Lesson 7: Pointers

7.1. Introduction

In our previous lesson, we discussed arrays as secondary data types. In this lesson we will discuss another secondary data type called pointer. You recall that a variable is a named memory location. Every memory location has its address defined. Memory address is accessed using ampersand (&) operator which denotes an address in memory. The & (ampersand) sign is read as "the address of". A pointer is a variable whose value is the address of another variable i.e. it is a direct address of the memory location. Referencing a value through a pointer is called indirection.

7.2. Lesson objectives

By the end of this lesson, the learner will be expected to:

- Define a pointer
- Declare and use a pointer
- Use a pointer in a program

7.3. Lesson outline

This lesson is organized as follows.

7.4. Accessing variable address

Pointers are variables that contain memory address as their values. Usually a variable will contain a specific value. However, a pointer will contain an address of another variable which in turn stores a specific value. Variable address is a hexadecimal value that can be accessed using ampersand (&) operator. See the example below.

Example: A program in c++ to display address of a variable.

```
#include <iostream>
 using namespace std;
void  main()
    {
 int x; //integer variable
char y[5]; //array of characters
cout << "The address of variable x= " << &x << endl;
cout<<"The address of variable y= "<<&y<<endl;
    }
```

Study the program above, then compile and run it to observe the output.

7.5. Definition of a pointer

A pointer is defined as a variable that contain an address of another variable. But since a variable has an address and stores a specific value, we can say a variable name directly references a value. Since a pointer stores an address of a variable which directly references a value we can say that a pointer indirectly references a value. Referencing a value through a pointer is called **indirection**.

7.6. Pointer declaration

Like any variable or constant, a pointer must be declared before it can be used. The general form of a pointer variable declaration is:

```
type *var_name;
```

Type is the pointer's base type and must be a valid data type.

Var_name is the identifier of the pointer variable.

The asterisk (*) must be used to declare a pointer- to designate a variable as a pointer.

Valid pointer declarations:

```
int    *p1;      //pointer to an integer
double *p2;      // pointer to a double
float  *p3;      // pointer to a float
char   *p4       // pointer to a character
```

Remark:

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

7.7. Using pointers

The following are some of the important operations we can do with the help of pointers:

- Assigning the address of a variable to a pointer**.**
- Accessing the value at the address available in the pointer variable.
- Creating dynamic data structures.

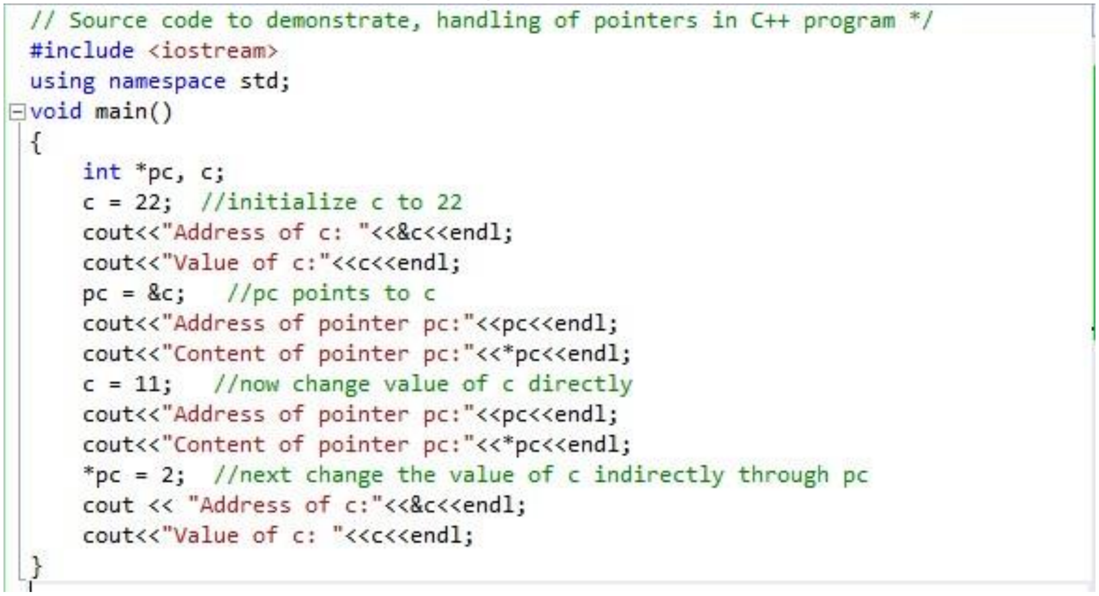
- Passing and handling variable parameters passed to functions.

7.8. Pointer operators

Operator name	Symbol	Meaning
Address operator	&	Returns an address of an operand
Indirection operator or deferencing operator	*	Returns the value of the object to which its operand (i.e. a pointer) points.

