Chapter 10: Pointers

**Pointers** are often seen as the most confusing aspect of C++. This is probably in part due to the fact that 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.

**So, 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 pass through the memory address (pointer) then any changes to the value through this pointer will stay changed.

Lets 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.

12.1 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:

int\* pMyInt;

12.2 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.

pMyInt = new int;

\*pMyInt = 5;

The second line uses the [\*] symbol again to get to the actual value. This is called dereferencing. Put simply, it gives you the value that is stored at the pointers address.

Note: For every *new* keyword used in your program, you must have a matching *delete* keyword.

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.

int myNum = 5;

int\* pPointerToMyNum = &myNum;

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.

Confused? Lets look at a full code example using the above code snippets.

**Program 33: Starting with Pointers**

This code listing will create a variable of *int* type, and then a pointer to point to it. We will output the values stored in each variable, and then use the address [&] & dereference symbol [\*] to mix things up.

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter12\_StartingWithPointers”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “PointerExample.cpp”
5. Replicate program listing 33.

#include <iostream>

using namespace std;

int main()

{

// Create a normal int variable.

int myNum = 5;

// Create a int pointer variable and point it toward the variable just created.

int\* pPointerToMyNum = &myNum;

// Output the values stored in each variable.

cout << “myNum = “ << myNum << endl;

cout << “pPointerToMyNum = “ << pPointerToMyNum << endl;

// Notice that pPointerToMyNum does not output ‘5’. We need to dereference the pointer.

cout << “\*pPointerToMyNum = “ << \*pPointerToMyNum << endl;

// All variables have their own memory addresses. This is how they are stored.

// The next two lines of code show the memory address of myInt and the address that

// pPointerToMyNum points to.

cout << “&myNum = “ << &myNum << endl;

cout << “pPointerToMyNum = “ << pPointerToMyNum << endl;

// To complete the circle lets look at the memory address of pPointerToMyNum.

cout << “&pPointerToMyNum = “ << &pPointerToMyNum << endl;

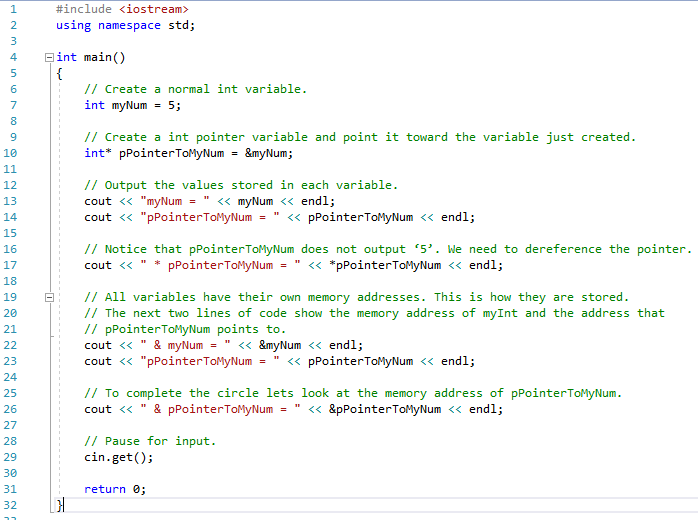
// Pause for input.

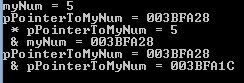
cin.get();

return 0;

}

Program Listing 33





Screenshot 33

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 tat has access to the pointer, which includes passing it into a function.

12.3 Example: Pointer Function parameters

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:

void addTwoNumbers ( int num1, int num2, **int\* result** );

Then we create the main() function, which makes a call to this function.

int main()

{

//Create a pointer that will hold the total.

int\* pResult = new int;

// Call the ‘addTwoNumbers()’ function passing through constant integers.

addTwoNumbers( 10, 5, pResult );

// Output the result.

cout << “MAIN: The result is: “ << \*pResult << endl;

// Free up the memory.

delete pResult;

return 0;

};

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.

void addTwoNumbers (int num1, int num2, int\* result )

{

// Store the result of the calculation in the pointer.

// REMEMBER: We need to dereference the pointer to access the value.

\*result = num1 + num2;

// Output the result.

cout << “FUNCTON: The result is: “ << \*pResult << endl;

};

**Program 34: 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.

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter12\_SwappingWithPointers”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “SwappingValuesWithPointerExample.cpp”
5. Replicate program listing 34.

