CS 162C++
Pointers

Pointers are variables that hold addresses. They have a variety of different uses.

Basic Concept

When we define a variable, we tell the compiler what sort of thing will be stored in it. Although every location in memory contains some pattern of ones and zeros, by knowing what the programmer's intent is allows the compiler to properly interpret those contents.

Consider the following illustration of how variables might be stored in memory. Here we are assuming that an integer and an address both take up 32 bits or 4 bytes, that memory addresses are displayed as hex values, and that items are stored in memory starting at 9xFFFF and going down.

```
int x = 4;
int y;
int z = 3;
```

Variable Name	Memory Location	Value
x	0xFFFC	4
y	0xFFF8	??
z	0xFFF4	3

The first variable allocated, x, is located four bytes from the top of memory so that there is room for the integer to be stored there. The next variable is four bytes lower and so forth. The second variable, y, has a value of ??, indicating that there is something stored in that location, but we do not know what it is.

We define a pointer by using the * to indicate that it is a pointer, along with specifying the type of thing that it will point to. For example:

```
int * p1;        // this is a variable that holds the address of an integer
float * p2;      // this is a variable that holds the address of a float
string * p3;     // this is a variable that holds the address of a string
```

The amount of memory that each of these variables takes up is the same, it is the length of an address on this computer.

Variable Name	Memory Location	Value
x	0xFFFC	4
y	0xFFF8	??
z	0xFFF4	3
p1	0xFFF0	??
p2	0xFFEC	??
p3	0xFFE8	??

Now, let's save the address of x in the variable p1:

```
p1 = &x;
```

Note that we use an &, which is the same operator that we used before when we were indicating pass by reference when defining a function. Now our table looks like:

Variable Name	Memory Location	Value
x	0xFFFC	4
y	0xFFF8	??
z	0xFFF4	3
p1	0xFFF0	0xFFFC
p2	0xFFEC	??
p3	0xFFE8	??

The location in memory for the variable p1 now contains the address of the variable x. If we executed the following code segment:

```
cout << &x << " = " << x << endl;

cout << &p1 << " = " << p1 << endl;
```

We would get the address of x followed by the contents at the location for x on one line and the address of p1 followed by the contents of the location for p1 on the next line:

```
0xFFFC = 4

0xFFF0 = 0xFFFC
```

You can run the following program to see what the output will be on your computer. Note that the actual addresses will be different, but the logic should remain the same.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
   int x = 5;
   int y = 6;
   int *p = &x;

   cout << &x << " = " << x << endl;
   cout << &y << " = " << y << endl;
   cout << &p << " = " << p << " and that contains " << *p << endl;

   return 0;
}
```

Note that this contains the **dereference operator** *, that allows us to access the contents of the location that the pointer points to (the contents of the address stored in the pointer).

Summary

Pointers hold the addresses of other variables. You need to define what sort of address when you define the pointer. To access the address of a variable, you use the &. This allows you to save that address in a pointer. If you display the contents of a pointer, it is the address that it is pointing to. If you want to reach through that address and get the contents of the location the pointer is pointing to, you need to use the dereference operator *.