Abstract Data Types Structures

Abstract data type (ADT)

We will learn about ADT but before understanding what ADT is let us consider different in-built data types that are provided to us. Data types such as int, float, double, long, etc. are considered to be in-built data types and we can perform basic operations with them such as addition, subtraction, division, multiplication, etc. Now there might be a situation when we need operations for our user-defined data type which have to be defined. These operations can be defined only as and when we require them. So, in order to simplify the process of solving problems, we can create data structures along with their operations, and such data structures that are not in-built are known as Abstract Data Type (ADT).

ADT (cont.)

Abstract Data type (ADT) is a type for objects whose behavior is defined by a set of values 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.

The user of data type does not need to know how that data type is implemented.

So a user only needs to know what a data type can do, but not how it will be implemented. Think of ADT as a black box which hides the inner structure and design of the data type.

Key Point

- Object-Oriented programming is based on the data and the functions that operate on it. Objects are instances of abstract data types that represent the data and its functions
- C++ allows the data and functions of an ADT to be defined together. It also enables an ADT to prevent access to internal implementation details, as well as to guarantee that an object is appropriately initialized when it is created.

Limitations of Procedural Programming

- If the data structures change, many functions must also be changed

- Programs that are based on complex function hierarchies are:
  - difficult to understand and maintain
  - difficult to modify and extend
  - easy to break

Structures

Structure

We have learned that by using an array, we only can declare one data type per array, and it is same for other data types. To use the same data type with different names, we need another declaration.

- *struct* data type overcomes this problem by declaring aggregate data types.

- A structure is a collection of related data items stored in one place and can be referenced by more than one names. Usually these data items are different basic data types. Therefore, the number of bytes required to store them may also vary.

- In order to use a structure, we must first declare a structure template. The variables in a structure are called structure elements or members.

- Structure is declared by the keyword (*struct*)

Declaration of structure

struct **name of structure** { members of structure
(different data items)
};

Ex:

```
struct first   {     int k;
                     float w;
                     char st[30];
                        };
```

Dont forget the semi colon

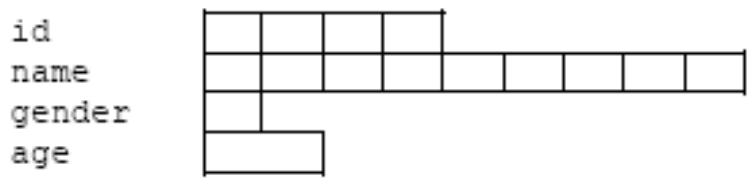
For example, to store and process a student's record with the elements: identification number, name, gender and age, the structure can be declared as follows :

```
struct  student {
                    char id_num[5];
                        char name[10];
                        char gender;
                        int age;    };
```

(members of structure)

Here, struct is a keyword that tells the compiler that a structure template is being declared and student is a tag name that identifies its data structure. It is not a variable; it is a label for the structure's template. Note that there is a semicolon after the closing curly brace.

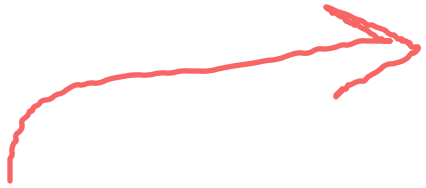
- A structure tags simply a label for the structure's template but you name the structure tag using the same rules for naming variables. The template for the structure can be illustrated as follows (note the different data size):
- size of structure student can be known using sizeof( ) function. (sizeof(student)) = 20 bytes



Compiler will not reserve memory for a structure until you declare an objects of the structure variable same as you would declare normal variables such as int or float.

- Creating objects of the structure variables can be done in any of the following three ways, (by referring to the previous example):

1.
```
struct  student{
        char id_num[5];
        char name[10];
        char gender;
        int age;
} studno_1, studno_2;
```



Global Defined

Or in programs we can declare objects of the struct like this:

2- main ( ){struct student studno_1, studno_2;……

Or directly:

3- student studno_1, studno_2;

In the above three cases, two structure variables, `studno_1` and `studno_2`, are declared. Each structure variable has 4 elements:

- In (1) , the structure variables are declared immediately after the closing brace in the structure declaration whereas in (2, 3) they are declared as `student`.

