

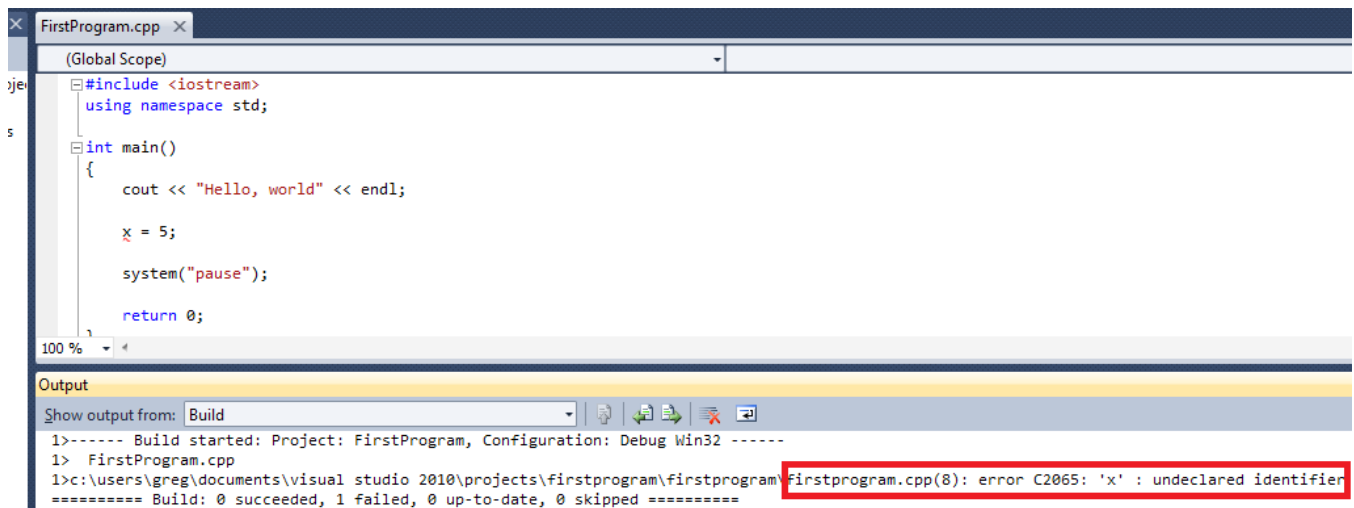
Common Beginner C++ Programming Mistakes

This documents some common C++ mistakes that beginning programmers make.

These errors are two types:

Syntax errors – these are detected at compile time and you won't be able to run your program until these are fixed. Listed are the errors that you will see if you are using the Visual Studio C++ compiler.

Syntax errors will be displayed on the "Output" window. Here is a screen-shot with an error shown circled in red:



```
FirstProgram.cpp
(Global Scope)
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, world" << endl;

    x = 5;

    system("pause");

    return 0;
}

Output
Show output from: Build
1>----- Build started: Project: FirstProgram, Configuration: Debug Win32 -----
1> FirstProgram.cpp
1>c:\users\greg\documents\visual studio 2010\projects\firstprogram\firstprogram\firstprogram.cpp(8): error C2065: 'x' : undeclared identifier
----- Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped -----
```

Logic errors – these are not compile errors, so your program will compile and run. However, it most likely won't give you the correct results. These can be difficult to track down.

Each common mistake shows an example of the error and the correction (in red) that will fix the mistake.

These examples came from several resources, including <http://www.cprogramming.com/tutorial/common.html>

Common Syntax Errors (detected at compile time)

1. Undeclared Variables

```
int main()
{
    cin >> x;
    cout << x;
}
```

"Huh? Why do I get this error?"

