Just because you use pointers does not mean you have to concern yourself with exact memory locations. When you use pointers, you will assign an address to a pointer using the address-of operator (&). The exact address returned by the operator is of importance only to the computer.

EXERCISE 10-1 DECLARING POINTERS

1. Enter the program below. Save the source code as *POINTER.CPP*.

```
#include <iostream.h> // necessary for cout command

int main()
{
  int i, j, k;     // declare three integers
  int *int_ptr;    // declare a pointer to an integer

  i = 1;
  j = 2;
  k = 3;
  int_ptr = &i;    // initialize the pointer to point to i

  cout << "i = " << i << '\n';
  cout << "j = " << j << '\n';
  cout << "k = " << k << '\n';
  cout << "int_ptr = " << int_ptr << '\n';
  return 0;
}
```

2. Compile and run the program to see that **i, j,** and **k** hold the values you expect. The variable **int_ptr** outputs a memory address to your screen when you print it. The address will probably print in a form called hexadecimal, which is a combination of numbers and letters. Learning the exact meaning of the address is of little importance: just realize that it is the address where the variable **i** is stored.
3. Leave the source code file on the screen for the next exercise.

Note

A pointer can be named using any legal variable name. Some programmers use names that make it clear the variable is a pointer, such as ending the name with **ptr** *or beginning the name with* **p_**, *but it is not necessary.*

The Hexadecimal Number System

Hexadecimal numbers (often called just "hex") are ideal for representing memory locations. The hexadecimal number system is a base 16 number system. Because base 16 is a multiple of base 2 (binary), the hexadecimal number system is compatible with the computer's internal representations. Hexadecimal numbers can display large values in a form that is easier to read than binary numbers.

In Exercise 10-1, you declared the pointer (**int *int_ptr;**) and initialized it in a different statement (**int_ptr = &i;**). Like other variables, you can initialize a pointer when you declare it. For example, the statement below could have been used in the program in Exercise 10-1 to declare the pointer and initialize it in one statement.

```
int *int_ptr = &i;
```


USING THE * AND & OPERATORS

The dereferencing operator (*) is used for more than declaring pointers. In the statement below, the dereferencing operator tells the compiler to return the value in the variable being pointed to, rather than the address of the variable.

```
result = *int_ptr;
```

The variable **result** is assigned the value of the integer pointed to by **int_ptr**.

EXERCISE 10-2 THE * AND & OPERATORS

1. Enter the following lines of code to the program you saved named *POINTER.CPP*:

```
cout << "&i = " << &i << '\n';
cout << "*int_ptr = " << *int_ptr << '\n';
```

The output statement with the **&i** does the same thing as outputting **int_ptr**, since **int_ptr** holds the address of **i.** Sending ***int_ptr** to the output stream prints the contents of the variable pointed to by **int_ptr**, rather than printing the pointer itself.

2. Compile and run to see the output of the new statements.
3. Enter the following statements at the end of the program:

```
int_ptr = &j; // store the address of j to int_ptr
cout << "int_ptr = " << int_ptr << '\n';
cout << "*int_ptr = " << *int_ptr << '\n';
```

4. Compile and run again. Because **int_ptr** now points to the integer **j** rather than **i**, the output statement prints the value of **j**, even though the exact statement (**cout << "*int_ptr = " << *int_ptr << '\n';**) printed the value of **i** just two statements back.
5. Enter the following statements at the end of the program:

```
int_ptr = &k; // store the address of k to int_ptr
cout << "int_ptr = " << int_ptr << '\n';
cout << "*int_ptr = " << *int_ptr << '\n';
```

6. Compile and run again. The pointer now points to the variable **k**, so ***int_ptr** returns the value of **k**, which is 3.
7. Save and close the source code file.

CHANGING VALUES WITH *

The dereferencing operator allows you to do more than get the value in the variable the pointer is pointing to. You can change the value of the variable the pointer points to. For example, the statement below assigns the value 5 to the integer to which **int_ptr** points.

```
*int_ptr = 5;
```