

Chapter-1

Structures and Pointers

Structure

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.

Structure Definition

To define a structure you use the struct statement. The format of structure statement is

<pre>struct structure_tag { data_type variable1; data_type variable2; ; data_type variableN; };</pre>	<pre>struct student { int adm_no; char name[20]; char group[10]; float fee; };</pre>
--	--

In the above syntax struct is the keyword to define a structure. `structure_tag` (or `structure_name`) is an identifier and `variable1`, `variable2`, `variableN` are identifiers to represent the data items.

Structure tag may be omitted while defining a structure in case variable is declared

The example shown below

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

Variable declaration and memory allocation

A variable is required to refer to a group of data. The variable is declared using the following syntax

struct structure_tag var1, var2,, varN;

or

structure_tag var1, var2,, varN;

Here **structure_tag** is the name of the structure and **var1, var2,, varN** are the structure variable

Example

date dob, today or struct date dob, today;

Here **date** is the structure_tag and **dob, today** is the var1, var2

Variable initialisation

- Variables should be declared for storing the details.
- Variables can be initialised also.
- The variable can be initialised as follows

structure_tag variable(value1, value2,, valueN);

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};
```

Here the values will be assigned to the elements of structure variable st in the order of their position in the definition.

The above statements allocate 38 bytes of memory space for the variable st = 38 Bytes (4 + 20 + 10 + 4)

Note:

If we not provide values for all the elements, the given values will be assigned to the elements on FCFS(First Come First Served) basis. The remaining elements will be assigned with 0(Zero) or '\0'(Null Character) depending upon numeric or string.

Accessing elements of a structure

The elements of a structure are accessed using the dot (.) operator or Period operator.

The syntax for accessing elements is,

[Object name][Operator][Member variable Name]

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;
st.fee = 2500;
```

```
cin.getline(st.name,20);
cout << st.group;
```

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

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

The elements of both the structure are the same in number, name and type. The structure variable t1 of type test_1 is initialised with 3 and 2.5 for a and b.

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

Example:

```
#include<iostream>
using namespace std;
struct student
{
    int rollno;
    int m1,m2,m3;
};
int main()
{
    student s; //Object declared
    cout<<"Enter the marks in Three subjects";
    cin>>s.m1>>s.m2>>s.m3;
    cout<<"Total="<<s.m1+s.m2+s.m3;
    return(0);
}
```

Nested structure

If an element of a structure is a variable of another structure, it is called nested structure. This concept is used for building of powerful data structures.

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;
```

Difference between Array and Structure

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

Pointer

Pointer is a variable that can hold the address of a memory location. The data stored in computer occupies a memory cell. Each cell has a unique address. A pointer points to the address of memory location.

We can declare a pointer, similar to a variable.

The syntax is

```
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 Operators & and *

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 = &num;
```

This statement on execution it establishes a link between two memory location.

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;**

Note:

The address of (&) operator and indirection(*) are unary operators.

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;

The syntax is

delete pointer_variable;

Examples:

ptr1 = new int;

ptr2 = new float;

ptr3 = new student;

```
#include<iostream>
using namespace std;
int main()
{
    int i,*p;
    p=new int[10];
    cout<<"Enter the elements";
    for(i=0;i<10;i++)
    {
        cin>>p[i];
    }
    for(i=0;i<10;i++)
    {
        cout<<p[i]<<"\t";
    }
    delete[p];
    return 0;
}
```

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.
- Assigning the address returned by new operator to a pointer that was already pointing to an allocated object

Remedy for memory leak is to ensure that the memory allocated through **new** is properly de-allocated through **delete** .

Note:

In static memory allocation the operating system takes the responsibility of allocation and deallocation without user's instruction. So there is no chance of memory leak.

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

It 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.

For example:

`P=new int[10];` //declares a dynamic array of size 10.

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

```

Pointers and array

There is a close relationship between pointers and array. C++ treats the name of array as a pointer. The name of array is a pointer pointing to the first element of the array.

