**Chapter 12: Pointers**

Pointers are often seen as the most confusing aspect of C++. This is probably in partly because so many languages these days have managed memory. That is to say, the compiler takes care of allocating / de-allocating memory. C++ however gives you full control of this. This chapter will step you through the process. Ensure you understand each of the examples before continuing. It is easy to get lost on this topic.

**What is a pointer?**

Well, a pointer is a variable like any other variable you have created up to now, but instead of it storing a value it stores a memory address. This memory address points to a location in memory where the actual value is stored.

**Why not just use variables?**

Using pointers allows us the ability to change the value from anywhere in the program that has access to the pointer. So, as we saw in the functions examples when you pass a value through as a parameter, any changes in the function are lost. If we access the memory address and pass by pointer, then any changes to the value through this pointer will persist.

Let us look at some examples...

The first thing we need to know is how to define a variable of pointer type. Well, we still need to know what data type we are using, exactly as you would for a normal variable. This is because the compiler needs to reserve a chunk of memory large enough to hold the data.

**Example: Declaring pointers**

The only difference in declaration of a pointer variable when compared to a normal variable is the use of the’\*’ symbol. See below:



**Example: Initialising pointers**

Straight forward enough, but at this point we have only declared a pointer variable, we have not yet reserved the memory required to hold the value. To do this we need to use the new keyword. There are several ways to initialise the pointer. One is to use the new keyword followed by the data type, then a second line of code using the ‘\*’ symbol again (indirection operator) to dereference the pointer. That is to say, access the value stored at the address held in the pointer (sounds more confusing than it really is). Like so:



You may notice that there is a green line under our pointer. This is to tell us that assignment and declaration can be joined:



Alternatively, we can set the value at declaration as well:



This can also be written thusly:



It is also possible to use your pointer to point to a normal variable of the same type. So, this example creates an int called myNum and then creates an int pointer, which points at myNum. Both variables can be used to change the value. Note the use of the ‘&’ symbol. This retrieves the address of a variable.

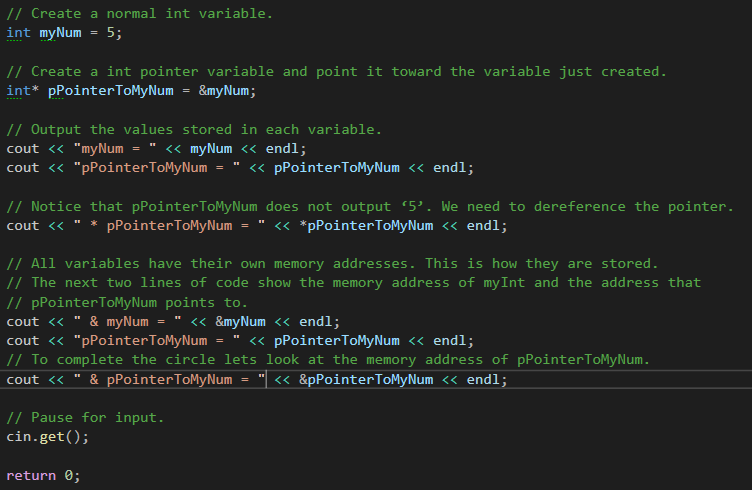


The second line in the above code snippet says, create enough memory to hold the address of an int and put the address of myNum in there.

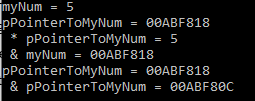
Note: For every *new* keyword used in your program, you must have a matching *delete* keyword to remove it from memory. This prevents memory leaks etc.

Confused? Let’s look at a full code example using the above code snippets.

**Example: Starting with Pointers**



**Output:**



The above example is straightforward with the comments. This is showing you the different options you have to output to console just as seen in the lecture. Unlike normal variables, we have three options rather than two. Outputting the value of the pointer, just as we have any other variable so far (this would output the address of the variable it’s linked with), output the value at the address held using the ‘\*’ symbol, and outputting the personal address of the pointer using the ‘&’.

Note: If you were to run this program yourself, don’t worry if the addresses do not match. These change with each execution of the program.

Next, we will take a look at how we can use pointers and functions. As said above, we can change the value stored at the pointer address from anywhere in our program that has access to the pointer, which includes passing it into a function.

