main()
```

}

{

Output:

```
Enter Mark 1= 41
Enter Mark 2= 98
Enter Mark 3= 65
Enter Mark 4= 75
Enter Mark 5= 35
Enter Mark 6= 82
Enter Mark 7= 64
Enter Mark 8= 5
Enter Mark 9= 58
Enter Mark 10= 68
The Total Marks: 591
The Average Mark: 59
```

C++ program to inputs 10 values and count the number of odd and even numbers

```
#include <iostream>
using namespace std;
int main()
{
      int num[10], even=0, odd=0;
      for (int i=0; i<10; i++)
       {
             cout<< "\n Enter Number " << i+1 <<"= ";
             cin>>num[i];
             if (num[i] \% 2 == 0)
             ++even;
             else
             ++odd;
       }
       cout << "\n There are "<< even <<" Even Numbers";</pre>
       cout << "\n There are "<< odd <<" Odd Numbers";</pre>
}
Output:
Enter Number 1 = 78
Enter Number 2 = 51
Enter Number 3 = 32
Enter Number 4= 66
Enter Number 5 = 41
Enter Number 6= 68
Enter Number 7= 27
Enter Number 8= 65
Enter Number 9=28
Enter Number 10= 94
There are 6 Even Numbers
There are 4 Odd Numbers
```

(HOTS : Rewrite the above program using the conditional operator instead of if)

```
Program to read the prices of 10 products in an array and then print the sum and average of all the prices
```

```
#include <iostream>
using namespace std;
int main()
{
```

```
float price[10], sum=0, avg=0, prod=1;
for(int i=0; i<10; i++)
{
     cout<< "\n Enter the price of item " << i+1 <<"= ";
     cin>> price[i];
     sum+=price[i];
}
avg=sum/10.0;
cout<< "\n Sum of all prices: " << sum;
cout<< "\n Average of all prices: " <<avg;</pre>
```

}

Program to accept the sales of each day of the month and print the average sales for each month

```
#include <iostream>
using namespace std;
int main()
{
       int days;
       float sales[5], avgSales=0, totalSales=0;
       cout<< "\n Enter No. of days: ";
       cin>> days;
       for (int i=0; i<days; i++)
        {
              cout << "\n Enter sales on day - " << i+1 <<": ";
              cin>> sales[i];
              totalSales+=sales[i];
        }
       avg=total sales/days;
       cout<< "\n Average Sales = " <<avgSales;</pre>
       return 0;
}
```

Searching in a one dimensional array:

Searching is a process of finding a particular value present in a given set of numbers. The linear search or sequential search compares each element of the list with the value that has to be searched until all the elements in the array have been traversed and compared.

```
Program for Linear Search
```

```
#include <iostream>
using namespace std;
int Search(int arr[], int size, int value)
{
    for (int i=0; i<size; i++)
    {
          if (arr[i] == value)
              return i: // return index value
    }
    return -1;
}
int main()
{
       int num[10], val, id;
       for (int i=0; i<10; i++)
       {
       cout<< "\n Enter value " << i+1 <<"= ";
       cin>>num[i];
       }
       cout<< "\n Enter a value to be searched: ";
       cin>>val:
       id=Search(num,10,val);
       if(id=-1)
       cout<< "\n Given value is not found in the array..";
       else
       cout<< "\n The value is found at the position" << id+1;
       return 0;
}
```

The above program reads an array and prompts for the values to be searched. It calls Search() function which receives array, size and value to be searched as parameters. If the value is found, then it returns the array index to the called statement; otherwise, it returns -1.

Strings

A string is defined as a sequence of characters where each character may be a letter, number or a symbol. Each element occupies one byte of memory. Every string is terminated by a null ('\0', ASCII code 0) character which must be appended at the end of the string. In C++, there is no basic data type to represent a string. Instead, it implements a string as an one-dimensional character array. When declaring a character array, it also has to hold a null character at the end, and so, the size of the character array should be one character longer than the length of the string.

Character Array (String) creation

To create any kind of array, the size (length) of the array must be known in advance, so that the memory locations can be allocated according to the size of the array. Once an array is created, its length is fixed and cannot be changed during run time. This is shown in figure 12.2





Syntax

Array declaration is:

char array_name[size];

In the above declaration, the size of the array must be an unsigned integer value.

For example,

char country[6];

Here, the array reserves 6 bytes of memory for storing a sequence of characters. The length of the string cannot be more than 5 characters and one location is reserved for the null character at the end.

Initialization

The character array can be initialized at the time of its declaration. The syntax is shown below:

char array_name[size]={ list of charecters separated by comma or a string } ;

For example,

char country[6]="INDIA";

In the above example, the text "INDIA" has 5 letters which is assigned as initial value to array country. The text is enclosed within double quotes. The memory representation is shown in Figure 13.3



Figure 12.3

In the above memory representation, each character occupies one byte in

memory. At the end of the string, a null character is automatically added by the compiler. C++ also provides other ways of initializing the character array:

char country[6]={'I', 'N', 'D', 'I', 'A', '\0'};

char country[]="INDIA";

char country[]={'I', 'N', 'D', 'I', 'A', '\0'};

If the size of the array is not explicitly mentioned, the compiler automatically calculate the size of the array based on the number of elements in the list and allocates space accordingly.

In the initialization of the string, if all the characters are not initialized, then the rest of the characters will be filled with NULL.

Example:

```
char str[5]={'5','+','A'};

str[0]; ---> 5

str[1]; ---> +

str[2]; ---> A

str[3]; ---> NULL

str[4]; ---> NULL

Note During initialization, the array of elements cannot be initialized more than its size.
```

For example

char str[2]={'5','+','A','B'}; // Invalid

In the above example, the compiler displays "initialize-string for array of chars is too long" error message.

```
Write a program to demonstrate various methods of initializing the
character arrays
#include <iostream>
using namespace std;
int main()
{
       char arr1[6]="INDIA";
       char arr2[6]={'I','N','D','I','A','0'};
       char arr3[]="TRICHY";
       char arr4[]={'T','R','I','C','H','Y','0'};
       char arr5[8]="TRICHY";
       cout<<"arr1 is :" <<arr1<< " and its size is "<<sizeof(arr1)<<endl;
       cout<<"arr2 is :" <<arr2<< " and its size is "<<sizeof(arr2)<<endl;
       cout<<"arr3 is :" <<arr3<< " and its size is "<<sizeof(arr3)<<endl:
       cout<<"arr4 is :" <<arr4<< " and its size is "<<sizeof(arr4)<<endl;
       cout<<"The elements of arr5"<<endl;
       for(int i=0;i<8;i++)
       cout<<arr5[i]<<" ";
       return 0;
}
Output
arr1 is :INDIA and its size is 6
arr2 is :INDIA and its size is 6
arr3 is :TRICHY and its size is 7
arr4 is :TRICHY and its size is 7
The elements of arr5
TRICHY
```

Read a line of Text

In C++, cin.get() is used to read a line of text including blank spaces. This function takes two arguments. The first argument is the name of the string and second argument is the maximum size of the array.

Write a program to display a line of text using get() function.

```
// str10.cpp
// To read a line of text
#include <iostream>
using namespace std;
int main()
{
    char str[100];
    cout<< "Enter a string: ";
    cin.get(str, 100);
    cout<< "You entered: " <<str<<endl;
    return 0;
}
Output
Enter a string: I am a student
You entered: I am a student</pre>
```

In the above program, str is the name of the string and 100 is the maximum size of the character array that represents the string str.

In C++, getline() is also used to read a line of text from the input stream. It can read the characters till it encounters a newline character or a delimiter specified by the user. This function is available in the <string> header.

```
Write a Program to check palindrome or not
#include<iostream>
using namespace std;
int main()
{
    int i, j, len, flag =1;
    char a [20];
    cout<<"Enter a string:";
    cin>>a;
    for(len=0;a[len]!='\0';++len)
```

```
for(!=0,j=len-1;i<len/2;++i,--j)
{
    if(a[j]!=a[i])
        flag=0;
    }

    if(flag==1)
        cout<<"\n The String is palindrome";
    else
        cout<<<"\n The String is not palindrome";
    return 0;
}
Output:
    Enter a string : madam
    The String is palindrome</pre>
```

12.3 Two-dimensional array

Two-dimensional (2D) arrays are collection of similar elements where the elements are stored in certain number of rows and columns. An example $m \times n$ matrix where m denotes the number of rows and n denotes the number of columns is shown in Figure 12.4

int arr[3][3];

2D array conceptual memory representation

	Co	lumn subscript	
script	arr[0] [0]	arr[0] [1]	arr[0] [3]
ow sub	arr[1] [0]	arr[1] [1]	arr[1] [2]
≃↓	arr[2] [0]	arr[2] [1]	arr[2] [2]

The array arr can be coneptually viewed in matrix form with 3 rows and coloumns. point to be noted here is since the subscript starts with 0 arr [0][0] represents the first element.



12.3.1 Declaration of 2-D array

The declaration of a 2-D array is

data-type array_name[row-size][col-size];

In the above declaration, data-type refers to any valid C++ data-type, array_name refers to the name of the 2-D array, row-size refers to the number of rows and col-size refers to the number of columns in the 2-D array.

For example

int A[3][4];

In the above example, A is a 2-D array, 3 denotes the number of rows and 4 denotes the number of columns. This array can hold a maximum of 12 elements.

Note

Array size must be an unsigned integer value which is greater than 0. In arrays, column size is compulsory but row size is optional.

Other examples of 2-D array are:

int A[3][3];

float x[2][3];

```
char name[5][20];
```

12.3.2 Initialization of Two-Dimensional array

The array can be initialized in more than one way at the time of 2-D array declaration.

For example

```
int matrix[4][3]={
{10,20,30},// Initializes row 0
{40,50,60},// Initializes row 1
{70,80,90},// Initializes row 2
{100,110,120}// Initializes row 3
};
int matrix[4][3]={10,20,30,40,50,60,70,80,90,100,110,120};
```

Array's row size is optional but column size is compulsory.

For example

```
int matrix[][3]={
{10,20,30},// row 0
{40,50,60},// row 1
{70,80,90},// row 2
{100,110,120}// row 3
};
```

12.3.3 Accessing the two-dimensional array

Two-dimensional array uses two index values to access a particular element in it, where the first index specifies the row value and second index specifies the column value.

matrix[0][0]=10;// Assign 10 to the first element of the first row

matrix[0][1]=20;// Assign 20 to the second element of the first row

matrix[1][2]=60;// Assign 60 to the third element of the second row

matrix[3][0]=100;// Assign 100 to the first element of the fourth row

```
Write a program to perform addition of two matrices
#include<iostream>
#include<conio>
using namespace std;
int main()
{
        int row, col, m1[10][10], m2[10][10], sum[10][10];
        cout<<"Enter the number of rows : ";</pre>
        cin>>row;
        cout<<"Enter the number of columns : ";</pre>
        cin>>col;
        cout<< "Enter the elements of first matrix: "<<endl;</pre>
        for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
        cin>>m1[i][j];
        cout<< "Enter the elements of second matrix: "<<endl;
        for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
        cin>>m2[i][j];
```

```
cout<<"Output: "<<endl;
for (int i = 0;i<row;i++ )
for (int j = 0;j<col;j++ )
{
    sum[i][j]=m1[i][j]+m2[i][j];
    cout<<sum[i][j]<<" ";
    }
    cout<<endl<<endl;
    }
getch();
return 0;
}
```



12.3.4 Memory representation of 2-D array

Normally, the two-dimensional array can be viewed as a matrix. The conceptual view of a 2-D array is shown below:

int A[4][3];

A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]
A[2][0]	A[2][1]	A[2][2]
A[3][0]	A[3][1]	A[3][2]

In the above example, the 2-D array name A has 4 rows and 3 columns.

Like one-dimensional, the 2-D array elements are stored in continuous memory.

There are two types of 2-D array memory representations. They are:

- Row-Major order
- Column-Major order

For example

int A[4][3]={

{ 8,6,5},
{ 2,1,9},
{3,6,4},
{4,3,2},

Row Major order

In row-major order, all the elements are stored row by row in continuous memory locations, that is, all the elements in first row, then in the second row and so on. The memory representation of row major order is as shown below;



12.4 Array of strings

An array of strings is a two-dimensional character array. The size of the first index (rows) denotes the number of strings and the size of the second index (columns) denotes the maximum length of each string. Usually, array of strings are declared in such a way to accommodate the null character at the end of each string. For example, the 2-D array has the declaration:

char Name[6][10];

In the above declaration, the 2-D array has two indices which refer to the row size and column size, that is 6 refers to the number of rows and 10 refers to the number of columns.

12.4.1 Initialization

For example

char Name[6][10] = {"Mr. Bean", "Mr.Bush", "Nicole", "Kidman", "Arnold", "Jodie"};

In the above example, the 2-D array is initialized with 6 strings, where each string is a maximum of 9 characters long, since the last character is null.

The memory arrangement of a 2-D array is shown below and all the strings are stored in continuous locations.



12.5 Passing Arrays to functions

In C++, arrays can be passed to a function as an argument. To pass an array to a function in C++, the function needs the array name as an argument.

Passing a two-dimensional array to a function

Write a program to display marks of 5 students by passing one-dimensional array to a function.

```
C++ program to display marks of 5 students (one dimensional array)
 #include <iostream>
 using namespace std;
 void display (int m[5]);
 int main()
 {
        int marks[5]={88, 76, 90, 61, 69};
        display(marks);
        return 0;
 }
  void display (int m[5])
 {
        cout << "\n Display Marks: " << endl;</pre>
               for (int i=0; i<5; i++)
                     cout << "Student " << i+1 << ": " << m[i]<<endl;
               }
 }
 Output:
 Display Marks:
 Student 1:88
 Student 2:76
 Student 3:90
 Student 4:61
 Student 5: 69
```

12.6 Passing 2"D array to a function

```
C++ program to display values from two dimensional array
 #include <iostream>
 using namespace std;
 void display (int n[3][2]);
 int main()
 {
                int num[3][2] = \{ \{3, 4\}, \{9, 5\}, \{7, 1\} \};
                display(num);
                return 0;
 }
void display(int n[3][2])
 {
        cout << "\n Displaying Values" << endl;</pre>
                for (int i=0; i<3; i++)
                {
                       for (int j=0; j<2; j++)
                       {
                              cout << n[i][j] << " ";
                       ļ
               cout << endl << endl;</pre>
                }
 }
 Output:
 Displaying Values
 34
 95
 71
```

In the above program, the two-dimensional array num is passed to the function display() to produce the results.

```
// Function with character array as argument
#include <iostream>
using namespace std;
int main()
{
       char str[100];
void display(char s[]);
       cout<< "Enter a string: ";</pre>
       getline(cin, str);
display(str);
       return 0;
}
void display(char s[])
{
cout<< "You entered char array: " << s <<endl;
}
output
Enter a string: welcome to C++ programming
You entered char array: welcome to C++ programming
```

Case Study:

- (1) Write a program to accept the marks of 10 students and find the average, maximum and minimum marks.
- (2) Write a program to accept rainfall recorded in four metropolitan cities of India and find the city that has the highest and lowest rainfall.
- (3) Survey your neighboring shops and find the price of any particular product of interest and suggest where to buy the product at the lowest cost.

PART – I

Choose the correct answer

1. Which of the following is the collection of variables of the same type that an referenced by a common name ? a) int b) float c) Array d) class Array subscripts is always starts with which number ? 2. b) 0 c) 2 d) 3 a)-1 3. int age[]={6,90,20,18,2}; How many elements are there in this array? a) 2 b) 5 c) 6 d) 4 4. cin>>n[3]; To which element does this statement accepts the value? a) 2 b) 3 c) 4 d) 5 5. By default, the string and with which character? b) \t d) \b a)\o c) \n Part – II Answer to all the questions (2 Marks): 1. What is Traversal in an Array? 2. What is Strings? What is the syntax to declare two – dimensional array. 3.

Part – III

Answer to all the questions(3 Marks):

1. Define an Array ? What are the types?

2. With note an Array of strings.

3. Write a C++ program to accept and print your name?

Part – IV

Answer to all the questions (5 Marks):

- 1. Write a C++ program to find the difference between two matrix.
- 2. How will you pass two dimensional array to a function explain with example.

Structures

Learning Objectives

After the completion of this chapter, the student will be able to

- Understand the purpose of user defined data types
- Able to construct C++ programs using structures
- Execute and debug programs with structure data type

12.7 Structures Introduction

Structure is a user-defined which has the combination of data items with different data types. This allows to group of variables of mixed data types together into a single unit.

12.7.1 Purpose of Structures

In any situation when more than one variable is required to represent objects of uniform data-types, array can be used. If the elements are of different data types, then array cannot support. If more than one variable is used, they can be stored in memory but not in adjacent locations. It increases the time consumption while searching. The structure provides a facility to store different data types as a part of the same logical element in one memory chunk adjacent to each other.

12.7.2 Declaring and defining structures

Structure is declared using the keyword 'struct'. The syntax of creating a structure is given below.

struct structure_name {

type member_name1;

Objects declared along with structure definition are called global objects

type member_name2;

} reference_name;

An optional field reference_name can be used to declare objects of the structure type directly.

Example:

struct Student

{

long rollno;

int age;

float weight;

};

In the above declaration of the struct, three variables rollno,age and weight are used. These variables(data element)within the structure are called members (or fields). In order to use the Student structure, a variable of type Student is declared and the memory allocation is shown in figure 12.5

Rollno	Age	weight
≮ 4 Bytes >∢	2 Bytes >∢ -	4 Bytes>

Fig 12.5 Memory Allocation

struct Student balu; // create a Student structure for Balu

This defines a variable of type Student named as Balu. Similar to normal variables, struct variable allocates memory for that variable itself. It is possible to define multiple variables of the same struct type:

struct Student frank; // create a structure for Student Frank.

For example, the structure objects balu and frank can also be declared as the structure data type as:

struct Student

{

longrollno;

int age;

float weight;

}balu, frank;

12.7.3 Referencing Structure Elements

Once the two objects of student structure type are declared (balu and frank), their members can be accessed directly. The syntax for that is using a dot (.) between the object name and the member name. For example, the elements of the structure Student can be accessed as follows:

balu.rollno

balu.age

balu.weight

frank.rollno

frank.age

frank.weight

If the members are a pointer types then ' \rightarrow ' is used to access the members. Let name is a character pointer ins student like char * name It can be accessed student \rightarrow name

(Anonymous Structure Vs Named Structure)
A structure without a name/tag is called anonymous structure.
struct
{
long rollno;
 int age;
 float weight;
} student;
The student can be referred as reference name to the above structure and the
elements can be accessed like student.rollno, student.age and student.weight.

12.7.4 Initializing structure elements

Values can be assigned to structure elements similar to assigning values to variables.

Example

balu.rollno= "702016";

balu.age= 18;

balu.weight= 48.5;

Also, values can be assigned directly as similar to assigning values to Arrays.

balu={702016, 18, 48.5};

12.7.5 Structure Assignments

Structures can be assigned directly instead of assigning the values of elements individually.

Example

If Mahesh and Praveen are same age and same height and weight then the values of Mahesh can be copied to Praveen

struct Student

{

int age;

float height, weight;

}mahesh;

The age of Mahesh is 17 and the height and weights are 164.5 and 52.5 respectively. The following statement will perform the assignment.

mahesh = {17, 164.5, 52.5};

praveen =mahesh;

will assign the same age, height and weight to Praveen.

Examples:

The following C++ program reads student information through keyboard and displays the same

#include <iostream>
using namespace std;
struct Student
{
 int age;
 float height, weight;

Structure assignment is possible only if both structure variables/ objects are same type.

```
} mahesh;
void main( )
```

```
{
```

```
cout<< " Enter the age:"<<endl;</pre>
      cin>>mahesh.age;
      cout<< "Enter the height:"<<endl;</pre>
      cin>>mahesh.height;
      cout<< "Enter the weight:"<<endl;</pre>
      cin>>mahesh.weight;
      cout<< "The values entered for Age, height and weight are"<<endl;
                               "\t"<<mahesh.height<< "\t"<<Mahesh.
      cout<<mahesh.age<<
weight;
```

Output:

}

```
Enter the age:
18
Enter the height:
160.5
Enter the weight:
46.5
The values entered for Age, height and weight are
18
       160.5
                    46.5
```

The following C++ Program assigns data to members of a structure variable and displays the contents

```
include<iostream>
using namespace std;
struct Employee
{
char name[50];
int age;
      float salary;
};
```

```
int main()
```

{

}

Employee e1; cout<< "Enter Full name: "; cin>>e1.name: cout<<endl<<"Enter age: ";</pre> cin>>e1.age; cout<<endl<< "Enter salary: ";</pre> cin>>e1.salary; cout<< "\nDisplaying Information." << endl;</pre> cout<< "Name: " <<e1.name <<endl;</pre> cout<<"Age: " <<e1.age <<endl;</pre> cout<< "Salary: " <<e1.salary;</pre> return 0; **Output:** Enter Full name: Ezhil Enter age: 27 Enter salary: 40000.00 Displaying Information. Name: Ezhil Age: 27

Salary: 40000.00

12.7.6 Nested Structures

The structure declared within another structure is called a nested structure. A structure 'Student'was used to hold the student's information in the earlier examples. Date of birth can be included in the student's information. There are three components in the date of birth namely, date, month and year like 25-NOV-2017. Hence, another structure is used to keep the date of birth of a student. The following code creates a structure for the date of birth.

```
struct dob
```

```
{
```

int date;

```
char month[3];
```

```
int year;
```

};

```
Values can be assigned to this structure as follows.
```

```
dob= {25,"NOV",2017}
```

```
The date of birth can be assigned as one of the elements in the student structure
 struct Student
 {
        int age;
        float height, weight;
        struct dob
        {
                                              nested structures act as members
         int date:
                                              of another structure and the
                                              members of the child structure
         char month[4];
                                              can be accessed as parent
         int year;
                                              structure name. Child structure
        };
                                              name. Member name.
 }mahesh;
 void main( )
 {
        cout<< " Enter the age:"<<endl;</pre>
        cin>>mahesh.age;
        cout<< "Enter the height:"<<endl;</pre>
        cin>>mahesh.height;
        cout<< "Enter the weight:"<<endl;</pre>
        cin>>mahesh.weight;
        cout<< "The Date of birth:"<<endl;</pre>
        cout<< " Enter the day:"<<endl;
        cin>>mahesh.dob.date;
        cout<< "Enter the month:"<<endl;</pre>
```

cin>>mahesh.dob.month; cout<< "Enter the year:"<<endl; cin>>mahesh.dob.year; cout<< "The values entered for Age, height and weightare"<<endl; cout<<mahesh.age<< "\t"<<mahesh.height<< "\t"<<mahesh.weight<<endl; cout<< "His date of Birth is:"<<endl<<mahesh.dob.date<< "-"<<mahesh. dob.month<< "-" <<mahesh.dob.year;</pre>

}

Output:

Enter the age: 18 Enter the height: 160.5 Enter the weight: 46.5 The Date of birth Enter the day: 25 Enter the month: NOV Enter the year: 2017 The values entered for Age, height and weight are 18 160.5 46.5 His date of Birth is: 25-NOV-2017

12.7.7 Array of Structures

A class may contain many students. So, the definition of structure for one student can also be extended to all the students. If the class has 20 students, then 20 individual structures are required. For this purpose, an array of structures can be used. An array of structures is declared in the same way as declaring an array with built-in data types like int or char.

The following program reads the details of 20 students and prints the same.

```
#include <iostream>
using namespace std;
struct Student
{
       int age;
       float height, weight;
       char name[30];
};
void main( )
{
Student std[20];
int i;
       cout<< " Enter the details for 20 students"<<endl:
       for(i=0;i<20;i++)
{
       cout<< " Enter the details of student"<<i+1<<endl;
       cout<< " Enter the age:"<<endl;</pre>
       cin>>std[i].age;
       cout<< "Enter the height:"<<endl;</pre>
       cin>>std[i].height;
       cout<< "Enter the weight:"<<endl;
       cin>>std[i].weight;
}
       cout<< "The values entered for Age, height and weight are"<<endl;
       for(i=0;i<20;i++)
cout<<"Student "<<i+1<< "\t"<<std[i].age<< "\t"<<std[i].height<< "\t"<<std[i].weight;
}
Output:
Enter the details of 20 students
Enter the details for student1
Enter the age:
18
Enter the height:
160.5
Enter the weight:
46.5
Enter the details for student2
Enter the age:
18
```

Enter the height:								
164.5								
Enter the weight:								
61.5	61.5							
•••••								
•••••								
The values en	ntered	for Age,	height and weight	ght are				
Student 1	18	160.5	46.5					
Student 2	18	164.5	61.5					

The above program reads age , height and weight of 20 students and prints the same details. The output is shown for only two students due to space constraints.

Let an organization has three employees. If we want to read and print all their details thenan array of structures is desirable for employees of this organization. This canbe done through declaring an array of employee structures.

```
include<iostream>
using namespace std;
struct Employee
{
char name[50];
int age;
float salary;
};
int main()
Employee e1[3];
int i;
cout<< "Enter the details of 3 employees"<<endl;
for(i=0;i<3;i++)
       cout<< "Enter the details of Employee"<< i+1<<endl;
       cout<< "Enter name: ";</pre>
       cin>>e1[i].name;
```
```
cout<<endl<<"Enter age: ";</pre>
       cin>>e1[i].age;
       cout<<endl<< "Enter salary: ";</pre>
       cin>>e[i]1.salary;
}
       cout<< "Displaying Information" << endl;</pre>
       for(i=0;i<3;i++)
{
       cout<< "The details of Employee" <<i+1<<endl;
       cout<< "Name: " <<e1[i].name <<endl;</pre>
       cout<<"Age: " <<e1[i].age <<endl;</pre>
       cout<< "Salary: " <<e1[i].salary;</pre>
return 0;
}
Output:
Enter the details of 3 employees
Enter the details of Employee 1
Enter name:
Lilly
Enter age:
42
Enter salary:
40000.00
Enter the details of Employee 2
Enter name:
Aster
Enter age:
38
Enter salary:
60000.00
Enter the details of Employee 3
Enter name:
Jasmine
Enter age:
45
```

Enter salary: 80000.00 Displaying Information. The details of Employee 1 Name:Lilly Age: 42 Salary: 40000.00 The details of Employee 2 Name:Aster Age:38 Salary:60000.00 The details of Employee 3 Name:Jasmine Age:45 Salary:80000.00

12.7.8 Passing structures to functions

A structure variable can be passed to a function in a similar way of passing any argument that is of built-in data type.

If the structure itself is an argument, then it is called "call by value". If the reference of the structure is passed as an argument then it is called, "call by reference".

1 Call by value.

When a structure is passed as argument to a function using call by value method, any change made to the contents of the structure variable inside the function to which it is passed do not affect the structure variable used as an argument.

