

LECTURE 18

1. Pointers:

The pointer is a powerful technique to access the data by indirect reference as it holds the address of that variable where it has been stored in the memory.

2. Pointer Declaration:

A pointer is a variable which holds the memory address of another variable. The pointer has the following advantages:

1. It allows passing variables, arrays, functions, strings and structures as function arguments.
2. A pointer allows returning structured variables from functions.
3. It provides functions which can modify their calling arguments.
4. It supports dynamic allocation and deallocation of memory segments.
5. With the help of a pointer, variables can be swapped without physically moving them.
6. It allow to establish links between data elements or objects for some complex data structures such as linked lists, stacks, queues, binary trees, tries and graphs.
7. A pointer improves the efficiency of certain routines.

In C++ pointers are distinct such as integer, float, character, etc. A pointer variable consists of two parts, namely, (i) the pointer operator and (ii) the address operator.

Pointer Operator:

A pointer operator can be represented by a combination of (*) with a variable, for example `int *ptr`; where `ptr` is a pointer variable which holds the address of an integer data type.

General Form of Pointer:
<code>Data_type *pointer_variable;</code>

Example: `int x, y; int *ptr1, *ptr2;`

Address Operator:

An address operator can be represented by a combination of & with a pointer variable. For example, if a pointer variable is an integer type and also declared (&) with the pointer variable, then it means that the variable is of type "address of". For example `m=&ptr`; Note that the pointer operator & is an operator that returns the address of the variable following it.

Note:

Notice the difference between the reference and dereference operators:

- & is the reference operator and can be read as "address of"
- * is the dereference operator and can be read as "value pointed by"

Thus, they have complementary (or opposite) meanings. A variable referenced with & can be dereferenced with *.

Examples:

- (1) `Ptr1 = &x;` The memory address of variable `x` is assigned to the pointer variable `ptr1`.
- (2) `Y=*ptr1;` The contents of the pointer variable `ptr1` is assigned to the variable `y`, not the memory address.
- (3) `Ptr1=&x;` The address of the `ptr1` is assigned to the pointer variable `ptr2`. The contents of both `ptr1` and `ptr2` will be the same as these two pointer variables hold the same address.

Example:

```
andy = 25;  
ted = &andy;
```

Right after these two statements, all of the following expressions would give true as result:

```
andy == 25  
&andy == 1776  
ted == 1776  
*ted == 25
```

Examples of invalid pointer declaration:

(1) int x;

```
int x_pointer;  
x_pointer = &x;
```

Error: pointer declaration must have the prefix of *.

(2) float y;

```
float *y_pointer;  
y_pointer = y;
```


Error: While assigning variable to the pointer variable the address operator (&) must be used along with the variable y.

(3) int x;

```
char *c_pointer;  
c_pointer = &x;
```


Error: Mixed data type is not permitted.

Example 1:

 This simple example to show how can create and use pointer of char.

```
#include <iostream.h>
int main()
{
    char c='a';
    char *p_c = &c;
    cout<< *p_c;
}
```

Example 2:

 This simple example to show how can create and pointer of integer.

```
#include <iostream.h>
main()
{
    int myval=10;
    int *p_myval;
    p_myval = &myval;
    cout<<*p_myval;
}
```

Example 3:

```
#include <iostream.h>
main()
{
    int myval = 7;
    int *p_myval = &myval;
    *p_myval = 6;
    cout<<*p_myval<<"\n";
    cout<<myval;
}
```

Example 4:

```
#include <iostream.h>
main()
{
    int myval=5;
    int myval2 = 7;
    int *p_primate;
    p_primate = &myval;
```

```

    *p_primate = 9;
    p_primate = &myval2;
    *p_primate = 10;
    cout<<myval<<" "<<myval2;
}

```

Example 5:

```

#include <iostream.h>
Void main(void)
{
Float value;
Float *ptr;
Value = 120.00;
Ptr = &value;
Cout<< "Memory address ="<<ptr<<endl;
Ptr --;
Cout<<"Memory address after decremter =";
Cout<<ptr<<endl;
}

```

3. Pointers and Functions:

Pointers are very much used in a function declaration. Sometimes only with a pointer a complex function can be easily represented and accessed. The use of the pointers in a function definition may be classified into two groups; they are call by value and call by reference.

Call by value:

Whenever a portion of the program invokes a function with formal arguments, control will be transferred from the main to the calling function and the value of the actual argument is copied to the function. Within the function, the actual value copied from the calling portion of the program may be altered or changed. When the control is transferred back from the function to the calling portion of the program, the altered values are not

transferred back. This type of passing formal arguments to a function is technically known as call by value.

Example 6:

A program to exchange the contents of two variables using a call by value.

```
#include <iostream.h>
Void main(void)
{
Int x,y;
void swap (int,int);
x=100;
y=20;
cout<<"values before swap"<<endl;
cout<<"x="<<x<<"and y="<<y<<endl;
swap(x,y); //call by value
cout<<"values after swap"<<endl;
cout<<"x="<<x<<"and y="<<y<<endl;
}

int func (int x, int y)
{
int temp;
temp=x;
x=y;
y=temp;
}
```

O/P:

```
values before swap
x=100 and y=20
values after swap
x=100 and y=20
```

Call by reference:

When a function is called by a portion of a program, the address of the actual arguments is copied onto the formal arguments, though they may be referred by different variable names. The content of the variables that are altered within the function block are returned to the calling portion of a program in the altered form itself, as the formal and the actual arguments are referencing the same memory location or address. This is technically known as call by reference or call by address or call by location.

Example 7:

A program to exchange the contents of two variables using a call by reference

```
#include <iostream.h>
Void main(void)
{
Int x,y;
void swap (int *x, int *y);
x=100;
y=20;
cout<<"values before swap"<<endl;
cout<<"x="<<x<<"and y="<<y<<endl;
swap(&x, &y); //call by reference
cout<<"values after swap"<<endl;
cout<<"x="<<x<<"and y="<<y<<endl;
}

int func (int *x, int *y)
{
int temp;
temp=*x;
*x = *y;
*y=temp;
}
```

O/P:

```
values before swap
x=100 and y=20
values after swap
x=100 and y=100
```