When you select the option for inserting a chart, you can see a chart wizard. The chart wizard of Calc is actually a set of simple and user friendly steps to perform charting operations in the spreadsheet package. What one has to do is just select the data arrange and invoke the chart wizard. Rest of the things are taken care by the wizard. The chart wizard dialog box is as shown in figure 8.2.

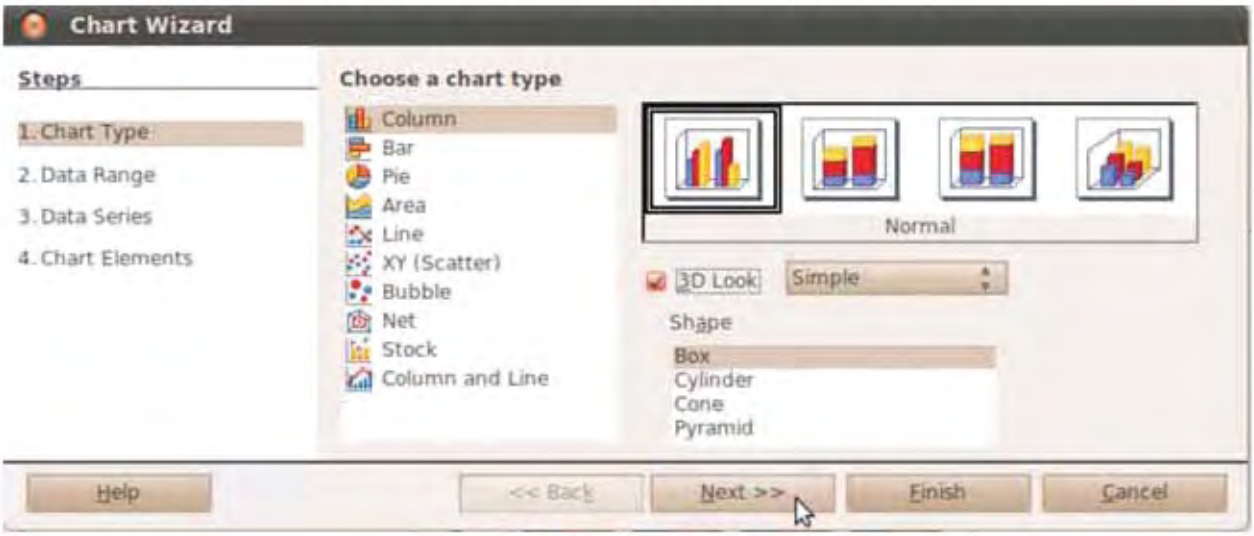


Figure 8.2 : Chart wizard

The Chart wizard as shown in figure 8.2 lists 4 steps seeking information about chart type, data range, data series and chart elements. These steps are discussed below:

Selecting Chart Type

As shown in figure 8.2, select a column chart. You can also see preview of how a typical column chart looks. The preview updates every time you select a different type of chart, and provides a good idea of what the selected chart will look like.

The chart wizard has the following main components.

- List of steps involved in setting up the chart;
- List of chart types;
- Preview of chart types;
- Options for each chart type; and
- Options to proceed further or going back and modify the choices.

You may see other chart types along with their respective preview in the chart wizard. The chart types are Column chart, Bar chart, Pie chart, Line chart, Area chart, XY chart, Bubble chart, Net chart, Stock chart and Column chart. Calc offers a choice of 10 basic chart types. Once you select a chart type, you may choose further options such as look of a chart and shape of the chart. The Calc also offers a few options for each type of chart. Options vary according to the type of chart you pick. At any time you can go back to a previous step and change selections.

There is a 3D view box also. If you select the 3D box (by clicking on it), you may see the changed preview. Only those types which are suitable for 3D (such as column chart, bar chart, pie chart, and area chart) give you an option to select a 3D look. You may also select shape of the chart such as "Box", "Cylinder", "Cone" and "Pyramid" by selecting appropriate choice in the Shape box. Appropriate preview is shown as and when you made choices.

For our example, we have selected following choices :

- The chart type is Column chart;
- The chart look is **3D** with **Simple** view;
- The chart shape is **Box.**

At the end, click the Next button. You may observe that the Calc has prepared a default chart in the background of the chart wizard. At this stage, if you select the Finish option, then you see the default chart, which you may modify or format afterwards. You may choose Back button to go back and modify your choices.

Data Range

Once you have selected the chart type, you need to provide necessary data. One possibility is that you have already selected a range of data before clicking the Insert → Chart option; as we did. If not so, we can still select it by clicking on the Data Range option as shown in figure 8.3.



Figure 8.3 : Data range selection

Calc is smart enough to identify data range even if the part of the data range is selected instead of the complete data range prior to insertion of a chart. That it, suppose you have selected only a cell such as B3, and you choose **Insert → Chart,** even though, Calc automatically guesses the data range. However, it is a good practice to verify the data range. You may use already existing data from other spreadsheet documents or other compatible documents as mentioned below.

Calc charts can be based on the following data :

- Spreadsheet values from Calc cell ranges.
- Cell values from a Writer table. You may create the data in packages such as Writer, Draw, or Impress, and you can copy and paste them into the Calc spreadsheet.
- Any properly formatted text file or table.

In our case the data are coming from the same sheet and its range is from A1 to C5. Note that the Calc has considered the data range as $Sheet1.$A$1:$C$5. That is, the Calc converts the range into an absolute address. Hence, if you move your chart to any other location, the data range remains same (absolute).

The next button is asking about the location of data series, in rows or as columns. Here we have

entered our marks in columns. Hence, we can select option **Data series in column.** Also, we need to select the option **First row as label.** This option is useful when the data contains multiple columns and each is having a heading. The first row makes all heading for every column. By mentioning the first row contains all headers, we can specify labels of columns. In our example of student's marks, the first row contains headings such as **Quiz Marks** and **Test Marks.**

Similarly, when every row has a heading, you may select **First column as label** option. You can select both the checkboxes as well.

Data Series

The chart wizard also allows you to customise data ranges for the chart preparation. Here we use values selected automatically by the chart wizard. Figure 8.4 shows the default data series considered by the chart wizard.

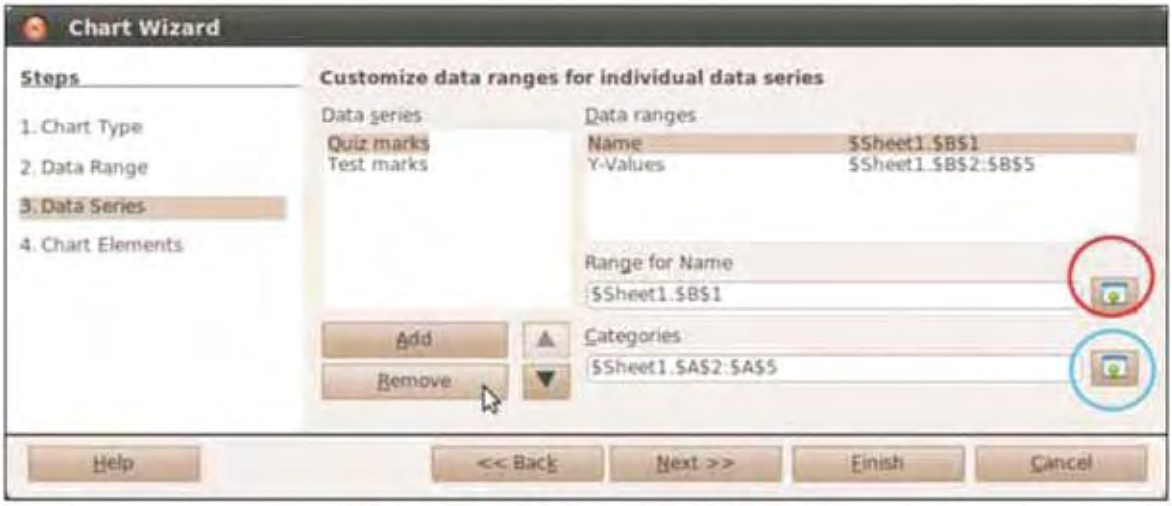


Figure 8.4 : Data series selection

You may add or remove the data series using this dialog box. To add data series, such as final exam marks in the current example, use **Add** button. To remove any data series, select the series, which you want to delete and choose, **Remove** (see mouse arrow in figure 8.4). The dialog box also provides facility to enter range name and range values. Here range name is $B$1, having value "Quiz marks". You can use **Select** button (encircled with red line in figure 8.4). It will minimize the chart wizard and lead you to the worksheet area. You may select appropriate cell. When you complete the selection, press the **Maximize** button of the chart wizard. This will be helpful if you do not remember the location of the cell(s).

Similarly, you can select values of the data series. In our example, the data series values are the quiz marks located within cells B2 to B5. You may write it as $Sheet1.$B$2: $B$5.

The data series phase of the chart wizard also lets you enter the Categories of the data. In our example, data series categories are the month names located within cells A2 to A5. You may write it as $Sheet1.$A$2: $A$5. Here also we can use the Select (encircled with blue line in figure 8.4) button to minimize the chart wizard and select the values directly from the spreadsheet. In the end, choose Maximize button.

Data Elements

Chart wizard lets you to enter values of different chart elements such as title, subtitle, X-axis captions, location of legends and options to display the grids. Figure 8.5 shows a screen of chart wizard that allows us to set data elements.



Figure 8.5 : Chart elements

The chart elements such as title of the chart, sub title of the chart and information about axis and legends are provided through this phase of the chart wizard. As shown in figure 8.5, the first option within the **Chart Elements** option is **Title.** Here you can provide main title of the chart. The second option is **Subtitle,** which lets you enter the subtitle of the chart. The title and subtitles are placed on top of the chart within the chart area. Once you are done with the title and subtitle, you may provide information about the axis. Title information regarding the three axes namely X axis, Y axis and Z axis can be provided here. You can also choose to display legend or not at this stage using **Display legend** option. If you have opted for **Display legend** option, you need to provide further choice about where you want to display the legends; left, right, top or bottom of the chart. By default, the legend is visible and placed right side of the chart within the chart area.

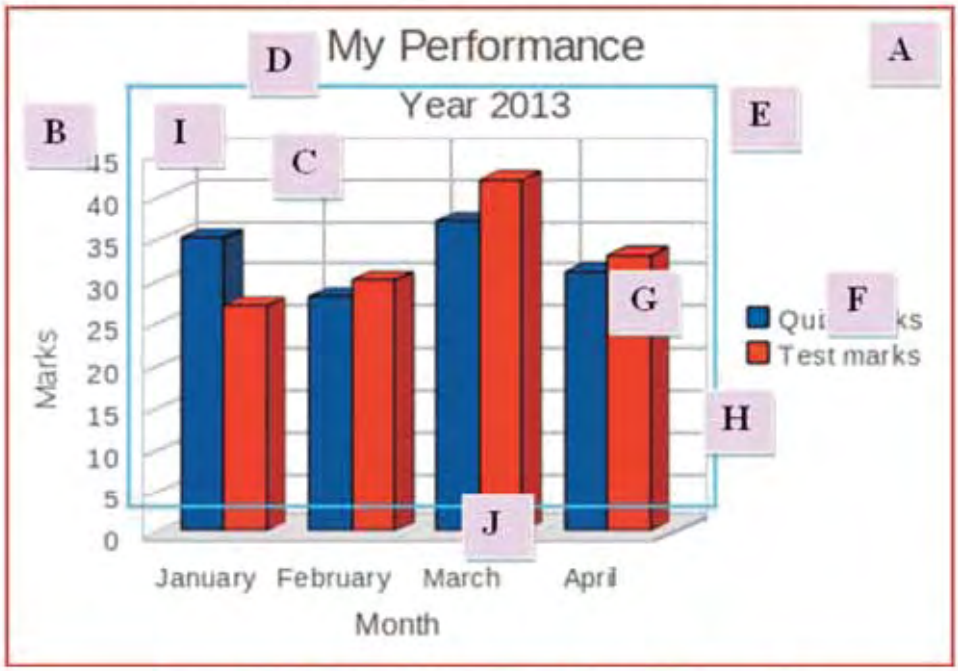


Figure 8.6 : Simple chart and chart elements

Once the correct options are given, you may click on Finish button to create chart. The chart is now placed in the current sheet. You can see the chart as shown in figure 8.6 with labels to the chart's elements. The elements of the charts are discussed in next section.

Elements of a Chart

A chart generally has components such as chart area, chart title and subtitles, chart axis with values, chart data area axis, chart data area, legends for the chart axis, data series, and chart floor. We have already created a simple chart showing a student's performance in quizzes and tests. To know about the specific component of a chart, you may just put the mouse arrow on it. It will display the component name. Let us examine some elements in the Calc chart. Figure 8.6 also shows some labels indicating elements of chart. These labels are described below :

A: Chart area : The chart area is the rectangular area in which the chart is embedded.

B: Y axis with values : The Y-Axis values are taken automatically according to the data we have provided. Here we have given marks not more than 50, the Y axis values are considered as 0 to 45.

C: Data area axis : The data area axis is the axis within the data area. By choosing this option, you may see vertical or horizontal (or both) grid lines on the background of the chart. The main utility of the axis within the data area is to identify value, if the chart does not display it.

D: Title and Subtitle : The title of the chart is a name given to the whole chart. Here we have given the title as My Performance. Along with the title, we may also give subtitle. The purpose of the subtitle is to indicate additional information or the purpose and use of the chart. Here we have used Year 2013 as subtitle.

E: Data area : The data area within the chart area actually holds the chart. It is also known as Chart. The chart is a core part of the chart area, hence usually placed in center of the chart area. The area surrounded by blue coloured line boundary in figure 8.6 is known as chart or data area.

F: Legend : The chart is a kind of drawing. It must have some text to indicate about its data and graphics. The chart shown in figure 8.6 has columns of blues and reds. From title, X-Axis and Y-Axis labels you can learn that the chart is about marks of various tests in different months. However, it is not clear which type (colour) of the columns represents which marks. To accommodate such information in a chart, legends are used. Here, in figure 8.6, we have used legend indicating blue colour for Quiz marks and red for Test marks.

G: Data series column : Chart may contain more than one data series. For every data series a colour is chosen and chart is build. For example, for a column chart, for each data series a colour is picked up and many columns of that colour are developed representing various values at different period of time/stage. Here the marks of quizzes are considered as a data series and a blue colour is chosen automatically. Calc prepares many columns of blue colour, each representing a mark in a month. These blue colour columns in figure 8.6 actually represent a data series for Quiz marks. Similarly, red colour columns in figure 8.6 actually represent a data series for Test marks.

H: Chart floor : The chart floor is the horizontal plane which provide base to the chart elements. For particular type of chart, such as bar and column charts, it is quite useful. You can see in figure 8.6 the chart floor is filled with grey colour.

I: Chart wall : The chart wall is the vertical plane which provide base to the Y axis of the chart. As shown in figure 8.6 it is not filled with colour. However, you may modify that to fill a colour of your choice.

J: legends on X-Axis : Consider a simple chart is given to you. The X-Axis of a chart has only values on it. Just like in figure 8.6, we have used month names on X-Axis, such as "January", "February", etc. These numbers are self explanatory. One can easily know that we are talking about

months of the year. What if we want to mention speed of a vehicle ? Of course, we can write 10, 20, 30 etc. But is there any way to specify these are kilometre per hour; or meter per minutes? These units and categories of the items on X-Axis can be specified with the legend on X-Axis. Similarly, we can specify the legend for the Y-Axis too.

An Alternative Way to Create a Chart

Alternatively, you can select the chart icon from the standard tool bar. If you cannot see the standard toolbar, you can make the toolbar visible by View ' Toolbars options as discussed earlier. Figure 8.7 shows the chart icon (encircled) on the toolbar.



Figure 8.7 : Chart Icon

You need to perform just three steps.

1. Enter data into a sheet.
2. Select it and press chart icon as mentioned above.
3. Select **Chart type** and click **Finish**. You may not give other options at this stage.

When you click the Finish button, your chart is ready and placed in the current sheet. Your chart may not be having all titles and legends. However, if you like the chart, you can always format it to make it more effective and attractive. The following section explains ways of formatting and modifying charts.

Formatting and Modifying Charts

Many times you need to customize the chart according to your requirement. That is, you may wish to modify the chart according to your style. Let us discuss different formatting options of the chart.

Modifying the Place and Size of the Chart

When a chart is created it assumes the default appearance and placement, usually in current worksheet. Sometimes you do not like the location of the chart where it is placed by default. Sometimes you do not like its size. You can move the whole chart by selecting it with just a left click and dragging it (by its handle with mouse) to a desired position. You may copy it or cut it and paste at a new location. Similarly, you can resize the chart by selecting the chart and dragging it using mouse through its handle. See figure 8.8.



Figure 8.8 : Modifying Chart Size

At times after the chart is prepared you may need to change some values in the data, feel free to change the values. The chart will be automatically changed. After a chart is inserted or selected in a spreadsheet using mouse, the toolbar of the Calc package changes to show chart buttons and commands. When the corresponding data in the cells are changed; the chart changes as well.

It is possible not only to format the chart, but also modify the chart. The following section describes various ways to format or modify a chart.

The first thing you may notice about the chart, besides its location and size, is its type. Say, from the column type of chart, you want a line chart now. The first thing you must remember is to bring the chart into edit mode. To do so double (left) click in a **Chart Area**. A grey border appears around the chart as shown in figure 8.9. You may right click the chart area to modify its various parameters.

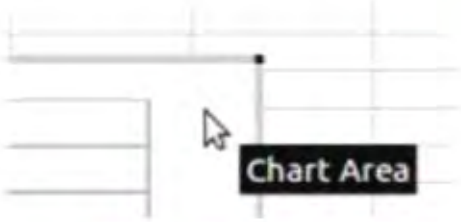


Figure 8.9 : Chart Area

Modifying Chart Type

To modify the chart type, perform the menu command **Format → Chart Type**. Alternatively, you may press the **Chart Type** icon on the chart formatting toolbar as shown in figure 8.10.



Figure 8.10 : Chart formatting toolbar

The chart formatting toolbar appears when you select the chart. The chart formatting toolbar is used to format the typical chart elements such as chart area, chart data series, chart axis, chart titles, etc. In figure 8.10 the third icon encircled with a blue colour line is about chart type. When you click the encircled icon, it will display a dialog box. The options in the dialog box allow as changing the chart type and look of the chart such as 3D chart and shape of the chart such as Box or Pyramid.

You may directly right click on the chart to modify its type. When you right click, a small vertical menu appears with possible operations you can perform at the current stage.

Formatting the Chart Area

As mentioned above, chart area is an element of a chart. When you click on a chart, the chart toolbar is activated as shown in figure 8.11. The first icon in the toolbar is presenting a list of all the elements you can format. You can select any particular chart element which you would like to format from this list. For this, use small triangular up and down arrow keys shown with the elements box as shown in figure 8.11. See the encircled portion with red line in figure 8.11. Once you select the appropriate element then choose the next option **Format Selection**. See blue colour circle in figure 8.11. In the same figure (8.11), we have selected **Chart area**. However, you can see the list of other elements of the chart. After selecting the **Chart area**, next step is to select **Format selection** button.

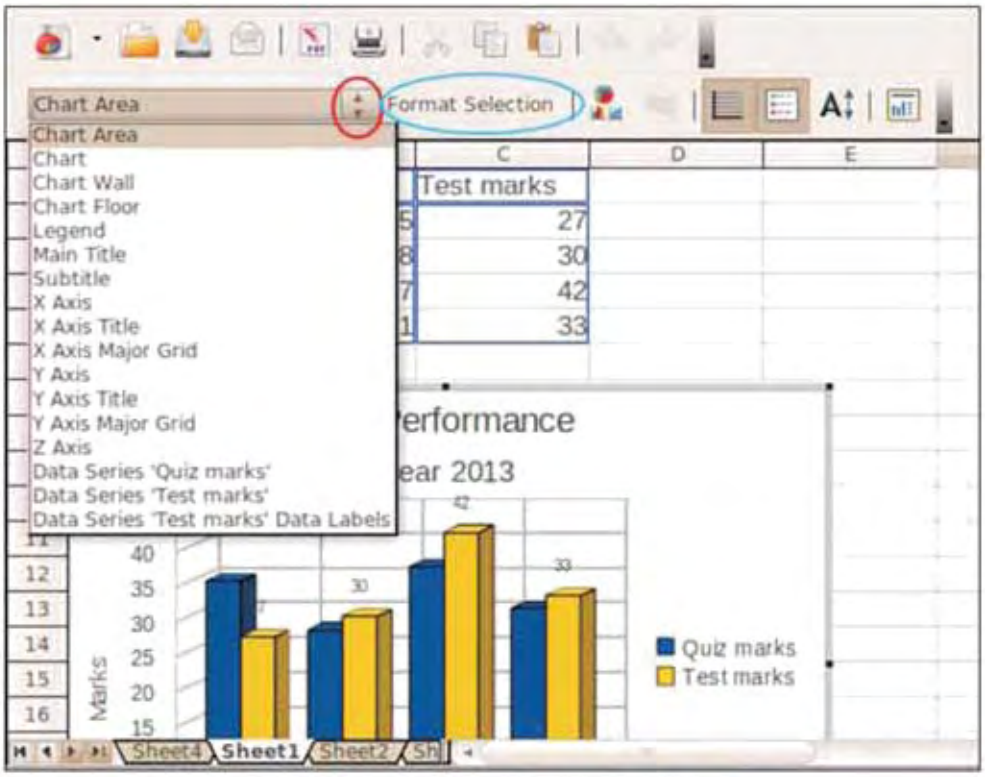


Figure 8.11 : Formatting chart area

This will allow us to format the selected chart element by providing suitable dialog box. The dialog box shown in figure 8.12 allows us to format border of the area, Area of the chart and transparency of the filled colour. The borders are by default invisible, however, you can make them visible. Further, you can also select border style such as continuous line, dotted line, thick line, etc and border colour. The area of a chart can also be filled with various colours, gradients or a bitmap image with different transparency levels. If you do not want to fill uniform colour (with or without transparency) in a chart area, you may wish to give gradient to the colours in linear, radial or axial directions.

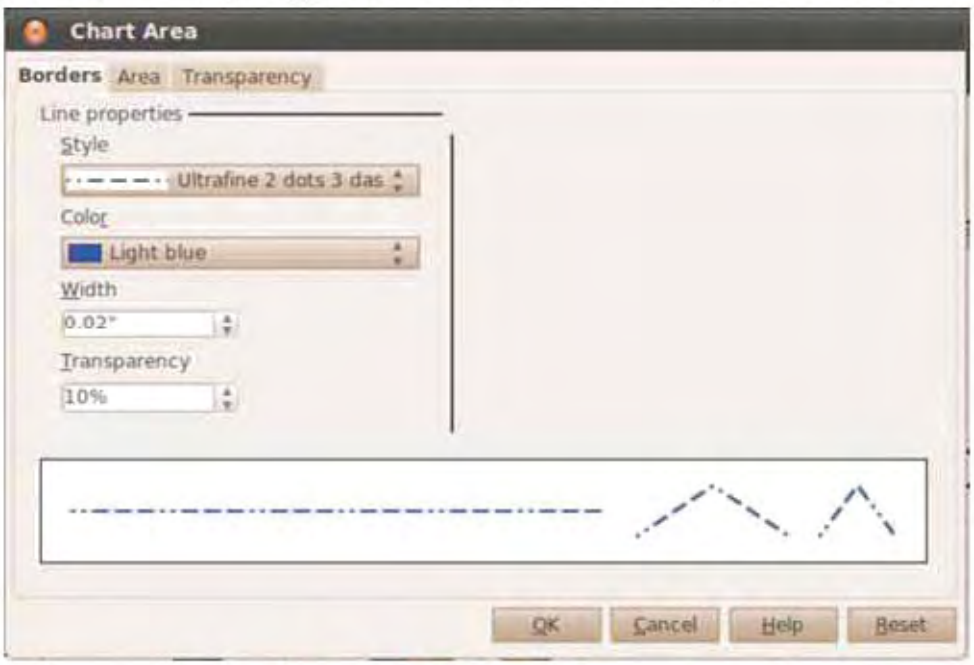


Figure 8.12 : Modifying the chart area

Formatting Titles to the Axis

To add titles to X-Axis and Y-Axis for better understanding of the chart, you can perform following actions.

- Select the chart.
- You can see the chart formatting toolbar as shown in figure 8.11. Select X-Axis title as an element to be formatted from the toolbar.

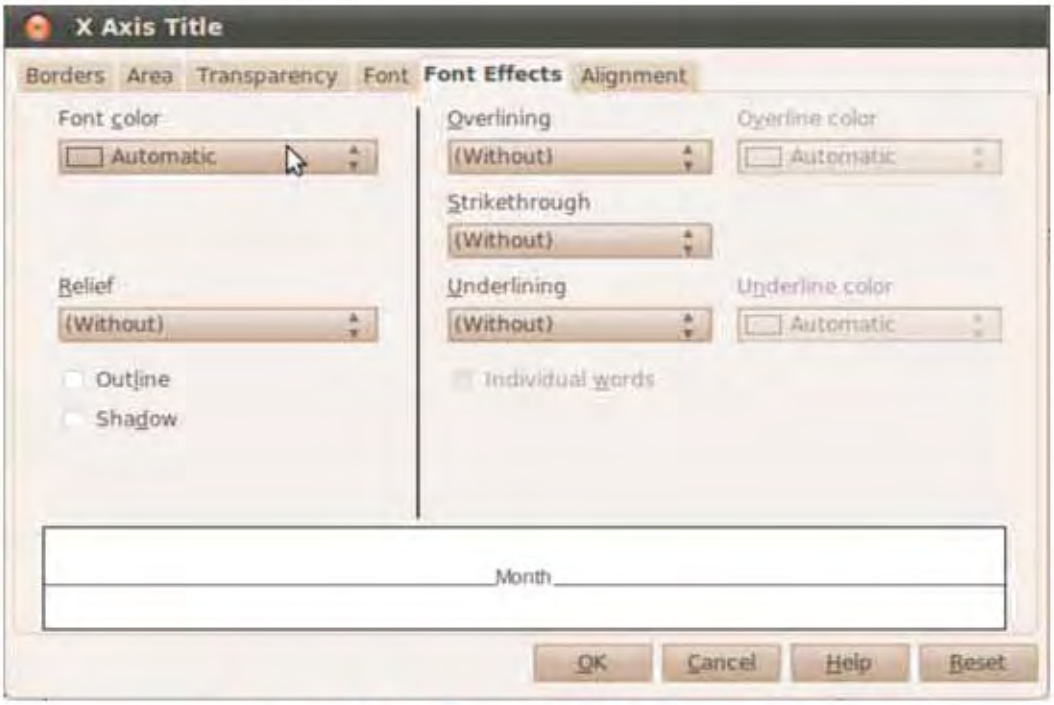


Figure 8.13 : Formatting X-Axis titles

- Press **Format Selection** button.
- You will see a dialog box called **X-Axis Titles**. This dialog box is shown in figure 8.13.
- Fill the necessary fields in the dialog box.

Using this facility of formatting, you can provide information about various attributes of the X-Axis titles such as borders of the area of title of the axis, title fonts, font effects and alignment.

This dialog box provides facility to set options for setting borders, filling the area of title and assign level of transparency to the filled area. As stated earlier, you can also format fonts and change font effects for values of the axis. The **Font** tab lets you select the appropriate fonts. If you have suitable fonts in your computer, you may display labels in regional language like Gujarati and Hindi also. You may include or exclude font's effects such as font colour, overline title fonts with different colour lines, strikethrough fonts, underline title fonts with different colour lines. You can also apply shadow and outline effect to the title fonts to look fancy and attractive. Further, you may choose text flow and text alignment (orientation) of labels by choosing the last tab of the dialog box named **Alignment** in the dialog box shown in figure 8.13. Similarly you can format the Y axis title also.

Formatting X-Axis

You can also format X-Axis using the chart formatting toolbar. Select appropriate element, here select **X-Axis;** and press **Format Selection** button. You will see a dialog box. Using this dialog box,

you may set options to set scale of the axis, which is generally automatic. Positioning of axis, that is where the axis crosses at start, end or other places with other axis. Along with the positioning of the axis, you can also mention where you want labels of the axis. You may set labels of the axis near it, outside the axis, starting of the axis or ending of the axis.

Many times labels are very descriptive, to manage the size of the chart, you may want to decrease their size or fonts. If you do not require labels at all, there is an option available called **Show labels.** You can tick mark and deselect the option, which make labels disappear for that axis. Similarly, you can format the Y-Axis also.

Formatting the Chart Wall

The chart wall can be modified by selecting options from the tool bar and selecting **Format Selection** button. By doing this a dialog box appears, which is similar to figure 8.11. The dialog box allows you to change border properties such as invisible or visible border, type of border, colour of border, thickness (weight) of the border, etc. Besides border properties, the dialog box also provides you an opportunity to change area colour and degree of transparency of the wall colour.

Formatting the Chart Floor

The chart floor can also be modified by selecting options from the chart formatting tool bar and selecting **Format Selection** button. You may select the chart floor formatting option by selecting the chart, right clicking on the chart floor and selecting the **Chart floor** option from the popup menu. By doing this, similar dialog box appears which allows you to change property of borders, area and degree of transparency of the floor of the chart.

Formatting Chart Legends

If you have totally forgotten entering the legend or skipped it at the time of building the chart, you may add chart legends by right clicking on chart and choosing option **Insert Legend.** Once the chart legends are visible, you can think for formatting them. The chart legends can be formatted using the chart toolbar shown in figure 8.11. As usual, when you opt for formatting chart legends from the chart formatting toolbar and select the Format Selection button, a dialog box appears. If you have noticed, the chart legends are also placed in a rectangular area with invisible borders. Using the dialog box, you can set borders of the legend area, format area, change degree of the transparency, etc. These options behave in a similar way as explained earlier. In addition to these options, other options such as font, font effects and position of the legend in the chart area are also provided by this dialog box. These options are required as the legend of chart also has some text besides some graphics. Further, we need to mention its location that is where we want to put the legend text. The default location is the right side of the chart; however, you can place the legend on top of the chart, bottom of the chart or left side of the chart. You can select any suitable fonts, its size and appearance such as bold, italics or underlined.

Formatting Data Series

To modify a series of data you may select the chart and right click on it. A vertical menu appears to you. Select format data series option and format it. Another method to do same is selecting the data series element from the chart formatting toolbar (as shown in figure 8.11), and select Format Selection button. This will offer an opportunity through a dialog box to set parameters for the data series.

There is another way to format data series as given below :

- When the chart is ready, click on any column representing the data series. If you have clicked on a single red colour column, you may notice that all red colour columns are activated. This represents a data series.
- See figure 8.14 showing data series for Test marks having red colour columns in selected manner.
- Formatting chart toolbar appears on the top of the sheet. See encircled area with red line. It shows **Data series 'Test Marks'.**
- Choose **Format Selection** button. You can see another rectangle dialog box along with the chart. See rectangular area with blue coloured outline.
- You can select appropriate tab, such as layout, area filling, transparency level, etc.
- In figure 8.14, you can see that we are changing the appearance of the area by changing the fill colour from red to yellow.

Similarly, you may format the other data series.

Inserting Data Labels

Another similar operation is to enter the data labels along with every data column in the chart. Suppose you want to enter the actual value along with the chart component such as columns, then just activate the data series and press right click. After that you can select **Insert data labels.** In figure 8.14 you can see that the data labels are already inserted. See encircled number with blue line in figure 8.14. Similarly you may format the other data series.

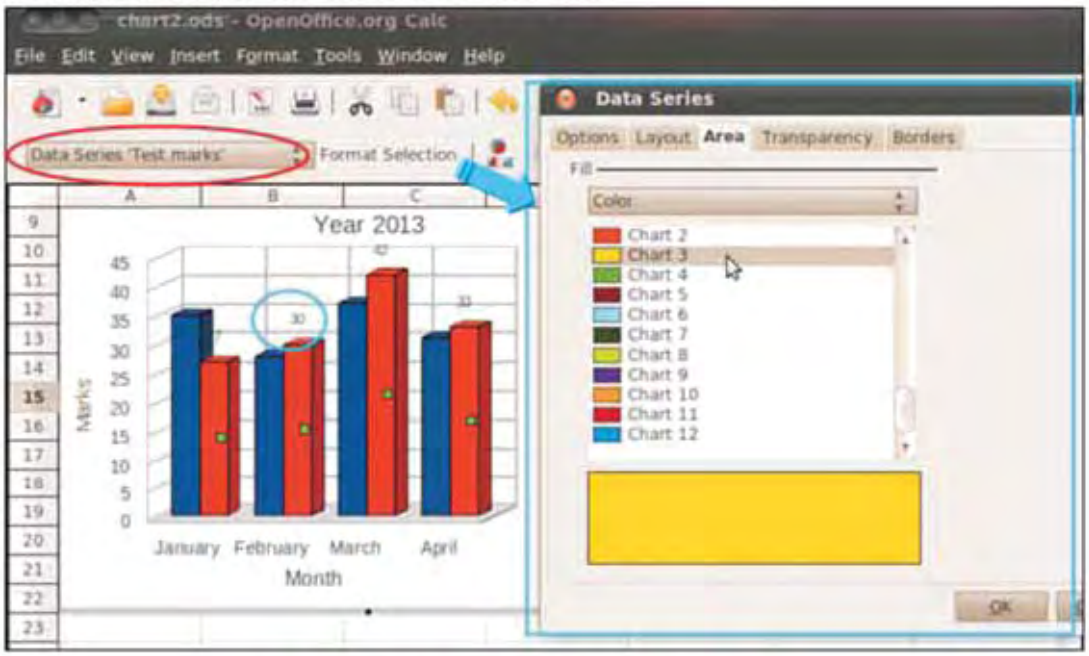


Figure 8.14 : Formatting data series

Working Example

Let us create one more chart of different type. Consider the data as follows. Let us consider a data about movies. You all might like to enjoy movies. However, everybody has got different choice and liking. Some of you like action movies and some like comedy movies. There is a special audience for science fiction films. Consider data given in table 8.2 representing the similar situation.

Movie type	Action	Comedy	Drama	Science-Fiction
Value in %	30	35	20	15

Table 8.2 : Liking of movies

Let us draw a chart using the information shown in the table 8.2. Perform following steps.

- Enter data shown in the table 8.2 into a Calc worksheet and save it by giving appropriate name.
- Check that the data range should be A1 to E2.
- Select **Insert → Chart.**
- Select the chart type as **Pie** chart;
- Choose **3D** look and **Simple** chart.
- Press **Finish** button.
- You can see the chart as shown in figure 8.15.
- However, you cannot see the data labels in your chart as well as the Drama category is also not pulled out. To add the data labels and to pull out the Drama category (represented by yellow colour), perform remaining steps as given below.
- Select the chart. You may see the grey border surrounding the chart.
- Right click the chart, which causes a vertical menu appears to you.
- Select **Insert Data Labels.**

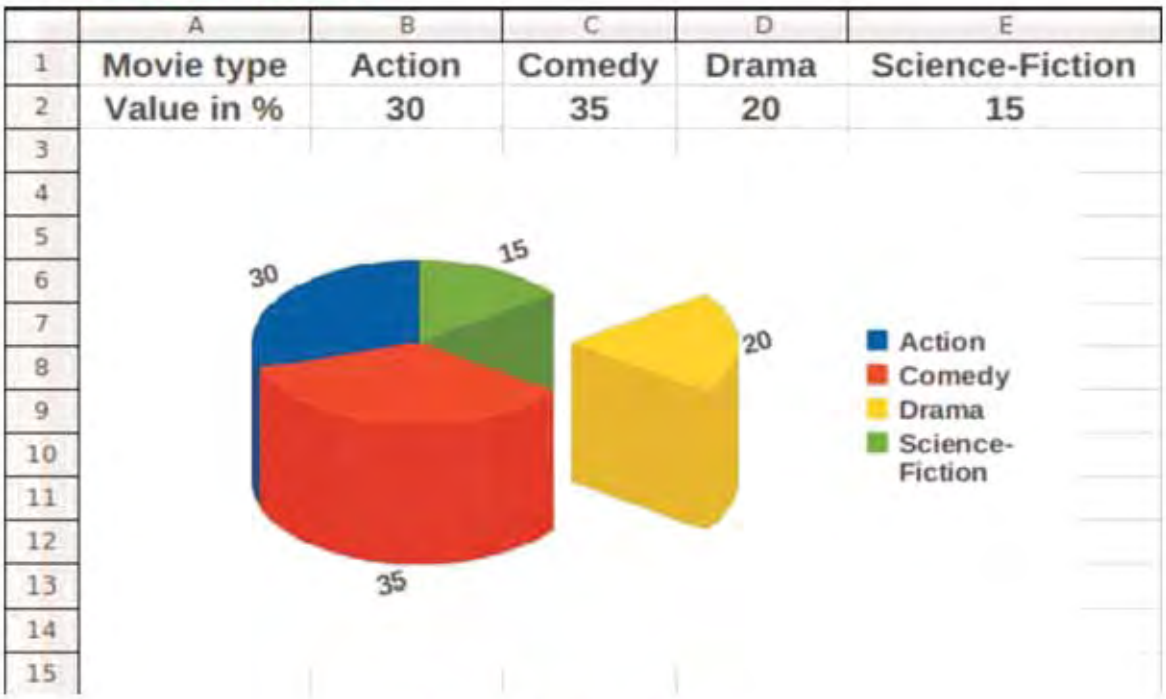


Figure 8.15 : Pie chart

- You can see the values are displayed along with the chart.
- To pull out a portion of the pie chart, just select the area using mouse key. Using the mouse, drag the area at the desired position.

- If you want to do further formatting, select the chart. The chart formatting tool bar appears to you. Select appropriate element to be formatted and choose **Format Selection**. Using this facility, you can change chart type, chart area, data series, font, font effects, appearance of axis, legends, grids, chart floor and wall, etc. Here we can make the fonts bold and increase the font size. We also have rotated the data labels with 15 degree.
- Similarly, you can format the legends of the chart and make the fonts of the legend bigger.

Adding Hyperlink to the Chart

Hyper link is a reference to data that can be followed by selecting it. As you know, a hyperlink may points to a whole document or part of a document. The text which embeds such links within it is called a hypertext. Once a chart is created, you can set hyperlink to the whole chart or to a component of the chart. If user clicks on the hyperlink, it will lead the user to predefined destination. You may set hyperlink to a document of the Web, document within your local area network/computer or to a new document.

For any type of hyperlink, you must have a chart. You have to create a chart first. Follow the below given steps to create hyperlink to the Web (Internet) document.

- Enter data and create a chart of suitable type.
- Verify the chart and select it using mouse keys.
- Select **Insert → Hyperlink**.
- It will offer you four options regarding the destination of the link. That is, you can hyperlink the chart (or selected entity) to the Internet, Mail & News, Document or New Document. Figure 8.16 shows these four ways (encircled with red oval) for linking the chart.



Figure 8.16 : Hyperlink options

By default, the first option **"Internet"** is highlighted and as a **Target** you may give the destination URL such as *Google.com*. You are supposed to have an Internet connection, if you want to experiment the link. If everything goes right, the chart will lead you to a Google's main page when clicked. Alternatively, any type of hyperlink is followed by pressing **Ctrl + Enter**.

Once you enter all necessary information, you may choose Apply. Use Close button to close the dialog box.

Linking a Chart to a Mail

If you want to connect the chart to the mail and news, you need to select Mail & News option. You will see different dialog box. Figure 8.17 shows the options used when linking to Mail & News.

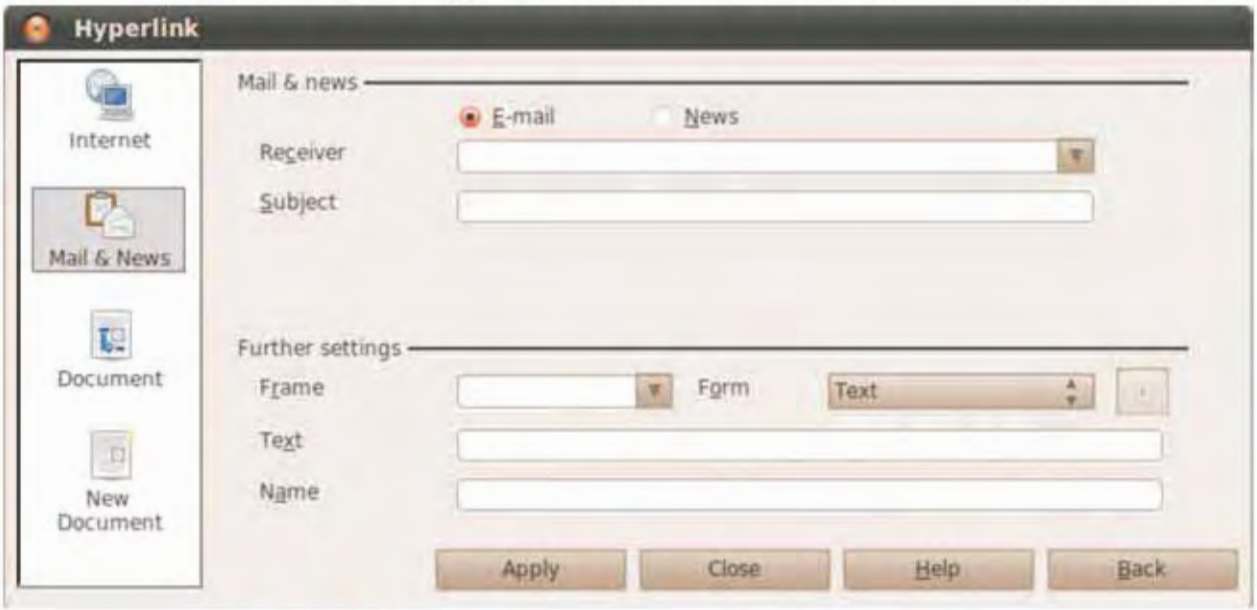


Figure 8.17 : Options for mail and news in hyperlink

You need to provide information about the receiver and subject of the mail. The chart will be queued for mailing to the receiver's address when you select the **Apply** option. However, this process remains offline and requires a mail client set at background. Similarly, for **News** option, you need to configure news server. Choose **Close** button at the end.

Linking a Chart to a Document

To link your chart to a document, you have to choose the third option Document. You may see screen as shown in figure 8.18.

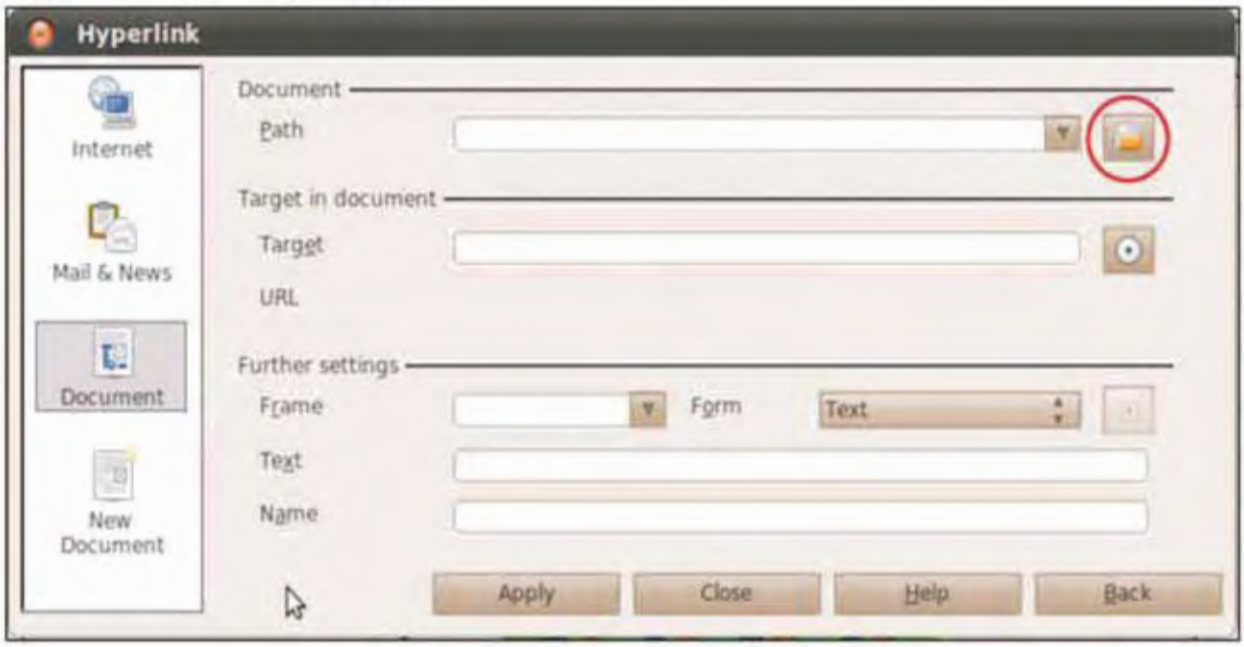


Figure 8.18 : Linking chart to a document

This option is useful when you have a small chart prepared, which is further explained in detail in another document. In this situation, for further explanation, you may link your chart to other document. Any way, if you think that a particular document is related to the chart, you may link it with the chart using this facility. The full path mentions the location of the destination document. You can browse the document, if you do not remember the complete path of the document using the Browse button (encircled with red line) given beside the path text box. You can also provide target within the destination document using the **Target** option.

At last choose **Apply** option. At the end use the **Close** button to complete the procedure and close the dialog box.

Linking a Chart to a New Document

To link the selected chart to a new document, you can choose the fourth option **New Document**. You can provide **File** name and **File type**. Spreadsheet, word processor, drawing etc. type of files you may create using this option. The file once created, is now available to edit, as **Edit now** option is selected by default. If you do not want to edit it, you may choose **Edit later** option. Figure 8.19 shows the above mentioned scenario.

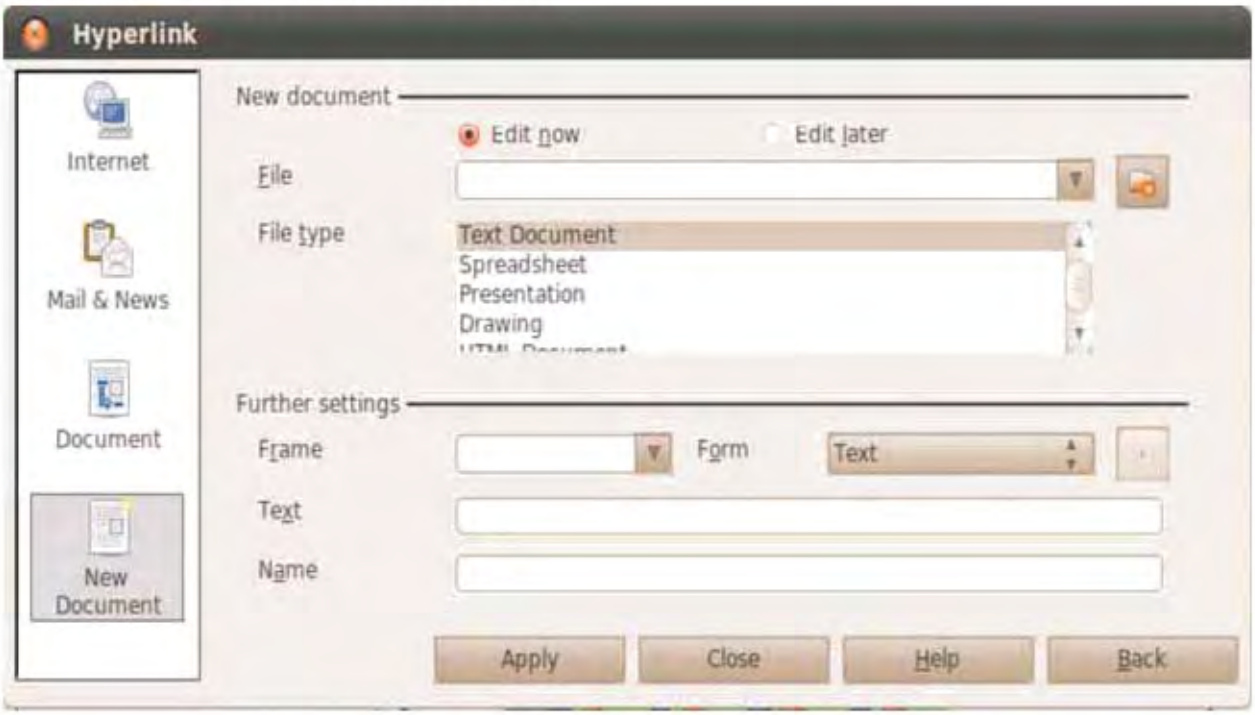


Figure 8.19 : Linking a Chart to a New Document

At the end, choose **Apply** option. Use **Close** button to close the dialog box.

