Unit I

CHAPTER 2

DATA ABSTRACTION

Learning Objectives

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

- what is Abstract Data structures.
- Abstract data type.
- Difference between concrete and abstract implementation.
- Pairs.
- Data Abstration in Structure.

2.1 Data Abstraction- Introduction

Data abstraction is a powerful concept in computer science that allows programmers to treat code as objects — for example, car objects, pencil objects, people objects, etc. Programmers need not to worry about how code is implemented — they have to just know what it does.

This is especially important when several people are doing a project. Here project refers to the programming .With data abstraction, your group members won't have to read through every line of your code to understand. They can just assume that it does work.

Abstraction provides modularity (modularity means splitting a program in to many modules). Classes (structures) are the representation for "Abstract Data Types", (ADT)

2.2 Abstract Data Types

Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations.

The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation independent view. ***The process of providing only the essentials and hiding the details is known as abstraction***.

You can see that these definitions do not specify how these ADTs will be represented and how the operations will be carried out. There can be different ways to implement an ADT, for example, the List ADT can be implemented using singly linked list or doubly linked list. Similarly, stack ADT and Queue ADT can be implemented using lists.

Data abstraction replicate how we think about the world. For example, when you want to drive a car, you don't need to know how the engine was built or what kind of material the tires are made of. You just have to know how to turn the wheel and press the gas pedal. ***To facilitate data abstraction, you will need to create two types of functions: constructors and selectors.***

2.3 constructors and selectors

Constructors are functions that build the abstract data type. Selectors are functions that retrieve information from the data type.

For example, say you have an abstract data type called city. This city object will hold the city's name, and its latitude and longitude. To create a city object, you'd use a function like

city = makecity (name, lat, lon)

To extract the information of a city object, you would use functions like

- getname(city)
- getlat(city)
- getlon(city)

The following pseudo code will compute the distance between two city objects:

```
distance(city1, city2):
    lt1, lg1 := getlat(city1), getlon(city1)
    lt2, lg2 := getlat(city2), getlon(city2)
    return ((lt1 - lt2)**2 + (lg1 - lg2)**2))^1/2
```

In the above code read distance(), getlat() and getlon() as functions and read lt as latitude and lg longitude. Read := as "assigned as" or "becomes"

lt1, lg1 := getlat(city1), getlon(city1)

is read as lt1 becomes the value of getlat(city1) and lg1 becomes the value of getlon (city1).

Notice that you don't need to know how these functions were implemented. You are assuming that someone else has defined them for us.

It's okay if the end user doesn't know how functions were implemented. However, the functions still have to be defined by someone.

Let us identify the constructors and selectors in the above code

As you already know that Constructors are functions that build the abstract data type. In the above pseudo code the function which creates the object of the city is the constructor.

city = makecity (name, lat, lon)

Here makecity (name, lat, lon) is the constructor which creates the object city.

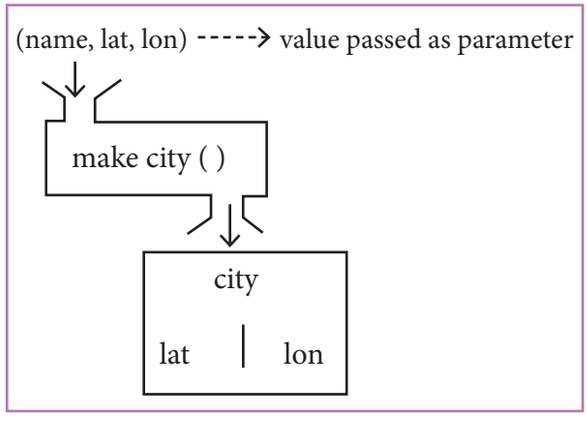
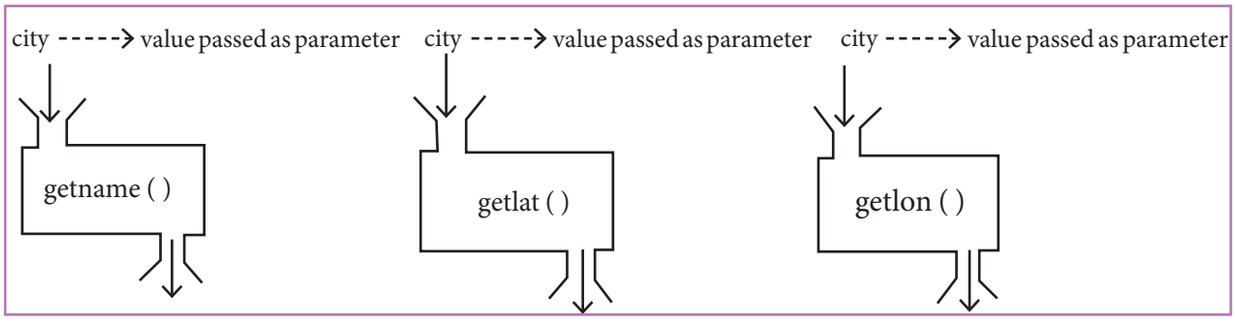