For example:

```

char ch[10],* ptr;
ptr=ch;

```

Here `ptr` points to the address of first array element in `ch`. To access fifth element you can use `h[4]` ;
or `*(ptr+4)`;

Note: Currently `ptr` points to address of first element in the array `ch`.

Operations on pointers

Arithmetic operations can be performed on a pointer.

For Example;

```
int a[50];
int *ptr;
ptr=&a[0] ; //ptr refers to the base address of array a.
```

ptr - - or - - ptr

Moves the pointer to the previous address.(If the base address was 1000 the operation – ptr moves the pointer to 998 memory location ie,1000 – 2.

```
cout<<ptr;           //Displays 1000 ie, the address of a[0](Base address);
cout<<*ptr;          //Displays the value at a[0];
```

Pointer and String

A string is an array of characters, and array name can be considered as a string variable. Every string in C++ is terminated by a Null character(\0). For example the string hello is represented in memory as

```
h e l l o \0
```

Intializing an array

The above array can be initialized as char ch[]="hello";

or

```
char ch={'h','e','l','l','o','\0'};
```

We can use a character pointer to reference the array.

```
char s[10]; //character array declaration
char *ptr; //character pointer declaration;
cin>>s;
```

```
ptr=s; //Copying contents of s to ptr
```

Example:

```
int main()
{
```

```
    char a[10];
    strcpy(a, "hello"); //copies the
```

string hello to a

```
    cout << a; //Output is hello
    return(0);
```

```
}
```

Advantages of character pointer

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

Self referential structure

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
};
```

Note:

Only equality(=) and not equality(!=) operators can be applied to a pointer

Program to find average marks of students

```
#include<iostream>
using namespace std;
int main( )
{
    int n,sum=0,*mark_ptr,i;
    float avg;
    mark_ptr=new int;
    cout<<"Enter the No of Students";
    cin>>n;
    for(i=0;i<n;i++)
    {
        cin>>*(mark_ptr + i);
        sum=sum+*(mark_ptr+i);
    }
    avg=(float)sum/n;
    cout<<"Average="<<avg;
    return 0;
}
```

Questions

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. State any two differences between static and dynamic memory allocation. (2) (SAY 2017)
14. Identify the correct errors in the following code fragment:


```
Struct
{
    int regno;
    char name[20];
    float mark = 100;
};
```

 (2) (March 2018)
15. What is the difference between static and dynamic memory allocation? (2) (March 2018)
16. Read the following code fragment:


```
int a[] = {5, 10, 15, 20, 25};
int *p = a;
```

 Predict the output of the following statements:


```
cout << *p;
cout << *p + 1;
cout << *(p + 1);
```

 (3) (March 2018)
17. What is self referential structure? (2) (SAY 2018)
18. What is the difference between the two declaration statements given below?


```
int *ptr = new int (10);
int *ptr = new int [10];
```

 (2) (SAY 2018)
19. What is a pointer in C++? Declare a pointer and initialize with the name of your country. (3) (SAY 2018)
20. Define a structure named 'Time' with elements hour, minute and second. (2) (March 2019)
21. Read the following C++ code:


```
int a[5] = {10, 15, 20, 25, 30};
int *p = a;
```

 Write the output of the following statements:


```
cout << *(p + 2);
cout << *p + 3;
```

 (2) (March 2019)

22. What is the different memory allocations used in C++? Explain. (3) (March 2019)
23. Consider the given structure definition:
- ```
struct complex
{
 int real;
 int imag;
};
```
- (a) Write a C++ statement to create a structure variable.
- (b) Write a C++ statement to store the value 15 to the structure member real. (2) (SAY 2019)
24. Write the use of \* and & operators used in pointer. (2) (SAY 2019)
25. Distinguish between Array and Structure. (3) (SAY 2019)
26. The \_\_\_\_\_ operator is used to allocate memory location during run time (execution). (1)(March 2020)
27. What is a pointer variable in C++ ? Write the syntax or example to declare a pointer variable. (2)(March 2020)
28. Write any two differences in static and dynamic memory allocation. (2)(March 2020)
29. Define structure. Write any two differences between structure and array. (3)(March 2020)

\*\*\*\*\*