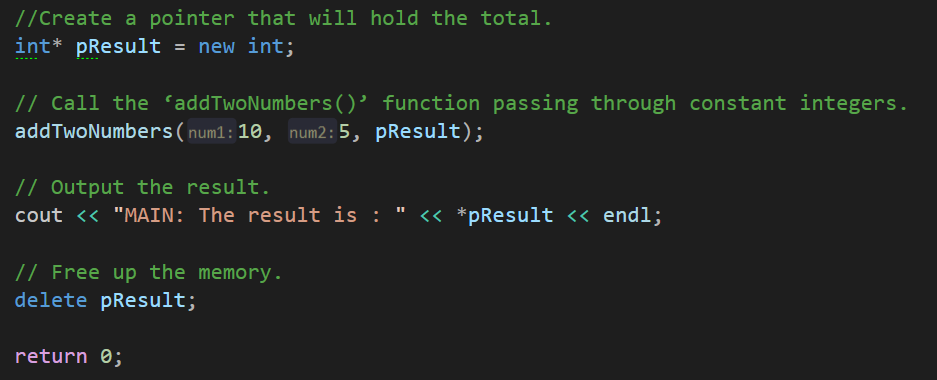
**Example: Passing by Pointer**

This is the same example that was used in Chapter 7: Functions, where a function is created that takes two integers and adds them together. In this version we will see how we can pass through the result pointer and see that it is permanently changed.

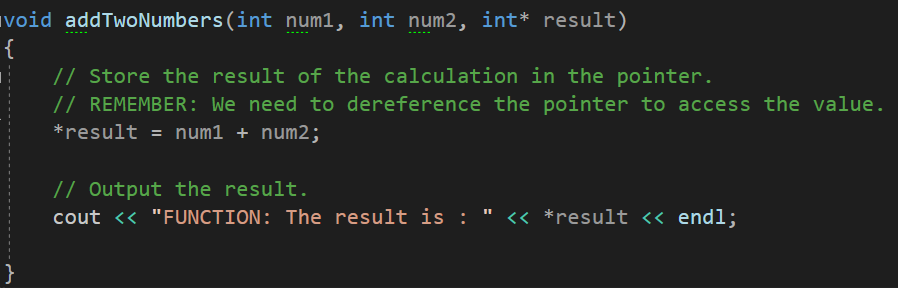
First, we create the prototype for the function above the main:



Next, let’s add code to the main function:



Finally, we add the function body below the main() function. Notice this time we store the result of the calculation in the pointer. It is important to remember to dereference the pointer in order to access the value stored at its address.



Run the program and examine the output to help you understand what is happening.

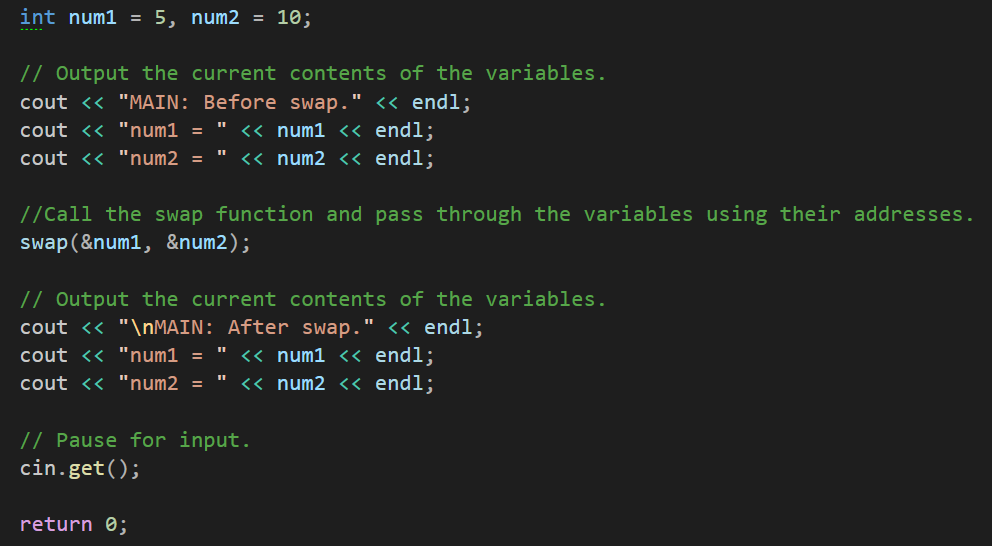
**Example: Swapping Values with Pointers**

This code listing will demonstrate how you can swap the values stored in ordinary variable by accessing their memory address. This will be achieved by passing the addresses of the variable through a function that takes pointers as its parameters.