Exporting the Chart and other Operations

You can export your document containing chart as portable document file having PDF extension. To do so, perform following steps:

- Select **File → Export**. You may see the screen as shown in figure 8.20.

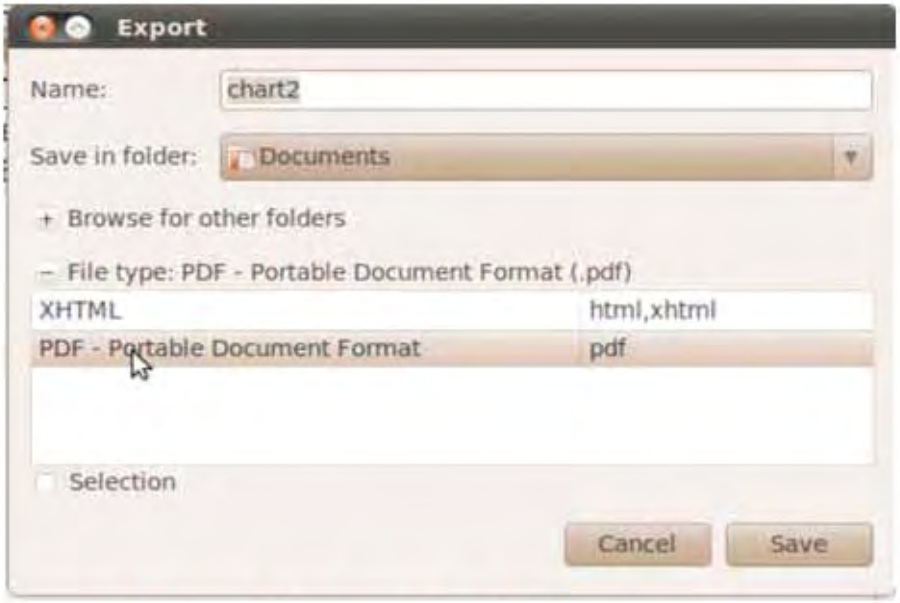


Figure 8.20 : Exporting the document

You will see two options such as export the file as XHTML (web page) or PDF (Portable Document Format).

- Choose **PDF.**
- Save the file in a desired folder.

When you give your choice as PDF; you will see a dialog box asking various parameters about the resulting PDF file as shown in figure 8.21. Using the dialog box shown in figure 8.21; options such as number of pages, format of images, (if it contains any) etc. can be set. If you have selected a range of pages that you wanted to export; content of only the given number of pages will be exported into the resulting PDF document. The default range is all the pages of the document.

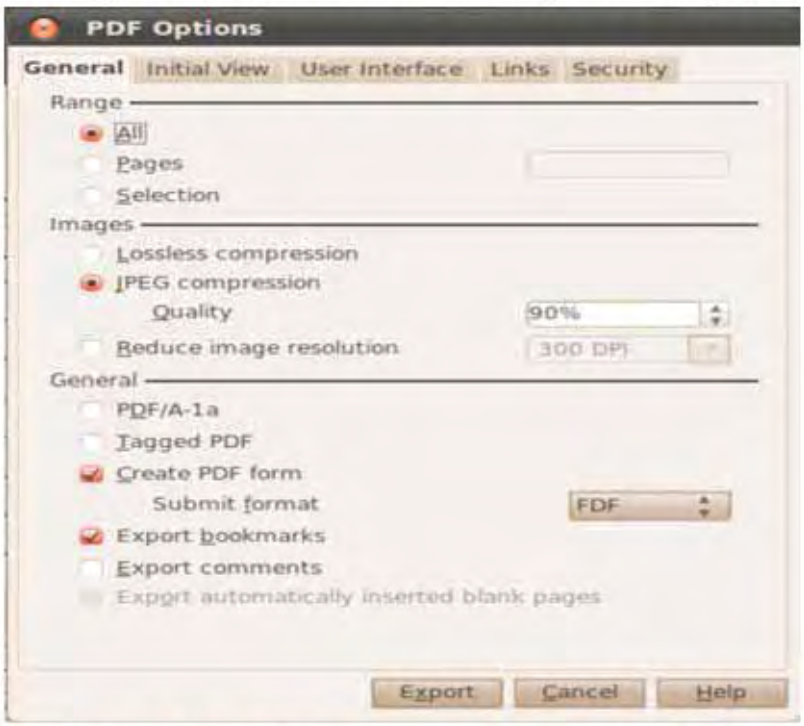


Figure 8.21 : Exporting document in PDF form

Further, the images of the documents are saved in jpeg format. Here you have options of lossless image compression as well as JPEG image compression with given percentage of quality. Select the Create PDF option, if it is not selected. Generally, these options are default options and automatically selected by the system. You may not disturb the default general option. You may choose Export button directly. By doing so, your document with the chart is exported in PDF form.

Copying the Chart

For copying chart to writer, impress and other docs you have to select chart, copy it and paste it as and when require. Perform following simple steps to copy the chart.

- Choose the chart with mouse keys.
- Right click and choose **Copy** or **Cut** option.
- You may go to **Edit** menu and choose required options also.
- Go to the destination document. And use **Paste** command.
- Save the destination file.

Deleting the Chart

To delete the chart, simply select the chart and press Delete button.

Printing the Chart

Many times you may require printing a chart and the worksheet content. Perform following simple steps to print the chart.

- Choose **File → Print.** You will see dialog box as shown in figure 8.22.
- Choose required printer.
- Click o OK button.

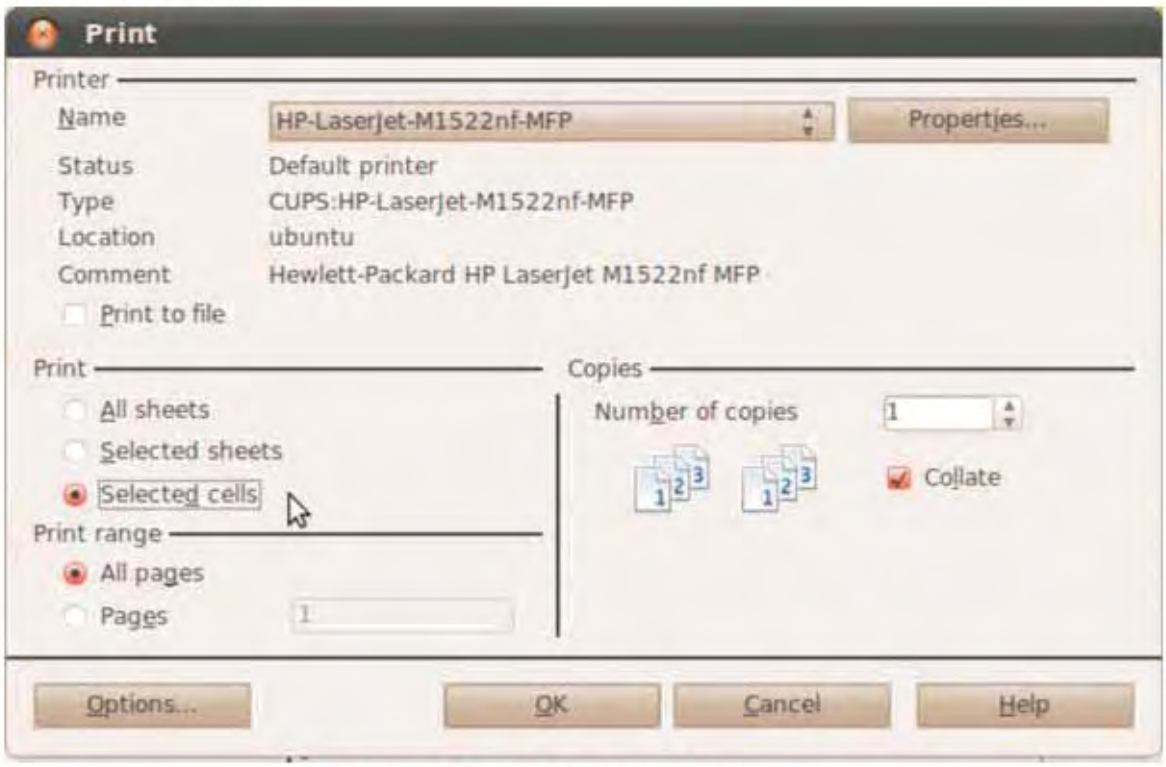


Figure 8.22 : Printing a document

Note that it is important to check the page preview before you print the document. This will give you a clear picture of how your document is going to be printed.

Inserting Other Multimedia Objects

Besides the charts, you can insert any multimedia object into a worksheet. The multimedia objects include objects of multiple media such as pictures and camera clips, video and movie clips, animated clips such as GIF files and 3D texts. For this, we normally use simple Insert menu commands. That is, to insert a camera clip or clip from your mobile, first you need to connect the device with your computer. You may use wired connection (if you have cord with your mobile of camera with USB connector) or Bluetooth technology. Once your computer has access of the object such as your photograph; select **Insert → Picture** to insert the desired picture. Similarly, you can insert a movie and an animated clip (or GIF file) into the Calc document.

You can add a 3D text in a worksheet also. The 3D text in Calc is known as graphical text art or Fontwork. An example is provided in figure 8.23.

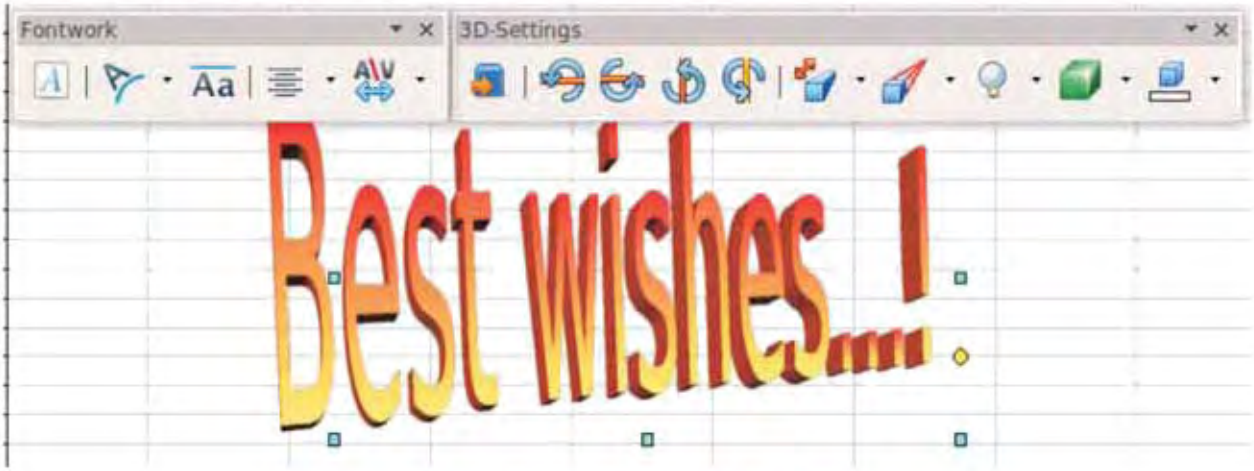


Figure 8.23 : Fontwork in Calc

To create a Fontwork object follow the steps given below.

- On the Drawing toolbar or on the **Fontwork** toolbar, click the **Fontwork Gallery** icon.
- In case, the Drawing toolbar or the **Fontwork** toolbar is not visible choose **View** menu and choose **Toolbars** option to enable the toolbar (**View → Toolbars**).
- In the **Fontwork Gallery** dialog, select a Fontwork style and click OK. You can use style from **Fontwork Gallery**, you can edit shape from Fontwork **Shape**, you can change height of letters, alignment of text, and character spacing of the text. You may use drawing toolbar to edit some more properties such as line colour, fill colour and fill style.
- The Fontwork object is inserted into your document. Double-click the object to enter your own text.
- Press Esc to exit text edit mode.

Once the Fontwork object is inserted; you can edit it by clicking it. The Fontwork toolbar is displayed. Edit the text. At the end, press Esc key when you finished the editing.

Summary

In this chapter we learnt about various charts, their importance and their purposes. We have also seen how charts are inserted and modified in the Calc spreadsheet package. There are several components of a chart. This chapter considers most common components of a chart and explains it in detail. We have also learnt how to refine and format such chart components for more effective presentation. The chapter has discussed two examples of chart namely: a column chart and a pie chart. We have also seen how a hyperlink can be set to a chart. The chapter also discussed mechanisms to print chart, to copy a chart, to delete a chart and to export a chart as portable document (PDF). At last, we have also seen how to insert a 3D and multi- media object in a spreadsheet document.

EXERCISE

1. Explain advantages of drawing a chart in Calc.
2. Explain various types of charts with one line description of each.
3. Explain how a chart can be inserted in Calc.
4. List the components of a chart wizard.
5. List elements of a chart with one line description of each.
6. How can we change the chart type once the chart is created ?
7. How to format chart axis?
8. Can we add hyperlink to the chart? How ?
9. Choose the correct option from the following :

(1) Which of the following option is used to print a chart ?

(a) Insert → Chart (b) File → View

(c) File → Print (d) View → Chart

(2) The charts in Calc may have how many of the following axis ?

(a) Two (b) Three (c) Two or three (d) Four

(3) Which of the following is used to enter 3D text in Calc ?

(a) Fontwork (b) Artwork

(c) Drawing work (d) Graphwork

(4) Which of the following is used to provide a link to the chart ?

(a) Activelink (b) Hyperlink

(c) Drawinglink (d) Connectionlink

(5) Which of the following showa us the preview of the chart ?

(a) Save as XHTML (b) Page preview

(c) Export chart (d) Any of these

(6) To fix the column or row in a data range - to make it absolute, which symbol is used ?

(a) # (b) $ (c) & (d) %

(7) To which of the entity can a Calc chart be linked ?

(a) To an existing document (b) To a new document

(c) Web page (d) All of these

LABORATORY EXERCISE

1. Find out your own marks in main subjects such as English, Science, Maths, etc. Also collect class highest marks in respective subjects. Draw a column chart comparing your marks with the highest marks in every subject.
2. Make a small survey in your neighbourhood about who is reading which newspaper. Collect data, arrange them in a clean form and prepare 3D pie chart based on this.
3. Consider a typical mathematical function such as Y= 2*X. Consider values of the X as given in the table given below.

Value of X	Value of Y
1	?
3	?
5	?
8	?
6	?

Do the followings :

(a) Enter these values in a Calc worksheet.

(b) Use formula to find out values of Y for each given value of X.

(c) Plot a line chart of the X and Y values together.

(d) Format chart as you like.

4. Consider following data related to household budget.

Category	July
Provision	1000
Milk	500
Vegetables	500
Servant	800
Electricity	1500
Gas	500
Fees	1500
Newspaper	100
Cable	200
Mobile	200

Do the followings :

(a) Enter these data into a spreadsheet and develop a pie chart.

(b) Once chart is ready format it as per your choice; e.g. change colour of any component, change font style, add/remove legends, add data labels to the chart and pull out a few segments.

(c) Export the chart as XHTML file. Open a web browser and verify how it looks.



Problem and Problem Solving

People in their day to day life solve lot of problems. Let us discuss one such process, consider that you are playing 'Hide and seek' with your friends. In this game our problem is to find the location where our friend has hidden. This can only be done only if we can predict the place of his/her hiding.

Let us take another problem, assume that we have to solve a mathematical equation to find the value of x, if we have been given equation $2x + 4 = 0$. To solve this problem we need to rearrange the equation as $2x = -4$, and then say that $x = -4/2$, the answer here is -2.

Let us take another example. We want to find the meaning of the word "Eloquent". We will need an English Language dictionary for the same. A dictionary contains thousands of words and its meaning. We are still able to find the desired meaning very quickly. Have you ever thought how we are able to do it so fast? We actually take advantage of the sequential order of words given in dictionary. We quickly eliminate the options that do not start with alphabet "E" and start looking for the words that start with alphabet "E"; we again use this elimination method to find words with second alphabet as "l". This elimination method is continued till we find the exact word that we are looking for.

The above example gave us a hint of what a problem can be? Problem solving can be a mental or machine (any systemic procedure) process that leads to the desired outcome required by the user. The examples discussed above also indicate that result of some problems is exactly predictable while for others predicting exact outcome may become difficult. Thus the problems can be classified into two types; well defined and ill-defined problems. In the above cases problem 2 and 3 are well defined problems, while problem number 1 is ill-defined. Note that well defined problems have clear goals and hence we can clearly define solution steps. It is fact that computers solve well defined problems only and therefore we will discuss only well defined problems in this chapter.

In the field of computers, solution of a given problem is a sequence of instructions given to a computer. Computer can solve variety of problems from the easiest to the most complex ones. To solve a problem it needs to be given a complete set of instructions. These instructions tell the computer what is to be done at every step. Remember one thing, computer does not solve a problem; it merely assists in solving the problem. We can solve any problem using the steps mentioned:

1. Define the problem.
2. Identify the input, output and constraint of the problem.
3. Identify different alternatives of solving the problem.
4. Choose the best possible alternative from the above list.
5. Prepare detailed stepwise instruction set of the identified alternative.
6. Compute results using this instruction set.
7. Check correctness of the answer obtained.

Steps 1 to 5 are performed by person who needs the solution, while step 6 and 7 are performed by a computer.

Assume that we need to find whether a given number is odd or even. The following set of instructions can be used to solve this problem.

1. Accept the number
2. Divide the number by 2 and find the remainder
3. If the remainder is 1, the given number is odd otherwise the number is even

The generalized solution to the problem is obtained using three different techniques mentioned below.

1. Pseudo code
2. Flowchart
3. Algorithm

The three steps given for finding whether a number is odd or even is known as Pseudo code. Pseudo means fake or simulated, we should consider the second meaning as it is more appropriate in our context. We can say that the steps mentioned to solve the problem is simulated code. Let us now discuss the other two techniques.

Flowchart

A flowchart is a technique in which we use pictorial representation of every action that we perform within the machine process that solves a problem. A set of symbols, showing different actions, is used to represent a flowchart. The symbols are also called components of flowcharts. We have a unique symbol corresponding to each action within a process. Let us discuss some commonly used components and its associated symbols.

Start and End : The Start and End components are used to show the beginning and the end of a flowchart. It is represented by an oval shape as shown in figure 9.1. The symbol is also called terminal symbol.



Figure 9.1 : Terminal Symbol

This symbol is used twice in a flowchart as it is used in the beginning as well as at the end of a flowchart.

Input-Output : Every problem needs an input, it then processes this input and generates an output. Consider the problem number 3 diacussed above, here we need to input one number, dividing this number by 2 and finding the remainder is the process and the decision whether the given number is odd or even is our output. Thus we need one symbol to show the input and one to show the output. The input and output in flowchart is represented by a parallelogram as shown in figure 9.2.

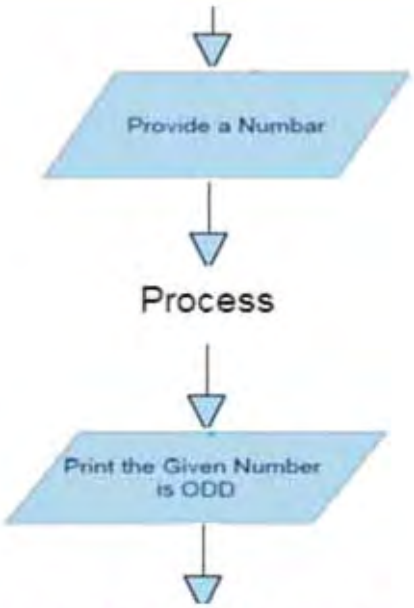


Figure 9.2 : Input-Output Symbol

Arrow : Observe that in both figure 9.1 and 9.2 we have shown arrows coming in and going out of the symbols. An arrow is used to show the sequence of the actions that are to be performed. It generally starts at one symbol and ends at another symbol. Thus there will be one arrow coming into the symbol and one arrow going out from the symbol.

Note that the start symbol will only have arrow going out, while the end symbol will have arrows coming in.

Process : Process is the core part of any solution procedure. A process is actually, a sequence of actions. To represent a process we use rectangle symbol as shown in figure 9.3.

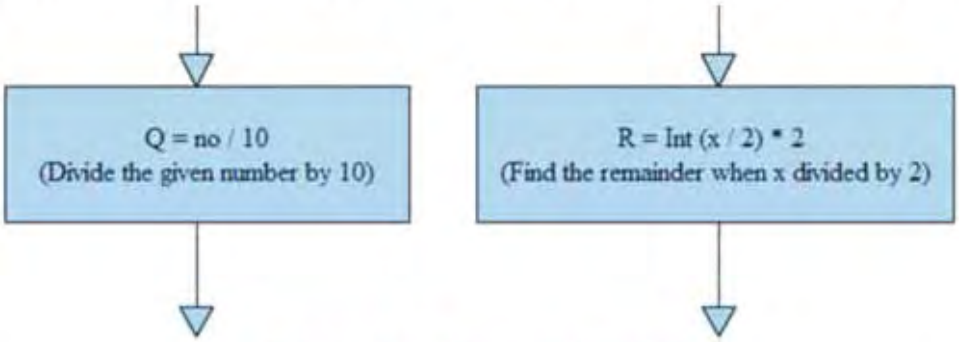


Figure 9.3 : Process Symbol

Normally a process in computers is either arithmetic or logical operation. The arithmetic operation we mean addition, subtraction, multiplication or division. Logical operations generally help in decision making, when used it answers in the form of true or false. For example we may ask a question such as is 10 greater than 5? The answer to this question is yes or true.

Decision : A logical decision of a process is represented by a diamond shaped symbol as shown in figure 9.4. It is also called a test symbol. A decision box is used when we want to alter the normal sequence of the solution (see in figure 9.4) or when a specific statement needs to be executed based on the result of decision.(See figure 9.5).

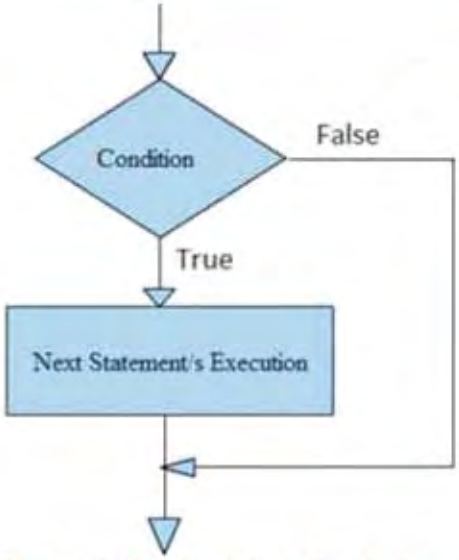


Figure 9.4 : Decision Symbol

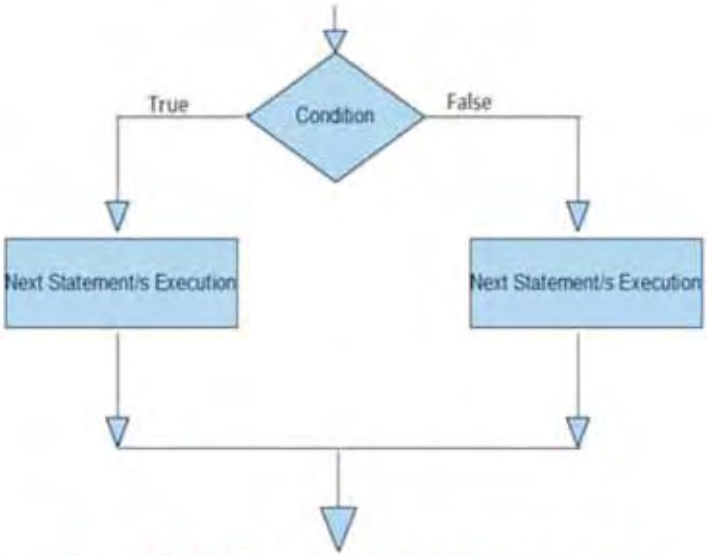


Figure 9.5 : Execution of different results

Sometimes we may require more than two alternatives to take a decision. In such a situation more than one decision box can be combined to form the required alternatives as shown in figure 9.6.

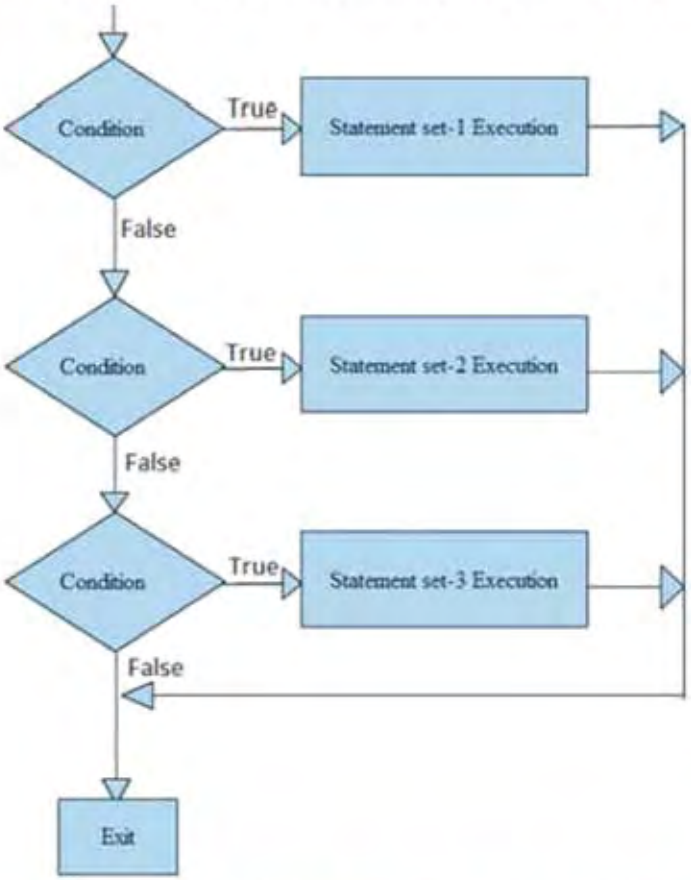


Figure 9.6 : Combining more than one decisions

As shown in the figure 9.6 first the condition will be checked; if it results into true, then first set of statements will be executed. In case of result being false next condition will be checked. This process will be repeated for all the given conditions. Note that when any condition results into true, the conditions following it will not be checked.

Connector: A circle symbol is used to represent a connector. At times it is possible that a flowchart becomes to large to fit into a single page or it may not be possible to use an arrow to link two processes. In such cases a connector can be used to join the two parts. Minimum two circles are required for a connection; one with an arrow coming into a circle and other with an arrow going out of the circle as shown in figure 9.7. To distinguish different connectors, different characters are used with a same character (alphabet) in a pair.

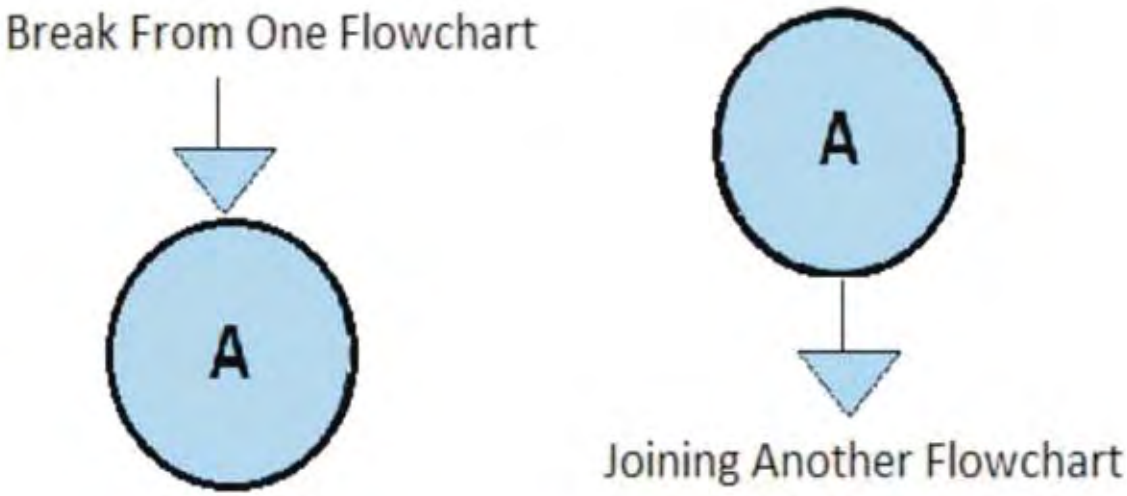


Figure 9.7 : Connector

Examples of Flowchart

Till now we have learnt what a flowchart is and also looked at some of the symbols used in it. Let us now solve some problems by creating flowcharts.

Problem 1 : The fitting charge of tiles is Rs. 10 per square feet. Now assume that the problem is to find the total cost of fitting tiles on the rectangular shaped town hall floor.

Solution :

To find the total cost of fitting the tiles we have to first need to find the area of the floor. Area of a rectangle, as we know, is product of length and breadth. Hence if we know the length and breadth we can find the area of a rectangle. The formula to compute area of the floor can be given as Area = Length * Breadth.

Next to find the total cost of fitting the tiles we multiple cost of fitting per square fit of tiles with the area of the floor. The formula to compute total cost thus is Cost = Area * 10.

To solve this problem we require four variables Area, Length, Breadth and Cost. Before we continue further let us first define the term variable as it will be used in all flowcharts.

Variable : An entity whose value can be assigned and changed during the execution of the process.

The flowchart to find cost of fitting the tiles on the floor is shown in figure 9.8.

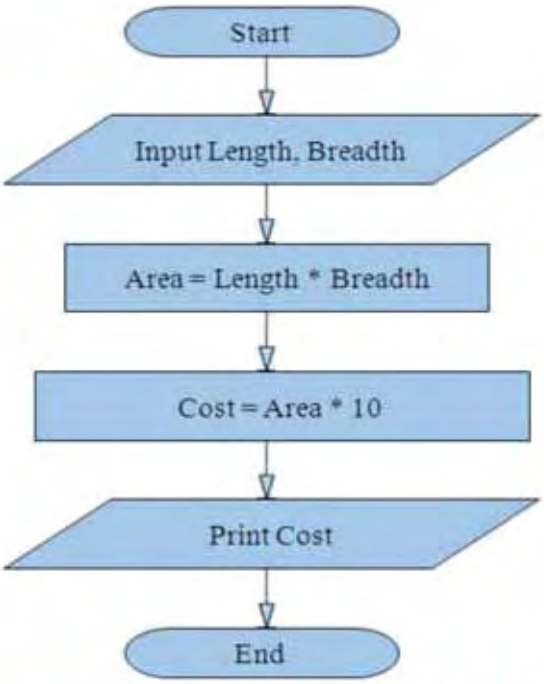


Figure 9.8 : Flowchart to find cost of tiles

Observe the value 10 in expression Cost = Area * 10, Here 10 is known as a constant. It is an entity whose value once assigned remains fixed and cannot be changed during the entire process. The solution that we have created in flowchart shown in figure 9.8 can only be used in the cases where the cost of fitting one square feet tile is Rs. 10. If the cost of fitting the tile changes, the above solution will not work. A generalized solution the above problem is given in figure 9.9.

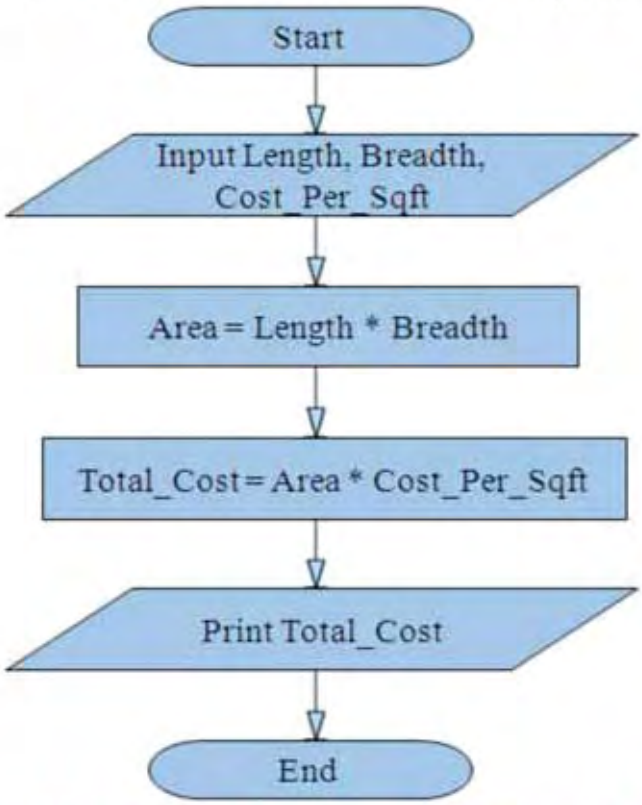


Figure 9.9 : Modified flowchart to find cost of fitting tiles

Observe the flowchart in figure 9.9; here we have used five variables instead of three. The variable Cost_Per_Sqft is now used to store the value of cost for fitting one square feet of tiles. With every run now we may change the value, and we will now be able to get the output. Table 9.1 lists some such outputs.

Length	Breadth	Area=Length*Breadth	Cost_Per_Sqft	Total_Cost=Area*Cost_Per_Sqft
50	50	2500	10	25000
25	75	1875	10	18750
45	35	1575	20	31500

Table 9.1 : Output of flowchart shown in figure 9.9

Problem 2 :

Suppose you have a very nice and round cricket ground at the city sports center. The authority wants to make fencing surrounding this ground. They also want to cover the total surface with a lawn. The authority wants to know how many meters the total fencing will run into? They also want to know the total surface area of the ground that needs to be covered with lawn.

Solution :

To solve this problem we need to find out the area and perimeter of the ground. The formula to find area of the circular ground is Area = $\pi * R^2$, similarly Perimeter = $2 * \pi * R$ (here value of Π is 3.14 and R refers to radius of the ground).

Thus to solve this problem we need three variable Radius, Area and Perimeter. The formula that we will use in the flowchart to calculate area and perimeter are given below:

Area = 3.14 * Radius * Radius, Perimeter = 2 * 3.14 * Radius

The flowchart for the said problem is shown in figure 9.10.

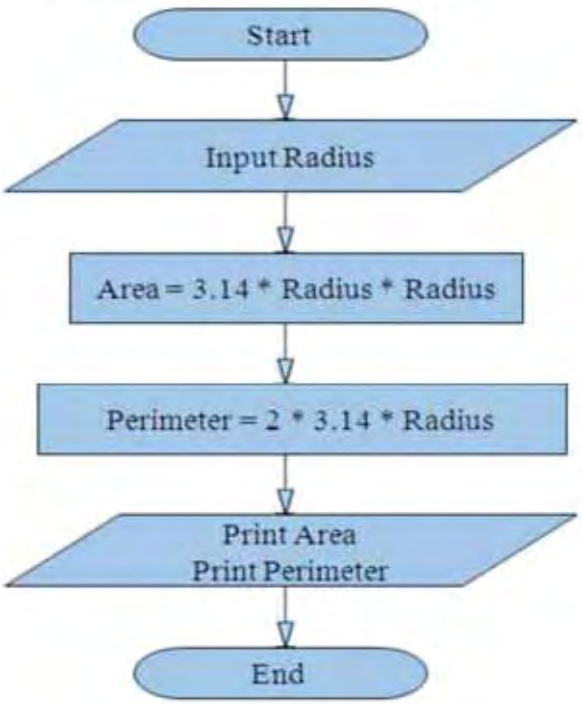


Figure 9.10 : Flowchart to find area and perimeter of a circle

Problem 3 : Ms. Jhanavi has borrowed a loan of Rs. 35,000 at the rate of 11.5 percent from a bank for 6 years. Calculate the simple interest that she will have to pay to the bank.

Solution :

The formula to compute simple interest 'I' on a loan of principle amount 'P' at an interest rate of 'R' for 'N' years is given as **I = (P*R*N)/100.**

To solve this problem we require four variables I, P, R, N and a constant value 100. Figure 9.11 shows the flowchart to solve the problem.

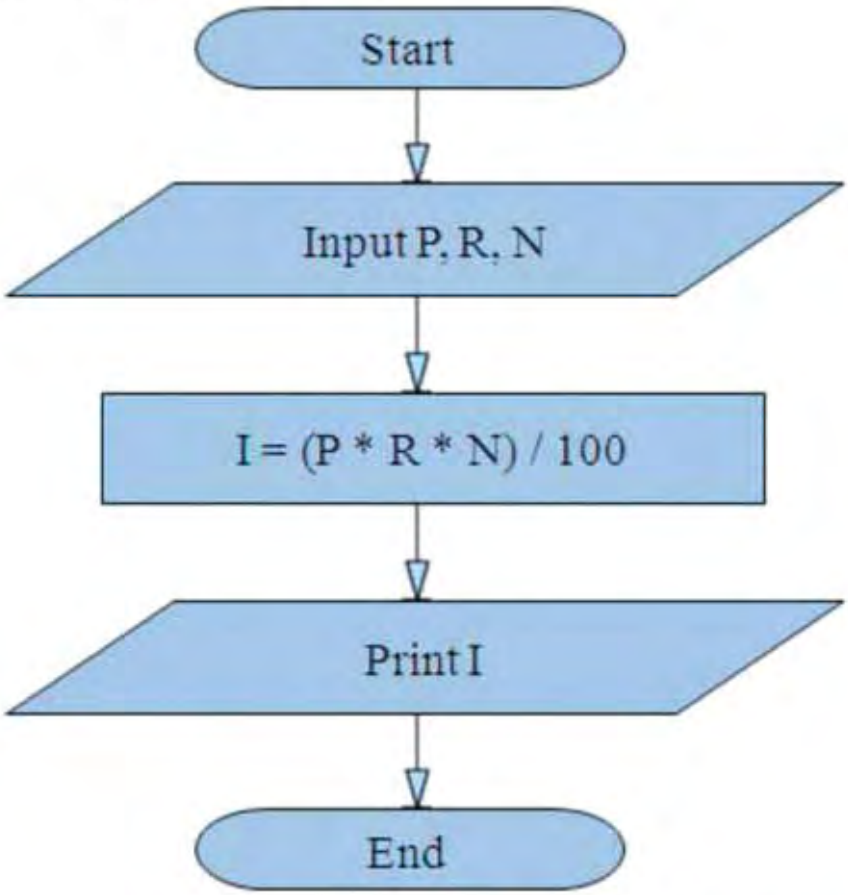


Figure 9.11 : Flowchart to find simple interest

Problem 4 : Assume that you want to find out the youngest student amongst two students. The input in this case will be the age of the student.

Solution :

The solution to the problem can be obtained by performing the steps mentioned below :

Take the ages of both the students; let it be assigned to variables Age_1 and Age_2. Now compare the value of Age_1 with Age_2. If both values are same, then we have two students with same age, hence both can be considered as youngest. If the value of Age_1 is less than Age_2 then first student is youngest. Otherwise second student is youngest. Figure 9.12 shows the flowchart to solve this problem.

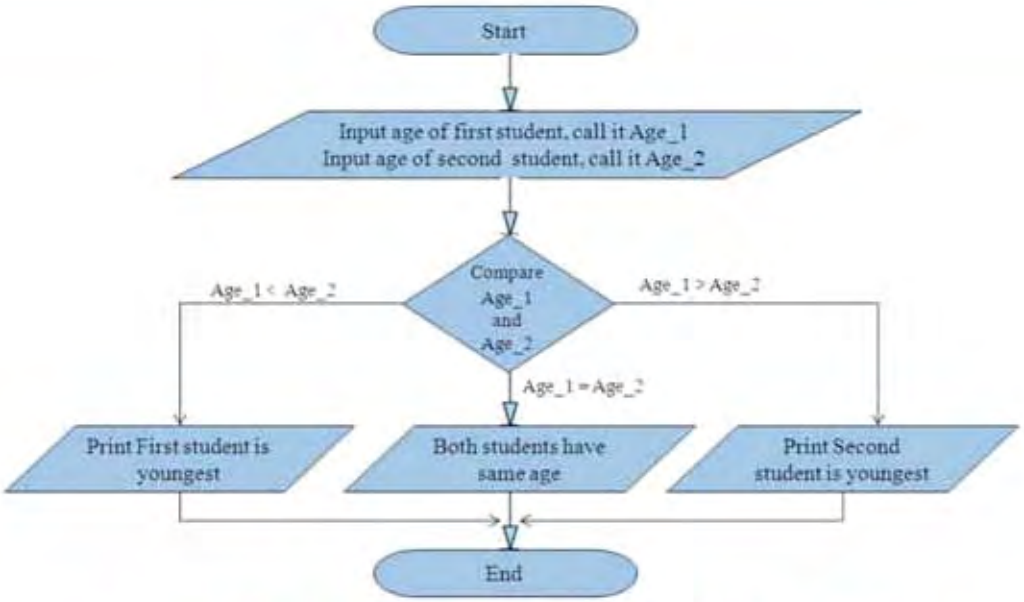


Figure 9.12 : Flowchart to find youngest of two students

Problem 5 : Let us now try to find youngest amongst three students.

Solution :

Take the ages of three students; let it be assigned to variables Age_1, Age_2 and Age_3. Now compare the value of Age_1, Age_2 and Age_3. If value of all the ages is same then, we have three equal aged students. Hence we can assign all three as youngest.

Otherwise compare Age_1 and Age_2. If the value of Age_1 is less than Age_2; then compare Age_1 and Age_3 if still Age_1 is less than Age_3, Then first student is youngest.

In the comparison made above, if Age_2 is less than Age_1, then compare Age_2 and Age_3 if still Age_2 is less than Age_3, Then second student is youngest. Otherwise third student is youngest. Figure 9.13 shows the flowchart to solve this problem.

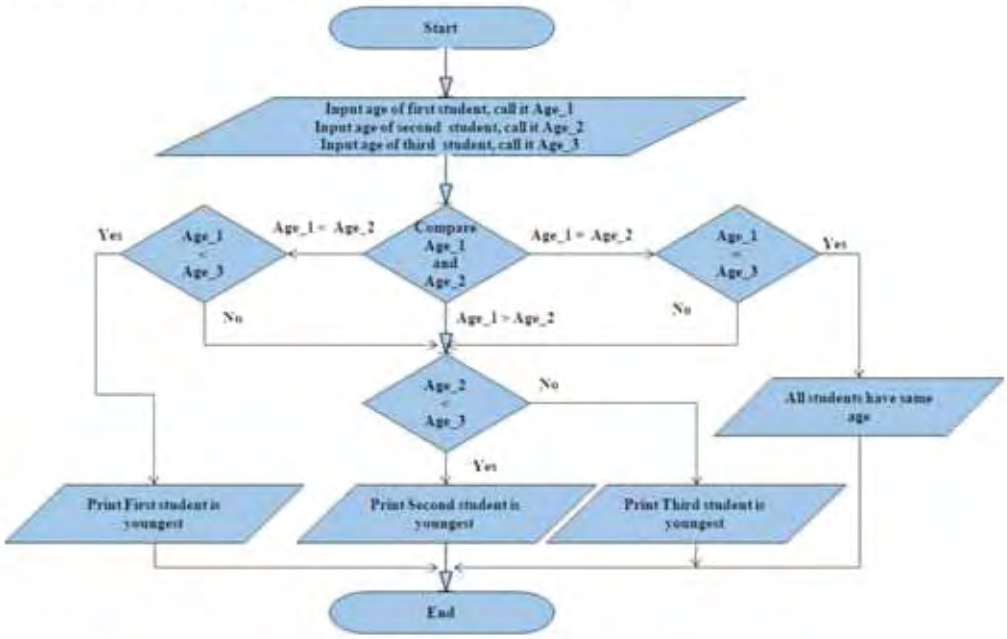


Figure 9.13 : Flowchart to find youngest of three students

Problem 6 : Let us further modify problem 5 to find the out youngest student from a class with any number of students in it. The steps mentioned below can be used to solve our problem.

Step 1: Choose first student, whatever his age, assume that it is MINIMUM.

Step 2 : Now choose another student and compare his/her age with the one that has been mentioned as having minimum age.

Step 3: If new student chosen is younger, we now make his/her age as MINIMUM and discard the old student's age.

Step 4: If both the students have same age then we discard the new student's age and assume that old student's age is still MINIMUM.

Step 5: We repeat step 3 and step 4 till we get the student that has youngest age. When we repeat the same set of statements for more than one time, it is called loop.

Figure 9.14 shows the flowchart to solve this problem.

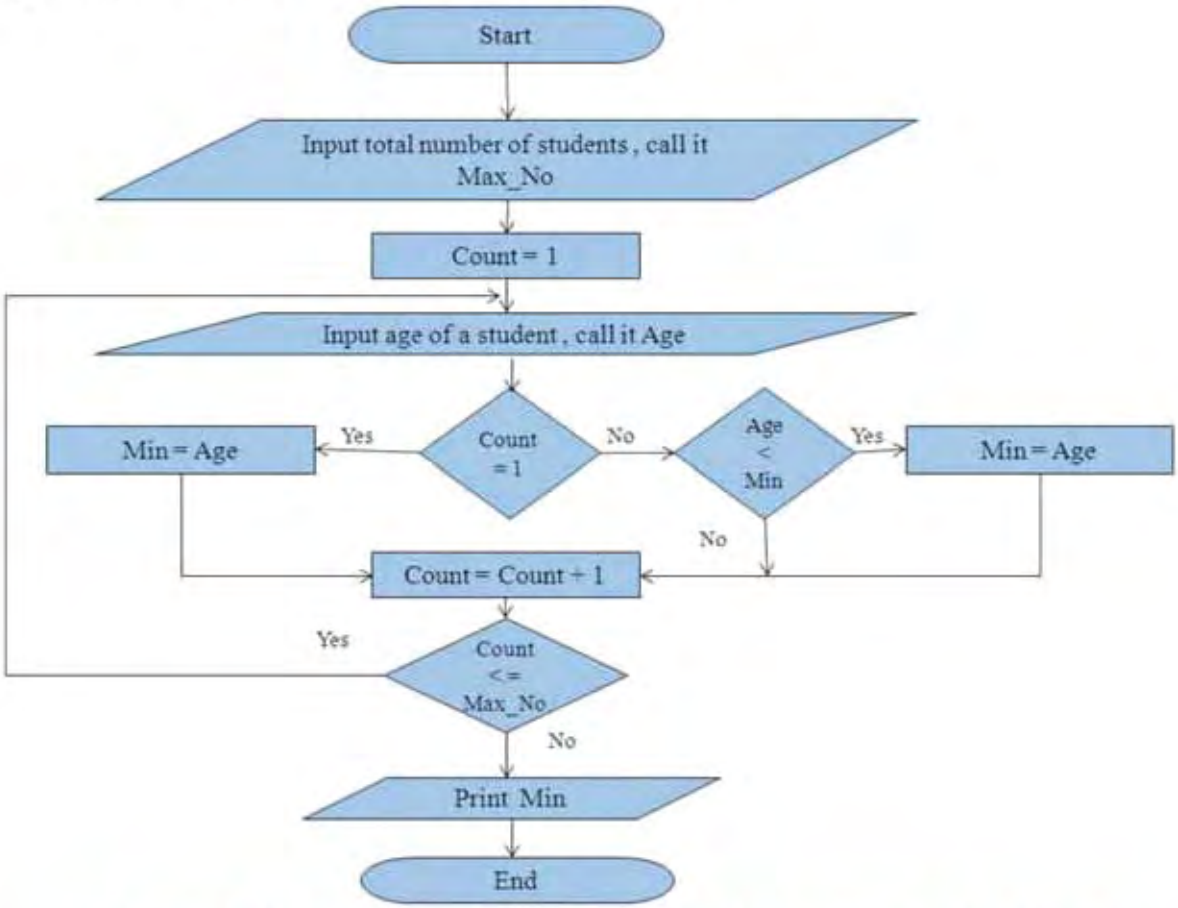


Figure 9.14 : Flowchart to find youngest of any number of students

Problem 7 : Let us now solve one mathematical problem. Assume that we want to find out sum of first 50 odd numbers. That is we want to calculate the sum of 1 + 3 + 5 + 7 + …….. + 99.

Solution :

Assume that you have two boxes, one in your left hand and other in your right hand. Now put one marble in the left box, and then empty it in the box of your right hand. Now again put three marbles in box in your left hand and transfer the marbles in the box of your right hand. What will

happen ? The box in your right hand contains a total of 1 + 3 marbles. If we repeat this process with 5, 7, ... 99 marbles. We will get sum of 1 + 3 + 5 + 7 + ... + 99 marbles. Note that we had assumed that the box in our right hand was initially empty.

