CHAPTER 1 Structures and Pointers

Structure:- Structure is a user defined data type to store different data types, under a name.

Structure definition:- Syntax to define a structure is

```
struct structure_tag
    {
        data_type   variable1;
        data_type   variable2;
        ..............................
        ..............................
        data_type   variableN;
    };
```

Example

```
struct date
    {
      int day;
      char month[10];
      int year;
    };
```

Variable declaration and memory allocation

Variable declaration syntax is

```
struct structure_tag var1,var2,.....varN;
        OR
structure_tag var1,var2,.....varN;
```

Example

```
date dob,today;
```

The size of the variable dob will be 18 bytes since it contain 2 int data type which require 4 bytes each and a char array of size 10 which need 10 bytes.

A structure variable can also be declared along with the definition also.

Example

```
struct date
    {
      int day;
      char month[10];
      int year;
    }dob,today;
```

We can avoid structure tag if you are declaring variable along with definition, but it is not possible to declare any other variables.

Example

```
struct
    {
      int a,b,c;
```

}eq1,eq2;

Variable initialisation :- General format to initialise a variable is

structure_tag variable={value1, value2..... valueN};

Example

date dob={22,"March",2002};

If we do not provide values for all the elements, the given values will be assigned to the elements on First Come First Served basis. The remaining elements will be assigned with 0 or \0 depending on numeric or string.

A structure variable can be assigned with the value of another structure of the same type.

Example

date x=dob;

Accessing elements of structure:- We use dot operator (.) to access the elements of a structure. General format is

structure_variable.element_name

Example

today.day=10;

cin>>dob.year;

Nested structure :- A structure within another structure is called nested structure.

Example

```
struct date
  {
    short day,month,year;
  };
struct student
 {
   int adm_no;
   char name[20];
  date dt_adm;
  float fee;
 };
```

Comparison between Array and Structure

Arrays	Structures
It is a derived data type	It is a user defined data type
A collection of same type of data	A collection of different types of data
Elements of an array are referenced using the corresponding subscripts.	Elements of structure are referenced using dot operator(.)
When elements of an array becomes another array, multi-dimensional array is formed.	When elements of a structure becomes another structure, nested structure is formed.
Array of structure is possible	Structure can contain arrays as elements

Pointer is a variable that can hold the address of a memory location.
Syntax to declare pointer variable: data_type * variable;
Examples: float *ptr2;
int *ptr1;
struct student *ptr3;
The data type of a pointer should be the same as that of the data pointed to by it. In the above examples, ptr1 can contain the address of an integer location, ptr2 can point to a location containing floating point number, and ptr3 can hold the address of location that contains student type data.
The address of operator (&), is used to get the address of a variable. If num is an integer variable, its address can be stored in pointer ptr1 by the statement:
ptr1 = #
The indirection or dereference operator or value at operator (*) is used only with pointers and it retrieves the value pointed to by the pointer .
The statement cout<<*ptr1; is equivalent to the statement cout<<num;
Two types of memory allocation:
The memory allocation that takes place before the execution of the program is known as **static memory allocation**. It is due to the variable declaration statements in the program. There is another kind of memory allocation, called **dynamic memory allocation**, in which memory is allocated during the execution of the program. It is facilitated by an operator, named **new**. As complementary to this operator, C++ provides another operator, named **delete** to de-allocate (free) the memory.
Syntax for dynamic memory allocation:
pointer_variable = new data_type;
Examples:
ptr1 = new int; ptr2 = new float; ptr3 = new student;
Memory leak: If the memory allocated using new operator is not freed using delete, that memory is said to be an **orphaned memory block**. This memory block is allocated on each execution of the program and the size of the orphaned block is increased. Thus a part of the memory seems to disappear on every run of the program, and eventually the amount of memory consumed has an unfavourable effect. This situation is known as **memory leak**.
The following are the reasons for memory leak:

1. Forgetting to delete the memory that has been allocated dynamically (using new).
2. Failing to execute the delete statement due to poor logic of the program code.

Remedy for memory leak is to ensure that the memory allocated through new is properly de-allocated through delete.
Operations on Pointers
The following statements illustrate various operations on pointers:

```
int *ptr1, *ptr2;     // Declaration of two integer pointers
ptr1 = new int(5);    /*Dynamic memory allocation (let theaddress be 1000)and
                      initialisation with 5*/
ptr2 = ptr1 + 1;      /*ptr2 will point to the very next
                      integer location with the address 1004 */
++ptr2;               //Same as ptr2 = ptr2 + 1
cout<< ptr1;          //Displays 1000
cout<< *ptr1;         //Displays 5
cout<< ptr2;          //Displays 1004
cin>> *ptr2;          /*Reads an integer (say 12) and
                      stores it in location 1004 */
```

```
cout<< *ptr1 + 1;  //Displays 6 (5 + 1)
cout<< *(ptr1 + 2);        //Displays 12, the value at 1004
ptr1--;                    //Same as ptr1 = ptr1 – 1
```

Only increment (++ or +) and decrement (-- or -) operators are used with pointers.

Dynamic array is a collection of memory locations created during run time using the dynamic memory allocation operator new . The syntax is: pointer = new data_type[size];
Here, the size can be a constant, a variable or an integer expression.
C++ treats the name of array as a pointer. The name of array is a pointer pointing to the first element of the array.

Strings can be referenced using character pointer.

Eg:

```
char *str;
str = “hello”;
cout << str;
```

The statement: char *week[7]={"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};
declares an array of 7 strings. These elements can be displayed as follows:

```
for (i=0; i<7; i++)
  cout<<name[i];
```

Advantages of character pointer

- Strings can be managed by optimal memory space.
- Assignment operator can be used to copy strings.
- No wastage of memory.

The syntax for accessing the elements of a structure using pointer is as follows:

structure_pointer->element_name;

Examples:

```
struct employee
   {
        int ecode;
        char ename[15];
        float salary;
    } *eptr;
eptr->ecode = 657346;          //Assigns an employee code
cin.getline(eptr->ename,15);   //inputs the name of an employee
cin>> eptr->salary;            //inputs the salary of an employee
cout<< eptr->salary * 0.12;    //Displays 12% of the salary
```

<u>Self referential structure</u> is a structure in which one of the elements is a pointer to the same structure.

Example:

```
        struct employee
          {
                int ecode;
                char ename[15];
                float salary;
                employee *ep;
          };
```

The element ep is a pointer of employee data type