Fig 1 constructor

Selectors are nothing but the functions that retrieve information from the data type. Therefore in the above code

- getname(city)
- getlat(city)
- getlon(city)

are the selectors because these functions extract the information of the city object



Now let us consider one more example to identify the constructor and selector for a slope.Read - - as comments.

```
- - constructor
makepoint(x, y):
    return x, y
- - selector
 xcoord(point):
    return point[0]
- -selector
ycoord(point):
    return point[1]
```

Note

Data abstraction is supported by defining an abstract data type (ADT), which is a collection of constructors and selectors. Constructors create an object, bundling together different pieces of information, while selectors extract individual pieces of information from the object.

2.4 Representation of Abstract datatype using Rational numbers

The basic idea of data abstraction is to structure programs so that they operate on abstract data. That is, our programs should use data in such a way, as to make as few assumptions about the data as possible. At the same time, a concrete data representation is defined as an independent part of the program.

Note

In concrete data representation, a definition for each function is known

Any program consist of two parts. **The two parts of a program are, the part that operates on abstract data and the part that defines a concrete representation**, is connected by a small set of functions that implement abstract data in terms of the concrete representation. To illustrate this technique, let us consider an example to design a set of functions for manipulating rational numbers.

Example

A rational number is a ratio of integers, and rational numbers constitute an important sub-class of real numbers. A rational number such as 8/3 or 19/23 is typically written as:

<numerator>/<denominator>

where both the <numerator> and <denominator> are placeholders for integer values. Both parts are needed to exactly characterize the value of the rational number. Actually dividing integers produces a float approximation, losing the exact precision of integers.

8/3 =2.6666666666666665

However, you can create an exact representation for rational numbers by combining together the numerator and denominator.

As we know from using functional abstractions, we can start programming productively before you have an implementation of some parts of our program. Let us begin by assuming that you already have a way of constructing a rational number from a numerator and a denominator. You also assume that, given a rational number, you have a way of selecting its numerator and its denominator component. Let us further assume that the constructor and selectors are also available.

We are using here a powerful strategy for designing programs: **'wishful thinking'**. We haven't yet said how a rational number is represented, or how the constructor and selectors should be implemented.

Note

Wishful Thinking is the formation of beliefs and making decisions according to what might be pleasing to imagine instead of by appealing to reality.

Example: An ADT for rational numbers

- - constructor

- - constructs a rational number with numerator n, denominator d

rational(n, d)

- - selector

numer(x) → returns the numerator of rational number x

denom(y) → returns the denominator of rational number y

Now you have the operations on rational numbers defined in terms of the selector functions numer and denom, and the constructor function rational, but you haven't yet defined these functions. What you need is some way to glue together a numerator and a denominator into a compound value.

The pseudo code for the representation of the rational number using the above constructor and selector is

```
x,y:=8,3
rational(n,d)
  numer(x)/denom(y)
 - - output :  2.6666666666666665
```

2.5 Lists,Tuples

To enable us to implement the concrete level of our data abstraction, Some languages like Python provides a compound structure called Pair which is made up of list or Tuple. The first way to implement pairs is with the List construct.

2.5.1 List

List is constructed by placing expressions within square brackets separated by commas. Such an expression is called a list literal. List can store multiple values. Each value can be of any type and can even be another list.

Example for List is [10, 20].

The elements of a list can be accessed in two ways. The first way is via our familiar method of multiple assignment, which unpacks a list into its elements and binds each element to a different name.

```
lst := [10, 20]

x, y := lst
```

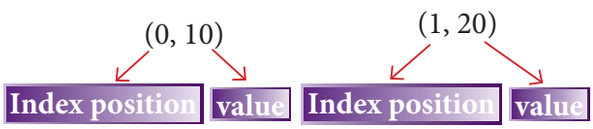
In the above example ***x*** will become10 and ***y*** will become 20.

