Programming 1: Tutorial 2

Using Visual Studio

It is important that you know how to use visual studio, it has a great many features, and some are useful some are not. The more you know however the easier it becomes to complete projects. Unfortunately there isn't enough time in the day for us to teach you the intricacies of Visual Studio so here is a guide that you're welcome to look at:
http://docs.msdnaa.net/tools/studenttoolsguidehtm/

This link takes you to a tutorial set that provides a great introduction to many of Visual Studio .NET's features.
If you wish to become extremely savvy with Visual Studio .NET then check this site out. They will also provide an excellent addition to these tutorials, so if you have any spare time, check them out.

Part 1: Properties

Algorithms should have the following properties:

- Finiteness – Algorithm must complete after a finite number of instructions have been executed
- Absence of Ambiguity – Each step must be clearly defined having only one interpretation
- Definition of Sequence – Each step must have a unique defined preceding and succeeding step. The first step (start step) and last step (halt step) must be clearly noted
- Feasibility – All instructions must be able to be performed. Illegal operations (e.g. division by 0) are not allowed.
- Input – 0 or more data values
- Output – 1 or more results

1. Input a real Number X.
2. If (X < 0) Then
    a. Output: X " cannot be negative."
    b. STOP.

   EndIf
3. Set a variable, S, to 10.
4. Set a counter variable, I, to 3.
5. While (I > 0) Do
    a. Calculate the value (S + X / S) / 2.0.
    b. Set S to the value calculated in step 5a.
    c. Subtract 1 from I.

EndWhile
6. Output: “The square root of “ X “ is about “ S.
7. STOP.

Question: Does this possess the properties that an algorithm should?

Part 2: Design and Pseudo-Code

I want you to design a program (write your answer in pseudo-code) for a program that prompts for the users age, reads in there answer and then prints it out to the screen.

To help you here is the pseudo-code for last week’s program:

1. Print “Hello World” to Screen
2. STOP.

Now write you answer for the problem in pseudo-code below:

Part 3: Write the program

Here is the hard part; I want you to attempt to code the program.
It is going to look a lot like last weeks:

```
/* Hello World C++ Program */

#include <iostream>

int main()
{
  std::cout << "Hello World! \n";

  return 0;
}
```

But with some added extras in the main function, hopefully you'll know the order in which they go due to part 2.

Currently you don't posses all the required information to complete the task so here it is, don't forget to read it all first though:

`std::cout <<` Prints to the screen
`std::cin >>` Reads in data that has been entered by the user

That's not quite it though, because we need some where to store our information. If you look at the `std::cout << "Hello World! \n";` statement we have some information after the << so it follows that we're going to need something after the >> that belongs to `std::cin`. Not only that but if we're reading something to and don't tell the computer where we're storing the information it's going to get upset – remember we need to be very explicit when programming.

Storing Information

For this we need something called a variable – we use variables to store information.
We declare a variable in our code like this:

```
int age;  // note that the – int – should turn blue
```

We declare them in our blocks of code:

```
int main()
{
  // block of code between the { and } brackets

  return 0;
}
```

We then need to read the data into our variable, which is called, age:

```
std::cin >> age;
```

Having a go
Now its your turn to try to write the program, don't worry if you struggle just put your hand up and I'll come help.

**remember after every programming statement to put semi-colons or you will generate errors*

When things go bad: Programming Errors and Debugging

Unfortunately when programming things often go wrong and rare is the occasion when your code compiles correctly first time round.
Because of this the ability to problem solve and debug your code is a very important skill.

In part of today's tutorial we are going to explore errors and how to solve them.

Types of Error

What is an Error
An error typically refers to a problem that exists in a program when it is written and compiled.

There are four main types of error:

1) Language Syntax (Compilation) Errors
2) Linking Error
3) Runtime (Execution) Error
4) Logic Errors

There are also Warnings, these are not really errors but are included here for the sake of completeness. Warnings are given by the compiler do indicate a potential error that may occur in your code. They can often avert runtime errors.

Compilers, as we have listed above, often fail to compile when errors are detected.
The problem is that when they return an error the hints on what the error is, and how to fix it, can be vague. If they could do all of this then they could probably code as well ☺
It is because of this that it is good to not only now to fix errors but also how to interpret the compilers message.

Looking at Errors

We are going to start looking at errors, this week we are looking only at Syntax Errors.