#include <iostream>

using namespace std;

// Prototype for the swap function.

void swap( int\* pNum1, int\* pNum2);

int main()

{

int num1 = 5, num2 = 10;

// Output the current contents of the variables.

cout << “MAIN: Before swap.” << endl;

cout << “num1 = “ << num1 << endl;

cout << “num2 = “ << num2 << endl << endl;

//Call the swap function and pass through the variables using their addresses.

swap( &num1, &num2 );

// Output the current contents of the variables.

cout << “MAIN: After swap.” << endl;

cout << “num1 = “ << num1 << endl;

cout << “num2 = “ << num2 << endl << endl;

// Pause for input.

cin.get();

return 0;

}

void swap( int\* pNum1, int\* pNum2 )

{

// We need a temporary variable to hold a value whilst we swap.

int temp;

// Output the current contents of the variables.

cout << “SWAP: Before swap.” << endl;

cout << “\*pNum1 = “ << \*pNum1 << endl;

cout << “\*pNum2 = “ << \*pNum2 << endl << endl;

// Do the actual swap.

temp = \*pNum1;

\*pNum1 = \*pNum2;

\*pNum2 = temp;

// Output the current contents of the variables.

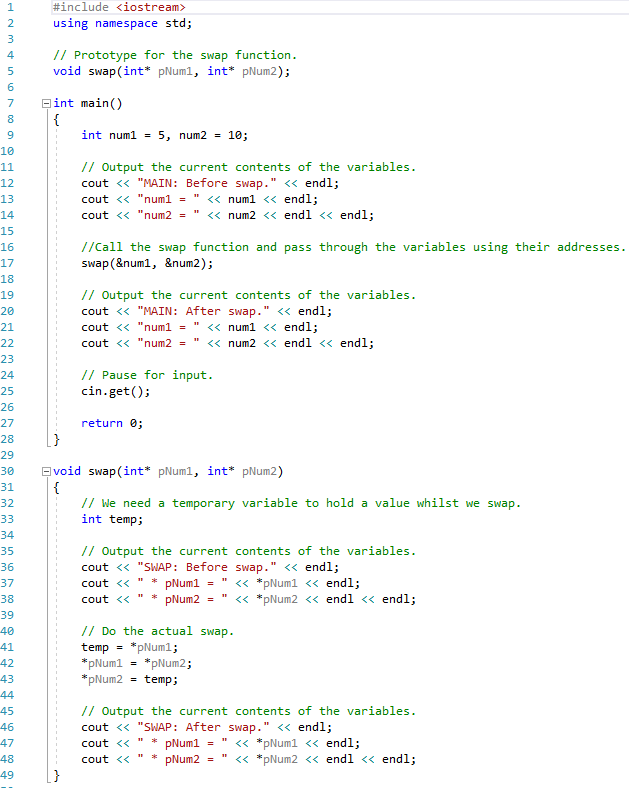
cout << “SWAP: After swap.” << endl;

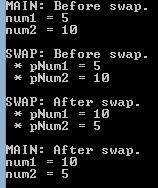
cout << “\*pNum1 = “ << \*pNum1 << endl;

cout << “\*pNum2 = “ << \*pNum2 << endl << endl;

}

Program Listing 34





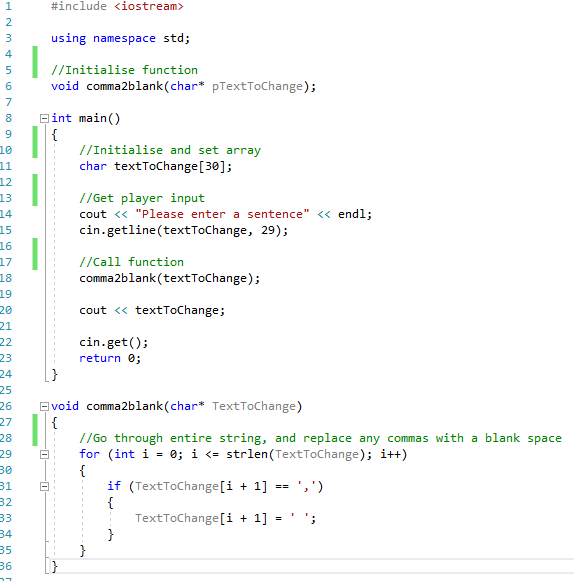
Screenshot 34

**Program 35: Replacing Commas with spaces**

Write a program that will ask the user to enter some text. 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:

void comma2blank( char\* textToChange );

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.





**Program 36: The changing pointer**

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:

void inputDetails( int\* n1, int\* n2 );

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

Within the main() function create a pointer to an int data type called pNum and point it to num1.

