Learning C++ Pointers for REAL Dummies

Introduction

Hello and welcome to "Learning C++ Pointers for REAL Dummies." This website was created by Paul DiLorenzo to fill the void of an easy understanding learning module for pointers.

Here are some rave reviews from people around the world:

"GOOD JOB with learning C++ pointers for dummies. Because of you I hope to get a 10 (that means A) at programming." Dan

"Sir, your tutorial on pointers is a masterpiece, I think you should continue writing more topics on C++." Mayur Jethwa

Pointers are a very difficult and troublesome area for most C++ programmers, beginners and intermediate alike. Most questions pop up about their use and why we even need them. I hope this website helps answer these questions for you and demystify the C++ pointer.

To understand pointers we have to understand how variables are stored. Variables are stored in memory cells inside the computer's memory. The computer's memory is made up of consecutive memory cells, a byte long, each with a unique address.

But we are not going to think in those terms. We are going to believe that computer memory is made up of a bunch of houses on one very long street. Thus, each house is a memory cell. Now, there must be a way for us to find this house. Well, in each house someone lives there. This person of course has a name and this will be our variable identifier. For example:

```
int paul;
```

This will put paul into a vacant house, of size int, somewhere along the street. We do not decide where paul will live. This is done by the operating system and the compiler at runtime. In a later section, we will discuss how to get paul to tell us where his house is.

Currently, paul does not have anything stored in his house. But, we all know that wouldn't be any fun to not store anything. So, each house can of course store a value. Continuing from above:

```
paul = 25;
```

This will store the value 25 into paul's house.

One last thing before we move into other topics. Let us remember that paul's house is a unique number in memory. In addition, if paul's house was numbered 1234 we know that his house is between houses 1233 and 1235. This is a very important concept for later sections.

This learning module is broken up into seven parts:

Where do you live (&) - Explains what the address operator (&) is and how it is used

What you got in your house (*) - Explains that the reference operator (*) is and how it is used

Don't point, it's rude! - Explains how to declare variables that are pointers

I **am pointing here! Where are you pointing?** - Explains how to initialize a pointer

How do I point to the neighbor? - Explains how to move a pointer from one house to the next

You guys are brothers? - Explains the relationship between arrays and pointers

Beam me up Scotty, multiple times!!! - Explains how pointers can point to other pointers

I would recommend starting at "Where do you live?" and moving from there. But you can also skip to a part that you need help with and learn from there. Have fun!!!

Where do you live? (&)

Following along with our smilie, when we declare a variable a person is put into the house so we can access what is stored there.

But, what if we wanted to know where this person lives. This is done by preceding the variable indentifier with an ampersand sign (&), which means "address of." For example:

melissa = &paul;

This line would store the address of paul into melissa's house.

Now, let's suppose paul's address is 1500.

paul = 21;

tom = paul;

melissa = &paul;

So, for this example, the first line will store 21 into paul's house. The second line will store the value 21 inside tom's house. So far no different then what we have usually done. The third line, though, will store the address of paul, 1500, into melissa's house.

The following diagram might help to understand this example:

For a Flash version, click here.

The variable melissa is what we call a pointer. We have not gotten to how we declare a pointer, since it has certain qualities that we will discuss in Don't point, it's rude!

What you got in your house? (*)

Previously in Where do you live?, we set up an example where we put paul into a house located at 1500 Computer Lane storing the value 21. Then, we put melissa into a house storing the address of paul's house, 1500.

Now, what if we wanted to know what was the value stored at the house in which melissa was pointing at. To do this we would use the reference operator (*). This could be thought of as saying, "value pointed by". For example:

dave = *melissa;

This will store 21 in dave's house. To follow our simile, dave will ask melissa what value she is storing. She will say, "1500." Dave will then go to address 1500 and ask the person living there what is stored in their house. That person, who we know is paul, will say, "21." So, dave will know to store 21 in his house. Graphically it would look something like this:

For a Flash version, click here.

So from this example, if we printed out the value of dave, it would print out 21. We could also type in:

*melissa = 30;

This would read, "The value pointed by melissa equals 30." So, paul will equal 30, but dave will not equal 30 since he does not get updated. He stores what he originally stored and does not change.

Until now, I have glossed over the detail on how melissa is created. The next section explains how to declare a pointer of a certain type.

Don't point, it's rude!

Ok, so we have learned about the address-of operator (&) and the reference operator (*), but we haven't learned how to declare a pointer to use these operators on. Well this is how you do it:

```
int *melissa;
```

That's it! This will move a person named melissa into a house, who's sole purpose is to store the address of someone else in computer town. Basically, she is a pointer!

There are a few rules that I should explain before we go and learn how to tell her where to point. If we have a line of code like this:

```
int *melissa, paul;
```

This does not create two pointers, melissa and paul. It actually creates a pointer of size int named melissa and a regular int named paul. If we want to declare two pointers of the same type we would do this:

```
int *melissa, *paul;
```

This will create two pointers of size int named melissa and paul.

Another thing is size. A pointer must know how big the house it is pointing to. So if paul is an int, only an int pointer can point to it. If paul is a float, then only a float pointer can point to it, and so on.

I am pointing here! Where are you pointing?