```
Consider this example:
```

```
#include <iostream>
using namespace std;
struct Employee
{
    char name[50];
int age;
    float salary;
};
```

```
void printData(Employee); // Function declaration
int main()
{
       Employee p;
       cout<< "Enter Full name: ";</pre>
       cin>>p.name;
       cout<< "Enter age: ";</pre>
       cin>>p.age;
       cout<< "Enter salary: ";</pre>
       cin>>p.salary;
       // Function call with structure variable as an argument
printData(p);
return 0;
}
void printData(Employee q)
{
       cout<< "\nDisplaying Information." << endl;</pre>
       cout<< "Name: " << q.name <<endl;</pre>
       cout<<"Age: " <<q.age<<endl;</pre>
       cout<< "Salary: " <<q.salary;</pre>
}
Output:
Enter Full name: Kumar
Enter age: 55
Enter salary: 34233.4
Displaying Information.
Name: Kumar
Age: 55
Salary: 34233.4
```

In the above example, a structure named Employee is declared and used. The values that are entered into the structure are name, age and salary of a Employee are displayed using a function named printData(). The argument for the above function is the structure Employee. The input can be received through a function named readData().

```
#include <iostream>
using namespace std;
struct Employee {
char name[50];
int age;
float salary;
};
Employee readData(Employee);
void printData(Employee);
int main()
{
       Employee p;
       p = readData(p);
       printData(p);
return 0;
}
       Employee readData(Employee p) {
       cout<< "Enter Full name: ";
       cin.get(p.name, 50);
       cout<< "Enter age: ";</pre>
       cin>>p.age;
       cout<< "Enter salary: ";</pre>
       cin>>p.salary;
return p;
void printData(Employee p)
{
       cout<< "\nDisplaying Information." << endl;</pre>
       cout<< "Name: " << p.name <<endl;</pre>
       cout<<"Age: " <<p.age<<endl;</pre>
       cout<< "Salary: " << p.salary;</pre>
}
```

The output of the program is the same as that of the previous programme.

2 Call by reference

In this method of passing the structures to functions ,the address of a structure variable /object is passed to the function using address of(&) operator. So any change made to the contents of structure variable inside the function are reflected back to the calling function.

```
#include <iostream>
using namespace std;
struct Employee {
char name[50];
int age;
float salary;
};
void readData(Employee &);
                                     Structures are usually passed by
void printData(Employee);
                                     reference method because it saves the
int main()
                                     memory space and executes faster.
{
       Employee p;
       readData(p);
       printData(p);
return 0;
}
void readData(Employee &p) {
       cout<< "Enter Full name: ";
       cin.get(p.name, 50);
       cout<< "Enter age: ";</pre>
       cin>>p.age;
       cout<< "Enter salary: ";</pre>
       cin>>p.salary;
}
void printData(Employee p)
{
       cout<< "\nDisplaying Information." << endl;</pre>
       cout<< "Name: " << p.name <<endl;</pre>
       cout<<"Age: " <<p.age<<endl;</pre>
       cout<< "Salary: " << p.salary;</pre>
}
Output:
Enter Full name: Kumar
Enter age: 55
Enter salary: 34233.4
Displaying Information.
Name: Kumar
Age: 55
Salary: 34233.4
```

3 Returning Structures from Functions

A structure can be passed to a function through its object. Therefore, passing a structure to a function or passing a structure object to a function is the same because structure object represents the structure. Like a normal variable, structure variable(structure object) can be passed by value or by references / addresses.

Similar to built-in data types, structures also can be returned from a function.

```
#include<iostream.h>
using namespace std;
struct Employee
{
int Id;
char Name[25];
int Age;
long Salary;
};
       Employee Input();
void main()
{
       Employee e;
       Emp = Input();
       cout<< "The values Entered are"<<endl:
       cout<< "\n\nEmployee Id : " <<e.Id;</pre>
       cout<< "\nEmployee Name : " << e.Name;</pre>
       cout<< "\nEmployee Age : " <<e.Age;
       cout<< "\nEmployee Salary : " <<e.Salary;</pre>
}
       Employee Input()
{
       Employee e;
       cout<< "\nEnter Employee Id : ";</pre>
       cin>>e.Id;
       cout<< "\nEnter Employee Name : ";</pre>
       cin>>e.Name;
       cout<< "\nEnter Employee Age : ";</pre>
       cin>>e.Age;
       cout<< "\nEnter Employee Salary : ";</pre>
       cin>>e.Salary;
return;
}
```

Output :

Enter Employee Id : 10 Enter Employee Name : Ajay Enter Employee Age : 25 Enter Employee Salary : 15000 Employee Id : 10 Employee Name : Ajay Employee Age : 25 Employee Salary : 15000

Points to Remember:

- Structure is a user-defined which has the combination of data items with different data types
- Structure is declared using the keyword 'struct'
- Structure elements are referenced using its object name followed by dot(.) operator and then the member name
- A structure without a name/tag is called anonymous structure.
- The structure elements can be initialized either by using separate assignment statements or at the time of declaration by surrounding its values with braces.

- A structure object can also be assigned to another structure object only if both the objects are of same structure type.
- The structure declared within another structure is called a nested structure
- A structure can contain array as its member element.
- Array of structure variable can also be created.
- Structures can be passed to a function either by call by value method or by call by reference method .Functions can also return structures or its references



Part – I

(b) it will allocate the memory

(d) it will be only declared

Choose the correct answer

- 1. The data elements in the structure are also known as
 - (a) objects (b) members (c) data (d) records
- 2. Structure definition is terminated by
 - (a): (b) $\}$ (c); (d)::
- 3. What will happen when the structure is declared?
 - (a) it will not allocate any memory
 - (c) it will be declared and initialized
- 4. What is the output of this program?

#include <iostream>

#include <string.h>

using namespace std;

int main()

{

```
struct student
{
  int n;
  char name[10];
};
student s;
s.n = 123;
```

```
strcpy(s.name, "Balu");
      cout<<s.n;
      cout<< s.name <<endl;</pre>
return 0; }
(a) 123Balu (b)BaluBalu (c) Balu123 (d) 123 Balu
A structure declaration is given below.
struct Time
{
int hours;
int minutes;
int seconds;
}t;
Using above declaration which of the following refers to seconds.
(a) Time.seconds
                    (b) Time::seconds
                                               (c)seconds
                                                             (d) t. seconds
What will be the output of this program?
#include <iostream>
using namespace std;
struct ShoeType
{
string name;
double price;
};
```

```
int main()
```

5.

6.

```
{
ShoeType shoe1, shoe2;
shoe1.name = "Adidas";
shoe1.price = 9.99;
cout<< shoe1.name<< " # "<< shoe1.price<<endl;
shoe2 = shoe1;
shoe2.price = shoe2.price / 9;
cout<< shoe2.name<< " # "<< shoe2.price;</pre>
```

return 0;

(a) Adidas # 9.99	(b) Adidas # 9.99	(c) Adidas # 9.99	(d) Adidas # 9.11
Adidas # 1.11	Adidas # 9.11	Adidas # 11.11	Adidas # 11.11

- 7. Which of the following is a properly defined structure?
 - (a) struct {int num;} (b) struct sum {int num;}
 - (c) struct sum int sum; (d)struct sum {int num;};
- 8. A structure declaration is given below.

struct employee

{

int empno;

char ename[10];

}e[5];

Using above declaration which of the following statement is correct.

(a) cout<<e[0].empno<<e[0].ename; (b) cout<<e[0].empno<<ename;

(c)cout<<e[0]->empno<<e[0]->ename; (d) cout<<e.empno<<e.ename;

- 9. Which of the following cannot be a structure member?
 - (a) Another structure (b) Function
 - (c) Array (d) variable of double datatype
- 10. When accessing a structure member ,the identifier to the left of the dot operator is the name of
 - (a) structure variable (b) structure tag
 - (c) structure member (d) structure function

Part – II

Answer to all the questions (2 Marks):

- 1. Define structure .What is its use?
- 2. To store 100 integer number which of the following is good to use?

Array or Structure

State the reason.

3. What is the error in the following structure definition.

struct employee{ inteno;charename[20];char dept;}

Employee e1,e2;

- 4. Write a structure definition for the structure student containing examno, name and an array for storing five subject marks.
- 5. Why for passing a structure to a function call by reference is advisable to us?
- 6. What is the size of the following highlighted variable in terms of byte if it is compiled in dev c++

struct A{ float f[3]; char ch[5]; long double d;};

struct B{ A a; int arr[2][3];}b[3]

7. Is the following snippet is fully correct. If not identify the error.

struct sum1{ int n1,n2;}s1;

struct sum2{int n1,n2}s2;

cin>>s1.n1>>s1.n2;

s2=s1;

- 8. Differentiate array and structure.
- 9. What are the different ways to initialize the structure members?
- 10. What is wrong with the following C++ declarations?
 - A. struct point (double x, y)

- B. struct point { double x, double y };
- C. struct point { double x; double y }
- D. struct point { double x; double y; };
- E. struct point { double x; double y; }

```
PART – III
```

Answer to all the questions (3 Marks):

3.

- 1. How will you pass a structure to a function ?
- 2. The following code sums up the total of all students name starting with 'S' and display it.Fill in the blanks with required statements.

```
struct student {int exam no,lang,eng,phy,che,mat,csc,total;char name[15];};
int main()
{
student s[20];
for(int i=0;i<20;i++)
{
..... //accept student details
}
for(int i=0;i<20;i++)
{
..... //check for name starts with letter "S"
..... // display the detail of the checked name
}
return 0;
}
What is called nested structure. Give example
```

4. Rewrite the following program after removing the syntactical error(s), if any.

```
Underline each correction.
```

struct movie

5.

```
{
charm_name[10];
charm_lang[10];
float ticket cost =50;};
Movie;
void main()
{
gets(m_name);
cin>>m_lang;
return 0;
}
What is the difference among the following two programs?
```

```
(a) #include <iostream.h>
struct point { double x; double y; };
int main() {
struct point test;
test.x = .25; test.y = .75;
cout<<test.x<<test.y;
return 0;
}</pre>
```

```
(b) #include <iostream.h>
```

```
struct { double x; double y; } Point;
int main(void) {
Point test={.25,.75};
return 0;
}
```

- 6. How to access members of a structure?Give example.
- 7. Write the syntax and an example for structure.
- 8. For the following structure definition write the user defined function to

accept data through keyboard.

struct date{ int dd,mm,yy};

struct item { int item id;char name[10];float price;date date_manif;}

- 9. What is called anonymous structure .Give an example
- 10. Write a user defined function to return the structure after accepting value through keyboard. The structure definition is as follows

struct Item{int item no;float price;};

```
Part – IV
```

Answer to all the questions (5 Marks):

- 1. Explain array of structures with example
- 2. Explain call by value with respect to structure.
- 3. How call by reference is used to pass structure to a function .Give an Example
- Write a C++ program to add two distances using the following structure definition struct Distance{

int feet;

float inch;

}d1 , d2, sum;

5. Write a C++ Program to Add two Complex Numbers by Passing Structure to a Function for the following structure definition

struct complex
{
float real;
float imag;
};
The prototype of the function is

complex add Complex Numbers(complex, complex);

- 6. Write a C++ Program to declare a structure book containing name and author as character array of 20 elements each and price as integer. Declare an array of book .Accept the name ,author,price detail for each book.Define a user defined function to display the book details and calculate the total price. Return total price to the calling function.
- 7. Write a c++ program to declare and accept an array of professors.Display the details of the department="COMP.SCI" and the name of the professors start with 'A'. The structure "college" should contain the following members .

prof_id as integer

name and Department as character array

8. Write the output of the following c++ program

#include<iostream>

#include<stdio>

#include <string>

#include<conio>

using namespace std;

struct books {

char name[20], author[20];

} a[50];

```
int main()
```

```
{
```

clrscr();

```
cout<< "Details of Book No " << 1 << "\n";
```

```
cout<< "-----\n";
```

```
cout<< "Book Name :"<<strcpy(a[0].name,"Programming ")<<endl;</pre>
```

```
cout<< "Book Author :"<<strcpy(a[0].author,"Dromy")<<endl;</pre>
```

```
cout<< "\nDetails of Book No " << 2 << "\n";
```

```
cout<< "-----\n";
```

```
cout<< "Book Name :"<<strcpy(a[1].name,"C++programming" )<<endl;</pre>
```

```
cout<< "Book Author :"<<strcpy(a[1].author,"BjarneStroustrup ")<<endl;</pre>
```

```
cout<<"\n\n";
```

```
cout<< " S.No\t| Book Name\t|author\n";</pre>
```

```
for (int i = 0; i < 2; i++) {
```

```
cout<< "\n " << i + 1 << "\t<br/>|" << a[i].name << "\t<br/>| " << a[i].author;
```

}

```
return 0;
```

```
}
```

9. Write the output of the following c++ program

#include <iostream>

```
#include <string>
```

using namespace std;

```
struct student
```

```
{
```

};

```
introll_no;
       char name[10];
       long phone_number;
int main(){
student p1 = {1,"Brown",123443};
student p2, p3;
p2.roll_no = 2;
strcpy(p2.name,"Sam");
p2.phone_number = 1234567822;
p3.roll_no = 3;
strcpy(p3.name,"Addy");
p3.phone_number = 1234567844;
cout<< "First Student" <<endl;</pre>
cout<< "roll no : " << p1.roll_no <<endl;</pre>
cout<< "name : " << p1.name <<endl;</pre>
cout<< "phone no : " << p1.phone_number <<endl;</pre>
cout<< "Second Student" <<endl;</pre>
cout<< "roll no : " << p2.roll_no <<endl;</pre>
cout<< "name : " << p2.name <<endl;</pre>
cout<< "phone no : " << p2.phone_number <<endl;</pre>
```

```
cout<< "Third Student" <<endl;
cout<< "roll no : " << p3.roll_no <<endl;
cout<< "name : " << p3.name <<endl;
cout<< "phone no : " << p3.phone_number <<endl;
return 0;
```

```
}
```

10. Debug the error in the following program

#include <istream.h>

structPersonRec

{

charlastName[10];

chaefirstName[10];

int age;

}

```
PersonRecPeopleArrayType[10];
voidLoadArray(PeopleRecpeop);
void main()
{
    PersonRecord people;
for (i = 0; i < 10; i++)
{
    cout<<people.firstName<<``<<people.lastName
    <<setw(10) <<people.age;
}
</pre>
```

```
LoadArray(PersonRecpeop)
{
for (int i = 0; i < 10; i++)
{
cout<< "Enter first name: ";
cin<<peop[i].firstName;
cout<< "Enter last name: ";
cin>>peop[i].lastName;
cout<< "Enter age: ";
cin>> people[i].age;}
```

References:

- 1. Object Oriented Programming with C++ (4th Edition), Dr. E. Balagurusamy, Mc.Graw Hills.
- 2. The Complete Reference C++ (Forth Edition), Herbert Schildt.Mc.Graw Hills.
- 3. Computer Science with C++ (A text book of CBSE XI and XII), SumitaArora, DhanpatRai&Co.
- 4. The C++ Programming Language, Bjarne Stroustrup
- 5. https://www.tutorialspoint.com
- 6. http://www.cs.princeton.edu
- 7. https://www.programiz.com

Unit IV

Object Oriented Programming with C++

FBZ9XT

Introducton to Object Oriented Programming Techniques

Learing Objectives

After learning this chapter, the students will be able to

- Understand the concept of **OOPS**
- Know the difference between *Procedural, Modular* and *Object Oriented Programming.*
- Understand the advantages and disadvantages of Object Oriented Programming.

13.1 Introduction

Object-Oriented Programming (OOP) is the term used to describe a programming approach based on classes and objects. The object-oriented paradigm allows us to organize software as a collection of objects that consist of both data and behaviour. This is in contrast to conventional functional programming practice, that loosely connects data and behaviour.

Since 1980's the word **'object'** has appeared in relation to programming languages, with almost all languages developed since 1990 having object-oriented features. This chapter introduces general OOP concepts.

13.2 Programming Paradigms

CHAPTER 3

Paradigm means organizing principle of a program. It is an approach to programming. There are different approaches available for problem solving using computer. They are Procedural programming, Modular Programming and Object Oriented Programming

13.2.1 Procedural programming

Procedural means a list of instructions were given to the computer to do something. Procedural programming aims more at procedures. This emphasis on doing things.

Important features of procedural programming

- Programs are organized in the form of subroutines or sub programs
- All data items are global
- Suitable for small sized software application
- Difficult to maintain and enhance the program code as any change in data type needs to be propagated to all subroutines that use the same data type. This is time consuming.
- Example: FORTRAN and COBOL.

13.2.2 Modular programming:-

Modular programming consist of a list of instructions that instructs the computer to do something. But this **Paradigm consists of multiple modules, each module has a set of functions of related types. Data is hidden under the modules.** Arrangement of data can be changed only by modifying the module

Important features of Modular programming

- Emphasis on algorithm rather than data
- Programs are divided into individual modules
- Each modules are independent of each other and have their own local data
- Modules can work with its own data as well as with the data passed to it.
- Example: **Pascal** and **C**

13.2.3 Object Oriented Programming:-

Object Oriented Programming paradigm emphasizes on the data rather than the algorithm. It implements programs using **classes** and **objects**.

Class: A Class is a construct in C++ which is used to bind data and its associated function together into a single unit using the encapsulation concept. Class is a user defined data type. Class represents a group of similar objects.

It can also be defined as a template or blueprint representing a group objects that share common properties and relationship. **Objects:** Represents data and its associated function together into a single unit. Objects are the basic unit of OOP. Basically an object is created from a class. They are instances of class also called as class variables

An identifiable entity with some characteristics and behaviour is called object.

Important features of Object oriented programming

- Emphasizes on data rather than algorithm
- Data abstraction is introduced in addition to procedural abstraction
- Data and its associated operations are grouped in to single unit
- Programs are designed around the data being operated
- Relationships can be created between similar, yet distinct data types
- Example: C++, Java, VB.Net, Python etc.

13.3 Basic Concepts of OOP

The Object Oriented Programing has been developed to overcome the drawbacks of procedural and modular programming. It is widely accepted that object-oriented programming is the most important and powerful way of creating software.

The Object-Oriented Programming approach mainly encourages:

• **Modularisation:** where the program can be decomposed into **modules**.

• **Software re-use:** where a program can be composed from existing and new modules.

Main Features of Object Oriented Programming

- Data Abstraction
- Encapsulation
- Modularity
- Inheritance
- Polymorphism

13.3.1 Encapsulation

The mechanism by which the data and functions are bound together into a single unit is known as **Encapsulation**. It implements abstraction.

Encapsulation is about binding the data variables and functions together in class. It can also be called **data binding**.

Encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object's data and the program. **This encapsulation of data from direct access by the program is called data hiding or information hiding.**

13.3.2 Data Abstraction

Abstraction refers to showing only the essential features without revealing background details. Classes use the concept of abstraction to define a list of abstract attributes and function which operate on these attributes. They encapsulate all the essential properties of the object that are to be created. The attributes are called **data members** because they hold information. The functions that operate on these data are called **methods or member function**.

13.3.3 Modularity

Modularity is designing a system that is divided into a set of functional units (named modules) that can be composed into a larger application.

13.3.4 Inheritance

Inheritance is the technique of building new classes (derived class) from an existing Class (base class). The most important advantage of inheritance is code reusability.

13.3.5 Polymorphism

Polymorphism is the ability of a message or function to be displayed in more than one form.

13.4 Advantages of OOP

Re-usability:

"Write once and use it multiple times" you can achieve this by using class.

Redundancy:

Inheritance is the good feature for data redundancy. If you need a same functionality in multiple class you can write a common class for the same functionality and inherit that class to sub class. Easy Maintenance:

It is easy to maintain and modify existing code as new objects can be created with small differences to existing ones.

Security:

Using data hiding and abstraction only necessary data will be provided thus maintains the security of data.

13.5 Disadvantages of OOP

Size:

Object Oriented Programs are much larger than other programs.

Effort:

Object Oriented Programs require a lot of work to create.

Speed:

Object Oriented Programs are slower than other programs, because of their size.

Points to Remember:

- Paradigm means organizing principle of a program.It is an approach to programming.
- Procedural or Modular programming means a list of instructions were given and each instructions tell the computer to do something.
- Procedural programming aims more ate procedures. In this Programs are organized in the form of subroutines or sub programs
- Modular programming combines related procedures in a module and hides data under modules.
- Object Oriented programming Paradigm emphasizes on the data rather than the algorithm. It implements programs using classes and objects
- Class is a user defined data type. Class represents a group of similar objects.

- Objects are the basic unit of OOP.It represents data and associated function together in to a single unit.
- The mechanism by which the data and functions are bound together into a single unit is known as ENCAPSULATION. It implements abstraction .
- Abstraction refers to showing only the essential features without revealing background details
- Modularity is designing a system that is divided into a set of functional units that can be composed into a larger application.
- Polymorphism is the ability of a message or function to be displayed in more than one form.
- Inheritance is the technique of building new classes (derived class) from an existing class. The most important advantage of inheritance is code reusability.Inheritance is transitive in nature.



PART I



Choose the correct answer:

1.	The term is used to de	escribe a programm	ing approach based	l on classes and objects is
	(A) OOP	(B) POP	(C) ADT	(D) SOP
2.	The paradigm which (A) Object Oriented I (C) Modular program	aims more at procec Programming nming	lures. (B)Procec (D)Struct	lural programming ural programming
3.	Which of the followin (A) class	ng is a user defined c (B) float	lata type? (C) int	(D) object
4.	The identifiable entity	with some characte	eristics and behavio	our is.
	(A) class	(B) object	(C) structure	(D) member
5.	The mechanism by w is known as (A) Inheritance (C) Polymorphism	hich the data and fu (B) Encapsu (D) Abstrac	nctions are bound lation tion	together into a single unit
6.	Insulation of the data (A) Data hiding (C) Polymorphism	from direct access b (B) Encapsu (D) Abstrac	by the program is c lation tion	alled as
7.	Which of the following that are to be created (A) class (C) Polymorphism	ng concept encapsula ? (B) Encapsu (D) Abstrac	ate all the essential lation tion	properties of the object
8.	Which of the followin (A) data hiding (C) code modification	ng is the most impor (B) code reu n (D) accessib	tant advantage of i Isability ility	nheritance?
9.	"Write once and use i (A) redundancy (C) modification	t multiple time" can (B) reusabili (D) compos	be achieved by ity ition	
10.	Which of the followin (A) Inheritance (C) Polymorphism	ng supports the trans (B) Encapsu (D) Abstrac	sitive nature of data lation tion	a?

PART II

Answer to all the questions (2 Marks):

- 1. How is modular programming different from procedural programming paradigm?
- 2. Differentiate classes and objects.
- 3. What is polymorphism?
- 4. How is encapsulation and abstraction are interrelated?
- 5. Write the disadvantages of OOP.

PART III

Answer to all the questions (3 Marks):

- 1. What is paradigm ?Mention the different types of paradigm.
- 2. Write a note on the features of procedural programming.
- 3. List some of the features of modular programming
- 4. What do you mean by modularization and software reuse?
- 5. Define information hiding.

PART IV

Answer to all the questions (5 Marks):

- 1. Write the differences between Object Oriented Programming and procedural programming
- 2. What are the advanatges of OOPs?
- 3 Write a note on the basic concepts that suppors OOPs?

Reference:

- (1) Object Oriented Programming with C++ (4th Edition), Dr. E. Balagurusamy, Mc.Graw Hills.
- (2) The Complete Reference C++ (Forth Edition), Herbert Schildt.Mc.Graw Hills.
- (3) Computer Science with C++ (A text book of CBSE XI and XII), SumitaArora, DhanpatRai& Co.
- (4) A text book of CBSE XI and XII computer science by PreetiArora and Pinky Gupta.
- (5) Computer Science with C++ Reeta shoo and Gagansahoo
- (6) The C++ Programming Language,BjarneStroustrup
- (7) C++ Primer (5th Edition) by S. B. Lippman, J. Lajoie

Unit IV

Object Oriented Programming with C++

Classes and objects





Learining Objectives

After learning this chapter, the students will be able to

- Understand the purpose of classes, objects Constructors and Destructors
- able to construct C++ programs using classes with Constructors and Destructors
- Execute and debug class programs with Constructors and Destructors

14.1 Introduction to Classes

The most important feature of C++ is the "Class". It is significance is highlighted by the fact that Bjarne Stroustrup initially gave the name 'C with classes'. C++ offers classes, which provide the four features commonly present in OOP languages: **Abstraction, Encapsulation, Inheritance,** and **Polymorphism**.

14.1.1 Need for Class

Class is a way to bind the data and its associated functions together. Classes are needed to represent real world entities that not only have data type properties but also have associated operations. It is used to create user defined data type 14.1.2 Declaration of a class

A class is defined in C++ using the keyword **class** followed by the name of the class. The body of the class is defined inside the curly brackets and terminated either by a semicolon or a list of declarations at the end.

Note

The only difference between structure and class is the members of structure are by default **public** where as it is **private in class**.

The General Form of a class definition
class class-name
{
private:
variable declaration;
function declaration;
protected:
variable declaration;
function declaration;
public:
variable declaration;
function declaration;
};

• The class body contains the declaration of its members (Data member and Member functions).

• The class body has three access specifiers (visibility labels) viz., private , public and protected.

14.1.3 Class Access Specifiers

Data hiding is one of the important features of Object Oriented Programming which allows preventing the functions of a program to access directly the internal representation of a class type. The access restriction to the class members is specified by public, private, and protected sections within the class body. The keywords public, private, and protected are called access specifiers. The default access specifier for members is private. A public member is accessible from anywhere outside the class but within a program.You can set and get the value of public data members even without using any member function.

The Private Members

A private member cannot be accessed from outside the class. Only the class member functions can access private members.By default all the members of a class would be private.

The Protected Members

A protected member is very similar to a private member but it Provides one additional benefit that they can be accessed

The Public Members

in child classes which are called derived classes (inherited classes).

Example

Keyword class intimates the comp Class name or tag name ac object of the same class type	piler that it is a class definition ts as a user defined data type. Using this, will be created.
class student	
{	
Private:	These are private access specifier members
char name [10]; int rollno, mark1, mark2, total;	That means these members cannot be accessed from outside
void accept(); void compute();	These are protected access specifier members These members also cannot be accessed from
public:	outside
<pre>void display();</pre>	→ Members under this specifier can be accessed from outside
))	jioni ouisiae

Note

If all members of the class are defined as private, then the class become freezed .The object of the class can not access anything from the class.

Activity 1

State the reason for the invalidity of the following code fragment

(i)	(ii)
<pre>class count { int first; int second; public: int first; };</pre>	<pre>class item { int prd; }; item int prdno;</pre>

14.1.4 Definition of class members

Class comprises of members. Members are classified as Data Members and Member functions. Data members are the data variables that represent the features or properties of a class. Member functions are the functions that perform specific tasks in a class. Member functions are called as methods, and data members are also called as attributes.

Example



Classes also contain some special member functions called as constructors and destructors.

Note

14.1.5 Defining methods of a class

Without defining the methods (functions), class definition will become incomplete. The member functions of a class can be defined in two ways.

- (1) Inside the class definition
- (2) Outside the class definition
- (1) Inside the class definition

When a member function is defined inside a class, it behaves like inline functions. These are called Inline member functions.



(2) Outside the class definition

When Member function defined outside the class just like normal function definition (Function definitions you are familiar with) then it is be called as **outline member function or non-inline member function. Scope resolution operator (::) is used for this purpose.** The syntax for defining the outline member function is

Syntax