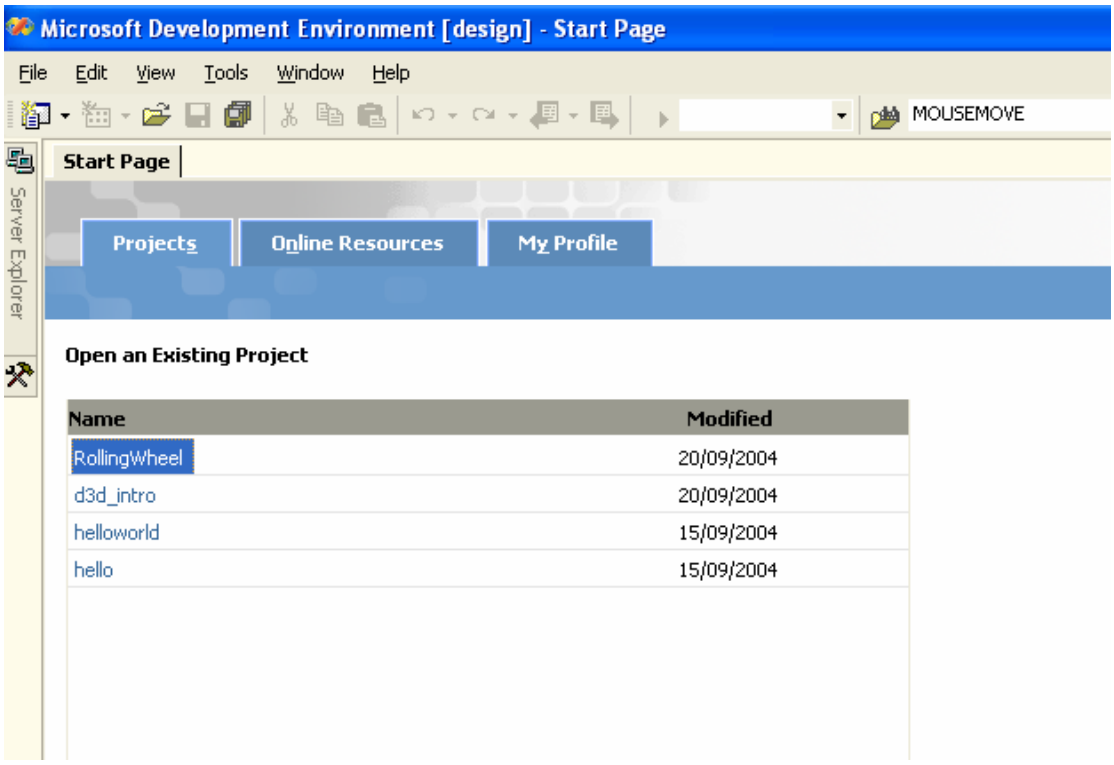
Language Syntax (Compilation) Errors

- The most common kind of error
- Error is in the form of the statement
  - Misspelled word
  - Unmatched parenthesis
  - Comma out of place
  - Missing a semicolon from the end of a statement
- Error is detected by the compiler (at compile time)
- Compiler cannot correct error so no object file is generated
- Compiler prints error messages but continues to compile