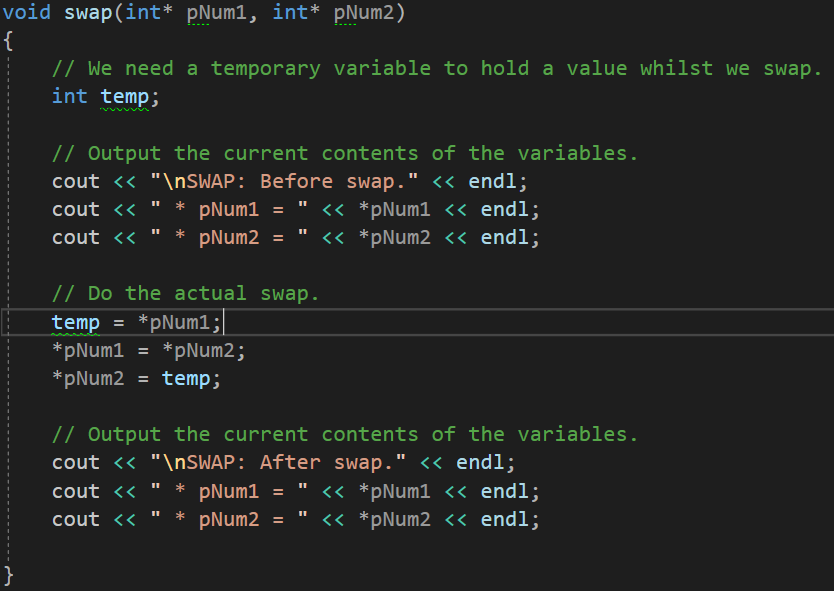
As before, we add a function prototype above the main:



Followed by our main:



And then our function body constructed under the main:



Run the program and examine the output to help you understand what is happening.

**Program 34: Replacing Commas with spaces**

Create a new program that will ask the user to enter some text (be sure to instruct them when adding text, in place of spaces add a comma). This text should then be passed to a function, which will replace any commas found with blank spaces. The function **must** be in the following format with this prototype:



The input text **must** be passed into the function and be changed within this. The altered text should be output to the console screen via the **main** function., i.e do not output the altered text in the comma2blank() function.

This can be achieved in a number of ways, feel free to include additional libraries.

**Program 34: Source code:**

**Program 34: Output Screenshot**

**Program 35: The Changing Pointer**

This might be a challenge for some. Write a program that has integer variables called num1 and num2.

num1 and num2 should have values assigned by request from the user. This **must** be done in a function called inputDetails().This function should have the following format:



Within the function it should ask the user to input two numbers and then populate the variables appropriately.

Hint: create two int variables as normal and store the answers as you have previously, just inside a function. Once you have these values your n1 and n2 need to equal these values…

Within the main() function create a pointer of int data type called pNum and point it to num1. Next call your inputDetails function passing the correct parameters.

Hint: you need to pass addresses…

Write another function called outputDetails() which takes num1, num2 and pNum as parameters.

This function should output the following details to the console screen:

1. num1 value.

2. num1 address in memory.

3. num2 value.

4. num2 address in memory.

5. pNum value (the address it currently holds)

6. pNum dereferenced value.

7. pNum address in memory.

Ensure the output is referring to num1, num2 and pNum and not local copies. This is where the function prototype is crucial. Your output must be clear. I’d recommend outputting some text to explain each of the above.

Back in the main() function call outputDetails. Next you are to reassign the pointer to point at num2 and output the same as above by calling your outputDetails() function. Might be useful to add a message stating the pointe change before calling outputDetails again.

Remember to set the pointer to point at nothing once you have finished with it.

**Program 35: Source Code**

****

**Program 35: Output Screenshot**



**Chapter 13: References**

References are another way of referring to a variable, but with a different name. The way this works is similar to pointers, but without all the messy syntax.

**So, what is a reference?**

A reference is an alias to an existing variable. The reference will store a memory address to the variable it is referring to and enable you to change its contents, unlike a pointer, a reference shares its address with the linked variable, whereas a pointer is its own separate entity. This is particularly useful with regards to functions.

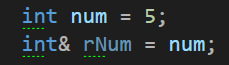
**Why not just use pointers?**

Using pointers can get a little messy at times with all the dereferencing that you must do. References as you will see shortly can be used in the exact same way as you would the original variable. No dereferencing required. The benefit of using a pointer though is that you can reassign it to point at another object of the same data type. This is not the case with a reference. A reference can only be assigned the once and must remain referring to that variable.

The advice should be to use references when you can and use pointers when you must.

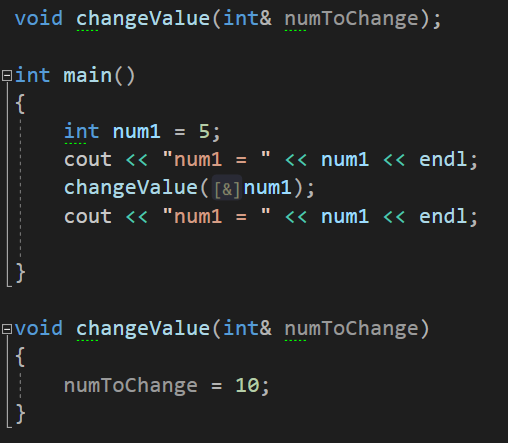
**Example: Declaring and initialising references.**

The only difference in declaration of a reference variable when compared to a normal variable is the use of the’&’ symbol. It must also be set to refer to another variable immediately. See below:



Notice the naming convention; pointers tend to start with a small p, references start with a small r. This isn’t strictly necessary, but it is good practice so when reading code, it is easy to spot which is which. The other obvious change is the use of the & rather than the \* after the data type when declaring one.