```
return_type class_name :: function_name (parameter list)
{
       function definition
}
          For example:
                                      <u> → Member function</u>
           void add :: display()
                                      \rightarrow Scope resolution operator
                                     \rightarrow Class name / tag
                                     \rightarrow Data type of the member function
  Illustration 14.1 Inline and Outline member function
  # include <iostream>
                                     Absence of access specifier means
  using namespace std;
                                     that members are private by default..
  class Box
  {
         double width;
                               // no access specifier mentioned
  public:
         double length;
                               //inline member function definition
  void printWidth()
          ł
                cout<<"\n The width of the box is..."<<width;</pre>
  void setWidth( double w ); //prototype of the function
  };
  void Box :: setWidth(double w) // outline member function definition
  {
  width=w;
  }
```

```
int main()
{
    Sox b; // object for class Box
b.setWidth(10.0); // Use member function to set the width.
b.printWidth(); // Use member function to print the width.
return 0;
}
Output:
The width of the box is... 10
```

Note

Declaring a member function having many statements, looping construct, switch or goto statement as inline is not advisable.

ACTIVITY 2

class area

{

int s;

public:

void calc();

};

Write an outline function definition for calc(); which finds the area of a square

Evaluate Yourself

- Define a class in general and in C++'s context
- 2. What is the purpose of a class specifier?
- 3. Compare a structure and a class in C++ context.
- 4. Compare private and puplic access specifier.
- 5. What is non-inline member function? Write its syntax.

14.2 Creating Objects

A class specification just defines the properties of a class. To make use of a class specified, the variables of that class type have to be declared. The class variables are called *object*. Objects are also called as *instance* of class.

For example

student s;

In the above statement **s** is an instance of the class **student**.

Objects can be created in two methods,

(1) Global object

(2) Local object

(1) Global Object

If an object is declared outside all the function bodies or by placing their names immediately after the closing brace of the class declaration then it is called as *Global object*. These objects can be used by any function in the program

(2) Local Object

If an object is declared with in a function then it is called *local object*. It cannot be accessed from outside the function.

Illustration 14.2 The	e use of local	object
<pre># include <iostream></iostream></pre>		
<pre># include <conio></conio></pre>		
using namespace std		
class add	//Global class	3
{		A global object can be declared only
int a,b;		for global class. If a class definition
public:		is specified outside the body of all
int sum;		functions in a program then it is called
<pre>void getdata()</pre>		global class
{	l	
a=5;		
b=10;		
sum = a+b;		
}		
} a1;	//global objec	ct
add a2;	//global objec	ct
<pre>int main()</pre>		
{		
add a3;	//Local objec	t for a global class
a1.getdata();		
a2.getdata();		
a3.getdata();		
cout< <a1.sum;< td=""><td>//public data</td><td>member accessed from outside the class</td></a1.sum;<>	//public data	member accessed from outside the class
cout< <a2.sum;< td=""><td></td><td></td></a2.sum;<>		
cout< <a3.sum;< td=""><td></td><td></td></a3.sum;<>		
return 0;		
}		
Output:		
151515		

```
ACTIVITY 3

Identify the error in the following code fragment

class A

{

float x;

void init()

{

A a1;

X1.5=1;

}};

void main()

{ A1.init(); }
```

14.3 Memory allocation of objects

The member functions are created and placed in the memory space only when they are defined as a part of the class specification. Since all the objects belonging to that class use the same member function, **no separate space is allocated for member functions when the objects are created. Memory space required for the member variables are only allocated separately for each object** because the member variables will hold different data values for different objects

Illustration 14.3 Memory alloca	tion for objects
<pre># include <iostream></iostream></pre>	
using namespace std;	The members will be allocated with
class product {	memory space only after the creation of the class type object
int code, quantity;	
float price;	
public:	
<pre>void assignData();</pre>	
void Print();	
};	

```
int main()
{
          product p1, p2;
          cout<<"\n Memory allocation for object p1 " <<sizeof(p1);
          cout<<"\n Memory allocation for object p2 " <<sizeof(p2);
          return 0;
}
Output:
Memory allocation for object p1 12
Memory allocation for object p2 12</pre>
```

Member functions assignData() and Print() belong to the common pool in the sense both the objects p1 and p2 will have access to the code area of the common pool.

Memory for Objects for p1 and p2 is illustrated:



14.4 Referencing class members

The members of a class are referenced (accessed) by using the object of the class followed by the dot (membership) operator and the name of the member.

The general syntax for calling the member function is:

Object_name . function_name(actual parameter);

For example consider the following illustration



Illustration 14.4 C++ program to illustrate the communication of object:

```
#include<iostream>
                                        Calling a member function of an object
using namespace std;
                                        is also known as sending message to
class compute
                                        object or communication with the
{
                                        object.
       int n1,n2; //private by default
public :
       int n:
       int add (int a, int b)
                             //inline member function
       {
                              //int c ; local variable for this function
               int c=a+b;
               return c;
       }
       int prd (int a, int b)
                                //inline member function
               int c=a*b;
               return c;
       // end of class specification
};
compute c1,c2;
                      //global object
int main()
{
       c1.n =c1.add(12,15); //member function is called
       c2.n = c2.add(8,4);
       cout<<"\n Sum of object-1 "<<c1.n;
       cout<<"\n Sum of object-2 "<<c2.n;</pre>
       cout<<"\n Sum of the two objects are "<<c1.n+c2.n;
       c1.n=c1.prd(5,4);
       c2.n=c2.prd(2,5);
       cout<<"\n Product of object-1 "<<c1.n;
       cout<<"\n Product of object-2 "<<c2.n;</pre>
       cout<<"\n Product of the two objects are "<<c1.n*c2.n;
       return 0;
}
```

Output:

Sum of object-1 27 Sum of object-2 12 Sum of the two objects are 39 Product of object-1 20 Product of object-2 10 Product of the two objects are 200

ACTIVITY 5

i) Write member function called display with no return. The function should display all the member value of the class objects.

ii) Try the output of the above coding with the necessary modifications

14.5 Array of objects

An array which contains the class type of element is called array of objects. It is declared and defined in the same way as any other type of array.

```
Illustration 14.5 Array of objects
#include<iostream>
#include<stdio.h>
                                     gets () can accept a string with space
using namespace std;
                                     which is not possible by cin.
class stock
{
       int itemno;
       char itemname[10];
                               // array as member variable
       float price;
       public:
       void getdata()
       cin>>itemno>>price;
       gets(itemname);
}
void putdata()
ł
       cout<<'\n'<<itemno<<'\t'<<itemname<<'\n'<<price;
};
```

$ \begin{cases} \\ \text{stock s[10];} \\ \text{int i;} \\ \text{cout<"Enter the details\n";} \\ \text{for(i=0;i<10;i++)} \\ & s[i].getdata(); \\ \text{cout<"\n Item Details\n";} \\ \text{cout<"\n Item Details\n";} \\ \text{cout<"\n ITEM NO \t ITEM NAME\t PRICE"< \textbf{Output} \\ \text{Item Details} \\ ITEM NO ITEM NAME MAME Mean of the second o$
stock s[10]; int i; cout<<"Enter the details\n"; for(i=0;i<10;i++) s[i].getdata(); cout<<"\n Item Details\n"; cout<<"\n ITEM NO \t ITEM NAME\t PRICE"< <endl; for(i=0;i<10;i++) s[i].putdata(); return 0; } Output Item Details ITEM NO ITEM NAME PRICE I01 APSARA PENCIL PRICE 101 APSARA PENCIL Price10.5 102 FABER CASTELL PENCIL Price10.5 103 NATARAJ PENCIL Price2.0001 103 NATARAJ PENCIL Price50</endl;
int i; cout<<"Enter the details\n"; for(i=0;i<10;i++) s[i].getdata(); cout<<"\n Item Details\n"; cout<<"\nITEM NO \t ITEM NAME\t PRICE"< <endl; for(i=0;i<10;i++) s[i].putdata(); return 0; } Output Item Details ITEM NO ITEM NAME PRICE ITEM NO ITEM NAME PRICE 101 APSARA PENCIL Price10.5 102 FABER CASTELL PENCIL Price10.5 103 NATARAJ PENCIL Price9.75 201 PILOT V7 PEN Price50</endl;
cout<<"Enter the details\n"; for(i=0;i<10;i++) s[i].getdata(); cout<<"\n Item Details\n"; cout<<"\nITEM NO \t ITEM NAME\t PRICE"< <endl; for(i=0;i<10;i++) s[i].putdata(); return 0; } Output Item Details ITEM NO ITEM NAME PRICE ITEM NO ITEM NAME PRICE 101 APSARA PENCIL Price10.5 102 FABER CASTELL PENCIL Price10.5 103 NATARAJ PENCIL Price9.75 201 PILOT V7 PEN Price50</endl;
for(i=0;i<10;i++) s[i].getdata(); cout<<"\n Item Details\n"; cout<<"\nITEM NO \t ITEM NAME\t PRICE"< <endl; for(i=0;i<10;i++) s[i].putdata(); return 0; } Output Item Details ITEM NO ITEM NAME PRICE 101 APSARA PENCIL PRICE 101 APSARA PENCIL Price10.5 102 FABER CASTELL PENCIL Price10.5 103 NATARAJ PENCIL Price12.0001 103 NATARAJ PENCIL Price9.75 201 PILOT V7 PEN Price50</endl;
s[i].getdata(); cout<<"\n Item Details\n"; cout<<"\nITEM NO \t ITEM NAME\t PRICE"< <endl; for(i=0;i<10;i++) s[i].putdata(); return 0; } Output Item Details ITEM NO ITEM NAME PRICE ITEM NO ITEM NAME PRICE 101 APSARA PENCIL PRICE 101 APSARA PENCIL Price10.5 102 FABER CASTELL PENCIL Price12.0001 103 NATARAJ PENCIL Price9.75 201 PILOT V7 PEN Price50</endl;
cout<<"\n Item Details\n"; cout<<"\nITEM NO \t ITEM NAME\t PRICE"< <endl; for(i=0;i<10;i++) s[i].putdata(); return 0; } Output Item Details ITEM NO ITEM NAME PRICE 101 APSARA PENCIL PRICE 101 APSARA PENCIL Price10.5 102 FABER CASTELL PENCIL Price12.0001 103 NATARAJ PENCIL Price9.75 201 PILOT V7 PEN Price50</endl;
cout<<"\nITEM NO \t ITEM NAME\t PRICE"< <endl; for(i=0;i<10;i++) s[i].putdata(); return 0; } Output Item Details ITEM NO ITEM NAME PRICE I01 APSARA PENCIL PRICE 101 APSARA PENCIL Price10.5 102 FABER CASTELL PENCIL Price12.0001 103 NATARAJ PENCIL Price9.75 201 PILOT V7 PEN Price50</endl;
for(i=0;i<10;i++) s[i].putdata(); return 0; } Output Item Details ITEM NO ITEM NAME PRICE 101 APSARA PENCIL PRICE 102 FABER CASTELL PENCIL Price10.5 102 FABER CASTELL PENCIL Price12.0001 103 NATARAJ PENCIL Price9.75 201 PILOT V7 PEN Price50
s[i].putdata(); return 0; } Output Item Details ITEM NO ITEM NAME PRICE 101 APSARA PENCIL PRICE 102 FABER CASTELL PENCIL Price10.5 103 NATARAJ PENCIL Price12.0001 103 NATARAJ PENCIL Price9.75 201 PILOT V7 PEN Price50
return 0; } Output Item Details ITEM NO ITEM NAME PRICE 101 APSARA PENCIL Price10.5 102 FABER CASTELL PENCIL Price12.0001 103 NATARAJ PENCIL Price9.75 201 PILOT V7 PEN Price50
 } Output Item Details ITEM NO ITEM NAME PRICE 101 APSARA PENCIL Price10.5 102 FABER CASTELL PENCIL Price12.0001 103 NATARAJ PENCIL Price9.75 201 PILOT V7 PEN Price50
OutputItem DetailsITEM NOITEM NAMEPRICE101APSARA PENCILPrice10.5102FABER CASTELL PENCILPrice12.0001103NATARAJ PENCILPrice9.75201PILOT V7 PENPrice50
Item DetailsPRICEITEM NOITEM NAMEPRICE101APSARA PENCILPrice10.5102FABER CASTELL PENCILPrice12.0001103NATARAJ PENCILPrice9.75201PILOT V7 PENPrice50201CELLO PLITTERELOWPrice10
ITEM NOITEM NAMEPRICE101APSARA PENCILPrice10.5102FABER CASTELL PENCILPrice12.0001103NATARAJ PENCILPrice9.75201PILOT V7 PENPrice50201CELLO PUTTERELOWPrice10
101APSARA PENCILPrice10.5102FABER CASTELL PENCILPrice12.0001103NATARAJ PENCILPrice9.75201PILOT V7 PENPrice50201CELLO PUTTERELOWPrice10
102FABER CASTELL PENCILPrice12.0001103NATARAJ PENCILPrice9.75201PILOT V7 PENPrice50201CELLO PUTTERELOWPrice10
103NATARAJ PENCILPrice9.75201PILOT V7 PENPrice50201CELLO PLITTERELOWPrice10
201PILOT V7 PENPrice50201CELLO PLITTERELOWPrice10
$201 \qquad \qquad CELLO DUTTTEDELOW \qquad Dute 10$
201 CELLO BUTTERFLOW Priceto
202PARKER PENPrice450
202PILOT FOUNTAIN PENPrice600
301APSARA ERASERPrice5
302FABER CASTELL ERASERPrice15.0001



Memory representation of array of 10 Stock objects

The above program accepts the details of 10 items and display the same. In the program if you observe there are 10 objects created and are accessing the member function through for loop. To invoke putdata() for a particular object for example for object 6 we will give

s[5].putdata(); //array index start with 0. So index 5 refers to 6th object
14.6 Functions in a class

A wide range of operations for the class members are performed using member functions of the class .These operations are defined in the member function.

14.6.1 Objects accessing inline and outline member functions

As you are aware the definition of the member function can be given either inside the class specification or outside the class

Let us see from the illustrated example program. This program defines a class called sales with the following description

private members of class salesman	
sno	integer(salesperson number)
sname	15 characters
hrwrk ,wage	float (hours worked and wage per hour)
totwage	float(<i>hrwrk</i> * <i>wage</i>)
calcwg()	a function to find <i>hrwrk</i> * <i>wage</i> and store it in <i>totwage</i>
public members of class salesman	
in_data()	a function to accept values for <i>sno</i> , <i>sname</i> , <i>hrwrk</i> , <i>wage</i> and invoke calcwg() to calculate <i>totwage</i>
out_data()	a function to display all the data members on the screen. you should give definitions of functions as outline

Illustration 14.6 Inline Outline and function

#include <iostream></iostream>	
using namespace std	An inline member function definition
class sales	should be placed above all the functions
{	that call it
int sno;	
char sname [15];	
float hrwrk,wage,totwage;	
void calcwg()	
{	
totwage=hrwrk * w	rage;
}	
public :	The prototype of the outline member
<pre>void in_data();</pre>	function given in a class specification,
<pre>void out_data();</pre>	instructs the compiler about its visibility
};	mode.

```
void sales :: in data()
{
       wage=75.26;
       totwage=0.0;
       cout<<"\nEnter the salesperson id";cin>>sno;
       cout<<"\nEnter the name" ;gets(sname);</pre>
       cout<<"\nEnter the hours worked" ;cin>>hrwrk;
       calcwg(); //member function called inside by another member function
}
void sales :: out_data()
       cout<<"\n Wage slip ";</pre>
       cout<<"\n ~~~~~ ";
       cout<<"\nID : " << sno;
       cout<<"\nName : " <<sname ;</pre>
       cout<<"\nHours worked :" <<hrwrk;</pre>
       cout<<"\nTotal Wage :"<<setprecision(2)<<totwage;</pre>
}
int main()
ł
       sales sal;
       sal.in_data();
       sal.out_data();
       return 0;
}
Output:
Enter the salesperson id 1201
Enter the name ARUL
Enter the hours worked 7
Wage slip
~~~~~~
ID:1201
Name : ARUL
Hours worked :7
Total Wage :526.82
```

14.7 Nesting of member functions

You know that only the public members of a class can be accessed by the object of that class, using dot operator. However a member function can call another member function of the same class directly without using the dot operator. This is called as *nesting of member functions*.

```
Illustration 14.7 The use of Nesting of Member Function
#include<iostream>
using namespace std
                                       A member function can call another
class nest
                                       member function of the same class for
{
                                       that you do not need an object.
        int a:
        int square_num( )
        ł
               return a* a;
        }
        public:
        void input_num( )
        {
               cout<<"\nEnter a number ";</pre>
               cin>>a;
        int cube_num( )
               return a* a*a;
void disp_num()
{
        int sq=square_num(); //nesting of member function
int cu=cube_num(); //nesting of member function
        cout<<"\nThe square of "<<a<<" is " <<sq;
        cout<<"\nThe cube of "<<a<<" is " <<cu;
}
};
int main()
{
        nest n1;
        n1.input_num();
        n1.disp_num();
        return 0;
}
Output:
Enter a number 5
The square of 5 is 25
The cube of 5 is 125
```

In the above program *disp_num()* function calls two other member function *square_num()* and *cube_num()*. Both are defined in different visibility mode.

A member function can access not only the public functions but also the private functions of the class it belongs to.

Note

14.7.1 The Scope Resolution Operator

If there are multiple variables with the same name defined in separate blocks then :: (scope resolution) operator will reveal the hidden file scope(global) variable.

Illustration 14.8 The use of scope	resolution operator
<pre>#include<iostream> using namespace std; int a=100; class A {</iostream></pre>	Recall :: is also used to identify the class to which a member function belongs to.
<pre>int a; public: void fun() { a=20; a+=::a; //using global v cout<<a; } }; int main()</a; </pre>	variable value
{	

In the above program the class data member and the global variable have same name. To use the global variable :: used.

14.8 Objects as Function arguments

Objects can also be passed as arguments to a member function just like any other data type of C++.Objects can also be passed in both ways

- (1) Pass By Value
- (2) Pass By Reference

14.8.1 Pass By Value

When an object is passed by value the function creates its own copy of the object and works on it . Therefore any changes made to the object inside the function do not affect the original object.

```
Illustration 14.9 C++ program to illustrate how the pass by value
method work
 #include <iostream>
 using namespace std;
class Sample
 {
private:
                                 we can assign one object to another
        int num:
                                 object, Similar to structure object
public:
        void set (int x)
        num = x;
void pass(Sample obj1, Sample obj2) //objects are passed
        obj1.num=100; // value of the object is changed inside the function
        obj2.num=200; // value of the object is changed inside the function
        cout<<"\n\n Changed value of object1 "<<obj1.num;
        cout<<"\n\n Changed value of object2 "<<obj2.num;
}
        void print( )
               cout<<num;
};
int main()
 {
        //object declarations
        Sample s1;
        Sample s2;
        Sample s3;
        //assigning values to the data member of objects
        s1.set(10);
        s2.set(20);
        cout<<"\n\t\t Example program for pass by value\n\n\n";
        //printing the values before passing the object
        cout<<"\n\nValue of object1 before passing";</pre>
        s1.print();
        cout<<"\n\nValue of object2 before passing ";</pre>
        s2.print();
```

```
//passing object s1 and s2
        s3.pass(s1,s2);
        //printing the values after returning to main
         cout<<"\n\nValue of object1 after passing ";</pre>
        s1.print();
        cout<<"\n\nValue of object2 after passing ";</pre>
        s2.print();
        return 0;
}
Output:
Example program for PASS BY VALUE
Value of object1 before passing 10
Value of object2 before passing 20
Changed value of object 1 100
Changed value of object 200
Value of object 1 after passing 10
Value of object 2 after passing 20
```

In the above program the objects s1,and s2 are passed to pass() method. They are copied to obj1 and obj2 respectively. The data member num's value is changed inside the function. But it didn't affected the s1 and s2 objects data member.

14.8.2 Pass By Reference

When an object is passed by reference, its memory address is passed to the function so the called function works directly on the original object used in the function call. So any changes made to the object inside the function definition are reflected in original object.

```
Illustration 14.10 C++ program to illustrate how the pass by reference
method work

#include <iostream>
using namespace std;
class Sample
{
    private:
        int num;
public:
        void set (int x)
        {
            num = x;
        }
```

```
void pass(Sample & obj1, Sample & obj2) //objects are passed
{
        obj1.num=100; // value of the object is changed inside the function
        obj2.num=200; // value of the object is changed inside the function
        cout<<"\n\n Changed value of object1 "<<obj1.num;
        cout<<"\n\n Changed value of object2 "<<obj2.num;
}
       void print()
       {
               cout<<num;
        }
};
int main()
{
       clrscr();
       //object declarations
       Sample s1;
       Sample s2;
       Sample s3;
       //assigning values to the data member of objects
       s1.set(10);
       s2.set(20);
       cout<<"\n\t\t Example program for pass by reference\n\n\n";
       //printing the values before passing the object
       cout<<"\n\nValue of object1 before passing";</pre>
       s1.print();
       cout<<"\n\nValue of object2 before passing ";</pre>
        s2.print();
       //passing object s1 and s2
        s3.pass(s1,s2);
        //printing the values after returning to main
        cout<<"\n\nValue of object1 after passing ";</pre>
        s1.print();
        cout<<"\n\nValue of object2 after passing ";</pre>
        s2.print();
        return 0;
```

}

Output:

Example program for PASS BY REFERENCE Value of object1 before passing10 Value of object 2 before passing 20 Changed value of object1 100 Changed value of object2 200 Value of object1 after passing 100 Value of object2 after passing 200

In the above program the objects s1,and s2 are passed as reference to pass() method. So obj1 and obj2 become reference(alias name) for s1 and s2 respectively. The data member num's value is changed inside the function affected s1 and s2 objects data member.

14.9 Functions Returning objects

Member Functions not only receive object as argument it can also return an object.

```
Illustration 14.11 C++ program to illustrate how an object is returned
to the calling function
 #include <iostream>
 using namespace std;
 class Sample
 {
        private:
              int num;
        public:
        void set (int x)
        {
        num = x;
        Sample pass(Sample obj1, Sample obj2) //
        {
         Sample s4;
         s4.num=obj1.num+obj2.num;
         return s4;
        }
        }
        void print()
        {
```

```
cout<<num;
       }
};
int main()
{
       //object declarations
       Sample s1;
       Sample s2;
       Sample s3;
       //assigning values to the data member of objects
       s1.set(10);
       s2.set(20);
       cout<<"\n\t\t Example program for Returning an object \n\n\n";
       //passing object s1 and s2
       s3=s3.pass(s1,s2);
       //printing the values of object
       cout<<"\nThe value of s1.num is ";</pre>
       s1.print();
       cout<<"\nThe value of s2.num is ";</pre>
       s2.print();
       //printing the sum
       cout<<"\nThe sum s3.num is ";s3.print();</pre>
       return 0;
}
Output:
Example program for Returning an object
The value of s1.num is 10
The value of s2.num is 20
The sum s3.num is 30
```

14.10 Nested class

When one class become the member of another class then it is called Nested class and the relationship is called containership.

Classes can be nested in two ways.

- 1. By defining a class within another class
- 2. By declaring an object of a class as a member to another class

Defining a class with in another

When a class is declared with in another class, the inner class is called as Nested class (ie the inner class) and the outer class is known as Enclosing class. Nested class can be defined in private as well as in the public section of the Enclosing class.

```
Illustration 14.12 C++ program to illustrate the nested class
#include<iostream>
using namespace std;
class enclose
private:
        int x:
        class nest
private :
        int y;
public:
int z;
void prn()
 {
        y=3;z=2;
        cout<<"\n The product of "<<y<<'*'<<z<<"= "<<y*z<<"\n";
}
}; //inner class definition over
nest n1;
public:
nest n2;
void square()
n2.prn(); //inner class member function is called by its object
        x=2;
        n2.z=4;
        cout<<"\n The product of " <<n2.z<<'*'<<n2.z<<"= "<<n2.z*n2.z<<"\n";
        cout<<"\n The product of " <<x<<"= "<<x*x;
        }
               //outer class definition over
};
int main()
 {
enclose e;
        e.square(); //outer class member function is called
}
```

Output: The product of 3*2=6 The product of 4*4=16 The product of 2*2=4

In the above program the inner class nest is defined inside the outer class enclose. nest is accessed by enclose by creating an object of nest

14.10.1 Containership in C++

Whenever an object of a class is declared as a member of another class it is known as a container class. In the container-ship the object of one class is declared in another class.

```
Illustration 14.13 C++ program to illustrate the containership
#include<iostream>
using namespace std;
class outer
 {
        int data;
        public:
        void get();
};
class inner
        int value;
        outer ot;
                        // object ot of class outer is declared in class inner
        public:
        void getdata();
};
void outer :: get()
 {
        cout<<"\nEnter a value";</pre>
        cin>>data;
         cout<<"\nThe given value is "<<data;
void inner :: getdata()
        cout<<"\nEnter a value";</pre>
        cin>>value;
        cout<<"\nThe given value is "<<value;</pre>
        ot.get();
                    //calling of get() of class outer in getdata() of class inner
}
```

```
int main()
{
    inner in;
    in.getdata();
    return 0;
}
Output:
Enter a value10
The given value is 10
Enter a value 20
The given value is 20
```

In the above program class outer and inner are defined separately. But both the classes are connected by the object '*ot*' which is a member of class inner

14.11 Introduction to Constructors

The definition of a class only creates a new user defined data type. The instances of the class type should be instantiated (created and initialized). Instantiating object is done using constructor.