With similar analogy, we have taken two variables, numb and sum. The variable numb is initialized with value 1 and the variable sum is initialized with 0 (similar to an empty box). The variable numb is incremented by 2, at iteration of the loop, and is added to sum. Thus the value of variable numb becomes 1, 3, 5, ... till 99. The statement sum = sum + numb keeps on adding the values of variable numb into sum till the loop ends. Figure 9.15 shows the flowchart of the problem.

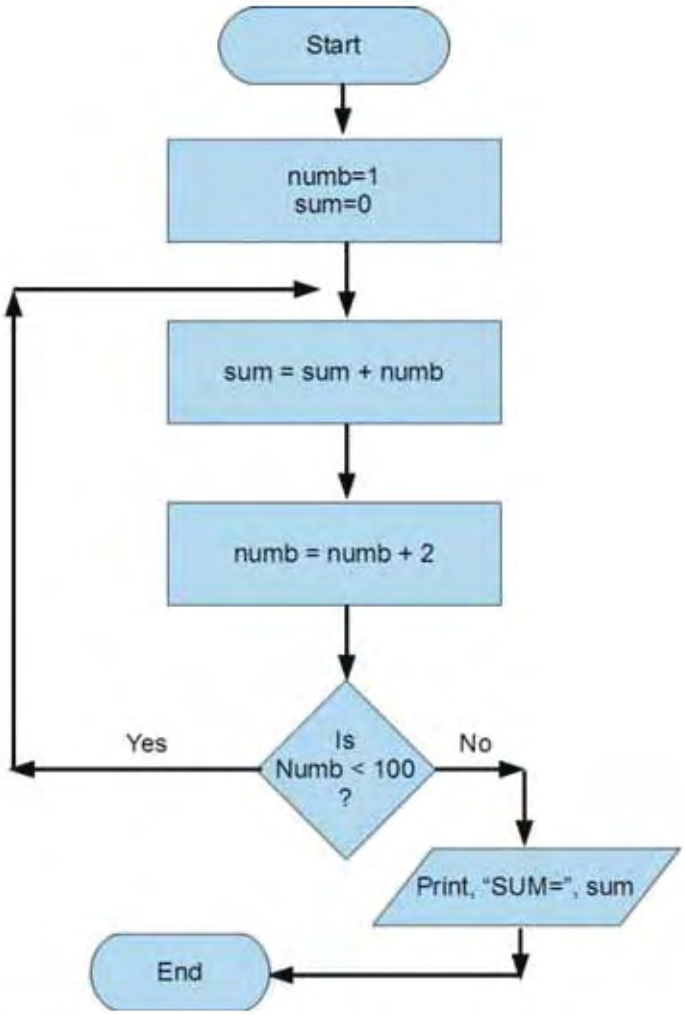


Figure 9.15 : Flowchart to sum of odd numbers between 1 to 99

Problem 8 : Let us now have a look at two different ways of exchanging values of two variables.

Solution :

Take two variables say A and B. If we replace the value of either A or B by any other value, the original value stored in them will be lost. Therefore we have used an extra variable C. To make sure that value of A is retained we first transfer it to C. Then value of B is transferred to A and then saved value of A (saved in C) is transferred to B. Figure 9.16 shows the flowchart of this solution.

Figure 9.17 gives a flowchart of the same problem, but it does not use an additional variable. Rather it performs addition and subtraction to interchange the values.

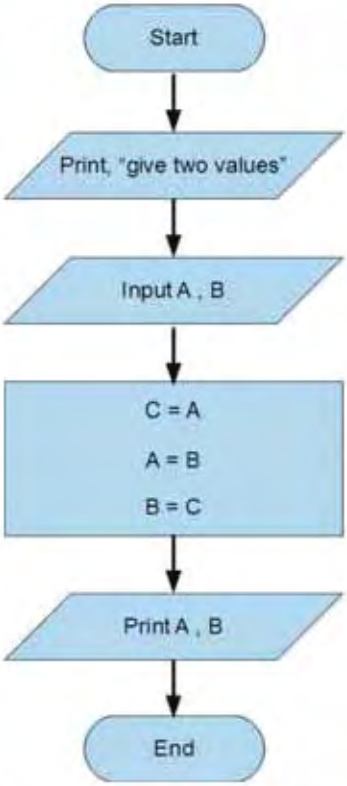


Figure 9.16 : Swapping with extra variable

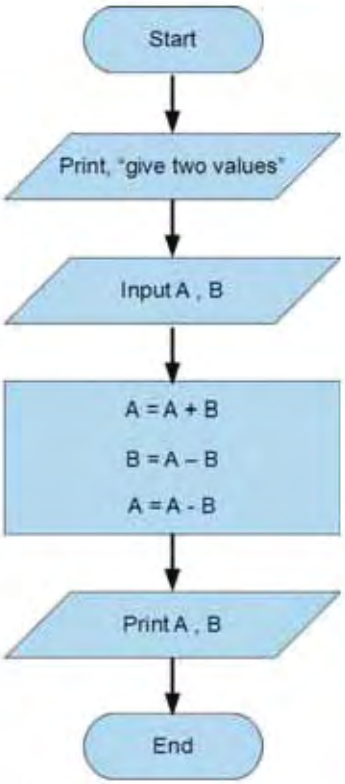


Figure 9.17 : Swapping without extra variable

Advantages and Disadvantages of Flowchart

Whatever tools we use they have some advantages and disadvantages. Flowchart is also not an exception. It also has some benefits and drawbacks.

Advantages of Flowchart :

1. As it provides pictorial representation of the problem solution, it is easy to understand.
2. It provides a convenient aid for writing computer instructions.
3. It assists the programmer in reviewing and correcting the solution steps.
4. It helps in discussion of different methods for solving a problem. This in turn provides us a facility to compare them accurately.

Disadvantages of Flowchart :

1. Drawing a flowchart for a large and complex problem is time consuming and laborious.
2. As flowchart consists of symbols, any changes or modification in the program logic will most of the times require a new flowchart to be drawn.
3. There are no standards specifying the details of data that should be shown in a flowchart. Hence flowchart for a given problem although with similar logic may vary in terms of data shown.

Algorithm

Your friend has reached your home and is now interested in coming to your school. He/She calls you and asks for directions to reach your school. To instruct him/her on how to reach the school you will first need to have some route locations in mind. Assume that you have a route map as shown in figure 9.18 to assist you.

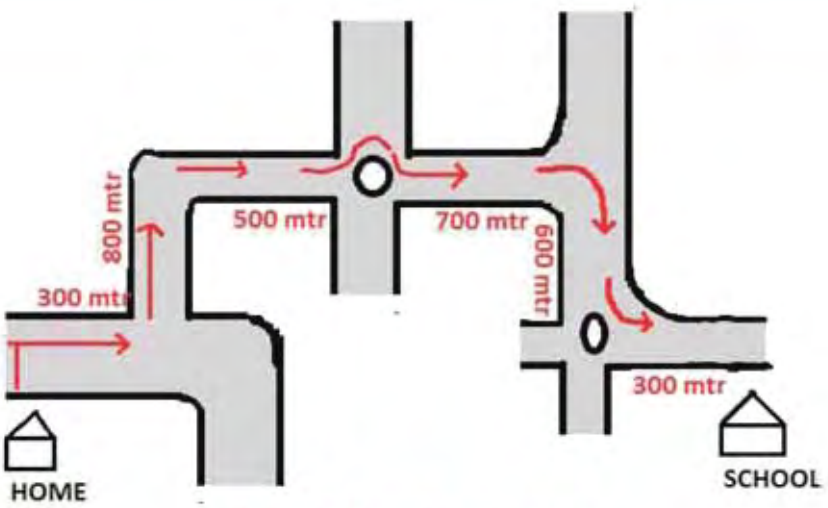


Figure 9.18 : Route Map

Solution :

Using the route map given in figure 9.18, you will be able to guide your friend step by step as mentioned below:

Step-1 : Start from my home and take right and go for 300 meters.

Step-2 : Now take a left turn and go straight for another 800 meters.

Step-3 : Take a right turn and walk for 500 meters, you will encounter a four cross roads with a circle.

Step-4 : Do not take turn at circle; instead keep on walking in the direction you were walking till another 700 meters.

Step-5 : Now take a right turn; after 600 meters you will reach another cross roads with an oval shaped fountain in the middle of cross roads.

Step-6 : Take a left turn from at this junction and walk for 300 meters.

Step-7 : You will be able to see my school on the right side of the street.

From the above example we can say that the problem of reaching your school is solved by following a sequence of steps. In computer science, an algorithm refers to a step by step procedure for solving a particular problem. An algorithm is written in a natural language like English. Algorithms once written then can be easily converted into computer programs. Let us now learn to write some algorithms.

Problem 1 :

Write an algorithm to find summation of numbers divisible by 11 within the range of 1 to 100.

Algorithm :

Step-1 : Start

Step-2 : Take two variable Num and Sum.

Step-3 : Initialize Num = 1 and Sum = 0.

Step-4 : Check whether Num is divisible by 11? To do this we have to find remainder when Num is divided by 11. Store the remainder in another variable say R.

```
R = Num - int(Num/11) * 11
```

Step-5 : If R is not equal to 0 go to Step-7.

Step-6 : Sum = Sum + Num

Step-7 : Num = Num + 1

Step-8 : If Num < 100 go to Step-4.

Step-9 : Print Sum

Step-10 : End

Explanation :

To find the remainder, we first divide the number stored in Num by 11 and then we take its integer part. For example if the value of Num is 39 then dividing it by 11, we will get 3.54545455. If we take its integer part, it will be only 3. The portion after decimal point will be removed. When the integer part is multiplied by 11, we will get 33. If this 33 is subtracted from the original number that is 39, we will get a value 6. This value is the remainder. Note that we have assumed a process called int which gives integer part of the number.

Step number 1 to 4 is self explanatory. Step-5 checks the remainder using the technique mentioned above. If the remainder is not equal to 0, it means that Num is not divisible by 11and we will proceed to take the next number. If Num is divisible by 11, we will jump to step-6. We add the value of Num to Sum and then proceed to step-8. In step-8 we check whether Num is less than 100 or not. This check is used to exit from the loop. We will thus continue till value of Num is less than 100. Finally we will print the value of Sum.

Problem 2 :

Write an algorithm to find compound interest for given, principal amount, time period and rate of interest, when interest is calculated every year.

Algorithm :

We know that the compound interest can be calculated using the formula

$$CI = P * (1 + R/100)^N - P$$

Here P is Principal Amount, R is rate of interest and N is time period in years.

Step-1 : Start

Step-2 : Take four variables P, N, R and CI.

Step-3 : Assign values to P, N and R.

Step-4 : Calculate compound interest using $CI = P * (1 + R/100)^N - P$

Step-5 : Print value stored in CI

Step-6 : End

Problem 3 :

Find out the total weekly pay to be given to an employee based on the number of hours he/she has worked. To get a solution here we need to know the pay per hour that is to be given to the employee.

Algorithm :

We want to find weekly pay, for this we need two inputs, number of hours worked and salary per hour.

Step-1 : Start

Step-2 : Take three variables No_Of_Hrs_Worked, Pay_Per_Hour and Weekly_Pay.

Step-3 : Assign values to No_Of_Hrs_Worked and Pay_Per_Hour.

Step-4 : Calculate weekly pay using formula

Weekly_Pay = No_Of_Hrs_Worked * Pay_Per_Hour

Step-5 : Print value stored in Weekly_Pay

Step-6 : End

Summary

In this chapter, we have learnt about the two most used techniques of problem solving, namely, flowchart and algorithm. We saw different symbols that can be used to draw a flowchart. The user needs to decide between using either a flowchart or an algorithm to solve a problem.

EXERCISE

1. Which are the basic components of a flowchart ?
2. Which kind of problem is solved by computers ?
3. Which types of operations are parts of the computer process ?
4. Which are the operations constitute Arithmetic operation ?
5. Decision box is used for which type of operation ?
6. How do you define a variable ?
7. What do you understand by algorithm ? How it is differ from flowchart ?
8. Choose the correct option from the following :

   (1) Which of the following symbol is used to begin a flow chart ?

   (a) 

   (b) 

   (c) 

   (d) 

(2) Which of the following refers to a list of instructions in a proper order to solve a problem called ?

(a) Algorithm (b) Flowchart (c) Sequence (d) Roadmap

(3) Which of the following symbol is used to test conditions in a flowchart ?

(a) Diamond (b) Circle (c) Arrow (d) Square

(4) Which of the following symbol is used to represent output in a flowchart ?

(a) Square (b) Circle (c) Parallelogram (d) Triangle

(5) Which of the following is the standard terminal symbol for a flowchart ?

(a) Circle (b) Diamond

(c) Rounded Rectangle (d) Square

(6) Which of the following refers to the purpose of Algorithm and Flowchart ?

(a) Know the memory capacity

(b) Identify the base of the number system

(c) Direct the output to the printer

(d) Specify the problem completely and clearly

(7) Which of the following is not a problem solving technique ?

(a) Pseudo code (b) Flowchart (c) Algorithm (d) Sequence

(8) Which of the following is a pictorial representation of a problem solving technique ?

(a) Pseudo code (b) Flowchart

(c) Algorithm (d) Computer program

(9) An arrow symbol in flowchart is used to show

(a) The flow of an action

(b) The sequence of action

(c) The start of actions

(d) The completion of an action

(10) Which of the following refers to the core part of any solution ?

(a) Input (b) Output (c) Process (d) Algorithm

(11) Which of the following symbol represents a Process ?

(a) Rectangle (b) Square (c) Circle (d) Diamond

(12) Which of the following is used to distinguish different connector pairs in flowchart ?

(a) Arrows are used

(b) Alphabets or other characters are used

(c) Circles are used

(d) Diamonds are used

LABORATORY EXERCISE

Draw a flowchart and write and algorithm to perform the following operations :

1. Convert the given meters into centimeters.
2. Convert the given centigrade to Fahrenheit.
3. Find minimum of the given two numbers.
4. Find minimum of the given two numbers and check whether the minimum number is odd or even.
5. Find minimum of the given three numbers.
6. Check whether a given number is divisible by other given number.
7. Find average of three numbers.
8. Given total marks of student find whether the student has passed or failed. (If total marks are less than 35 then the student is declared as fail.)
9. Given total marks of student find what grade the student has obtained. (If total marks are less than 35 then the student gets D grade. If total marks are between 35 and 60 then C grade, if total marks between 60 and 70 then B grade, above 70 A grade)
10. Calculate the compound interest on a loan amount of Rs. X taken a t the rate of R% for N years.
11. Find the summation of digits of a given number. For example if the given number is 8327 the output will be 20 (8 + 3 + 2 + 7)



Introduction to C Language

In chapter 9 we have learnt what is a flowchart and algorithm. We also learnt how to develop an algorithm. The fact is that flowchart and algorithm form basic solution to any problem. We then convert this flowchart or algorithm into a program. For writing programs many languages are there. C language is one of them. In this chapter we will be introducing you to C language.

Need of Programming Language

The first question we might ask is what is the need of programming language? Why not write program in the languages that we already know how to write and read in like English or Gujarati. Well the answer is, a sentence in such languages may not have just one meaning. This does not work well with computers. For computers to work properly each sentence has to be clear and precise.

Programming language when used allows us to write an instruction that has only one meaning. It consists of set of predefined rules. These rules form syntax of that language. Hence learning a programming language is only learning new syntax to represent the instructions we want to give to the computer. It is like learning new grammar when we learn new language.

Need of Translator

Computers however do not understand the language that we speak neither does it understand the programming language. It only understands 0 and 1. The problem discussed here is similar to the problem that we face at times in our lives. Consider that there are two persons, person X and person Y who don't know each other's language. Person X knows Gujarati while person Y knows Hindi. How would these two persons communicate? The option is third person Z who knows both languages. So now when person X wants to communicate a message to person Y, he communicates the message to person Z in Gujarati first. Person Z then translates it to Hindi and then communicates the message to person Y. Same process is repeated in reverse order when person Y wants to communicate with person X. Here person Z is known as translator and the process of converting one language to another is known as translation.

The problem of computers not understanding our language is solved by using software programs called translators. This translator is known as compiler. The process of translation has been discussed at the end of this chapter.

Program and Characteristics of Program

A program can be defined as finite set of precise and clear instructions given to a computer for performing a predefined task. The process of writing these step by step instructions using a chosen language is known as programming.

A good program should possess following characteristics :

1. A program must end after finite number of steps.
2. The instructions of program must be precisely defined, i.e. it should not have multiple meaning.
3. All the instructions must be effective, i.e. they should be carried out exactly.

4. A program may take zero or more inputs.

5. A program may produce one or more outputs.

As can be seen these characteristics are similar to characteristics of an algorithm. Example 10.1 gives a sample C program. Figure 10.1 shows the code listing and output of example 10.1 as can be seen in SciTE editor. We will learn how to use SciTE editor at the end of this chapter.



Figure 10.1 : My first C Program

This program is used for printing a message "Welcome to the world of C programming using SciTE". Here main() is known as user defined function, while printf( ) is known as inbuilt or library function. User defined functions are the functions that a user creates as per his or her requirement. The functions already available in C that a user can use in his or her program are known as inbuilt or library functions.

Before we start discussing in detail about various C language components let us have a look at another C program shown in figure 10.2 and its output shown in figure 10.3.

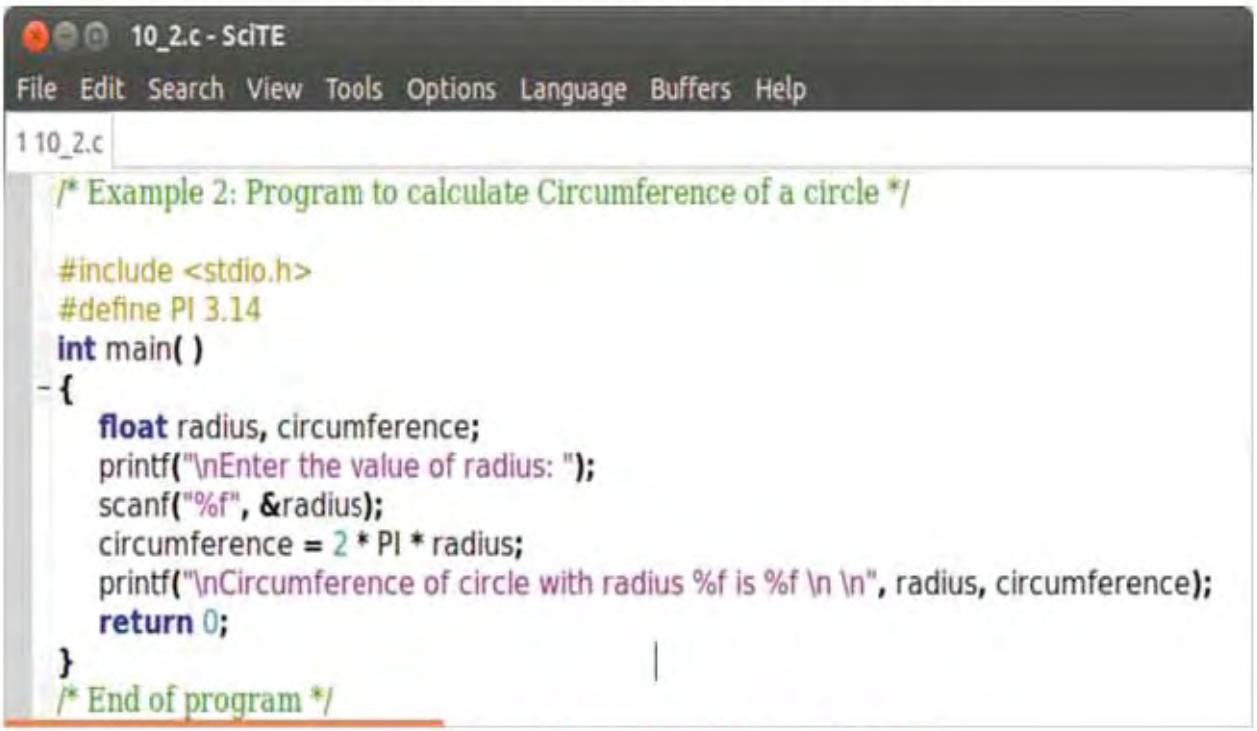


Figure 10.2 : Code listing of Example 10.2

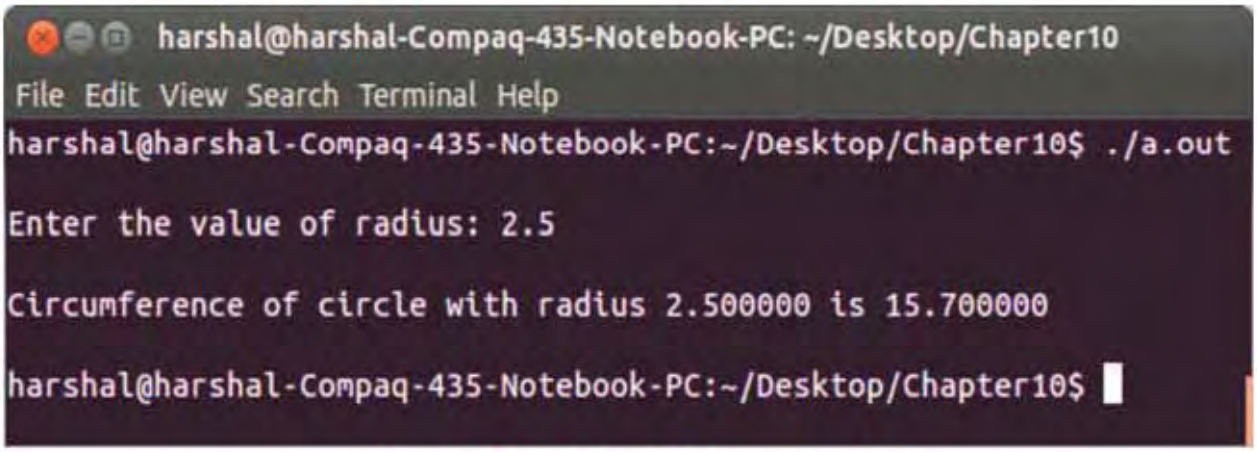


Figure 10.3 : Output of Example 10.2

Explanation

The program given here is used for calculating circumference of a circle for a given radius.

- The first statement is a comment that indicates the purpose of a program.
- The second statement instructs the compiler to include the contents of header file called "stdio.h". This file contains information about the inbuilt functions to be used.
- The third statement defines a symbolic constant PI that has value 3.14. This statement is known as pre processor directive. It instructs the translator to replace all occurrence of PI with 3.14 before executing the program.
- The fourth statement is a user defined function main(), a must for all executable C programs. Here this function returns and integer value hence the statement int main().
- The fifth statement is beginning of function main( ).
- The sixth statement defines two variables radius and circumference. These variables are capable of storing real values. An entity in C that is capable of storing different values is known as variable.
- The seventh statement uses an inbuilt function printf to print a message.
- The eighth statement uses inbuilt function scanf to read a value of "radius" from user.
- The ninth statement is a C expression; it is a formula for calculating the circumference of circle.
- The tenth statement prints a message along with "radius" and "circumference" on the screen.
- The eleventh statement provides normal exit from the function. If we don't write this statement we will get a warning message "Function should return a value". This may not affect the working of the program, but a normal exit is always a better option.
- The twelfth statement indicates the end of the function main( ).

As can be seen from example 10.1 and 10.2 main() function forms an integral part of C programs. Rather we should say that all executable C programs will have one user defined function named as main(). The execution of every C program starts from the first open curly bracket when main() is encountered.

Structure of C Program

As seen from the previous examples a C program is a set of blocks called functions. A function is made up of one or more statements used for performing a predefined task. A complete C program structure is given in figure 10.4. Let us discuss these sections in detail.

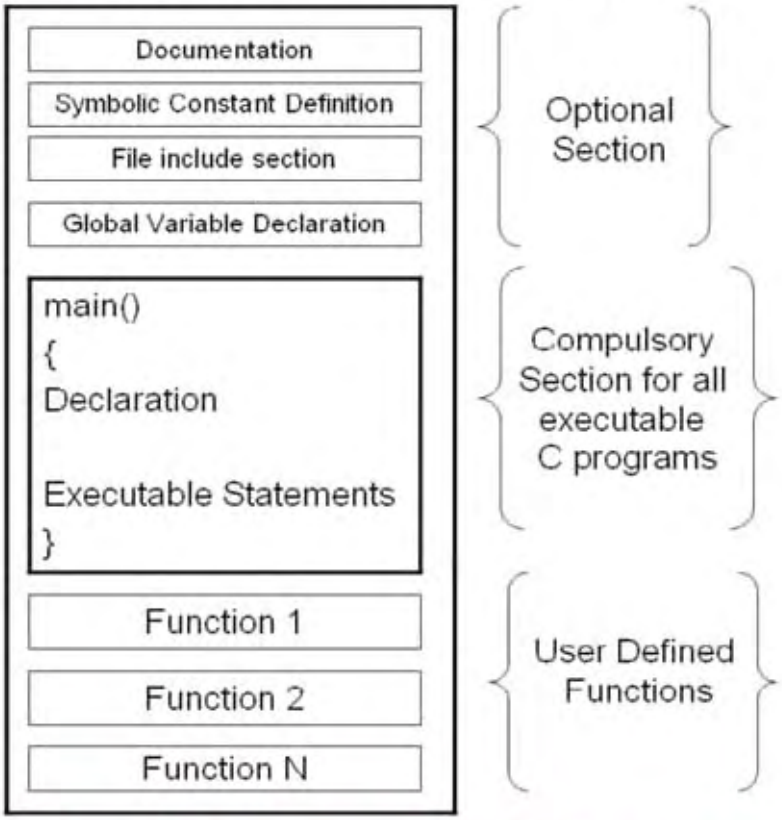


Figure 10.4 : Structure of complete C program

Documentation Section

This section is an optional section. As the name indicates it is used for the purpose of documentation. It is enclosed within /* and */. Whenever the text is enclosed between /* and */, it is considered as a comment in C. Comments are not processed by C compiler and are left as it is.

This section is normally used to tell about the purpose of program, author, date of creation etc. Comments in C program can be added anywhere. It is always a good practice to use comments within the functions as it improves the readability and understanding of the program.

Symbolic Constant Definition

As seen in example 10.2 we have used a symbolic constant PI in our program. To use such symbolic constant in our program we need to use #define as a prefix to it. Here #define is known as pre processor directive. It instructs the compiler to replace all occurrences of symbolic constants with the values specified against it. Normally symbolic constants are defined using capital letters. Doing this differentiates them from normal variables.

File include Section

C provides inbuilt or library functions. Some examples are pow( ), sqrt( ) etc. These functions have a predefined purpose like pow( ) is used for calculating value of x raised to given power,

sqrt( ) is used to find square root of a given number. We can use these functions in our program if needed. To use them we have to include files that hold information about these functions. These files are known as header files in C. The extension of header file is ".h". We use the syntax #include <filename.h> to include header files in our program. Appendix IV gives the list of header files available in C and their uses.

Global Variable Declaration Section

C variables are governed by scope. A scope of C variable is decided by using opening and closing curly braces { }. The variables defined within curly braces are known as local variables. For example the variables "radius" and "circumference" used in example 10.2 are local variables of function main(). These variables cannot be used outside the scope. At times we need to use a variable in all the functions, such a variable is known as global variable. This variable is defined before defining all the functions.

Main Function

All C programs contain one function with the name main(). This is the function from where the execution of any C program starts. The control is first transferred to this function and from here rest of the operations are carried out.

User Defined Function

C provides us a facility of breaking a single program into set of small pieces. These pieces are known as functions. In this section we define all the additional functions used in our program. Example 10.3 shown in figure 10.5 shows a program that consists of two user defined functions.

10_3.c - SciTE

File Edit Search View Tools Options Language Buffers Help

1 10_3.c

```
/* Example 3: Program to subtract two numbers */

#include <stdio.h>

/* Prototype statement of function sub */
int subtract(int, int);

int main ( )
{
    int first, second, difference;
    printf("\nEnter the value of first: ");
    scanf("%d", &first);
    printf("\nEnter the value of second: ");
    scanf("%d", &second);
    difference = subtract(first, second);
    printf("\nThe value of %d - %d is %d \n \n", first, second, difference);
    return;
}
/* End of main program */

/* Beginning of user defined functions other than main( ) */
int subtract(int first, int second)
{
     int result;
     result = first - second;
     return(result);
}
/* End of function subtract( ) */
```

Figure 10.5 : Code listing of Example 10.3

Explanation

This program is used to calculate subtraction of two given numbers. Although it is a small program we have divided it into two user defined functions. The first function main() accepts the values to be subtracted and second function subtract() performs the actual subtraction.

Observe that we have written a prototype statement before actually using the function subtract. It is needed in C if we write a user defined function after main(). The main function declares three integer variables, and asks user to input value of two of them. These values are then passed to function subtract() by using statement difference = subtract(first, second). Here first and second are known as parameters of function subtract(). After encountering this statement the control is transferred to function subtract(). This function then subtracts the value and stores it in "result". The value of result is then returned to statement in main() and assigned to "difference". Finally main() function prints the value of subtraction and exits the program. The output of example 10.3 is shown in figure 10.6.

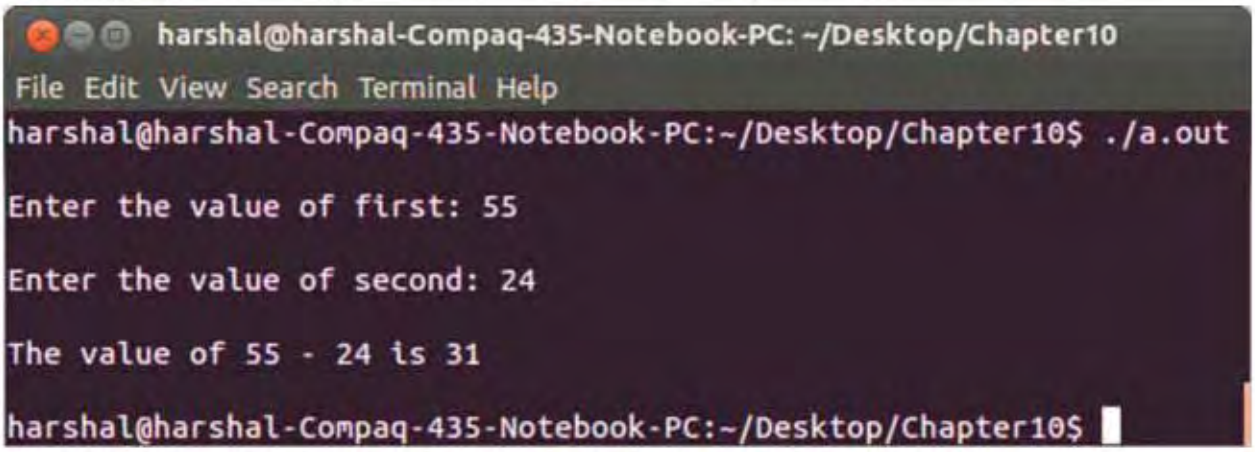


Figure 10.6 : Output of Example 10.3

The structure of C program shown in figure 10.4 although used by almost all programmers is not a standard. It is possible that some sections may not be there at all as can be seen in example 10.1. It is also possible to interchange certain section with others. For example user defined function section can be interchanged with main program section as can be seen in code listing 10.1. Here we have kept the user defined function before the main() function.

```
/* Example 4: Program to subtract two numbers */

#include <stdio.h>

/* Beginning of user defined functions other than main( ) */
int subtract(int first, int second)
{
        int result;
        result = first - second;
return(result);
 }
/* End of function subtract( ) */
```

```
/* Beginning of function main() */
int main ( )
{
      int first, second, difference;
     printf("\nEnter the value of first: ");
      scanf("%d", &first);
     printf("\nEnter the value of second: ");
      scanf("%d", &second);
     difference = subtract(first, second);
     printf("\nThe value of %d - %d is %d \n \n", first, second, difference);
      return;
}
/* End of main program */
```

Code Listing 10.1 : Code of Example 10.4

This program is similar to the one shown in example 10.3 except for two differences. First the function subtract() is defined before main() and second, we have not used the prototype statement. In example 10.3 on encountering the statement difference = subtract(first, second); the control flow goes below main(), in case of example 10.4 the control flow will go above main().

C Character Set

Recall that when we learnt a new language in school, we first had to learn its alphabets. These alphabets form a basic set in constructing words and sentences in that language. C language also has its own character set. The characters here can be divided into four categories.

1. Letter
2. Digits
3. White spaces
4. Special Characters.

Table 10.1 lists the characters that fall under these categories.

Letters	Digits	White Spaces
A to Z a to z	0 to 9	Blank Space
		Form feed
		Horizontal tab
		New line
		Vertical tab

Special Characters			
Operator	**Use**	**Operator**	**Use**
&	Ampersand	>	Greater than
'	Apostrophe	#	Hash sign
*	Asterisk	<	Less than
@	At the rate	-	Minus
\	Backward slash	( )	Parenthesis left / right
{ }	Braces left / right	%	Percent
[ ]	Bracket left / right	.	Period
^	Caret	+	Plus
:	Colon	?	Question mark
,	Comma	"	Quotation mark
$	Dollar	;	Semi colon
=	Equal to	~	Tilde
!	Exclamation	_	Under score
/	Forward slash	|	Vertical bar

Table 10.1 : C Character set

As can be seen in table 10.1, we have used characters that belong to English language. Hence now we would learn to use these characters as per C language syntax.

C Words

The character set given in table 10.1 is used to form words in C language. These words are used to construct a C statement. Set of logically sequential C statements thus is identified as a C program. A word in C is known as a token of C. Each character discussed in table 10.1 is itself a token. C basically identifies six types of tokens keyword, identifier, constant, string, operator and special character. Figure 10.7 shows how these tokens are used in C program.

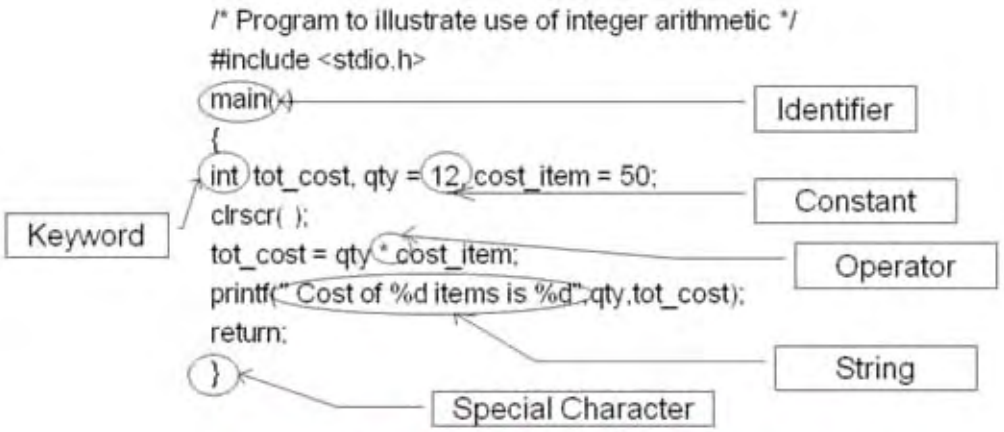


Figure 10.7 : Use of words in C program

Keyword

All of us must have used a dictionary at some point of time. We used it for finding a meaning of a predefined word of that language. C also has such predefined words. To be specific ANSI C standard supports 32 predefined words. These predefined words in C language are known as keyword. Each keyword has a predefined meaning associated to it. Table 10.2 gives the list of the keywords available in ANSI C.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Table 10.2 : ANSI C keywords

The keywords and their use have been discussed in subsequent chapters throughout the course of study.

Identifier

A word that a user can construct by making use of C character set is known as identifier. It consists of set of letters, digits and some special characters. Some example of identifiers are "difference", "main()", "PI", "float" etc. Here "difference" is name of a variable, "main()" is name of a function, "PI" is name of a symbolic constant and "float" is a predefined word. Before continuing further let us see what is variable.

Variable

C program generally accepts input, this input comes in a form of data. To store and manipulate a data we use memory space. The data within this memory space can vary based on the operations performed on it. The data within the memory space is referred by a name known as variable name. The data as it is capable of changing itself is called variable. To define a variable name in C we have to adhere to certain rules. These rules have been mentioned below :

1. Variable name cannot be same as keyword.
2. Variable name consists of letter, digit and under score. No other special character is allowed.
3. The first character of variable name must be a letter or under score.
4. The maximum length of variable name as per ANSI standards is 31 characters. However, it is dependent on compiler.
5. The variable names are case sensitive hence num, nuM, nUM, nUm and Num are considered as different variables.

Table 10.3 gives some examples valid and invalid variables of C language.

Valid
Double → Double is not same as keyword double.
INTEREST → Capital letters are part of C character set.
total_quantity → Underscore and characters are allowed.
mark100 → As first letter is character it is allowed.
_file → First letter can be underscore.
Invalid
char → Use of keyword not allowed.
Total Value → Blank space not allowed.
total&value → Special character not allowed.
10mark → First character has to be character or underscore.

Table 10.3 : Example of C variable

Constant

The entities of C that do not change its value throughout the execution of program are known as constant. C constants can be divided into different categories as shown in figure 10.7. Let us discuss these constants in detail.

Numeric Constant

The constant that store numeric value is known as numeric constant. The numeric constants are divided in two categories, integer constants and real constants.

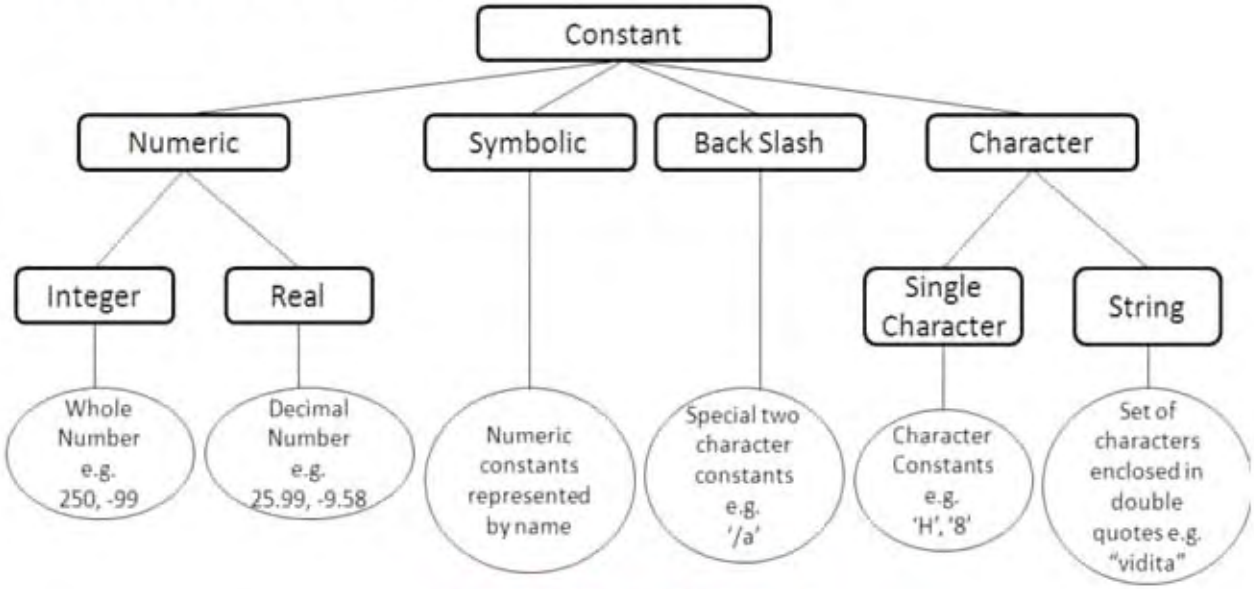


Figure 10.7 : C Constants

Integer Constant

Integer constant refers to whole numbers. We have three types of integer constants decimal, hexadecimal and octal. Example of various number systems is given in Appendix I.

The decimal constants use numeric digits from 0 to 9. We can use as prefix unary plus or minus with such numbers.

The hexadecimal constants use numeric digits from 0 to 9 and letters A to F. Here A to F refers to numeric values 10 to 15 respectively. The hexadecimal numbers have base 16. When using this value in C we differentiate it from decimal numbers by using 0x or 0X as prefix.

The octal constants use numeric digits from 0 to 7. The base of this number system is 8. When using this value in C we differentiate it from decimal numbers by using 0 as prefix. Table 10.4 gives a list of some valid and invalid integer constants

Valid
40000 → Valid decimal positive whole number.
-120 → Valid decimal negative whole number.
79999L → Valid decimal positive long whole number.
0xB5 → Valid Hexadecimal number.
0X79 → Valid Hexadecimal number.
045 → Valid Octal number.
Invalid
25,000 → comma is not allowed.
-100.0 → it is real constant.
Rs79999 → Alphabets are not allowed.
0xG → G is not part of hexadecimal numbers.
0X8,9 → Comma is not allowed.
096 → 9 is not part of octal numbers

Table 10.4 : Integer Constants

Real Constant

Real constant refer to decimal numbers that have fractional part. Example of real constant are 10.50, -65.75 etc. Real constant can also be represented by using scientific form. In this form a number is represented using a mantissa and exponent. For example the value 25.75 can be represented as 0.2575 e 2 or 0.2575 E 2. Here 0.2575 is known as mantissa and 2 is known as exponent, also "e" and "E" is equivalent. Table 10.5 gives a list of some valid and invalid real constants.

Valid
250.00 → Valid decimal fractional number.
295 → Valid decimal number. Automatically converted to 295.00
-15.55 → Valid negative decimal number.
-20.5e5 → Valid scientific value.
15E-2 → Valid scientific value.
Invalid
19,800.00 → comma is not allowed.
$786 → Special character not allowed.
-9.4 e 5.0 → Exponent should be integer.
51E -2 → Blank spaces not allowed.

Table 10.5 : Real Constants

Character Constant

Character constants as the name suggests contain characters. There are two types of character constants in C. Single character constant and String constant.

Single Character Constant

A single character constant is represented by using single character enclosed within single quotes. Some example of single character constants are 'H', 'v', '7', '$' etc. Here each character constant is associated with a numeric value known as ASCII (American Standard Code for Information Interchange) value. Appendix II gives detail of ASCII values associated with different characters.

String Constant

String constants are denoted by using sequence of characters enclosed within double quotes. Some examples of string constants are "C Language", "Bye", "V". String constants don't have any ASCII value associated with it. Here string constant "V" is not equal to single character constant 'V'. Here string "V" will be allocated two memory spaces while character 'V' will be allocated only one memory space. This is due to the fact that strings in C are ended by adding '\0', a null character.

Back Slash Characters

The single character constant as the name suggests use single character, while string character uses sequence of characters. C also provides a special character constant that uses two characters. These constants are known as back slash characters or escape sequences. They are known as back slash characters as the first character is always a back slash i.e. "\", while they are known as escape sequence characters as the output of these constants is not a character but white spaces. Similar to single character constants, these constants also have an ASCII value associated with them. Table 10.6 lists the back slash constant available in C along with their use and ASCII values.

Back Slash Character	Use	ASCII Value
\0	Inserting null value	0
\a	Inserting audible alert	7
\b	Inserting backspace	8
\t	Inserting horizontal tab	9
\n	Inserting new line	10
\v	Inserting vertical tab	11
\f	Inserting form feed	12
\r	Inserting carriage return	13
\"	Inserting double quotes	34
\'	Inserting single quote	39
\?	Inserting question mark	63
\\	Inserting back slash	92

Table 10.6 : Back Slash characters

The escape sequences are mostly used for formatting purpose. For example '\n' is used for inserting new line at the time of input or output. Throughout the book you will be able to see the use of these characters.

Symbolic Constant

Certain numeric and character constants can be defined using symbolic identifier. Such constants are known as symbolic constant. To define a symbolic constant we use the given syntax:

#define identifier value

Here #define is known as a preprocessor directive. Identifier is symbolic name of the constant that we want to define, and value is the constant itself. Some examples of symbolic constants are

#define PI 3.14

define MAXVALUE 100

#define f float

The first statement here defines a symbolic constant called "PI" that refers to a real value "3.14". Second statement defines a symbolic constant called "MAXVALUE" that refers to an integer value "100" and the last statement defines a symbolic constant called "f" that refers to a keyword "float" in C language.

The preprocessor directive statement instructs the compiler that the occurrence of the symbolic constant used in program should be replaced by the constant value specified in the definition. Example 10.5 shows a program to find surface area of a sphere and uses preprocessor directives. Figure 10.8 shows the code listing of example 10.5 while Figure 10.9 shows the output of the program.

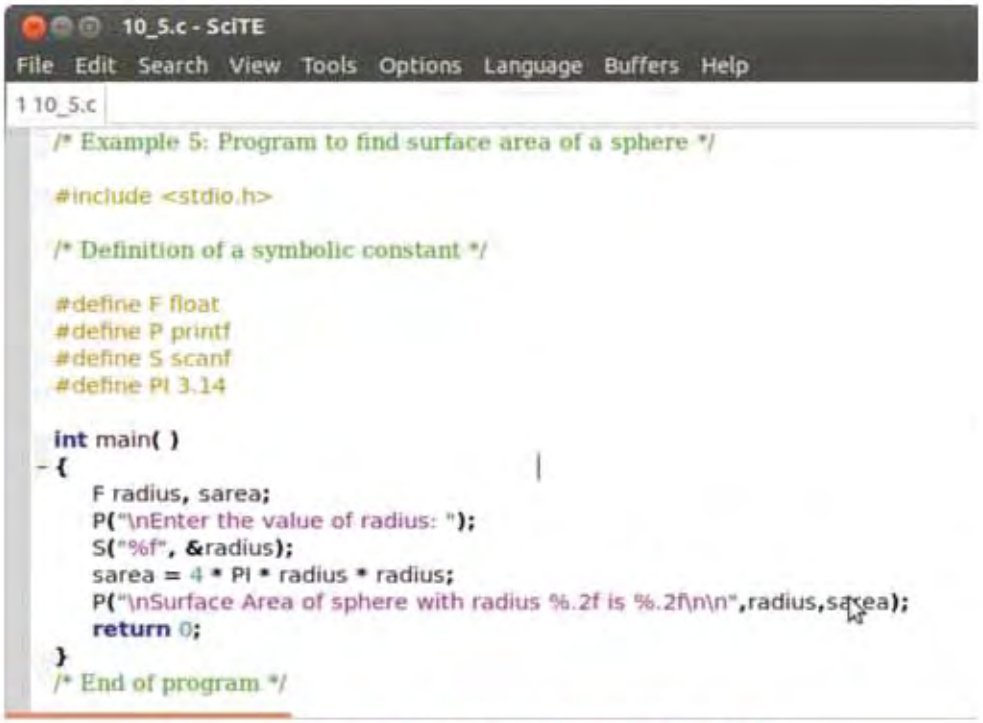


Figure 10.8 : Code listing of Example 10.5

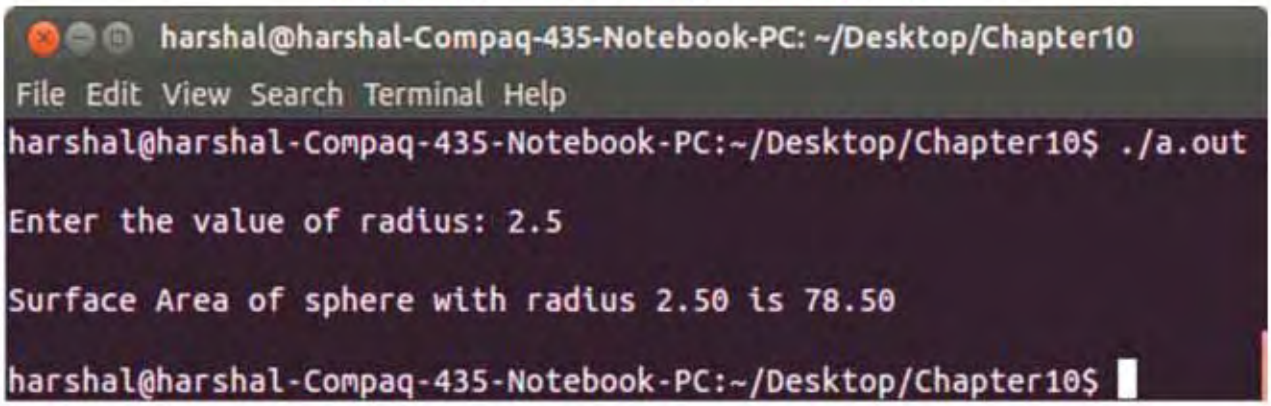


Figure 10.9 : Output of Example 10.5

Explanation

In this program we are defining four symbolic constants F that refers to a keyword float, P that refers to a function printf, S that refers to a keyword scanf and PI that refers to a float value 3.14. The first statement within function main defines two variables of type F. As F represents float this statement actually defines two float variables. The third statement uses P to print a message. Then we read the value of radius using S and calculate the surface area of sphere. Finally we again use P to print a message and value of surface area.