A second method for accessing the elements in a list is by the element selection operator, also expressed using square brackets. Unlike a list literal, a square-brackets expression directly following another expression does not evaluate to a list value, but instead selects an element from the value of the preceding expression.

```
lst[0]
10
lst[1]
20
```

In both the example mentioned above mathematically we can represent list similar to a set.

lst[(0, 10), (1, 20)] - where



Any way of bundling two values together into one can be considered as a pair. Lists are a common method to do so. Therefore List can be called as Pairs.

Representing Rational Numbers Using List

You can now represent a rational number as a pair of two integers in pseudo code : a numerator and a denominator.

```
rational(n, d):
    return [n, d]
numer(x):
    return x[0]
denom(x):
    return x[1]
```

2.5.2 Tuple

Remember, a pair is a compound data type that holds two other pieces of data. So far,we have provided you with two ways of representing the pair data type. The first way is using List construct and the second way to implement pairs is with the tuple construct.

A tuple is a comma-separated sequence of values surrounded with parentheses. Tuple is similar to a list. The difference between the two is that you cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.

Example colour= ('red', 'blue', 'Green')

Representation of Tuple as a Pair

```
nums := (1, 2)
nums[0]
1
nums[1]
2
```

Note the square bracket notation is used to access the data you stored in the pair. The data is zero indexed, meaning you access the first element with nums[0] and the second with nums[1].

2.6 Data Abstraction in Structure

As you already know that List allow data abstraction in that you can give a name to a set of memory cells. For instance, in the game Mastermind, you must keep track of a list of four colors that the player guesses. Instead of using four separate variables (color1, color2, color3, and color4) you can use a single variable 'Predict', e.g.,

```
Predict =['red', 'blue', 'green', 'green']
```

What lists do not allow us to do is name the various parts of a multi- item object. In the case of a Predict, you don't really need to name the parts:

using an index to get to each color suffices.

But in the case of something more complex, like a person, we have a multi- item object where each 'item' is a named thing: the firstName, the lastName, the id, and the email. One could use a list to represent a person:

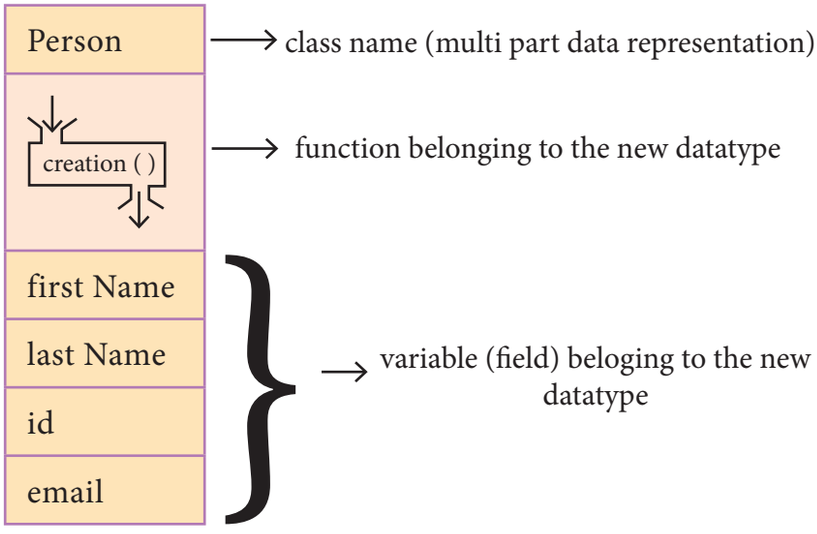
```
person=['Padmashri', 'Baskar', '994-222-1234', 'compsci@gmail.com']
```

but such a representation doesn't explicitly specify what each part represents.

For this problem instead of using a list, you can use the structure construct (In OOP languages it's called class construct) to represent multi-part objects where each part is named (given a name). Consider the following pseudo code:

```
class Person:
    creation( )
    firstName := " "
    lastName := " "
    id := " "
    email := " "
```

The new data type Person is pictorially represented as



Let main() contains	
p1:=Person()	statement creates the object.
firstName := " Padmashri "	setting a field called firstName with value Padmashri
lastName :="Baskar"	setting a field called lastName with value Baskar
id :="994-222-1234"	setting a field called id value 994-222-1234
email="compsci@gmail.com"	setting a field called email with value compsci@gmail.com
- - output of firstName : Padmashri	

The class (structure) construct defines the form for multi-part objects that represent a person. Its definition adds a new data type, in this case a type named Person. Once defined, we can create new variables (instances) of the type. In this example ***Person is referred to as a class or a type, while p1 is referred to as an object or an instance.*** You can think of class Person as a cookie cutter, and p1 as a particular cookie. Using the cookie cutter you can make many cookies. Same way using class you can create many objects of that type.

