Chapter-3

Topic to be covered

- Pointer
- Array of pointer
- Function and pointer

C++ Pointers

- In C++, pointers are variables that store the memory addresses of other variables.

Address in C++

- If we have a variable var in our program, &var will give us its address in the memory.

Cont..

```
#include <iostream>
 using namespace std;
 int main() {
 // declare variables
 int var1 = 3;
 int var2 = 24;
int var3 = 17;
// print address of var1
 cout << "Address of var1: "<< &var1 << endl;
 // print address of var2
 cout << "Address of var2: " << &var2 << endl;
 // print address of var3
cout << "Address of var3: " << &var3 << endl;
 }
```

Output

```
Address of var1: 0x7fff5fbff8ac
Address of var2: 0x7fff5fbff8a8
Address of var3: 0x7fff5fbff8a4
```

Cont..

- Here, 0x at the beginning represents the address is in the hexadecimal form.
- Notice that the first address differs from the second by 4 bytes and the second address differs from the third by 4 bytes.
- This is because the size of an int variable is 4 bytes in a 64-bit system.

What are Pointers?

- A **pointer** is a variable whose value is the address of another variable.
- Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is –

```
type *var-name;
int *pointVar;
```

- Here, we have declared a pointer pointVar of the int type.
- We can also declare pointers in the following way.

```
int* pointVar; // preferred syntax
```

- Let's take another example of declaring pointers.

```
int* pointVar, p;
```

Cont..

- Here, we have declared a pointer pointVar and a normal variable p.
  Note: The * operator is used after the data type to declare pointers.
- There are few important operations, which we will do with the pointers very frequently.

(a) We define a pointer variable.

(b) Assign the address of a variable to a pointer.

(c) Finally access the value at the address available in the pointer variable.

Assigning Addresses to Pointers

- Here is how we can assign addresses to pointers:

```
int* pointVar, var;
var = 5;
// assign address of var to pointVar pointer
    pointVar = &var;
```

- Here, 5 is assigned to the variable var. And, the address of var is assigned to the pointVar pointer with the code pointVar = &var.

Get the Value from the Address Using Pointers

- To get the value pointed by a pointer, we use the * operator. For example:

```
int* pointVar, var;
 var = 5;
// assign address of var to pointVar
pointVar = &var;
 // access value pointed by pointVar
 cout << *pointVar << endl;
// Output: 5
```

Cont..

- In the above code, the address of var is assigned to pointVar. We have used the *pointVar to get the value stored in that address.
- When * is used with pointers, it's called the **dereference operator**.
- It operates on a pointer and gives the value pointed by the address stored in the pointer. That is, *pointVar = var.
- **Note: In C++,** pointVar and *pointVar is completely different. We cannot do something like *pointVar = &var;

```
#include <iostream>
using namespace std;
int main() {
   int var = 5;

   // declare pointer variable
   int* pointVar;

   // store address of var
   pointVar = &var;

   // print value of var
   cout << "var = " << var << endl;

   // print address of var
   cout << "Address of var (&var) =
" << &var << endl
        << endl;

   // print pointer pointVar
   cout << "pointVar = " << pointVar
<< endl;

   // print the content of the
address pointVar points to
   cout << "Content of the address
pointed to by pointVar (*pointVar)
= " << *pointVar << endl;

   return 0;
}
```

Output

```
var = 5
Address of var (&var) = 0x61ff08

pointVar = 0x61ff08
Content of the address pointed to by pointVar (*pointVar) = 5
```

Changing Value Pointed by Pointers

- If pointVar points to the address of var, we can change the value of var by using *pointVar.

For example:

```
int var = 5;
int* pointVar;

// assign address of var
pointVar = &var;

// change value at address pointVar
*pointVar = 1;

cout << var << endl; // Output: 1
```

- Here, pointVar and &var have the same address, the value of var will also be changed when *pointVar is changed.

Common mistakes when working with pointers

- Suppose, we want a pointer varPoint to point to the address of var. Then,

```
int var, *varPoint;

// Wrong!
// varPoint is an address but var is not
varPoint = var;

// Wrong!
// &var is an address
// *varPoint is the value stored in &var
*varPoint = &var;

// Correct!
// varPoint is an address and so is &var
varPoint = &var;

 // Correct!
// both *varPoint and var are values
*varPoint = var;
```

Pointers and Arrays

- In C++, Pointers are variables that hold addresses of other variables. Not only can a pointer store the address of a single variable, it can also store the address of cells of an array.
- Consider this example:

```
int *ptr;
int arr[5];

// store the address of the first
// element of arr in ptr
ptr = arr;
```

- Here, ptr is a pointer variable while arr is an int array. The code ptr = arr; stores the address of the first element of the array in variable ptr.

- Notice that we have used arr instead of &arr[0]. This is because both are the same.
- The addresses for the rest of the array elements are given by &arr[1], &arr[2], &arr[3], and &arr[4].