Points to Remember in C Program

C programming though easy needs a bit of a practice. The points given in this section needs to be remembered by all C programmers.

- Header file should be included before the main() function.
- An executable C program should always contain a main() function.

- Execution of C program starts from the first opening curly brackets after main().
- Two functions with same name are not allowed in C program.
- Normally C programs consist of lower letter alphabets.
  Identifiers are case sensitive.
- There should be atleast one blank space between two words of a C program.
- Usually C statements end with a semi colon.
- Comment can be inserted wherever a blank space can be inserted.
- Every opening curly bracket should have a corresponding closing curly bracket.

Execution of C Program

Till now we have seen so many C programs and their output. Let us now learn to execute the C program. Before taking hands on experience let us understand the actual steps that we need to perform. Figure 10.9 (a) shows the entire process.

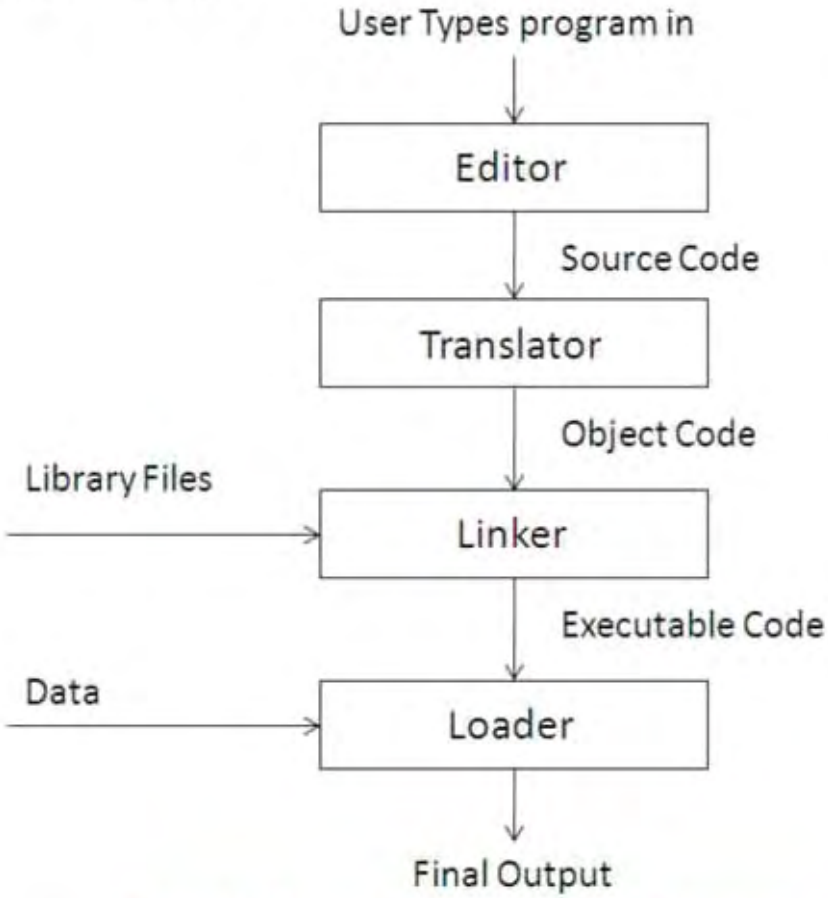


Figure 10.9 (a) : Steps of executing C program

As seen in figure 10.9 (a), the first step is to write a program using any text editor, the program that we write using an editor is known as source code or source program. The extension of the source file written in C language is "c". We then compile the source program using a compiler. A compiler is the translator that converts a source program into machine language program known as object code or object program. Different formats of object code are available. A program called linker is then used to link the object code with the library functions giving executable program or executable code. Finally executable code is loaded on to memory by a program called loader along with the data required to give us the output.

Throughout this book we have discussed a compiler known as gcc compiler. The gcc compiler is provided by Free Software Foundation, it is a Unix/Linux based ANSI C compiler. It is usually operated via the command line but can also be integrated with an text editor like SciTE or an Integrated Development Environment (IDE) like Eclipse. It often comes bundled with almost all Linux installations hence we can simply start using it without any problem.

Let us try to use the compiler, First we need to write a program using a text editor. We can use any good text editor to create a C program. In Linux environment, we have editors such as vi, gedit, emacs and many more. Choose the one you are comfortable with. The only thing that should be kept in mind is the fact that the file created should be saved with an extension c. Make sure that the character c is written in lower case. Let us try to use gedit to create a C program. Open terminal, at the prompt type gedit myprogram.c this will open a blank editor as shown in figure 10.10.

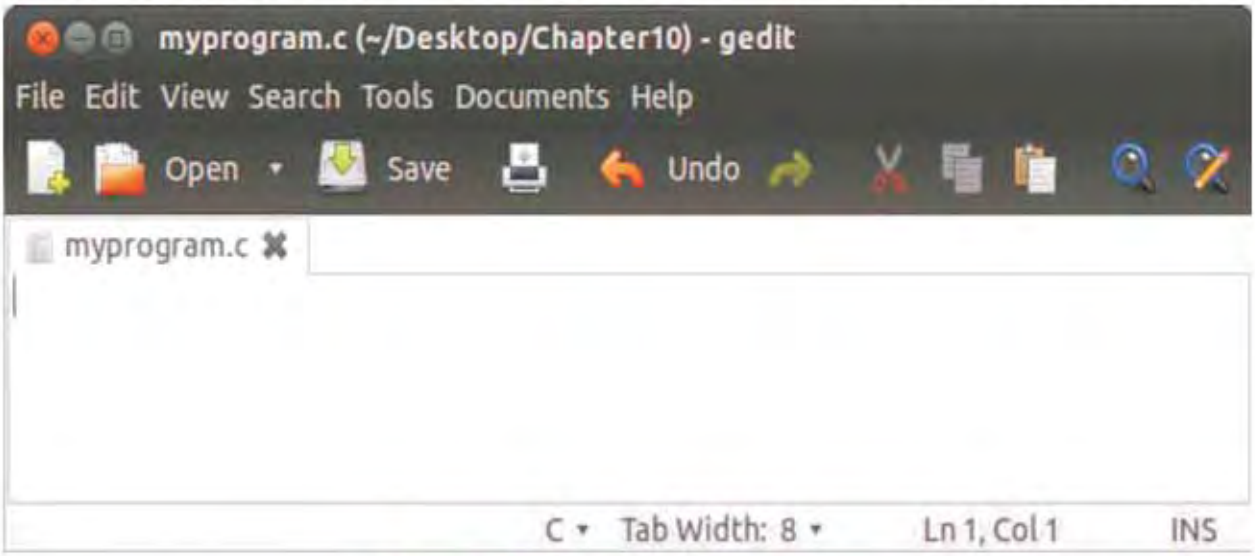


Figure 10.10 : Blank editor screen

Now type the contents given in code listing 10.2 in the blank editor screen.

```
/* Example 6: My C program */

#include <stdio.h>
int main()
{
    printf("\nWelcome to the world of C programming using gcc\n");
     return 0;
}
```

Code Listing 10.2 : Code of Example 10.6

Once the program is ready your editor will look similar to the figure 10.11, save it and close the editor.

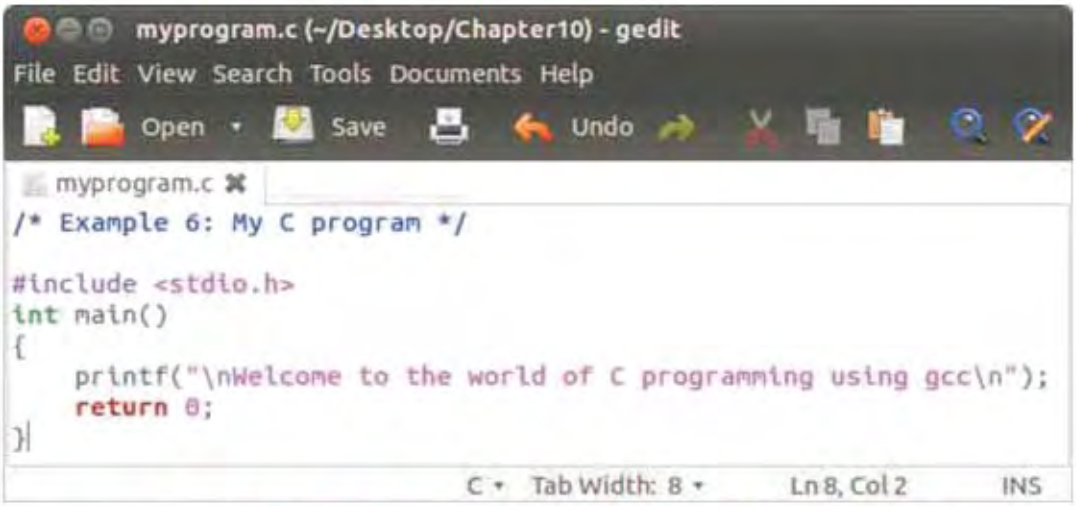


Figure 10.11 : Editor with code listing of Example 10.6

Now we are ready to compile it, go back to the terminal and type the command mentioned:

$ gcc myprogram.c

If there is no error in the program, a new prompt line will be visible. It means that the compilation is successful and an executable file with the default name a.out is created in the same directory as the source file. In case, we have errors, then appropriate error message would appear on the screen. If error occurs, we need to rectify the program and recompile. To see the output of the program, type the command as mentioned and press Enter key:

$ ./a.out

This command will display the output of myprogram.c on the screen as shown in figure 10.12.

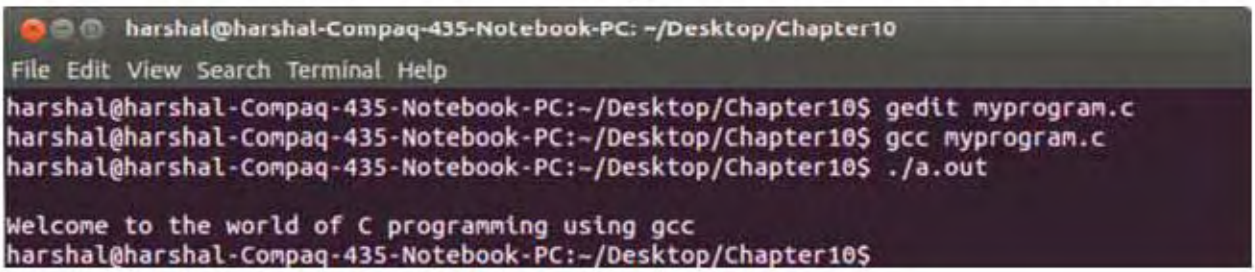


Figure 10.12 : Output of Example 10.6

As compilation of all C programs generates a default output file named a.out, it is a good practice to give a name to this output file. Thus, an output file with the given name once created can be reused many times until there is a change in the actual source code. The following command shows how to give a name to output file using gcc command :

$ gcc -o myprogram.o myprogram.c

In this command the first parameter myprogram.o represents the output file name, while the second parameter represents the source file name. Observe that we have written .o as an extension for the first parameter, it is just an indication that this file is an output file. The compiler though does not require this extension. We may simple write myprogram instead of myprogram.o. After a successful execution of the preceding command, the compiler will now create a file named myprogram.o instead of a.out. We can now see the output of the program by using the command mentioned herewith :

$ ./myprogram.o

Many other parameters can also be used along with gcc command. To get the detailed help of gcc command type the help command as mentioned :

$ man gcc

Let us try to write and compile one more program. Example 10.7 uses an inbuilt function to calculate the square root of a number entered by the user. Figure 10.13 gives the code listing of the example.

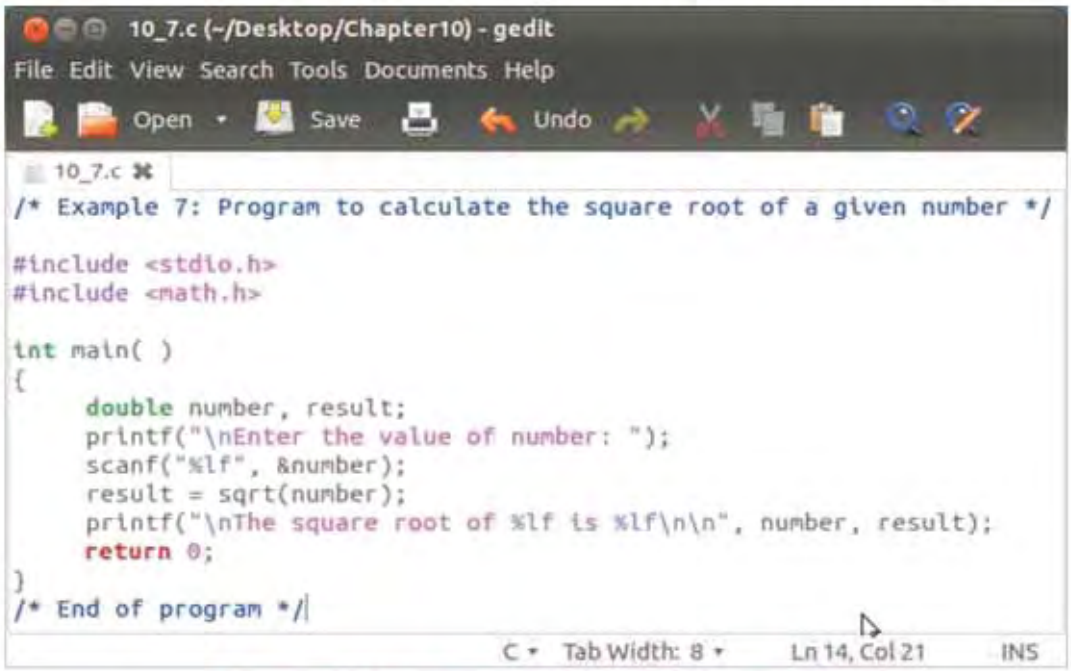


Figure 10.13 : Code listing of Example 10.7

Let us now try to compile this program using gcc. Open terminal and type the command mentioned below at the prompt.

$ gcc -o 10_7.o 10_7.c

Figure 10.14 gives the output of this gcc command.

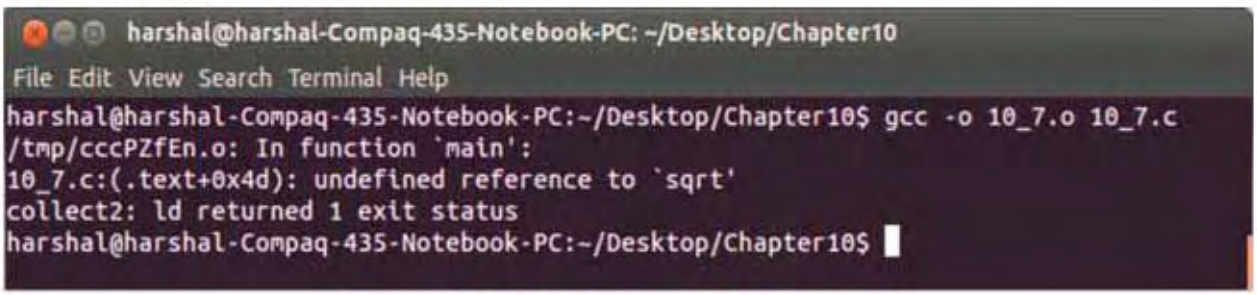


Figure 10.14 : Output of Example 10.7

Observe that we are getting a new prompt along with a message "undefined reference to 'sqrt'" on screen. As we have used an inbuilt function sqrt() in our program we need to link the math library at the time of compilation. The following command will let us link the math library.

$ gcc -o 10_7.o 10_7.c -lm

Once we issue this command we will get a new prompt as our program did not have errors. Now at the prompt type ./10_7.o and observe the output. Figure 10.15 gives us the correct output.

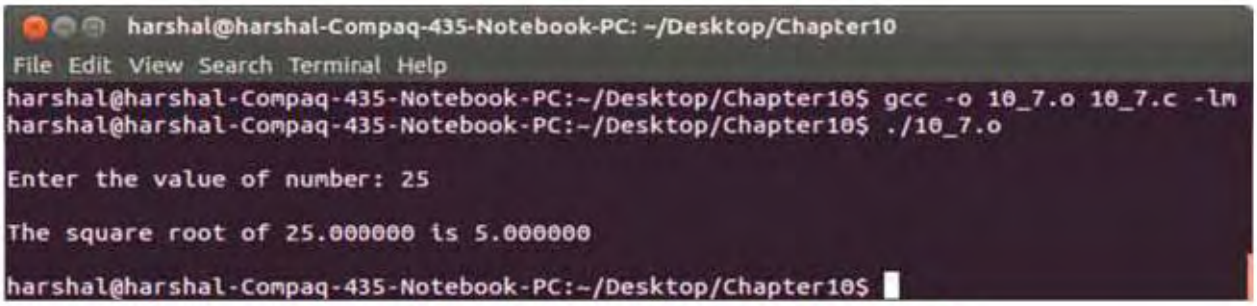


Figure 10.15 : Correct Output of Example 10.7

The method mentioned here is the most basic method for creating and executing a C program on Linux environment. Instead of using any normal text editor we will use an editor named SciTE to create and execute our program in this book. SciTE allows us to compile and run simple programs with one window itself. For programs with user inputs we may compile the program using SciTE, but will use terminal to execute the program. Let us try to see the usage of SciTE editor.

Open the SciTE Text Editor from appropriate location of your machine. We have already seen the look the SciTE editor window in previous examples. Type the contents of example 10.1 shown in figure 10.1 in the blank screen of the SciTE editor. Now save the file with the name 10_1.c. You can use CTRL + S or use **File → Save.** The window will now look similar to the one shown in figure 10.16.



Figure 10.16 : Example 10.1 as written in SciTE window

Once the program has been written and saved we need to find syntax errors if present. To find syntax errors we need to compile the program. Press CTRL key along with F7 key (CTRL + F7) or select Tools → Compile from the menu. If no errors are there in the program then we will see a screen as shown in figure 10.17.

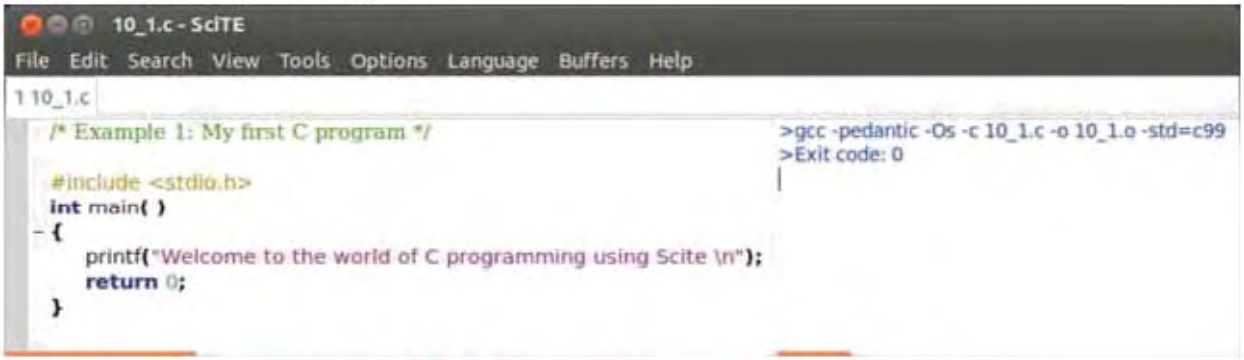


Figure 10.17 : Successful compilation message

Now we need to execute this program. For this press the F5 key or select Tools → Go. We will now be able to see the output window along with source code window as shown in figure 10.18.

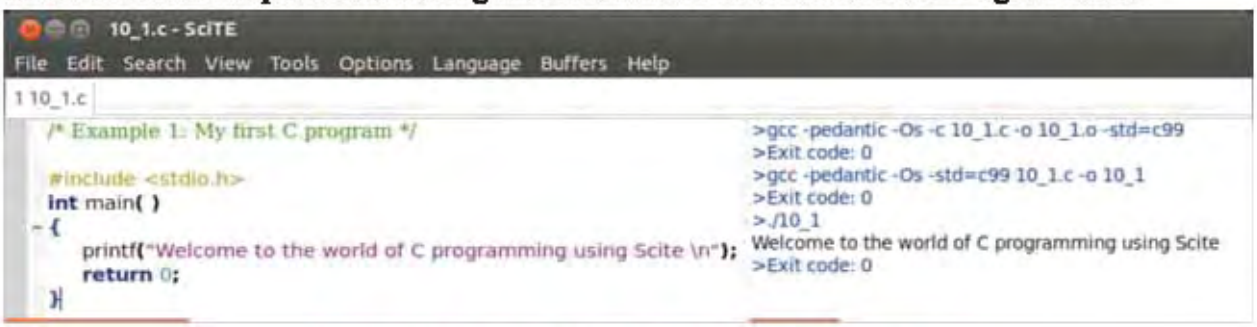


Figure 10.18 : SciTE editor with two windows

Observe that we are able to see output of both the activities that we have performed. Observe the first output shown below that we had got when we just compiled the program.

```
> gcc -pedantic -Os 10_1.c -o 10_1.o -std=c99
> Exit code: 0
```

Here the file 10_1.o although an output file is not executable. Now observe the output shown below that we got when we used Go.

```
>gcc -pedantic -Os -std=c99 10_1.c -o 10_1
>Exit code: 0
>./10_1
Welcome to the world of C programming using scite
>Exit code: 0
```

Here the file was processed in two phases. First the file was compiled and an executable file with the name 10_1 was created. This operation is represented by first two lines. In the second phase the output file was executed using command ./10_1 (as shown in the third line). The last two lines are outcome of this execution.

Note: the text succeeding > symbol refers to actions taken by compiler, while the text shown without a preceding > symbol is output that user wants to display on screen.

It is also possible to enable and disable the output window by pressing F8 key or by selecting **View → Output**. To clear all the previous outputs press Shift + F5 or select **Tools → Clear Output**.

In this example we have taken a program that worked properly. Hence the process of compilation gave us no error. Let us now see what would happen if we have entered a wrong program. Assume that we have entered example 10.2 in the editor with a small change. Instead of writing the statement "circumference = 2 * PI * radius;" we did a typing mistake and wrote "circumfernce = 2 * PI * radius;". Now if we press CTRL + F7 or F5 we will get the screen as shown in figure 10.19.

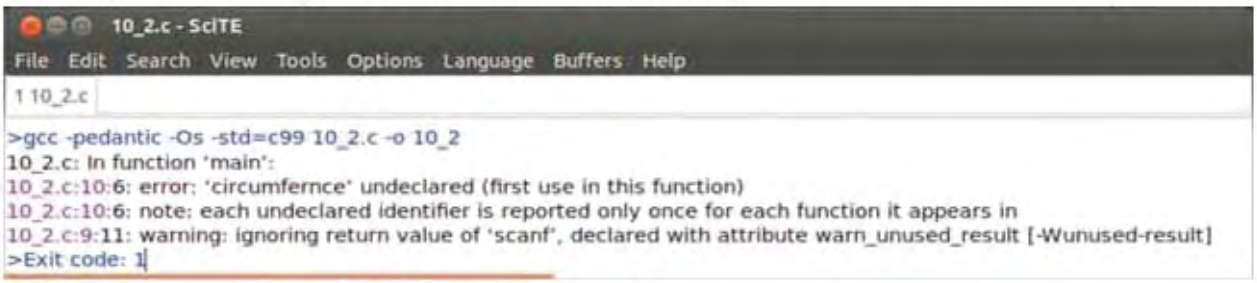


Figure 10.19 : Compilation of program containing error

As complete image could not be shown properly we have shown only the output window. To see only output select Options → Vertical Split. Observe that we are able to see errors in figure 10.19. Observe the line "10_2.c:10:6: error: 'circumfernce' undeclared (first use in this function)" in the figure. It informs us that the variable with a name circumfernce is not declared in the program. Rectify the error and repeat the process of compilation. If no errors are found by the compiler, it will try to execute the program if we have pressed F5. Although we will not get any output in our case as we have used scanf() function. The Ubuntu version of SciTE seems to have a bug as of now as it does not wait for user inputs. We can execute the program using the name of the file without any extension from the terminal.

History of C

Having learnt how to program in C, let us now have some historical overview of C. The origin of C has been dated back to 1972 in Bell laboratories. The man who owns the credit of creating C language is Dennis M. Ritchie. It was derived from *Basic Combined Programming Language* commonly known as BCPL. The aim of developing C was to build robust system software. But it became a pet of programmers in coming years and has been used for developing all kind of software. Hence it has come to be known as General Purpose Programming Language.

Although born in 1972, it was standardized in 1989 by American National Standards Institute (ANSI). From there on it came to be known as ANSI C. Different operating systems and compilers support this standard.

C is a structured language. It allows our program to be broken into small pieces known as functions. These functions once generated are reusable. Set of such functions then becomes a C program. Taking such an approach helps us to solve problem arising in the program easily, as we have to concentrate on only one function rather than an entire program. Usually programs written in C can be ported to different machines having different operating systems and compilers with almost negligible modification. Thus it is considered to be portable language. C is also considered to be a middle level language by some and higher level language by others. In any case it has all the features that a programmer would like to have.

Summary

In this chapter we learnt about the structure of C program. We looked at different components that can be part of C program. We then learnt about C character as mentioned can be divided into four categories namely letter, digits, white spaces and special characters. We saw how to use these characters and form valid words in C language. Then we looked at the steps of creating and executing a C program. We then learnt how to use gcc compiler to compile and execute our programs. Finally we learnt how to use the SciTE Text Editor to create and execute the C program.

Instruction for Teachers

The discussion of SciTE Text Editor has been made based on the assumptions that the editor is installed and its short cut would be available on the computer. Before starting to write C programs in your SciTE editor make sure you select Language -> C/C++ at least once. This will ensure the highlighting of C keywords when programming.

Also you may need to add the line mentioned below in properties file of SciTE to allow direct execution of C programs by using F5 or Tools → Go.

command.go.needs.*.c=gcc $(ccopts) -std=c99 $(FileNameExt) -o $(FileName)

The steps mentioned here will allow you to change the properties file.

1. Find location of cpp.properties file in your machine. (In our computer it is /usr/share/scite/cpp.properties
2. Open terminal and type sudo gedit your_file_path/cpp.properties and press Enter key.
3. You will be prompted for administrator's password. Key in the correct password and press Enter key.
4. Now cpp.properties file will be opened in gedit window. Locate the code given in table 10.7 in the file.

```
ccopts=-pedantic -Os
cc=g++ $(ccopts) -c $(FileNameExt) -o $(FileName).o
ccc=gcc $(ccopts) -c $(FileNameExt) -o $(FileName).o

make.command=make
command.compile.*.c=$(ccc) -std=c99
command.build.*.c=$(make.command)
command.build.*.h=$(make.command)
#command.go.*.c=./a.out
command.go.*.c=./$(FileName)
```

Table 10.7 : Code to be searched in cpp.properties

The make sure that the last two lines look similar on your computers. If they are different, then change it as can be seen in table 10.9. This change makes sure that the executable (output) file will always be saved as the filename that you specify for your source code when using SciTE.

5. Add the following line to this file and save it.

   # To make the Go command both compile (if needed) and execute, use this setting:

   command.go.needs.*.c=gcc $(ccopts) -std=c99 $(FileNameExt) -o $(FileName)

6. Close the gedit and terminal windows. You are now ready to use the SciTE editor.

The screens given in this chapter are sample screens. These may differ based on the version of Ubuntu available in the school. The functionality of the screens though would remain same.

EXERCISE

1. List and explain the characteristics of a program.
2. Explain the significance of main( ) function.
3. What is the purpose of file include section in a C program ?
4. What is an identifier ? How is it useful in C program ?
5. What is variable ? State the rules for defining a variable.
6. Differentiate between single character constant and string constant.

7. State whether true or false :

(a) The extension of C program is ".h".

(b) Usually C statements end with a comma.

(c) Amount is a valid variable name.

(d) #define PI 3.24 includes a file named PI in C program.

(e) "X" is a valid single character constant.

8. Choose the correct option from the following :

(1) Which of the following is an extension of C program file ?

(a) c (b) h (c) s (d) t

(2) Which of the following number refers to number of C character categories?

(a) 0 (b) 2 (c) 4 (d) 8

(3) Which of the following C character categories does the symbol = belong ?

(a) Letter (b) Blank Space

(c) Special Character (d) Digit

(4) Which of the following is a valid keyword of C ?

(a) ofsize (b) sizeof (c) forsize (d) sizefor

(5) Which of the following is an invalid variable name in C ?

(a) Register (b) RegIster (c) registre (d) register

(6) Which of the following is an invalid integer constant in C ?

(a) 0xG (b) 0xA (c) 0xB (d) 0xD

(7) Which of the following is valid real constant in C ?

(a) -2.0.5e5 (b) -20.5e5.5 (c) -20.5e5 (d) -20.5e.5

(8) Which of the following is valid single character constant in C ?

(a) 'a' (b) '\a' (c) "a" (d) Both a and b

(9) The preprocessor directive *#define* is used to define which of the following in C ?

(a) String constant (b) Symbolic constants

(c) Integer constant (d) Single character constant

(10) Which of the following function key is used to directly execute a program ?

(a) F7 (b) F9 (c) F5 (d) F8

LABORATORY EXERCISE

1. Write a C program to display your name, school name, the standard you study in and school address in the center of the screen.

```
**************************
* Name :                 *
* School Name :          *
* Standard :             *
* Address :              *
*                        *
**************************
```

2. Write a C program to print first letter of your surname on the screen. For example if the first letter of your surname is P then the output should be.

```
****
*   *
****
*
*
```

3. Write a C program to print a greeting of your choice on the screen. For example if you want to wish someone on Diwali, print "Wishing you a happy and prosperous Diwali".



Data Types, Operators and Expression in C Language

In the previous chapter we saw some simple C programs. We also learnt about the character set and tokens of C language. One of the tokens discussed was identifier. We make use of identifiers in all C programs. Suppose in our program we have defined an identifier named "date", that is capable of storing value of date. What do you think is the value that can be stored in it? Can we assign it a value 12.50? The answer would be no. The correct value could be 12 or 13 or any valid positive two digit whole number between 1 and 31. This example suggests that, not only the identifier but the values stored in it are also important. C Language uses certain keywords to identify the type of value that can be stored in an identifier. These keywords associate a data with its types and hence are known as data types. In this chapter we will discuss in detail different data types available in C.

What is Data Type ?

The type of value that can be assigned to an identifier is known as its data type. If an identifier "date" has been assigned a value 12, then we would say that data type of "date" is integer. Similarly if an identifier "amount" has value 99.50, then we would say that data type of "amount" is real.

C language uses set of keywords to relate a data with its value. These keywords identify two things, the type of value that can be stored in an identifier and the memory space required by an identifier. Each data type is allocated a fixed memory space in C. It is denoted by bytes. 1 byte is combination of 8 bits.

Basic Data Types of C

C language supports basic data types called integer, decimal and character. These data types are represented using keywords int, float and char respectively. We also have one more primitive data type known as void. An identifier is associated with its data type by using the syntax:

Data type identifier 1 [, identifier 2, identifier 3, ..., identifier n];

The contents written in square brackets for the given syntax are optional. Let us now have a look at each of these basic data types one by one.

Integer

In the previous topic we mentioned that an identifier "date" can be assigned a value 12. Here 12 is an integer value. An integer is a positive or negative whole number with no fractional part in it. Some example of integer numbers are -99, 12, -10, 900, 30000 etc. To declare a variable of integer type we use the syntax:

int identifier 1 [, identifier 2, identifier 3, ..., identifier n];

Such a statement in C is called a declaration statement. Some examples of declaration statement are :

int roll_number;

int date, month, year;

The first statement declares an identifier with a name "roll_number" capable of storing an integer value. The second statement declares three identifiers "date", "month" and "year" each capable of storing integer values.

In ANSI C the data type int uses 4 bytes of memory space. It has a range of -2147483648 to +21474836411. All the variables defined till now are signed, i.e. they are capable of storing both positive as well as negative values. In many situations we may not require a negative value at all. None of the variables declared by us can have negative value. It is possible to allocate only positive values in C language. For this purpose we use a keyword unsigned. The examples now can be modified to

```
unsigned int roll_number;

unsigned int date, month, year;
```

These statements indicate that roll_number, date, month and year are identifiers of type integer and are capable of storing only positive value. Table 11.1 gives a brief description of integer data type.

Data Type	Range	Bytes required	Example
int	-2147483648 to +2147483647	4	int balance_amount;
unsigned int	0 to 4294967295	4	unsigned int counter;

Table 11.1 : Integer data type

In table 11.1 the value of range depends on the number of bytes used. For signed integers the value of range is calculated as $-2^{(n-1)}$ to $+2^{(n-1)} - 1$. Here "n" refers to number of bits required. Similarly for unsigned integers the range is calculated as 0 to 2^{n-1}. It is possible to increase the range of integer numbers by using a prefix long with the integer data type. Adding long as a prefix to int allocates 8 bytes of memory to the variable. We can define such a variable using the syntax shown below :

```
long int population;
```

Having seen different forms of integer data type available in C let us now see a sample program. Figure 11.1 shows the code listing of a program that illustrates the use of integer data type.

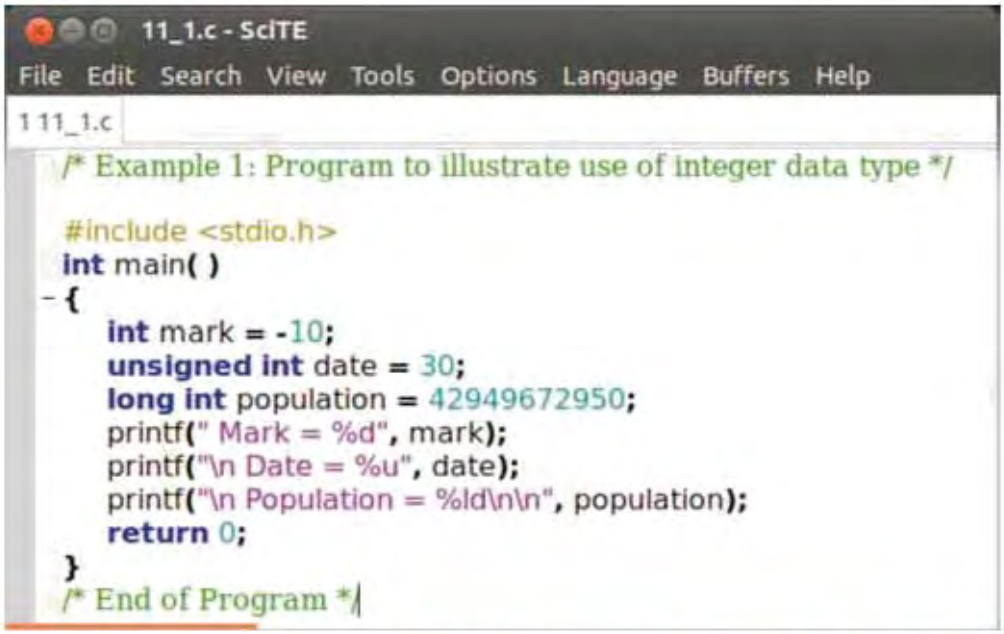


Figure 11.1 : Program illustrating use of integer data type

Explanation

Let us now see what each of the line in figure 11.1 mean. The first statement after the opening curly brackets declares an integer variable called "mark". In the same statement we assign it value -10. The second statement declares positive integer variable called "date" and assigns it a value 30. The third statement declares an integer variable with higher range called "population" and assigns it value 42949672950. The next three statements prints the values assigned to these variables on the screen. Figure 11.2 shows the output of example 11.1.



Figure 11.2 : Output of Example 11.1

Real

Many a times we need real numbers instead of integer numbers. For example amount to be paid by a person can have value 95 or 95.50. In such situations the data type *int* does not work. To use real values C language supports a data type identified by keyword float. It uses 4 bytes of storage space. Float values are precise to 6 and at times 7 decimal digits. If we need higher precision then we have to use keyword double rather than float. It is an extension of float data type. Double values use 8 bytes and are precise to 16 and at times 17 decimal digits. Examples of real variables are :

float amount_to_pay;

double balance_amount;

The first statement declares an identifier with a name "amount_to_pay" capable of storing float value. The second statement declares an identifier "balance_amount" capable of storing float values but having higher range and precision.

We can also use long as a prefix to double. This increases the range of double. Table 11.2 gives a brief description of float data type.

Data Type	Range	Significant Digits	Bytes required	Example
float	+/-3.4 e -38 to +/-3.4 e +38	6	4	float price;
double	+/-1.7 e -308 to +/-1.7 e +308	16	8	double tot_price;
long double	+/-3.4 e - 4932 to +/-1.1 e + 4932	16	16	long double credit;

Table 11.2 : Float data type

As can be seen in table 11.2 we have specified range in scientific form. This is so because C also supports scientific format. For example 95.50 could be represented as 0.9550 e 2. Here 0.9550 is known as mantissa while 2 is known as exponent. The float values can be negative or positive. The memory representation of float is given in figure 11.3.

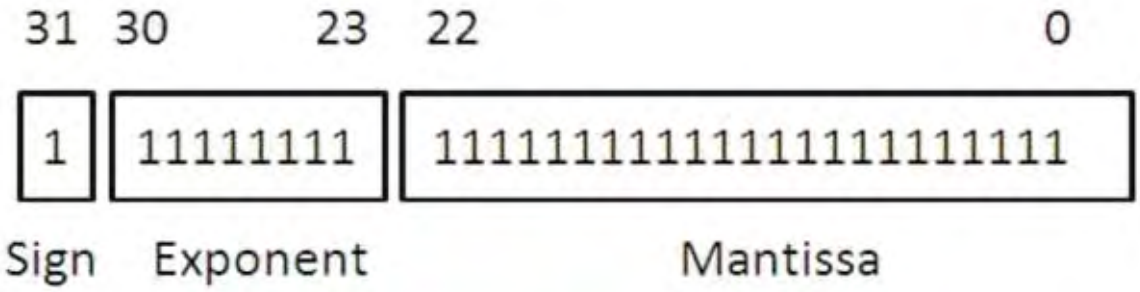


Figure 11.3 : Memory representation of float

Let us now see an example that shows the use of float, double and long double data type. Figure 11.4 shows the code listing of a program that illustrates the use of float data type.

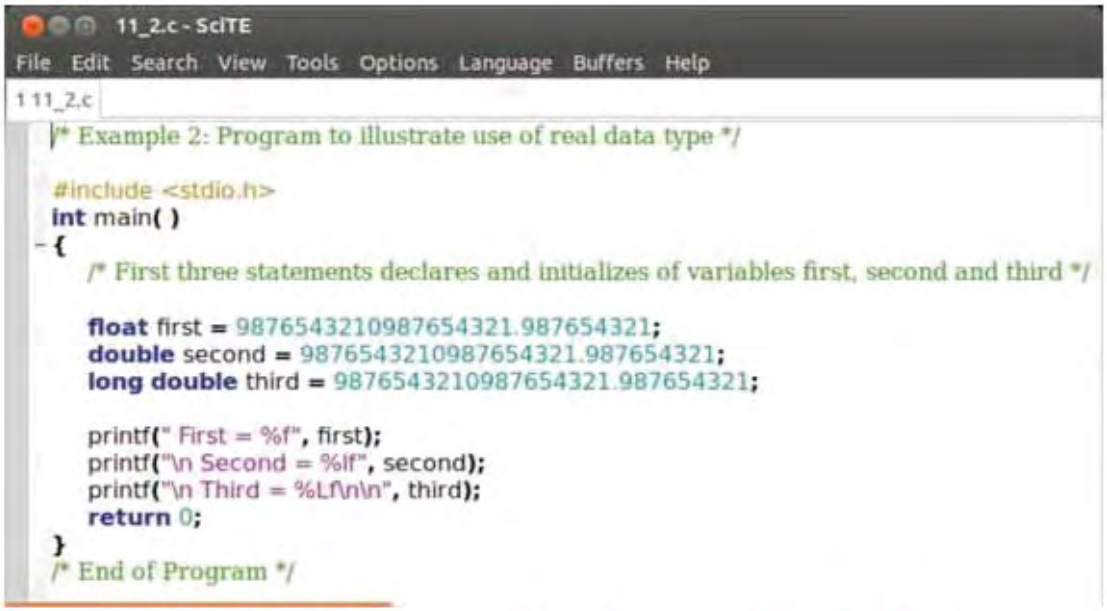


Figure 11.4 : Program illustrating use of float data type

Explanation

The first three statements after the opening curly brackets declare three variables "first", "second" and "third". Each of these variables is assigned a float value of 9876543210987654321.987654321. The next three statements prints the values assigned to these variables on the screen. Figure 11.5 shows the output of example 11.2.

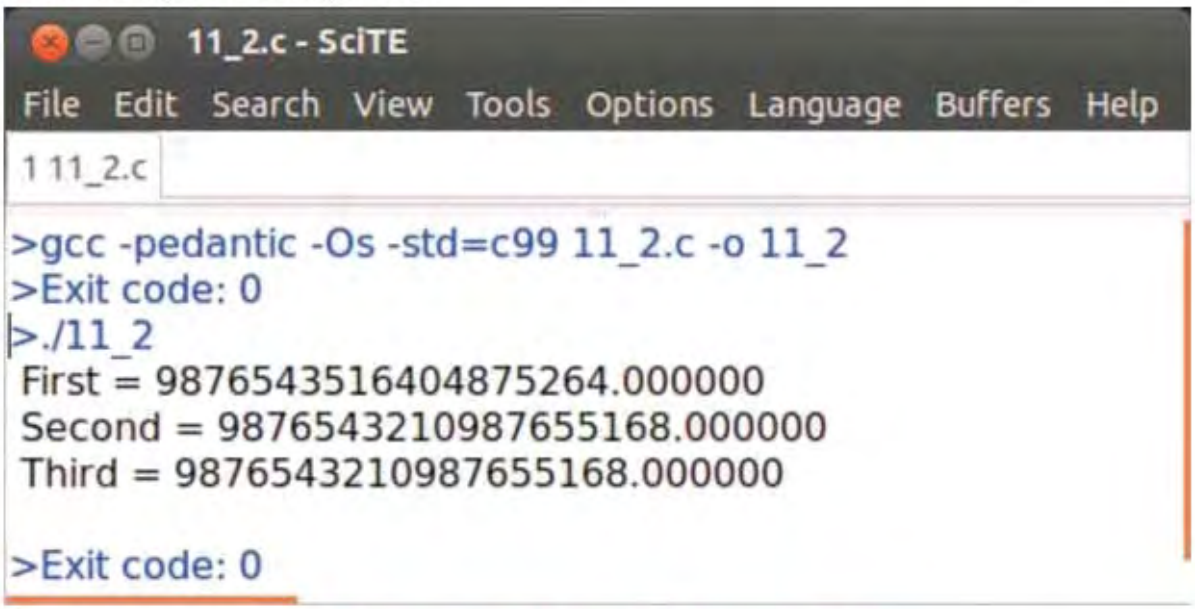


Figure 11.5 : Output of Example 11.2

Observe that the values of variable first, second and third are 9876543516404875264.000000, 9876543210987655168.000000 and 9876543210987655168.000000 respectively. Looking at the output properly we can see only first 7 digits of variable "first" are matching the actual entry. Similarly in variables "second" and "third" only first 15 digits are matching. This is the reason we have mentioned that the float is precise to 6 and at times 7 decimal digits. Similarly double and long float are precise to 15 and at times 16 decimal digits.

Floating point numbers when used at times lead to loss of precision. We may not get an exact result as expected. Hence to get better accuracy of results it is advisable to use higher precision. The benefit of using float values is that it can represent much broader range on values compared to integer.

Character

In both the above cases we have stored numbers. What happens if we need to store a name of city or gender of a person say 'M' for male and 'F' for female. Here we only have alphabets. Such values cannot be stored by using int or float. To store such values we have to use a keyword char. It requires 1 byte of memory space. Each character is associated with an integer value called ASCII (American Standard Code for Information Interchange) value. Appendix II gives details of all ASCII values. By default char is unsigned. It is also possible to have signed char. Table 11.3 gives a brief description of character data type.

Data Type	Range	Bytes required	Example
char	- 128 to + 127	1	char gender;
unsigned char	0 to 255	1	unsigned char choice;

Table 11.3 : Character Data type

Figure 11.6 shows the code listing of a program that illustrates the use of character data type.

```
11_3.c - SciTE
File  Edit  Search  View  Tools  Options  Language  Buffers  Help
1 11_3.c
/*Example 3: Program to illustrate use of character data type */

#include <stdio.h>
int main( )
{
    char gender = 'm';
    char answer = 'M';
    char number = '2';
    char sp_char = ' ';
    printf("\nThe character stored in gender is %c and its ASCII value is %d", gender, gender);
    printf("\nThe character stored in answer is %c and its ASCII value is %d", answer, answer);
    printf("\nThe character stored in number is %c and its ASCII value is %d", number, number);
    printf("\nThe character stored in sp_char is %c and its ASCII value is %d\n\n", sp_char, sp_char);
    return 0;
}
/* End of Program */
```

Figure 11.6 : Program illustrating use of char data type

Explanation

The first four statements after the opening curly brackets declare four variables "gender", "answer", "number" and "sp_char" and are assigned values 'm', 'M', '2' and ' ' (blank space) respectively. The next four statements prints the values assigned to these variables along with their ASCII values on the screen. Figure 11.7 shows the output of example 11.3.

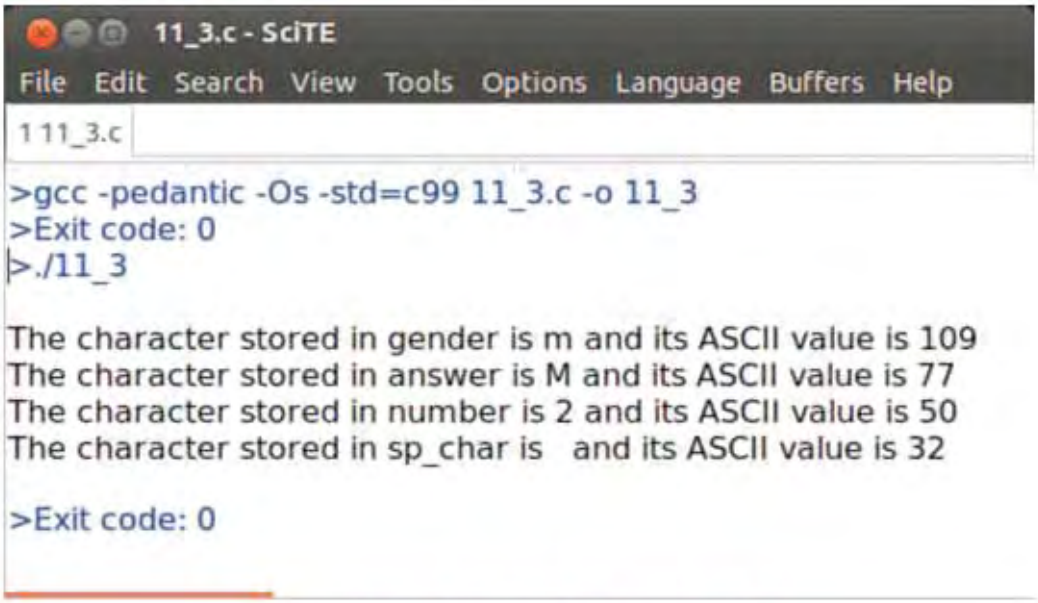


Figure 11.7 : Output of Example 11.3

Note the difference in the ASCII values of small m and capital M. We can see that it is also possible to use a number in character form. To use numeric value 2 as character we have to write it as '2'. Also observe that though the character assigned to sp_char is not visible on the screen we still do see its ASCII value.

Empty Data Set

C provides a special data type identified by keyword void. This data type has no value; hence we denote it to be empty. It is used mainly as a return type of functions. As mentioned in previous chapter C program consist of function. Functions in C may return a value. If we don't want the function to return a value then we can prefix it with void.

As seen in previous examples the last statement of all programs is return 0;. This statement tells the compiler to exit from the function body. If we don't write this statement then we will get a warning message "Function should return a value". To avoid getting such a message we can prefix main( ) with void.

