Q.7 Write a C++ program having two function names **area** and **perimeter** to find the area and perimeter of a square.

Q.8 Write a C++ program to read a number from the keyboard and then pass it to a function to determine whether it is prime or composite.

Q.10 Write a C++ program to get an integer number from keyboard in *main* program and pass it as an argument to a function where it calculate and display the table.

Q.11 Define function and differentiate between built-in and user-defined functions with the help of examples.

Q.12 How function prototype and declarator differ from each other? Explain with the help of examples.

Q.13 Define default arguments. Describe the advantages and disadvantages of default argument.

Q.14 What is meant by the term function overloading? How a function can be overloaded in C++? Describe it with the help of an example program.

Q.15 Describe the role of the following in function overloading.

- Data types of arguments
- Number of arguments
- Order of arguments

UNIT 7

POINTERS

After the completion of Unit-7, Students will be able to:

- Define pointers
- Understand memory addresses
- Know the use of reference operator &
- Know the use of dereference operator *
- Declare variables of pointer types
- Initialize the pointers

INTRODUCTION

When variables are declared, computer reserves space in its memory for some objects. The variable name refers to that memory space that is occupied by that object. This allows us to store the value of those variables in the space reserved. Computer refers to these spaces using their addresses. To refer to memory addresses, C++ provides the facility of pointers. The objective of this Unit is to understand how variables are stored in computer memory and how spaces are allocated to them. Sometimes, a programmer needs to refer to another memory location which can only be done by the use of **pointers**. *Pointer* is one of the most *powerful* tool to handle memory addresses. This Unit covers the concept of memory addresses in detail and explains the working of reference operator & and dereference operator * with the help of examples. It also focuses on how to declare pointer type variables and how to initialize them in a program.

7.1 Pointers

Pointer is an important and powerful feature of C++ language, in which a variable points to another variable by holding its address.

7.1.1 Pointer

A **pointer** is a variable that holds the address of another variable. In memory, each and every variable has an address assigned to it by the compiler and if a programmer wants to access that address, another variable called "pointer" is needed. For the declaration of pointer, an *asterisk* * symbol is used between the data type and the variable name.

Consider the following pointers declaration of the integer variable **marks**, floating point variable **percentage** and character type variable **name**:

```
int * marks;
float * percentage;
char * name;
```

7.1.2 Memory addresses

When writing a program, variables need to be declared. Declaration of variable is simple and its declaration tells the computer to reserve space in memory for this variable. The name of the variable refers to that memory space. This task is automatically performed by the operating system during runtime. When variables are declared, programmers store data in these variables. Computer refers to that space using an *address*. Therefore, everything a programmer declares has an address. It is analogous to the home address of some person. Using pointer variables, one can easily find out the address of a particular variable.

7.1.3 Reference operator &

As pointers are the variables which hold the addresses of other variables, therefore, while assigning addresses to them, a programmer needs a special type of operator called **address-of operator** that is denoted by ampersand **&** symbol. This provides address of a memory location.

To understand it, consider the following segment of code.

```
loat x = 5.5;
loat *fPointer;
Pointer;= &x; // assign address of x to fPointer;
```

Conceptually, the above lines of code can be pictorially represented as:

Figure 7.1: Use of the Pointer Variable

this pictorial representation, the variable x has a *float* value 5.5 that is stored at location 000. The pointer variable *fPointer* points to the variable x by holding its address 1000 as value. In this case, if we want to print the value of the variable x it will display 5.5 but if e print the value of *fPointer* it will display 1000.

onsider the following program:

```
* use of pointer in a program */
include <iostream.h>
include<conio.h>
t main()
```

```
{
float x = 5.5;
float *fPointer;
fPointer= &x;
cout << "The address of x= "<<&x << endl;
cout<< "The value of x= "<<x<<endl;
cout << "The value of fPointer = "<< fPointer;
getch();
return 0;
}
```

Output of the Program

The address of x= 1000
The value of x= 5.5
The value of fPointer =1000

This program gives some other output (address) on another computer depending upon the availability of the memory.

7.1.4 Dereference operator *

Pointers are used to store the address of another variable. If we want to store the value of the variable through the pointer, then we need a special type of operator called **dereference operator** denoted by **asterisk** *. To use dereference operator, precede the pointer variable with **asterisk** * symbol. This operator can be read as "value pointed by".

Consider the following program to demonstrate the use of **dereference operator** *.


```
/* use of dereference operator (*) in a program */
#include <iostream.h>
#include<conio.h>
int main()
{
int n = 100;
int *Pn; //defines a pointer to n
Pn=&n; //Pn stores the address of n
int valueN;
valueN=*Pn;
cout << "The address of n= "<<&n << endl;
cout<< "The value of n= "<<n<<endl;
cout << "The value of Pn = "<< Pn<<endl;
cout << "The value of *Pn = "<< (*Pn)<<endl;
cout << "The value of valueN = "<< valueN;
getch();
return 0;
}
```

Output of the Program

The address of n= 0x8fc5fff4
The value of n= 100
The value of Pn = 0x8fc5fff4
The value of (*Pn) = 100
The value of valueN= 100

In this example, the instructions at lines 10 and 14 make use of the **dereference operator** * and thus access the actual values of the original variable n which is pointed out by the pointer Pn. In pointers, the ampersand operator & is the **reference** operator

and can be read as "address of" and * is the **dereference** operator that can be read as "value pointed by". These operators are complementary of each other and have opposite meanings. A variable referenced with & can be dereferenced with *.