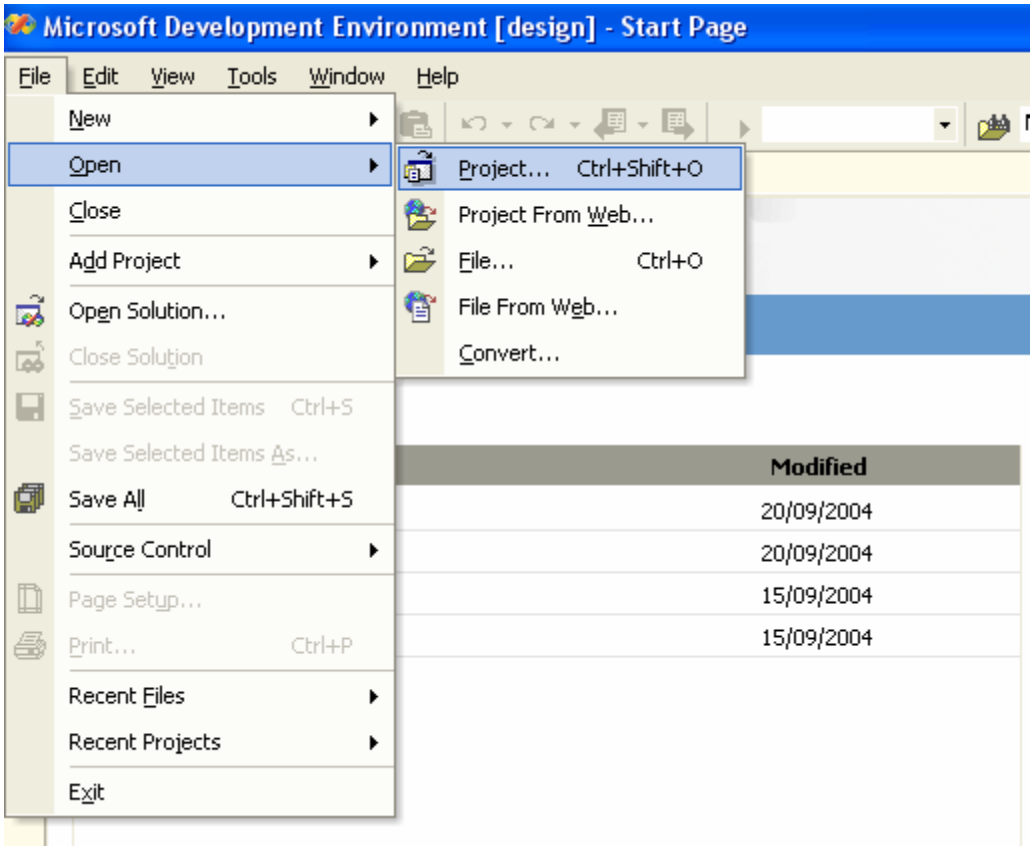
Purposely Coding Errors

Now we are going to input some syntax code, to generate errors. This will give you an idea of how recognise and debug the code.

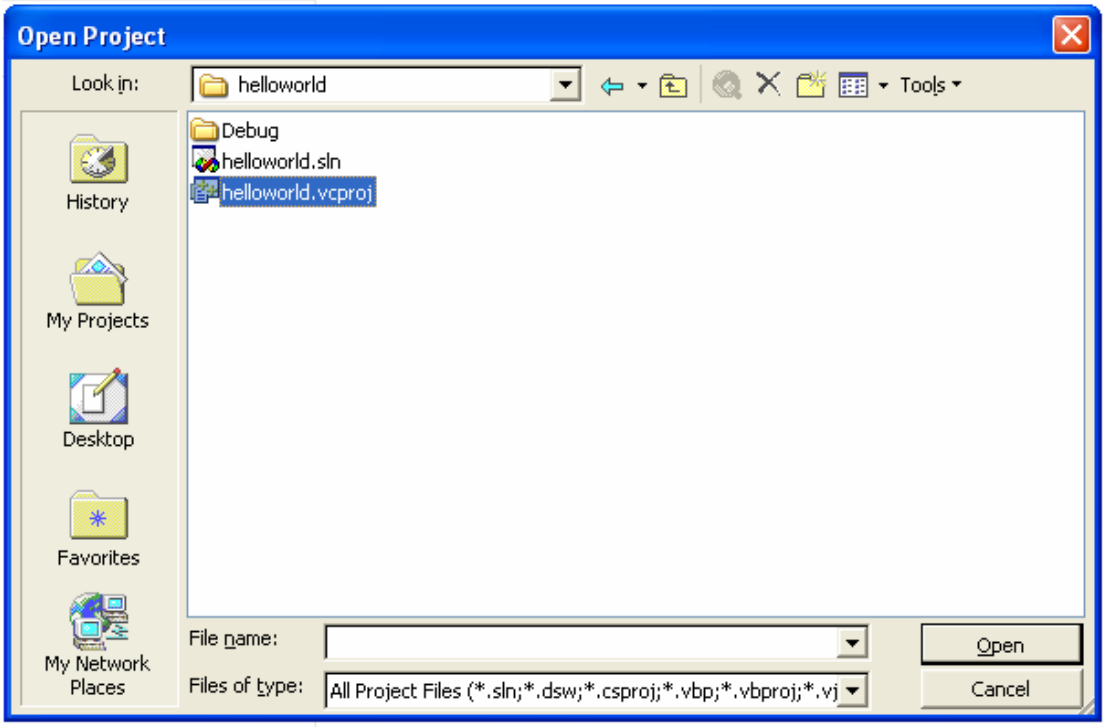
So first things first, open up Visual Studio .NET
Hopefully you will see something similar to what is below. With visual studio displaying your projects. If you don't then don't worry, follow the step below this image.



If you haven't seen the above image then simply click File | Open | Project or press Ctrl+Shift+O



This should open a browser box similar to the one below.



Simply find where you stored your project last week and double click on the .vcproj icon. That should open your project work space.
Hopefully you remember where you stored your project last week. If you deleted, can't remember or weren't here last week then see last weeks tutorial (located on www.activehelix.com/proj1).