**Example: Passing by Reference**



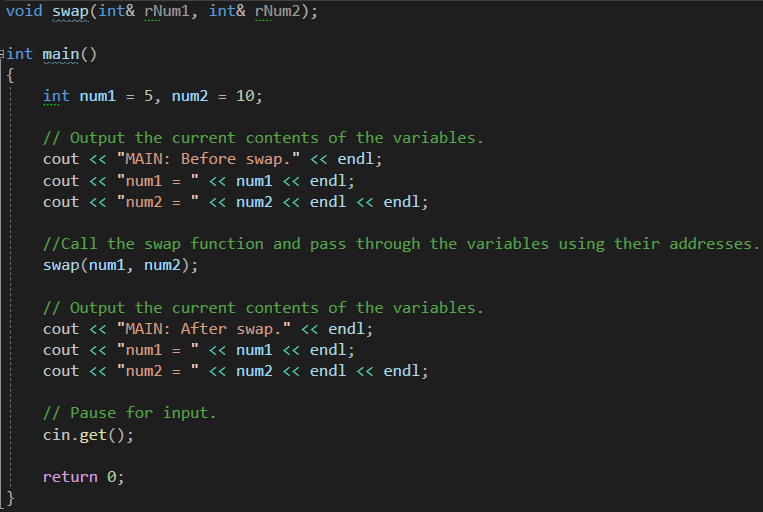
**Output:**

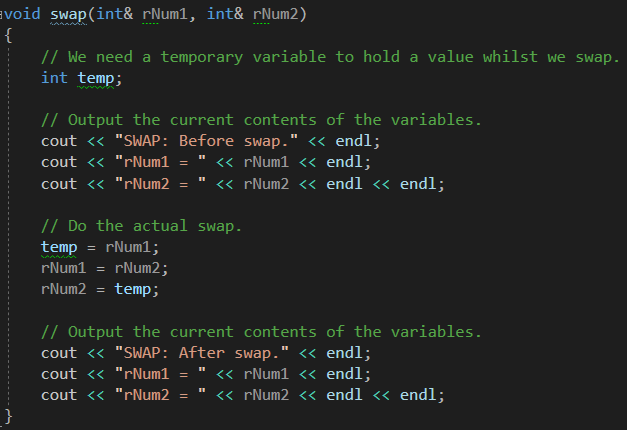


Notice how the reference is used as though it were a normal variable.

**Example: Swapping by Reference**

This code listing will demonstrate how you can swap the values stored in ordinary variable by accessing their memory address. This will be achieved by passing the variable through a function that takes references as its parameters. You used these incidentally in program 35 for the output.





Run the program and examine the output to help you understand what is happening.

**Program 36: Using References**

Write a program that creates an int variable called num and has two functions, one for plus25, one for minus25. Next create a reference to num called rNum; All the below tasks **must** be done on the **reference**, with the output of **num** shown.

1. Ask the user to input a number and store it in num.

2. Output the value of num.

3. Add 25 to the current value stored in num by using rNum in the function.

4. Output num.

5. Ask the user to input another number and store this in num.

6. Output num.

7. Minus 25 from the current value of num by using rNum in the function.

8. Output num.

Remember, only cout num to show changes to rNum affect num.

**Program 36 Source Code:**

**Program 36 Output Screenshot:**

**Program 37: Designated Username**

This program does not require a lot of code; however, it does introduce new functions that will be used throughout your coding career. They are, srand() and rand(), and these are used to generate random numbers. Well, I say random, they aren’t truly random and creating a random device would be better (contact Jay if you would like an example of how to create a random device) but for what we need rand and srand are work perfectly well.

Srand is used to create a *seed*. If the *seed* is set to 1, the generator reinitialized to its original value and produces the same values as before any call to rand. We use time in our case via the **<ctime>** to create a new seed every time the program is run based of the time.

Rand produces a number generated by an algorithm that returns a sequence of apparently non-related numbers each time it is called. This algorithm uses the seed we created with srand to generate the series of numbers.

Now, the program! Create a program that asks the user for a username and then states that their chosen name is taken and provide them with a new one.

Steps:

* Create a function prototype that takes a string reference.
* In main, create a string and ask the user for name which is stored in said string.
* Inform them the name is taken
* Call your function
* And output from main the string username value
  + The function will require the following:
  + 
  + Using the reference, you must add the random numbers to the end of the name
    - Hint: ints don’t join strings without some help…

**Program 37 Source Code:**

**Program 37 Output Screenshot:**