Assigning Values to Variables

Variables once declared can be assigned values during the execution of the program. The value assigned has to be in lines of the specification of data type. To assign a value we use the given syntax:

Variable = value;

It is also possible to initialize and declare a variable in same statement. To do so we use syntax:

Data type Variable = value;

In all the examples discussed so far in this chapter we have used this technique of initialization.

User Defined Data Type

C is a flexible language. It allows user to create new entities from existing one. We can create a new data type by making use of existing data type. Two keywords typedef and enum are used in C for generating user defined data type. Both these keywords define a variable that can be used as data type in our program.

Type Definition

The syntax for using type definition is:

typedef datatype variable;

Here data type refers to an existing data type for example int or float. Variable refers to new name that we want to assign to the available data type. In real sense C does not allow us to create a new data type. Instead it allows us to give an alias to an existing data type. In normal life also we use aliases. One example would be pet name of a person. Some examples of typedef are:

typedef char alpha;

typedef double twice;

The first statement gives an alias "alpha" to character data type, while second statement gives an alias "twice" to double data type. After providing an alias, to declare a character variable called "choice" and double variable called "amount" we can use the given statements:

alpha choice;

twice amount;

Enumerated data type

Enumerated data types can be created in C by using keyword enum. The syntax for using enum is :

enum identifier {value 1, value 2, value 3, ..., value n};

Here enum is the keyword. Identifier is the name that can be used to define a new variable. Value 1 to Value n refers to names of numeric constants 0, 1, 2, ...etc. The variable defined using enum can be assigned any numeric value or values given in curly brackets. An example of enum would be

enum money {rupee, dollar, pound, yen};

enum money currency;

currency = dollar;

The first statement defines enumerated data type "money". The next statement declares a variable "currency" of type money. The last statement assigns value dollar to currency and is equivalent to currency = 1. Here rupee, dollar, pound and yen refer to numeric constants 0, 1, 2 and 3 respectively. These numeric constants are assigned by the compiler automatically. It is possible to modify these values by assigning new values. For example if we want rupee to be equivalent to 10, dollar to be equivalent to 50, pound to be equivalent to 75 and yen to be equivalent to 100 then we may rewrite the first statement as: enum money {rupee = 10, dollar = 50, pound = 75, yen = 100}; Now the statement currency = dollar would be equivalent to currency = 50.

User defined data types improve the readability of C programs to some extent. It also allows user to define data types with meaningful names.

Derived Data Type

As seen in the previous topics, C is an extensible language. The basic data types may suffice our needs in many cases. But it may not be appropriate to use them due to complexity involved. Say for example we need to store grades of 15 students. In such cases we need to define 15 variables.

Handling 15 variables may increase the complexity of the program. Similarly if we need to store complete information of 15 students like their roll number, gender, name and address then we may need additional variables. Grouping of the variables may become a problem here. To overcome these problems C provides us derived data types. Array, structure, union and pointers are examples of derived data types.

Array

It is possible to create a data structure called "array" in C. This data structure consists of group of variables having same property. The syntax for defining an array is:

data type variable[size];

Here data type can be any basic data type, user defined data type or derived data type. Variable is name of the array and size refers to number of elements in the array. For example C statement **char name_of_subject[10];** would define an array called name_of_subject having size 10. This means that it is possible to store group of 10 characters and identify it by using variable "name_of_subject". The data type of each element here would be character.

It is also possible to have a statement like **char name_of_subject[ ] = "C Language";** in C. The compiler will automatically decide length of the array. In the given case the length would be 11. Arrays are discussed in detail in chapter 15.

We also have some more derived data types like structure, union and pointers. The discussion of all these data types is at present out of scope of this book.

Operators and Expression

Till now we learnt about the data types available in C. Let us now learn about different operators available in C. We will also learn how to make use of the operators and operands to construct an expression. Finally we will see how expressions are evaluated.

Operators

When learning basic mathematics in school we have performed operations like addition of two numbers, subtraction of two numbers. For example 5 + 3 and 9 - 7, here 5, 3, 9 and 7 are constants while '+' and '-' are symbols that identify the operations to be performed on the constants.

An operator is a symbol that identifies the operation that can be performed on operands. The operators in C can be categorized into eight types as mentioned :

1. Arithmetic Operators.
2. Assignment Operators.
3. Relational Operators.
4. Increment and Decrement Operators.
5. Conditional Operators.
6. Logical Operators.
7. Bitwise Operators.
8. Special Operators.

Let us now discuss these operators in detail.

Arithmetic Operators

To perform an arithmetic operation C provides various basic operators like '+', '-', '*', '/', '%'. An operator for exponent is not available in C. Table 11.4 lists the operators and its use.

Operator	Use
+	Addition of two operands or Unary Plus.
-	Subtraction of two operands or Unary Minus.
*	Multiplication of two operands.
/	Division of two operands resulting in Quotient.
%	Division of two operands resulting in Remainder.

Table 11.4 : Arithmetic Operators

We are familiar with the first four operators given in table 11.4. C provides us an additional operator '%' known as modulo operator. It is used to get a remainder when performing division of two integer operands. Note that it cannot be used with float operands.

Let us now see how we can use the operators. Consider an example of C statement

total_cost = quantity * cost_item;

This C statement forms a C expression, here total_cost, quantity and cost_item are known as C variables. As can be seen the expression consists of two operators and three operands. This statement when executed performs two operations. First, it multiplies quantity and cost_item and second, it stores the result of multiplication in total_cost.

Based on the operand used in the expression, the arithmetic can be categorized as integer arithmetic, real arithmetic or mixed mode arithmetic.

Integer Arithmetic

Arithmetic is considered to be integer arithmetic when the operands used in the expression are positive or negative whole numbers. The expression here is called integer expression. The result of integer arithmetic is always integer.

Let us write a program to find cost per item if we are given the total cost and number of items. Figure 11.8 shows the program illustrating use of integer arithmetic.

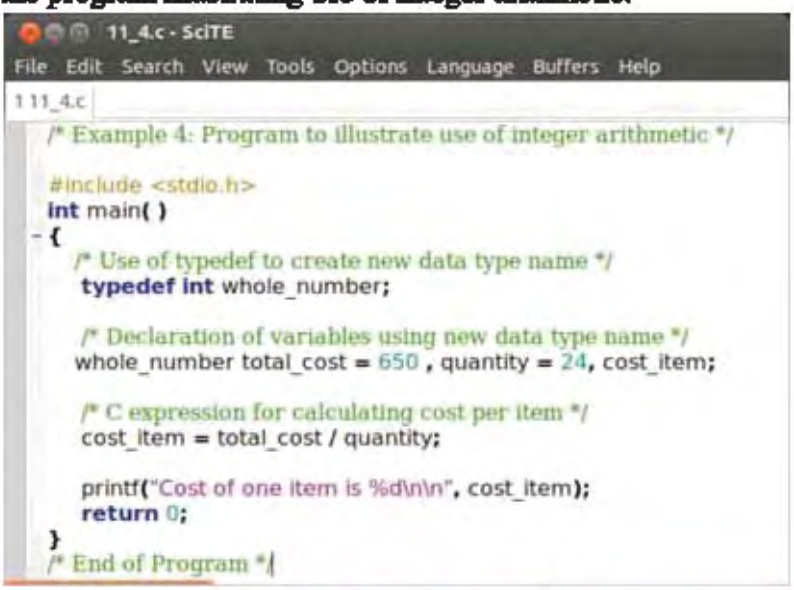


Figure 11.8 : Program illustrating use of integer arithmetic

Explanation

Here the first statement uses typedef keyword to create an alias for int. Then three integer variables "total_cost", "quantity" and "cost_item" are defined using the new alias. We also assign value 650 to "total_cost" and 24 to "quantity". Next statement divides "total_cost" by "quantity" and assigns the result to "cost_item". Then we print a message and value of "cost_item". Figure 11.9 shows the output of example 11.4.

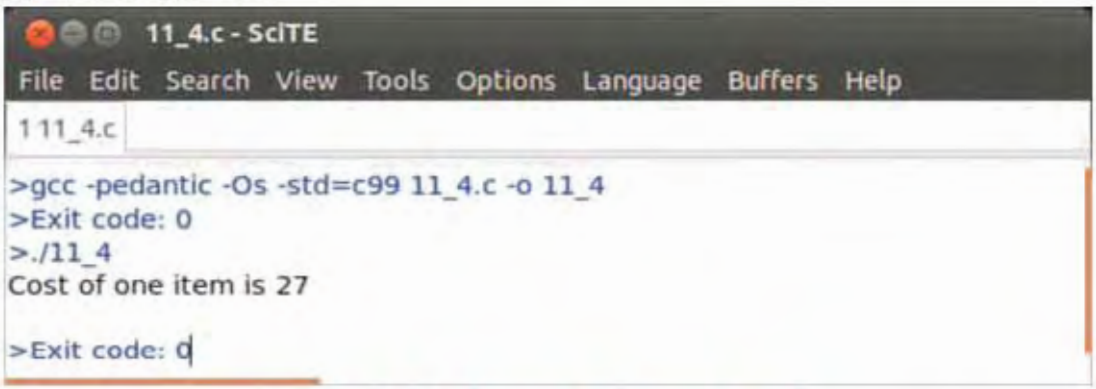


Figure 11.9 : Output of Example 11.4

As per the conventional mathematics 650/24 should give us 27.083333333. But we can see in the output that the value of "cost_item" is 27. From this we can say that the output of integer arithmetic will always be integer. In C the division operator works differently from conventional mathematics. In conventional mathematics 5/10 would give us 0.5, here it would give 0.

Real Arithmetic

If all the operands used in an expression are float the expression is called real expression. The result of real arithmetic is always represented in decimal values. Let us modify the program given in figure 11.8 such that we have real arithmetic instead of integer arithmetic. Figure 11.10 shows the modified program while its output is shown in figure 11.11.

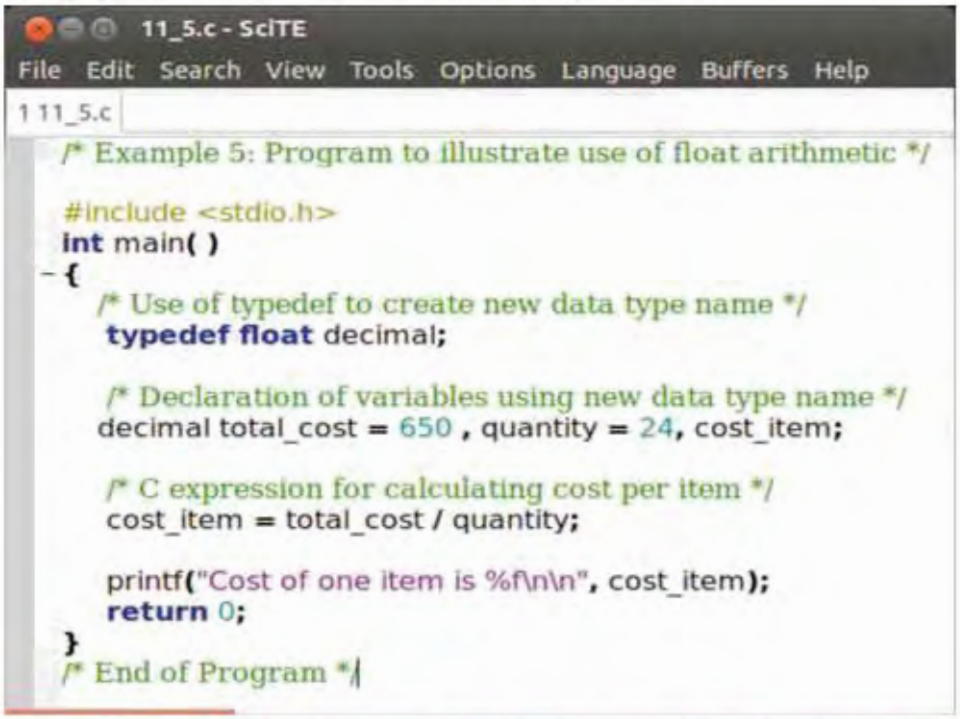


Figure 11.10 : Program illustrating use of real arithmetic

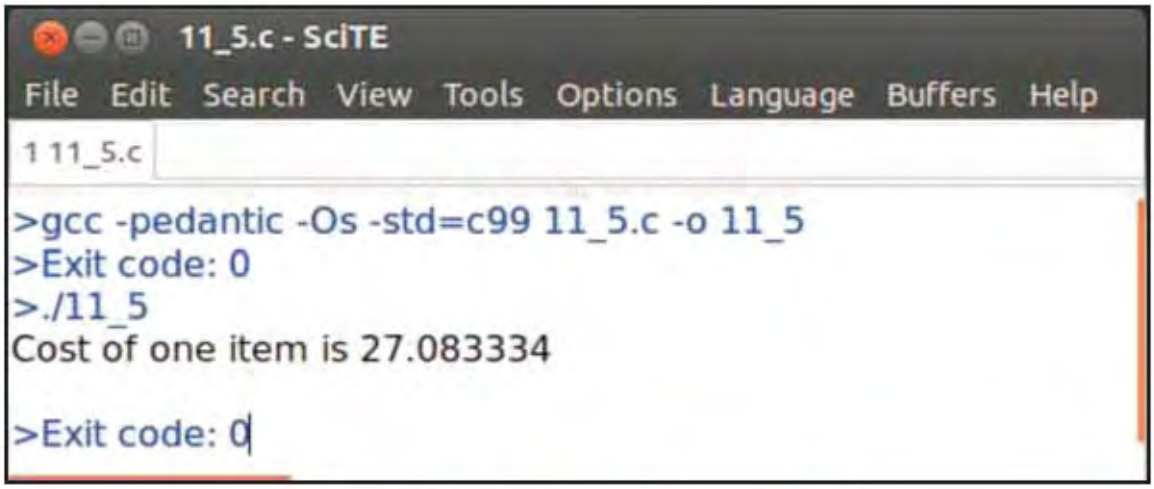


Figure 11.11 : Output of Example 11.5

Explanation

This example is similar to example 11.4 except for one thing. Here the variables are declared as float. Although the values given to "total_cost" and "quantity" are integers but internally they are stored as float. The output is also a float value.

Mixed Mode Arithmetic

In this mode of arithmetic we are able to use integer as well as float operands within an expression. The output will depend upon the data type of the variable to which value is assigned. Although an expression like result = 25.75 * 5; is allowed in C. It is not directly possible to evaluate it. An expression can only be evaluated if all the operands used in it are of same data type. C as it allows mixed mode arithmetic takes care of this problem. C allows automatic conversion of lower range data type to higher range data type.

Assignment Operators

In examples discussed till now we have used statements like total_cost = quantity * cost_item; or date = 30; here the symbol "=" is known as assignment operator. As can be seen it is used to assign a constant value or result of an expression to a variable.

C also provides an operator known as short hand operator. It is obtained by prefixing arithmetic operators to "=". The syntax for its use is:

variable op= constant value; or variable op= expression;

Here "op" refers to an arithmetic operator. While "op=" is known as short hand operator. Some examples of use of short hand operators are as given:

first -= 1; This statement is equivalent to first = first - 1;

first += 3; This statement is equivalent to first = first + 3;

first *= (second + third); This statement is equivalent to first = first * (second + third);

As can be seen the short hand operator provides us with ease of use. Though easy it requires certain amount of practice to use it efficiently.

Relational Operators

The relational operators allow us to compare two similar types of operands and generally are used to change the flow of execution of program. Say for example to give some discount to a customer we may check for the amount of purchase he/she has made. If it exceeds certain predefined amount then we may give discount to the customer. This example may be represented in C as *if (total_purchase > 10000) discount = 500;*

C provides us with six relational operators for the purpose of comparison. Table 11.5 lists relational operators and their meanings.

Operator	Use
= =	Check equality between two operands.
!=	Check non equality between two operands.
>	Check for greater value between two operands.
<	Check for smaller value between two operands.
>=	Check for greater value or equality between two operands.
<=	Check for smaller value or equality between two operands.

Table 11.5 : Relational Operators

Observe that to check equality of two operands we have use "= =" instead of "=", as "=" is used as an assignment operator in C. The syntax for using relational operators is:

expression-1 Rop expression-2

Here "Rop" refers to a relational operator; expression-1 and expression-2 can be an arithmetic expression, variable or constant. Relational operators are used along with decision structures like "if" and control structures like "for", "while" and "do..while". These structures have been discussed in detail in chapter 13 and 14. you will find many programs in this book that make use of relational operators.

Increment and Decrement Operators

At times we need to increase or decrease the value of a variable by 1. Such situations arise when set of statements are to be repeated. Here we need to use a variable that keeps track of the number of times a loop has been executed. We can achieve this objective by using short hand operator. For example we can use the syntax variable += 1 or variable -= 1 for incrementing or decrementing.

C provides two special unary operators "++" and "- -" for this purpose. These operators require only one operand. "++" is known as increment operator while "- -" is known as decrement operator. The syntax for using these operators is:

++variable or variable++

--variable or variable - -

Here "++" and "- -" are used as either prefix or suffix to a variable. When it is prefixed we call it pre increment or pre decrement operator. On the other hand when it is suffixed we call it post increment or post decrement operator.

The result of this operation though is increment or decrement in variable value by 1. The effect of this increment and decrement depends upon the way it is used in program.

Suppose we have statements like

int first = 15, second = 20, result;

result = first + second++;

Then here the value of "result" would be 35 and not 36. While the value of variable "second" at the end of statement two, would be 21. A value 1 will added to its old value 20.

Note that post increment operator, when used in an expression uses the old value of a variable to evaluate the expression, then it increments the value of variable by 1.

Similarly if we have statements

int first = 15, second = 20, result;

result = first + ++second;

Then the result value of "result" would be 36. While the value of variable "second" at the end of statement two, would be 21. *Note that pre increment operator, when used in an expression, first increments the value of variable by 1 then uses the new value of a variable to evaluate the expression.*

Conditional Operators

C provides us an operator known as ternary operator or conditional operator for condition checking. This operator is identified by combination of two symbols "? :" The syntax for using conditional operator is:

(condition) ? (True statement) : (False statement);

The condition here refers to a relational expression like first < second. Let us see how to use conditional operator by making a program to find minimum of two numbers. Figure 11.12 gives the code listing of the program.

11_6.c - SciTE

File Edit Search View Tools Options Language Buffers Help

1 11_6.c

```
/* Example 6: Program to illustrate use of conditional operator */

#include <stdio.h>

int main( )
{
    int first = 100, second = 50, minimum;
    minimum = (first <= second) ? (first) : (second);
    printf("The minimum value of the two is %d\n\n", minimum);
    return 0;
}
/* End of Program */
```

Figure 11.12 : Program illustrating use of conditional operator

Explanation

Here the first statement of the program defines three integer variables and initializes two of them. It also assigns values 100 and 50 to first and second respectively. Then we make use of conditional operator to check whether first is less than or equal to second. If the condition is true then minimum is assigned value of first, otherwise it is assigned value of second. We then print the message and value of minimum on screen. Finally we exit the program. The output of example 11.6 is shown in figure 11.13.

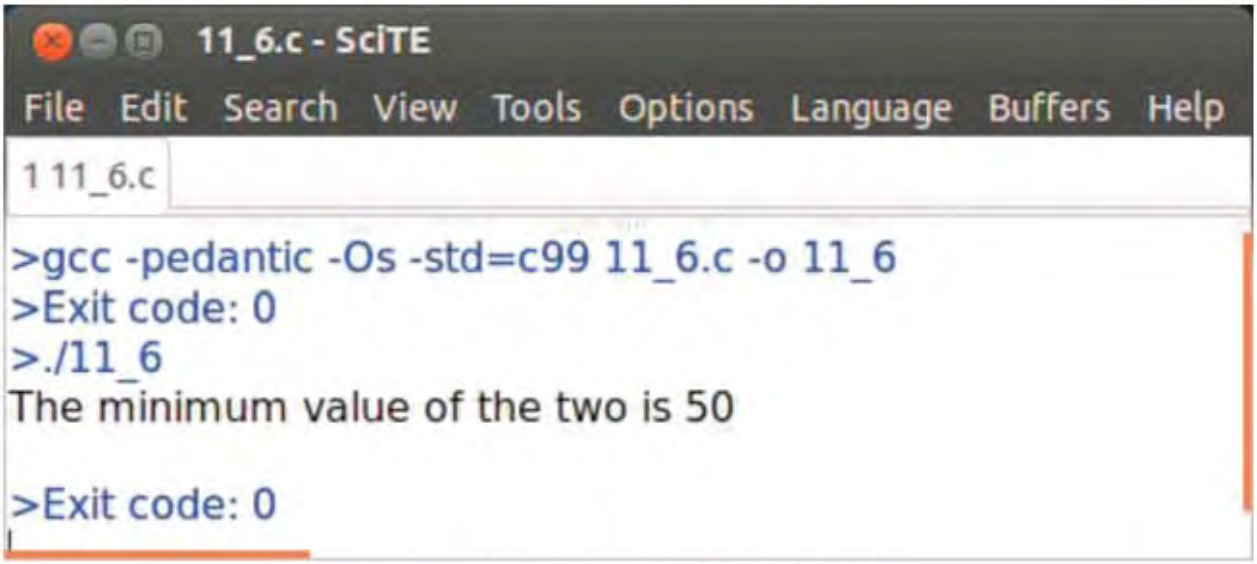


Figure 11.13 : Output of Example 11.6

Logical Operators

At times we may need to evaluate multiple conditions together to give some output. While at times we may get an output even if one out of the given set of conditions is satisfied. Such a relation is generally known as logical relation. For example consider a case where a student gives exam of two subjects. The student now will be declared as passed, if and only if he or she scores 35 or more marks in both the subjects. Here the relationship can be expressed as If (Marks in subject 1 >= 35) and (Marks in subject 2 >= 35) then pass else fail.

To represent such a relation, C provides logical operators. Table 11.6 gives the list and use of these logical operators.

Operator	Use
&&	Logical AND
||	Logical OR
!	Logical NOT

Table 11.6 : Logical Operators

Logical AND is used when all the given conditions must be satisfied. Logical OR is used when any one of the given conditions must be satisfied. Logical expression gives as an output values either 0 or 1. Here 0 refers to false and 1 refers to true. Table 11.7 gives the output of different logical operations.

Expression 1	Expression 2	! (Expression 2)	Expression 1 && Expression 2	Expression 1 || Expression 2
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Table 11.7 : Truth table (Results) for logical operations

As can be seen AND operator returns 1 only if both expressions return 1. While OR operator returns 1 when any of the expressions return 1.

Let us write a program that tells us the grade of a student based on the criteria as mentioned:

Grade	Marks
A	If percentage greater than or equal to 70
B	If percentage between 60 and 69
C	If percentage between 35 and 59
F	If percentage less than 35

Figure 11.14 gives the code listing of the program while figure 11.15 shows its output.

11_7.c - SciTE

File Edit Search View Tools Options Language Buffers Help

11_7.c

```
/* Example 7: Program to illustrate use of logical operator */

#include <stdio.h>
int main( )
{
    int percentage = 55;

    /* Verify whether percentage is between 0 and 100 using AND */
    if ( percentage >= 0  &&  percentage <= 100  )
    {
      /* Check whether percentage greater than 70 */
      if ( percentage >= 70 )
      printf("\nYou have secured A grade\n\n");

      /* Check whether  percentage between 60 and 69 */
      if ( percentage >= 60 && percentage < 70 )
      printf("\nYou have secured B grade\n\n");

      /* Check whether  percentage between 35 and 60 */
      if ( percentage >= 35 && percentage < 60 )
      printf("\nYou have secured C grade\n\n");

      /* Check whether percentage less than 35 */
     if ( percentage < 35 )
     printf("\nYou have secured F grade\n\n");
    }
    else    /* Executed when percentage below 0 or above 100 */
    {
     printf("\nYou have entered wrong value in percentage");
     printf("\nPlease enter value between 0 and 100\n\n");
    }
    return 0;
}
/* End of Program */
```

Figure 11.14 : Program illustrating use of logical operator

Explanation

Here the first statement declares and initializes an int variable called "percentage". Initially we check whether the value of "percentage" is between 0 and 100. If the answer is no then we move on to else part and print messages "You have entered wrong value in percentage", "Please enter value between 0 and 100" and then exit the program.

If the answer to the comparison is true, we then compare the value of percentage using if statement and logical conditions. The message specified in the if statement that returns true for the checked criteria is printed on screen see figure 11.15.

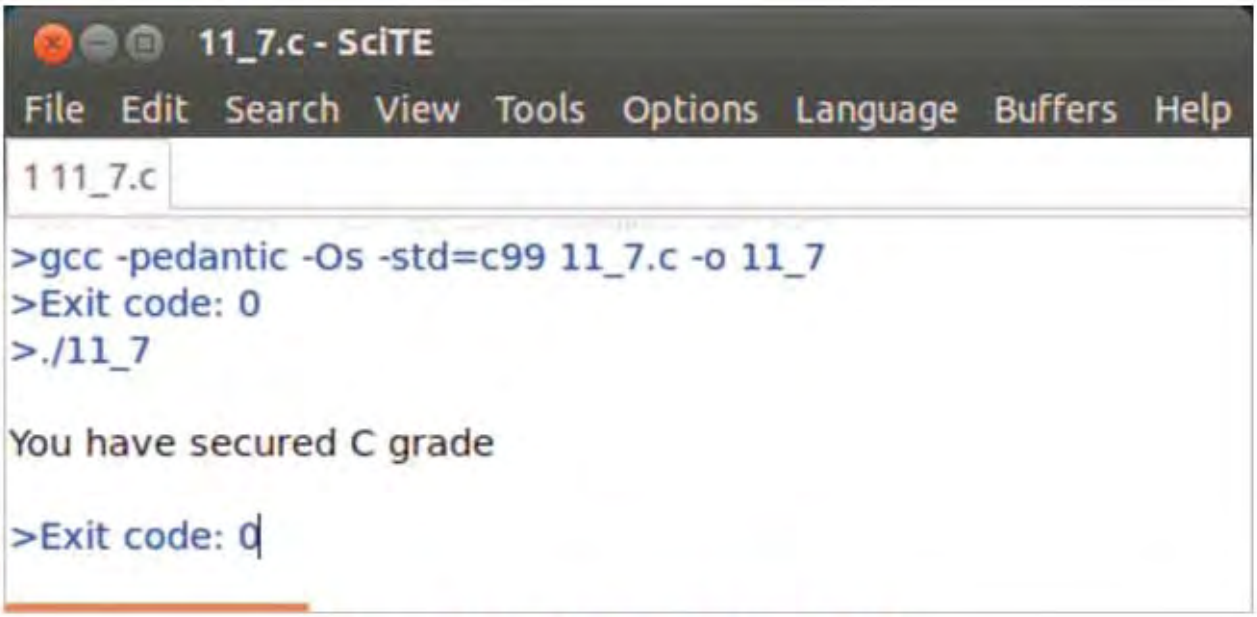


Figure 11.15 : Output of Example 11.7

Bitwise Operators

We have mentioned in this chapter that data is stored in the form of bits within the memory location. C allows us to operate directly at bit level using bitwise operators. Table 11.8 lists the bitwise operators available in C and their use.

Operator	Use
&	Bitwise AND
|	Bitwise OR
~	Bitwise NOT
^	Bitwise Exclusive OR
<<	Left Shift by number of bits specified.
>>	Right Shift by number of bits specified.

Table 11.8 : Bitwise Operators

To perform bitwise AND, OR, Exclusive OR (XOR), Left shift and Right shift we require two operands. NOT on the other hand only requires one operand. The value that the operands can hold is 0 and 1. Detailed discussion of these operators is out of scope of this book.

Special Operators

C provides special operators like sizeof( ), ",", ".", "→", "&" and "*". In this section we will discuss only "," and sizeof() operators. The "," operator is used on many occasions. The comma operator is also used in decision structures and control structures discussed later chapters.

The sizeof( ) operator is a special operator used to return size of bytes required to store an entity. For example size = *sizeof(int)* will assign value "4" to "size". This is so because data type int uses 4 bytes of memory space.

Figure 11.16 gives the code listing, while figure 11.17 shows the output of the program that makes use of comma and sizeof operator.

11_8.c - SciTE

File Edit Search View Tools Options Language Buffers Help

1 11_8.c

```
/* Example 8: Program to illustrate use of comma and size of operator */

#include <stdio.h>
int main( )
{
    float first, second, result;
    int  size;
    /* Use of comma operator */
    result = (first = 125, second = 15, first / second);
    printf("\nFirst = %f, Second = %f, Result = %f", first, second, result);

    /* Use of sizeof operator */
    size = sizeof(result);
    printf("\nSize allocated by compiler to variable result is %d bytes\n\n", size);

}
/* End of Program */
```

Figure 11.16 : Program illustrating use of special operator

11_8.c - SciTE

File Edit Search View Tools Options Language Buffers Help

1 11_8.c

```
>gcc -pedantic -Os -std=c99 11_8.c -o 11_8
>Exit code: 0
>./11_8

First = 125.000000, Second = 15.000000, Result = 8.333333
Size allocated by compiler to variable result is 4 bytes

>Exit code: 0
```

Figure 11.17 : Output of Example 11.8

Explanation

Here the first statement declares three float variables. Second statement declares an integer variable. The third executable statement is an expression that uses comma operator. The statement works as mentioned. The RHS is evaluated first. The first operation is assignment of 125 to "first", second operation is assignment of 15 to "second", the third operation is division of "first" by "second". Finally the value of division is assigned to "result". Next statement prints the value of "first", "second" and "result". Then we have assigned size with the value of number of bytes allocated to the variable result.

Evaluation of Expression

Until now we learnt the operators available in C and how to use them in expression. Let us now see how the expression would be evaluated. Consider the expression in statement

*result = first + second * third - fourth;*

Here first multiplication of second and third would be done. Its value of multiplication would then be added to first. And from this intermediate value fourth would be subtracted. If two operators with same priority appear in an expression, then they will be evaluated from left to right. To modify this sequence we can make use of brackets.

*result = first + second * (third - fourth) / fifth ;*

Here third - fourth will be evaluated first, the intermediate result will be multiplied with second then division with fifth will be done and finally addition with first will be done. See figure 11.18.

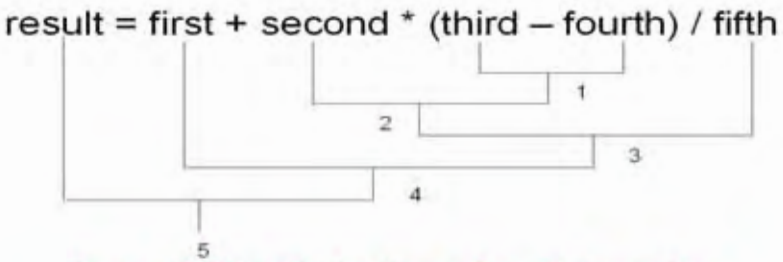


Figure 11.18 : Evaluation order of expression

Priority of Operators

The operators mentioned so far have their own priorities. As seen in the example result = first + second * (third - fourth) / fifth ; * was at higher priority then +, - or /. C language designers have pre decided these priority. The listing of some of which is given in table 11.9. The detail is available in appendix III.

Operator	Operation Used for	Associativity	Precedence
( )	Function call	Left to Right	First
[ ]	Array expression		
++	Increment	Right to Left	Second
- -	Decrement		
sizeof( )	Size of operand		
*	Multiplication	Left to Right	Third
/	Division		
%	Modulo division		
+	Binary addition	Left to Right	Fourth
-	Binary subtraction		

Table 11.9 : Summary of C operators

Type Conversion

An expression can be evaluated only if all the operands involved are of same data type. This property of an expression requires internal conversion of data types to be performed. For example if we have following code snippet

int year=2;

float principal, rate, interest;

interest = (principal * rate * year) / 100;

Then the evaluation of the expression shown above would be done as shown in figure 11.19.

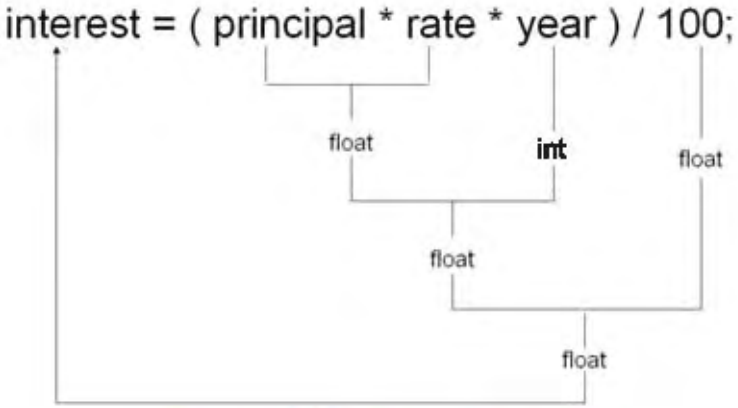


Figure 11.19 : Type Conversion

It is possible to override the internal conversion by performing a process called type casting. To type cast any value we use the syntax

(data type) variable or (data type) constant

Figure 11.20 gives the code listing of the program that makes use of type casting. The output of the program is shown in figure 11.21.

11_9.c - SciTE

File Edit Search View Tools Options Language Buffers Help

1 11_9.c

```
/* Example 9: Program to illustrate use of type cast */

#include <stdio.h>
int main( )
{
    int total_cost = 575 , quantity = 24;
    float cost_item;

    /* C expression for calculating cost per item without type casting */
    cost_item = total_cost / quantity;

    printf("\nCost of one item without type casting is %f", cost_item);

    /* C expression for calculating cost per item without type casting */
    cost_item = total_cost / (float) quantity;

    printf("\nCost of one item after type casting is %f\n\n", cost_item);
    return 0;
}
/* End of Program */
```

Figure 11.20 : Program illustrating use of type casting

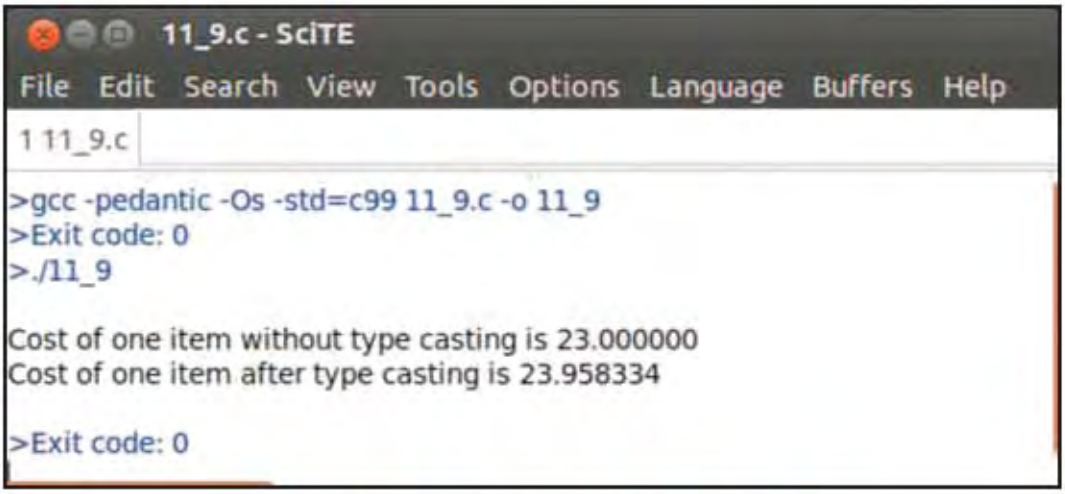


Figure 11.21 : Output of Example 11.9

Explanation

The program here is to find out cost per item, given the cost of 24 items. We have defined "total_cost" and "quantity" as integers while "cost_item" is float. As can be seen the first C expression for calculating cost per item gives value 23.000000 as both the operands used are integer. While the same expression when used along with type cast gives value 23.958334 which is desired result.

Storage Classes

The variables used in C program are generally stored in primary memory of the computer. It is also possible to store some variables in registers. C language provides us four storage classes namely, automatic, external, register, and static. These storage classes allows user to specify the location where he/she wants to store the variable. Let us now discuss them one by one.

Automatic Variables

All the variables by default have storage class as automatic. To explicitly define the variable as automatic, we use the keyword auto. The following syntax allows us to define an automatic storage class for a variable:

auto data type identifier;

For example, the definition auto int number; indicates that the variable number is of type integer and its storage class is automatic. Such variables are created automatically when the function is called and destroyed when the function returns the control back to the calling function. These variables are by default, are assigned a garbage value (value that does not have any significance to the user) and are stored in the primary memory.

External Variables

At times the programmer needs to share a variable between two functions or programs. We can use storage class external to share variables between two different functions or programs. It is necessary though that the variable must be defined as global variable in at least one program. The following syntax allows us to define an external storage class for a variable:

extern data type identifier;

For example, the definition extern char choice; indicates that the variable choice is of type character and its storage class is external. The default value of the external variable is zero, and they are stored in the primary memory.

Register Variables

To have fast access to variables, we can store the variables in the CPU registers. C language provides us a storage class called register to store the variables in the CPU registers. The following syntax allows us to define a register storage class for a variable:

register data type identifier;

For example, the definition register int counter; indicates that the variable counter is of type integer and its storage class is register. The register class variables are assigned garbage value by default.

Static Variables

The static storage class when used along with variable makes the variable value permanent within a specified region. A static storage class variable is stored permanently in the primary memory and, by default, is assigned value zero. The following syntax allows us to define a static storage class for a variable:

static data type identifier;

For example, the definition static int max; indicates that the variable max is of type integer and its storage class is static.

Summary

In this chapter we learnt about the basic data types like int, float, char and void. We also learnt about some user defined and derived data types like typedef, enum and arrays. We also saw how to assign values to the variables using assignment operators. Later we looked at various operators like arithmetic, relational, increment - decrement, conditional, logical, bitwise and some special operators. We saw how to create expression using different operators and how to evaluate them. Finally we saw four storage classes that C allows us to use.

Instruction for Teachers

All the data types discussed here are based on ANSI C99 standards. The examples discussed here show how to use the operators. Teacher can design more exercises and then give it to students.

EXERCISE

1. What is data type ? Explain its significance.
2. Explain the significance of void data type.
3. Explain the significance of user defined data type.
4. Explain the significance of derived data type.
5. State whether true or false :
   (a) char data type requires two bytes of memory space.
   (b) An unsigned integer variable can store value less than zero.
   (c) The data type double is used to store integer value.
   (d) To increase the range of int data type it is prefixed by keyword long.
   (e) Scientific form of real value is expressed as exponent * 10 mantissa

6. Choose the correct option from the following:

(1) Which of the following keywords refer to integer data type ?

(a) integer (b) Integer (c) INTEGER (d) int

(2) Which of the following refers to an empty value ?

(a) void (b) Void (c) char (d) float

(3) Which of the following refers to size of memory allocated to float as per gcc compiler ?

(a) 1 (b) 2 (c) 4 (d) 6

(4) Which of the following is a valid keyword in C that identifies valid character value ?

(a) char (b) character (c) CHAR (d) CHARACTER

(5) Which of the following is a user defined data type in C ?

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

(6) Which of the following is used to provide an alias to existing data types in C ?

(a) enum (b) def (c) pointer (d) typedef

(7) The symbol & belong to which of the following operator types in C ?

(a) Relational (b) Arithmetic (c) Logical (d) Bitwise

(8) The symbol | belong to which of the following operator types in C ?

(a) Relational (b) Arithmetic (c) Logical (d) Bitwise

(9) The = symbol is used for which of the following operations in C ?

(a) Equality Check (b) Assignment

(c) Comment (d) Formula

(10) Which of the following refer to the meaning of value++ in C ?

(a) Post Increment (b) Post Decrement

(c) Pre Increment (d) Pre Decrement

LABORATORY EXERCISE

1. Write a C program to convert given meters into millimeters.
2. Write a C program to convert given Fahrenheit into Celsius using formula C = (F-32) * 5/9.
3. Write a C program to find average of three numbers.
4. Write a program to find out how many tiles of 50 square cms can be fitted in the plot of 2000 sq meters.
5. Write a C program to find area of rectangle.
6. Write a C program to check whether a given number is divisible by other given number.
7. Write a C program to calculate the compound interest on a loan amount of Rs. P taken at the rate of R% for N years. The values of P, R and N are to be specified by student.
8. Write a C program to find volume of a cube.
9. Write a C program to calculate area of circle.
10. Write a C program to calculate the simple interest on a loan amount of Rs. P taken at the rate of R% for N years. The values of P, R and N are to be specified by student.



Using I/O Operations

All programming languages allow to read input (known as input operations) and to write output (known as output operations). Here, input means to read data from any input device like keyboard and output means to write data to output devices like monitor, printers, etc. Input/Output operations are more popularly known as I/O operations. The C language does not have any built-in statement to perform input and output operation. These data input and output operations are carried out by the standard input/output built-in library functions. So far we have used scanf() as a standard input and printf() as a standard output function in our programs. In this chapter we will discuss how to perform I/O operations using functions like getchar(), getch(), gets(), scanf(), printf(), putchar(), putc(), puts(), etc. We will also discuss how to perform formatted input and output operations.

We use variables to process data in our program. There are two basic ways to give input to the variables. One way is to use assignment operator. For example, the statement number=5; will give input 5 to the number variable. Second way is to read data from user during execution of a program. For this you may use any inbuilt functions related to input operations. Let us discuss commonly used inbuilt input functions.

Inbuilt Input Functions

Input in a program can be possible through any of the input devices like keyboard, mouse etc. C provides several inbuilt input related functions which are stored in a C library. To use any inbuilt function from C library, we have to include respective library of that function in the beginning of program using **#include** statement. You may have noticed that in many programs we have used following statement in beginning of our program:

```
#include<stdio.h>
```

The name stdio.h stands for standard input-output header file. This header file contains various input and output operations related functions. The statement #include informs the compiler to find stdio.h file and place its contents in the beginning of our program. The content of header file is then become a part of our program. More discussion related to using C library function is given in chapter of C Function. Note that on some of the computer system it is not necessary to include this header file when we use scanf() function. In following section now we discuss input functions getchar(), getch(), getc() and gets().

getchar()

One of the simplest way to read a character in a C program at runtime is to use the getchar() function. The standard form of getchar() is as under:

```
variable_name = getchar();
```

variable_name is any valid C variable name with character as a data type. The getchar() function has no parameters. Every time when we call it, it reads single character input. The function returns an int, the ASCII code of the relevant character, but we can assign the input given by user to a char variable.

Let us try to understand the use of getchar() function using example 12.1. Here we want to use the answer typed in by the user to print a message in our program. Figure 12.1 gives the code listing of the example 12.1 while figure 12.2 shows its output.

9-1.c - SciTE

File Edit Search View Tools Options Language Buffers Help

9-1.c

```
/* Example 1: Program to demonstrate use of getchar() function. */

#include<stdio.h>
int main()
{
      char answer;

      printf("Would you like to study? \n");
      printf("Enter 'Y' for yes and 'N' for no: ");

      answer = getchar();

      if(answer =='Y' || answer =='y')
            printf("Go to study room...\n");
      else
            printf("Go to play ground...\n");

      return 0;
}
// End of Program
```

Figure 12.1 : Code listing of Example 12.1

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 9-1.c
kpp@ubuntu:~$ ./a.out
Would you like to study?
Enter 'Y' for yes and 'N' for no: Y
Go to study room...
kpp@ubuntu:~$
```

Figure 12.2 : Output of Example 12.1

Explanation

The first statement of example 12.1 declares a variable answer of character data type to store a single character. When getchar() function is encountered, the program will wait for the user to press a key. Any character pressed by the user from keyboard will be stored in the answer variable. The same will also be echoed on the screen. Say for example if the user enters character 'Y' or 'y' then a message "Go to study room..." will be displayed. If user enters any other character then message "Go to play ground..." will be displayed.

getch()

The function getch() is also used to read a character from the user. The difference between getchar() and getch() is that the character keyed in will not be echoed on the screen, i.e. the character entered by the user will not be visible on screen. This function is useful when we don't want to show a character typed by the user on screen. Replace the function getchar() with getch() in example 12.1 and observe the difference between these two functions.

getc()

Like getchar() and getch(), getc() is also used to read a single character. But the difference here is that getc() reads a character from a file instead of the standard input device. More discussion related to getc() is out of scope of your syllabus.

gets()

So far we have seen library function getchar(), getch() and getc() which can read only one character at a time. To read two characters, we can call getchar() function two times. Calling getchar() function many times is not good logic when we want to read a group of characters. The better alternative is to use gets() function to read a string. Note that we call a group of characters as a string. The general syntax of gets() function is as under:

gets(variable_name);

The gets() function takes one string variable as an argument. Here variable_name is the character array. More about character array is discussed in Array chapter. During execution of program when gets() is encountered, it will wait for user to enter characters from the input device. The function will read characters until a new line character is entered by the user and then appends a null character ('\0') to the end of string. After execution of this function, all the characters and null value at the end of all the characters are stored into the declared variable_name. Let us try to understand the use of gets() function using example 12.2. Figure 12.3 gives the code listing of the example 12.2 while figure 12.4 shows its output.

9-2.c - SciTE

File Edit Search View Tools Options Language Buffers Help

9-2.c

```
/* Example 2: Program to demonstrate use of gets() function. */

#include<stdio.h>
int main()
{
      char state_name[30];
      printf("What is your state name? \n");

      gets(state_name);

      printf("Your state name is %s \n", state_name);
      return 0;
}
//End of Program
```

Figure 12.3 : Code listing of Example 12.2

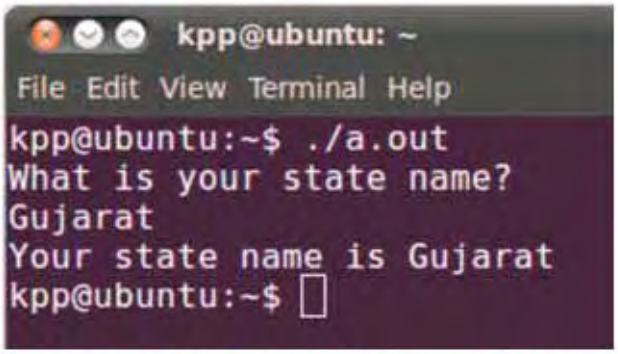


Figure 12.4 : Output of Example 12.2

In the example 12.2, the user input will be stored in state_name variable. The printf() function displays the same string on the screen. Details about printf() statement is discussed later in this chapter.

Note that during compilation of any program containing gets() function, you may get warning message like "the gets function is dangerous and should not be used". But as it is only warning message, you may continue to execute your program.

Formatted Input

There are some situations in our program where use of input functions discussed so far are not much useful. The getchar(), getch() and getc() can be used to read only single character at a time whereas gets() can be used to read multiple characters until user press enter key. What happens if user wants to input data like

33 Mudra Ahmedabad

In the said example first part of data is the roll number of student; second and third parts of data are name and city of the student respectively. As can be observed roll number is of integer data type and remaining two are of character data type.

C provides the facility called formatted input functions using which we can read such type of data. We have already used one of the formatted input function scanf() in many examples. Let us explore various options available for formatted data input using scanf() function.

scanf()

The **scanf()** stands for **scan** formatted. This function allows us to input data into particular data format like int, char, float, etc. The common syntax of scanf() function is as under:

scanf("control string", &variable1, &variable2, ...&variableN);

The *control string* specifies the data format in which values of variables are to be stored. The variable1, variable2, ...variableN are names of variables. These variable names are preceded by &(ampersand) sign and are separated by comma ','. The & sign is known as address of operator in C language and it specifies the memory location where the input given by user is stored. This may be clear from the figure 12.5.

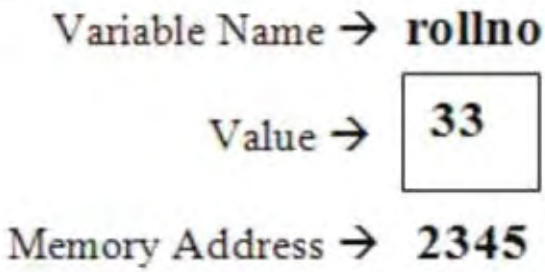


Figure 12.5 : Memory Layout

As shown in figure 12.5 variable name is rollno and its value is 33. Actual value of variable is stored at memory address 2345(this number is just an example; it may be any valid memory address of computer). In C each variable is identified by its name as well as its memory address. The C compiler uses memory address for processing whereas we refer memory location by its name in our program. More discussion related to the address of operator is out of scope of this book. You will learn this concept in detail while studying C pointers.

The *control string* is also known as **format string**. This will helps the compiler to interpret data input given by the user. Control string also specifies the order in which variable values appear in the input. Each format must be preceded by '%' sign and a character specifying data type. Let us see how to use control string in our program with example.

Reading Integers

Following program segment can be used to read three integer values from the users which are stored into marks1, marks2 and marks3 variables.