- The most widely used is no (3) where we put the declaration of the `struct` and use it anywhere in the program as follows:

```
struct student{……
              ……..…….};
main( )
    { student studno_1, studno_2;
```

- Where the `studno_1, studno_2` are variables declared as usual but the type here is **`struct`** `student` instead of integral type such as `int`, `char` and `float`.

Accessing The Structure Element

A structure element can be accessed and assigned a value by using the **structure variable name**, the **dot operator** (.) and the **element's name**. For example the following statement:

studno_1.name = " Mohammed";

Studno_1.age= 20;

- Assigns string "Mohamed" to the element name in the structure variable studno_1. The **dot operator** simply qualifies that name is an element of the structure variable studno_1. The other structure elements are referenced in a similar way.
- Unfortunately, we cannot assign string "Mohamed" (const char) directly to an array in the structure (char [ ]). For this reason, we have to use other methods such as

1- receiving the string from user input

2- Use strcpy( ) function, strcpy(studno_1.name,"Mohamed");

3- or by using pointers.

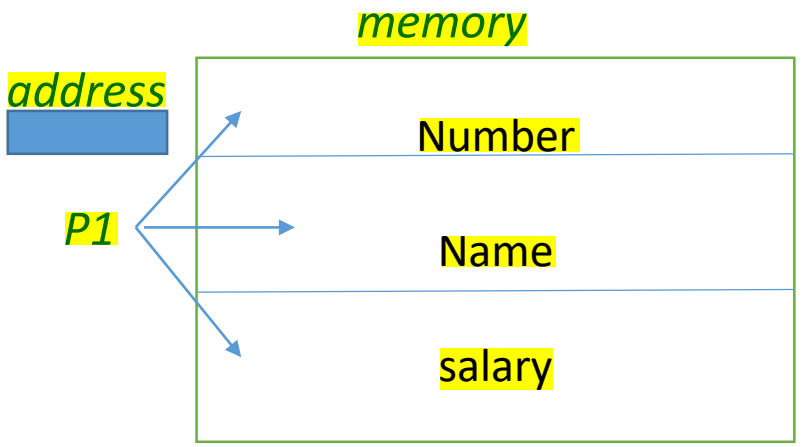
this method is valid only when instialization

هاام

all elements of struct are adjacent in memory

Initializing of structures

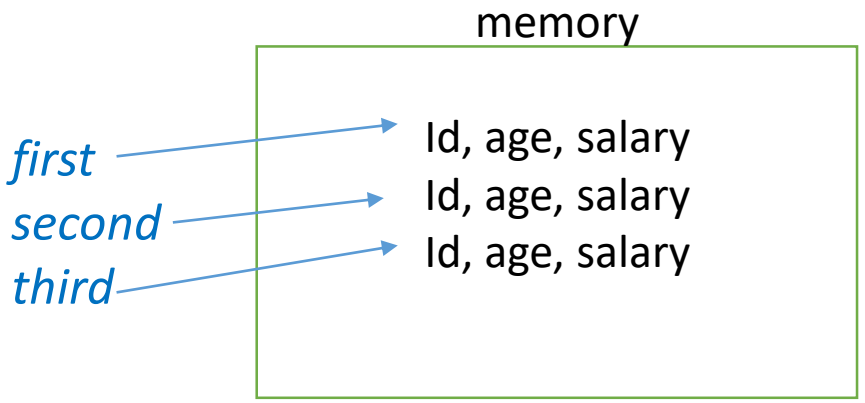
Ex:
```
struct person
        { int number;
          char name[30];
          float salary;};
main( )
        {   person p1;
            p1.number=10;
            strcpy(p1.name,"Youssife");
            p1.salary=4000;
            cout<<p1.number<<"   "<<p1.name<<"   "<<p1.salary;}
```



Output: 10 Youssife 4000

In the following example, we will use many methods to initialize structures:

Initialization of structure




```
struct Employee
{   short id;
   int age;
   double salary; };

 int main( )
{          Employee first = {  id: 2,    age: 49,     salary: 4000  };
      cout<<endl<<"   first Employ  "<<endl; cout << "ID:   " << first.id << "\n";
      cout << "Age:  " << first.age << "\n";
       cout << “salary: " << first.salary << "\n";
     Employee second= { 14, 32, 2000 };
 cout<<endl<<"   second  Employee  "<<endl; cout << "ID:   " << second..id << "\n";
   cout << "Age:  " << second.age << "\n";
   cout << “salary: " << second.salary << "\n";
    Employee third= {    .id = 1,     .age = 22,     .salary = 5000   };
   cout<<endl<<"   third  Employee  "<<endl; cout << "ID:   " << third.id << "\n";
   cout << "Age:  " << third.age << "\n";
   cout << “salary: " << third.salary << "\n";}
```

Output

first Employee

ID: 2
Age: 49
salary: 4000

second Employee

ID: 14
Age: 32
salary: 2000

third Employee

ID: 1
Age: 22
salary: 5000

Arrays in structure

We can declare an array as member of a structure. Example of *structure subjects* that contains year number, number of courses and name of courses as array of strings, creates three objects, reads the data of two objects and print their data.

```
// structure can be declared in the main function
 int main()
     { struct subjects
    {  int year ; int num_courses;   string name[5];
    // string name[num_courses];      error
                              };
     subjects p1, p2, p3;
      p1.year=1;
     cin>>p1.num_courses;
     for(int i=0; i<p1.num_courses ;i++)
                getline (cin,p1.name[i]);
     p2.year=2;
                cin>>p2.num_courses;

     for(int i=0; i<p2.num_courses; i++)
                getline (cin,p2.name[i]);
```

Cont. of the above example

```
cout<<" print the database  of first object "<<endl<<endl;
        cout<<  " year    "<<p1.year<<endl;
     cout<<"   courses of first object "<<endl<<endl;
              for(int i=0;i<p1.num_courses;i++)
               cout<<p1.name[i]<<endl;

 cout<<" print the database  of second object "<<endl<<endl;
         cout<<  " year    "<<p2.year<<endl;
     cout<<"   courses of second object "<<endl<<endl;
              for(int i=0;i<p2.num_courses;i++)
               cout<<p2.name[i]<<endl;
            // assign structure to another one
     p3=p2;   //correct

cout<<"  data base of assigned structure"<<endl<<endl;
             cout<<  " year    "<<p3.year<<endl;
     cout<<"   courses of third object "<<endl<<endl;
              for(int i=0;i<p3.num_courses;i++)
               cout<<p3.name[i]<<endl;
              //cout<<p1;     error
          return 0;  }
```

If the input:

5
Math_1, Electronics_1, Computer C++, Circuits_1, Physics
5
Math_2, Digital Design, Electronics_2, Circuits_2, Signals

output

print the database of first object
year 1
courses of first object
Math_1
Electronics-1
Computer C++
Circuits_1
Physics
print the database of second object
year 2
courses of second object
Math_2
Digital Design
Electronics_2
Circuits_2
Signals
data base of assigned structure
year 2
courses of third object
Math_2
Digital Design
Electronics_2
Circuits_2
Signals

Structure as member of another structure

A structure can contain another structure as one of its members

Ex: struct distance { int m, cm; } ;

```
struct room {  struct distance  d1,d2; //must be declared as shwon   };

int main()
        { room r1;      struct distance dist;
//              r1.m=10;   // error
                r1.d1.m=10;   r1.d1.cm=30;
                r1.d2.m=8;    r1.d2.cm=40;
                dist.m=12;     dist.cm=60;    //correct
                cout<<" dimension of room1    "<<r1.d1.m<<"    "<<r1.d1.cm<<endl;
                cout<<" dimension of room2    "<<r1.d2.m<<"    "<<r1.d2.cm<<endl;
                cout<<"   distance dist   "<< dist.m<<"    "<<dist.cm;}
```

Output

```
dimension  of room1    10     30
dimension  of room2    8     40
distance dist   12     60
```

Example:

Construct the structure distance that contains the dimension of any distance in kilometer, meter and centimeter. Write a main program that reads two distances and add them.

Solution:

Structure will have three members km, m and cm that will be declared as integers.

struct distance { int km, m, cm};

In the main( ), we will define three objects of the structure distance, the two required distances and the third one represents the sum of them.