Now that we know how to make a pointer we have to initialize it. We have actually done this before. We use the (&) symbol to get the address of the variable we want to point to and store in that variable. Such as:

```
int paul = 21;

int *melissa;

melissa = &paul;
```

So here we move paul into his house, of size int, and store 21 in it. We then declare that melissa will be a person that only points to houses of size int. We then tell melissa to point to paul's house. And that is it! We have done it before, just had to put it all together. We can also tell the pointer to point to the same thing someone else is pointing to. For example:

```
int paul = 21;

int *melissa, *dave;

melissa = &paul;

dave = melissa;
```

This will create two people who point, melissa and dave. Melissa will store the value of the address of paul, as seen on line 3. But then dave will also store the value of paul, because he gets it from melissa. So, we now have two people pointing to the same house. Cool, huh?

Ok, so let's put it all together and try a complete example:

```
int a = 5, b = 10;

int *p1, *p2;

p1 = &a;

p2 = &b;

*p1 = 10;

p1 = p2;

*p1 = 20;

printf("a = %d\n", a);

printf("b = %d\n", b);
```

So, what will be printed out? Click here to see the answer.

How do I point to the neighbor?

What if we have a pointer pointing to some address, but we want to know what are the values around it. You might be thinking, "Why would we want to do that?" Well, let's say we declared an array. And we wanted to point to the second element. But, then, farther down we wanted to point to the third element. This is how we would do it:

```
int a[5] = {1,2,3,4,5};

int *p1;
```

p1 = &a[1]; // gets address of this element

printf("*p1 = %d\n", *p1);

p1++; // point to the next element

printf("*p1 = %d\n", *p1);

Click here to see the answer. And if we wanted to go back an element it would be:

p1--;

Remember, that the position of the (++) and (--) operators in respect to the variable matters. For example, if I wrote:

int *p2;

p1 = &a[1];

p2 = p1++;

p2 would not point to the third element, but actually to the second element since the (++) operator is after the variable it is done after the assign (=) operator. So *p1 would equal 3 while *p2 would equal 2.

You might now be thinking, "Well if I wanted to point two elements down, I would have to do 'p1++' twice?" And I would say, "That is a very good question. The answer to your question is no."

Pointers are able to do math. Continuing from the first example:

p1 = &a[1];

printf("*p1 = %d\n", *p1);

p1 = p1+2;

printf("*p1 = %d\n", *p1);

This would print out:

*p1 = 2

*p1 = 4

The pointer knows to go down two houses. Pointers can also get the value of an address for another variable, report it, but still point to the same place. Let's see an example:

```
p1 = &a[1];

int dave;

dave = *(p1+2);

printf("*p1 = %d\n", *p1);

printf("dave = %d\n", dave);
```

What do you think is printed out. Click here to see the answer.

As you probably can already tell, arrays and pointers seem to be almost brothers. See You guys are brothers? for a better explanation of how arrays and pointers are related.

You guys are brothers?

Well, if you noticed with the lesson How do I point to the neighbor?, arrays and pointers are very closely linked. In fact, they are kind of like brothers.

Let's say we have two brothers. One brother finds a girl, settles down with her, and stays with that woman for the rest of his natural life. He is very constant. The other brother is the complete opposite. He might stay with a woman for a day, a week, a month, or maybe a couple of years. He might somewhere in his life settle down, but he doesn't stick with only one girl. Well this is the same with pointers and arrays.

An array is really just a constant pointer. When you declare an array like:

```
int ramon[5];
```

ramon is actually a pointer that constantly points at the beginning of the array. We cannot point ramon somewhere else. To make a const pointer we would type:

```
const int *tony;
```

This will create a const pointer called tony. Once we initialize where tony will point, we cannot point him somewhere else or his wife will be mad... I mean the compiler will give you an error.

But, remember the wild brother can always point to the constant pointer...

```
int ramon[5];
```

```
int *paul;

paul = ramon;
```

Now, paul and ramon are pointing to the same thing; the first element of the array.

Beam me up Scotty, multiple times!!!

The question might come up, "How can I get a pointer to point to a pointer?" Ok, maybe this question won't come up in your mind, but it might come up in your programming. I just want to touch on the subject on how to do it.

First let's look at some code:

```
int **ramon;

int *paul;

int melissa = 5;

paul = &melissa;

ramon = &paul;

printf("ramon = %d\n", ramon);

printf("&paul = %d\n", &paul);

printf("*ramon = %d\n", *ramon);

printf("&melissa = %d\n", &melissa);

printf("**ramon = %d\n", **ramon);
```

Let's take this line by line. The first line declares a double pointer called ramon. We then declare a pointer called paul. And finally we declare a regular integer called melissa which we initialize to 5.

Next, we point paul to melissa, which stores melissa's address into paul's house. Then we point ramon to paul, which will store paul's address into ramon's house.

So, now let's look at what will be printed out using the diagram below's addresses.

ramon = 1000

&paul = 1000

*ramon = 500

&melissa = 500

**ramon = 5

Well the first printout is pretty simple. What is stored in ramon's house which is the address of paul's house. We check that with the second printout. Then we printout the value of what ramon is pointing to. This would be melissa's address since that is what paul is storing. We check this with the fourth line of code. Now, the final line is the tricky part. It simply asks, "Whatever is the value that ramon is pointing to, take that value and see what is the value that it is pointing to." If that made no sense at all, another way to think of it is breaking it up. For example:

*(*ramon)

Take the value of what ramon is pointing to, 500. Now take 500 and see what is stored in that address, 5. See, it is that easy. Ok, well, maybe the picture below will help you out.

The best way to learn this stuff is just doing some programming examples on your own and see what comes out.

Hope you had fun learning this material as I have had making it. Make sure you take the QUIZ for REAL Dummies to see if you learned anything. Take care.

Ref: http://alumni.cs.ucr.edu/~pdiloren/C++_Pointers/