14.11.1 Need for Constructors

An array or a structure in c++ can be initialized during the time of their declaration.

For example

struct sum	
{ int n1,n2; }; class add	Member function of a class can access all the members irrespective of their associated access specifier.
<pre>{ int num1,num2; }; int main()</pre>	
<pre>{ int arr[]={1,2,3}; sum s1={1,1}; add a1={0,0}; }</pre>	<pre>//declaration and initialization of array //declaration and initialization of structure object // class object declaration and initialization throws</pre>

The initialization of class type object at the time of declaration similar to a structure or an array is not possible because the class members have their associated access specifiers (private or protected or public). Therefore Classes include special member functions called as *constructors*. The constructor function initializes the class object.

14.12 Declaration and Definition

When an instance of a class comes into scope, a special function called the *constructor* gets executed. The constructor function name has the same name as the class name. The constructors return nothing. They are not associated with any data type. It can be defined either inside class definition or outside the class definition.

Example 1:

```
Illustration 14.14 A constructor defined inside the class specification.
#include<iostream>
using namespace std;
class Sample
        {
        int i,j;
            public :
        int k;
            Sample()
        {
            i=j=k=0;//constructor defined inside the class
        }
      };
```

$Illustration\,14.15\,A\,constructor\,defined\,inside\,the\,class\,specification.$

```
#include<iostream>
using namespace std;
class Sample
        {
        int i,j;
            public :
        int k;
            Sample()
            {
            i=j=k=0;//constructor defined inside the class
            }
        };
        int main()
        {
            Samples1;
            return 0;
        }
        Output:
```

ACTIVITY 6

In the above program justify your reason for no output

Example 2:

<pre>specification. #include<iostream> using namespace std; class Data { int p,q; public : int r; Data(); //only prototype to be specified here to intimate its access specifier }</iostream></pre>
<pre>#include<iostream> using namespace std; class Data { int p,q; public : int r; Data(); //only prototype to be specified here to intimate its access specifier }</iostream></pre>
using namespace std; class Data { int p,q; public : int r; Data(); //only prototype to be specified here to intimate its access specifier
<pre>class Data { int p,q; public : int r; Data(); //only prototype to be specified here to intimate its access specifier } }</pre>
{ int p,q; public : int r; Data(); //only prototype to be specified here to intimate its access specifier
<pre>int p,q; public : int r; Data(); //only prototype to be specified here to intimate its access specifier }</pre>
<pre>public : int r; Data(); //only prototype to be specified here to intimate its access</pre>
int r; Data(); //only prototype to be specified here to intimate its access specifier
Data(); //only prototype to be specified here to intimate its access specifier
specifier
Data ::Data()
{
p=q=r=0; // constructor defined outside the class
}
int main()
{
Data d1;
return 0;
}

Note

A constructor can be defined either in private or public section of a class. But it is advisable to defined in public section of a class ,so that its object can be created in any function.

```
Illustration 14.17 illustrate a constructor defined inside the private
visibility.
#include<iostream>
using namespace std;
class X
 {
       int num;
       X()
   {
       num=k=0;
    }
 public:
 int k;
};
int main()
{
X x; // The constructor of X cannot accessed by main() because main() is a
       //non member function
       //and the compiler throws error message
                                                   [Error] 'X::X()' is private
return 0;
}
```

14.12.1 Functions of constructor

As we know now that the constructor is a special initialization member function of a class that is called automatically whenever an instance of a class is declared or created. The main function of the constructor is

1) To allocate memory space to the object and

2) To initialize the data member of the class object

There is an alternate way to initialize the class objects but in that case we have to explicitly call the member function.

```
Illustration 14.18 illustrate a member function initializes the data
member.
#include<iostream>
                                    After creating the object the getvalue()
using namespace std;
                                    should be explicitly called to initialize
class Sample
                                    the object.
 {
       int i, j;
       public :
       int k:
       void getvalue()
        {
                                  //member function
               i=j=k=0;
        }
  };
int main()
{
Sample s1;
   s1.getvalue();
                            //member function initializes the class object
return 0;
}
```

14.12.2 Default Constructors

A constructor that accepts no parameter is called default constructor. For example in the class data program Data ::Data() is the default constructor. Using this constructor Objects are created similar to the way the variables of other data types are created.

Example

int num;	//ordinary variable declaration	
Data d1;	// object declaration	

If a class does not contain an explicit constructor (user defined constructor) the compiler automatically generate a default constructor implicitly as an inline public member.

Illustration 14.19 illustrate the compiler generated constructor

```
#include<iostream>
using namespace std;
class Sample
     {
       int i, j;
       public:
       int k; //no user defined constructor in this program
       void getvalue()//member function
        {
          i=j=k=0;
        }
     };
int main()
{
       Sample s1; //uses the default constructor generated by the compiler
       s1.getvalue();
       return 0;
}
```

Note

In the absence of user defined constructor the compiler automatically provides the default constructor. It simply allocates memory for the object.

Illustration 14.20 to illustrate the constructor and other member function in a class

```
#include<iostream>
using namespace std;
class simple
{
    private:
        int a,b;
    public:
    simple()
    {
            a= 0;
            b= 0;
            cout<< "\n Constructor of class-simple ";
    }
}</pre>
```

```
void getdata()
               cout<<"\n Enter values for a and b (sample data 6 and 7)... ";
               cin>>a>>b;
       void putdata()
       {
               cout<<"\nThe two integers are... "<<a<<'\t'<< b<<endl;
               cout<<"\n The sum of the variables "<<a<<" + "<<b<<" = "<<a+b;
       }
};
int main()
{
       simple s;
       s.getdata();
       s.putdata();
       return 0;
}
Output:
Constructor of class-simple
Enter values for a and b (sample data 6 and 7)... 67
The two integers are... 6
                           7
The sum of the variables 6 + 7 = 13
```

14.12.3 Parameterized Constructors

A constructor which can take arguments is called parameterized constructor .This type of constructor helps to create objects with different initial values. This is achieved by passing parameters to the function.

```
Illustration 14.21 to illustrate the Parameterized constructor used for
creating objects
#include<iostream>
using namespace std;
class simple
{
    private:
    int a,b;
    public:
    simple(int m, int n)
    {
```

```
a = m;
b=n;
cout<< "\n Parameterized Constructor of class-simple "<<endl;</pre>
}
void putdata()
{
cout<<"\nThe two integers are... "<<a<<'\t'<< b<<endl;
cout<<"\n The sum of the variables "<<a<<" + "<<b<<" = "<< a+b;
}
};
int main()
{
simple s1(10,20),s2(30,45); //Created two objects with different values created
cout<<"\n\t\tObject 1\n";</pre>
s1.putdata();
cout<<"\n\t\tObject 2\n";</pre>
s2.putdata();
return 0;
}
Output:
Parameterized Constructor of class-simple
Parameterized Constructor of class-simple
Object 1
The two integers are .. 10
                             20
The sum of the variables 10 + 20 = 30
Object 2
The two integers are... 30
                             45
The sum of the variables 30 + 45 = 75
```

Note

Declaring a constructor with arguments hides the compiler generated constructor .After this we cannot invoke the compiler generated constructor.

```
Illustration 14.22 to illustrate the creation of object with no argument
after defining parameterized constructor throws error
#include<iostream>
using namespace std;
                                      Note:- Just like normal function
class simple
                                      parameterized constructor can also
{
                                       have default arguments.
private:
 int a,b;
public:
simple(int m, int n)
{
a = m;
b=n:
cout<< "\n Parameterized Constructor of class-simple "<<endl;</pre>
void putdata()
cout<<"\nThe two integers are .. "<<a<<'\t'<< b<<endl;
cout<<"\n The sum of the variables "<a<<" + "<b<<" = "<a+b;
};
int main()
{
simple s,s1(10,20) // [Error] no matching function for call to 'simple::simple()'
s1.putdata();
s2.putdata();
return 0;
}
```

Note

Just like normal function parameterized constructor can also have default arguments.

Illustration 14.23 to illustrate the default argument in parameterized constructor

```
#include<iostream>
using namespace std;
class simple
{
    private:
        int a,b;
    public:
    simple(int m, int n=100) //default argument
    {
        a= m;
    }
}
```

```
b=n;
cout<< "\n Parameterized Constructor with default argument"<<endl;
}
void putdata()
cout<<"\nThe two integers are... "<<a<<'\t'<< b<<endl;
cout<<"\n The sum of the variables "<<a<<" + "<<b<<" = "<< a+b;
}
};
int main()
{
simple s1(10,20),s2(50);
cout<<"\n\t\tObject 1 with both values \n";</pre>
s1.putdata();
cout<<"\n\t\tObject 2 with one value and one deafult value\n";</pre>
s2.putdata();
return 0;
}
Output:
Parameterized Constructor with default argument
Parameterized Constructor with default argument
         Object 1 with both values
The two integers are... 10
                             20
The sum of the variables 10 + 20 = 30
         Object 2 with one value and one deafult value
The two integers are... 50
                             100
The sum of the variables 50 + 100 = 150
```

14.13 Significance of Default constructors

Default constructors are very useful to crate objects without having specific initial value. It is also used to create array of objects.

Illustration 14.24 to illustrate the significance of default constructor
#include <iostream></iostream>
using namespace std;
class simple
private:
int a, b;
public:
simple() //default constructor

```
{
a=0;
b = 0;
cout<< "\n default constructor"<<endl;</pre>
}
int getdata();
};
int simple :: getdata()
{ int tot;
cout<<"\nEnter two values ";
cin>>a>>b;
tot=a+b;
return tot;
}
int main()
{
int sum=0;
simple s1[3];
cout<<"\n\t\tObject 1 with both values \n";
for (int i=0;i<3;i++)
  sum+=s1[i].getdata();
cout<<"\nsum of all object values is"<<sum;</pre>
return 0;
}
Output:
default constructor
default constructor
default constructor
         Object 1 with both values
Enter two values 10 20
Enter two values 30 40
Enter two values 50 50
sum of all object values is200
```

14.14 Invocation of constructors

There are two ways to create an object using parameterized constructor

- Implicit call
- Explicit call

14.14.1 Implicit call

In this method ,the parameterized constructor is invoked automatically whenever an object is created. For example simple s1(10,20); in this for creating the object s1 parameterized constructor is automatically invoked.

14.14.2 Explicit call

In this method, the name of the constructor is explicitly given to invoke the parameterized constructor so that the object can be created and initialized .

For example

simple s1=simple(10,20); //explicit call

Explicit call method is the most suitable method as it creates a temporary object ,the chance of data loss will not arise. A temprory object lives in memory as long as it is being used in an expression. After this it get destroyed.



s2.putdata(); return 0; }	Explicit call to constructor creates a temporary instance
Constructor of class-simple invoked for	implicit and explicit call
Constructor of class-simple invoked for	implicit and explicit call
Object 1	
The two integers are 10 20	
The sum of the variables $10 + 20 = 30$	
Object 2	
The two integers are 30 45	
The sum of the variables $30 + 45 = 75$	

14.15 Copy Constructors

A constructor having a reference to an already existing object of its own class is called copy constructor .In other words Copy Constructor is a type of constructor which is used to create a copy of an already existing object of a class type. It is usually of the form simple (simple&), where simple is the class name. The compiler provides a default Copy Constructor to all the classes.

14.15.1 Calling copy constructors

A copy constructor is called

1) When an object is passed as a parameter to any of themember functions

Example void simple::putdata(simple x);

2) When a member function returns an object

Example simple getdata() { }

3) When an object is passed by reference to an instance of its own class

For example, simples1, s2(s1); // s2(s1) calls copy constructor

Illustration 14.26 to illustrate Copy constructor

#include <iostream>
using namespace std;
class Test
{

```
private:
  int X;
  int Y;
  public:
        Test (int , int ); //parameterized constructor declaration
        Test (Test &); //Declaration of copy constructor to initialize data members.
void Display();
};//End of class
Test:: Test(int a, int b) //Definition of parameterized constructor.
{
  X = a;
  Y = b;
}
Test::Test(Test &T) //Definition of copy constructor.
{
  X = T.X;
  Y = T.Y;
}
void Test:: Display()//Definition of Display () member function.
{
cout<<endl<< "X: " << X;
cout<<endl<< "Y: " << Y <<endl;
}
int main()
{
        Test T1(10,20); //Parameterized Constructor automatically called when
        //object is created.
        cout<<endl<<"T1 Object: " <<endl;</pre>
        cout<< "Value after initialization : ";</pre>
        T1.Display();
        Test T2(T1);//Intialize object with other object using copy constructor
        cout<<endl<< "T2 Object: " <<endl;</pre>
        cout<< "Value after initialization : ";</pre>
        T2.Display();
        return 0;
}
Output:
T1 Object:
Value after initialization :
X: 10
Y: 20
T2 Object:
Value after initialization :
X: 10
Y: 20
```

While defining copy constructor the argument (object) should be passed only by reference not by value method.

Note

14.16 Order of constructor invocation

The constructors are executed in the order of the object declared. (If it is in same statement left to right)

For example

Test t1;

Test t2; // the order of constructor execution is first for t1 and then for t2.

Let us consider the following example

Sample s1,s2,s3; //The order of construction is s1 then s2 and finally s3

But if a class containing an object of another class as its member then that class constructor is executed first.

```
Illustration14.27 to illustrate order of execution of constructor
#include<iostream>
using namespace std;
class outer
 {
       int data;
       public:
       outer()
       {
       cout<<"\nconstructor of class outer ";</pre>
        }
  };
class inner
 {
                    // object ot of class outer is declared in class inner
       outer ot;
       public:
       inner()
       {
       cout<<"\n constructor of class inner ";}</pre>
 };
```

```
int main()
{
    inner in;
    return 0;
}
Output:
constructor of class outer
constructor of class inner
```

14.17 Dynamic initialization of Objects

When the initial values are provided during runtime then it is called dynamic initialization.

```
Illustration14.28 to illustrate dynamic initialization
#include<iostream>
using namespace std;
class X
{
   int n;
  float avg;
  public:
       X(int p,float q)
       {
       n=p;
       avg=q;
       }
void disp()
       {
       cout<<"\n Roll numbe:- " <<n;</pre>
       cout<<"\nAverage :- "<<avg;</pre>
       }
};
int main()
{
int a ; float b;
```

```
cout<<"\nEnter the Roll Number";
cin>>a;
cout<<"\nEnter the Average";
cin>>b;
X x(a,b); // dynamic initialization
x.disp();
return 0;
```

Output:

}

Enter the Roll Number 1201 Enter the Average 98.6 Roll numbe:- 1201 Average :- 98.6

14.18 Characteristics of Constructors

- The name of the constructor must be same as that of the class
- No return type can be specified for constructor
- A constructor can have parameter list
- The constructor function can be overloaded
- They cannot be inherited but a derived class can call the base class constructor
- The compiler generates a constructor, in the absence of a user defined constructor.
- Compiler generated constructor is public member function
- The constructor is executed automatically when the object is created
- A constructor can be used explicitly to create new object of its class type

14.19 Destructors

When a class object goes out of scope, a special function called the *destructor* gets executed. The destructor has the same name as the class tag but prefixed with a ~(tilde). Destructor function also return nothing and it does not associated with anydata type.

14.19.1 Need of Destructors

The purpose of the destructor is to free the resources that the object may have acquired during its lifetime. A destructor function removes the memory of an object which was allocated by the constructor at the time of creating a object.

14.20 Declaration and Definition

A *destructor* is a special member function that is called when the lifetime of an object ends and destroys the object constructed by the constructor. Normally declared under public.

```
Illustration14.29 to illustrate destructor function in a class
#include<iostream>
using namespace std;
class simple
{
private:
int a, b;
public:
simple()
{
a = 0;
b = 0;
cout<< "\n Constructor of class-simple ";</pre>
}
void getdata()
{
cout<<"\n Enter values for a and b (sample data 6 and 7)... ";
cin>>a>>b;
}
void putdata()
{
cout<<"\nThe two integers are .. "<<a<<'\t'<< b<<endl;
cout<<"\n The sum of the variables "<<a<<" + "<<b<<" = "<<a+b;
}
~simple()
{ cout<<"\n Destructor is executed to destroy the object";}
};
int main()
{
```

simple s; s.getdata(); s.putdata(); return 0; } Output: Constructor of class-simple Enter values for a and b (sample data 6 and 7)... 67 The two integers are .. 6 7 The sum of the variables 6 + 7 = 13Destructor is executed to destroy the object

14.21 Characteristics of Destructors

- The destructor has the same name as that of the class prefixed by the tilde character '~'.
- The destructor cannot have arguments
- It has no return type
- Destructors cannot be overloaded i.e., there can be only one destructor in a class
- In the absence of user defined destructor, it is generated by the compiler
- The destructor is executed automatically when the control reaches the end of class scope to destroy the object
- They cannot be inherited

Points to Remember:

- A class binds data and associated functions together.
- A class in C++ makes a user defined data type using which objects of this type can be created.
- While declaring a class data members , member functions ,access specifiers and class tag name are given.
- The member functions of a class can either be defined within the class (inline) definition or outside the class definition.
- The public members of the class can be accessed outside the class directly by using object of this class type.

Points to Remember:

- A class binds data and associated functions together.
- A class in C++ makes a user defined data type using which objects of this type can be created.
- While declaring a class data members , member functions, access specifiers and class tag name are given.
- The member functions of a class can either be defined within the class (inline) definition or outside the class definition.
- The public members of the class can be accessed outside the class directly by using object of this class type.
- A class supports OOP features ENCAPSULATION by binding data and functionsassociated together.
- A class supports Data hiding by hiding the information from the outside world through private and protected members.
- When a member function is called by another member function of the same class , it is called as nesting of member functions.
- The scope resolution operator (::), when used with the class name depicts that the members belong to that class as in class_ name :: function_name and only used with the variable name as in :: s variable -name, depicts the global variable.(the one with file scope).

- When an instance of a class comes into scope, a special function called the constructor gets executed.
- The constructor function allocates memory and initializes the class object.
- When an instance of a class comes into scope, a special function called the constructor gets executed.
- The constructor function allocates memory and initializes the class object.
- When a class object goes out of scope, a special function called the destructor gets executed.
- The constructor function name and the destructor have the same name as the classtag.
- A constructor without parameters is called as default constructor.
- A constructor with default argument is equivalent to a default constructor
- Both the constructors and destructor return nothing. They are not associated with any data type.
- Objects can be initialized dynamically .



1 Define a class Employee with the following specification

private members of class Employee

empno- integer

ename – 20 characters

basic – float

netpay, hra, da, - float

calculate () - A function to find the basic+hra+da with float return type

public member functions of class employee

havedata() - A function to accept values for empno, ename, basic, hra,

da and call calculate() to compute netpay

dispdata() - A function to display all the data members on the screen

2. Define a class MATH with the following specifications

private members

num1, num2, result – float

init() function to initialize num1, num2 and result to zero

protected members

add() function to add num1 and num2 and store the sum in result

diff() function to subtract num1 from num2 and store the difference in

the result

public members

getdata() function to accept values for num1 and num2

menu() function to display menu

1. Add...

2. Subtract...

invoke add() when choice is 1 and invoke prod when choice is 2 and also display the result.

3. Create a class called Item with the following specifications

private members

code, quantity - Integer data type

price - Float data type

getdata()-function to accept values for all data members with no return

public members

taxt – float

dispdata() member function to display code,quantity,price and tax .The tax is calculated as if the quantity is more than 100 tax is 2500 otherwise 1000.

4. Write the definition of a class FRAME in C++ with following description

Private members

FrameID - Integer data type

Height, Width, Amount - Float data type

SetAmount() -Member function to calculate and assign amount as

10*Height*Width

Public members

GetDetail() Afunction to allow user to entervalues of

FrameID, Height, Width. This function should also call SetAmount() to calculate the amount.

ShowDetail() A function to display the values of all data members.

5. Define a class RESORT in C++ with the following description :

Private Members :

Rno //Data member to store Room No

RName //Data member t store customer name

Charges //Data member to store per day charges

Days //Data member to store number of days of stay

COMPUTE() //A function to calculate and return Amount as

//Days*Chagres and if the value of Days*Charges is

more than 5000 then as 1.02*Days*Charges

Public Members :

Getinfo() //A function to enter the content Rno, Name, Charges //and Days

Displayinfo() //A function to display Rno, RName, Charges, Days and

// Amount (Amount to displayed by calling functionCOMPUTE())

7. struct pno

{

int pin;

float balance;

}

Create a BankAccount classwith the following specifications Protected members pno_obj //array of 10 elements

init(pin) // to accept the pin number and initialize it and initialize

// the balance amount is 0

public members

deposit(pin, amount):

Increment the account balance by accepting the amount and pin. Check the pin number for matching. If it matches increment the balance and display the balance else display an appropriate message

withdraw(self, pin, amount):

Decrement the account balance by accepting the amount and pin. Check the pin number for matching and balance is greater than 1000 and amount is less than the balance. If it matches withdraw the amount and display the balance else display an appropriate message

8. Define a class Hotel in C++ with the following description :

Private Members :

Rno //Data member to store Room No

Name //Data member t store customer name

Charges //Data member to store per day charges

Days //Data member to store number of days of stay

Calculate() //A function to calculate and return Amount as

//Days*Chagres and if the value of Days*Charges is

more than 12000 then as 1.2*Days*Charges

Public Members :

Hotel() //to initialize the class members

Getinfo() //A function to enter the content Rno, Name, Charges //and Days

Showinfo() //A function to display Rno, RName, Charges, Days and

//Amount (Amount to displayed by calling function

CALCULATE())

9. Define a class Exam in C++ with the following description :

Private Members :

Rollno	- Integer data type
Cname	- 25 characters
Mark	- Integer data type

public :

Exam(int,char[],int) //to initialize the object
~Exam() // display message "Result will be intimated shortly"
void Display() // to display all the details if t the mark is
// above 60 other wise display "Result Withheld"

10. Define a class Studentin C++ with the following specification :

Private Members :

A data member Rno(Registration Number) type long

A data member Cname of type string

A data member Agg_marks (Aggregate Marks) of type float

A data member Grade of type char

A member function setGrade () to find the grade as per the aggregate marks

obtained by the student. Equivalent aggregate marks range and the

respective grade as shown below.

Aggregate Marks	Grade
>=90	А
Less than 90 and $>=75$	В
Less than 75 and $\geq=50$	С
Less than 50	D
Public members:

A constructor to assign default values to data members:

A copy constructor to store the value in another object

Rno=0,Cname="N.A",Agg_marks=0.0

A function Getdata () to allow users to enter values for Rno.Cname, Agg_marks and call

functionsetGrade () to find the grade.

A function dispResult() to allow user to view the content of all the data members.

A destructor to display the message "END"



Evaluation	

PART I

Choose the correct answer

- 1. The variables declared inside the class are known as data members and the functions are known as
 - (A) data functions (B) inline functions
 - (C) member functions (D) attributes
- 2. Which of the following statements about member functions are True or False?

i) A member function can call another member function directly with using the dot operator.

ii) Member function can access the private data of the class.

- (A) i-True, ii-True (B) i-False, ii-True
- (C) i-True, ii-False (D) i-False, ii-False
- 3. A member function can call another member function directly, without using the dot operator called as
 - (A) sub function
 - (B) sub member

	(C) nesting of member function		
	(D) sibling of member function		
4.	4. The member function defined within the class behave like		
	(A) inline functions	(B) Non inline function	
	(C) Outline function	(D) Data function	
5.	Which of the following access specifier protects data from inadvertent modification		
	(A) Private	(B) Protected	
	(C) Public	(D) Global	
6. clas	S X		
{			

int y;

public:

 $x(int z){y=z;}$

} x1[4];

int main()

```
{ x x2(10);
```

return 0;}

How many objects are created for the above program

(A) 10 (B) 14 (C) 5 (D) 2

7. State whether the following statements about the constructor are True or False.

i) constructors should be declared in the private section.

ii) constructors are invoked automatically when the objects are created.

(A) True, True (B) True, False (C) False, True (D) False, False

8. Which of the following constructor is executed for the following prototype ?
 add display(add &); // add is a class name

- (A) Default constructor (B) Parameterized constructor
- (C) Copy constructor (D) Non Parameterized constructor
- 9. What happens when a class with parameterized constructors and having no default constructor is used in a program and we create an object that needs a zero-argument constructor?
 - (A) Compile-time error (B) Domain error
 - (C) Runtime error (D) Runtime exception.
- 10. Which of the following create a temporary instance?
 - (A) Implicit call to the constructor (B) Explicit call to the constructor
 - (C) Implicit call to the destructor (D) Explicit call to the destructor

PART II

Answer to all the questions (2 Marks):

- 1. What are called members?
- 2. Differentiate structure and class though both are user defined data type.
- 3. What is the difference between the class and object in terms of oop?
- 4. Why it is considered as a good practice to define a constructor though compiler can automatically generate a constructor ?
- 5. Write down the importance of destructor.

PART III

Answer to all the questions(3 Marks):

1. Rewrite the following program after removing the syntax errors if any and underline the errors:

```
{}
void register () {cin>>stdid;gets(name);
}
void display ()
{cout<<studid<<": "<<name<<endl;}
}
int main()
{mystud MS;
register.MS();
MS.display();
}</pre>
```

- 2. Write with example how will you dynamically initialize objects?
- 3. What are advantages of declaring constructors and destructor under public accessability?
- 4. Given the following C++ code, answer the questions (i) & (ii).

```
class TestMeOut
```

{

public:

```
~TestMeOut() //Function 1
```

{cout<<"Leaving the examination hall"<<endl;}

TestMeOut() //Function 2

{cout<<"Appearing for examination"<<endl;}

void MyWork() //Function 3

{cout<<"Attempting Questions//<<endl;}

};

(i) In Object Oriented Programming, what is Function 1 referred as and when doesit get invoked / called ?

(ii) In Object Oriented Programming, what is Function 2 referred as and when doesit get invoked / called ?