Now finally down to coding.
We are going to start with the second code example we used last week:

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
	string name;

	cout << "What is your name:\n";
	cin >> name;
	cout << "\nWelcome to Salford Uni "
		<< name << ", hope you enjoy it"
		<< endl;

	return 0;
}
```

The code above is what we used last week. If you make sure it compiles correctly. If you have a document open you can copy and paste it into visual

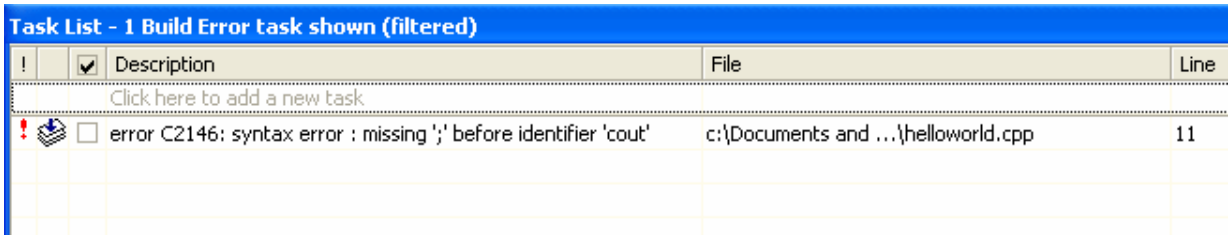
studio it should work (this isn't normally recommended as word and adobe uses formatted text – this is bad practice, but just this once).

Syntax Error

Now remove the semicolon (;) from the end of the `string name` statement. Recompile. You should generate an error. The computer will identify this to you via the following message box:

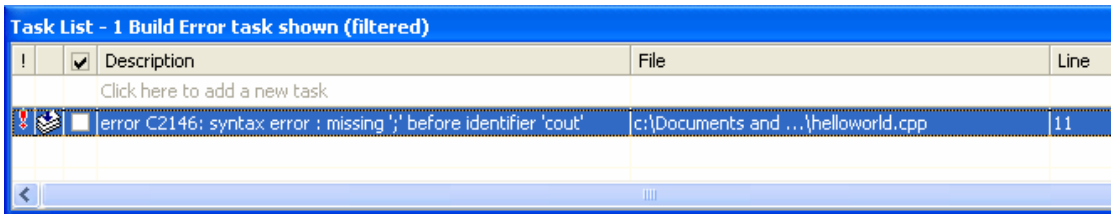
You now press the **No** button.

At the bottom of visual studio you should now see the following:

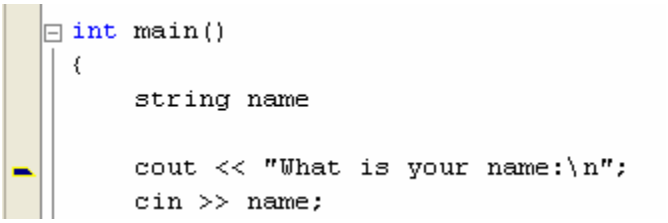


This tells you that there is a syntax error. In this case it even tells you what it is. It tells you the file where the error is located, necessary if you are working with multiple files, and the line where the error is.

By double clicking the error description the cursor will go to near where the error is located.



Visual Studio will also add a little mark where the error was detected. Please look closely though.



```
int main()
{
    string name

    cout << "What is your name:\n";
    cin >> name;
```

Hopefully you have seen that the actual error pointer is directing you towards the line with `cout` on. This is obviously not where the error lies. This is because the compiler expects a semicolon after the `string name` statement and doesn't report an error until it can't find one, in this case because we moved on to the next chunk of code.
In the case of a missing semicolon it is best to work backwards from the beginning of the indicated line until you see the missing semicolon.

You can now try a different syntax error.
Add the semicolon to its correct place and now try removing other parts of the code. Try the following:

```
int main( // notice the missing parenthesis ‘)’
```

This should generate a - syntax error: missing ‘)’ before ‘{‘

```
      return 0;
// notice the missing parenthesis ‘}’ that should go here
```

This should generate a - fatal error end of file found before the left brace ‘{‘

```
<< name ", hope you enjoy it"
/* notice the missing parenthesis << that should go between name and “,
hope.. */
```

This will actually generate 3 errors –

- Syntax error: missing ‘;’ before ‘string’
- Mismatch in formal parameter list
- ‘<<’: unable to resolve function overload

3 Errors! That's a lot for one missing operator. Luckily they all point to the line where the error is or just below it enabling you to examine your code in the correct place and detect that you need a << after `name`.

You should take heart from this. If you get many errors when compiling your own code it could just be a few syntax errors in your code causing the compiler to tell you that there are lots of things wrong.

Hopefully these should give you an idea of the sorts for syntax error you can face and how to deal with them effectively.
If you wish to experiment in your own time then feel free. There are many possible syntax errors to explore.