So far, you've seen how a class defines a data abstraction by grouping related data items. A class is not just data, it has functions defined within it. We say such functions are subordinate to the class because their job is to do things with the data of the class, e.g., to modify or analyze the data of a Person object.

Therefore we can define ***a class as bundled data and the functions that work on that data***. From All the above example and explanation one can conclude the beauty of data abstraction is that we can treat complex data in a very simple way.

Points to remember:

- Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations.
- The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.
- ADT does not specify how data will be organized in memory and what algorithms will be used for implementing the operations
- Constructors are functions that build the abstract data type.
- Selectors are functions that retrieve information from the data type.
- Concrete data types or structures (CDT's) are direct implementations of a relatively simple concept.
- Abstract Data Types (ADT's) offer a high level view (and use) of a concept independent of its implementation.

Points to remember:

- A concrete data type is a data type whose representation is known and in abstract data type the representation of a data type is unknown
- Pair is a compound structure which is made up of list or Tuple
- List is constructed by placing expressions within square brackets separated by commas
- The elements of a list can be accessed in two ways. The first way is via multiple assignment and the second method is by the element selection operator
- Bundling two values together into one can be considered as a pair
- List does not allow to name the various parts of a multi-item object.

NK14H

Evaluation

Part - I

Choose the best answer **(1 Mark)**

1. Which of the following functions that build the abstract data type ?

   (A) Constructors (B) Destructors (C) recursive (D)Nested

2. Which of the following functions that retrieve information from the data type?

   (A) Constructors (B) Selectors (C) recursive (D)Nested

3. The data structure which is a mutable ordered sequence of elements is called

   (A) Built in (B) List (C) Tuple (D) Derived data

4. A sequence of immutable objects is called

   (A) Built in (B) List (C) Tuple (D) Derived data

5. The data type whose representation is known are called

   (A) Built in datatype (B) Derived datatype

   (C) Concrete datatype (D) Abstract datatype

6. The data type whose representation is unknown are called

   (A) Built in datatype (B) Derived datatype

   (C) Concrete datatype (D) Abstract datatype

7. Which of the following is a compound structure?

   (A) Pair (B) Triplet (C) single (D) quadrat

8. Bundling two values together into one can be considered as

(A) Pair (B) Triplet (C) single (D) quadrat

9. Which of the following allow to name the various parts of a multi-item object?

(A) Tuples (B) Lists (C) Classes (D) quadrats

10. Which of the following is constructed by placing expressions within square brackets?

(A) Tuples (B) Lists (C) Classes (D) quadrats

Part - II

Answer the following questions **(2 Marks)**

1. What is abstract data type?
2. Differentiate constructors and selectors.
3. What is a Pair? Give an example.
4. What is a List? Give an example.
5. What is a Tuple? Give an example.

Part - III

Answer the following questions **(3 Marks)**

1. Differentiate Concrete data type and abstract datatype.
2. Which strategy is used for program designing? Define that Strategy.
3. Identify Which of the following are constructors and selectors?

(a) N1=number() (b) accetnum(n1) (c) displaynum(n1)

(d) eval(a/b) (e) x,y= makeslope (m), makeslope(n)

(f) display()

4. What are the different ways to access the elements of a list. Give example.
5. Identify Which of the following are List, Tuple and class ?

(a) arr [1, 2, 34] (b) arr (1, 2, 34) (c) student [rno, name, mark]

(d) day= ('sun', 'mon', 'tue', 'wed') (e) x= [2, 5, 6.5, [5, 6], 8.2]

(f) employee [eno, ename, esal, eaddress]

Part - IV

Answer the following questions (5Marks)

1. How will you facilitate data abstraction. Explain it with suitable example
2. What is a List? Why List can be called as Pairs. Explain with suitable example
3. How will you access the multi-item. Explain with example.

Reference Books

1. *Data structure and algorithmic thinking with python by narasimha karumanchi*
2. *sign and analysis of algorithms by s sridhar*
3. *Data Structures and Algorithms in Python by Goodrich, Tamassia & Goldwasser*
4. *https://www.tutorialspoint.com*