```
int marks1, marks2, marks3;
scanf("%d %d %d", &marks1, &marks2, &marks3);
```

During execution of above statement, if user enters: 70 80 90 then 70 will be stored in marks1, 80 in marks2 and 90 in marks3. We can also specify the field width of a number to be read along with %d as under:

```
scanf("%2d %4d", &marks1, &marks2);
```

During execution of previous scanf(), if user enters: 70 1234 then 70 will be stored in marks1 and 1234 in marks2 variable.

But for previous scanf() statement, if user enters: 1234 70 then variable marks1 will be assigned value 12 (because of field width %2d) and mark2 will be assigned 34 (which is unread part from 1234). The value 70 will remain unused for the said scanf() statement.

Reading Real Numbers

To read real numbers (floating point numbers) like percentages of a student in a program, following program segment can be used.

```
float  per1, per2;
scanf(" %f %f ", &per1, &per2);
```

Here we have defined two float variables per1 and per2. The scanf statement stores float values in these variables. While executing above code, if user enters: **85.25 90.65** then 80.25 will be stored in variable per1 and 90.65 in variable per2. The scanf() statements reads real numbers using specification **"%f "**. If user wants to read a number with **double** data type then specification **"%lf "** is to be used in place of "%f ".

Reading Character and Word

We have seen how to read a character and characters (string) using in-built function like getchar() and gets(). Same task of reading a characters and a words can be achieved using scanf() statement with %c and %s specification respectively. Observe the given program segment:

```
char c1, c2;
char city[20];
scanf(" %c %c %s", &c1, &c2, city);
```

During execution of the given scenario if user enters data like: **A B Gandhinagar** then 'A' will be stored in variable c1, 'B' in c2 and 'Gandhinagar' in character array named city. More discussion related to character array is given in Chapter 15 of this book.

Note that while reading string using scanf() statement, **&** (ampersand) is not required with the variable name. Also remember that %s specification terminates its reading when blank space encounters in input. See the following example :

```
char city[20];
scanf("%s ", city);
```

During execution, if user enters: **New Delhi** then only "New" will be stored into variable city. The word "Delhi" is ignored by the scanf() statement as %s specification terminates its reading at space after "New".

Restricted user input with %[characters] and %[^ characters]

C language scanf() function provides wonderful facility of %[characters] using which we can specify the permissible characters in the input string. If user input string contains any character which is not listed in the %[characters] specification then the string will be terminated at the first occurrence of such character. Let us try to understand this concept using example 12.3. Figure 12.6 gives the code listing of the example 12.3 while figure 12.7 shows its output.

9-3.c - SciTE

File Edit Search View Tools Options Language Buffers Help

9-3.c

```
/* Example 3 Program to demonstrate use of %[characters] specification. */

#include<stdio.h>
int main()
{
      char student_name[30];

      printf("Enter your name containing only alphabets and space: \n");
      scanf("%[a-z A-Z]", student_name);
      printf("Your name stored in variable is %s \n", student_name);
      return 0;
}
//End of Program
```

Figure 12.6: Code listing of Example 12.3

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 9-3.c
kpp@ubuntu:~$ ./a.out
Enter your name containing only alphabets and space:
Ragi Bharat Patel
Your name stored in variable is Ragi Bharat Patel
kpp@ubuntu:~$ ./a.out
Enter your name containing only alphabets and space:
Ragi B. Patel
Your name stored in variable is Ragi B
kpp@ubuntu:~$
```

Figure 12.7 : Output of Example 12.3

In example 12.3, specification %[a-z A-Z] in scanf() specifies that only a-z, space and A-Z are accepted and stored into variable student_name. Hence in first execution, full name of student is stored into variable. But during second execution, user input contains dot (.) which is not listed into specification hence only "Ragi B" is stored into variable.

By using %[^characters] specification in the scanf(), the process of inputting the value is immediately terminated when any of the character from the list is encountered. Let us try to understand this concept using example 12.4. Figure 12.8 gives the code listing of the example 12.4 while figure 12.8 shows its output.

9-4.c - SciTE

File Edit Search View Tools Options Language Buffers Help

9-4.c

```
/* Example 4:  Program to demonstrate use of %[^characters] specification. */

#include<stdio.h>
int main()
{
      char user_data[100];

      printf("Enter your details:\n");
      printf("Enter * to exit:\n");
      scanf(" %[^*] ", user_data);
      printf("Your details stored in variable is \n%s \n", user_data);
      return 0;
}
//End of Program
```

Figure 12.8 : Code listing of Example 12.4

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 9-4.c
kpp@ubuntu:~$ ./a.out
Enter your details:
Enter * to exit:
My name is Purva.
My mother's name is Krina*
Your details stored in variable is
My name is Purva.
My mother's name is Krina
kpp@ubuntu:~$
```

Figure 12.9 : Output of Example 12.4

During execution of example 12.4 we can input any character (including new line character). When * is detected in the input string, the process of accepting input string is stopped. Note that in our example, we will be able to store only 100 characters as user_data variable length is declared as 100.

Reading Mixed Data

It is possible to read different types of data (like integer, float, char, string) using a single scanf() statement. In such cases we have to make sure that input data items must match with the control specification of scanf() in order and data types. Observe the following program segment to understand this concept.

```
int roll_no;
char grade, name[30];
float percen;
scanf("%d %f %s %c", &roll_no, &percen, name, &grade);
```

If during execution the input specified is: **11 90.55 Vani A** then 11 is stored in roll_no, 90.55 in percen, 'Vani' in name and 'A' in grade variable. But for the same program code, if user inputs data as : **11 Vani A 90.55** then the system will give error because user had entered a characters when compiler was looking for floating point number. In this case scanf() statement will terminate its reading process after reading the first value.

The list of characters which are used in scanf() control string to read different types of data is given in table 12.1

Data Type	Corresponding Character
For reading a decimal integer	%d
For reading a character	%c
For reading a floating point value	%f or %e or %g
For reading a string	%s
For reading a unsigned integer	%u
For reading a short integer	%h
For reading a long integer	%ld
For reading a double	%lf
For reading a long double	%L

Table 12.1 : Characters used in scanf() control string

Inbuilt Output Functions

The computer system performs three basic tasks: input, process and output. We had seen that how inbuilt input functions can be used to read data from user. Now let us study how inbuilt output functions can be used for displaying processed data. The output can be obtained on output devices like monitor, printer and file. We will discuss how output can be displayed on the monitor, which is considered as a standard output device in C. We will not discuss file and printer management as it is out of scope of your syllabus.

Like input, sending output to output devices is possible through C library functions. The inbuilt output functions are available in the <stdio.h> header file. Let us discuss the output related inbuilt functions: putchar(), puts() and printf().

putchar()

The function putchar() can be used to write a single character to the standard output device. The general syntax of putchar( ) is

putchar(character);

Here character may be the char type variable or it may be any valid C language character. When this function is executed, character will be displayed on the monitor. Let us try to understand this concept using example 12.5. Figure 12.10 gives the code listing of the example 12.5 while figure 12.11 shows its output.

9-5.c - SciTE

File Edit Search View Tools Options Language Buffers Help

9-5.c

```
/* Example 5: Program to demonstrate the use of putchar() function. */

#include<stdio.h>
int main()
{
      char grade;

      printf("Enter your grade:\n");
      grade=getchar();      //reads a single character from user
      printf("Your grade is ");
      putchar(grade);       //displays a single character to screen
      printf(". \n");
      return 0;
}
//End of Program
```

Figure 12.10 : Code listing of Example 12.5

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 9-5.c
kpp@ubuntu:~$ ./a.out
Enter your grade:
A
Your grade is A.
kpp@ubuntu:~$
```

Figure 12.11 : Output of Example 12.5

The program will read a single character using getchar() function and displays it using putchar() function on the screen.

The putchar() function has one limitation that it can display only one character at a time. To display multiple character at a time you may use loop concept of C. The simpler solution to display multiple characters is puts() function.

puts()

We may use puts() function to display multiple characters (string or character array) on an output device. The general syntax of puts( ) is as under:

puts(variable_name);

where *variable_name* is the character array or string. The puts() functions writes the content stored into *variable_name* to the monitor till the null character is encountered. Note that every string contains a null character at the end but puts() function will not display this character on screen. Observe the example 12.6 to make this concept clear. Figure 12.12 gives the code listing of the example 12.6 while figure 12.13 shows its output.

9-6.c - SciTE

File Edit Search View Tools Options Language Buffers Help

9-6.c

```
/* Example 6: Program to demonstrate the use of puts() function.*/

#include<stdio.h>
int main()
{
      char state[20];

      printf("Enter your state name:\n");
      gets(state);
      printf("Your state name is ");
      puts(state);
      return 0;
}
//End of Program.
```

Figure 12.12 : Code listing of Example 12.6

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ ./a.out
Enter your state name:
Gujarat
Your state name is Gujarat
kpp@ubuntu:~$
```

Figure 12.13 : Output of Example 12.6

The program will reads multiple characters at a time using gets() function and stores it to the state variable. Using puts() function value stored into state variable is displayed on the screen.

Formatted Output

The output functions discussed so far give output without any formatting. The value stored in the variable is simply displayed on screen. But sometimes to prepare reports from our program, we may want to

display output with formatting which makes it good in look and easy to understand. The printf( ) function provides features which can be used to display output with different types of formatting.

printf()

The printf( ) function is used to display formatted output on screen. The general syntax of the printf( ) is as under:

printf("control string", var1, var2, ...varN);

where var1, var2 ...varN may be variables or constants or expressions. The output format specification of variables is given in control string. The control string may contain all or some of the following items :

- Set of characters (string) which will be printed on monitor as they appear in control string
- Format specifier for each variables
- Escape sequence characters like \n (new line), \t(tab), \b(back space)

Like scanf( ), the control string entries are separated by space and preceded by '%'. The later part of printf( ) contains list of variables whose values are to be printed. These variables must match in number, order and its types with format specification given in control string.

Followings are some of the simple examples of printf( ) statements which we had used in our programs so far to display output on the screen. These statements will print output without any formatting.

- printf("Hello World");
- printf("Your age is %d", age);
- printf("Your name is %s", student_name);

To display formatted output with different types of alignment, we may use format specification in the control string of printf( ) statement. The control string has following general format:

%l.m p

Here, 'l' is an integer number specifying total character positions in which the variable value is to be printed. 'm' is used for specifying the number of digits after decimal point(of real number) or number of characters to be printed from a string variable. Specifying value of l and m are optional. The 'p' shows the type of variable. The list of different data types used with printf( ) is given in table 12.2

Data Type	Corresponding Character
For printing a decimal integer	%d
For printing a character	%c
For printing a floating point value without exponent	%f
For printing a floating point value in exponent form	%e
For printing a string	%s
For printing a unsigned integer	%u
For printing a long decimal integer	%ld
For printing a double	%lf
For printing a long double	%Lf
For printing integer in octal form	%o
For printing integer in hexadecimal form	%x

Table 12.2 : Data type used in printf( ) control string

Let us try to understand the code listing of example 12.7 shown in figure 12.14. It shows various formats of printf( ) statement.

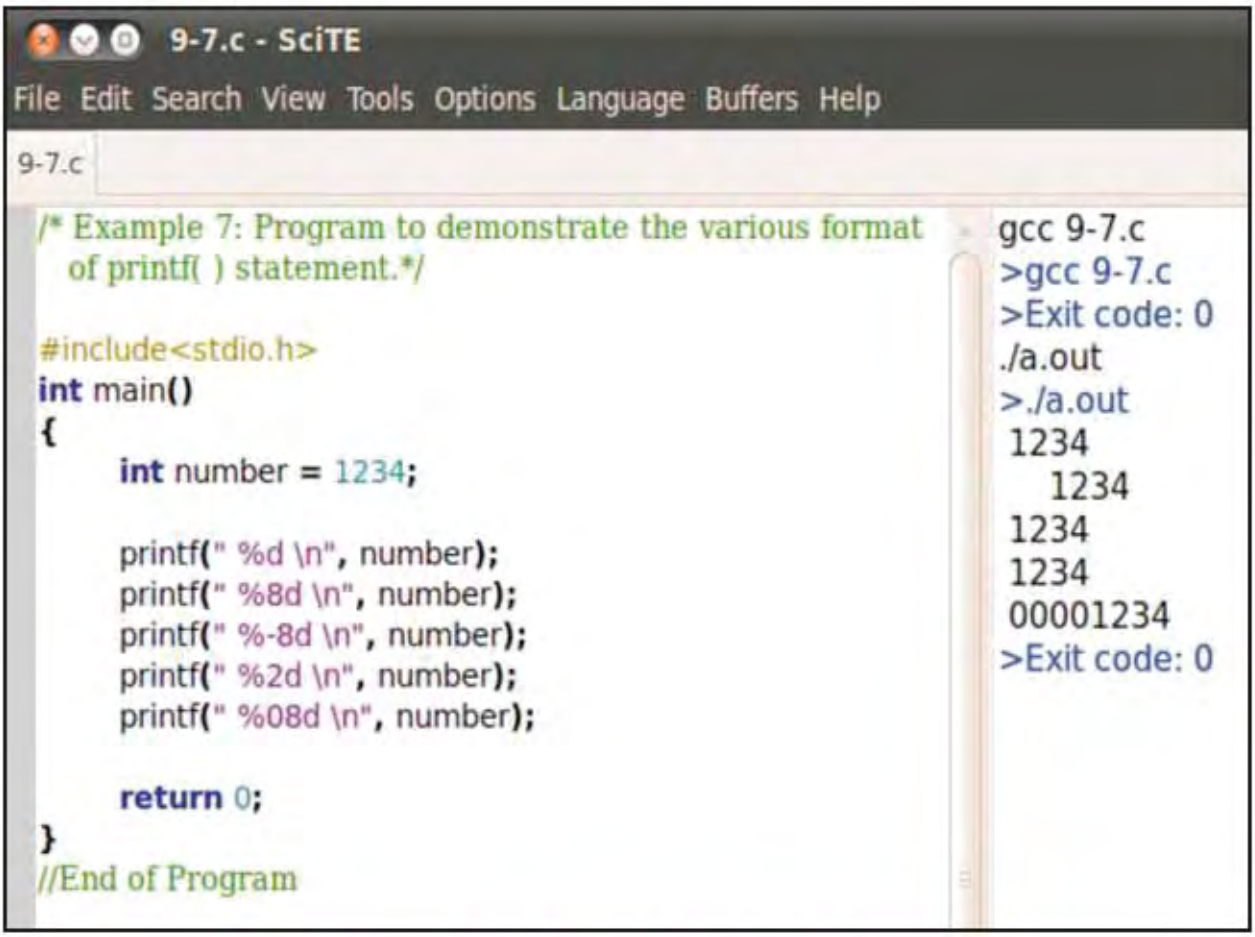


Figure 12.14 : Code listing and output of Example 12.7

Explanation

As can be observed in the first output by default the printf( ) statement uses left justification to display the output. The second printf( ) statement uses total 8 characters width to display output with right side justification. Third printf() statement shows that it is possible to print output with left side justification using minus sign(-) after % character. In fourth printf( ) statement there is no effect of specifying %2d as the specified width is less then width of actual variable. Last printf( ) statement shows that it is possible to print additional padding characters (zero) before the actual value of number variable.

In previous example we had seen that how integer variable can be displayed with different types of formatting on screen. Let us check that how to specify various formatting options with floating point/real numbers. Recall the general specification %1 .m p where m indicates number of digits to be displayed after decimal point. When float variable is displayed, it is rounded to m decimal places and printed right justified in the width of l columns. By default 6 digits are displayed after decimal point when m is not specified. Figure 12.15 shows how to use printf( ) statements to display float variable f1 with 123.456 value using different formatting.

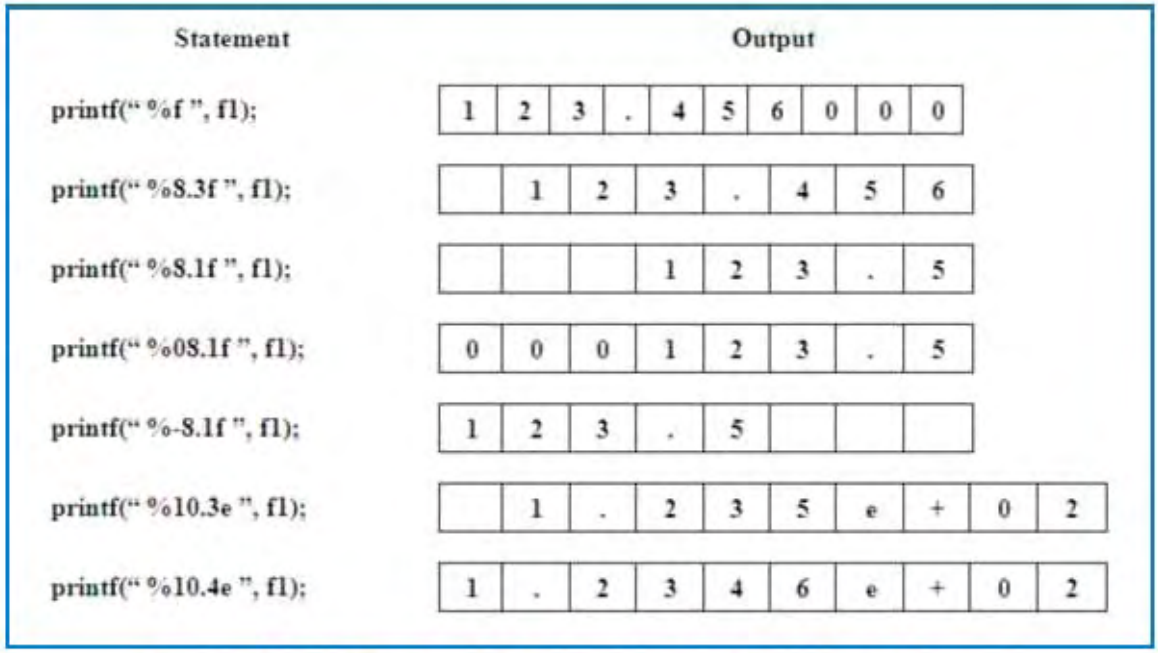


Figure 12.15 : Float variable displayed using different formats

Using printf( ) statement formatting can also be applied to character and string. The format specification is similar to that of real number. We use value of l to specify the field width for display and the value of m for number of characters to be printed. The string in output will be printed left justified when l is not specified. When l is specified the right justified string will be displayed on screen. Like integer format specification, by writing minus sign, we can print the string left justified.

Let us make string output format specification concept clear using following printf( ) statements. We use string (character array) variable str with value "GUJARAT INDIA" as value stored in it. Figure 12.16 shows various formatted output.

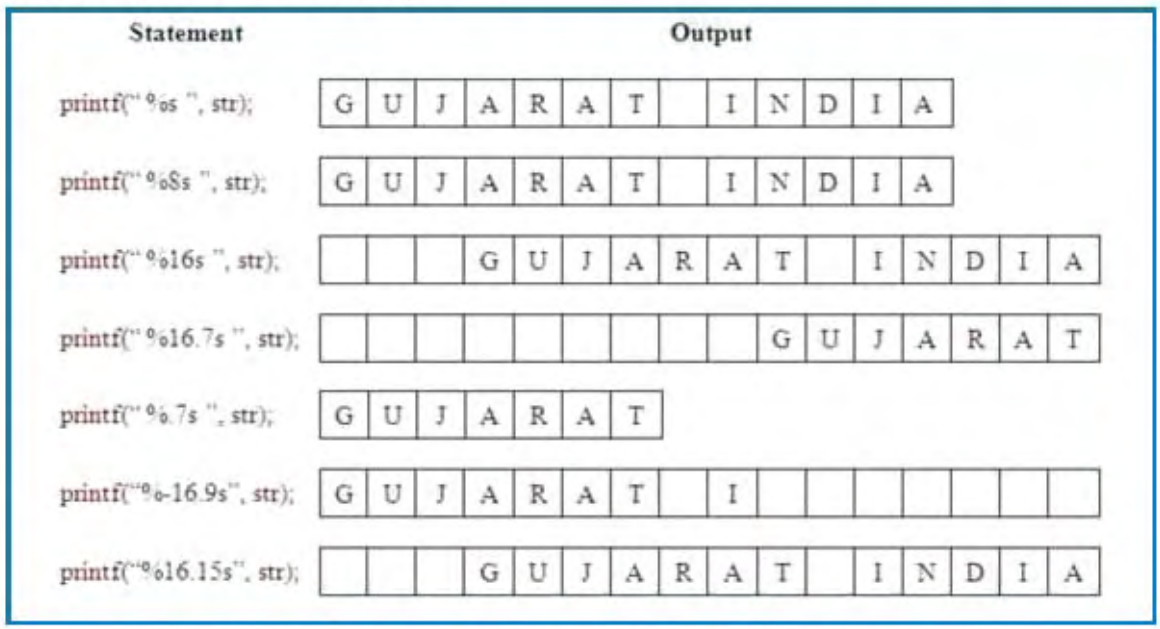


Figure 12.16 : Character variable displayed using different formats

In last printf( ) statement the value of m is 15 which is greater than length of string stored in str, hence all characters from str variable will be displayed with right justification.

Summary

In this chapter we had learnt that how formatted input and output related inbuilt function can be used in our program. We can display output without formatting also, but the output displayed with the help of format specification makes it more readable and attractive to the user. Now it is upto the programmers how they apply these functions to make their programs more attractive.

EXERCISE

1. How is the control string of scanf( ) different than the control string of printf( ) ?
2. What is the purpose of formatted output functions?
3. Discuss the use of formatted input functions in a program. Giving suitable example, explain how scanf() function can be used to read the integer and real numbers.
4. Explain the concept of restricted user input with %[characters] and %[^ characters] with suitable examples.
5. State whether true or false :
   (a) The function getchar( ) can be used to read multiple character at a time.
   (b) The function gets( ) can be used to read a single character.
   (c) The single printf( ) statement may generate multiple lines of output.
   (d) The %.10s format specification will print first 10 characters of a given string.
   (e) A single scanf( ) statement can be used to read multiple values.
   (f) By default real values are printed with a precision of six decimal points.
6. Do as directed :
   (1) State the errors if any in the following program statements.
      (a) scanf("%d %c", &number, city_code);
      (b) scanf("%d %d %s" \n, &marks1, &marks2, city_name);
      (c) scanf("%d %f %c %s", &num1, &price, item_code);
   (2) State the output produced by the following statements.
      (a) printf("%d %c %f", 101,'X', 20.20);
      (b) printf(" %3d %8.2f ", 12345, 222.123);
      (c) printf(" %.5s", "Hello World");
      (d) printf(" %15.8s", "Hello Student");
      (e) printf("%0.5s", "Hello World");

(3) Write output of following program.

```
#include<stdio.h>
int main( )
{
    float result = 98.12345;

    printf("Your result is %f \n", result);
    printf("Your result is %.2f \n", result);
    printf("Your result is %5.3f \n", result);
    return 0;
}
```

(4) Find errors if any, correct it and then write output of the following program.

```
#include<std.h>
int main( )
{
    int sum = 1234;
    float price = = 550.50;
    char City[20] = "Ahmedabad";

    printf("The sum is %d....", price);
    printf("The price is %s...", price);
    printf("The City name is %.3s...", city);
    return 0;
}
```

7. Choose the correct option from the following :

(1) Which of the followings are useful in C language I/O operations ?

(a) Monitor (b) Keyboard (c) Mouse (d) All of these

(2) Input in a program can be possible using which of the following device ?

(a) Keyboard (b) Speaker (c) Printer (d) Monitor

(3) Which of the following is not an input functions ?

(a) getchar() (b) getch() (c) puts() (d) gets()

(4) What is the output of statement: printf("%4s", "Krusha"); ?

(a) Krus (b) usha (c) Krusha (d) Krush

(5) Which of the following function is more appropriate for accepting a string ?

(a) getch() (b) gets() (c) getchar() (d) getc()

LABORATORY EXERCISE

1. Write a C program to read string "Hello Gujarat" from the user and display it in the following different format:

   (a) Hello (b) Gujarat (c) Hello Gujarat (d) Hello G

2. Write a C program to display the Multiplication table based on user input. Use various output formatting options to display it with proper alignment.

3. Write a C program to read only capital letter from the users. [Use the concept of restricted input provided by C].

4. Write a C program to read few integers from users. Display all integers on the screen with right justification.

5. Write a C program which will read following the following numbers.

   123.45 34.56 -7878.123 -0.965

   Store these numbers in to appropriate data type variables, print the rounded numbers to the nearest integers.

6. Write a C program to read value of A and B, then print the result in one line for following expressions:

   (i) (A + B) / (A-B) (ii) (A+B) (A-B) (iii) (A*A)(B+B)



Decision Structures

Most of the C programs presented so far follow the sequential structure of program execution. The instructions are executed one by one sequentially in the order in which they appear in a program.

In real life practice during execution of C program, we may want to execute only some of the statements based on some conditions. For this purpose it is required to change the flow of the sequence of instructions. C language provides facility through special kind of statements that can change the flow of a sequence of instructions in the program. Such statements are called decision structure statements.

Need for Decision Structures in C

Decision structure statements help us to jump from one part of the program to another part of program based on result of some conditions. Sometimes decision structure statements are also known as selective structures statements, branching statements or decision making statements. As these statements are controlling the flow of execution, they are also known as control statements.

C language provides two basic types of decision structure statements: ***if*** and ***switch.*** In this chapter, we will discuss how ***if*** and ***switch*** statements can be used to change the flow of instructions in the program.

The if Statement

The ***if*** statement is one of the powerful decision making statement which can be used to transfer control of instruction execution. The ***if*** statement can be used in following different ways:

- Simple if statement
- if-else statement
- nested if statement and
- else-if ladder statement

Simple if statement

The simplest form of the decision structure statement is the if statement. It is frequently used in the program for decision making and allowing us to change the flow of program execution.

Example 13.1 : Let us try to understand a C program that asks to input marks of one subject and display the congratulation message if marks entered by user are greater than 80. Figure 13.1 gives the code listing of the example 13.1 while figure 13.2 shows its output.



Figure 13.1 : Code listing of Example 13.1

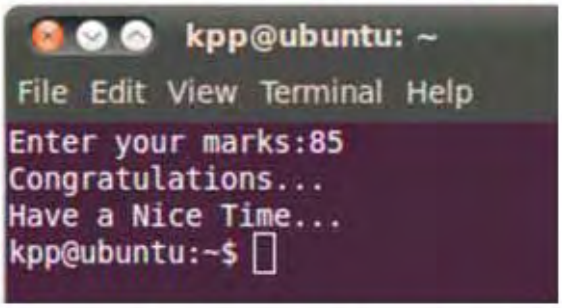


Figure 13.2 : Output of Example 13.1

Explanation

The first statement in the program after main() function declares the variable named marks. Second statement clears the screen. The printf statement is used to display the message to user. The scanf statement is used to read a value from user and store the entered value in the marks variable. The if statement checks the condition: Is entered marks are greater than 80? If the test expression (marks > 80) is evaluated to true then it executes the printf statement displaying the "Congratulations..." message. If the test expression (marks > 80) is evaluated to false then the printf statement displaying the "Congratulations..."message is skipped. The next printf statement always displays the "Have a Nice Time..." message.

Syntax of simple if statement

The if statement has following syntax:

```
if (test expression)
        {
                Statement-block;
        }
```

Here the test expression is any valid C language expression. The logical operators are used in test expression. The statement-block may contain any valid C language statement or statements. Note that test expression part should not end with a semicolon.

The syntax says if the test expression is true then executes the statement-block and if the test expression is false then computer skips the statement-block and executes the next instruction of the program.

C programming language assumes any **non-zero** and **non-null** values as true and if it is either **zero** or **null** then it is assumed as false value. Figure 13.3 shows the flow chart of simple if statement.

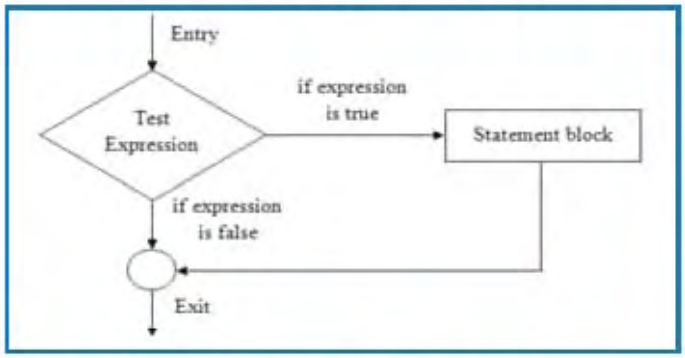


Figure 13.3 : Flowchart of Simple if Statement

The if ... else statement

In simple if statement, there is only one statement block, which gets executed only when test expression is true. Simple if statement takes no action in case when the test expression is false. What happens if we want to take some action in case of false answer from test expression? For example if marks of student are greater than 40 then we want to display message "Student Passed" otherwise we want to display message "Student Failed". In such cases, the if...else structure is useful.

Example 13.2 : Let us try to understand a C program that asks to input marks of one subject and display the message "Congratulation...you are passed" if entered marks are above 40 otherwise display message "Better Luck Next Time...you are failed". Figure 13.4 gives the code listing of the example 13.2 while figure 13.5 shows two different runs of output.

10-2.c - SciTE

File Edit Search View Tools Options Language Buffers Help

10-2.c

```
/* Example 2: Program to illustrate use of if..else statement. */
#include<stdio.h>
#include<stdlib.h>

int main()
{
    int marks;                          // declare a variable
    system("clear");                    // clears the screen
    printf("Enter your marks:");        // message to the user
    scanf("%d", &marks);                // read a number from user

    if ( marks > 40 )                   // check whether marks greater than 40
    {
        printf("Congratulations...you are passed.\n");  // Condition true
    }
    else
    {
        printf("Better Luck Next Time...you are failed.\n");  // Condition false
    }

    printf("Have a Nice Time...\n");
    return 0;
}
/* End of Program */
```

Figure 13.4 : Code listing of Example 13.2



Figure 13.5 : Output of Example 13.2

Explanation

The first statement of program declares marks variable. Second statement clears the screen. Third and fourth statement displays the message and read marks from user respectively. Fourth statement checks whether entered marks are greater than 40 or not. If user enters marks greater than 40 then message "Congratulations…you are passed." will be displayed otherwise the message "Better Luck Next Time…you are failed." will be displayed. The next printf statement always displays the message "Have a Nice Time…" message.

Syntax of if...else statement

The *if*…else statement has following syntax:

```
if (test expression)
{
        True statement-block;
}
else
{
        False statement-block;
}
```

Here the test expression is any valid C language expression. When test expression is evaluated to true, the flow of control is transferred to True statement-block. In case when test expression is evaluated to false, the flow of control is transferred to False statement-block. In both the cases, after execution of if statement the flow control is transferred to the next statement in a sequence of the program. The said concept is explained in the Figure 13.6 using Flowchart of if...else statement.

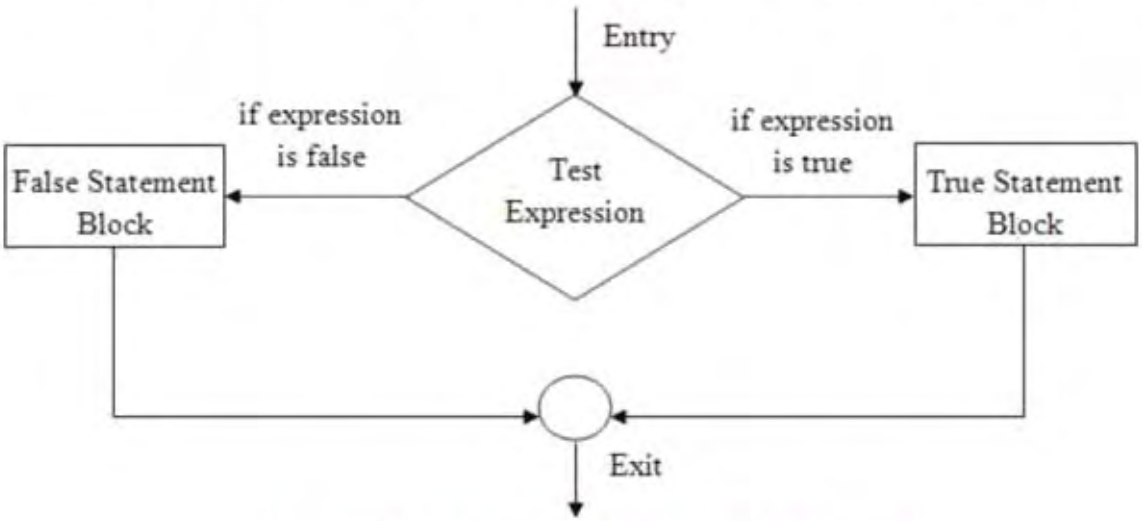


Figure 13.6 : Flowchart of if...else statement

Nested if...else statement

There are situations where we need to execute series of decisions in our program. We may use series of if..else statement in nested form to achieve our task. Consider the following case where one if..else statement block is used inside another if..else block.

```
if (test expression-1)
{
      if(test expression-2)
       {
            Statement block-1;
       }
       else
       {
            Statement block-2;
       }
}
else
{
      Statement block-3;
}
```

The logic of execution of nested if statement is as under :

- If test expression-1 is true, then test expression-2 is checked.
- If test expression-2 is true then Statement block-1 will be executed otherwise Statement block-2 will be executed.
- If test expression-1 is false, then Statement block-3 will be executed. Note that we may also use more if...else statement as a part of Statement block-3.

Note: The structure shown above is just for example. A user may change this structure as per his/her needs. For example the below structure is also valid.

```
if (test expression-1)
{
      Statement block-1;
}
else
{
     if(test expression-2)
       {
            Statement block-2;
       }
       else
       {
            Statement block-3;
       }
}
```

Example 13.3 : Let us try to understand a C program to read three numbers from users and display the largest of three numbers using nested if statement. Figure 13.7 gives the code listing of the example 13.3 while figure 13.8 shows its output.

10-3.c - SciTE

File Edit Search View Tools Options Language Buffers Help

10-3.c

```
/* Example 3: Program to demonstrate the use of nested if statement. */
#include<stdio.h>

int main()
{
    int number1, number2, number3;      //declares three variables
    printf("Enter three numbers:");     // message to the user
    scanf("%d %d %d", &number1, &number2, &number3); //read 3 numbers

    if ( number1 > number2 )            // test condition 1
    {
        if ( number1 > number3 )        // test condition 2
            printf("The largest number is %d \n", number1);
        else
            printf("The largest number is %d \n", number3);
    }
    else
    {
        if ( number3 > number2 )        // test condition 3
            printf("The largest number is %d \n", number3);
        else
            printf("The largest number is %d \n", number2);
    }
    return 0;
}
/* End of Program.*/
```

Figure 13.7 : Code listing of Example 13.3

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 10-3.c
kpp@ubuntu:~$ ./a.out
Enter three numbers:4 6 8
The largest number is 8
kpp@ubuntu:~$
```

Figure 13.8 : Output of Example 13.3

Explanation

The first statement inside main() will declare three variables. Second statement will display message to user. The third statement will allow users to enter three numbers which are stored in number1, number2 and number3 variables. Fourth statement (test condition 1) will check whether value stored in number1 is greater than value stored in number2 or not. If test condition1 evaluates to true the inner if-statement (test condition 2) will be executed to check whether number1 > number3 or not. Depending on true/false evaluation of test condition2, the inner if-statement message will be displayed to user. But when fourth statement (test condition 1) evaluates to false in that case the else part of outer if statement will be executed. Here the test condition 3 checks whether number3 > number2 or not, and depending on true/false evaluation the message will be displayed to the user.

The else-if ladder statement

There is another way of using many if statements together when we want to evaluate multiple decisions. When chain of if statements are used in else (false) block of nested if statements, it becomes else-if ladder statement.

Let us use the else-if ladder statement to give class/grade of student based on given subject marks. We will follow the grading rules given in table 13.1 during our program:

Range of Marks	Class/Grade
70-100	First Class with Distinction
60-69	First Class
50-59	Second Class
40-49	Pass Class
0-39	Fail

Table 13.1 : Sample Grading Rules

Example 13.4: Let us try to understand a C program to read marks from the user and assign class/grade to the student based on entered marks. Figure 13.9 gives the code listing of the example 13.4 while figure 13.10 shows two runs of its output.

10-4.c - SciTE

File Edit Search View Tools Options Language Buffers Help

10-4.c

```
/* Example 4: Program to illustrate use of else...if ladder statement. */
#include<stdio.h>
#include<stdlib.h>

int main()
{
    int marks;                              // declare a variable
    system("clear");                        //clears the screen
    printf("Enter your marks:");            // message to the user
    scanf("%d", &marks);                    // read a number from user

    if ( marks > 69 )                       // check condition 1
       printf("First Class with Distinction \n");
    else if ( marks > 59 )                  // check condition 2
            printf("First Class \n");
         else if ( marks > 49 )             // check condition 3
                 printf("Second Class \n");
              else if ( marks > 39 )        // check condition 4
                      printf("Pass Class \n");
                   else
                      printf("Fail \n");          // default else statement
     return 0;
}
/* End of Program */
```

Figure 13.9 : Code listing of Example 13.4



Figure 13.10 : Output of Example 13.4

Explanation

First, second, third and fourth statement after main() declares a variable, clears the screen, display message and read marks from the user respectively. Fifth statement checks the first condition of the program that is marks > 69. If condition 1 evaluates to true then message "First Class with Distinction" will be displayed and then program ends. If condition 1 evaluates to false then condition 2 will be evaluated. If condition 2 is false then condition 3 is checked and so on. When all four conditions are evaluated to false in that case last default else statement block will be executed displaying message "Fail".

Syntax of else-if ladder statement

```
if (test expression-1)
        Statement block-1;

else if (test expression-2)
            Statement block-2;

    else if (test expression-3)
                Statement block-3;
          .....
          .....
        else if (test expression-n)
      Statement block-n;
                else
                    Default-statement-block;
program-statement-x;
```

This construct is known as else-if ladder. The test expressions are evaluated from the top to bottom in the ladder. When any test expression in a ladder evaluated to true, the associated statement block is executed. After which control is transferred to the program-statement-x skipping rest of the ladder. When all the test expressions are evaluated to false then control is transferred to the Default-statement-block of final else. Figure 13.11 shows flowchart of else-if ladder statement.

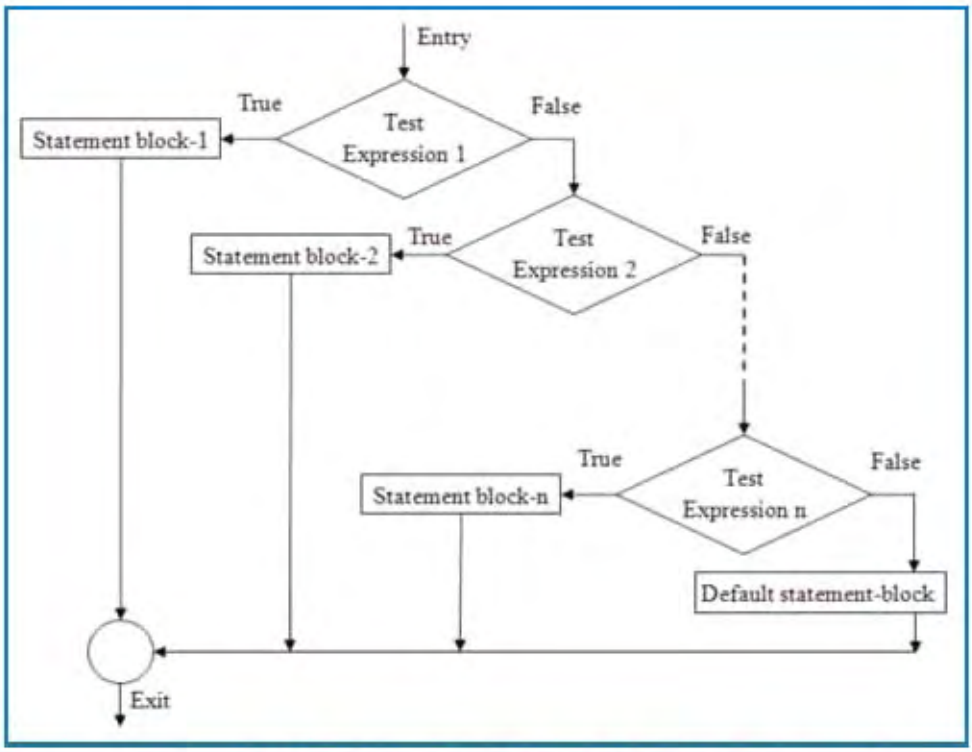


Figure 13.11 : Flowchart of else-if ladder statement

The Switch Statement

We have used the if...else statement for selection of various statement blocks during program execution. However when we use number of if...else statement then our program becomes more complex. The program becomes difficult to read and interpret the logic. To simplify our program, the C language has offered the built-in facility of multiway decision statement called switch. When an action is to be taken based on given multiple choices, switch statement is very useful.

Syntax of the Switch Statement

```
switch (expression)
{
        case value-1:
                statement block-1
                break;
        case value-2:
                statement block-2
                break;
        ....
        ....
        case value-n:
                statement block-n
                break;
        default:
                default statement block
                break;
}
statement-x;
```

Note the following important points related to switch statement :

- The switch statement uses one argument (expression or variable value) which is checked for equality with number of case options listed inside the switch statement block. The argument should be either a variable or expression of type integer or character.
- Each case option contains constant or constant expressions. This constant is known as case label and it ends with colon (:). Each case label must be unique. Any two case labels cannot have the same value.
- Statement block-1, statement block-2 and so on are statement lists which may contain zero or more statements. There is no need to use braces around this statement blocks.
- When switch statement is executed, it first evaluates the value of expression and then compares it with case constants from top to bottom. When a case found whose value is matching with the value of expression, then the corresponding statement block of that case is executed.
- After each statement block in switch, break statement signals the end of case causing the exit from switch statement. Then after flow of control is transferred to the next statement (statement-x) in a program.
- The default is an optional case. When *default* is written and no case found during execution at that time the default statement block is executed. The *default* can be used only once and may be placed anywhere inside switch but usually we place it at last inside the switch.

The execution of switch statement is illustrated in the flowchart shown in Figure 13.12. In the figure it is assumed that each case has break statement as the last statement inside block.

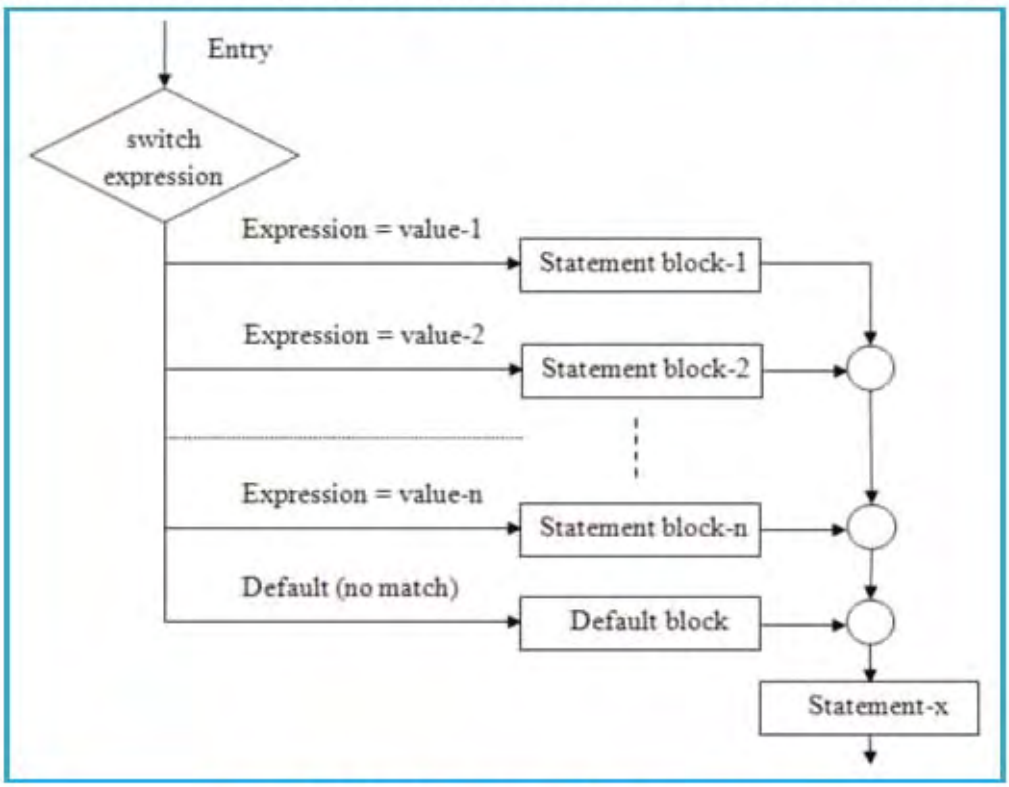


Figure 13.12 : Flowchart of switch statement

Example 13.5: Let us try to understand a C program to convert a given single digit (0 to 9) into its corresponding word representation. Figure 13.13 gives the code listing of the example 13.5 while figure 13.14 shows its output.

10-5.c - SciTE

File Edit Search View Tools Options Language Buffers Help

10-5.c

```c
/* Example 5: Program to illustrate use of switch statement.*/
#include<stdio.h>
int main()
{
    int number;
    printf("Enter a single digit: ");          // message to the user
    scanf("%d", &number);                      // read a number from user
    switch (number)
     {
        case 0:
                printf("0 = Zero \n");          break;
        case 1:
                printf("1 = One \n");           break;
        case 2:
                printf("2 = Two \n");           break;
        case 3:
                printf("3 = Three \n");         break;
        case 4:
                printf("4 = Four \n");          break;
        case 5:
                printf("5 = Five \n");          break;
        case 6:
                printf("6 = Six \n");           break;
        case 7:
                printf("7 = Seven \n");         break;
        case 8:
                printf("8 = Eight \n");         break;
        case 9:
                printf("9 = Nine \n");          break;
        default:
                printf("Out of range number entered. \n");
     }
     return 0;
} /* End of Program */
```

Figure 13.13 : Code listing of Example 13.5

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 10-5.c
kpp@ubuntu:~$ ./a.out
Enter a single digit: 5
5 = Five
kpp@ubuntu:~$ ./a.out
Enter a single digit: 105
Out of range number entered.
kpp@ubuntu:~$
```

Figure 13.14 : Output of Example 13.5

Explanation

First, second and third statement after main() declares a variable, display message and read a digit from user respectively. Fourth statement of switch uses argument as a value stored in variable named number. Based on value stored in number, the corresponding case block will be executed. If no case found its matching value then default block will display the message "Out of range number entered." and program ends.

Now Let us try to understand how to use character constant as a part of case label in the switch statement using Example 13.6.

Example 13.6 : Write a C program to read two numbers from user and ask the choice of arithmetic operator using menu. Depending on the choice given performs arithmetic operations on given numbers and display the result. Figure 13.15 gives the code listing of the example 13.6 while figure 13.16 shows its output.

10-6.c - SciTE

File Edit Search View Tools Options Language Buffers Help

```c
/* Example 6: Program to illustrate use of switch statement using character case label.*/
#include<stdio.h>
#include<stdlib.h>
int main() {
      char choice;
      int number1, number2;
      system("clear");
      printf("Enter two numbers:");                  // message to the user
      scanf("%d %d", &number1, &number2);    // read two numbers from user
      printf("\nMENU OF ARITHMETIC OPERATION \n");
      printf("A - Addition \n");
      printf("S - Subtraction \n");
      printf("M - Multiplication \n");
      printf("D - Division \n");
      printf("Enter your choice (A/S/M/D) : ");      // message to the user
      scanf(" %c", &choice);                         // read choice from user
      switch (choice)
      {
        case 'A':
            printf("Addition of entered numbers: %d \n", number1 + number2);
            break;
        case 'S':
            printf("Subtraction of entered number: %d \n", number1 - number2);
            break;
        case 'M':
            printf("Multiplication of entered number: %d \n", number1 * number2);
            break;
        case 'D':
            printf("Division of entered number: %d \n", number1 / number2);
            break;
         default:
            printf("Invalid menu choice. \n");
            break;
       }
      return 0;
} /* End of Program */
```

Figure 13.15 : Code listing of Example 13.6

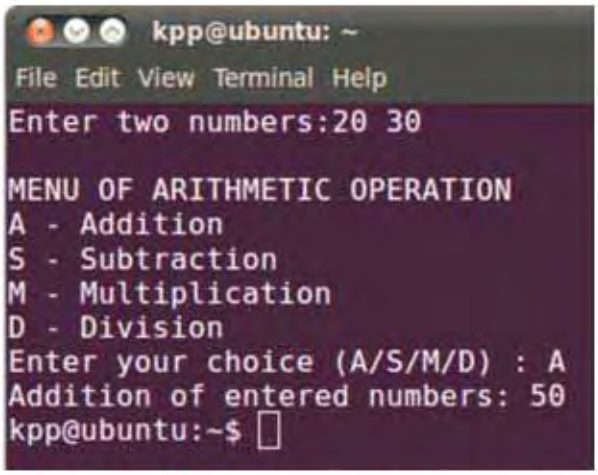


Figure 13.16 : Output of Example 13.6

Explanation

The fourth and fifth statement after main( ) will display message and reads two numbers from user respectively. Statement number six to ten will display menu options on screen. In eleventh and twelfth statement we display message and read menu choice from the user respectively. Based on the choice given by user, the corresponding case block will be executed. If no case found its matching value then default block will display the message "Invalid menu choice." and program ends. Note that in this program you may add additional statement at appropriate location to verify special arithmetic conditions like division by zero error.

Compound Relational Test

The C language provides the necessary mechanisms to perform compound relational tests (multiple test expression). The compound relational test is nothing but simple one or more relational tests joined together by either the logical AND operators or logical OR operators. These logical operators are represented by the character pairs && and || respectively. Use of compound relational test will help us to reduce number of if...else statements in our program.

Summary

In this chapter, we have learnt the special kind of C language facility that can change the flow of a sequence of instructions in the program. We had learnt that how two basic types of decision structure statements: if and switch can be used to change the flow of instructions in the program.

EXERCISE

1. What is the disadvantage of if statement as compared to if...else statement ?
2. What do you mean by nested if statement ?
3. Give suitable example in which the use of switch statement is advisable.
4. State the importance of break statement in the switch construct.
5. Explain the use of else-if ladder statement with suitable example.

6. State True or False :

(a) We can write if statement within switch statement.

(b) When test expression of if statement is evaluated to false, the statements inside body of if are skipped.

(c) The default case statement is mandatory in a switch statement.

(d) The default case must be the last case in the switch statement.

(e) The break statement stops execution of program.

(f) We can write switch statement within if statement.

7. Choose the correct option from the following :

(1) What is the value of variable flag after the execution of following if statement ?

