C++ Pointers

A pointer is a C++ data type. The value that a pointer can hold is a memory location. We say that a pointer variable *points* to the memory location. There are actually many different data types that are pointers, depending on the data type of the value stored in the memory location to which the pointer points. For example, an **`int`** pointer points to an **`int`** value in memory, a **`double`** pointer points to a **`double`** value stored in memory.

Declaring Pointers

Unlike other data types in C++, the pointer types aren't given a specific name. This means that they are declared in a slightly different way. The syntax to declare a pointer is

*dataType *identifier;*

To declare an **`int`** pointer you would have

```
int *k;
```

And to declare a **`double`** pointer, you would have

```
double *y;
```

In the above two examples, the asterisk (`*`) indicates that the identifier is a pointer to the specified type.

All three of the following declarations are equivalent:

```
int *k;

int* k;

int  *  k;
```

In the following statement, only **`i`** is a pointer. Variable **`j`** is an **`int`**, not an **`int`** pointer.

```
int* i, j;
```

To declare both **`i`** and **`j`** as **`int`** pointers, you would have to do this

```
int *i, *j;
```

Assigning Values to Pointers

Since a pointer holds a memory location, we use the address of operator **&** to assign a value to a pointer, like this

```
int x = 7; int *p = &x;
```

Here, **x** is an **int** variable and **p** is a pointer that holds the memory location where the value of **x** is stored. We say the **p** points to **x**. To access the value of **x** using the pointer **p**, we dereference the pointer like this

```
cout << *p;
```

Which causes the value of **x** to be displayed. The program in Listing 8.1 and its output in Figure 8.1 demonstrate how a pointer works.

Programming Example: Pointers1.cpp

Pointers are related to references.

Programming Example: Pointers2.cpp

Reference Parameters Using Pointers

We saw earlier how to use references to pass parameters to functions "by reference". The same thing can be done using pointers.

Programming Example: Pointers3.cpp

There are four ways to pass a pointer to a function:

A non-constant pointer to non-constant data

> This is what we've seen so far. The pointer can be dereferenced to modify the object to which it points. The pointer itself can be changed to point to different objects. Programming Example: Pointers4.cpp

A non-constant pointer to constant data

> Here the object pointed to can no longer be modify through the use of the pointer, but the pointer can be made to point to different objects.

A constant pointer to non-constant data

> The object pointed to can be changed, but the pointer can no longer be made to point to different objects.

A constant pointer to constant data

> The object pointed to cannot be changed, nor can the pointer be made to point to different objects.

The `sizeof` Operator

The C++ sizeof operator returns the number of bytes of any valid data type. This can be used to determine the number of elements in an array.

Programming Example: Pointers5.cpp

Pointers and Arrays

As we saw earlier, an array is simply a reference to a memory location. Arrays and pointers are therefore closely related. When a pointer references an array, we can do *pointer arithmetic*.

Programming Examples: Pointers6.cpp and Pointers7.cpp