Instructor created

6.20 Pointers

The following materials were taken from [Programiz]

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. For example,

Code Editor 6.20.1: Printing Variable Addresses in C++

Click run to see the output.

Load default template...

```
#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;
}
```

Run

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.

C++ Pointers

As mentioned above, pointers are used to store addresses rather than values.

Here is how we can declare pointers.

```
int *pointVar; // * pointer is next to variable.
```

Here, we have declared a pointer `pointVar` of the `int` type.

```
int* pointVar; // preferred syntax; * pointer is next to the type.
```

Let's take another example of declaring pointers.

```
int* pointVar, p;   // Note: only the first variable is a pointer.  The second variable is just an interger.
```

Here, we have declared a pointer `pointVar` and a normal variable `p`.

Note: The `*` operator is used after the data type to declare pointers.

Common Error: the above code is equivalent to:

int *pointvar;

int p;

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

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;`

Code Editor 6.20.2: Working of C++ Pointers

Click run to see the output.

Load default template...

```
#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;
}
```

Run

Figure 6.20.1: Working of C++ pointers

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.

Code Editor 6.20.3: Changing Value Pointed by Pointers

Click run to see the output.

Load default template...

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

  // store address of var
  pointVar = &var;

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

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

  cout << "Changing value of var to 7:" << endl;
}
```

Run

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;
```