Overview

A pointer is a variable or constant that holds a memory address. Pointers may be a new term to you, or you may have heard that they are difficult to understand. As with any new concept, however, once you become familiar with the principles and how to apply them you will see that pointers are not difficult. This chapter will cover the basics of pointers and give you a firm foundation that you will use in chapters to come.

You will also take another look at character arrays. You will learn how to access individual characters using a method called subscript notation and using pointers. Finally, you will use a feature of C++ that lets you create your own data types.

CHAPTER 10, SECTION 1

Pointer Basics

C++'s extensive support of pointers is one of the things that makes it such a powerful language. In this chapter, you will not see all of the power that pointers bring to the language. You will, however, learn the basics of pointers that you need to unleash that power in later chapters.

REFRESH YOUR MEMORY ABOUT MEMORY

Each byte of a microcomputer's memory has a unique address. The address is just a number. Memory addresses start at zero and count up from there. Actually, the way memory is organized and the way addresses are assigned varies among computers. The important thing to know is that each byte is numbered in order. For example, memory location 221345 is next to memory location 221346.

Programming would be more difficult if you had to remember the addresses where you stored your data. Instead, the compiler lets you assign names to memory locations when you declare variables.

WHAT IS A POINTER?

A *pointer* is a variable or constant that holds a memory address. In fact, you have been using pointers already. For example, when you declare a character array like the one shown below, `state_code` is a pointer to the first character in the character array.

```
char state_code[3];
```

Figure 10-1 shows how the character array looks in memory.

The pointer occupies four bytes of RAM. The pointer stores the value 140003, which is the memory location where the character array begins.

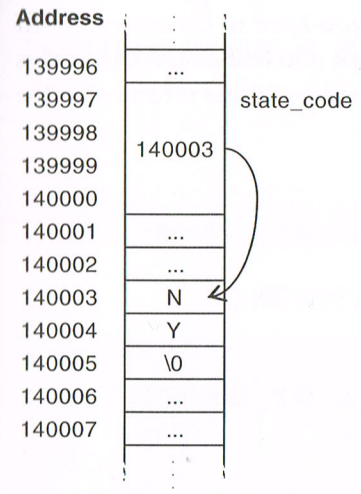

FIGURE 10-1
Declaring a character array creates a pointer to the first character contained in the array.

Note

Pointer size varies among computers and operating systems. For our purposes, the size of pointers is unimportant.

Pointers can point to more than arrays. You can create a pointer that points to any data type. For example, suppose you want a pointer that points to an integer. Figure 10-2 shows how the actual integer is stored, and how the pointer variable points to the integer.

In this example, the integer `i` (with value 3) is stored in two bytes of memory at address 216801. The pointer `iptr` points to the variable `i`.

You may be wondering why anyone would want a pointer to an integer. At this level, pointers may seem "pointless." In later chapters, however, you will learn how to put the power of pointers to work for you when more advanced methods of handling data are discussed.

DECLARING POINTERS

The code below shows how an integer and pointer like the one in Figure 10-2 is declared.

```
int main()
{
  int i;        // declare an integer i
  int *iptr;    // declare a pointer to an integer

  iptr = &i;          // initialize the pointer to the address of i
  i = 3;              // initialize i to 3;
  return 0;
}
```

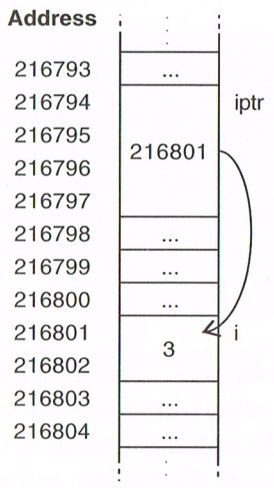

FIGURE 10-2
A pointer can point to an integer or other data type.

Working with pointers requires the use of two new operators: the *dereferencing operator* (*) and the *address-of operator* (&). Let's examine what is accomplished by the code above.

The statement `int *iptr;` declares a pointer by preceding the variable name with a dereferencing operator (*). Notice this is the same symbol used to indicate multiplication; however, your compiler can tell the difference by the way it is used.

Notice that pointers have types just like other variables. The pointer type must match the type of data you intend to point to. In the example above, an int pointer is declared to point to an int variable. The * before the variable name tells the compiler that we want to declare a pointer, rather than a regular variable. The variable `iptr` cannot hold just any value. It must hold the memory address of an integer. If you try to make a pointer point to a variable of a type other than the type of the pointer, you will get an error when you try to compile the program.

Like any other variable, a pointer begins with a random value and must be initialized to "point" to something. The statement `iptr = &i;` is what makes `iptr` point to `i`. Reading the statement as "`iptr` is assigned the address of `i`" helps the statement make sense. The address-of operator (&) returns the "address of" the variable rather than the variable's contents.