5. Write the output of the following C++ program code :

```
#include<iostream>
using namespace std;
class Calci
{
char Grade;
int Bonus;
public:
      Calci() {Grade='E'; Bonus=0;} //ascii value of A=65
void Down(int G)
{
      Grade-=G;
}
void Up(int G)
{
      Grade+=G;
      Bonus++;
}
void Show()
{
      cout<<Grade<<"#"<<Bonus<<endl;</pre>
      }
};
int main()
{
Calci c;
c.Down(3);
c.Show();
c.Up(7);
c.Show();
c.Down(2);
c.Show();
return 0;
```

```
}
```

PART IV

Answer to all the questions (5 Marks):

- 1. Explain nested class with example.
- 2. Mention the differences between constructor and destructor
- Define a class RESORT with the following description in C++ : Private members:

Rno // Data member to store room number

Name //Data member to store user name

Charges //Data member to store per day charge

Days //Data member to store the number of days

Compute () // A function to calculate total amount as Days * Charges and if the

//total amount exceeds 11000 then total amount is 1.02 * Days *Charges Public member:

getinfo() // Function to Read the information like name, room no, charges and days

dispinfo () // Function to display all entered details and total amount calculated //using COMPUTE function

```
4. Write the output of the following
```

#include<iostream>

#include<stdio.h>

using namespace std;

class sub

{

```
int day, subno;
public :
     sub(int,int); // prototype
void printsub()
     { cout<<" subject number : "<<subno;
     cout<<" Days : " <<day;
     }
};
sub::sub(int d=150,int sn=12)
```

```
{ cout<<endl<<"Constructing the object "<<endl;
       day=d;
       sub no=sn;
}
class stud
{
       int rno;
       float marks;
public:
       stud()
       { cout<<"Constructing the object of students "<<endl;
       rno=0;
       marks=0.0;
}
void getval()
{
       cout<<"Enter the roll number and the marks secured ";</pre>
       cin>>rno>>marks;
}
void printdet()
       { cout<<"Roll no : "<<rno<<"Marks : "<<marks<<endl;
       }
};
class addmission {
       sub obj;
       stud objone;
       float fees;
public :
       add mission ()
       { cout<< "Constructing the object of admission "<<endl;
       fees=0.0;
}
       void printdet( )
       { objone.printdet();
       obj.printsub( );
```

```
cout<<"fees : "<<fees<<endl ;</pre>
}
};
       int main()
       {system("cls");
       addmission adm;
       cout<<endl<< "Back in main ( )";</pre>
       return 0; }
Write the output of the following
#include<iostream>
#include<stdio.h>
       using namespace std;
class P
{ public:
       P()
       { cout<< "\nConstructor of class P "; }
       ~ P ( )
       { cout<< "\nDestructor of class P "; }
       };
       class Q
{ public:
       Q()
       { cout<<"\nConstructor of class Q "; }
       ~ Q()
```

{ cout<< "\nDestructor of class Q "; }
};</pre>

```
class R
{ P obj1, obj2;
Q obj3;
```

public:

5.

```
R ( )
{ cout<< "\nConstructor of class R ";}
~ R ( )
{ cout<< "\nDestructor of class R ";}
};
```

int main () { Ro R; Q oq; P op; return 0; }

Reference:

- (1) Object Oriented Programming with C++ (4th Edition), Dr. E. Balagurusamy, Mc.Graw Hills.
- (2) The Complete Reference C++ (Forth Edition), Herbert Schildt.Mc.Graw Hills.
- (3) Computer Science with C++ (A text book of CBSE XI and XII), SumitaArora, DhanpatRai& Co.
- (4) A text book of CBSE XI and XII computer science by PreetiArora and Pinky Gupta.
- (5) Computer Science with C++ Reeta shoo and Gagansahoo
- (6) The C++ Programming Language,BjarneStroustrup
- (7) C++ Primer (5th Edition) by S. B. Lippman, J. Lajoie.



Learning Objectives

After learning this chapter, the students will be able to

- Understand the purpose of overloading
- Construct C++ programs using function, constructor and operator overloading
- Execute and debug programs which contains the concept of polymorphism

15.1 Introduction

The word polymorphism means many forms (poly – many, morph – shapes) Polymorphism is the ability of a message or function to be displayed in more than one form. In C++, polymorphism is achieved through function overloading and operator overloading. The term overloading means a name having two or more distinct meanings. Thus an **'overloaded function'** refers to a function having more than one distinct meaning.

15.2 Function overloading

The ability of the function to process the message or data in more than one form is called as function overloading. In other words function overloading means two or more functions in the same scope share the same name but their parameters are different. In this situation, the functions that share the same name are said to be overloaded and the process is called function overloading. The number and types of a function's parameters are called the **function's signature**. When you call an overloaded function, the compiler determines the most appropriatedefinition to use, by comparing the argument types you have used to callthe function with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called **overload resolution**

15.2.1 Need For Function overloading

sometimes it's hard to find many different meaningful names for a single action.

Consider the situation to find the area of circle ,triangle and rectangle the following function prototype is given

float area_circle(float radius) // to calculate the area of a circle

float area_triangle(float half,floatbase,float height) // to calculate the area of a triangle

float area_rectangle (float length , float breadth) $\prime\prime$ to calculate the area of a rectangle

This can be rewritten using a single function header in the following manner

float area (float radius);

float area (float half, float base, float height);

float area (float length , float breadth);

```
Illustration 15.1 C++ Program to demonstrate function overloading
#include <iostream>
using namespace std;
void print(int i)
 {cout<< " It is integer " << i <<endl;}
void print(double f)
 { cout<< " It is float " << f <<endl;}
void print(string c)
{ cout<< " It is string " << c <<endl;}
int main() {
        print(10);
        print(10.10);
        print("Ten");
        return 0;
 }
Output:
It is integer 10
 It is float 10.1
It is string Ten
```

Tip Notes

Function overloading is not only implementing polymorphism but also reduces the number of comparisons in a program and makes the program to execute faster. It also helps the programmer by reducing the number of function names to be remembered.

15.2.2 Rules for function overloading

- 1. The overloaded function must differ in the number of its arguments or data types
- 2. The return type of overloaded functions are not considered for overloading same data type
- 3. The default arguments of overloaded functions are not considered as part of the parameter list in function overloading.

```
Illustration 15.2 C++ Program to demonstrate function overloading
#include <iostream>
using namespace std;
long add(long, long);
long add(long,long,long);
float add(float, float);
int main()
 {
 long a, b, c,d;
 float e, f, g;
 cout << "Enter three integers\n";</pre>
 cin >> a >> b>>c;
d=add(a,b,c); //number of arguments different but same data type
cout << "Sum of 3 integers: " << d << endl;</pre>
cout << "Enter two integers\n";</pre>
cin >> a >> b;
c = add(a, b); //two arguments data type same with above function call
                                      and different with below function call
cout << "Sum of 2 integers: " << c << endl;</pre>
cout << "Enter two floating point numbers\n";</pre>
 cin >> e >> f;
 g = add(e, f); //two arguments similar to the above function call but data
                                                                type different
cout << "Sum of floats: " << g << endl;</pre>
long add(long c, long g)
 {
 long sum;
 sum = c + g;
 return sum;
}
```

```
float add(float c, float g)
{
float sum;
sum = c + g;
return sum;
}
long add(long c, long g,long h)
{
long sum;
sum = c + g + h;
return sum;
}
Output
Enter three integers
345
Sum of 3 integers: 12
Enter two integers
46
Sum of 2 integers: 10
Enter two floating point numbers
2.1 3.1
Sum of floats: 5.2
```

Illustration 15.3 Default argument and return type are not considered in overloading

#include <iostream>

using namespace std;

long add(long, long);

long add(long long g=0);

int add(long,long); // [Error] ambiguating new declaration of 'int add(long int, long int)

```
int main()
{
long a, b, c;
int d;
cout<< "Enter two integers\n";</pre>
cin >> a >> b;
d=add(a,b); //arguments and datatype are same but return is different
cout<< "Sum of 2 integers: " << d <<endl;</pre>
cout<< "Enter two long integers\n";</pre>
cin >> a >> b;
c = add(a, b);
cout<< "Sum of 2 long integers: " << c <<endl;
cout<< "Enter a long integers\n";</pre>
cin >> a;
c = add(a); // [Error] ambiguating new declaration of 'int add(long int, long int)
cout<< "Sum of 2 long integers: " << c <<endl;</pre>
}
long add(long c, long g) // 'long int add(long int, long int)' previously defined here
{
long sum;
sum = c + g;
return sum;
}
int add(long c, long g) // [Error] ambiguating new declaration of 'int add(long int,
long int)
{
int sum;
sum = c + g;
return sum;
}
long add(long c, long g=20) // [Error] redefinition of 'long int add(long int, long
int)'
{
long sum;
sum = c + g;
return sum;
}
```

8 C	ompiler	(8) 🍓 Resources 🏦 Compile Log 🤗 Debug	🔂 Find Results 🖉 Close
Line	Col	File	Message
5	18	C:\TC\BIN\cpp\book\programs\load3.cpp	(Error) ambiguating new declaration of 'int add(long int, long int)'
1	6	C:\TC\BIN\cpp\book\programs\load3.cpp	[Note] old declaration 'long int add(long int, long int)'
		C:\TC\BIN\cpp\book\programs\load3.cpp	In function 'int add(long int, long int)':
35	23	C:\TC\BIN\cpp\book\programs\load3.cpp	[Error] ambiguating new declaration of 'int add(long int, long int)'
28	6	C:\TC\BIN\cpp\book\programs\load3.cpp	[Note] old declaration 'long int add(long int, long int)'
		C:\TC\BIN\cpp\book\programs\load3.cpp	In function 'long int add(long int, long int)':
13	6	C:\TC\BIN\cpp\book\programs\load3.cpp	(Error) redefinition of 'long int add(long int, long int)'
28	6	C:\TC\BIN\cpp\book\programs\load3.cpp	[Note] 'long int add(long int, long int)' previously defined here

15.3 Constructor overloading

Function overloading can be applied for constructors, as constructors are special functions of classes .A class can have more than one constructor with different signature. Constructor overloading provides flexibility of creating multiple type of objects for a class.

Illustration 15.4 constructor overloading				
#include <iostream></iostream>				
using namespace std;	Compiler identifies a given member function is			
class add	a constructor by its name and the return type.			
{				
int num1, num2, sum;				
public:				
add()				
{				
cout<<"\n Constructor without parameters ";				
num1= 0;				
num2= 0;	num2= 0;			
sum = 0;				
}				
add (int s1, int s2)	add (int s1, int s2)			
{	{			
cout<<"\n Parameterized	cout<<"\n Parameterized constructor ";			
num1= s1;				
num2=s2;				
sum=0;				
}				

```
add (add &a)
{
cout<<"\n Copy Constructor ... ";</pre>
num1 = a.num1;
num2=a.num2;
sum = 0;
}
void getdata()
{
cout<<"\nEnter data ... ";</pre>
cin>>num1>>num2;
}
void addition()
{
sum=num1+num2;
}
void putdata()
{
cout<<"\n The numbers are..";</pre>
cout<<num1<<'\t'<<num2;</pre>
cout<<"\n The sum of the numbers are.. "<< sum;</pre>
}
};
int main()
{
add a, b (10, 20), c(b);
a.getdata();
a.addition();
b.addition();
c.addition();
cout<<"\n Object a : ";</pre>
a.putdata();
cout<<"\n Object b : ";</pre>
b.putdata();
cout<<"\n Object c.. ";</pre>
c.putdata();
return 0;
}
```

Output

Constructor without parameters.. Parameterized constructor... Copy Constructor ... Enter data ... 20 30 Object a : The numbers are..20 30 The sum of the numbers are.. 50 Object b : The numbers are..10 20 The sum of the numbers are.. 30 Object c.. The numbers are..10 20 The sum of the numbers are.. 30

Note

Since, there are multiple constructors present, argument to the constructor should also be passed while creating an object.

Illustration 15.5 to find the perimeter of a rectangle using constructor overloading in a class.

```
// constructor declared as outline member function
#include<iostream>
using namespace std;
class Perimeter
{
    int l, b, p;
    public:
    Perimeter ();
    Perimeter (int);
    Perimeter (int,int);
    Perimeter (Perimeter&);
    void Calculate();
    };
    Perimeter :: Perimeter ()
```

```
{
cout<<"\n Enter the value of length and breath";</pre>
cin>>l>>b;
cout<<"\n\nNonParameterizd constructor ";</pre>
}
Perimeter :: Perimeter (int a)
{
l=b=a;
cout<<"\n\n Parameterizd constructor with one argument ";</pre>
Perimeter ::Perimeter (int l1, int b1)
cout<<"\n\n Parameterizd constructor with 2 argument ";</pre>
l=l1;
b=b1;
}
Perimeter ::Perimeter (Perimeter &p)
l=p.l;
b= p.b;
cout<<"\n\n copy constructor ";</pre>
}
void Perimeter ::Calculate(){
p = 2^{*}(l+b);
cout<<p;
}
int main ()
{
Perimeter Obj;
cout<<"\n perimeter of rectangle is ";</pre>
Obj. Calculate ();
Perimeter Obj1(2);
cout<<"\n perimeter of rectangle ";</pre>
Obj1.Calculate ();
Perimeter Obj2 (2, 3);
cout<<"\n perimeter of rectangle ";</pre>
Obj2.Calculate ();
Perimeter obj3 (Obj2);
cout<<"\n perimeter of rectangle ";</pre>
obj3.Calculate ();
return 0;
}
```

Output

Enter the value of length and breath 10 20 Non Parameterizd constructor perimeter of rectangle is 60 Parameterizd constructor with one argument perimeter of rectangle 8 Parameterizd constructor with 2 argument perimeter of rectangle 10 copy constructor perimeter of rectangle 10

15.4 Operator overloading

The term operator overloading, refers to giving additional

functionality to the normal C++ operators like +,++,-,—,+=,-=,*.<,>. It is also a type of polymorphism in which an operator is overloaded to give user defined meaning to it .

For example '+' operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc.

Almost all operators can be overloaded in C++. However there are few operator which can not be overloaded. Operator that are not overloaded are follows

- scope operator ::
- sizeof
- member selector •
- member pointer selector *
- ternary operator **?**:

Operator Overloading Syntax



15.4.1 Restrictions on Operator Overloading

Following are some restrictions to be kept in mind while implementing operator overloading.

- 1. Precedence and Associativity of an operator cannot be changed.
- 2. No new operators can be created, only existing operators can be overloaded.
- 3. Cannot redefine the meaning of an operator's procedure. You cannot change how integers are added.Only additional functions can be to an operator
- 4. Overloaded operators cannot have default arguments.
- 5. When binary operators are overloaded, the left hand object must be an object of the relevant class

```
Illustration 15.6 binary operator overloading using '+' and - symbol
```

```
//Complex number addition and subtraction
#include<iostream>
using namespace std;
class complex
{
    int real,img;
    public:
    void read()
    {
        cout<<"\nEnter the REAL PART : ";
        cin>>real;
        cout<<"\nEnter the IMAGINARY PART : ";
        cin>>img;
    }
}
```

```
complex operator +(complex c2)
{
  complex c3;
  c3.real=real+c2.real;
  c3.img=img+c2.img;
  return c3;
}
  complex operator -(complex c2)
{
  complex c3;
  c3.real=real-c2.real;
  c3.img=img-c2.img;
  return c3;
}
  void display()
{
  cout<<real<<"+"<<img<<"i";
}
};
int main()
{
  complex c1,c2,c3;
  int choice, cont;
  do
   {
  cout<<"\t\tCOMPLEX NUMBERS\n\n1.ADDITION\n\n2.SUBTRACTION\n\n";
  cout<<"\nEnter your choice : ";</pre>
  cin>>choice;
  if(choice==1||choice==2)
{
  cout<<"\n\nEnter the First Complex Number";</pre>
  c1.read();
  cout<<"\n\nEnter the Second Complex Number";</pre>
  c2.read();
}
  switch(choice)
{
  case 1 : c3=c1+c2; // binary + overloaded
  cout<<"\n\nSUM = ";</pre>
      c3.display();
      break;
```

```
case 2 : c3=c1-c2; // binary -overloaded
cout<<"\n\nResult = ";</pre>
      c3.display();
      break;
default : cout<<"\n\nUndefined Choice";</pre>
}
cout<<"\n\nDo You Want to Continue?(1-Y,0-N)";
cin>>cont;
}while(cont==1);
return 0;
}
Output
       COMPLEX NUMBERS
1.ADDITION
2.SUBTRACTION
Enter your choice : 1
Enter the First Complex Number
Enter the REAL PART: 3
Enter the IMAGINARY PART: 4
Enter the Second Complex Number
Enter the REAL PART : 5
Enter the IMAGINARY PART : 8
SUM = 8 + 12i
Do You Want to Continue?(1-Y,0-N)1
        COMPLEX NUMBERS
1.ADDITION
2.SUBTRACTION
Enter your choice : 2
Enter the First Complex Number
Enter the REAL PART: 8
Enter the IMAGINARY PART: 10
Enter the Second Complex Number
Enter the REAL PART: 4
Enter the IMAGINARY PART : 5
Result = 4+5i
Do You Want to Continue?(1-Y,0-N)0
```

```
Illustration 15.7 concatenation of string using operator overloading
 #include<string.h>
 #include<iostream>
 using namespace std;
 class strings
 {
        public:
        char s[20];
        void getstring(char str[])
 {
        strcpy(s,str);
   }
        void operator+(strings);
 };
 void strings::operator+(strings ob)
 {
        strcat(s,ob.s);
        cout<<"\nConcatnated String is:"<<s;</pre>
 }
 int main()
 {
        strings ob1, ob2;
        char string1[10], string2[10];
        cout<<"\nEnter First String:";</pre>
        cin>>string1;
        ob1.getstring(string1);
        cout<<"\nEnter Second String:";</pre>
        cin>>string2;
        ob2.getstring(string2);
        //Calling + operator to Join/Concatenate strings
        ob1+ob2;
        return 0;
 }
 Output
 Enter First String:COMPUTER
 Enter Second String:SCIENCE
```

Concatenated String is: COMPUTERSCIENCE

Points to Remember:

- n C++, polymorphism is achieved through function overloading and operator overloading.
- The term overloading means a name having two or more distinct meanings.
- Overloaded function' refers to a function having more than one distinct meaning.
- Overloaded functions have same name but different signatures (Number of argument and type of argument)
- A function's argument list is known as a function signature
- Two function cannot be overloaded when the only difference is that one takes a reference parameter and the other takes a normal, call-by-value parameter.
- Ordinary functions as well member functions can be overloaded

- A class can have overloaded constructorswhereasdestructor function cannot be overloaded.
- The mechanism of giving special meaning to an operator is known as operator overloading.
- Operator overloading provides new definitions for most of the C++ operators.
- Even user defined types (objects) can be overloaded.
- The definition of the overloaded operator is given using the keyword 'operator' followed by an operator symbol.
- We can overload all the C++ operators except the following:
- Class member access operator (., .*) ,Scope resolution operator (::), Size operator (sizeof) andConditional operator (?:)



Evaluation



PART I

Choose the correct answers

- 1. Which of the following refers to a function having more than one distinct meaning?
 - (A) Function Overloading
 - (C) Operator overloading (D) Operations overloading
- 2. Which of the following reduces the number of comparisons in a program ?
 - (A) Operator overloading

(C) Function Overloading

- (B) Operations overloading
- (D) Member overloading

(B) Member overloading

```
3. void dispchar(char ch='$',int size=10)
```

```
{
  for(int i=1;i<=size;i++)
  cout<<ch;
}</pre>
```

How will you invoke the function dispchar() for the following input?

To print \$ for 10 times

(A) dispchar();	<pre>(B) dispchar(ch,size);</pre>
(C) dispchar(\$,10);	(D)dispchar('\$',10 times);

4. Which of the following is not true with respect to function overloading?

- (A) The overloaded functions must differ in their signature.
- (B) The return type is also considered for overloading a function.
- (C) The default arguments of overloaded functions are notconsidered for Overloading.
- (D) Destructor function cannot be overloaded.
- 5. Which of the following is invalid prototype for function overloading
 - (A) void fun (intx);

void fun (char ch);

(B) void fun (intx);

void fun (inty);

- (C) void fun (double d);void fun (char ch);
- (D) void fun (double d);void fun (inty);
- 6. Which of the following function(s) combination cannot be considered as overloaded function(s) in the given snippet ?
 void print(char A,int B); // F1
 void printprint(int A, float B); // F2
 void Print(int P=10); // F 3
 void print(); // F4
 (A) F1,F2,F3,F4
 (B) F1,F2,F3
 (C) F1,F2,F4
 (D) F1,F3,F4

7. Which of the following operator is by default overloaded by the compiler?

```
(A) *
             (B) +
                           (C) +=
                                         (D) ==
Based on the following program answer the questions (8) to (10)
#include<iostream>
using namespace std;
class Point {
private:
int x, y;
public:
Point(int x1,int y1)
 {
       x=x1;y=y1;
 }
void operator+(Point &pt3);
void show() {cout << "x = " << x << ", y = " << y; }
};
void Point::operator+(Point &pt3)
{
  x += pt3.x;
  y += pt3.y;
}
int main()
{
 Point pt1(3,2),pt2(5,4);
 pt1+pt2;
 pt1.show();
 return 0;
}
```

8. Which of the following operator is overloaded?

(A) + (B) operator (C) :: (D) =

9. Which of the following statement invoke operator overloading?

(A) pt1+pt2; (B) Point pt1(3,2),pt2(5,4);

(C) pt1.show(); (D) return 0;

10. What is the output for the above program?

(A) x=8, y=6 (B) x=14, y=14 (C) x=8, y=6 (D) = x=5, y=9

PART II

Answer to all the questions (2 Marks):

- 1. What is function overloading?
- 2. List the operators that cannot be overloaded.
- 3. class add{int x; public: add(int)}; Write an outline definition for the constructor.
- 4. Does the return type of a function help in overloading a function?
- 5. What is the use of overloading a function?

PART III

Answer to all the questions (3 Marks):

- 1. What are the rules for function overloading?
- 2. How does a compiler decide as to which function should be invoked when there are many functions? Give an example.
- 3. What is operator overloading? Give some example of operators which can be overloaded.
- 4. Discuss the benefit of constructor overloading ?
- 5. class sale (int cost, discount ;public: sale(sale &); Write a non inline definition for constructor specified;

```
Answer to all the questions (5 Marks):
```

- 1. What are the rules for operator overloading?
- 2. Answer the question (i) to (v) after going through the following class.

```
classBook
{
intBookCode; char Bookname[20];float fees;
public:
Book()
                //Function 1
{
fees=1000;
BookCode=1;
strcpy (Bookname,"C++");
}
void display(float C) //Function 2
{
cout<<BookCode<<":"<<Bookname<<":"<<fees<<endl;
}
~Book()
             //Function 3
{
cout<<"End of Book Object"<<endl;</pre>
}
     Book (intSC, char S[], float F);
                                   //Function 4
};
```

(i) In the above program, what are Function 1 and Function 4 combined together referred as?

- (ii) Which concept is illustrated by Function3? When is this function called/ invoked?
- (iii) What is the use of Function3?
- (iv) Write the statements in main to invoke function1 and function2
- (v) Write the definition for Function4 .

```
Write the output of the following program
3.
      include<iostream>
      using namespace std;
      class Seminar
      {
      int Time;
      public:
      Seminar()
       {
          Time=30;cout<<"Seminar starts now"<<endl;
        }
      void Lecture()
       {
      cout<<"Lectures in the seminar on"<<endl;
      }
      Seminar(int Duration)
      {
        Time=Duration;cout<<"Welcome to Seminar "<<endl;
      }
      Seminar(Seminar &D)
      {
         Time=D.Time;cout<<"Recap of Previous Seminar Content "<<endl;</pre>
      }
      ~Seminar()
      {
      cout<<"Vote of thanks"<<endl;</pre>
      }
      };
```

```
int main()
      {
             Seminar s1,s2(2),s3(s2);
             s1.Lecture();
             return 0;
       }
4.
      Debug the following program
       #include<iostream>
       using namespace std;
      class String
      {
      public:
      charstr[20];
      public:
       void accept_string
                {
      cout<<"\n Enter String
                                     : ";
       cin>>str;
                }
       display_string()
                {
       cout<<str;
                }
                String operator *(String x) //Concatenating String
                {
                    String s;
      strcat(str,str);
      strcpy(s.str,str);
```

```
goto s;
         }
}
int main()
{
    String str1, str2, str3;
    str1.accept_string();
    str2.accept_string();
cout<<"\n\n First String is
                                 : ";
    str1=display_string();
cout<<"\n\n Second String is
                                   : ";
    str2.display_string();
    str3=str1+str2;
cout>>"\n\n Concatenated String is : ";
    str3.display_string();
return 0;
}
Answer the questions based on the following program
#include<iostream>
#include<string.h>
using namespace std;
class comp {
public:
chars[10];
void getstring(char str[10])
  {
strcpy(s,str);
  }
```

5.

```
void operator==(comp);
};
void comp::operator==(comp ob)
{
if(strcmp(s,ob.s)==0)
cout<<"\nStrings are Equal";</pre>
else
cout<<"\nStrings are not Equal";</pre>
}
int main()
{
comp ob, ob1;
char string1[10], string2[10];
cout<<"Enter First String:";</pre>
cin>>string1;
ob.getstring(string1);
cout<<"\nEnter Second String:";</pre>
cin>>string2;
ob1.getstring(string2);
ob == ob1;
return 0;
}
```

(i) Mention the objects which will have the scope till the end of the program.

(ii) Name the object which gets destroyed in between the program

(iii)Name the operator which is over loaded and write the statement that invokes it.

(iv) Write out the prototype of the overloaded member function

(v)What types of operands are used for the overloaded operator?

(vi) Which constructor will get executed? Write the output of the program

278

CASE STUDY

Suppose you have a Kitty Bank with an initial amount of Rs500 and you have to add some more amount to it. Create a class 'Deposit' with a data member named 'amount' with an initial value of Rs500. Now make three constructors of this class as follows:

- 1. without any parameter no amount will be added to the Kitty Bank
- 2. having a parameter which is the amount that will be added to the Kitty Bank
- 3. whenever amount is added an additional equaly amount will be deposited automatically

Create an object of the 'Deposit' and display the final amount in the Kitty Bank.

Reference:

- 1. Object Oriented Programming with C++ (4th Edition), Dr. E. Balagurusamy, Mc.Graw Hills.
- 2. The Complete Reference C++ (Forth Edition), Herbert Schildt. Mc.Graw Hills.
- 3. Computer Science with C++ (A text book of CBSE XI and XII), SumitaArora, DhanpatRai& Co.
- 4. A text book of CBSE XI and XII computer science by PreetiArora and Pinky Gupta.
- 5. Computer Science with C++ Reeta shoo and Gagansahoo
- 6. The C++ Programming Language,BjarneStroustrup
- 7. C++ Primer (5th Edition) by S. B. Lippman, J. Lajoie

Unit IV

Object Oriented Programming with C++

CHAPTER 16

Inheritance

Learning Objectives

After the completion of this chapter, the student will be able to

- Understand the purpose of Inheritance
- Construct C++ programs using Inheritance



• Execute and debug programs which contains the concept of Inheritance

16.1 Introduction to Inheritance

Inheritance is one of the most important features of Object Oriented Programming is Inheritance.In object-oriented programming, inheritance enables new class and its objects to take on the properties of the existing classes. A class that is used as the basis forinheritance is called a superclass or base class. A class that inherits from a superclass is called a subclass or derived class

16.2 Need for Inheritance

Inheritance is an important feature of object oriented programming used for code reusability. It is a process of creating new classes called derived classes, from the existing or base classes. Inheritance allows us to inherit all the code (except declared as private) of one class to another class. The class to be inherited is called base class or parent class and the class which inherits the other class is called derived class or child class. The derived class is a power packed class, as it can add additional attributes and methods and thus enhance its functionality.