```
int flag=0;
if(5 < 8) {  flag=1; }
```

(a) 0 (b) 1 (c) 5 (d) 8

(2) What is the value of variable 's' after the execution of following switch statement ?

```
x = 3;
switch ( x ) {
  case 1: s = 'A'; break;
  case 2: s = 'B'; break;
  case 3: s = 'C'; break;
  default: s = 'D'; break;
}
```

(a) A (b) B (c) C (d) D

(3) What is the value of variable sum after the execution of following statements ?

```
int number1=5, number2=10, sum=0;
if (number1 > number2)
        sum = sum + number1;
else
        sum = sum + number2;
```

(a) 0 (b) 5 (c) 10 (d) 12

(4) What will be the output when following program segment is executed ?

```
int number1=10, number2=20;
if ( ( number1+number2) > 35 || (number1 > number2) )
        printf("%d", number1);
else
        printf("%d", number2);
```

(a) 10 (b) 20 (c) 35 (d) Error

(5) What will be the output when following program segment is executed ?

```
char chr='A';
switch (chr)
{
        case 'A': printf("A"); break;
        case 'B': printf("B"); break;
        case 'C': printf("C"); break;
}
```

(a) A (b) B (c) C (d) Error

LABORATORY EXERCISE

Write a C program to perform following tasks :

1. Check whether given number is positive or negative.
2. Read age of person and display message whether person is eligible to vote or not.
3. Read three numbers from users and display the smallest of three numbers using nested if statement.
4. Check whether entered number is odd or even.
5. Check whether entered character is Vowel or not by using switch statement.



Loop Control Structures

In a C program there may be situations where we may want to execute same block of statements many times. All programming languages offers loop control structure (also known as **looping**) allowing programmers to execute a statement or group of statements multiple times. In this chapter we will discuss the loop (repetitive) control structure provided by the C programming language. Here 'control structure' mean that the flow of execution may not be sequential and control may be transferred to any statement based on given conditions in our program.

In looping, the sequences of statements are executed until some exit condition is satisfied. The looping construct is composed of two parts : *body of loop and control statement*. Depending on the place of control statement in loop, it can be classified as **entry controlled** and **exit controlled** loop. In entry controlled loop the exit/control condition is checked before executions of statements inside loop body. In exit controlled loop the exit/control condition is checked after executions of loop body. This means that in case of exit controlled loop, body will be executed at least once before exiting from the loop. Further discussion related to the same is given during explanation of different types of loop.

Now let us discuss syntax and general rules related to following three basic loop control structure provided by C language :

- for
- while
- do...while

The For Loop

The *for* loop is normally used when we want to execute block of statements fixed number of times. To make *for* loop more dynamic, we can use the exit condition inside for loop. Let us try to understand the use of simple *for* loop statement by example14. 1. This example displays five " * " using simple *for* loop statement. Figure 14.1 gives the code listing and output of example 14.1.

11-1.c - SciTE

File Edit Search View Tools Options Language Buffers Help

11-1.c

```
/* Example 1: Program to illustrate simple for loop statement.*/
#include<stdio.h>

int main()
{
    int count;                              // declaration of variables
    for ( count = 0; count < 5 ; count++)  // for loop body repeats 5 times
        printf(" * ");
    return 0;
}
/*End of Program */
```

```
gcc 11-1.c
>gcc 11-1.c
>Exit code: 0
./a.out
>./a.out
 *  *  *  *  * >Exit code: 0
```

Figure 14.1 : Code listing and output of Example 14.1

Explanation

In the beginning of program one integer variable count is declared. Variable count is used as a counter variable in a for loop. Firstly variable count is initialized to zero. Then second expression (count < 5) is checked. When it is evaluated to true, the body of for loop is executed. In our case one " * " will be displayed on screen using printf statement. Then third for loop expression (count++) will be executed, incrementing value of count by 1. Again second expression (count<5) is evaluated and depending on its result, body of for loop will be executed. In our case five times body part will be executed displaying five stars " * * * * * " on the screen.

Syntax of For Loop

```
for ( expression1; expression2; expression3)
{
        Statement-block;
}
```

The header of *for* loop contains three expressions separated by semicolons in parenthesis. All these expressions are optional. Statement-block which is also known as body of *for* loop may contain simple statement or compound statement. Following figure 14.2 shows the flowchart of *for* loop.

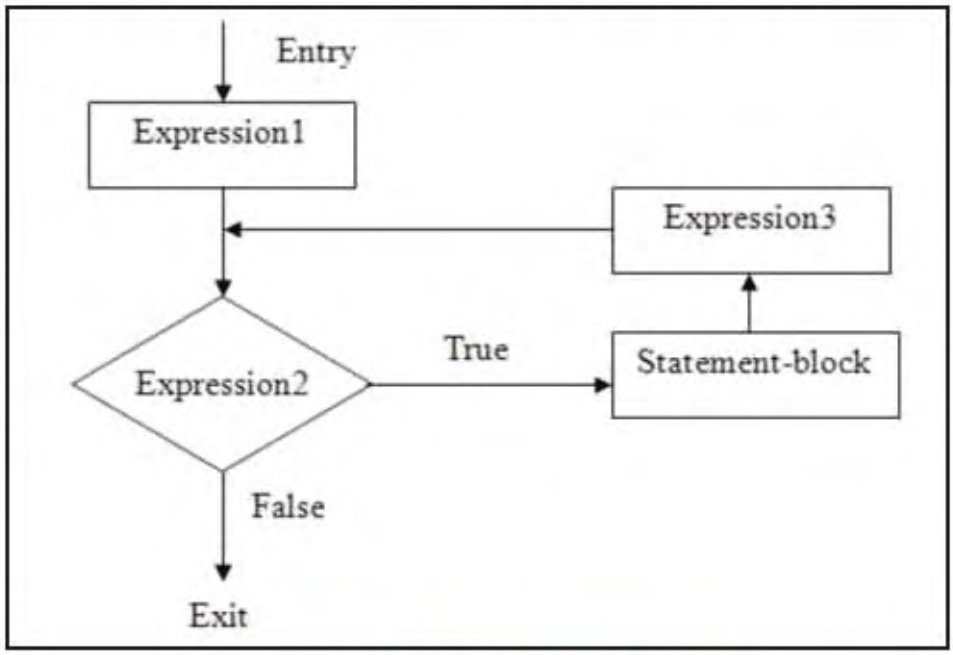


Figure 14.2 : Flowchart of for loop

As shown in figure14.2, the steps of executing for loop are as follows :

Step 1 : Evaluate expresison1. The expression1 will be evaluated only once when the for loop starts.

Step 2 : Evaluate expresison2. If its value is false then terminate the loop.

Step 3 : If expression2 is evaluated to true then execute the statements in the body of loop.

Step 4 : Evaluate expression3.

Step 5 : Go to Step 2.

Note following important points related to for loop :

- expression1 works for initializing value of counter variable that is used to control the number of times a loop is to be executed. This variable is known as control variable.
- expression2 works as test condition. A control variable is used for checking loop terminating criteria.
- expression3 works for incrementing or decrementing value of control variable.

Nested for loop

Nested for loop means using one for loop within another for loop. Let us try to understand this concept using example14.2. This example prints the following pattern for given number of lines. For example if number of lines are 4 then display output as follow:

```
1
1 2
1 2 3
1 2 3 4
```

Figure 14.3 gives the code listing of the example 14.2 while figure 14.4 shows its output.

11-2.c - SciTE

File Edit Search View Tools Options Language Buffers Help

11-2.c

```
/* Example 2: Program to illustrate the use nested for loop.*/
#include<stdio.h>

int main()
{
    int i, j, lines;                        // declaration of variables
    printf("Enter number of lines : ");     // message to user
    scanf("%d", &lines);                    // stores value in lines variable

    for ( i = 1; i <= lines ; i++)          // Outer loop, repeat number of lines time
    {
        for( j = 1 ; j <= i ; j++)    // inner loop for printing numbers from 1 to value of i
        {
                printf("%d ", j);           // print value of j
         }
        printf("\n");                       // move cursor to next line
    }
    return 0;
}
/* End of Program */
```

Figure 14.3 : Code listing of Example 14.2

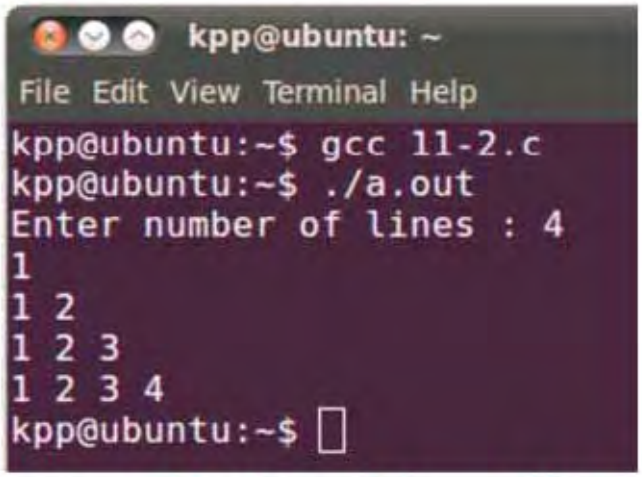


Figure 14.4 : Output of Example 14.2

Explanation

The program contains outer for loop with control variable i, which is executed 'lines' number of times by varying value of i 1 by 1 up to the value of lines. For each i in the outer loop, inner loop with control variable j is executed i times, and display numbers varying 1 by 1 up to i. After printing i numbers in line i, printf("\n") statement of outer loop will bring cursor on the next line. This enables the next line number to be printed in new line.

Use of Comma Operator in for Loop

In for loop we can initialize more than one parameters using comma operator. Similarly we may use comma operator to increment or decrement various variables in a for loop. Let us try to understand the use of comma operator using example14.3. Figure 14.5 gives the code listing of the example 14.3 while figure 14.6 shows its output.

11-3.c - SciTE

File Edit Search View Tools Options Language Buffers Help

11-3.c

```
/* Example 3: Program to illustrate the use comma operator in a for loop
   to print the addition table.*/

#include<stdio.h>
int main()
{
    int i, j, number;                                  // declaration of variables
    printf("Enter number of lines in a table: "); // message to user
    scanf("%d", &number);                              // stores value in number variable
    for ( i=0, j=10; i < number; i++, j- -)
    {
                printf("%d  +  %d =  %d\n", i, j, i + j );
    }
    return 0;
}
/* End of Program */
```

Figure 14.5 : Code listing of Example 14.3

```
kpp@ubuntu: ~
File Edit View Terminal Help
kpp@ubuntu:~$ gcc 11-3.c
kpp@ubuntu:~$ ./a.out
Enter number of lines in a table: 5
0  +  10 =  10
1  +  9 =  10
2  +  8 =  10
3  +  7 =  10
4  +  6 =  10
kpp@ubuntu:~$
```

Figure 14.6 : Output of Example 14.3

Note that in example 14.3 we have initialized two variables i=0 and j=10. Also we are incrementing i with value one and decrementing j with value 1 in the same *for* loop. Based on the input given we will get output as shown in the example.

The While Loop

When number of iteration cannot be pre-determined and when loop terminating condition is to be tested before entering the loop, at that time use of while loop is more suitable. For example,

- Find sum of all numbers entered until user enters zero
- Display menu options and take proper action until user select exit option

Syntax of While Loop

```
while (test expression)
{
        statement-block;    /* body of while loop */
}
```

Flow chart of execution of while loop is shown in figure14.7.

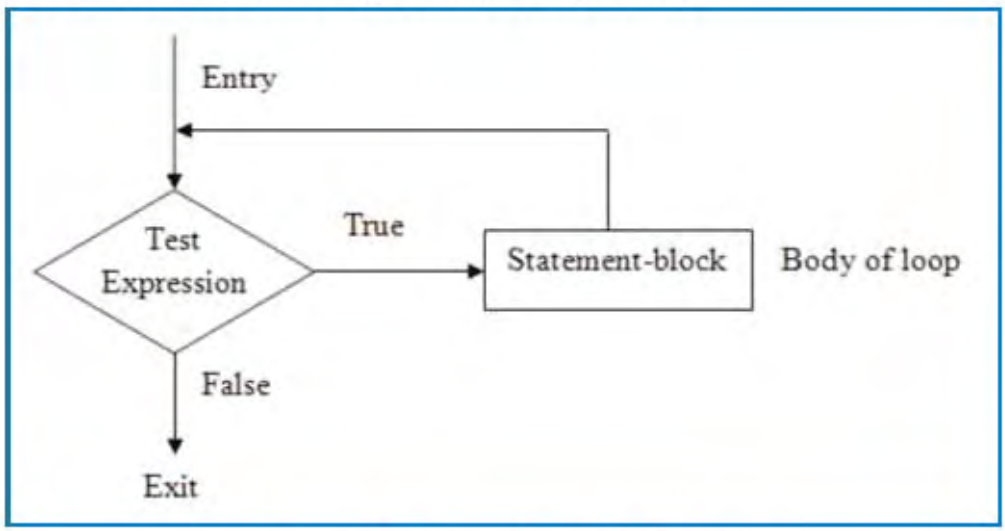


Figure14.7 : Flowchart of while loop

As seen in the figure14.7, it evaluates the test expression first. If test expression is evaluated to true, then body of loop containing statement-block is executed. Again program evaluates the test expression. This process is repeated until the test expression is evaluated to false. When test expression is evaluated to false then loop is terminated and control is passed to the next statement after the loop in a program. Body of loop may contain simple or compound statement.

Note that as we are checking condition at the entry point, it is known as *entry-controlled* loop. If test expression is evaluated to false, the body part will not be executed at all.

Let us try to understand the use of *while* loop using example 14.4. The program given in example14.4 finds sum of all numbers entered until user enter zero. Figure 14.8 gives the code listing of the example 14.4 while figure 14.9 shows its output.

11-4.c - SciTE

File Edit Search View Tools Options Language Buffers Help

11-4.c

```
/* Example 4: Program to illustrate use of while loop.*/
#include<stdio.h>

int main()
{
   int number, sum=0;                          // declaration of variables
   printf("Enter 0 to exit from program.\n");
   printf("Enter your number : ");             // message to user
   scanf("%d", &number);                       // stores value in number variable
   sum = number;

   while(number != 0)
   {
      printf("Enter your number : ");          // message to user
      scanf("%d", &number);                    // stores value in number variable
      sum = sum + number;
    }
   printf("The sum of entered numbers : %d \n", sum);
   return 0;
}
/*End of Program */
```

Figure 14.8 : Code listing of Example 14.4

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 11-4.c
kpp@ubuntu:~$ ./a.out
Enter 0 to exit from program.
Enter your number : 5
Enter your number : 10
Enter your number : 0
The sum of entered numbers : 15
kpp@ubuntu:~$
```

Figure 14.9 : Output of Example 14.4

Explanation

First statement after main function declares two variables required in the program. The second statement informs user about how to exit from the program. Third and fourth statement displays message and read a number from the user respectively. Fifth statement assigns the value of number variable to sum variable. In sixth statement, we check the test condition i.e. value stored in number variable is not equal to zero. When this test condition evaluates to true, statement number seven and eight displays message and read a number from the user respectively. The tenth number of statement will be adding the entered number into the sum variable. After this again test condition of *while* loop will be evaluated. When test condition is evaluated to false, the program executes the tenth statement displaying the sum of all the numbers entered by the user.

The do-while loop

We have seen that in *while* loop, the test expression is checked before executing body of loop. There are certain cases where we may want to execute body of loop before test expression. The do...*while* loop should be used when the test expression is to be checked after executing body of loop. As we are checking test condition at the end of loop, the do...*while* loop is a kind of *exit-controlled* loop. Note that in *do...while* loop, body of loop will be executed at least once.

Syntax of do...while loop

```
do
{
        statement-block;        /* body of loop */
}
while( test expression);
```

Flow chart of *do*...while loop is shown in figure14.10.

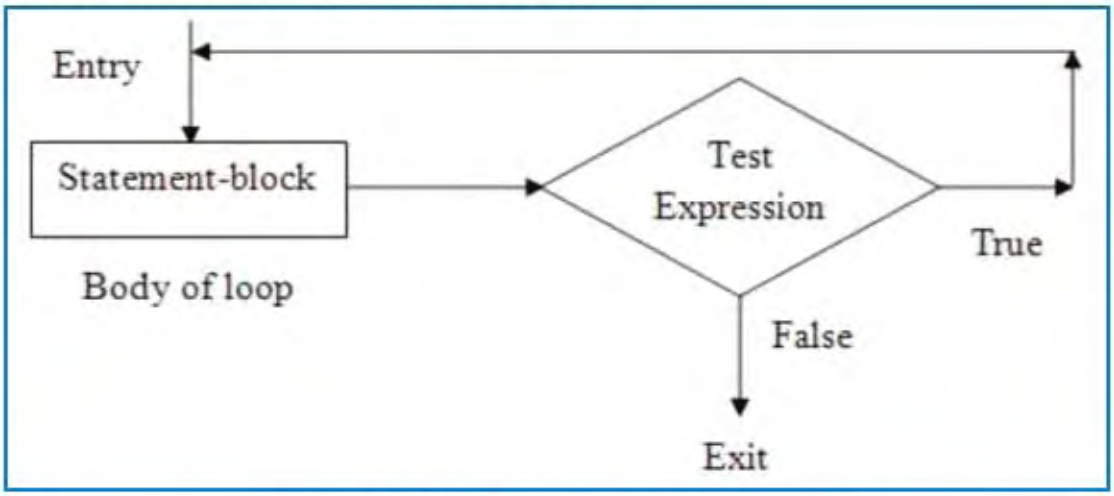


Figure 14.10 : Flowchart of do...while loop

As seen in the figure14.10 the body of loop containing statement-block is executed first. Then it evaluates the test expression. If the test expression is evaluated to true then body of loop containing statement-block is executed again. After executing statement-block, once again text expression is evaluated. This process is repeated until the test expression is evaluated to false. When test expression is evaluated to false then loop is terminated and control is passed to the next statement after the loop in a program. Body of loop may contain simple or compound statement.

Let us try to understand the use of do..*while* loop using example 14.5. Program given in the example14.5 will find sum of all numbers entered by the user, until user enters zero using do…*while* loop. Figure 14.11 gives the code listing of the example 14.5 while figure 14.12 shows its output.

11-5.c - SciTE

File Edit Search View Tools Options Language Buffers Help

11-5.c

```
/* Example 5: Program to illustrate use of do...while loop.*/
#include<stdio.h>

int main()
{
    int number, sum=0;                    // declaration of variables
    printf("Enter 0 to exit from program.\n");
    do
    {
        printf("Enter your number : ");   // message to user
        scanf("%d", &number);             // stores value in number variable
        sum = sum + number;
     } while(number != 0);

    printf("The sum of entered numbers : %d \n", sum);
    return 0;
}
/*End of Program */
```

Figure 14.11 : Code listing of Example 14.5

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 11-5.c
kpp@ubuntu:~$ ./a.out
Enter 0 to exit from program.
Enter your number : 3
Enter your number : 4
Enter your number : 0
The sum of entered numbers : 7
kpp@ubuntu:~$
```

Figure 14.12 : Output of Example 14.5

Explanation

First statement after main function declares two variables required in the program and initializes one of them. Second statement displays the message about how to exit from the program. At statement three the *do...while* loop starts. Statement fourth and fifth displays message and read a number from user respectively. Statement sixth adds number entered by user into sum variable. Statement seven evaluates the test condition. Statement fourth to sixth is executed repeatedly every time when test expression is evaluated to true. When test expression is evaluated to false, the program executes the eighth statement displaying the sum of all the numbers entered by the user.

Nested Loops

Nesting of different types of loops is possible by using one loop control structure within another, irrespective of the type of loop control structures used. For example, in *for* loop, *while* or *do...while* loop can be used. For example, display all prime numbers between 1 and 100. Loop for numbers can be written using *for* loop. Within this loop, while loop can be used to determine whether number is prime or not.

Selecting a Loop

So far in this chapter, we have discussed three looping construct for, while and do..while of C programming language. To select the more appropriate loop for our program, we may use the following steps :

- For a given problem, identify whether entry-controlled or exit-controlled loop is required.
- For entry controlled loop, we may use *for* or *while* loop construct. Use of for loop is advisable for counter based exit condition.
- For exit controlled loop, we have choice of *do..while* construct.

Also while using any loops in our program following three important points are to be considered carefully to avoid miscellaneous behavior of a loop:

- Initialization of loop counter
- To test exit condition to come out from the loop
- Incrementing or decrementing loop counter

Skipping a Part of Loop

During execution of a program, sometimes we may want to skip some of the statements or iterations of the loop. This is possible in C language by using following statements :

(i) break statement

(ii) continue statement

The break Statement

Sometimes during execution of any loop, we may want to terminate loop instantly without even execution of other statements of loop. This is possible using the break statement. The break statement in C programming language has following usages :

- When the break statement is encountered inside the loop, the loop is immediately terminated and program control executes the next statement following the loop.
- The break statement can also be used to terminate a case in the *switch* statement.

In case of nested loop, the *break* statement will stop the execution of currently executing loop and starts execution of next statement after the block of code.

Example14.6 shows how the *break* statement works in a program. Figure 14.13 gives the code listing as well as output of example 14.6.

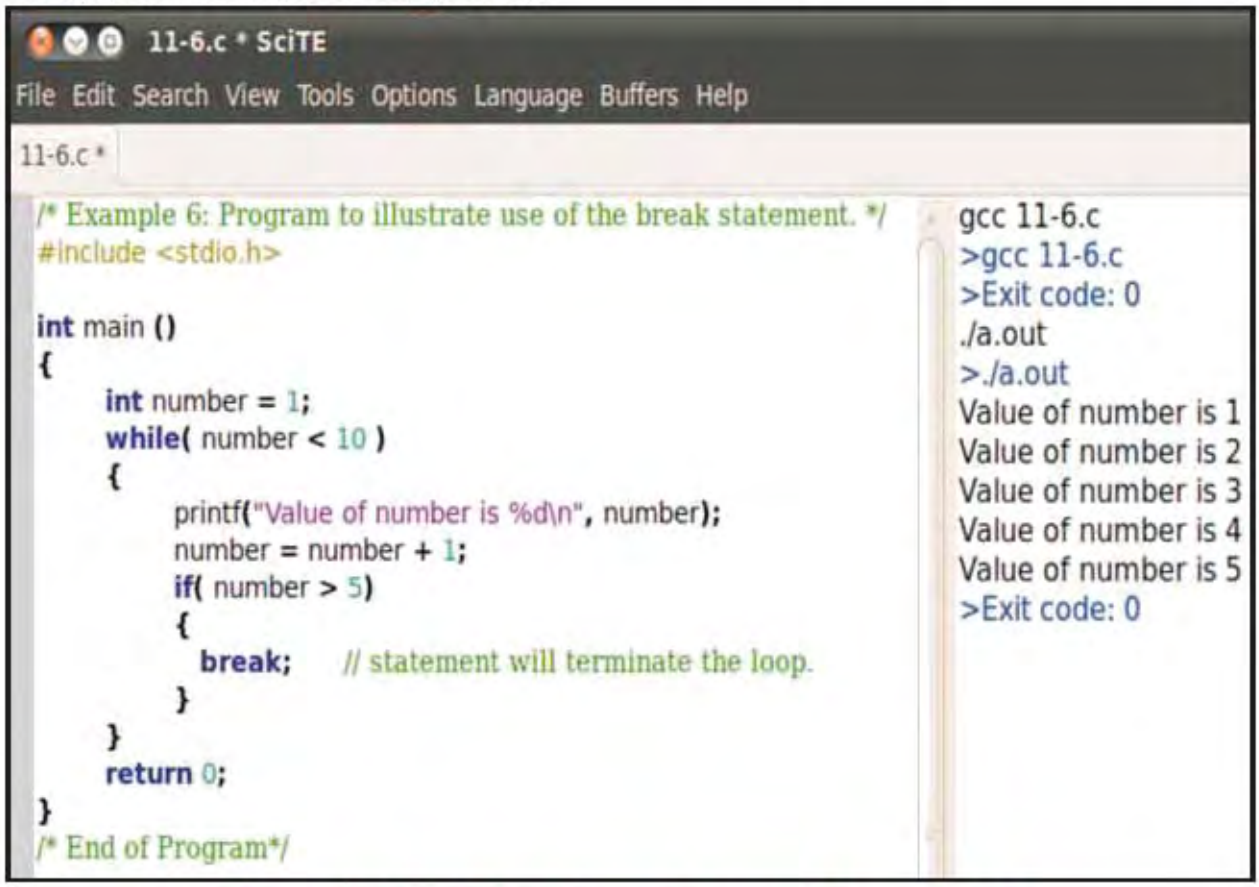


Figure 14.13 : Code listing and output of Example 14.6

Explanation

First statement after main function initializes an integer variable. The test condition in *while* statement is written to execute loop nine times. The printf statement inside loop will print the value of number variable. After printing value of number variable it is incremented by 1. Next, *if* statement will be executed. But during execution of program when *if* statements is evaluated to true, the *break* statement executes; causing termination of while loop. Hence in our program, printf statement inside the while loop will be executed only five times.

The continue Statement

The continue statement works to some extent like the break statement. Instead of forcing termination of loop, however, continue statement forces the next iteration of the loop to take place, skipping any code in between.

For example, in for loop, continue statement causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, continue statement causes the program control passes to the conditional tests.

Let us see example14.7 which shows how continue statement works in a program. Figure 14.14 gives the code listing and output of example 14.7.

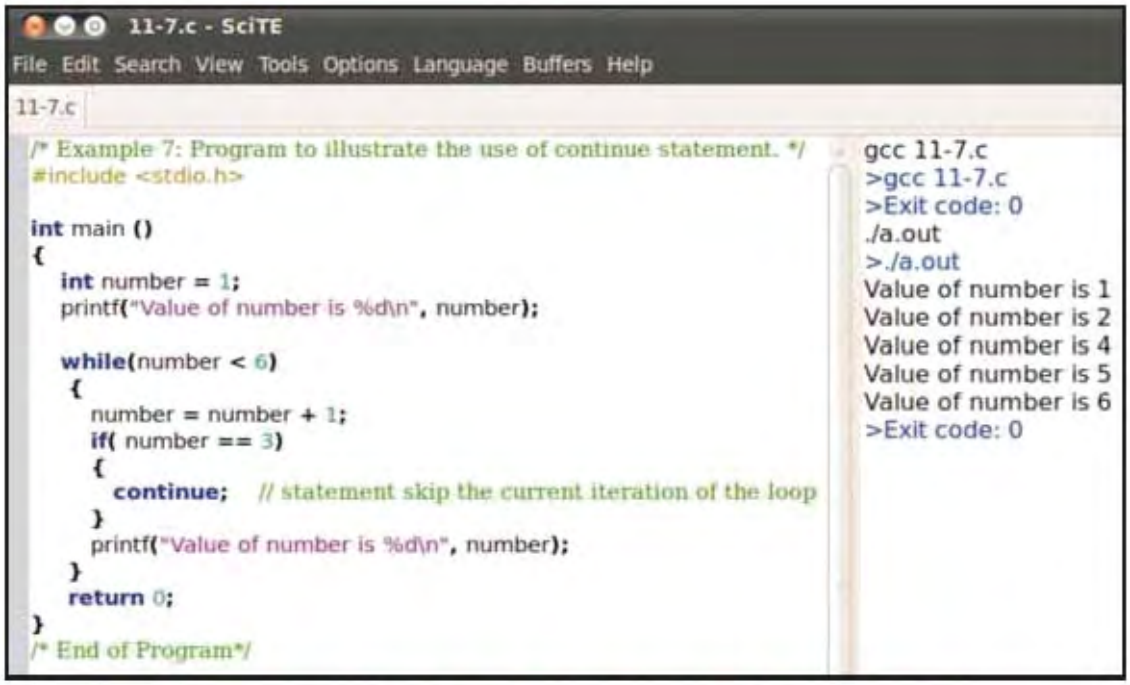


Figure 14.14 : Code listing and output of Example 14.7

Explanation

First statement declares and initializes a number variable required in the program. Second statement prints value of number variable. The test condition in while statement is written to execute loop five times. Inside while loop, the number is incremented by value 1. After this the if statement checks the test condition. When it is evaluated to true, the continue statement executes, which terminates the current iteration of while loop. Hence value of number 3 will not be printed. Program executes next iteration of loop causing program to print value of number variable 4, 5 and 6.

The Infinite Loop

Any loop in a program becomes infinite loop if it runs forever and program control never comes out of it. The loop becomes infinite due to non availability of exit condition in the logic of loop. Observe the program code logic of example 14.8 shown in figure 14.15.

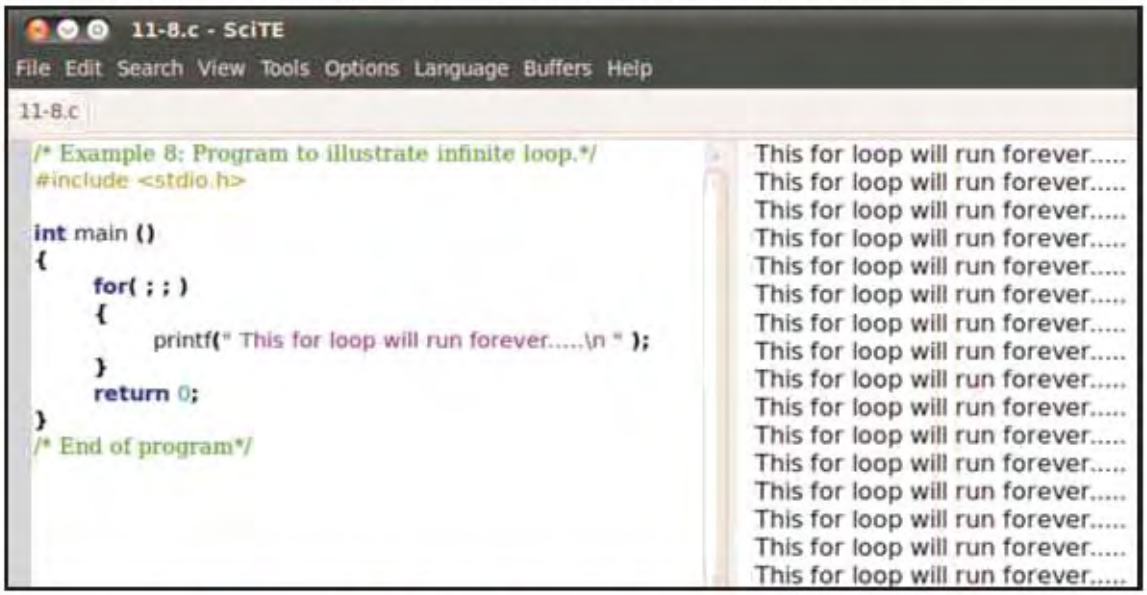


Figure 14.15 : Code listing and output of Example 14.8

Explanation

When we compile and execute this code it will runs forever. In C program *for* loop all three expressions are optional. When the conditional expression is absent inside *for* loop, it is assumed to be true. We may have initialization expression and increment expression inside for loop, but C program runs forever when we use for(; ;) construct. Note that you may use CTRL + C keys to terminate your infinite loop.

Summary

In this chapter we had learnt that how C languages offers **loop control structure** allowing programmers to execute a statement or group of statements multiple times. We had learnt three basic loop control structure (for, while and do...while) provided by the C language. We had discussed that how entry controlled and exit controlled loops are useful in our programs. We had also learnt that how nesting of various loops can be performed. At the end of chapter we had learnt that how during execution of a program; we can skip some of the statements or iterations of the loop using break and continue statements.

EXERCISE

1. Explain the syntax of for loop construct with suitable examples.
2. What do you mean by looping statement ? State use of looping statement in a C program.
3. What is the difference between while and do-while loop ?
4. Describe how to skip part or a loop using break and continue statement.
5. What do you mean by entry controlled and exit controlled loop ? Give suitable example for the same.
6. Fill in blanks :
   (a) Depending on the place of control statement in loop, it can be classified as _____ controlled and _____ controlled loop.
   (b) The for loop is example of _____ controlled loop.
   (c) The do-while loop is example of _____ controlled loop.
   (d) In exit controlled loop, body will be executed _____ time before exiting from the loop.
   (e) The _____ statement causes immediate exit from the loop control structure.
   (f) The _____ statement skips the subsequent statements of the loop and continues the next iteration of the loop.
7. State True or False :
   (a) Within for loop do...while loop cannot be used.
   (b) Execution of break statement inside the loop immediately terminates the execution of program.

(c) Initialization, test condition and increment parts are optional in a for loop header.

(d) The break statement cannot be used to terminate a case in the switch statement.

(e) Initialization, test condition and increment parts are separated by a commas.

(f) For every while clause, there must be a do.

(g) For every do clause, there must be a corresponding while.

8. Choose the correct option from the following :

(1) How many times printf statement is executed in the following program segment ?

```
int num1 = 3, num2 = 6;
while(num1 < num2) {
    printf(" Hello Students...");
    num1 = num1 + 1;
}
```

(a) 1 (b) 2 (c) 3 (d) 6

(2) What is the output of following program segment ?

```
int a = 5, b = 10;
do
{
    a = a + 1;
}while( a < = b);
printf(" %d", a);
```

(a) 10 (b) 9 (c) 11 (d) 15

(3) What is the output of following program segment ?

```
int main( ){
int i;
for(i = 0; i < 10; i++);
printf("%d", i);
}
```

(a) 0123456789 (b) 9 (c) 10 (d) Error

(4) What is the value of a number variable after execution of following code ?

```
int number = 1;
while(number < 5){
    number = number + 1;
    if( number = = 3) {
       continue;
    }
}
```

(a) 1 (b) 3 (c) 5 (d) None of these

LABORATORY EXERCISE

1. Change the following for loop into its equivalent while loop

(a)
```
for(number=0; number <5; number++) {
printf("%d", number);
}
```

(b)
```
for(number=5; number > 0; number- -) {
    printf("%d", number);
}
```

2. Write a program to demonstrate use of the compound statement within a while loop.
3. Write a program to find the sum of first 100 integers using for loop.
4. Demonstrate the use of while loop to find the sum of first 100 even integers.
5. Write a program to print following pattern for a given number of lines. For example, if number of lines entered are 4 then out is as under:

```
*
* *
* * *
* * * *
* * *
* *
*
```

6. Write a for loop program for printing following series of numbers on screen.

(a) 1, 3, 5, 7, 9, 11 (b) 1, 2, 4, 8, 16, 32, 64

(c) 10, 8, 6, 4, 2, 0 (d) -10, -8, -6, -4, -2, 0

7. Write a program to display ASCII value of uppercase and lowercase letters.
8. Write a program to count and display the number of characters entered by user until user enters 'Z';



Arrays

In C programming, so far we have used only basic data types, namely int, float, char, double, etc. These types of variables can store only one value at a time. To process one value we need one variable in a program. If we want to process five different values, we need five different variables. Now imagine the program where we want to handle very large amount of data. To handle such kind of situations, C supports a special kind of data type called an array. In this chapter we will learn how to use different types of an array available in C language.

Need of Arrays

Imagine some of the following real life situations for which we want to write a C program :

- Maintain Marks of all the students of class
- Maintain Cost of all the products of Super Market
- List of employees and their contact numbers
- Maintain height and weight of each student of class

To handle marks of each student, instead of declaring different variables, such as marks0, marks1, marks2, marks3, marks4, ... and marks59, we may declare one array variable such as marks[60]. To refer individual student's mark now we can use variable like marks[0], marks[1], marks[2] ... and so on. In computer memory array of marks[59] will look like the one shown in figure 15.1.

First element	Second element	Third element	....	....	Last element
marks[0]	marks[1]	marks[2]	marks[3]	.....	marks[59]

Figure 15.1 : Sample Structure of Array in Memory

Note following important points related to an arrays.

- It is a collection of elements having same data type.
- It is a fixed-size sequential collection of elements. An array element occupies contiguous memory locations.
- The element of an array is accessed by an index number. The index number contained within square brackets is also known as subscript.
- A subscript number must be an integer or an integer expression.
- The subscript number starts with zero.

Types of Arrays

The arrays in C language can be classified in to following two categories :

(i) Single or One dimensional array

(ii) Multidimensional array

The single dimensional array may have one row or one column while multidimensional array has one or more rows and one or more columns. The number of rows and columns are specified by user as per the program requirement. We will discuss only one and two dimensional arrays offered by the C language.

Declaration of Single Dimensional Array

To declare an array in C language, we specify the type of data which we want to store in the elements and the number of elements to be stored in an array as follows:

```
datatype arrayname [ size ];
```

- ***datatype:*** It specify the type of elements that an array stores. If we want to store characters in an array then type of an array is 'char'. In the same way to store integers, type of an array is 'int'. The datatype can be any valid C language data type.
- ***arrayname:*** This is the name given to an array variable by the programmer. It can be any string but it is usually suggested that some meaningful name should be given to an array. The array name should be in context with what is being stored in that array in a program.
- ***size:*** The value of size indicates the number of elements the array can stores. The size must be an integer constant greater than zero.

For example, to store marks of 60 students, we may use following statement :

```
int marks[60];
```

Note that the marks array declared here will be able to contain only 60 integer values. The example of arrays that can store characters, floating point values and very large values (double) is shown in the declarations below:

```
char strings[20];

float percentages[20];

double numbers[20];
```

Assigning Values to Single Dimensional Arrays

C language allows array initialization in two different ways:

(i) Compile time array initialization

(ii) Runtime array initialization

Compile Time Array Initialization

An array can be assigned values similar to a normal variable at the time of declaration, the general syntax to set the values of various array elements is as follows:

```
datatype arrayname[size] = { value1, value2, .... , valueN };
```

Here *value1, value2, ... , valueN* provide the initial values for successive elements of the array. Following is the example where we specify size and set values of all elements of arrays.

```
int marks[5] = {60,65,70,75,80};
```

This will declare an array variable marks that can store five elements with the values set as: marks[0]=60, marks[1]=65, marks[2]=70, marks[3]=75 and marks[4]=80. During such declaration, if list of values are less than the size of an array, then only specified number of elements will be initialized and remaining elements will be initialized to zero. For example, the declaration

```
int marks[5] =  {60,65,70};
```

will initialize first three array elements as marks[0] = 60, marks[1] = 65 and marks[2] = 70. The remaining two array elements marks[3] and marks[4] will be initialized to value zero.

If the array elements are assigned value at the time of declaration itself then specifying size is optional. If we don't specify the size, compiler will automatically calculate the size by counting total number of values written inside the curly bracket { }. For example, the following statement

```
int marks[ ] = {60,65,70,75,80};
```

will declare the marks array containing five elements whose values are initialized with the values given in the curly bracket.

Let us learn the concept of compile time initialization of one dimensional array with example15.1. Figure 15.2 gives the code listing and output of the example 15.1.

12-1.c - SciTE

File Edit Search View Tools Options Language Buffers Help

12-1.c

```
/* Example 1: Program to illustrate compile time initialization of
   one dimensional array.*/

#include<stdio.h>
int main()
{
    int number[5] = {10,20,30,40,50}; // Compile time initialization
    int i;
    for( i = 0; i < 5 ; i++)          // Loop to display array elements
    {
        printf(" %d  \n", number[i] );
    }
   return 0;
}
/* End of Program*/
```

```
gcc 12-1.c
>gcc 12-1.c
>Exit code: 0
./a.out
>./a.out
 10
 20
 30
 40
 50
>Exit code: 0
```

Figure 15.2 : Code listing and output of Example 15.1

Explanation

The first line after main function declares an array called number and initializes all array elements with list of values given in curly bracket at compile time. The for loop will display the individual elements of a number array.

Runtime Array Initialization

When we need to read data from user while executing our program, we can use runtime array initialization. The code given in example15.2 will allows user to enter marks of five different students at runtime of the program. The program also demonstrates how to use stored values from the array.

Figure 15.3 gives the code listing of the example 15.2 while figure 15.4 shows its output.

12-2.c * SciTE

File Edit Search View Tools Options Language Buffers Help

12-2.c *

```
/* Example 2: Program to illustrate runtime array initialization and accessing values
   from one dimensional array. Program will read marks of five students and display the
   average marks.*/

#include<stdio.h>
int main()
{
      int marks[5], i, total=0;
      for( i = 0; i < 5; i++)                    // loop to read marks of five students
      {
            printf("Enter marks[%d] :", i);
            scanf("%d", &marks[i]);
      }
      for( i = 0; i < 5; i++)                    // loop to find total marks
      {
            total = total + marks[i];
      }
      printf("\nThe average marks of students are %d. \n", total/5);
      return 0;
}
/* End of Program*/
```

Figure 15.3 : Code listing of Example 15.2

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 12-2.c
kpp@ubuntu:~$ ./a.out
Enter marks[0] :5
Enter marks[1] :10
Enter marks[2] :15
Enter marks[3] :20
Enter marks[4] :25

The average marks of students are 15.
kpp@ubuntu:~$
```

Figure 15.4 : Output of Example 15.2

Explanation

The first *for* loop in the program will read marks of five students. The second *for* loop will find the total marks by reading individual elements of marks array. The last printf statement will display the average marks of the students.

String as a Character Arrays

A string is a series of characters that is normally treated as a single block. C language does not

provide the *string* as inbuilt data type. However it allows us to represent string as a character array. A string variable is any valid C variable name and it is always declared as the character array. The general form of string declaration is:

```
char stringname[ size ];
```

Here the size indicates the number of characters in the variable called stringname. Followings are some of the valid string declarations:

```
char student_name[20];
char city[50];
char state[20];
```

Note that strings in a C language are terminated by the special character called null character ('\0'); this helps the program to identify the end of the string. Since strings are terminated by the null character '\0', we require one extra storage location in the character array to accommodate all the characters of strings.

Similar to numeric arrays, character arrays can also be initialized at compile time and at run time of a program. Character array initialization can be possible in following different manners:

```
char student_name[6] = "PURVA";
char student_name[6] = {'P', 'U', 'R', 'V', 'A', '\0'};
```

The reason behind keeping 6 elements in student_name array is that the string PURVA contains five characters and one element is used for null character '\0'. Note that when we initialize the character array by listing its elements, it is necessary to specify null character '\0' at the end of string. We can also declare character arrays without specifying size as follow:

```
char state[ ] = {'G', 'U', ' J', 'A', 'R', 'A', 'T', '\0' };
```

In this case the character array state will be declared and initialized with eight elements. Using following statements we can refer previously discussed character arrays.

```
printf("The name of student is %s", student_name);
printf("The name of state is %s", state);
printf("The first character of your name is %c", student_name[0]);
```

Multidimensional Array

The single dimensional array keeps track of information in linear order. However, the data associated with certain real life systems (such as digital image, a chess game board, matrix, etc.) are available in two dimensions. To think about this type of system in a program, we need a multi-dimensional data structure.

C language allows us to create multidimensional array. As mentioned earlier, multidimensional arrays have more than one rows and columns. We can create two dimensional, three dimensional, or N-dimensional array in C. The simplest form of the multidimensional array is the two-dimensional array. The two dimensional array have row index as well as column index. We will discuss only the two dimensional arrays in this chapter.

Declaration of Two Dimensional Arrays

So far we had discussed one dimensional array that can store a list of values in it. There are many situations in real life where we may want to store data tables. Consider the case given in the table 15.1. It shows the details about sale of an item by different salesmen from Monday to Friday.

	Monday	Tuesday	Wednesday	Thursday	Friday
Salesman1	100	150	200	250	200
Salesman2	200	250	300	350	300
Salesman3	150	200	250	300	250

Table 15.1 : Sales details

The table contains a total 15 different values. Here use of one dimensional array is not advisable. Better option is to use two dimensional arrays. The table 15.1 can easily be represented by using two dimensional arrays of C. The two dimensional arrays are declared as follows:

datatype arrayname [row size][column size];

The *row size* and *column size* must be an integer constant greater than zero and datatype can be any valid C language data type. Note that in C language, the multidimensional arrays are row major. In other words, the first bracket in the declaration of an array specifies number of rows.

To store the data shown in table 15.1 we can declare a two dimensional array as under:

```
int sales[3][5];
```

Here row size=3 which is used to refer salesman (Salesman1 = 0, Salesman2 = 1, and Salesman3 = 2) while column size=5 which is used to refer days. (Monday = 0, Tuesday = 1, Wednesday = 2, Thursday = 3 and Friday = 4) We have assumed that unit sales values are in integer only. Figure 15.5 shows the sample memory layout of the Sales details.

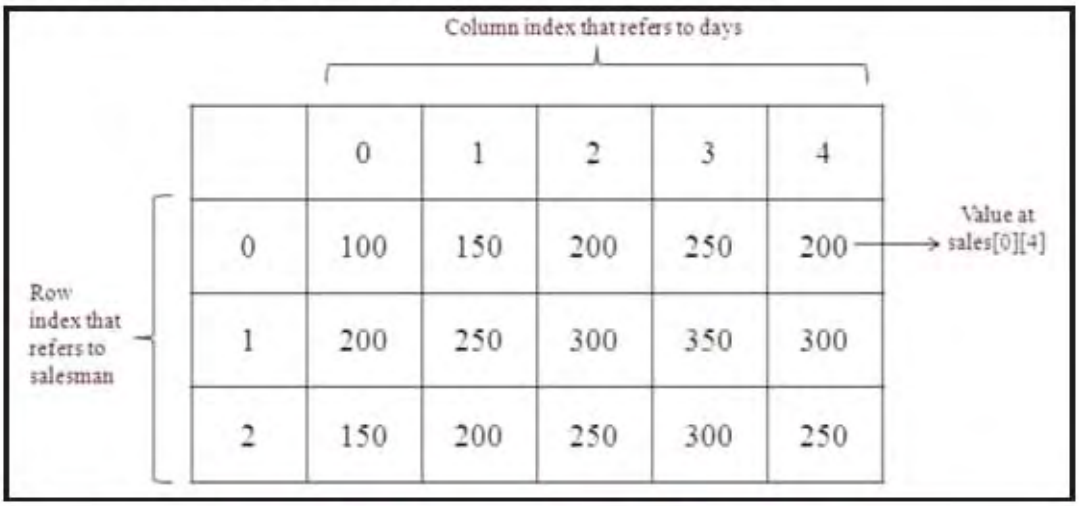


Figure 15.5 : Memory layout of Sales details

Now to check the sales of item done by Salesman1 on Monday we may use array element sales[0][0]. The value stored at sales[0][0] is 100. Similarly to check the sales of item done by Salesman2 and Salesman3 on Monday, we may use array element sales[1][0] and sales[2][0] respectively. As shown in figure 15.5 the value stored at sales[0][4] is 200 and it refers to the sales of item done by Salesman1 on Friday.

