Chapter 1
Structures and Pointers

Structure is a user-defined data type of C++ to represent a collection of logically related data items, which may be of different types, under a common name.

Syntax to define the structure:

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

Example:

```
struct student
{
    int adm_no;
    char name[20];
    char group[10];
    float fee;
};
```

Variables should be declared for storing the details. Variables can be initialised also. For example, the details of a student can be stored in a variable during its declaration itself as shown below:

```
student st = {3452, "Vaishakh", "Science", 270.00};
```

Memory space for the variable **`st`** = 38 Bytes (4 + 20 + 10 + 4)

Structure tag may be omitted while defining a structure in case variable is declared as in the following example:

```
struct
{
    int adm_no;
    char name[20];
    char group[10];
    float fee;
} st;
```

The dot operator (**.**) connects a structure variable and its element using the following syntax:

```
structure_variable.element_name
```

Examples for accessing elements of a structure :

```
cin >> st.adm_no;        cin.getline(st.name,20);
st.fee = 2500;           cout << st.group;
```

If two different structures contain same elements, there variables cannot be assigned.
Consider two structures `test_1` and `test_2`.

```
struct test_1
{
    int a;
    float b;
}t1={3, 2.5};
```

```
struct test_2
{
    int a;
    float b;
}t2;
```

The assignment `t2 = t1;` is invalid. But `t2.a=t1.a;` and `t2.b=t1.b;` are valid.

If an element of a structure is a variable of another structure, it is called ***nested structure***. The following is an example:

HSSLiVE.IN

Definition A	Definition B

<pre>struct date
{
    short day;
    short month;
    short year;
};
struct student
{
    int adm_no;
    char name[20];
    date dt_adm;
    float fee;
};</pre>
	<pre>struct student
	{
	    int adm_no;
	    char name[20];
	    struct date
	    {
	        short day;
	        short month;
	        short year;
	    } dt_adm;
	    float fee;
	};</pre>

Table 1.2: Two styles of nesting

Elements of nested structure are accessed as follows:

```
student st;
cin >> st.dt_adm.day
    >> st.dt_adm.month
    >> st.dt_adm.year;
```

Array V/s Structure:

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

Table 1.3: Comparison between arrays and structures

^^^

<u>Pointer</u> is a variable that can hold the address of a memory location.

Syntax to declare pointer variable: `data_type * variable;`

Examples: `int *ptr1;` `float *ptr2;` `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 <u>used only with pointers</u> and it retrieves the value pointed to by the pointer.

The statement `cout<<*ptr1;` is equivalent to the statement `cout<<num;`

<u>Two types of memory allocation:</u>

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 unfavorable effect. This situation is known as ***memory leak***.

The following are the reasons for memory leak:

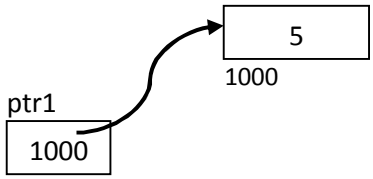
- Forgetting to delete the memory that has been allocated dynamically (using new).
- Failing to execute the delete statement due to poor logic of the program code.

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 the
                          address 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
```



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.

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];
```

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
```

Self referential structure 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

Questions from Previous Years' Question Papers

1. Represent a structure named student with attributes of different types and give advantages of using structure. (3) (March 2016)
2. Structure within a structure is termed as _____. (1) (March 2016)
3. Orphaned memory blocks are undesirable. How can they be avoided? (2) (March 2016)
4. Discuss problems created by memory leak. (2) (March 2016)
5. (a) How will you free the allocated memory? (1) (SAY 2016)
   (b) Define a structure called time to group the hours, minutes and seconds. Also write a statement that declares two variables current_time and next_time which are of type struct time. (2) (SAY 2016)
6. Write a program to store and print information (name, roll and marks) of a student using structure. (3) (SAY 2016)
7. Write a program in C++ to input the total marks obtained by a group of students in a class and display them in descending order using pointers. (3) (SAY 2016)
8. Compare the aspects of arrays and structures. (3) (March 2017)
9. Run time allocation of memory is triggered by the operator ____. (3) (March 2017)
10. Represent the names of 12 months as an array of strings. (2) (March 2017)
11. A structure can contain another structure. Discuss. (2) (March 2017)
12. If 'ptr' is a pointer to the variable 'num', which of the following statements is correct?
    (i) 'ptr' & 'num' may be of different data types.
    (ii) If 'ptr' points to 'num', then 'num' also points to 'ptr'.
    (iii) The statement num=&ptr; is valid.
    (iv) *ptr will give the value of the variable 'num'. (1) (SAY 2017)

13. Predict the output of the following program:

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int x=100, y=20;
    void swap(int *x, int *y);
    cout<< "values before swap" <<endl;
    cout<< "x = "<<x<< ", y = "<<y<< "\n";
    x = x+15; y = y+25;
    swap(&x, &y);
    cout<< "values after swap" <<endl;
    cout<< "x = "<<x<< ", y = "<<y<< "\n";
    getch();
}
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

(3) (SAY 2017)

14. Predict the output:

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int x[]={15, 25, 28, 67, 40};
    int *p = x, k, kk;
    cout<< "values of p=" <<*p<<endl;
    k = *p + (*p+5);
    cout<< "k=" <<k<< "\n";
    kk= *x+5;
    cout << "kk=" <<kk<< "\n";
    getch();
}
```

(3) (SAY 2017)

15. State any two differences between static and dynamic memory allocation.

(2) (SAY 2017)