- Suppose we need to point to the fourth element of the array using the same pointer ptr.
- Here, if ptr points to the first element in the above example then ptr + 3 will point to the fourth element. For example,

```
int *ptr;
int arr[5];
ptr = arr;
```

- ✓ ptr + 1 is equivalent to &arr[1];
- ✓ ptr + 2 is equivalent to &arr[2];
- ✓ ptr + 3 is equivalent to &arr[3];
- ✓ ptr + 4 is equivalent to &arr[4];

- Similarly, we can access the elements using the single pointer. For example,

```
// use dereference operator
*ptr == arr[0];
```

- ✓ *(ptr + 1) is equivalent to arr[1];
- ✓ *(ptr + 2) is equivalent to arr[2];
- ✓ *(ptr + 3) is equivalent to arr[3];
- ✓ *(ptr + 4) is equivalent to arr[4];

Cont..

Suppose if we have initialized ptr = &arr[2]; then

- ✓ ptr - 2 is equivalent to &arr[0];
- ✓ ptr - 1 is equivalent to &arr[1];
- ✓ ptr + 1 is equivalent to &arr[3];
- ✓ ptr + 2 is equivalent to &arr[4];

```
#include <iostream>
using namespace std;
int main()
{
   float arr[3];
   // declare pointer variable
   float *ptr;
   cout << "Displaying address using arrays: " << endl;

   // use for loop to print addresses of all array
elements
   for (int i = 0; i < 3; ++i)
   {
      cout << "&arr[" << i << "] = " << &arr[i] << endl;
   }
   // ptr = &arr[0]
   ptr = arr;
   cout<<"\nDisplaying address using pointers: "<<
endl;
   // use for loop to print addresses of all array
elements
   // using pointer notation
   for (int i = 0; i < 3; ++i)
   {
      cout << "ptr + " << i << " = "<< ptr + i << endl;
   }

   return 0;
}
```

output

```
Displaying address using arrays:
&arr[0] = 0x61fef0
&arr[1] = 0x61fef4
&arr[2] = 0x61fef8
Displaying address using pointers:
ptr + 0 = 0x61fef0
ptr + 1 = 0x61fef4
ptr + 2 = 0x61fef8
```

Cont..

- In the above program, we first simply printed the addresses of the array elements without using the pointer variable ptr.
- Then, we used the pointer ptr to point to the address of a[0], ptr + 1 to point to the address of a[1], and so on.
- In most contexts, array names decay to pointers. In simple words, array names are converted to pointers.
- That's the reason why we can use pointers to access elements of arrays.

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

int main() {
   float arr[5];

  // Insert data using pointer notation
   cout << "Enter 5 numbers: ";
   for (int i = 0; i < 5; ++i) {

      // store input number in arr[i]
      cin >> *(arr + i) ;

   }

   // Display data using pointer notation
   cout << "Displaying data: " << endl;
   for (int i = 0; i < 5; ++i) {

      // display value of arr[i]
      cout << *(arr + i) << endl ;

   }

   return 0;
}
```

output

```
Enter 5 numbers: 2.5
3.5
4.5
5
2
Displaying data:
2.5
3.5
4.5
5
2
```

- We first used the pointer notation to store the numbers entered by the user into the array arr.

- cin >> *(arr + i) ; This code is equivalent to the code below:

  cin >> arr[i];

- Notice that we haven't declared a separate pointer variable, but rather we are using the array name arr for the pointer notation.

- As we already know, the array name arr points to the first element of the array. So, we can think of arr as acting like a pointer.

- Similarly, we then used for loop to display the values of arr using pointer notation.

```
cout << *(arr + i) << endl ;
This code is equivalent to
cout << arr[i] << endl ;
```

Pointer and function

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

// function prototype with pointer as parameters
void swap(int*, int*);

int main()
{

   // initialize variables
   int a = 1, b = 2;

   cout << "Before swapping" << endl;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;

   // call function by passing variable addresses
   swap(&a, &b);

   cout << "\nAfter swapping" << endl;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;
   return 0;
}

// function definition to swap numbers
void swap(int* n1, int* n2) {
   int temp;
   temp = *n1;
   *n1 = *n2;
   *n2 = temp;
}
```

Output

```
Before swapping
a = 1
b = 2
After swapping
a = 2
b = 1
```

- Here, we can see the output is the same as the previous example. Notice the line,

```
// &a is address of a
// &b is address of b
swap(&a, &b);
```

- Here, the address of the variable is passed during the function call rather than the variable.
- Since the address is passed instead of value, a dereference operator * must be used to access the value stored in that address.

```
temp = *n1;
*n1 = *n2;
*n2 = temp;
```

- *n1 and *n2 gives the value stored at address n1 and n2 respectively.

- Since n1 and n2 contain the addresses of a and b, anything is done to *n1 and *n2 will change the actual values of a and b.

- Hence, when we print the values of a and b in the main() function, the values are changed.