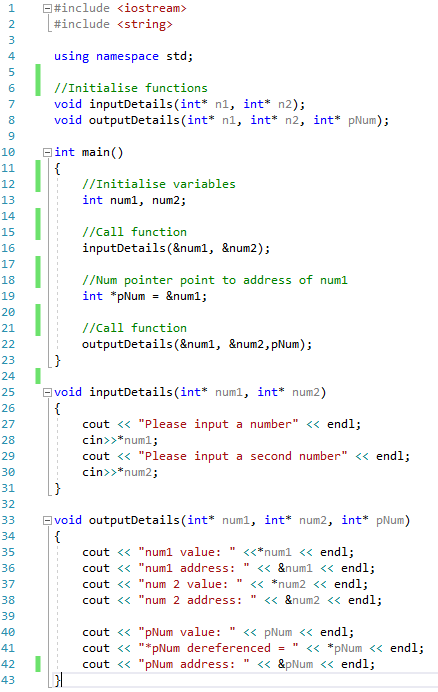
Write another function called outputDetails() which takes num1, num2 and pNum as parameters. This function should output the following details to the console screen:

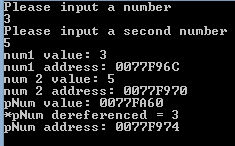
1. num1 value.
2. num1 address in memory.
3. num2 value.
4. num2 address in memory.
5. pNum value (the address it currently holds)
6. pNum dereferenced value.
7. pNum address in memory.

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 reassign the pointer to point at num2 and output the same as above by calling your outputDetails() function.

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





Chapter 11: 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. 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 have to 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 have to.

13.1 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:

int num = 5;

int& rMyInt;

13.2 Example: Pass by reference

References really become useful when using them with functions. The reference within the function will be referring to the original and will therefore allow you to change its value.

#include <iostream>

using namespace std;

void changeValue( int& numToChange )

{

numToChange = 10;

}

int main()

{

int num1 = 0;

cout << “num1 = “ << endl;

changeValue( num1 );

cout << “num1 = “ << endl;

return 0;

}

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

**Program 37: Swapping Values with References**

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.

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter13\_SwappingWithReferences”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “SwappingValuesWithReferencesExample.cpp”
5. Replicate program listing 37.

#include <iostream>

using namespace std;

// Prototype for the swap function.

void swap( int& rNum1, int& rNum2);

int main()

{

int num1 = 5, num2 = 10;

// Output the current contents of the variables.

cout << “MAIN: Before swap.” << endl;

cout << “num1 = “ << num1 << endl;

cout << “num2 = “ << num2 << endl << endl;

//Call the swap function and pass through the variables using their addresses.

swap( num1, num2 );

// Output the current contents of the variables.

cout << “MAIN: After swap.” << endl;

cout << “num1 = “ << num1 << endl;

cout << “num2 = “ << num2 << endl << endl;

// Pause for input.

cin.get();

return 0;

}

void swap( int& rNum1, int& rNum2 )

{

// We need a temporary variable to hold a value whilst we swap.

int temp;

// Output the current contents of the variables.

cout << “SWAP: Before swap.” << endl;

cout << “rNum1 = “ << rNum1 << endl;

cout << “rNum2 = “ << rNum2 << endl << endl;

// Do the actual swap.

temp = rNum1;

rNum1 = rNum2;

rNum2 = temp;

// Output the current contents of the variables.

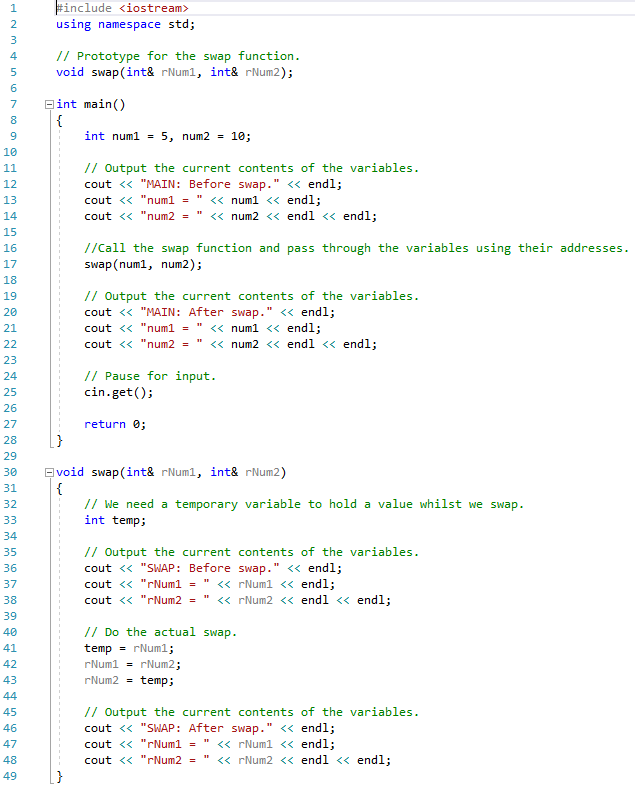
cout << “SWAP: After swap.” << endl;

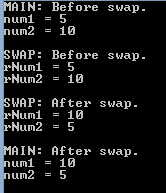
cout << “rNum1 = “ << rNum1 << endl;

cout << “rNum2 = “ << rNum2 << endl << endl;

}

Program Listing 37





Screenshot 37

**Program 38: Using References**

Write a program that creates an int variable called num. 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.
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.
8. Output num.