7.1.5 Declaring variables of pointer types

The declaration of pointer is simple and is similar to the declaration of a regular variable with a minor difference of the use of an *asterisk* * symbol between the data type and the variable name. Consider the following general format:

Data type * nameVariable;

Here, *data type* is the type of the value of that variable to which this pointer will point. This *date type* is not the type of the pointer itself but the type of the data the pointer points to. To understand it, consider the following examples of pointers declaration.

*int** totalMarks;
char *Name;
*float** percentage;

In this example, three pointers *totalMarks*, *Name* and *percentage* are declared. Each one is intended to point to a different data type. All these pointers occupy space in memory. The first pointer points to an **int**, the second to a **char** and the last one to a **float**.
It is notable that the asterisk * symbol used between the data type and the name of the variable is the indication for the pointer declaration and is not used for multiplication. There are three ways to place the asterisk. These are: placing next to the data type, the variable name, or in the middle. All these variations are shown in the above declarations of variables *totalMarks*, *Name* and *percentage*.

A pointer of type *int* can only points to a variable of type *int* and cannot to some other type of variable. Same is the case for other data types as well, but, there is a special type of pointer named **void pointer** or generic pointer that can point to an objects of any data type. Its declaration is similar to the declaration of normal pointers with the only difference of the use of *void* keyword as the pointer's type. Consider the following statement of declaration of the *void* pointer:

void *pointerVoid; // pointerVoid is a void pointer

As, a *void* pointer can point to objects of any data type, therefore, consider the following statements.

```
int X;
float Y;
void *pointerVoid;
pointerVoid = &X; // valid
pointerVoid = &Y; // valid
```

In this segment of code, *pointerVoid*, stores the address of both *integer* variable X and *floating point* variable Y. The compiler decides at run time that the address of which type of variable should be assigned to this pointer.

7.1.6 Pointer initialization

Assigning values to pointers at declaration time is called pointer initialization. As we know that the values of pointers are the addresses of other variables, therefore, sometimes when we declare pointers we may want to explicitly specify to which variables they will point.

Consider the following segment of code to understand the concept of pointer initialization:

***float* Temperature;**
***float* *PTemperature = &Temperature;**

Here, *PTemperature* is a pointer variable to a floating point variable. As this pointer is created with the statement '*float *PTemperature*', immediately the address of a float variable '*Temperature*' is assigned to it. The behavior of the above code is being equivalent to the following code.

***float* Temperature;**
***float* *PTemperature;**
PTemperature = &Temperature;

It should be considered that at the moment of declaring a pointer, the asterisk * indicates that it is a pointer variable and not the **dereference operator.**

Consider the following program to explain the concept of pointer initialization.

```
/* pointer initialization program */
#include <iostream.h>
#include<conio.h>
int main()
{
float Temperature;
float *PTemperature = &Temperature;
cout << "The address of Temperature is = "
<<&Temperature << endl;
cout<< "The value of *PTemperature is = "
<< PTemperature <<endl;
getch();
```



```
return 0;
}
```

Output of the Program

The address of Temperature is = 0x8f98fff2
The value of PTemperature is = 0x8f98fff2

Here, the pointer *PTemperature* is initialized with the address of the variable *Temperature*.

Sometimes we need to initialize a pointer to zero. Such pointers are called null pointers and they do not point to anything. Null pointers can be defined by assigning address 0 to them.

Consider the following initialization to demonstrate null pointer:

```
int *NPtr; //defines Null pointer
NPtr = 0; // this assigns address 0 to NPtr
```

The use of NULL pointers is mostly done in dynamic memory allocation.

Summary

- A pointer is a variable that points to or refers to another variable. That is, if we have a pointer variable of type "pointer to int" it might point to the int variable.
- Some tasks in C++ are easier to do with pointers, such as accessing the addresses of variables indirectly and operating their values.
- Like normal variables, Pointer variables are also declared in programs before using them. To declare a pointer variable, the data type of the variable is followed by * symbol.
- While using pointers in C++ programs, address-of-operator denoted by ampersand & symbol is used to assign the address of variables to the pointer variable.
- In C++ program, if we want to use the value of the variable to which pointer points a special type of operator called dereference operator, denoted by asterisk *; is used with pointers.
- In C++ programs, pointers are initialized to addresses of other variables like the initialization of ordinary variables.

Exercise

Q.1 Fill in the Blanks.

i. Pointers are special type of variable that store _____ of other variables.
ii. If we have a statement int n; and we want to define a pointer to n then its data type must be of the type _____.
iii. Assigning values to a pointer during its declaration time is called _____.
iv. & is the _____ operator.
v. * is the _____ operator which can be read as "value pointed by" operator.

Q.2 Select the correct choice for the following Multiple Choice Questions.

i. If we have the statement int *Ptr; then to what Ptr point?
   a. Points to an integer type variable
   b. Points to a character type variable
   c. Points to a floating point type variable
   d. None of above
ii. A pointer is:
   a. A keyword used to create variables
   b. A variable that stores address of an instruction
   c. A variable that stores address of another variable
   d. All of the above
iii. The operator used to get value at address stored in a pointer variable is
   a. *
   b. &
   c. &&
   d. ||
iv. Which of the statements is correct about the following segment code?

```
int i=10;
int *j=&i;
```

   a. j and i are pointers to an int
   b. i is a pointer to an int and stores address of j
   c. j is a pointer to an int and stores address of i
   d. j is a pointer to a pointer to an int and stores address of i