The program will be in the next slide

```
int main( )
 {  struct distance {int km, m,cm;};

 distance d1,d2, d3;
  cin>>d1.km>>d1.m>>d1.cm;
  cin>>d2.km>>d2.m>>d2.cm;
  cout<<" first distance  "<<d1.km<<"    "<<d1.m<<"   "<<d1.cm<<endl;
  cout<<" second distance  "<<d2.km<<"    "<<d2.m<<"   "<<d2.cm<<endl;

  d3.km =d3.m=d3.cm=0;
  d3.cm=d1.cm+d2.cm;
  if(d3.cm>=100){d3.cm-=  100; d3.m+=1;}
  d3.m+=d1.m+d2.m;
  if(d3.m>=1000){d3.m-=1000; d3.km+=1;}
  d3.km+=d1.km+d2.km;
  cout <<" sum of two distances  "<<d3.km<<"    "<<d3.m<<"   "<<d3.cm<<endl;
  return 0;}
```

If the input will be given as follows:

Distance d1= 40, 600, 70
Distance d2 =30, 500, 66

Output

first distance 40 600 70
second distance 30 500 66
sum of two distances 71 101 36

Example

1- Construct a structure to create the data base of a student that contains:

Student number, name, number of subjects, scores of subjects, sum of all scores and average score.

2- Write a main program that reads the data of certain student, get his sum and average score and print the information of the student.

Solution

```
struct student
{int id, number;    char  name[20];    float score[6],  sum,  av;};

int main()
{   student p;        int i, j;
     cout<<“ student number”;                cin>> p.id;
    cout<<" enter name  "<<endl;
   gets(p.name);
cout<<" enter number of courses  "<<endl;
      cin>>p.number;
    cout<<" enter scores  "<<endl;
   for(j=0;j< p. number; j++)cin>>p.score[j];
    cout<<endl<<" print the database  "<<endl<<endl;
cout<<"  student number =  "<<p.id<<endl;
   cout<<"  number of courses =  "<<p.number<<endl;
   cout<<"  name   "<<p.name<<endl; p.sum=0;
         cout<<"  scores   "<<endl;
      for(j=0;j<p.number; j++){cout<<p.score[j]<<"   ";p.sum+=p.score[j];}
     cout<<endl<<endl;   cout<<"  the total score   "<< p.sum<<endl;
      p.av=p.sum/p.number;
      cout<<"  the Average score   "<< p.av;  return 0;  }
```

Giving the following input

102
Ahmed Abdel Rahman
5
90 80 70 70 80

Output

print the database
Student number = 102
number of courses = 5
name Ahmed Abdel Rahman
scores
90 80 90 70 80

the total score 410
the Average score 82

Take care when create array of struct that contain array

name [index of object] . array [index of array]

Array of structures

We can construct array of objects of certain structure with any data type as the following example:

1- Construct the structure student that contains: number of courses, name, array of scores, total score and average score.

2- Create array of 100 objects of structure student

3- Read the data of the students

4- Get the final score and average of each one

Solution

```
struct student
   {int n_course;
     char  name[30];
     float score[6],sum, av;};

    int main()
  {   student st[100];
   int i,j;
            for(i=0;i<100;i++)
   {       cout<<" enter name  "<<endl;
         gets(st[i].name);
        cout<<" enter number of courses  "<<endl;
         cin>>st[i].n_course;
         cout<<" enter scores  "<<endl;

  for(j=0;j<st[i].n_course;j++)cin>>st[i].score[j];}
```

Solution, cont.

```
cout<<endl<<" print the database of students"<<endl<<endl;
          for(i=0;i<100;i++)
    {cout<<endl<<" the database of student no. "<<i<<endl<<endl;
   st[i].sum=0;
        cout<<" the name  "<< st[i].name<<endl;
            cout<<  " the number of courses
"<<endl<<st[i].n_course<<endl;
          cout<<"  scores   "<<endl;
    for(j=0;j<st[i].n_course;j++){cout<<st[i].score[j]<<"   ";
            st[i].sum+=st[i].score[j];}cout<<endl<<endl;
     cout<<"  the total score   "<< st[i].sum<<endl;
            st[i].av=st[i].sum/st[i].n_course;
            cout<<"  the average score =  "<<st[i].av<<"%";}    }
```