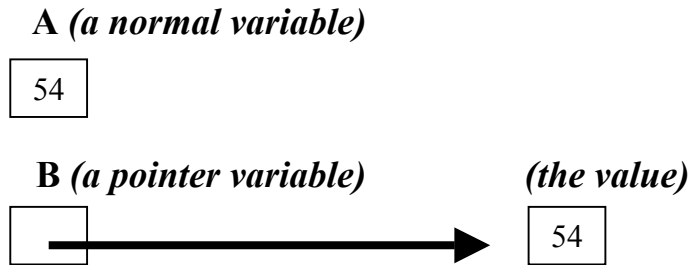


Chapter 40 – Pointers

1. Pointers

A normal variable stores a value at the RAM address of the variable. A pointer variable does not store a value at its RAM address. Instead, a pointer variable contains the RAM address of a value. In the parlance of C++, a pointer variable points to the location of a value or points to a value.



To create a pointer variable, the following syntax is used:

```
class-or-type *variable-name;
class-or-type* variable-name;
```

Here are some examples of creating pointers:

```
int *a;                // may point to an integer value location
phonecard *cardpointer; // may point to an object of struct type phonecard
phonebook *p;         // may point to an object of class phonebook
```

Just because a pointer exists does not mean it points to an object. Pointers may point to the address of a normal variable (obtained by using the operator & to return the address of a variable) or may point to an unnamed variable called into existence with the function *new*.

Here are some examples of using a pointer to point to a named variable object:

```
int *a, c = 3;        // creates pointer to integer a, c is an integer with the value of 3
a = &c;              // assigns a the address of the integer variable c
```

The value in variable *c* can be output in the normal fashion:

```
cout << c;           // outputs the value in the variable c
```

The value in the variable *c* can also be output by using the variable *a* with an asterisk in front:

```
cout << *a;      // outputs the value that the variable a points to, which is the value
                  // in the variable c
```

The value in variable *c* can be changed by the normal means:

```
c = 6;
```

This results in *c* having the value of 6 and **a* pointing to the value of 6. The value of *c* can also be changed by using **a*.

```
*a = 8;
```

Now both *c* and **a* points to the value of 8.

Pointers may point to variables that do not have names. These variables are created with the function *new*, which secures a memory location from the operating system just as declaring a variable in the usual way would, except that variables created with the function *new* do not have names associated with them.

Given the following struct:

```
struct dailytemps {
    float hi, low;
};
```

A pointer to an object for struct type *dailytemps* can be created and used as follows.

```
dailytemps *t1;
t1 = new dailytemps();    // creates an unnamed object of struct dailytemps and
                          // assigns the address of it to the pointer variable t1
```

t1 can be used to access the parts of the unnamed struct object by use of the pointer operator *->*. For example:

```
t1->hi = 55;
t1->low = 43;
```

There is a problem with calling unnamed objects into existence. A programmer that uses *new* to create an object is responsible for returning the memory allocated by *new* back to the operating system when it is no longer needed. This is done with the function *delete*.

Since *t1* points to an unnamed object created with *new*, here is how *t1* can be used with *delete* to return the addresses of the unnamed object back to memory:

```
delete t1;
```

In this example, *t1* is still available, but the thing that *t1* pointed to is not available.

Pointers have many uses. Chiefly, they are used to create data structures that are not possible or are not easily represented using arrays. The next several chapters will concentrate on data structures that are built using pointers.

Not so strangely, array elements can be accessed by indexed pointers. Array names really are actually pointers to the first element of an array. Each successive element of the array is accessible via an ever larger offset from the start called an index. For example, an array can be traversed by adding the index variable to the array name in the following manner. (Notice that the asterisk (*) must be used with the array name and index to indicate that the array name is to be used as a pointer.)

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

void main( ) {
    int a[5] = {23, 45, 1, 98, 26};

    for (int x=0; x<5; x++) {
        cout << "a[" << x << "] is ";

        cout << *(a+x);    // here the array name is being used as a
                           // pointer and the index is being added to it
                           // to reach the individual elements

        cout << endl;
    }
}
```

When run, this program produces the following output:

```
a[0] is 23
a[1] is 45
a[2] is 1
a[3] is 98
a[4] is 26
Press any key to continue.
```

Programming Exercise 40.1

Enter the following code as a C++ program. Before running it, predict the output. Then run the program and compare the values output with your predictions. If there are differences, determine why your predictions were incorrect.

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

void main() {
    int a = 1;
    int *b = new int();
    *b = 2;
    cout << "address of a is " << &a << endl
         << "value of a is " << a << endl
         << endl
         << "address of b is " << &b << endl
         << "the address that b points to is " << b << endl
         << "the value that b points to is " << *b << endl;
    delete b;
}
```

Programming Exercise 40.2

Write a program that contains 3 pointers to double. Use *new* to assign memory addresses to each, then assign values to them. Output the address of each pointer, the address that each points to, and the values at these address. Use *delete* to return the memory assigned by *new* back to memory for later use.