16.3 Types of Inheritance

There are different types of inheritance viz., Single Inheritance, Multiple inheritance, Multilevel inheritance, hybrid inheritance and hierarchical inheritance.

1. Single Inheritance

When a derived class inherits only from one base class, it is known as single inheritance

2. Multiple Inheritance

When a derived class inherits from multiple base classes it is known as multiple inheritance

3. Hierarchical inheritance

When more than one derived classes are created from a single base class, it is known as Hierarchical inheritance.

4. Multilevel Inheritance

The transitive nature of inheritance is itself reflected by this form of inheritance. When a class is derived from a class which is a derived class – then it is referred to as multilevel inheritance.

5. Hybrid inheritance

When there is a combination of more than one type of inheritance, it is known as hybrid inheritance. Hence, it may be a combination of Multilevel and Multiple inheritance or Hierarchical and Multilevel inheritance or Hierarchical, Multilevel and Multiple inheritance.

The following diagram represents the different types of inheritance



16.4 Derived Class and Base class

While defining a derived class, the derived class should identify the class from which it is derived. The following points should be observed for defining the derived class.

- i The keyword class has to be used
- ii The name of the derived class is to be given after thekeyword class

- iii A single colon
- iv The type of derivation (the visibility mode), namely private, public or protected. If no visibility mode is specified, then by default the visibility mode is considered as private.
- The names of all base classes(parent classes) separated by comma. v

class derived_class_name :visibility_mode base_class_name

{

// members of derivedclass

};

The following are some of the examples for different forms of inheritance

16.4.1 Single Inheritance

Illustration 16.1 single inheritance

```
# include <iostream>
                                 Though the derived class inherits
using namespace std;
                                 all the members of base class, it has
class student
                  //base class
                                 access privilege only to non-private
{
                                 members of the base class.
private :
char name[20];
int rno;
public:
void acceptname()
{
cout<<"\n Enter roll no and name .. ";</pre>
cin>>rno>>name;
void displayname()
cout<<"\n Roll no :-"<<rno;</pre>
cout<<"\n Name :-"<<name<<endl;</pre>
}
};
```


```
class exam : public student
                                   //derived class with single base class
{
public:
       int mark1, mark2, mark3, mark4, mark5, mark6, total;
 void acceptmark()
 {
  cout<<"\n Enter lang,eng,phy,che,csc,mat marks.. ";</pre>
   cin>>mark1>>mark2>>mark3>>mark4>>mark5>>mark6;
}
void displaymark()
cout<<"\n\t\t Marks Obtained ";</pre>
cout<<"\n Language.. "<<mark1;</pre>
cout<<"\n English .. "<<mark2;</pre>
cout<<"\n Physics .. "<<mark3;</pre>
cout<<"\n Chemistry.. "<<mark4;</pre>
cout<<"\n Comp.sci.. "<<mark5;</pre>
cout<<"\n Maths .. "<<mark6;
}
};
int main()
 exam e1;
 e1.acceptname();
                        //calling base class function using derived class
                                                                    object
e1.acceptmark();
e1.displayname();
                         //calling base class function using derived class
                                                                    object
e1.displaymark();
return 0;
}
```

Output

Enter roll no and name .. 1201 KANNAN Enter lang,eng,phy,che,csc,mat marks.. 100 100 100 100 100 100 Roll no :-1201 Name :-KANNAN Marks Obtained Language.. 100 English .. 100 Physics .. 100 Chemistry.. 100 Comp.sci.. 100

In the above program the derived class "exam" inherits all the members of the base class "student". But it has access privilege only to the non private members of the base class.

16.4.2 Multiple Inheritance

Program to illustrate Multiple inheritance



```
;
}
};
class detail
                     //Base class
       int dd,mm,yy;
       char cl[4];
public:
void acceptdob()
cout<<"\n Enter date,month,year in digits and class .. ";
cin>>dd>>mm>>yy>>cl;
}
void displaydob()
{
cout<<"\n class:-"<<cl;</pre>
cout<<"\t\t DOB : "<<dd<<" - "<<mm<<" - " <<yy<<endl;
}
};
class exam : public student, public detail //derived class with multiple
base class
{
   public:
  int mark1, mark2, mark3, mark4, mark5, mark6, total;
 void acceptmark()
  {
   cout<<"\n Enter lang,eng,phy,che,csc,mat marks.. ";</pre>
   cin>>mark1>>mark2>>mark3>>mark4>>mark5>>mark6;}
void displaymark()
{
cout<<"\n\t\t Marks Obtained ";</pre>
cout<<"\n Language.. "<<mark1;</pre>
cout<<"\n English .. "<<mark2;</pre>
cout<<"\n Physics .. "<<mark3;</pre>
cout<<"\n Chemistry.. "<<mark4;</pre>
cout<<"\n Comp.sci.. "<<mark5;</pre>
cout<<"\n Maths .. "<<mark6;
}
};
```

```
int main()
{
 exam e1:
el.acceptname();
                       //calling base class function using derived class
                                                                  object
e1.acceptdob();
                        //calling base class function using derived class
                                                                  object
e1.acceptmark();
e1.displayname();
                        //calling base class function using derived class
                                                                  object
e1.displaydob();
                        //calling base class function using derived class
                                                                  object
e1.displaymark();
return 0;
}
Output:
Enter roll no and name .. 1201 MEENA
Enter date, month, year in digits and class .. 7 12 2001 XII
Enter lang,eng,phy,che,csc,mat marks.. 96 98 100 100 100 100
Roll no :-1201
Name :- MEENA
class
         :-XII
                      DOB
                              : 7 - 12 - 2001
         Marks Obtained
Language.. 96
English .. 98
Physics .. 100
Chemistry.. 100
Comp.sci.. 100
Maths .. 100
```

In the above program the class "exam" is derived from class "student" and "detail"

Hence it access all the members of both the classes.

Notes

In multiple inheritance the base classes do not have any relationship between them. However the derived class acquires all the properties of both the classes

16.4.3 Multilevel Inheritance

```
Illustration 16.3 single inheritance
# include <iostream>
using namespace std;
class student
                      //base class
{
                                  In multilevel inheritance a derived
private :
                                  class itself acts as a base class to derive
char name<sup>[20]</sup>;
                                  another class.
int rno;
public:
void acceptname()
cout<<"\n Enter roll no and name .. ";
cin>>rno>>name;
                                                        B
}
void displayname()
cout<<"\n Roll no :-"<<rno;
cout<<"\n Name :-"<<name<<endl;</pre>
                                                Multilevel Inheritance
}};
                                    //derived class with single base class
 class exam : public student
{
       public:
        int mark1, mark2, mark3, mark4, mark5, mark6;
        void acceptmark()
        ł
        cout<<"\n Enter lang,eng,phy,che,csc,mat marks.. ";</pre>
        cin>>mark1>>mark2>>mark3>>mark4>>mark5>>mark6;
        }
void displaymark(){
cout<<"\n\t\t Marks Obtained ";</pre>
cout<<"\n Language... "<<mark1;</pre>
cout<<"\n English... "<<mark2;</pre>
cout<<"\n Physics... "<<mark3;</pre>
cout<<"\n Chemistry... "<<mark4;</pre>
cout<<"\n Comp.sci... "<<mark5;</pre>
cout<<"\n Maths... "<<mark6;
}
 };
```

```
class result : public exam
{
       int total;
       public:
              void showresult()
              {
                     total=mark1+mark2+mark3+mark4+mark5+mark6;
                     cout<<"\nTOTAL MARK SCORED : "<<total;</pre>
       }
};
int main()
{
       result r1;
r1.acceptname();
                    //calling base class function using derived class object
r1.acceptmark();
                    //calling base class function which itself is a derived
                    // class function using its derived class object
r1.displayname();
                    //calling base class function using derived class
                    //object
r1.displaymark();
                    //calling base class function which itself is a derived
                    //class function using its derived class object
r1.showresult();
                    //calling the child class function
return 0;
}
Output:
Enter roll no and name .. 1201 SARATHI
Enter lang, eng, phy, che, csc, mat marks.. 96 98 100 100 100 100
Roll no :-1201
Name :-SARATHI
         Marks Obtained
Language... 96
English... 98
Physics... 100
Chemistry... 100
Comp.sci... 100
Maths... 100
TOTAL MARK SCORED : 594
```

In the above program class "result " is derived from class "exam" which itself is derived from class student.



```
void displaymark()
{
cout<<"\n\t\t Marks Obtained in quarterly";</pre>
cout<<"\n Language.. "<<mark1;</pre>
cout<<"\n English .. "<<mark2;</pre>
cout<<"\n Physics .. "<<mark3;</pre>
cout<<"\n Chemistry.. "<<mark4;</pre>
cout<<"\n Comp.sci.. "<<mark5;</pre>
cout<<"\n Maths .. "<<mark6;
}
};
 class hexam : public student //derived class with single base class
{
  public:
       int mark1, mark2, mark3, mark4, mark5, mark6;
 void acceptmark()
  ł
   cout<<"\n Enter lang,eng,phy,che,csc,mat marks for halfyearly exam.. ";
   cin>>mark1>>mark2>>mark3>>mark4>>mark5>>mark6;
 }
void displaymark()
{
cout<<"\n\t\t Marks Obtained in Halfyearly";</pre>
cout<<"\n Language.. "<<mark1;</pre>
cout<<"\n English .. "<<mark2;</pre>
cout<<"\n Physics .. "<<mark3;</pre>
cout<<"\n Chemistry.. "<<mark4;</pre>
cout<<"\n Comp.sci.. "<<mark5;</pre>
cout<<"\n Maths .. "<<mark6;
}
};
```

```
int main()
```

```
{
```

```
qexam q1;hexam h1;q1.acceptname();//calling base class function using derived class objectq1.acceptmark();//calling base class functionh1.acceptname();//calling base class function using derived class objecth1.displayname();//calling base class function using derived class objecth1.acceptmark();h1.displaymark();//calling base class function using its//derived class object
```

return 0;

}

Output

Enter roll no and name .. 1201 KANNAN Enter lang, eng, phy, che, csc, mat marks for quarterly exam.. 95 96 100 98 100 99 Roll no :-1201 Name :-KANNAN Marks Obtained in quarterly Language.. 95 English .. 96 Physics .. 100 Chemistry.. 98 Comp.sci.. 100 Maths.. 99 Enter roll no and name .. 1201 KANNAN Enter lang, eng, phy, che, csc, mat marks for halfyearly exam.. 96 98 100 100 100 100 Roll no :-1201 Name :- KANNAN Marks Obtained in Halfyearly Language.. 96 English .. 98 Physics .. 100 Chemistry.. 100 Comp.sci.. 100 Maths .. 100

In the above program the class "qexam" and "hexam" are derived from class "student". Here for single base class more than one derived class.So this comes under hierarchical inheritance.





```
void displaymark() {
cout<<"\n\t\t Marks Obtained ";</pre>
cout<<"\n Language.. "<<mark1;</pre>
cout<<"\n English .. "<<mark2;</pre>
cout<<"\n Physics .. "<<mark3;</pre>
cout<<"\n Chemistry.. "<<mark4;</pre>
cout<<"\n Comp.sci.. "<<mark5;</pre>
cout<<"\n Maths .. "<<mark6;
}
};
class detail
                   //base classs 2
{
 int dd,mm,yy;
 char cl[4];
public:
 void acceptdob()
{
cout<<"\n Enter date,month,year in digits and class .. ";</pre>
cin>>dd>>mm>>yy>>cl;
}
void displaydob()
{
cout<<"\n class :-"<<cl;
cout<<"\t\t DOB : "<<dd<<" - "<<mm<<" -" <<yy<<endl;
}
};
class result : public exam, public detail //inherits from exam, which itself is a
                                   //derived class and also from class detail
{
       int total;
       public:
              void showresult()
       {
              total=mark1+mark2+mark3+mark4+mark5+mark6;
              cout<<"\nTOTAL MARK SCORED : "<<total;</pre>
       }
};
```

```
class detail
                    //base classs 2
{
 int dd,mm,yy;
 char cl[4];
public:
 void acceptdob()
{
cout<<"\n Enter date,month,year in digits and class .. ";
cin>>dd>>mm>>yy>>cl;
void displaydob()
{
cout<<"\n class
                  :-"<<cl;
cout<<"\t\t DOB : "<<dd<<" - "<<mm<<" -" <<yy<<endl;
}
};
class result : public exam, public detail //inherits from exam , which itself is a
                                  //derived class and also from class detail
{
       int total:
       public:
              void showresult()
       {
              total=mark1+mark2+mark3+mark4+mark5+mark6;
              cout<<"\nTOTAL MARK SCORED : "<<total;</pre>
       }
};
int main()
{
 result r1;
r1.acceptname();
                     //calling base class function using derived class object
r1.acceptmark();
                     //calling base class which itsel is a derived class function
using its derived class object
```

```
int main()
```

```
{
```

result r1;

```
r1.acceptname(); //calling base class function using derived class object
r1.acceptmark(); //calling base class which itsel is a derived class function
using its derived class object
```

r1.acceptdob();

```
cout<<"\n\n\t\t MARKS STATEMENT";</pre>
```

```
r1.displayname(); //calling base class function using derived class object
r1.displaydob();
```

```
r1.displaymark(); //calling base class which itsel is a derived class function using its derived class object
```

r1.showresult(); //calling the child class function

return 0;

}

Output:

```
Enter roll no and name .. 1201 RAGU
```

Enter lang, eng, phy, che, csc, mat marks.. 96 98 100 100 100 100

```
Enter date, month, year in digits and class .. 7 12 2001 XII
```

MARKS STATEMENT

```
Roll no :-1201
Name :-RAGU
```

```
class :-XII DOB : 7 - 12 -2001
```

Marks Obtained

```
Language.. 96
```

English .. 98

```
Physics .. 100
```

Chemistry.. 100

```
Comp.sci.. 100
Maths .. 100
```

TOTAL MARK SCORED : 594

In the above program the derived class "result" has acquired the properties of class "detail" and class "exam" which is derived from "student". So this inheritance is a combination of multi level and multiple inheritance and so it is called hybrid inheritance

16.5 VISIBILITY MODES

An important feature of Inheritance is to know which member of the base class will be acquired by the derived class. This is done by using visibility modes.

The accessibility of base class by the derived class is controlled by visibility modes. The three visibility modes are private, protected and public. The default visibility mode is private. Though visibility modes and access specifiers look similar, the main difference between them is Access specifiers control the accessibility of the members with in the class where as visibility modes control the access of inherited members with in the class.

16.5.1 Private visibility mode

When a base class is inherited with private visibility mode the public and protected members of the base class become 'private' members of the derived class



16.5.2 protected visibility mode

When a base class is inherited with protected visibility mode the protected and public members of the base class become 'protected members ' of the derived class



16.5.3 public visibility mode

When a base class is inherited with public visibility mode, the protected members of the base class will be inherited as protected members of the derived class and the public members of the base class will be inherited as public members of the derived class.

BASE CLASS	when inherited with public visibility	DERIVED CLASS
private members	r	private members
protected members	→	protected members
public members	become >>	public members
	I	Tin Notes

When classes are inherited with public, protected or private the private members of the base class are notinherited they are only visible i.e continue to exist in derived classes, and cannot be accessed

Illustration 16.6 explains the significance of different visibility modes.

```
//Implementation of Single Inheritance using public visibility mode
#include <iostream>
using namespace std;
class Shape
{
    private:
        int count;
    protected:
        int width;
        int height;
public:
        void setWidth(int w)
        {
            width = w;
            }
```

```
void setHeight(int h)
{
height = h;
ł
};
class Rectangle: publicShape
{
public:
int getArea()
{
return (width * height);
}
};
int main()
{
Rectangle Rect;
Rect.setWidth(5);
Rect.setHeight(7);
// Print the area of theobject.
cout<< "Total area: "<<Rect.getArea() <<endl;</pre>
return 0;
}
Output
Total area: 35
```

The following table contain the members defined inside each class before inheritance

MEMDEDS of door	visibility modes			
MEMDERS OF Class	Private	Protected	Public	
Shape(base class)	int count;	int width; int height;	<pre>void setWidth(int) void setHeight(int)</pre>	
Rectangle (derived class only with its defined members)			intgetArea();	

The following table contain the details of members defined after inheritance

MEMBERS of class	visibility modes –public for acquiring the properties of the base class		
	Private	Protected	Public
Shape(base class)	int count;	int width; int height;	<pre>void setWidth(int) void setHeight(int)</pre>
Rectangle (derived class acquired the properties of base class with public visibility)	Private members of base classes are not directly accessible by the derived class	int width; int height;	int getArea(); void setWidth(int) void setHeight(int)

Suppose the class **rectangle is derived with protected visibility** then the properties of class rectangle will change as follows

MEMBERS of class	visibility modes –protected for acquiring the properties of the base class		
	Private	Protected	Public
Shape(base class)	int count;	int width; int height;	<pre>void setWidth(int) void setHeight(int)</pre>
Rectangle (derived class acquired the properties of base class with protected visibility)	Private members of base classes are not directly accessible by the derived class	int width; int height; void setWidth(int) void setHeight(int)	intgetArea();

In case the class rectangle is derived with private visibility mode from its base class shape then the property of class rectangle will change as follows

MEMBERS of class	visibility modes –private for acquiring the properties of the base class		
	Private	Protected	Public
Shape(base class)	int count;	int width; int height;	<pre>void setWidth(int) void setHeight(int)</pre>
Rectangle (derived class acquired the properties of base class with private visibility)	Private members of base classes are not directly accessible by the derived class	int width; int height; void setWidth(int) void setHeight(int)	int getArea();

When you derive the class from an existing base class, it may inherit the properties of the base class based on its visibility mode. So one must give appropriate visibility mode depends up on the need.

Private inheritance should be used when you want the features of the base class to be available to the derived class but not to the classes that are derived from the derived class.

Protected inheritance should be used when features of base class to be available only to the derived class members but not to the outside world.

Public inheritance can be used when features of base class to be available the derived class members and also to the outside world.

16.6 Inheritance and constructors and destructors

When an object of the derived class is created ,the compiler first call the base class constructor and then the constructor of the derived class. This because the derived class is built up on the members of the base class. When the object of a derived class expires first the derived class destructor is invoked followed by the base class destructor.

```
Illustration 16.7 The order of constructors and destructors
#include<iostream>
using namespace std;
class base
 {
public:
base()
 {
cout<<"\nConstructor of base class...";
 }
~base()
cout<<"\nDestructor of base class.... ";
}
 };
class derived:public base
public :
derived()
```

```
{
cout << "\nConstructor of derived ...";</pre>
}
~derived()
{
cout << "\nDestructor of derived ...";</pre>
}
};
class derived1 :public derived
{
public :
derived1()
{
cout << "\nConstructor of derived1 ...";</pre>
}
~derived1()
{
cout << "\nDestructor of derived1 ...";</pre>
}
};
int main()
{
derived1 x;
return 0;
}
Output:
Constructor of base class...
Constructor of derived ...
Constructor of derived1 ...
Destructor of derived1 ...
Destructor of derived ...
Destructor of base class....
```

The constructors are executed in the order of inherited class i.e., from base constructor to derived. The destructors are executed in the reverse order.

Some Facts About the execution of constructor in inheritance

- Base class constructors are executed first ,before the derived class constructors execution
- Derived class can not inherit the base class constructor but it can call the base class constructor by using

Base_class name::base_class_constructor() in derived class definition

- If there are multiple base classes ,then its start executing from the left most base class
- In multilevel inheritance, the constructors will be executed in the order of inheritance.

size of derived class object=size of all base class data members + size of all data members in derived class.

16.7 Inheritance and Access Control

When you declare a derived class, a visibility mode can precede each base class in the base list of the derived class. This does not alter the access attributes of the individual members of a base class, but allows the derived class to access the members of a base class with restriction.

As you already know, you can derive classes using any of the three visibility mode:

- In a public base class, public and protected members of the base class remain public and protected members of the derived class.
- In a protected base class, public and protected members of the base class are protected members of the derived class.
- In a private base class, public and protected members of the base class become private members of the derived class.

In all these cases, **private members of the base class remain private and cannot be used by the derived class**. However it can be indirectly accessed by the derived class using the public or protected member function of the base class since they have the access privilege for the private members of the base class.

```
Illustration 16.8 the private member of base class is accessed by the
derived class
#include<iostream>
using namespace std;
class add
{
int num1,sum;//private data member can't be derived by derived class
protected:
 int num2;
public:
add()
{
 num1 = num2 = sum=0;
cout<<"\n Add constructor .. ";</pre>
}
accept()
 ł
cout<<"\n Enter two numbers .. ";</pre>
cin>>num1>>num2;
}
plus()
 {
sum = num1 + num2;
cout<<"\n The sum of two numbers is .. "<< sum;</pre>
}
int difference(){ return num1-num2;} //return an integer value
 \simadd()
 {cout<<endl<<"Add destructor"; }
};
class subtract :public add
                                       //subtract derived from add
{
int sub;
public:
subtract()
{
```

```
sub = 0;
cout<<"\n Subtract constructor .. ";</pre>
}
minus()
{
 accept(); // member function of base class can access its datamember
 sub= difference();
cout<<"\n The difference of two numbers are ... "<< sub;</pre>
}
~subtract()
{cout<<endl<<"Subtract destructor"; }</pre>
};
int main()
{
                             A member function can call another member
subtract s;
                             function without dot operator and object
int choice = 0;
cout<<"\n Enter your choice ";</pre>
cout<<" \n1. Add..\n2. Subtract ..";</pre>
cin>>choice;
switch( choice )
{
case 1:
s.accept();
s.plus();
break;
case 2:
s.minus();
break;
}
return 0;
}
```

Output:

Add constructor .. Subtract constructor .. Enter your choice 1. Add.. 2. Subtract .. 2 Enter two numbers .. 20 10 The difference of two numbers are ... 10 Subtract destructor Add destructor

In the above program the data member num1 which is under private visibility in add class cannot be inherited by the subtract class.But subtract class can inherit the protected and public members of add class. Hence using the inherited accept() function the value of num1 is accepted and using the inherited difference() function the values are subtracted and passed to the subtract class.

16.7.1 Access control in publicly derived class

From a publicly derived class, public and protected members of the base class remain public and protected members of the derived class. The public members can accessed by the object of the derived class similar to its own members in public.

```
Hustration 16.9 the publicly inherited derived class
#include<iostream>
#include<string>
using namespace std;
class Employee
{
    private:
        char name[50];
        int code;
    public:
        void getinfo();
        void dispinfo();
};
class staff: public Employee
```

```
{
       private:
       int ex;
       public:
       void getdata();
       void display();
};
void Employee::getinfo()
{
       cout<<"Name:";</pre>
       gets(name);
       cout<<"Code:";</pre>
       cin>>code;
}
void Employee::dispinfo()
{
       cout<<"Name:"<<name<<endl;</pre>
       cout<<"Code:"<<code<<endl;</pre>
}
void staff::getdata()
{
       cout<<"Experience:";</pre>
       cin>>ex;
}
void staff::display()
{
  cout<<"Experience:"<<ex<<" Years"<<endl;</pre>
}
```

```
int main()
{
      staff s:
      cout<<"Enter data"<<endl;
      s.getinfo();
                         //derived member function
                          //defined member function
      s.getdata();
      cout<<endl<<"\tDisplay Data"<<endl;</pre>
                          //derived member function
      s.dispinfo();
      s.display();//defined member function
return 0;
}
Output:
Enter data
Name:USHA
Code:1201
Experience: 30
    Display Data
Name:USHA
Code:1201
Experience:30 Years
```

In the above program since "staff" is derived publicly even the derived function can be accessed by the object of the class.



16.7.2 Access control in privately derived class

From a privately derived class, public and protected members of the base class become private members of the derived class. Hence it is not possible to access the derived members using the object of the derived class. The Derived members are invoked by calling it from the publicly defined members

Illustration 16.10 the privately inherited derived class

```
#include<iostream>
#include<string>
using namespace std;
class Employee
{
  private:
       char name[50];
       int code;
  public:
       void getinfo();
       void dispinfo();
};
class staff: private Employee
{
  private:
       int ex;
  public:
       void getdata();
       void display();
};
void Employee::getinfo()
{
       cout<<"Name:";</pre>
       gets(name);
       cout<<"Code:";</pre>
       cin>>code;
}
void Employee::dispinfo()
{
       cout<<"Name:"<<name<<endl;</pre>
       cout<<"Code:"<<code<<endl;</pre>
}
```

```
void staff::getdata()
{
                    //invoked inside
  getinfo();
cout<<"Experience:";</pre>
  cin>>ex;
}
void staff::display()
               //member function called inside another member function
 dispinfo();
cout<<"Experience:"<<ex<<" Years"<<endl;
}
int main()
{
  staff s:
  cout<<"Enter data"<<endl;
      s.getdata();
  cout<<endl<<"\tDisplay Data"<<endl;
      s.display();
  return 0;
}
Output:
Enter data
Name: BALAMURUGAN
Code: 1201
Experience: 30
    Display Data
Name: BALAMURUGAN
Code: 1201
Experience: 30 Years
```

If you look at the output of publicly derived class and privately derived class are same the way of defining is different. This is because the private members of the derived class cannot be accessed by its object. In the above program since staff is privately derived getinfo() and dispinfo() become private to class "staff". To access both the member function they are invoked inside the publ; ic member functions getdata() and display() respectively.