'x' : undeclared identifier

The compiler doesn't know what x means. You need to declare it as a variable. (The 'x' will be replaced with your specific variable name.)

```
int main()
{
    int x;
    cin >> x;
    cout << x;
}
```

Another common issue for undeclared variables/functions is misspelling a variable/function name or using the case of the letters inconsistently. Remember, C++ is case sensitive, so

```
int Main() is not the same as
int main()
```

In Visual Studio, if you enter Main() instead of main(), you get the following linker error:

error LNK2019: unresolved external symbol _main referenced in function ___tmainCRTStartup

2. Undeclared Functions

```
int main()
{
    menu();
}
void menu()
{
    //...
}
```

"Why do I get an error about menu being unknown?"

error C3861: 'menu': identifier not found

The compiler doesn't know what `menu()` stands for until you've told it, and if you wait until after using it to tell it that there's a function named `menu`, it will get confused. Always remember to put either a prototype for the function or the entire definition of the function above the first time you use the function.

```
void menu();

int main()
{
    menu();
}
void menu()
{
    ...
}
```

3. Missing semicolons

```
int main()
{
    int x;
    cin >> x
    cout << x;
}
```

"Huh? Why do I get an error?"

error C2146: syntax error : missing ';' before identifier 'cout'

The compiler sees the cin and cout lines as one line of code, since there is only one semicolon. In larger programs, a single missing semicolon can generate multiple cascading errors. Fix the first occurrence of it and recompile before wasting time trying to track down extraneous errors. Note that in your specific program, "cout" in the above example will be replaced with the first word from the statement following the one in your program with the missing ;

```
int main()
{
    int x;
    cin >> x;
    cout << x;
}
```

4. Extra semicolons

```
void menu();

int main()
{
    //...
    menu();
}

void menu();
{
    //...
}
```

"Huh? Why do I get an error?"

error C2447: '{' : missing function header (old-style formal list?)

Semicolons go at the end of complete statements. The following are not complete statements, so they don't get a semicolon:

Function declarations

#include lines

if lines

switch lines

Some of these will cause syntax errors, some will cause logic errors.

```
void menu();

int main()
{
    //...
    menu();
}

void menu() ← removed ;
{
    //...
}
```

5. Incorrect number of braces

```
int main()
{
    int x;
    cin >> x;
    if (x == 5)
    {
        cout << x;
    }
}
```

"Huh? Why do I get an error?"

fatal error C1075: end of file found before the left brace '{' ...

Braces are the "begin" and "end" around blocks of code. You must have a } for every {. Missing (or extra) braces can lead to a cascading number of errors.

Hint: Fix the braces issue and recompile before chasing down extraneous errors.

```
int main()
{
    int x;
    cin >> x;
    if (x == 5)
    {
        cout << x;
    }
}
```

Logic Errors (detected at run time)

1. Uninitialized variables

```
int count;  
while (count < 100)  
{  
    cout << count;  
    count++;  
}
```

"Why doesn't my program enter the while loop?"

In C++ variables are not initialized to zero. In the above snippet of code, `count` could be any value in the range of `int`. It might, for example, be 586, and in that situation the while loop's condition would never be true. Perhaps the output of the program would be to print the numbers from -1000 to 99. In that case, once again, the variable was assigned a memory location with garbage data that happened to evaluate to -1000.

Visual Studio will give you the following warning, but will still let the compile succeed and will let you run the program. You should strive to fix all warnings in addition to all errors.

warning C4700: uninitialized local variable 'count' used

Remember to initialize your variables.

```
int count = 0;  
while (count < 100)  
{  
    cout << count;  
    count++;  
}
```

2. Setting a variable to an uninitialized value

```
int a, b;  
int sum = a + b;  
cout << "Enter two numbers to add: ";  
cin >> a;  
cin >> b;  
cout << "The sum is: " << sum;
```

When Run:

```
Enter two numbers to add: 1 3  
The sum is: -1393
```

"What's wrong with my program?"