Initializing Two Dimensional Arrays

Like single dimensional array, we can also initialize two dimensional arrays at compile time and runtime of a program.

Compile Time Initialization of Two Dimensional Arrays

The compile time array initialization of table 15.1 can be performed in a program as under:

```
int sales[3][5]={
                {100, 150, 200, 250, 200},
                {200, 250, 300, 350, 300},
                {150, 200, 250, 300, 250}
            };
```

The nested inner curly braces, which indicate the intended salesman row, are optional. The following initialization is equivalent to previous example:

```
int sales[3][5]={100, 150, 200, 250, 200, 200, 250, 300, 350, 300, 150, 200, 250, 300, 250};
```

Run Time Initialization of Two Dimensional Arrays

The run time array initialization of data shown in table 15.1 can be performed using program segment given in code listing 15.1.

```
int sales[3][5], row, column;

for( row = 0; row < 3; row++) {
        for (column = 0; column < 5; column++){
            printf("Enter sales item [%d][%d]:", row, column);
            scanf("%d", &sales[row][column]);
        }
}
```

Code Listing 15.1 : Program segment to add data at runtime

The program segment declares two dimensional array sales with three rows and five columns. The logic given in for loops will read total fifteen different values for sales array. Individual elements of sales array can be displayed on screen using the program segment given in code listing 15.2.

```
for( row = 0; row < 3; row++) {
    for (column = 0; column < 5; column++){
        printf("[%d][%d] : = %d \n", row, column, sales[row][column]);
    }
}
```

Code Listing 15.2 : Program segment to show contents of array

Example of Two Dimensional Arrays

Let us now understand a program to illustrate runtime array initialization and accessing values from one and two dimensional arrays. The program will declare two dimensional arrays for maintaining marks of students. The code of this program is given in code listing 15.3, along with its output.

```
/* Example 3: Program to illustrate use of one and two dimensional array. The program will
maintain marks of two students in three quizzes using two dimensional arrays. The total marks
of each student will be displayed.*/

#include<stdio.h>
#define STD 2     /*Number of rows for student data*/
#define QUIZ 3    /*Number of columns for quiz data*/

int main(){
        int marks[STD][QUIZ];                  /* Marks array for 2 students and 3 quizzes */
        int i, j, total_quiz[STD] = {0};

        for( i = 0; i < STD; i++)              /* Loop for student row */
        {
            for( j = 0; j < QUIZ;  j++)        /* Loop for quiz column */
                        {
                printf("Enter marks of student %d in quiz %d: ", i+1, j+1);
                scanf("%d", &marks[ i ][ j ]); /*  Reading marks of quiz */
                total_quiz[ i] = total_quiz[ i] + marks[ i][ j];
            }
        }

         /* Logic to print student no., quiz marks and total... */

        for ( i = 0 ; i < QUIZ; i++)           /* Loop for printing quiz number... */
        {
          printf(" \tQuiz %d" , i+1);
        }

        printf(" Total \n");
        for( i = 0; i < STD; i++)              /*  Loop for student row */
        {
          printf("Student %d:", i+1);
           for( j = 0; j < QUIZ;  j++)         /* Loop for quiz column */
           {
                        printf("%d \t", marks[ i ][ j ]);
           }
          printf("%d \n", total_quiz[ i]);
        }
        return 0;
}
/* End of Program*/
```

```
/*************************************************************************/
Output:

Enter marks of student 1 in quiz 1: 20
Enter marks of student 1 in quiz 2: 30
Enter marks of student 1 in quiz 3: 40
Enter marks of student 2 in quiz 1: 50
Enter marks of student 2 in quiz 2: 40
Enter marks of student 2 in quiz 3: 20

             Quiz1      Quiz2      Quiz3      Total
Student 1:   20         30         40         90
Student 2:   50         40         20         110
/*************************************************************************/
```

Code Listing 15.3 : Code and output of example 15.3

Explanation

Here we have used two symbolic constants STD and QUIZ in our program. STD refers to the number of students to be maintained in a program. QUIZ refers to the number of quizzes to be maintained in a program. We have declared two dimensional array marks to maintain quiz marks of students. One dimensional array total_quiz is declared for storing total marks of all quizzes by a student. At the time of declaration of total_quiz, all elements are initialized to zero. Two counter variables i and j are declared to repeat *for* loop body statements as per our requirement.

A nested for loop is used to read quizzes marks of students. Within the loop we had calculated total marks of all quizzes by a student. The remaining for loops are used to display output appropriately on the screen. The given program will be able to maintain two students' marks for three quizzes only. But we can make our program more dynamic by changing values of STD and QUIZ variable in our program.

No Array Bound Check

Arrays are one of the strength of C language but it is important to note that there is no array bound check in C language. Array bound check means checking the boundaries of array declared in a program. Consider the following array declaration statement:

```
int number[5];
```

This will declare an array variable number capable of storing five integer values. We can access individual array elements by using number[0], number[1].....number[4]; C provides power to the programmer to write any index value within the [ ] of an array. This means we can access number[-1] and also number[6] or any other illegal index location. As there is no bound check of array index, we will not get any error but the program will give us the garbage data or it will crash.

Summary

In this chapter we had learnt important characteristics related to use of one dimensional and two dimensional arrays in C. We need to remember that the array datatype can be any valid C language data type. In array declaration the subscript number must be an integer or integer expression. An array index number starts with zero. Character arrays (string) are terminated by the special null character '\0'. There is no array boundary check in C language. It is programmer's responsibility to refer correct array index number in a program.

EXERCISE

1. Give the advantages of using one dimensional array in a program.
2. State the difference between one and two dimensional arrays.
3. What do you mean by runtime and compile time array initialization of an array?
4. Find errors, if any in the following code, correct it and write the output.

(a)
```
#include <stdio.h>
int main( ) {
    int num[3][3] = {{1, 2}, {3, 4}};
    printf("%d \n", number[1][1]);
    return 0;
}
```

(b)
```
#include<stdio.h>
void main()
{
    char str[ ] = 'INDIA'
    printf("%c %c %c %c  c", str[0], str[1], str[2], str[0], str[4]);
}
```

5. Fill in blanks :

(a) The two-dimensional arrays have rows and _____.

(b) An array element occupies _____ memory locations.

(c) Index number contained within square brackets of an array is also known as _____.

(d) A subscript number must be an integer or _____ expression.

(e) Strings in a C language are terminated by the special character _____.

6. State True or False :

(a) An array can be initialized only at runtime of a program.

(b) The C array index number starts with one.

(c) The simplest form of multidimensional array is one-dimensional array.

(d) C language provides the string as inbuilt data type.

(e) First bracket in the declaration of two dimensional arrays specifies number of rows.

(f) The statement: int temperature[6] = {35, 32, 42}; is valid array initialization.

(g) The statement: int num[ ] = { {1,1}, {2,2}}; is valid initialization.

7. Choose the correct option from the following :

(1) What happens if a C program tries to access a value from an array element whose subscript exceeds the size of array ?

(a) The element will be set to 0.

(b) The compiler would report an error.

(c) The program may crash or gives garbage data.

(d) The array size increases automatically.

(2) What will be the output of following program segment?

```
int num[5] = {1, 2, 3, 4, 5};
int i, j;
i = num[1];
j = num[2];
printf("%d, %d, %d", i, j, num[0]);
```

(a) 1, 2, 3 (b) 2, 3, 1 (c) 1, 2, 0 (d) 3,4,5

(3) What will be the output of following program segment ?

```
int num[5] = {1, 2, 3};
printf("%d, %d", num[0], num[3]);
```

(a) 1, 2 (b) 1, 0 (c) 1, 3 (d) 2,3

LABORATORY EXERCISE

Write a C program for performing followings tasks :

1. Read two five by five matrices from the user. Add these two matrices into third matrix and display the answer on screen.

2. Read marks of 20 students using an array in a program as shown below:

Roll No	Marks
1	60
2	70
...	....
20	75

   - Display the Roll No. of students having highest marks amongst all.
   - Display total number of students having marks greater than 50.
   - Display total number of students having marks less than 50.

3. Read 10 integer numbers in one dimensional array. Display the array elements into reverse order. For example, if user input is 10, 20, 30, 40 then out output is 40, 30, 20, 10.

4. Find maximum, minimum and average of N elements of an array.

5. Read integer values for 3 X 3 Matrix from the user. Calculate and display sum of all elements of a Matrix.



Function

One of the strength of C language is its ability to define and use functions. Function is a group of statements that performs a specific task. The function is also knows as *method*, *sub-routine* or *procedure*. So far we have used number of C functions in our programs. We are familiar with the use of main( ), printf( ) and scanf( ). They are examples of C functions. Every executable C program has at least one main( ) function.

The use of function in a program makes it modular. Modularity means partitioning a complex problem into small sub-problems which are easy to understand and maintain. Once a problem is divided into small sub-problem, we can write a function for each sub-problem. The problem is then solved by using all functions together in a program. In this chapter we will discuss various types of C functions. We will also learn how function can be defined and used in a program.

There are two categories of functions in C.

(1) Library functions or System defined functions

(2) User defined functions

Library Functions

The C language standard library provides many built-in functions that our program can use. They are also known as system defined functions. For example printf( ), scanf( ), sqrt( ) and cos() are examples of library functions. Library functions are not required to be written by normal user. These functions are available in compiled form in C library. Library functions are placed in various header files. For example, sqrt( ) and cos( ) are related to mathematics hence they are placed in <math.h> header file. To use any library function, we have to include its corresponding header file in our program. The list of some known header files and functions in that file are given in Appendix IV of this book.

Let us understand the use of one of the library function through example 16.1. The program uses inbuilt library function pow( ) that returns x raised to the power of y. Figure 16.1 gives the code listing and output of the example 16.1.

In the example 16.1, we had used printf( ) function which is defined in <stdio.h> header file. We had also used pow( ) function which returns the value of 8 raised to power of 2. The pow() function is defined in <math.h> file.

13-1.c - SciTE

File Edit Search View Tools Options Language Buffers Help

1 13-1.c

```
/* Example 1: Program to illustrate use of pow() library
funtion */

#include<stdio.h>
#include<math.h>
int main()
{
    float answer;

    answer = pow(8,2);
    printf("The value of 8 raised to 2 is %f\n",answer);

    return 0;
}
```

```
>gcc -pedantic -Os -std=c99 13-1.c -o 13-1
>Exit code: 0
>./13-1
The value of 8 raised to 2 is 64.000000
>Exit code: 0
```

Figure 16.1 : Code listing and output of Example 16.1

User Defined Functions

The functions developed by user are known as user defined functions. The main( ) is special type of user defined function in C. The program execution starts from main( ) function. If we write our program using only main( ) function then it may become large. Testing and debugging of such program may be become difficult for programmer. If program is divided into small subprograms (functions) then it is much easier to understand, test and debug it.

Advantages of Functions

Followings are the advantages of using functions in our program :

- Dividing program into functions makes understanding and maintenance of the program easier.
- Functions written once can be called many times in our program. This reduces the actual length of program and thus saves memory.
- Functions help us to avoid redundant programming of similar tasks.
- A function written for one program may be used in other program.
- Same function can be used with different values of arguments.

Let us now try to understand how to use different functions in our program. Figure 16.2 shows control flow of a program that uses multiple functions.

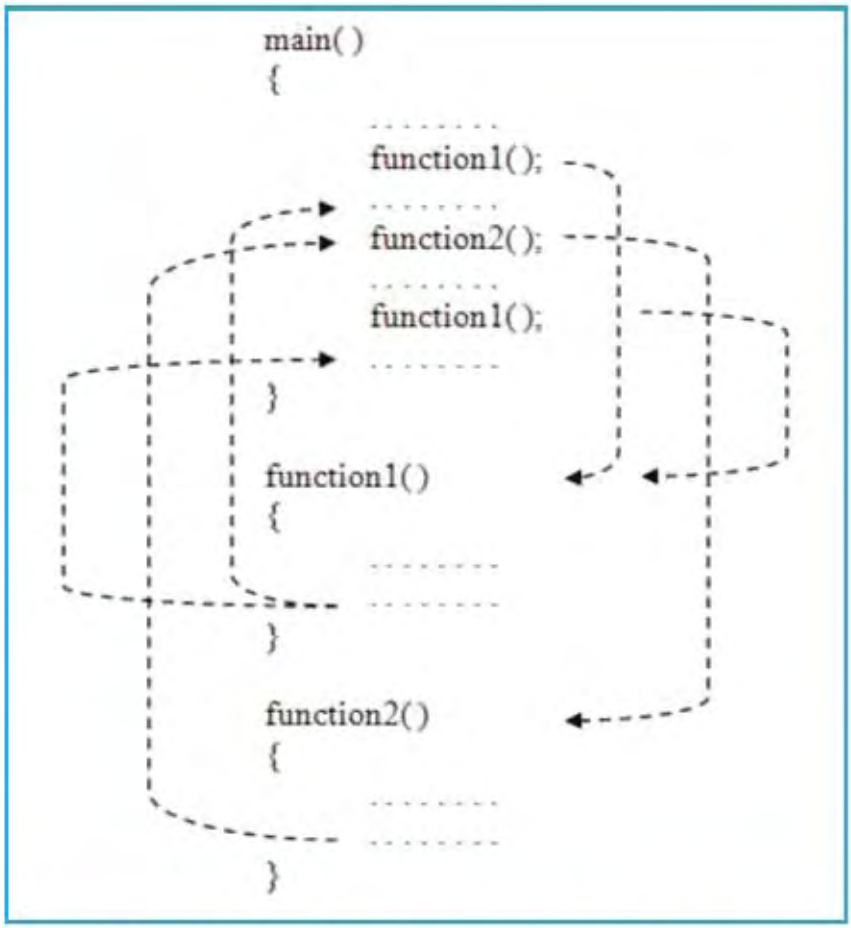


Figure 16.2 : Flow of control in a program with multiple functions

Figure 16.2 shows that in main( ), function1( ) is called twice and function2( ) is called once. During execution of statements in main( ) when control reaches to statement function1( ), then control is transferred to the body of function1( ). After executing all the statements inside body of function1 ( ) again control is returned to statement after function1( ) in the main ( ) function. In the same way again function2( ) and then function1( ) is called from the main program.

Function Definition

If we want to use user defined function in our program then we have to inform this to compiler. We can inform the compiler by writing function definition. A function cannot be used without its definition. In C language, function can be defined using following syntax.

```
return_data_type   function_name (arguments)
{
        function statements;
}
```

Here return_data_type specifies the type of information return by the function. If the return value from the function is integer value then return_data_type is int. If function does not return any value then the return_data_type is void.

The function_name specifies the name of user defined function. The rules for giving name to functions are same as rules of naming variable. We should give meaningful function name. For example add ( ) for performing addition, avg( ) for performing average, etc. The arguments show the input values provided to the function along with their data types. For more than one argument, all arguments are separated by commas in the function definition. Statements between opening and closing curly brace are known as function body. Based on requirements in a program, function body may contains local variable declarations and return statements. We will discuss about variable declaration in function and return statements later in this chapter.

Note that there is no need to define library function as they are already defined. All user defined functions needs to be defined as function prototype before being used in a program. Function prototype means declaration of a function before their use in a main ( ) function.

Function Call

As stated earlier, every C program starts with function main( ). From main( ) functions we are calling other library or user defined functions. To call a function, use function name with required number of parameters. Let us look at the example 16.2 which illustrate how to call a function from the main( ). Figure 16.3 gives the code listing of the example 16.2, its output is shown in figure 16.4.

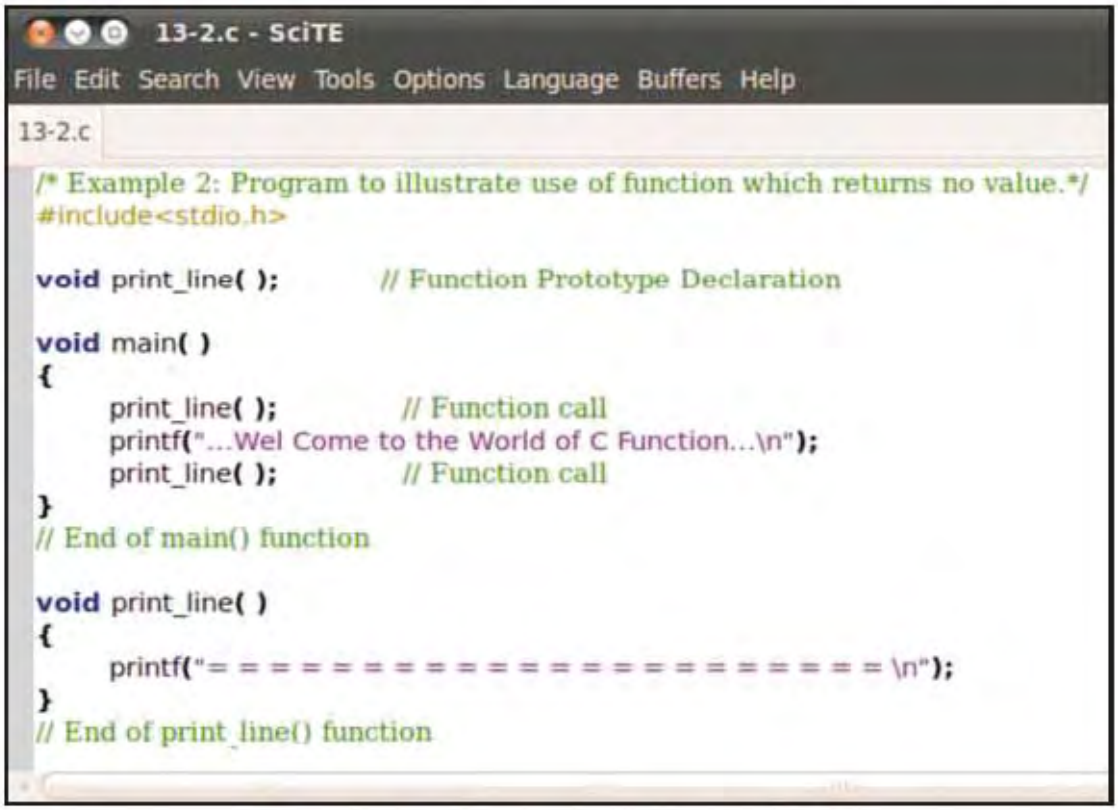


Figure 16.3 : Code listing of Example 16.2

```
gcc 13-2.c
>gcc 13-2.c
>Exit code: 0
./a.out
>./a.out
= = = = = = = = = = = = = = = = = = = = = = = = =
...Wel Come to the World of C Function...
= = = = = = = = = = = = = = = = = = = = = = = = =
>Exit code: 45
```

Figure 16.4 : Output of Example 16.2

Explanation

In the program one user defined function print_line( ) is used. The print_line( ) statement within the main function calls this function. The control moves from main( ) to print_line( ) function where it prints the line pattern, then control moves back to the main( ). The next printf statement in main prints the message. Next statement in main( ) once again calls print_line( ) function. In the example print_line( ) function is called twice. As this function does not return any value to the main( ) hence its return type is void.

The Return Statement

User defined functions may or may not return any value to the calling function. To return any value to the calling function, we may use return statement inside function body. The format of return statement is as under :

return; or return (expression);

The first form of return does not return any value. It returns only control back to the calling function. When return type of function is void then there is no need to use return statement at the end of function body.

Look at figure 16.5 that shows the code listing of example 16.3 to understand how the function returns a value to the calling function. The output is shown in figure 16.6.

Explanation

In figure 16.5 we have used a user defined function add( ); In main( ) function we have declared a variable called total. When we call the add( ) function from the main( ), the control is transferred to the add() function. Two local variables are declared namely num1 and num2 in function add. The next statements within add( ) displays message and read two numbers using printf and scanf statement respectively. Last statement returns addition of two numbers which is stored in the total variable of main( ) function. The total is then printed on screen.

13-3.c - SciTE

File Edit Search View Tools Options Language Buffers Help

13-3.c

```c
/* Example 3: Program to illustrate use of function which returns a value.*/

#include<stdio.h>
int add( );                    // Function Prototype Declaration

void main( )
{
      int total;
      total = add( );          // Function call
      printf("The addition of two numbers: %d \n", total);
}
// End of main() function...

int add( )
{
      int num1, num2;
      printf("Enter two numbers: ");
      scanf("%d %d", &num1, &num2);
      return(num1 + num2);
}
// End of add() function...
```

Figure 16.5 : Code listing of Example 16.3

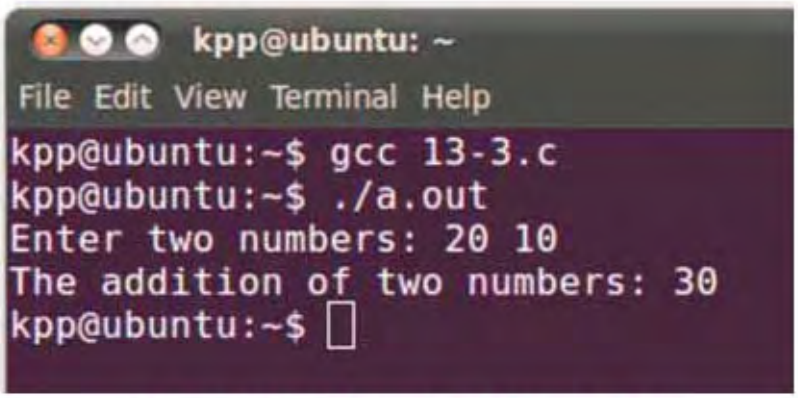


Figure 16.6 : Output of Example 16.3

Scope of Variables

By scope of variable we mean the part of program where the variable is accessible. In C scope of variables are of two types:

(i) Global variables

(ii) Local variables

Figure 16.7 explains the scope of variable in a C program.

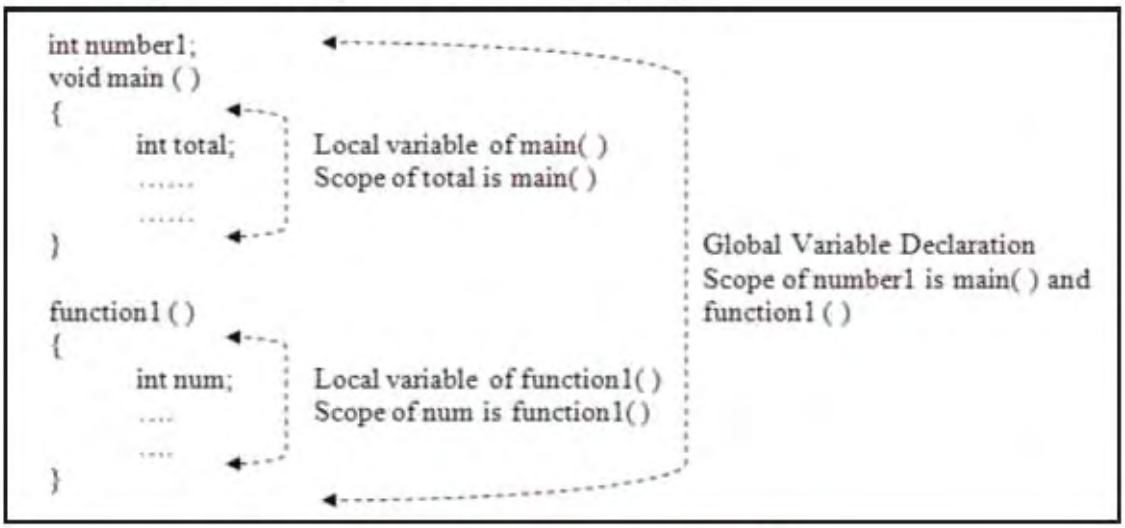


Figure 16.7 : Scope of Variable

Here the variable *number1* as it is defined above function main is known as global variable. The global variable *number1* is accessible in both main( ) and function1( ). The local variable *int total* variable is accessible only within main( ) and not in function1( ). Similarly the local variable *int num* is accessible only within function1( ) and not in main( ).

Before getting in to the detail characteristics of C function, let us see the figure 16.8 which shows various components of the function.

As shown in figure 16.8 initially we have defined function prototype as int add(int, int); this shows that there are two arguments of integer types. Function prototype is discussed later in the chapter. The void main( ) { .... } is the caller function which calls add( ) function with two actual parameters num1 and num2. Here user defined function int add( ) indicates that it returns integer value to the caller function void main( ). Two integer formal parameters "a" and "b" are defined in add( ) function.

These formal parameters accept the value sent at the time of calling function from the main( ). The integer variable result returns the addition of two numbers to the caller function which is stored in total variable of main( ).

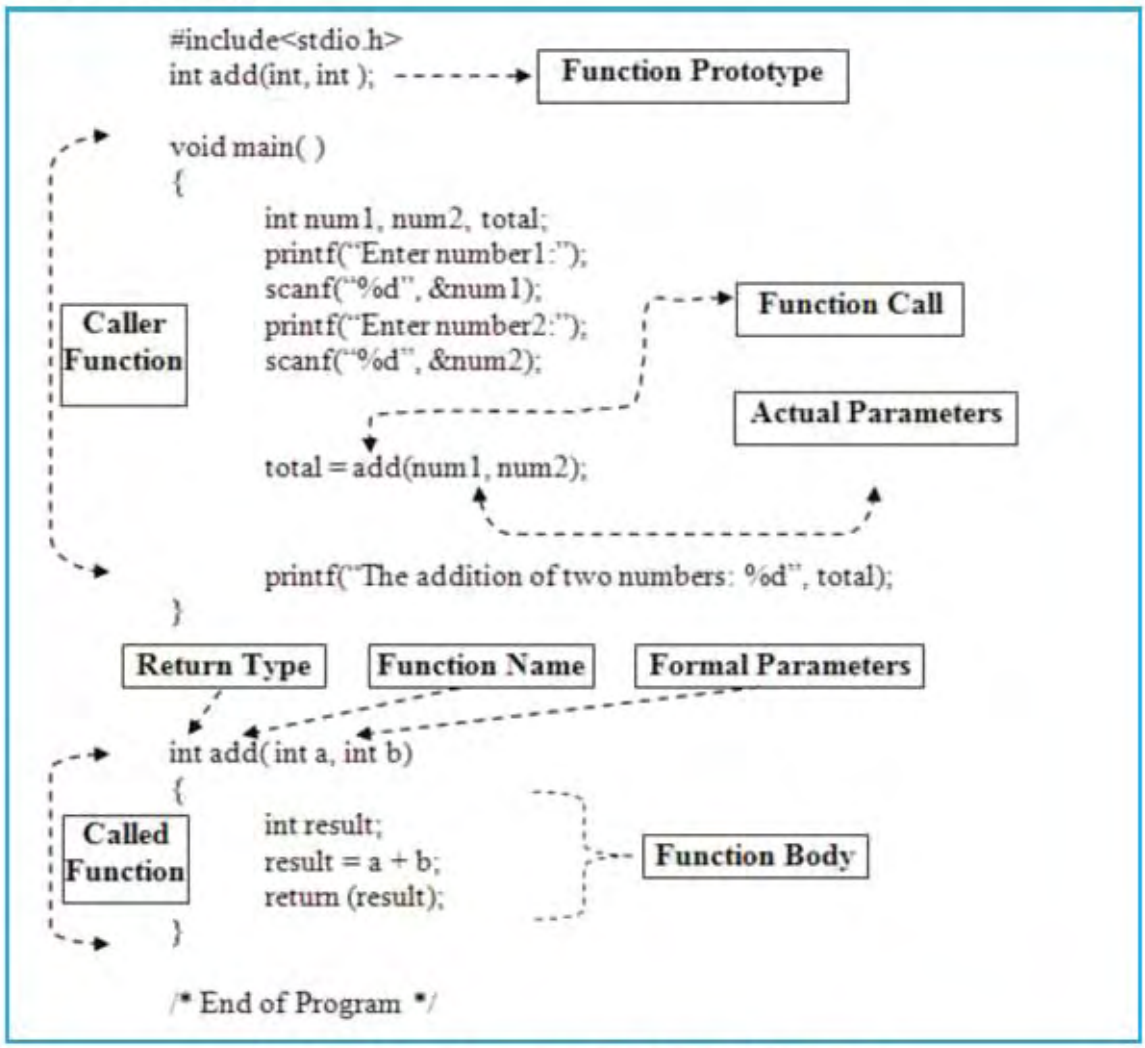


Figure 16.8 : Components of the C functions

Function Prototype

The function prototype is needed when the function is defined after the main( ) in a program. When we call any function in our program, compiler searches for a corresponding function definition. This is done to verify whether the function call is correct or not. If function call matches the header of the function in terms of number of arguments with their data types, it is valid otherwise compiler issues an error. For example

int add(int, int); which is a prototype of function int add(int a, int b)

Here prototype shows that add( ) is a function having two integer arguments and returns integer value. The function prototypes are written for each user defined function in beginning of the program.

Function Arguments or Parameters

The basic idea behind using arguments or parameters in functions is to pass data between the calling and the called function. When the calling function sends the data to the called function it is called as argument or parameter passing.

The arguments used at the time of function definition are known as *formal* arguments. The call to such function needs same number of arguments to be passed to it at the time of function call. These arguments are known as *actual* arguments. The formal and actual arguments should match in number, type and order. If actual arguments are more than formal arguments, the additional actual arguments are simply discarded. When actual arguments are less than formal arguments, the unmatched formal arguments are initialized to garbage value. Also any mismatch in data types of arguments will result in garbage initialization.

Types/Categories of Functions

Depending on presence of arguments and return value in a function, they fall under following categories:

(i) Function with no arguments and no return values

(ii) Function with arguments and no return values

(iii) Function with arguments and return values

Function with No Arguments and No Return Values

When function has no arguments and no return value, it falls under this category. The called function does not receive any data value from calling function. Similarly due to no return value, the calling function does not receive any data from called function. In short there is no data transfer between calling and called functions. This is shown in figure 16.9. The dotted line in the figure indicates no data transfer but only transfer of control.

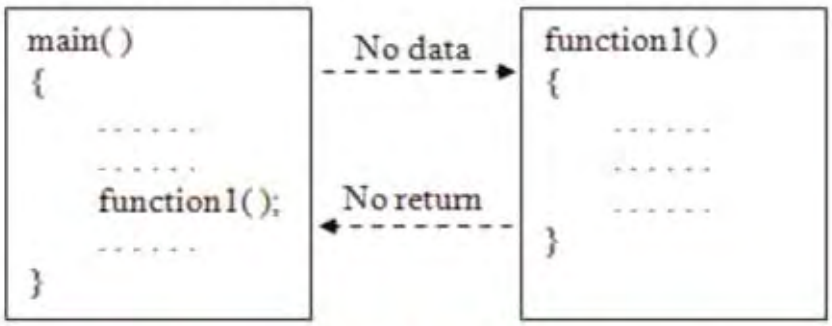


Figure 16.9 : Functions with no data communications

Example 16.2 explained earlier in this chapter is the example of category function with no arguments and no return values.

Function with Arguments and No Return Values

The communication between the calling function and the called function with the arguments but no return value is shown in figure 16.10.

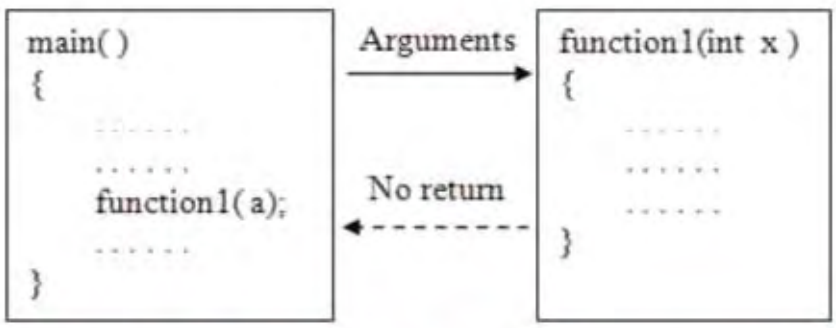


Figure 16.10 : Function with one way data communication

Let us try to understand the example 16.4 which will read a number from user and find its cube using function. We will send a number from calling function main( ) to called function cube( ) as an argument. The called function finds the cube and displays it on screen. Figure 16.11 gives the code listing of the example 16.4 while figure 16.12 shows its output.

13-4.c - SciTE

File Edit Search View Tools Options Language Buffers Help

13-4.c

```
/* Example 4: Program to illustrate use of function with arguments and no return value.*/
#include<stdio.h>

void cube ( int );                  // Function Prototype Declaration

void main( )
{
      int number;
      printf("Enter your number: ", number);
      scanf("%d", &number);
      cube( number );               // Function call
}
//End of main () function

void cube(int num1)        // Function to find and display cube of a given number.
{
      printf("The cube of entered number is : %d \n", num1*num1*num1);
}

// End of cube() function
```

Figure 16.11 : Code listing of Example 16.4

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 13-4.c
kpp@ubuntu:~$ ./a.out
Enter your number: 10
The cube of entered number is : 1000
kpp@ubuntu:~$
```

Figure 16.12 : Output of Example 16.4

Explanation

Here main( ) function accepts a number from the user which is stored in to a number variable. This number variable is passed as an actual argument to the cube( ) function. In cube( ) function formal argument num1 gets the value of number variable. The cube( ) function then calculate and display the cube of given number using printf statement.

Function with Arguments and Return Values

Let us understand the two way communication between the calling function and the called function. The function with arguments and return value is shown in figure 16.13.

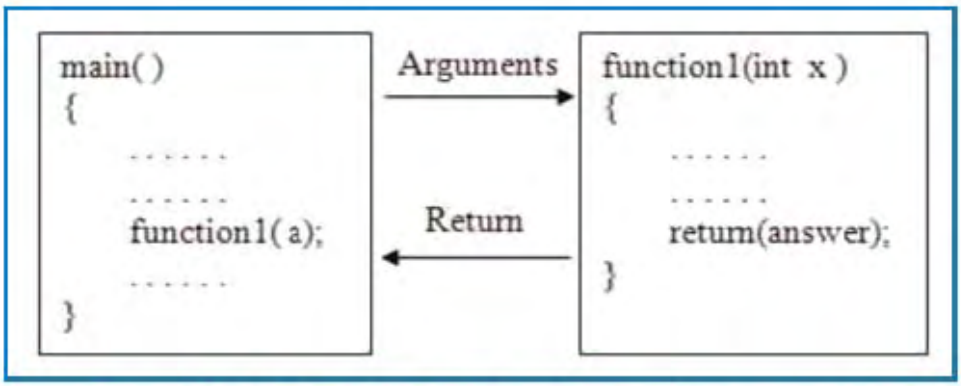


Figure 16.13 : Function with two way data communication

Let us try to understand the example 16.5 which will read a number from user and find its square using sqr( ) function. We will send a number from calling function main( ) to called function sqr() as an argument. The called function finds the square of number and returns it to the calling function. Figure 16.14 gives the code listing of the example 16.5 while figure 16.15 shows its output.

13-5.c - SciTE

File Edit Search View Tools Options Language Buffers Help

13-5.c

```
/* Example 5: Program to illustrate use of function with arguments and return value.*/
#include<stdio.h>

int sqr ( int );          // Function Prototype Declaration

void main( )
{
      int number, result;
      printf("Enter your number: ", number);
      scanf("%d", &number);
      result = sqr( number );          // Function call with argument
      printf("The square of %d is %d. \n", number, result);
}
//End of main() function

int sqr(int num1)  // Function to find square of a number and return the answer
{
      int answer;
      answer = num1 * num1;
      return (answer);
}
//End of sqr() function
```

Figure 16.14 : Code listing of Example 16.5

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 13-5.c
kpp@ubuntu:~$ ./a.out
Enter your number: 10
The square of 10 is 100.
kpp@ubuntu:~$
```

Figure 16.15 : Output of Example 16.5

Explanation

The main( ) function accepts a number from the user which is stored in to a number variable. Then number variable is passed as an actual argument to the sqr( ) function. In sqr( ) function formal argument num1 gets the value of number variable. The sqr( ) function then calculates the square and stores it into answer variable. The last statement in sqr( ) return the value stored in answer variable to the calling function. The caller function accepts the return value and stored it into result variable. Last statement in the main( ) displays the square of given number on screen.

Summary

In this chapter we had discussed basic concepts related to the C functions. We had seen that use of function in a program makes it modular. We had learnt that how to use Library and User defined functions in our program. We had learnt that how to use global and local variables in our program. We had seen that how function can return a value to the calling function. The basic idea behind using arguments or parameters in the functions is discussed.

EXERCISE

1. What do you mean by a function ? State advantages of using a function in a program.
2. State the advantages of using library function in a program.
3. What do you mean by scope of a variable ?
4. State how data/information can be passed to a function.
5. What is the meaning of function prototype ?
6. Differentiate between followings :
   (a) Global and Local variables
   (b) Formal and Actual parameters
   (c) User defined function and Library function
7. Find errors (if any) in the following code; correct it and write the output :

   (a)

```
#include<stdio.h>
void draw_line( );
void main( )
{
   draw_line;
            printf("...Wel Come to the C Function...\n");
   draw_line;
}
void print_line( )
{
   printf("= = = = = = = = = = = = = = = = = = = = \n");
}
```

(b)

```
#include<stdio.h>
int sqr ( int );
void main( )
{
   int number = 5, result;
   result = sqr( );
   printf("The square of 5 is %d.", result);
}
int sqr(int num1)
{
   int answer;
   answer = num1 * num1;
   print (answer);
}
```

8. Fill in blanks :

(a) The _____ statement returns the information from a function.

(b) If function does not return any value then its return data type is _____

(c) The library function sqrt( ) is defined in _____ header file.

(d) The variable declared above main( ) function are called _____ variables.

9. State True or False :

(a) C program functions can pass only one value.

(b) Function cannot send multiple arguments together.

(c) Same user defined function cannot be called multiple times from main( ).

(d) The main( ) is also one type of function in C program.

(e) The function prototype declarations for library functions are not required in the beginning of a program.

10. Choose the correct option from the following :

(1) The main() function in C language is a

(a) User defined function (b) Library function

(c) Keyword (d) Reserved function

(2) By default the main() function returns

(a) Char value (b) Float value

(c) Integer value d. Double value

(3) Which keyword is used as a return type when function does not return any value?

(a) int (b) main (c) void (d) auto

(4) Which of the following is not an inbuilt library function ?

(a) pow() (b) printf() (c) sum() (d) sqrt()

(5) What will be the value of variable number2 after execution of following code :

```
int main(){
    int number1 = 2, number2;
    number2 = sqr(number1);
    printf("number2 = %d",number2);
    return 0;
}
int sqr(int n){
    return(n*n);
}//End of program
```

(a) number2 = 4 (b) number2 = 6

(c) number2 = 8 (d) number2 = 18

LABORATORY EXERCISE

Write a C program to perform following tasks :

1. Read an integer number N in main( ) function. Pass the value of N to the function as an argument. Display the sum of following series using a function for a given N number.

   1 + 2 + 3 + 4 +..... + N

2. Write a user defined function which will display Welcome message 10 times.
3. Define a function which will calculate the x to the power n.
4. Write a user defined function to find maximum from the given two numbers.
5. Read a number from main( ), pass it to the user defined function as an argument and display whether given number is positive or negative.
6. Read a number in your program and apply library function abs( ), sin( ), cos( ) and log( ) on it and display the result.
7. Accept a character arrays (string) from user and display the string in reverse order.



Appendix I



List of decimal numbers with corresponding octal and hexadecimal numbers

Decimal Number (0,1,2,…,9)	Binary Number (0,1)	Octal Number (0,1,2,…,7)	Hexadecimal Number (0,1,2,…,9,A,B,C,D,E,F)
1	00001	1	1
2	00010	2	2
3	00011	3	3
4	00100	4	4
5	00101	5	5
6	00110	6	6
7	00111	7	7
8	01000	10	8
9	01001	11	9
10	01010	12	A
11	01011	13	B
12	01100	14	C
13	01101	15	D
14	01110	16	E
15	01111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14



Appendix II



ASCII Values of Characters

ASCII		ASCII		ASCII		ASCII	
Value	Character	Value	Character	Value	Character	Value	Character
000	NUL	032	blank	064	@	096	
001	SOH	033	!	065	A	097	a
002	STX	034	"	066	B	098	b
003	ETX	035	#	067	C	099	c
004	EOT	036	$	068	D	100	d
005	ENQ	037	%	069	E	101	e
006	ACK	038	&	070	F	102	f
007	BEL	039	'	071	G	103	g
008	BS	040	(	072	H	104	h
009	HT	041	)	073	I	105	i
010	LF	042	*	074	J	106	j
011	VT	043	+	075	K	107	k
012	FF	044	,	076	L	108	l
013	CR	045	-	077	M	109	m
014	SO	046	.	078	N	110	n
015	SI	047	/	079	O	111	o
016	DLE	048	0	080	P	112	p
017	DC1	049	1	081	Q	113	q
018	DC2	050	2	082	R	114	r
019	DC3	051	3	083	S	115	s
020	DC4	052	4	084	T	116	t
021	NAK	053	5	085	U	117	u
022	SYN	054	6	086	V	118	v
023	ETB	055	7	087	W	119	w
024	CAN	056	8	088	X	120	x
025	EM	057	9	089	Y	121	y
026	SUB	058	:	090	Z	122	z
027	ESC	059	;	091	[	123	{
028	FS	060	<	092	\	124	|
029	GF	061	=	093	]	125	}
030	RS	062	>	094	?	126	~
031	US	063	?	095	-	127	DEL

The characters 33 to 126 are printable. The other characters are control characters; they cannot be printed.



Appendix III



Summary of C Operators

Operator	Operation Used for	Associativity	Precedence
( )	Function call	Left to Right	First
[ ]	Array expression		
→	Structure operator		
.	Structure operator		
+	Unary plus	Right to Left	Second
-	Unary minus		
++	Increment		
- -	Decrement		
!	Logical NOT		
~	Bitwise NOT		
*	Pointer operator		
&	Address operator		
sizeof( )	Size of operand		
*	Multiplication	Left to Right	Third
/	Division		
%	Modulo division		
+	Binary addition	Left to Right	Fourth
-	Binary subtraction		
<<	Left shift	Left to Right	Fifth
>>	Right shift		
<	Less than	Left to Right	Sixth
<=	Less than equal to		
>	Greater than		
>=	Greater than equal to		
= =	Equal to	Left to Right	Seventh
!=	Not equals to		
&	Bitwise AND	Left to Right	Eighth
^	Bitwise XOR	Left to Right	Ninth
|	Bitwise OR	Left to Right	Tenth
&&	Logical AND	Left to Right	Eleventh
||	Logical OR	Left to Right	Twelfth
? :	Conditional operator	Right to Left	Thirteenth
=, *=, - =, &=, +=,			
^=, |=, <<=, >>=	Assignment Operator	Right to Left	Fourteenth
,	Comma operator	Left to Right	Fifteenth



Appendix IV



Some of the Regularly Used Header Files

As we already know that in c Language functions are very important. C has a collection of inbuilt functions to perform various operations. These functions are grouped and stored in header files. Collection of such header files are called C library. Following is the list of some of the header files that programmers might like to use.

File name	Purpose
<stdio.h>	Standard I/O header file
<ctype.h>	Character testing and conversion functions
<math.h>	Mathematical functions
<stdlib.h>	Different types of functions used for string conversions, memory allocations, random number generation.
<string.h>	String manipulation functions

The some of the functions available in the above header files are shown below. At times function need to be provided an input. The input is part of function definition and is known as arguments. The arguments may have different data types. To identify the arguments we have used the character codes given below :

c	- argument of type char
d	- argument of type double
f	- argument of type file
i	- argument of type int
l	- argument of type long
p or *	- argument of type pointer
s	- argument of type string
u	- argument of type unsigned integer

The functions at times are also capable of returning a value of different data types. The list below gives idea about different function, its argument type and the type of value it returns.

Functions in <stdio.h> file

Function	Return Value	Description
getc(f)	int	Enter a single character from file f.
getchar(void)	int	Enter a single character from the standard input device.
gets(s)	char*	Enter string s from the standard input device.
printf(...)	int	Send data items to the standard output devices.
putc(c,f)	int	Send a single character to file f.
putchar(c)	int	Send a single character to the standard output device.
puts(s)	int	Send string s to the standard output device.
scanf(...)	int	Enter data items from the standard input device.

Functions in <cctype.h> file

Function	Return Value	Description
isalnum(c)	int	Determine if argument is alphanumeric. Returns nonzero value if true; 0 otherwise.
isaplpha(c)	int	Determine if argument is alphabet. Returns nonzero value if true; 0 otherwise.
isascii(c)	int	Determine if argument is an ASCII character. Returns non zero value if true; 0 otherwise.
iscntrl(c)	int	Determine if argument is an ASCII control character. Returns nonzero value if true; 0 otherwise.
isdigit(c)	int	Determine if argument is a decimal digit. Returns nonzero value if true; 0 otherwise.
islower(c)	int	Determine if argument is lowercase. Returns nonzero value if true; 0 otherwise.
isodigit(c)	int	Determine if argument is an octal digit. Returns nonzero value if true; 0 otherwise.
isprint(c)	int	Determine if argument is a printable ASCII character. Returns nonzero value if true; 0 otherwise.
ispunct(c)	int	Determine if argument is a punctuation character. Returns nonzero value if true; 0 otherwise.
isspace(c)	int	Determine if argument is whitespace character. Returns non-zero value if true; 0 otherwise.
isupper(c)	int	Determine if argument is uppercase. Returns nonzero value if true; 0 otherwise.
isxdigit(c)	int	Determine if argument is a hexadecimal digit. Returns nonzero value if true; 0 otherwise.
toascii(c)	int	Convert value of argument to ASCII.
tolower(c)	int	Convert letter to lowercase.
toupper(c)	int	Convert letter to uppercase.

Functions in <math.h> file

Function	Return Value	Description
acos(d)	double	Returns the arc cosine of d.
asin(d)	double	Returns the arc sine of d.
atan(d)	double	Returns the arc tangent of d.
atan2(d1, d2)	double	Returns the arc tangent of d1/d2.
ceil(d)	double	Returns a value rounded up to the next higher integer.
cos(d)	double	Returns the cosine of d.
cosh(d)	double	Returns the hyperbolic cosine of d.
exp(d)	double	Raises e to the power d.
fabs(d)	double	Returns the absolute value of d.
floor(d)	double	Returns a value rounded down to the next lower integer.
fmod(d1,d2)	double	Returns the remainder of d1/d2 (with same sign as d1).
labs(1)	long int	Returns the absolute value of 1.
log(d)	double	Returns the natural logarithm of d.
log10(d)	double	Returns the logarithm (base 10) of d.
pow(d1,d2)	double	Return d1 raised to the d2 power.
sin(d)	double	Returns the sine of d.
sinh(d)	double	Returns the hyperbolic sine of d.
sqrt(d)	double	Returns the square root of d.
tan(d)	double	Returns the tangent of d.
tanh(d)	double	Returns the hyperbolic tangent of d.

Functions in <stdlib.h> file		
Function	**Return Value**	**Description**
abs(i)	int	Returns the absolute value of i.
atof(s)	double	Converts string s to a double-precision quantity.
atoi(s)	int	Converts string s to an integer.
atol(s)	long	Converts string s to a long integer.
calloc(u1,u2)	void*	Allocates block of memory having u1 elements each of length u2 bytes. Returns a pointer to the beginning of the allocated space.
exit(u)	void	Closes all files and buffers and terminate the program. (value of u is assigned by the function to indicate termination status)
free(p)	void	Frees a block of allocated memory whose beginning is indicated by p.
malloc(u)	void*	Allocates u bytes of memory. Returns a pointer to the beginning of the allocated space.
rand(void)	int	Returns a random positive integer.
realloc(p,u)	void*	Allocates u bytes of new memory to the pointer variable p. Returns a pointer to the beginning of the new memory space.

Functions in <string.h> file		
Function	**Return Value**	**Description**
strcmp(s1, s2)	int	Compares two strings lexicographically. Returns a negative value if s1 < s2; 0 if s1 and s2 are identical; and a positive value if s1 > s2.
strcmpi(s1,s2)	int	Compares two strings lexicographically irrespective of case. Returns a negative value if s1 < s2; 0 if s1 and s2 are identical; and a value if s1 > s2
strcpy(s1, s2)	char*	Copy string s2 to s1.
strlen(s)	int	Returns the number of character in string s.