16.8 Overriding / Shadowing Base class functions in derived class

In case of inheritance there are situations where the member function of the base class and derived classes have the same name. If the derived class object calls the overloaded member function it leads confusion to the compiler as to which function is to be invoked. The derived class member function have higher priority than the base class member function. This shadows the member function of the base class which has the same name like the member function of the derived class. The scope resolution operator resolves this problem.

```
Illustration 16.11 the use of scope resolution operator in derived class
#include<iostream>
#include<string>
using namespace std;
class Employee
{
  private:
    char name[50];
    int code:
  public:
    void getdata();
    void display();
};
class staff: public Employee
{
  private:
    int ex;
  public:
       void getdata();
    void display();
};
void Employee::display()
{
       cout<<"Name:"<<name<<endl;
       cout<<"Code:"<<code<<endl;
}
```

```
void staff::display()
{
       Employee :: display();//overriding
       cout<<"Experience:"<<ex<<" Years"<<endl;</pre>
}
int main()
{
       staff s:
       cout<<"Enter data"<<endl;
       s.getdata();
       cout<<endl<<endl<<"\tDisplay Data"<<endl;</pre>
       s.display();
       return 0;
}
Output
Enter data
Name: SUGANYA
Code: 1201
Experience: 30
    Display Data
Name: SUGANYA
Code:1201
Experience:30 Years
```

In the above program getdata() and display() are defined both in base and in derived class. So when the derived class staff inherits the properties of Employee class it will have two getdata() and display() each. To differentiate the derived getdata() and display() from the defined getdata() and display() :: (scope resolution) operator is given along with the base class name to the base class members

When a derived class member function has the same name as that of its base class member function ,the derived class member function shadows/hides the base class's inherited function .This situation is called function overriding and this can be resolved by giving the base class name followed by :: and the member function name.

16.8.1 thisPointer

'this' pointer is a constant pointer that holds the memory address of the current object. It identifies the currently calling object. It is useful when the argument variable name in the member function and the data member name are same. To identify the datamember it will be given as this->data member name

```
Illutration 16.8.1 illustrates the use of this pointer
#include<iostream>
using namespace std;
class T
   public:
int x;
   void foo()
   {
              // same as this->x = 6;
     x = 6;
     this->x = 5; // explicit use of this->
     cout<<endl<<x<<" "<<this->x:
   }
   void foo(int x) // parameter x shadows the member with the same name
                             // unqualified x refers to the parameter.'this->'
      this-x = x;
required for disambiguation
     cout<<endl<<x<<" "<<this->x;
  }};
```

```
int main()
{
    T t1,t2;
    t1.foo();
    t2.foo();
    }

Ouput
5 5
5 5
------
Process exited after 0.1 seconds with return value 0
```

```
Press any key to continue . . .
```

Another Example Program 16.8.1A using this pointer

```
#include <iostream>
using namespace std;
```

```
class Container {
 public:
   // Constructor definition
     Container(double l = 2.0, double b = 2.0, double h = 2.0) {
     cout<<"Constructor called." << endl;</pre>
     length = l;
     breadth = b;
     height = h;
   }
   double Volume() {
   return length * breadth * height;
   }
   int compare(Container container)
   {
   return this->Volume() >Container.Volume();
   }
```

```
private:
   double length; // Length of a Container
   double breadth; // Breadth of a Container
   double height; // Height of a Container
};
int main(void) {
 Container Container1(3.3, 1.2, 1.5); // Declare Container1
 Container Container2(8.5, 6.0, 2.0); // Declare Container2
 if(Container1.compare(Container2)) {
   cout << "Container2 is smaller than Container1" << endl;
 } else {
   cout << "Container2 is equal to or larger than Container1" << endl;
 }
   return 0;
}
Output
Constructor called.
Constructor called.
Container2 is equal to or larger than Container1
_____
Process exited after 0.09358 seconds with return value 0
Press any key to continue . . .
```

Points to Remember:

- The mechanism of deriving new class from an existing class is called inheritance.
- The main advantage of Inheritance is it supports reusability of code.
- The derived class inherits all the properties of the base class. It is a power packed class, as it can add additional attributes and methods and thus enhance its functionality.
- The various types of Inheritance are Single inheritance, multiple inheritance, hierarchical inheritance and hybrid inheritance
- When a derived class inherits only from one base class, it isknown as single inheritance
- When a derived class inherits from multiple base classes it is known as multiple inheritance
- When a class is derived from a class which is a derived class itself – then this is referred to as multilevel inheritance. The transitive nature of inheritance is reflected by this form of inheritance.
- When more than one derived classes are created from a single base class, it is known as Hierarchical inheritance.
- When there is a combination of more than one type of inheritance, it is known as hybrid inheritance.
- In multiple inheritance, the base classes are constructed in the order in which they appear in the declaration of the derived class.
- A sub-class can derive itself publicly, privately or protectedly.

- The private member of a class cannot be inherited .
- In publicly derived class, the public members of the base class remain public and protected members of base class remain protected in derived class.
- In privately derived class, the public and the protected members of the base class become protected in derived class
- In publicly derived class, the public members of the base class remain public
- and protected members of base class remain protected in derived class.
- When class is derived in protected mode, the public and protected members of base class become protected in derived class.
- constructors and destructors of the base class are not inherited but during the creation of an object for derived class the constructors of base class will automatically invoked.
- The destructors are invoked in reverse order .The destructors of the derived classes are invoked first and then the base class.
- size of derived class object=size of all base class data members + size of all data members in derived class
- overriding of the members are resolved by using :: Scope resolution operator.
- this pointer used to refer the current objects members



PART I



Choose the correct answers

1.	Which of the follow	wingis the process of cr	eating new classes from an existing class		
	(a) Polymorphism	(b) Inheritance	(c) Encapsulation	(d) super class	
2.	Which of the following derives a class student from the base class school				
	(a) school: student		(b) class student : p	ublic school	
	(c) student : public	school	(d) class school : pu	ıblic student	
3.	The type of inherit	ance that reflects the tr	ansitive nature is		
	(A) Single Inherita	nce	(B) Multiple Inher	itance	
	(C) Multilevel Inh	eritance	(D) Hybrid Inherit	tance	
4.	Which visibility m be available to the class?	ode should be used w derived class but not t	hen you want the fea o the classes that are	tures of the base class to derived from the derived	
	(A) Private	(B) Public	(C) Protected	(D) All of these	
5.	Inheritance is process of creating new class from				
	(A) Base class	(B) abstract	(C) derived class	(D) Function	
6.	A class is derived f	rom a class which is a c	derived class itself, the	en this is referred to as	
	(A) multiple inher	itance	(B) multilevel inher	ritance	
	(C) single inheritat	nce	(D) double inherita	ince	
7.	Which amongst th	e following is executed	in the order of inher	itance?	
	(A) Destructor	(B) Member function	(C) Constructor	(D) Object	
8.	Which of the follow	wing is true with respe	ct to inheritance?		
	(A) Private membe	ers of base class are inh	erited to the derived	class with private	
	(B) Private member accessibility	ers of base class are n	ot inherited to the d	erived class with private	
	(C) Public member	rs of base class are inhe	erited but not visible t	to the derived class	

(D) Protected members of base class are inherited but not visible to the outsideclass

9. Based on the following class declaration answer the questions (from 9.1 o 9.5)

```
class vehicle
{ int wheels;
public:
void input_data(float,float);
void output_data();
protected:
int passenger;
};
class heavy_vehicle : protected vehicle {
int diesel_petrol;
protected:
int load;
protected:
int load;
public:
voidread_data(float,float)
voidwrite_data(); };
class bus: private heavy_vehicle {
charTicket[20];
public:
voidfetch_data(char);
voiddisplay_data(); };
};
```

9.1. Which is the base class of the class heavy_vehicle?

(a) Bus (b) heavy_vehicle (c) vehicle (d) both (a) and (c)

- 9.2. The data member that can be accessed from the function displaydata()
 - (a) passenger (b) load (c) Ticket (d) All of these

9.3. The member function that can be accessed by an objects of bus Class is

(a) input_data(),	(b) read_data() ,output_data()write_data()
(c) fetch_data()	(d) All of these display_data()

9.4. The member function that is inherited as public by Class Bus

(a) input_data() ,	(b) read_data(), output_data()write_data()

(c) fetch_data() (d) All of these display_data()

10.

class x { int a; public : x()			
<pre>{} }; class y { x x1; public : y(){} }; class z : public y,x { int b; public: z(){} }z1;</pre>			

What is the order of constructor for object z1 to be invoked?

(A) z , y,x,x	(B) x,y,z,x	(c) y,x,x,z	(D) x,y,z
---------------	-------------	-------------	-----------

PART II

Answer to the all qustions (2 Marks):

- 1. What is inheritance?
- 2. What is a base class?
- 3. Why derived class is called power packed class?
- 4. In what multilevel and multiple inheritance differ though both contains many base class?
- 5. What is the difference between public and private visibility mode?

PART III

Answer to the all questions (3 Marks):

- 1. What are the points to be noted while deriving a new class?
- 2. What is difference between the members present in the private visibility mode and the members present in the public visibility mode
- 3. What is the difference between polymorphism and inheritance though are usedfor reusability of code?
- 4. What do you mean by overriding?
- 5. Write some facts about the execution of constructors and destructors in inheritance

PART IV

Answer to the all questions (5 Marks):

- 1. Explain the different types of inheritance
- 2. Explain the different visibility mode through pictorial representation

3.

```
#include<iostream>
#include<string.h>
#include<stdio.h>
using name spacestd;
class publisher
{
    char pname[15];
    char hoffice[15];
    char address[25];
    double turnover;
    protected:
    char phone[3][10];
    void register();
```

```
public:
publisher();
~publisher();
void enter data();
void disp data();
};
class branch
{
charbcity[15];
char baddress[25];
protected:
intno_of_emp;
public:
charbphone[2][10];
branch();
~branch();
void have data();
void give data();
};
class author: public branch, publisher
{
intaut_code;
charaname[20];
float income;
public:
author();
~author();
voidgetdata();
voidputdata();
};
```

Answer the following questions based on the above given program:

- 3.1. Which type of Inheritance is shown in the program?
- 3.2. Specify the visibility mode of base classes.
- 3.3 Give the sequence of Constructor/Destructor Invocation when object of class author is created.
- 3.4. Name the base class(/es) and derived class (/es).
- 3.5 Give number of bytes to be occupied by the object of the following class:

```
(a) publisher (b) branch (c) author
```

- 3.6. Write the names of data members accessible from the object of class author.
- 3.7. Write the names of all member functions accessible from the object of class author.
- 3.8 Write the names of all members accessible from member functions of class author.

4. Consider the following c++ code and answer the questions

class Personal { int Class, Rno; char Section; protected: char Name[20]; public: personal(); void pentry(); voidPdisplay(); }; class Marks:private Personal { float $M{5};$ protected: char Grade[5]; public: Marks(); void M entry(); void M display(); }; class Result:public Marks { float Total, Agg; public: char FinalGrade, Commence[20]; Result(); void R calculate(); void R display(); }:

- 4.1. Which type of Inheritance is shown in the program?
- 4.2. Specify the visibility mode of base classes.
- 4.3 Give the sequence of Constructor/Destructor Invocation when object of class Result is created.
- 4.4. Name the base class(/es) and derived class (/es).

4.5 Give number of bytes to be occupied by the object of the following class:

(a) Personal (b) Marks (c) Result

- 4.6. Write the names of data members accessible from the object of class Result.
- 4.7. Write the names of all member functions accessible from the object of class Result.
- 4.8 Write the names of all members accessible from member functions of class Result.

5. Write the output of the following program

```
#include<iostream>
using namespace std;
class A
{
protected:
int x;
public:
void show()
{
cout<<"x = "<<x<<endl;
}
A()
{
     cout<<endl<<" I am class A "<<endl;
}
~A()
{
     cout<<endl<<" Bye ";</pre>
}
};
class B : public A
{
```

```
{
protected:
int y;
public:
B(int x, int y)
{
this->x = x; //this -> is used to denote the objects datamember
this->y = y;
                  //this -> is used to denote the objects datamember
}
B()
{
     cout<<endl<<" I am class B "<<endl;
}
~B()
{
     cout<<endl<<" Bye ";</pre>
}
void show()
{
cout<<"x = "<<x<<endl;
cout<<"y = "<<y<<endl;
}
};
int main()
{
A objA;
B objB(30, 20);
objB.show();
return 0;
}
```

6. Debug the following program

```
Output
_____
15
14
13
Program :
_____
%include(iostream.h)
#include<conio.h>
Class A
{
public;
int a1,a2:a3;
Void getdata[]
{
      a1=15;
      a2=13;a3=13;
}
}
Class B:: public A()
{
     PUBLIC
     voidfunc()
     {
      int b1:b2:b3;
      A::getdata[];
      b1=a1;
      b2=a2;
      a3=a3;
      cout<<b1<<'\t'<<b2<<'t\'<<b3;
     }
void main()
{
     clrscr()
     B der;
     der1:func();
     getch();
}
```

CASE STUDY

All the banks operating in India are controlled by RBI. RBI has set certain guidelines (e.g. minimum interest rate, minimum balance allowed, maximum withdrawal limit etc) for all banks to follow. For example, suppose RBI has set minimum interest rate applicable to a saving bank account to be 5% annually; however, banks are free to use 5% interest rate or to set any rates above it.

Write a program to implement bank functionality in the above scenario. Note: Create few classes namely Customer, Account, RBI (Base Class) and few derived classes (SBI, ICICI, PNB,IOBetc). Assume and implement required member variables and functions in each class.

Hint:

```
class Customer
{
//Personal Details ...
// Few functions ...
class Account
{
// Account Detail ...
// Few functions ...
}
class RBI
Customer c; //hasA relationship
Account a; //hasA relationship
Public double GetInterestRate() { }
Public double GetWithdrawalLimit() {
                                        }
class SBI: public RBI
{
//Use RBI functionality or define own functionality.
class ICICI: public RBI
//Use RBI functionality or define own functionality.
class IOB: public RBI
//Use RBI functionality or define own functionality.
}
```

CASE STUDY 2

Write a class for a class Stock

- 1. Each Stock has a data member which holds the net price, and a constructor which sets this price.
 - Each Stock has a method get_Price(), which calculates and returns the gross price (the gross price includes VAT at 21%)
- 2. Write 2 classes which inherit from the general Stock class, of type Notebook and Book
 - The gross price for Notebook includestax at 21%,
 - Books are free of tax, so the gross price is unchanged from the net price, and you will need to re-define the getGrossPrice method in this class
- 3. Write a program which does the following:
 - a. Declare an array of 10 objects to Stock
 - b. Declare aobject to a Book and an object to Notebook
 - c. Ask the user to enter details of the book, and of the Notebook item,.
 - d. Check your method getGrossPrice works correctly with each type and then displaythe result

Reference:

- (1) Object Oriented Programming with C++ (4th Edition), Dr. E. Balagurusamy, Mc.Graw Hills.
- (2) The Complete Reference C++ (Forth Edition), Herbert Schildt. Mc.Graw Hills.
- (3) Computer Science with C++ (A text book of CBSE XI and XII), SumitaArora, DhanpatRai& Co.
- (4) A text book of CBSE XI and XII computer science by PreetiArora and Pinky Gupta.
- (5) Computer Science with C++ Reeta shoo and Gagansahoo
- (6) The C++ Programming Language,BjarneStroustrup
- (7) C++ Primer (5th Edition) by S. B. Lippman, J. Lajoie

Computer Ethics And Cyber Security

CHAPTER 7

Learning Objectives

After learning this chapter, the students will be able to

- To know about cyber-crimes.
- To understand the guidelines and need for ethics in cyber-world.
- To understand issues related to cyber-crimes.
- To know the functionality of firewalls and proxy servers.
- To learn aboutencryption and decryption.
- To gain knowledge on the IT Act.

17.1 INTRODUCTION

Internet is a communication media which is easily accessible and open to all. Information Technology iswidespread through computers, mobile phones and internet. There is a lot of scope and possibility for misuse of Information Technology.

Computer systems in general are vulnerable. They play an important role in the daily lives of individuals and businesses. Special care must be taken explicitly in order to ensure that the valuable data do not get into wrong hands. Hence, the data need to be protected.

A cyber-crime is a crime which involves computer and network. This is becoming a growing threat to society and is caused by criminals or irresponsible action of individuals who are exploiting the widespread use of Internet. It presents a major challenge to the ethical use of information technologies. Cyber-crime also poses threats to the integrity, safety and survival of most business systems.

Figure. 17.1 presents the types of cyber-crimes that happen across the world.





Figure 17.1 Types of cyber - crimes

ETHICS

Ethicsmeans "WhatiswrongandWhat is Right". It is a set of moral principles that rule the behavior of individuals who use computers. An individual gains knowledge to follow the right behavior, using morals that are also known as ethics. Morals refer to the generally accepted standards of right and wrong in the society. Similarly, in cyberworld, there are certain standards such as

- Do not use pirated software
- Do not use unauthorized user accounts
- Do not steal others' passwords
- Do not hack

The core issues in computer ethics are based on the scenarios arising from the use of internet such as privacy, publication of copyrighted content, unauthorized distribution of digital content and user interaction with web sites, software and related services.

COMPUTER ETHICS

With the help of internet, world has now become a global village. Internet has

been proven to be a boon to individuals as well as various organizations and businesses. e-Commerce is becoming very popular among businesses as it helps them to reach a wide range of customers faster than any other means.

Computer ethics deals with the procedures, values and practices that govern the process of consuming computer technology and its related disciplines without damaging or violating the moral values and beliefs of any individual, organization or entity.

GUIDELINES OF ETHICS

Generally, the following guidelines should be observed by computer users:

- **1. Honesty:** Users should be truthful while using the internet.
- **2. Confidentiality:** Usersshould not share any important information with unauthorized people.
- **3. Respect:** Each user should respect the privacy of other users.

- 4. Professionalism: Each user should maintain professional conduct.
- 5. Obey The Law: Users should strictly obey the cyber law in computer usage.
- 6. Responsibility: Each user should take ownership and responsibility for their actions

Ethics is a set of moral principles that govern the behavior of an individual in a society, and Computer ethics is set of moral principles that regulate the use of computers by users.

17.2 ETHICAL ISSUES

An Ethical issue is a problem or issue that requires a person or organization to choose between alternatives that must be evaluated as right (ethical) or wrong (unethical). These issues must be addressed and resolved to have a positive influence in society.

Some of the common ethical issues are listed below:

- Cyber crime
- Software Piracy
- Unauthorized Access
- Hacking
- Use of computers to commit fraud
- Sabotage in the form of viruses
- Making false claims using computers

CYBER CRIME

Cybercrime is an intellectual, white-collar crime. Those who commit such crimes generally manipulate the computer system in an intelligent manner.

For example – illegal money transferviainternet.

Examples of some Computer crimes and their functions are listed below in Table 17.1:

Table 17.1	Computer	Crime
------------	----------	-------

Crime	Function
Crime Function	Hacking, threats, and blackmailing towards a business or a person.
Cyber stalking	Harassing through online.
Malware	Malicious programs that can perform a variety of functions including stealing, encrypting or deleting sensitive data, altering or hijacking core computing functions and monitoring user's computer activity without their permission.
Denial of service attack	Overloading a system with fake requests so that it cannot serve normal legitimate requests.
Fraud	Manipulating data, for example changing the banking records to transfer money to an unauthorized account.
Harvesting	A person or program collects login and passwordinformation from a legitimate user to illegally gainaccess to others' account(s).
Identity theft	It is a crime where the criminals impersonate individuals, usually for financial gain.
Intellectual property theft	Stealing practical or conceptual information developed by another person or company.
Salami slicing	Stealing tiny amounts of money from each transaction.
Scam	Tricking people into believing something that isnot true.
Spam	Distribute unwanted e-mail to a large number ofinternet users.
Spoofing	It is a malicious practice in which communication is send from unknown source disguised as a source known to the receiver.

SOFTWARE PIRACY

Software Piracy is about the copyright violation of software created originally by an individual or an institution. It includes stealing of codes / programs and other information illegally and creating duplicate copies by unauthorized means and utilizing this data either for one's own benefit or for commercial profit.

In simple words,Software Piracy is "unauthorized copying of software". **Figure 17.2** shows a diagrammatical representation of software piracy.



Figure 17.2- Diagrammatic representation of Software piracy

Most of the commercial software is licensed for use at a single computer site or for use by only one user at any time. When a user buys any software, he becomes a licensed user for that software. He is allowed to make copies of the program for backup purposes, but it is against the law to distribute duplicate copies to others. Such illegal copying and distribution of commercial software should not be practiced.

An entirely different approach to software piracy is called shareware, acknowledges the futility of trying to stop people from copying software and instead relies on people's honesty. Shareware publishers encourage users to give copies of programs to friends and colleagues but ask everyone who uses that program regularly to pay a registration fee to the program's author directly. Commercial programs that are made available to the public illegally are often called warez.

UNAUTHORIZED ACCESS

Unauthorized access is when someone gains access to a website, program, server, service, or other system by breaking into a legitimate user account. For example, if someone tries guessing a password or username for an account that was not theirs until they gained access, it is considered an unauthorized access.

To prevent unauthorized access, Firewalls, **Intrusion Detection Systems** (IDS), Virus and Content Scanners, Patches and Hot fixes are used.

HACKING

Hacking is intruding into a computer system to steal personal data without the owner's permission or knowledge (liketo steal a password). It is also gaining unauthorized access to a computer system, and altering its contents. It may be done in pursuit of a criminal activity or it may be a hobby. Hacking may be harmless if the hacker is only enjoying the challenge of breaking systems' defenses, but such ethical hacking should be practiced only as controlled experiments. **Figure 17.4** shows a diagrammatic representation of Hacking.



Figure 17.3 Diagramatic representation of Hacking

CRACKING

Cracking is where someone edits a program source so that the code can be exploited or modified. A cracker (also called a black hat or dark side hacker) is a malicious or criminal hacker. "Cracking" means trying to get into computer systems in order to steal, corrupt, or illegitimately view data.

A cracker is someone who breaks into someone else's computer system, often on a network, bypassing passwords or licenses in computer programs.

Software cracking is the most often used type of cracking which is nothing but removing the encoded copy protection. There is another type of cracking called password cracking. This is mainly used to crack the passwords. Password cracking can be perform either by using an automated program or can be manually realized.

One more interesting fact about cracking is social engineering. It is a method of getting passwords and information using human weakness. These crackers trick people, not software. They can use just the phone for getting information, they can pretend being your friend and talk to you on **Internet Relay Chat**(IRC) or by Instant messenger. e-mail can also be a source for them. They may send official e-mail requesting some sensitive information. It may look like a legitimate e-mail from bank or other official institution.

The other method that uses social engineering crackers is password guessing. They find your personal information from some personal data/facts and try to guess a password.

Usually a cracker maintains knowledge of the vulnerabilities he or she finds and exploits them for personal advantage, not revealing them to either to the general public or to the manufacturer.

17.3 Cyber Security and Threats

Cyber attacks are launched primarily for causing significant damage to a computer system or for stealing important information from an individual or from an organization.Cyber security is a collection of various technologies, processes and measures that reduces the risk of cyber attacks and protects organizations and individuals from computer based threats.

TYPES OF CYBER ATTACKS

Malware is a type of software designed through which the criminals gain illegal access to software and cause damage. Various types of cyber-attacks and their functions are given in **Table 17.2**.

S.No.	Cyber Attack	Function
		A virus is a small piece of computer code that can repeat itself
		and spreads from one computer to another by attaching itself
		to another computer file. One of the most common virus is
1.	Virus	Trojan.
		A Trojan virus is a program that appears to perform one
		function (for example, virus removal) but actually performs
	malicious activity when executed.	
		Worms are self- repeating and do not require a computer
2	Mone	program to attach themselves. Worms continually look for
Ζ.	vvorins	vulnerabilities and report back to the author of the worm when
	weaknesses are discovered.	
		Spyware can be installed on the computer automatically
3.	Spyware	when the attachments are open, by clicking on links or by
	downloading infected software.	
4. Ransomware	Ransomware is a type of malicious program that demands	
	payment after launching a cyber-attack on a computer system.	
	Kallsolliwale	This type of malware has become increasingly popular among
		criminals and costs the organizations millions each year.

Table 17.2 – 0	Cyber	Attacks	and	Functions

Cyber Security Threats

In recent years, most of the individuals and enterprises are facing problems due to the weaknesses inherent in security systems and compromised organizational infrastructures. Different types of Cyber Security Threats are categorized as below:

Social engineering

A misuse of an individual's weakness, achieved by making them to click malicious links, or by physically accessing the computer through tricks. Phishing and pharming are examples of social engineering.

Phishing

Phishing is a type of computer crime used to attack, steal user data, including login name, password and credit card numbers. It occurs when an attacker targets a victim into opening an e-mailor an instant text message. The attacker uses phishing to distribute malicious links or attachments that can perform a variety of functions, including the extraction of sensitive login credentials from victims.



Figure 17.4 Diagrammatic representation of Phishing

Pharming

Pharming is a scamming practice in which malicious code is installed on a personal computer or server, misdirecting users to fraudulent web sites without their knowledge or permission. Pharming has been called "phishing without a trap". It is another way hackers attempt to manipulate users on the Internet. It is a cyber-attack intended to redirect a website's traffic to a fake site.



Figure 17.5 Diagrammatic representation of Pharming

Man In The Middle (MITM)

Man-in-the-middle attack (MITM; also Janus attack) is an attack where the attacker secretly relays and possibly alters the communication between two parties who believe they are directly communicating with each other.

Example: Suppose Alice wishes to communicate with Bob. Meanwhile, Mallory wishes to intercept the conversation to overhear and optionally to deliver a false message to Bob.



Figure 17.6 - An illustration of the man-inthe-middle attack

Cookies

A cookie (also called HTTP cookie, web cookie, Internet cookie, browser cookie, or simply cookie) is a small piece of data sent from a website and stored on the user's computer memory (Hard drive) by the user's web browser while the user is browsing internet. Cookies were designed to be a reliable mechanism for websites to remember stateful information (such as items added in the shopping cart in an online store) or to record the user's browsing activity (including clicking particular buttons, logging in etc.). They can also be used to remember arbitrary pieces of information that the user previously entered into form fields such as names, addresses, passwords, and credit card numbers. From the security point of view, if cookie data is not encrypted, any anonymous user (hacker) can access the cookie information and misuse it.

Web sites typically use cookies for the following reasons:

- To collect demographic information about who has visited the Web site.
- Sites often use this information to track how often visitors come to the site and how long they remain on the site.
- It helps to personalize the user's experience on the Web site.
- Cookies can help store personal information about users so that when a usersubsequently returns to the site, a more personalized experience is provided.

If you ever returned to a site and have seen your name mysteriously appear on the screen, it is because on a previous visit, you gave your name to the site and it was stored in a cookie. A good example of this is the way some online shopping sites will make recommendations to users based on their previous purchases. It helps to monitor advertisements. Cookies do not act maliciously on computer system. They are merely text files that can be deleted at any time.