7.9. Working with pointers

Example: Demonstrate Working of Pointers



```
// Source code to demonstrate, handling of pointers in C++ program */
#include <iostream>
using namespace std;
void main()
{
    int *pc, c;
    c = 22;  //initialize c to 22
    cout<<"Address of c: "<<&c<<endl;
    cout<<"Value of c:"<<c<<endl;
    pc = &c;   //pc points to c
    cout<<"Address of pointer pc:"<<pc<<endl;
    cout<<"Content of pointer pc:"<<*pc<<endl;
    c = 11;   //now change value of c directly
    cout<<"Address of pointer pc:"<<pc<<endl;
    cout<<"Content of pointer pc:"<<*pc<<endl;
    *pc = 2;  //next change the value of c indirectly through pc
    cout << "Address of c:"<<&c<<endl;
    cout<<"Value of c: "<<c<<endl;
}
```

Study this program, then compile, build and run to see the results.

Explanation of program and figure

[i]. Code int *pc, p; creates a pointer *pc* and a variable *c*. Pointer *pc* points to some address and that address has garbage value. Similarly, variable *c* also has garbage value at this point.

[ii]. Code c=22; makes the value of c equal to 22, i.e., 22 is stored in the memory location of variable *c*.

[iii]. Code pc=&c; makes pointer pc, point to address of c. Note that, &c is the address of variable *c* (because *c* is normal variable) and *pc* is the address of *pc* (because pc is the pointer variable). Since the address of pc and address of c is same, *pc (value of pointer pc) will be equal to the value of *c*.

[iv]. Code c=11; makes the value of *c*, 11. Since, pointer *pc* is pointing to address of *c*. Value of **pc* will also be 11.

[v]. Code *pc=2; change the address pointed by pointer *pc* to change to 2. Since, address of pointer *pc* is same as address of *c*, value of *c* also changes to 2.

Displaying value in address pointed

To access a value at the address available in the pointer variable use unary operator often referred to as indirection operator (***).** This returns the value of the variable located at the address specified by its operand. The asterisk (*) dereferences the pointer and is read as "the memory location pointed to by," unless in a declaration, as in the line int *p.

Example:

```
// Source code to display value in address pointed */
#include<iostream>
using namespace std;
 void main()
{
 int x = 3; // declare and initialize x
int *p;// declare pointer variable
p = &x; //assign address of the variable to pointer
variable
 cout <<"address of x = "<< &x << endl;
cout << "address stored in p = " << p << endl;
cout << "the value of *p = "<< *p << endl;

}
```

Study this program carefully, then compile, build and run to observe the output.

Common errors when working with pointers

Suppose, the programmer want pointer p to point to the address of num. Then,

```
int num, *p;  //Declaration of pointer and num is correct p=num;
//p is address whereas, num is not an address. Error.
*p=&num;     //&num is address whereas, *p is not an address.Error.
```

NULL Pointers

It is always a good practice to assign a NULL value to a pointer variable in case you do not have exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.

The NULL pointer is a constant with a value of zero defined in several standard libraries.

Example

```
#include <iostream>
 using namespace std;
```

```
int main()
{
 int  *p = NULL;
cout<<"The value of p is : "<<p<<endl;
return 0;
}
```

Study the above program carefully, then compile, build and run to observe the output. On most of the operating systems, programs are not permitted to access memory at address 0 because that memory is reserved by the operating system. However, the memory address 0 has special significance; it signals that the pointer is not intended to point to an accessible memory location. But by convention, if a pointer contains the null (zero) value, it is assumed to point to nothing.

7.10. Revision questions

[a]. Explain any **TWO** applications of pointers in a c++ program.
[b]. Explain the syntax for pointer declaration.
[c]. Write a simple program in c++ to demonstrate use of pointers in C++.

7.11. Summary

In this lesson we have learnt that pointers are variables that point to an area in memory. Pointers are defined by adding an asterisk (*) in front of the variable name (i.e. int *p). To get the address of any variable by add an ampersand (&) in front of variable identifier, i.e. p = &num. The asterisk, unless in a declaration (such as int *p), should be read as "the memory location pointed to by." The ampersand, unless in a declaration (such as int &num), should be read as "the address of." We have noted that we can allocate memory using the **new** keyword.

7.12. Suggested reading

[1]. Object oriented programming with C++ by E Balagurusamy 3rd ed; publisher: Tata Mcraw Hill
[2]. Sams teach yourself c++ in 24 hours by Jesse Liberty and Rogers Cadenhead.
[3]. Object oriented programming in c++ by Joyce Farrel
[4]. Object-oriented programming with c++ by Sourav Sahay.