Often beginning programmers believe that variables work like equations - if you assign a variable to equal the result of an operation on several other variables that whenever those variables change (a and b in this example), the value of the variable will change. In C++ assignment does not work this way: it's a one shot deal. Once you assign a value to a variable, it's that value until you reassign the values. In the example program, because a and b are not initialized, sum will equal an unknown random number, no matter what the user inputs.

Visual Studio will give you a warning about referencing uninitialized variables. Heed the warning!

To fix this error, move the addition step after the input line.

```
int a, b;  
int sum;  
cout << "Enter two numbers to add: ";  
cin >> a;  
cin >> b;  
sum = a + b;  
cout << "The sum is: " << sum;
```


3. Using a single equal sign to check equality

```
char done = 'Y';  
while (done = 'Y')  
{  
    //...  
    cout << "Continue? (Y/N)";  
    cin >> done;  
}
```

"Why doesn't my loop ever end?"

If you use a single equal sign to check equality, your program will instead assign the value on the right side of the expression to the variable on the left hand side, and the result of this statement is TRUE. Therefore, the loop will never end. Use == to check for equality.

```
char done = 'Y';  
while (done == 'Y')  
{  
    //...  
    cout << "Continue? (Y/N)";  
    cin >> done;  
}
```

4. Extra Semicolons

```
int x;  
for (x = 0; x < 100; x++);  
    cout << x;
```

"Why does it just output 100?"

You put in an extra semicolon. Remember, semicolons don't go after if statements, loops, or function definitions. If you put one in any of those places, your program will function improperly.

```
int x;  
for (x = 0; x < 100; x++) ← removed ;  
    cout << x;
```

This is also a common mistake with a `while` statement, an `if` statement, and a `switch` statement.

5. Forgetting a break in a switch statement

```
int x = 2;
switch(x)
{
    case 2:
        cout << "two" << endl;

    case 3:
        cout << "three" << endl;
}
```

"Why does it print two and three?"

Remember that C++ does not break out of a switch statement when a case is encountered. It only breaks out when it hits the `break;` statement.

```
int x = 2;
switch(x)
{
    case 2:
        cout << "two" << endl;
        break;

    case 3:
        cout << "three" << endl;
        break; // in case more cases are added later
}
```

6. Overstepping array boundaries

```
int array[10];  
//...  
for (int x = 1; x <= 10; x++)  
    cout << array[x];
```

"Why doesn't it output the correct values?"

Arrays begin indexing at 0; they end indexing at length-1. For example, if you have a ten element array, the first element is at position zero and the last element is at position 9.

```
int array[10];  
//...  
for (int x = 0; x < 10; x++)  
    cout << array[x];
```

7. Misusing the && and || operators

```
int value;
do
{
    //...
    value = 10;
} while(!(value == 10) || !(value == 20))
```

"Huh? Even though value is 10 the program loops. Why?"

Consider the only time the while loop condition could be false: both `value==10` and `value==20` would have to be true so that the negation of each would be false in order to make the `||` operation return false. In fact, the statement given above is a tautology; it is always true that value is not equal to 10 or not equal to 20 as it can't be both values at once. Yet, if the intention is for the program only to loop if value has neither the value of ten nor the value of 20, is necessary to use `&&` : `!(value==10) && !(value==20)`, which reads much more nicely: "if value is not equal to 10 and value is not equal to 20", which means if value is some number other than ten or twenty (and therein is the mistake the programmer makes - he reads that it is when it is "this" or "that", when he forgets that the "other than" applies to the entire statement "ten or twenty" and not to the two terms - "ten", "twenty" - individually). A quick bit of boolean algebra will help you immensely: `!(A || B)` is the equivalent of `!A && !B` (Try it and see). The sentence "value is other than [ten or twenty]" (brackets added to show grouping) is translatable to `!(value==10 || value==20)`, and when you distribute the `!`, it becomes `!(value==10) && !(value==20)`.

The proper way to rewrite the program:

```
int value;
do
{
    //...
    value = 10;
} while (!(value == 10) && !(value == 20))
```