Cookies cannot be used to spread viruses and they cannot access your hard drive. However, any personal information that you provide to a Web site, including credit card information, will most likely be stored in a cookie unless the cookie feature is explicitly turned off in your browser. This is the way in which cookies threaten privacy.

Firewall and Proxy Servers

A firewall is a computer network security based system that monitors and controls incoming and outgoing network traffic based on predefined security rules. A firewall commonly establishes a block between a trusted internal computer network and entrusted computer outside the network. They are generally categorized as network-based or host-based. Network based firewalls are positioned on the gateway computers of LANs [local area Network], WANs [Wide Area Network] and intranets. Host-based firewalls are positioned on the network node itself. The host-based firewall may be a service as a part of the operating system or an agent application such as endpoint security or protection. Each has advantages and disadvantages. However, each has a role in layered security. Firewalls also vary in type depending on where communication originates, where it is intercepted, and the state of communication being traced. Figure 17.7 shows the working of firewall server.

A proxy server acts as an intermediary between the endusers and a web server. A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resources available from a different server. The proxy server examines the request, checks authenticity and grants the request based on that. Proxy servers typically keep the frequently visited site addresses in its cache which leads to improved response time.

Figure 17.8 shows the working of a proxy server.



Figure 17.7 Firewall Server



Figure 17.8 Working of Proxy server

Encryption and Decryption

Encryption and decryption are processes that ensure confidentiality that only authorized persons can access the information.

Encryption is the process of translating the plain text data (plaintext) into random and mangled data (called cipher-text).

Decryption is the reverse process of converting the cipher-text back to plaintext. Encryption and decryption are done by cryptography. In cryptography a key is a piece of information (parameter) that determines the functional output of a cryptographic algorithm.

Figure 17.9 shows the encryption and decryption process.



Figure 17.9 Encryption and Decryption

Encryption has been used by militaries and governments to facilitate secret communication. It is now commonly used in protecting information within many kinds of civilian systems. It is also used to protect data in communication system, for example data being transferred via networks (e.g. the Internet, ecommerce), mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. There have been numerous reports of data in communication being intercepted in recent years. Data should also be encrypted when transmitted across networks in order to protect against the network traffic by unauthorized users.

TYPES OF ENCRYPTION

There are two types of encryption schemes as listed below:

- Symmetric Key encryption
- Public Key encryption

SYMMETRIC KEY ENCRYPTION

Symmetric encryption is a technique to use the same key for both encryption and decryption. The main disadvantage of the symmetric key encryption is that all authorized persons involved, have to exchange the key used to encrypt the data before they can decrypt it. If anybody intercepts the key information, they may read all message. **Figure 17.10** depicts the working of symmetric key encryption.



Figure 17.10 Symmetric key encryption

PUBLIC KEY ENCRYPTION

Public key encryption is also called Asymmetric encryption. It uses the concept of a key value pair, a different key is used for the encryption and decryption process. One of the keys is typically known as the private key and the other is known as the public key.

The private key is kept secret by the owner and the public key is either shared amongst authorized recipients or made available to the public at large.

The data encrypted with the recipient's public key can only be decrypted with the corresponding private key. **Figure 17.11** shows the public key encryption.



Figure 17.11 Public key encryption

Asymmetric Encryption in Digital Certificates:

A digital certificate in a client-server model of communication is one of the example of Asymmetric Encryption. A certificate is a package of information that identifies a user and a server. It contains information such as an organization's name, the organization that issued the certificate, the users' email address and country, and user's public key.

When a server and a client require a secure encrypted communication, they send a query over the network to the other party, which sends back a copy of the certificate. The other party's public key can be extracted from the certificate. A certificate can also be used to uniquely identify the holder.

Digital Signature

Digital signatures are based on asymmetric cryptography and can provide assurances of evidence to origin, identity and status of an electronic document, transaction or message, as well as acknowledging informed by the signer.

To create a digital signature, signing software (email) creates a one-way hash of the electronic data to be signed. The user's private key to encrypt the hash, returning a value that is unique to the hashed data. The encrypted hash, along with other information such as the hashing algorithm, forms the digital signature. Any change in the data, even to a single bit, results in a different hash value. This attribute enables others to validate the integrity of the data by using the signer's public key to decrypt the hash. If the decrypted hash matches a second computed hash of the same data, it proves that the data hasn't changed since it was signed. If the two hashes don't match, the data has either been tampered with in some way (indicating a failure of integrity) or the signature was created with a private key that doesn't correspond to the public key presented by the signer (indicating a failure of authentication). **Figure 17.12** shows the function of a digital signature.



Figure 17.12 – Function of Digital Signature

17.4 INTRODUCTION TO INFORMATION TECHNOLOGY ACT

In the 21stcentury, Computer, Internet and ICT or e-revolution has changed the life style of the people. Today paper based communication has been substituted by e-communication. Accordingly we have new terminologies like cyber world, e-transaction, e-banking, e-return and e-contracts. Apart from positive side of e-revolution there is also negative side of computer, that is, the internet and ICT in the hands of criminals which has become a weapon of offence. Accordingly a new panel of members emerged to tackle the problems of cyber crimes in cyber space i.e. Cyber Law or Cyber Space Law or Information Technology Law or Internet Law.

In India Cyber law and IT Act 2000, modified in 2008 are being articulated to prevent computer crimes. IT Act 2000 is an act to provide legal recognition for transactions carried out by means of **ElectronicData Interchange(EDI**) and other means of electronic communication. It is the primary law in India dealing with cybercrime and electronic commerce(e-Commerce). e-Commerce is electronic data exchange or electronic filing of information.



Cyber law or Internet law is a term that encapsulates the legal issues related to use of the Internet.

PREVENTION

25% of cyber crime remains unsolved. To protect the information the following points to be noted:

- Complex password setting can make your surfing secured.
- When the internet is not in use, disconnect it.
- Do NOT open spam mail or emails that have an unfamiliar sender.
- When using anti-virus software, keep it up-to-date.

YOU

Awareness is the key to security.

- Information security is the immune system in the body of business.
- A check that does not bounce is called the Security Check. Do it every day before you leave!
- Do Your Part Be Security Smart !!!
- Don't be Quick to Click... be wary when you shop online.
- Restart is Smart job
- Passwords are like toothbrushes. They are best when new and should never be shared.
- When you and your system part away, your system should be first off for the day.
- Your mind is a storage room of information, keep the door locked.
- _ a _ _word is not a PaSSword without Protect, Save and Secure!
- Link Link stop neglect....Think Think before connect.....

Evaluation



PART - I

Choose the best Answer.

- 1.Which of the following deals with procedures, practices and values?a. piracyb. programsc. virusd. computer ethics
- 2. Commercial programs made available to the public illegally are known as a. freeware b. warez c. free software d. software
- 3. Which one of the following are self-repeating and do not require a computer program to attach themselves?

a. viruses b. worms c. spyware d. Trojans

4. Which one of the following tracks a user visits a website?a. spyware b. cookies c. worms d. Trojans

- 5. Which of the following is not a malicious program on computer systems? a. worms d. Trojans c. spyware d. cookies
- A computer network security that monitors and controls incoming and outgoing traffic is
 a. Cookies b.Virus c. Firewall d. worms
- 7. The process of converting cipher text to plain text is calleda. Encryptionb. Decryptionc. key d. proxy server
- 8. e-commerce means
 a. electronic commerce
 b. electronic data exchange
 c. electric data exchange
 d. electronic commercialization.
- 9. Distributing unwanted e-mail to others is called.a. scamb. spamc. fraudd. spoofing
- 10. Legal recognition for transactions are carried out by
 a. Electronic Data Interchange
 b. Electronic Data Exchange
 c. Electronic Data Transfer
 d. Electrical Data Interchange

PART - II

Answer to all the questions (2 Marks):

- 1. What is harvesting?
- 2. What are Warez?
- 3. Write a short note on cracking.
- 4. Write two types of cyber attacks.
- 5. What is a Cookie?

PART - III

Answer to all the questions (3 Marks):

- 1. What is the role of firewalls?
- 2. Write about encryption and decryption.
- 3. Explain symmetric key encryption.
- 4. What are the guidelines to be followed by any computer user?
- 5. What are ethical issues? Name some.

PART - IV

Answer to all the questions (5 Marks):

- 1. What are the various crimes happening using computer?
- 2. What is piracy? Mention the types of piracy? How can it be prevented?
- 3. Write the different types of cyber attacks.

Reference Books :

- Computer Network Security and Cyber Ethics by Joseph MiggaKizza
- "Investigating Cyber Law and Cyber Ethics: Issues, Impacts and Practices: 1" by Alfreda Dudley and James Braman



Tamil Computing

CHAPTER



Unit V

18.1 Introduction " பிறநாட்டு நல்லறிஞர் சாத்திரங்கள் தமிழ்மொழியிற் பெயர்த்தல் வேண்டும்; இறவாத புகழுடைய புதுநூல்கள் தமிழ்மொழியில் இயற்றல் வேண்டும்; மறைவாக நமக்குள்ளே பழங்கதைகள் சொல்வதிலோர் மகிமை இல்லை; திறமான புலமையெனில் வெளிநாட்டோர்; அதை வணக்கஞ் செய்தல் வேண்டும்."

- மகாகவி பாரதி

Human civilization developed with the innovation of computer in the twentieth century. Computer development began as the early calculating tool that was essential ingredient for gigantic growth for the existence of human life without computers.

It is true that any language will be outdated when it does not have the ability to adapt itself to the changing technologies. Tamil is the living language for thousands of years. Development of modern technologies, does not affect the growth of classical Tamil as it is ready to adopt the growing technological changes. **Tamil is not just a language, it is our identity, our life and our sense.**

"எங்கள் வாழ்வும், எங்கள் வளமும் மங்காத தமிழென்று சங்கே முழங்கு" – புரட்சி கவி.

18.2 Tamil in Internet

We know that the internet today is a plays a vital role in every man's life. Internet is the best information technological device, through which we get know everything from Internet .

In 2017 a study conducted by KPMG a Singapore based organizationalong with google, reported that, Tamil topped the list, among the most widely used languages in India where 42% are using the Internet in Tamil

68%^[9] Internet users consider local language digital content to be more reliable than English

Currently, Tamil (42%^[9]) has the highest internet adoption levels followed by Hindi and Kannada among the Indian language users



Moreover in 2021 onwards, 74% of people in India will access internet using Tamil and it will be in the top usage of Internet in India.



These statistical data will be useful to improve internet services in Tamil.

18.3 Search Engines in Tamil

The "Search Engines" are used to search any information from the cyber space. Although there are many search engines, but only a few of them are frequently in use. In the top ten search engines, Google, Bing and Yahoo are takes first three places respectively. Google and Bing provide searching facilities in Tamil, which means you can search everything through Tamil. A Google search engine gives you an inbuilt Tamil virtual keyboard.

		Go	ogle	
		Ű.		
		Google தேடல்	அதிர்ஷடம் என் பக்கம்	உள்ளீட்டுக் கருவிகள்
	Google g	த்தில் வழங்குகிறது: English ஜ≓	ा गाला २००० मशक अंधराती हतूर	ർ മലയാളം പ്രഷ
இந்தியா				
விளம்பரப்படுத்தல்	வணிகம்	எங்களைப் பற்றி	K7Tot Success	alSecurity 🤏 🛪 fully completed the update! കണ് ഖീന്ദ്രப്பங്കണ

Figure 18.1(a) Google Search Engine (India)

	GC	ogle	
	Google දිනුහ Google නී නිත ඛශුණ්ල	அதிர்ஷ்டம் என் பக்கம் தெற்து: English 中文(醫体) Melayu	ீ டி உள்ளீட்டுக் கருவிகள்
சிங்கப்பூர் விளவாய்படுக்கல் வனிகம் எங்கலை	ານເມັນທີ	acflu	றிரை விகியாணகள் விகப்பங்கள்

Figure 18.1(b) Google Search Engine (Singapore)



🗲 🔿 🖸 🛢 Secure | https://www.google.co.in/search?q=告細創ப்பொறியின்+தலைமுறைகள்&cq=告細創ப்பொறி&aqs=chrome.4.69:57j015.23874j01

Figure 18.2 Searching in Tamil

18.4 e – Governance:

Getting Government services through internet is known as e-Governance. Govt. of Tamilnadu has been giving its services through Internet. One can communicate with Govt. of Tamilnadu from any corner of the state. One can get important announcements, government orders, and government welfare schemes from the web portal of Govt. of. Tamilnadu.



Figure 18.3 Official Website of Govt. of Tamilnadu

E-Governance through Tamil	Web Address
Official Website of Govt. of Tamilnadu	http://www.tn.gov.in/ta
Department of Agricultural Engineering	http://www.aed.tn.gov.in/
Department of Environment	http://www.environment.tn.nic.in/
Directorate of Govt. Examinations	http://www.dge.tn.nic.in/
Tamilnadu Health Department	http://www.tnhealth.org/
Tamilnadu Micro, Small and Medium Enterprises Department	http://www.msmeonline.tn.gov.in/
Rural Development and Panchayat Raj Department	http://www.tnrd.gov.in/
Backward, Most Backward and Minorities Welfare Department	http://www.bcmbcmw.tn.gov.in/
Tamilnadu Forest Department	https://www.forests.tn.gov.in/
Hindu Religious and Charitable Endowments Department.	http://www.tnhrce.org/
Tamil Nadu Public Service Commission (TNPSC)	http://www.tnpsc.gov.in/tamilversion/index. html
Official Website of Govt. of Srilanka	https://www.gov.lk/index.php

Outside India, Government of Srilanka provides all their services through the official website in Tamil.

18.5 e-Library

E-Libraries are portal or website of collection of e-books. Tamil e-Library services provide thousands of Tamil Books as ebooks mostly at free of cost. It is the most useful service to Tamil people who live far away from their home land.

Tamil e-Library	Website address
Tamilnadu School Education	
and Teacher Education Training	http://www.textbooksonline.tn.nic.in/
Textbooks and Resource Books	
Tamil Virtual Academy	http://www.tamilvu.org/library/libindex.htm
Connomero Dublic Librory	http://connemarapubliclibrarychennai.com/
Connemara Public Library	Veettukku_oru_noolagam/index.html
Tamil Digital Library	http://tamildigitallibrary.in/
Chennai Library	http://www.chennailibrary.com/
Thamizhagam	http://www.thamizhagam.net/parithi/ parithi.html

Project Madurai	http://www.projectmadurai.org/pmworks. html
Old Books and Manuscripts	http://www.tamilheritage.org/old/text/ ebook/ebook.html
Noolaham	http://www.noolaham.org/wiki/index.php/
Anna Centenary Libraray	http://www.annacentenarylibrary.org/

18.6 Tamil Typing and Interface software

Tamil is mostly used to type documents in word processors and search information from internet. Typing Tamil using Tamil interface software is the familiar one among the different methods of typing. This is the simplest method of typing Tamil in both Computer and Smart phones.

18.6.1 Familiar Tamil Keyboard Interface:

- NHM Writer, E-Kalappai and Lippikar are familiar Tamil keyboard interfaces software that is used for Tamil typing which works on Tamil Unicode, using phonetics.
- Sellinam and Ponmadal are familiar Tamil keyboard layouts that works on Android operating system in Smart phone using phonetics.



Figure 18.4 eKalappai Opening screen

18.7 Tamil Office Automation Applications

Famous Office automation software like Microsoft Office, Open Office etc., provides complete Tamil interface facility. These softwares are downloadable and installed in your computer. After installation, your office automation software environment will completely changed to Tamil. Menu bars, names of icons, dialog boxes will be shown in Tamil. Moreover, you can save files with Tamil names and create folders with Tamil names.



Figure 18.5 Libra Office Writer Environments in Tamil

Apart from that Tamil Libra Office, Tamil Open Office, Azhagi Unicode Editor, Ponmozhi, Menthamiz, Kamban, Vani are office automation software working exclusively for Tamil. You can these applications are designed to work completely in Tamil.

18.8 Tamil Translation Applications

Thamizpori (தமிழ்பொறி) is a Tamil tranlation application having more than 30000 Tamil words equalent to English words. Using this application, we can transalte small english sentences into Tamil. Google also gives an online translation facility, using this online facility we can translate from Tamil to any other language vice versa.

18.9 Tamil Programming Language

Programming languages to develop software to computers and smart phones are available only in English. Now, efforts are taken to develop programming languages in Tamil. Based on Python programming language, the first Tamil programming language "Ezhil" (எழில்) is designed. With the help of this programming language, you can write simple programs in Tamil.

TSCII (Tamil Script Code for Information Interchange)

Computers are handle data and information as binary system. Every data should be converted into binary while it is feed into a computer system. You learnt about all these things in the first unit of this text book. Computers use ASCII encoding system to handle data and information. The ASCII encoding system is applicable only for handling English language. Therefore, TSCII (Tamil Script Code for Information Interchange) is the first coding system to handle our Tamil language in an analysis of an encoding scheme that is easily handled in electronic devices, including non-English computers. This encoding scheme was registered in IANA (Internet Assigned Numbers Authority) unit of ICANN.

ISCII (Indian Script Code for Information Interchange)

This is one of the encoding schemes specially designed for Indian languages including Tamil. It was unified with Unicode.

Unicode:

Unicode is an encoding system, designed to handle various world languages, including Tamil. Its first version 1.0.0 was introduced on October 1991. While introduction of this scheme, can be able to handle nearly 23 languages including Tamil. Among the various encoding scheme, Unicode is the suitable to handle Tamil.

18.11 Tamil Operating System

An operating system is needed to access electronic systems such as computer and smart phone. Microsoft Windows is very popular operating system for personal computers. Linux is another popular open source operating system. Operating systems are used to access a computer easily. An operating system should be easy to work and its environment should be in understandable form. Thus, all operating systems used in computers and smart phones offered environment in Tamil.

Windows Tamil Environment interface should be downloading and install from internet. It shows all windows elements such as Taskbar, desktop elements, names of icons, commands in Tamil.

18.12 Organisation and projects to develop Tamil

Tamil Virtual Academy:

With the objectives of spreading Tamil to the entire world through internet, Tamil Virtual University was established on 17th February 2001 by the Govt. of Tamilnadu. Now, this organisation functioning with the name "Tamil Virtual Academy". This organisation offers different courses regarding Tamil language, Culture, heritage etc., from kindergarten to under graduation level.

Website: http://www.tamilvu.org/index.php

Tamil Language Council, Singapore

With the objectives of promoting the awareness and greater use of Tamil among the Singaporeans, in 2001 the council of Tamil Language was formed by the ministry of Information Communications and Arts, Govt. of Singapre. The council is called as "வளர்தமிழ் இயக்கம்".



Website: http://tamil.org.sg/ta

Madurai Project

Project Madurai is an open and voluntary initiative to collect and publish free electronic editions of ancient tamil literary classics. This means either typing-in or scanning old books and archiving the text in one of the most readily accessible formats for use on all popular computer platforms.

Since its launch in 1998, Project Madurai released in Tamil script form as per TSCII encoding. Since 2004 they started releasing ebooks in Tamil unicode as well.

Web Site: http://www.projectmadurai.org/

Tamil Wikipedia:

Wikipedia is a open source encyclopedia. Any person can write article about any subject. In Tamil Wikipedia has more than 1 lacks articles.

Web Site: https://ta.wikipedia.org/

In order to make Tamil as a living language, it is the duty every Tamilian to make

participate Tamil in development of technology. Those who forgotten their values, the will be considered as "Nomads". If we learn about how many great technologies we have to add Tamil as a symbol of our race. It is our duty to combine our world's first language and language for more than five thousand years with growing technology.

Points to Remember:

- Tamil topped the list of the most widely used languages in India by the end of 2016, while 42% are using the Internet.
- Google and Bing provide searching facilities in Tamil.
- Getting Government services through internet is known as e-Governance.
- Tamil e-Library services provide thousands of Tamil Books as ebooks mostly at free of cost.
- Thamizpori (தமிழ்பொறி) is a Tamil tranlation application having more than 30000 Tamil words equalent to English words.
- The first Tamil programming language is "Ezhil" (எழில்)
- Unicode is an encoding system, designed to handle various world languages, including Tamil.
- Among the various encoding scheme, Unicode is the suitable to handle Tamil.
- Windows Tamil Environment interface should be downloading and install from internet.

Evaluation

Answer to the following questions

- 1. List of the search engines supporting Tamil.
- 2. What are the keyboard layouts used in Android?
- 3. Write a short note about Tamil Programming Language.
- 4. What TSCII?
- 5. Write a short note on Tamil Virtual Academy.







WORD	MEANING
Paradigm	Organizing principle of a program.
Abstraction	Abstraction refers to showing only the essential features without re- vealing background details
Modularity	Designing a system that is divided into a set of functional units (named modules) that can be composed into a larger application.
Base class	A class whose properties are inherited by other newly created classes .Also called as parent class
Derived class	A class which inherits the properties of the base class. Also called as child class or subclass.
<u>Class</u>	Class represents a group of similar objects that share common properties
Object	Identifiable entity with some characteristics and behaviour
Encapsulation	Mechanism by which the data and function sare bound together into a single unit
Inheritance	Process of creating new classes called derived classes, from the exist- ing or base classes.
Signature	Number of argument and type of argument
Polymorphism	many forms
Default argument	Initializing the argument with a value
Base Class:	A class from which another class inherits (Also called Super class or parent class)
Derived Class:	A class inheriting properties from another class. (Also called Sub class)
Inheritance	The process of one class to inherit properties from another class
Inheritance Hierarchy (Inheritance Path):	The chain depicting relationship between a base class and the derived class (Also called Derivation Hierarchy)
Visibility mode	The public, private or protected specifier that controls the visibility and availability of a member in a class
Vulnerability	The possibility of being attacked or harmed.
Ethics	Moral principles that govern a person's behaviour or the conducting of an activity.

Cyber	Characteristic of the culture of computers, information technology,
· Commenter Crimes	and virtual reality.
Computer Crime	Computer crime is an intellectual crime to manipulate computer system.
Authenticity	The quality of being real or true.
Sabotage	Deliberately destroy, damage, or obstruct.
Perpetrator	A person who carries out a harmful, illegal, or immoral act.
Software Piracy	Software Piracy is the copyright violation of software created original- ly by one person and illegally used by someone else.
Hacking	Hacking is gaining unauthorized access to computer system without the owner's permission.
Cracking	Cracking is gaining unauthorized access to computer systems to commit a crime, such as stealing the code to make a copy-protected program run thus denying service to legitimate users.
Malicious	Intentionally doing harm.
Freeware	Freeware is a software available free of charge.
Shareware	Shareware is a software that is distributed free of charge on a trial basis for a limited time.
Phishing	Phishing is a term used to describe a malicious individual or group of individuals who scam users by sending e-mails or creating web pages that are designed to collect an individual's online bank, credit card, or other login information.
Fraudulent	Dishonest, cheating, swindling, corrupt, criminal, illegal, unlawful.
Anonymous	Unnamed, nameless, unidentified, unspecified.
Cookies	Cookies are messages that web servers pass to your web browser when you visit Internet sites
Tampering	Interfere in order to cause damage.
Immune	Resistant to a particular infection or toxin.
Firewall	A firewall is a network security system that monitors and controls incom- ing and outgoing network traffic based on predetermined security rules.
Proxy server	A proxy server is a gateway between a local network and a larg- er-scale network such as the Internet. Proxy servers provide increased performance and security.
Encryption	Encryption is the process of encoding a message or information so that only authorized users can decrypt it
Decryption	Decryption is theprocess of decoding the encrypted text by converting it back into normal text.

COMPUTER SCIENCE – XI VOL-II List of Authors and Reviewers

Domain Expert

Dr. T.V.Gopal Professor , Dept. of Computer Science and Technology, College of Engineering, Guindy, Anna University, Chennai.

Reviewers

Dr. Ranjani Parthasarathi Professor , Dept. of Info. Sci and Technology, College of Engineering, Guindy, Anna University, Chennai.

Mr. Munivel E. Scientist/Engineer 'C' IT Group (Information Security) , NIELIT Calicut (MeitY, Govt. of India), NIT Campus, Calicut, Kerala.

Content Experts

Dr. Radha P. Assistant Professor, Dept. of Information Technology, Govt. Arts & Science College (A), Coimbatore.

Dr. Nester Jeyakumar M. Associate Professor and Head of the Department, Dept of Computer Science, Loyola College ,Chennai.

Mr. Sankar K. Assistant Professor, Dept. of Computer Science, RKM Vivekananda College, Mylapore, Chennai.

Art and Design Team

Layout THY Designers and Computers, Chennai.

Wrapper Design Kathir Arumugam

QC Manohar Radhakrishnan Jerard Wilson P. Arun Kamaraj

Co-ordination Ramesh Munisamy

Typist Meena T. SCERT, Chennai.

Authors

Mr. Kannan K. Post Graduate Teacher, Chennai Girls Hr. Sec. School, Rotler street , Chennai.

Mr. Ramakrishnan V.G. Post Graduate Teacher, Karnataka Sangha Hr. Sec. School, T. Nagar, Chennai.

Mrs. Bindhu Mohandas Post Graduate Teacher, Vijayanta Model Hr Sec School, H.V.F Estate, Avadi, Chennai.

Mr. Gowrisankar N.V. Post Graduate Teacher, Chennai Girls Hr Sec School, Nungambakkam, Chennai.

Mr. Sreenivasan R. Post Graduate Teacher, Santhome Hr Sec School, Mylapore, Chennai.

Mr. Lenin K. Post Graduate Teacher, Chennai Girls Hr Sec School, Saidapet, Chennai

Miss. Sangeetha A. Post Graduate Teacher, Govt. Hr Sec School, Rajanthangal, Thiruvannamalai District.

Dr. Valarmathi K.E. Post Graduate Teacher, Velammal Vidhyashram, Surapet, Chennai.

Mrs. Gajalakshmi R Post Graduate Teacher, Jaigopal Garodia Hindu Vidyalaya Hr Sec School, West Mambalam, Chennai.

Mrs. Vidhya H. Post Graduate Teacher, DAV Boys Senior Seconary School, Gopalapuram, Chennai.

Academic Coordinator

Mr. Ravikumar Arumugam Deputy Director, State Council of Educational Research and Training, Chennai.

Mrs. Tamil Selvi R. B.T. Assistant, Government High School, Poonampalayam, Trichy District.

QR Code Team

R. Jaganathan, S.G.T., PUMS, Ganesapuram- Polur, Thiruvannamalai Dist.

N. Jagan, B.T. Asst., GBHSS, Uthiramerur, Kancheepuram Dist.

J.F. Paul Edwin Roy, B.T. Asst., PUMS, Rakkipatti, Salem Dist.

This book has been printed on 80 G.S.M. Elegant Maplitho paper. Printed by offset at: