C++ Beginner's Mistakes

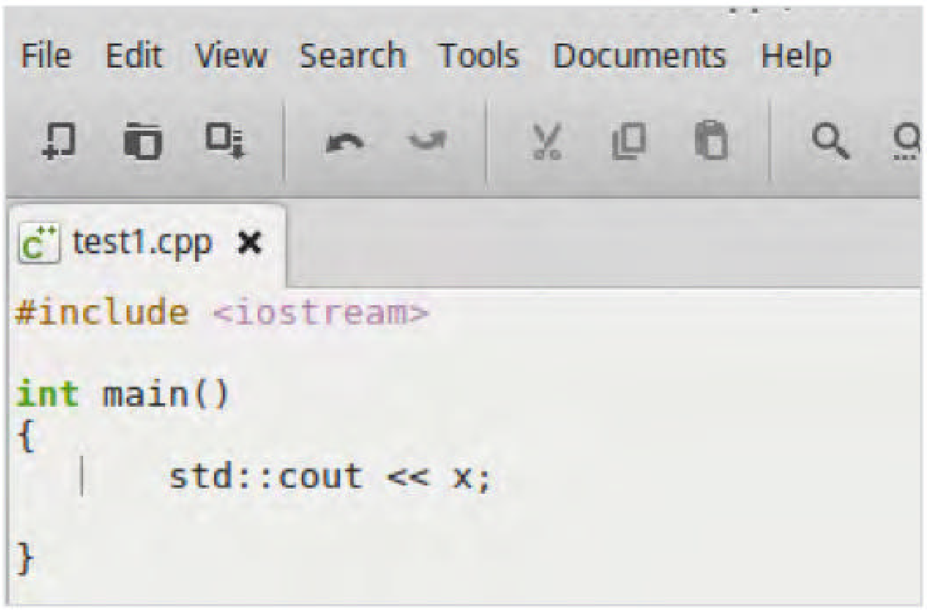
There are many pitfalls the C++ developer can encounter, especially as this is a more complex and often unforgiving language to master. Beginners need to take C++ a step at a time and digest what they've learned before moving on.

VOID(C++, MISTAKES)

Admittedly it's not just C++ beginners that make the kinds of errors we outline on these pages, even hardened coders are prone to the odd mishap here and there. Here are some common issues to try and avoid.

UNDECLARED IDENTIFIERS

A common C++ mistake, and to be honest a common mistake with most programming languages, is when you try and output a variable that doesn't exist. Displaying the value of x on-screen is fine but not if you haven't told the compiler what the value of x is to begin with.



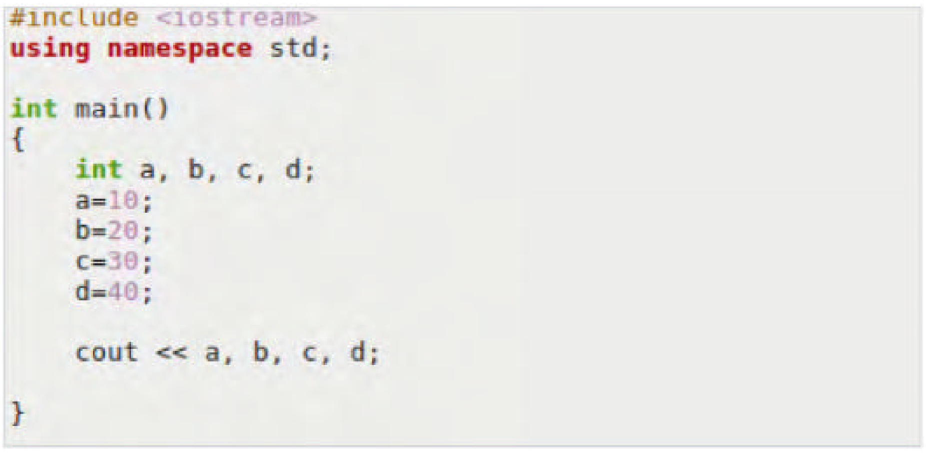
```
#include <iostream>

int main()
{
	std::cout << x;

}
```

STD NAMESPACE

Referencing the Standard Library is common for beginners throughout their code, but if you miss the std:: element of a statement, your code errors out when compiling. You can combat this by adding:

```
using namespace std;
```

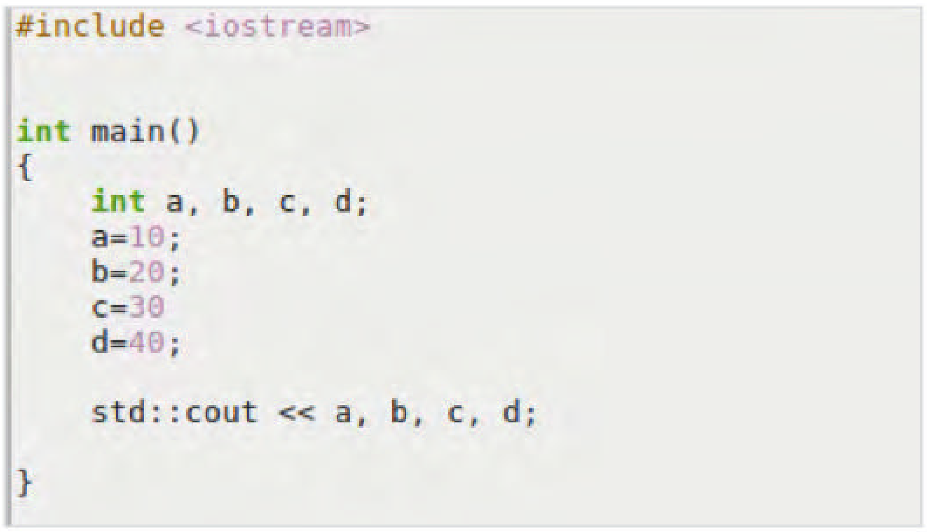
Under the #include part and simply using cout, cin and so on from then on.



```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30;
    d=40;

    cout << a, b, c, d;
}
```

SEMICOLONS

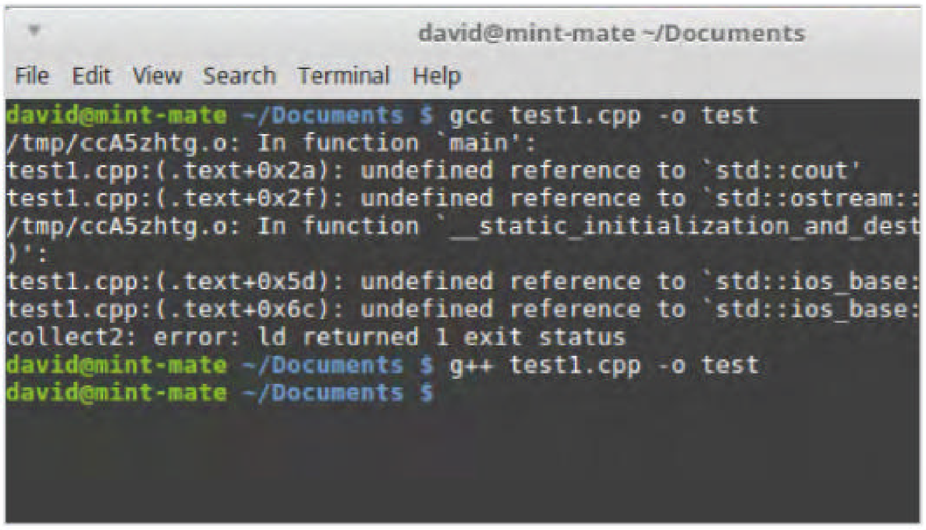
Remember that each line of a C++ program must end with a semicolon. If it doesn't then the compiler treats the line with the missing semicolon as the same line with the next semicolon on. This creates all manner of problems when trying to compile, so don't forget those semicolons.



```
#include <iostream>


int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30
    d=40;

    std::cout << a, b, c, d;

}
```

GCC OR G++

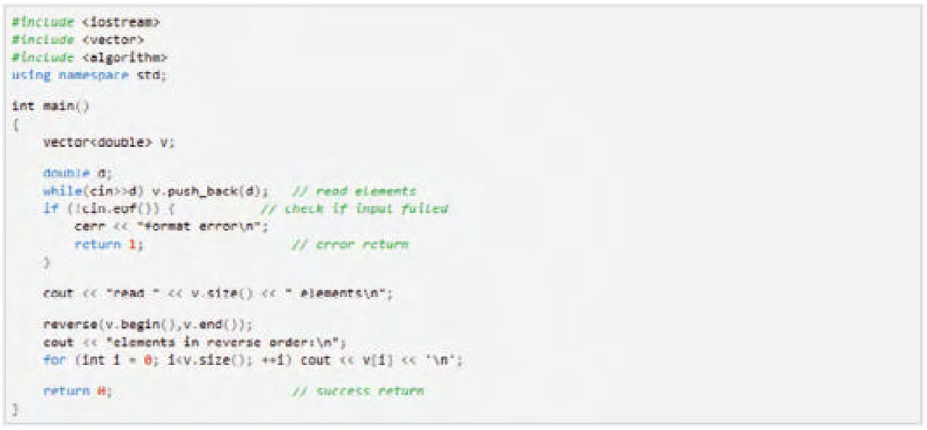
If you're compiling in Linux then you will no doubt come across gcc and g++. In short, gcc is the Gnu Compiler Collection (or Gnu C Compiler as it used to be called) and g++ is the Gnu ++ (the C++ version) of the compiler. If you're compiling C++ then you need to use g++, as the incorrect compiler drivers will be used.



```
david@mint-mate ~/Documents $ gcc test1.cpp -o test
/tmp/ccA5zhtg.o: In function `main':
test1.cpp:(.text+0x2a): undefined reference to `std::cout'
test1.cpp:(.text+0x2f): undefined reference to `std::ostream::
/tmp/ccA5zhtg.o: In function `__static_initialization_and_dest
)':
test1.cpp:(.text+0x5d): undefined reference to `std::ios_base:
test1.cpp:(.text+0x6c): undefined reference to `std::ios_base:
collect2: error: ld returned 1 exit status
david@mint-mate ~/Documents $ g++ test1.cpp -o test
david@mint-mate ~/Documents $
```

COMMENTS (AGAIN)

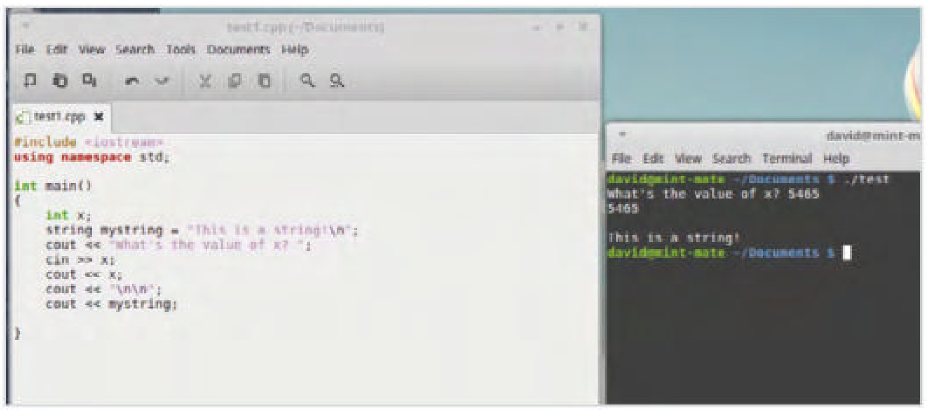
Indeed the mistake of never making any comments on code is back once more. As we've previously bemoaned, the lack of readable identifiers throughout the code makes it very difficult to look back at how it worked, for both you and someone else. Use more comments.



```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    vector<double> v;

    double d;
    while(cin>>d) v.push_back(d);   // read elements
    if (!cin.eof()) {             // check if input failed
        cerr << "format error\n";
        return 1;                 // error return
    }

    cout << "read " << v.size() << " elements\n";

    reverse(v.begin(),v.end());
    cout << "elements in reverse order:\n";
    for (int i = 0; i<v.size(); ++i) cout << v[i] << '\n';

    return 0;                     // success return
}
```

QUOTES

Missing quotes is a common mistake to make, for every level of user. Remember that quotes need to encase strings and anything that's going to be outputted to the screen or into a file, for example. Most compilers errors are due to missing quotes in the code.


test1.cpp (~/Documents)
File Edit View Search Tools Documents Help
test1.cpp
david@mint-m
File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    cout << "\n\n";
    cout << mystring;

}
```

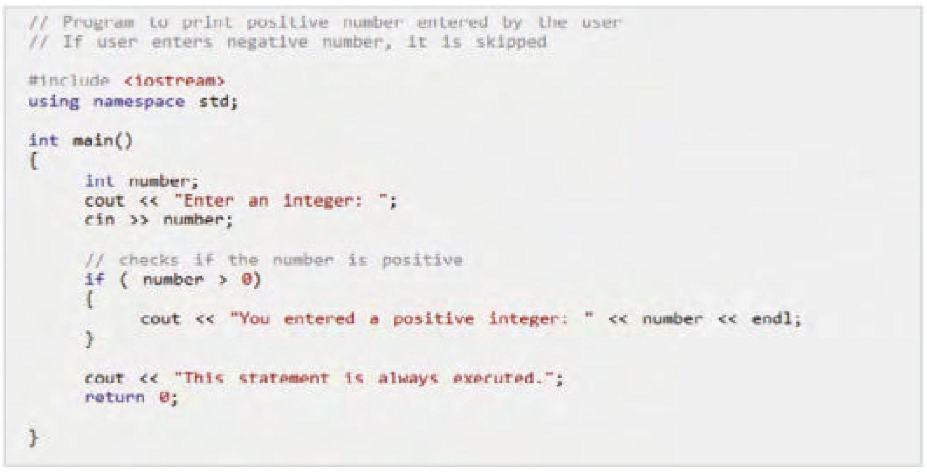
```
david@mint-mate ~/Documents $ ./test
What's the value of x? 5465
5465

This is a string!
david@mint-mate ~/Documents $
```

EXTRA SEMICOLONS

While it's necessary to have a semicolon at the end of every C++ line, there are some exceptions to the rule. Semicolons need to be at the end of every complete statement but some lines of code aren't complete statements. Such as:

```
#include
if lines
switch lines
```

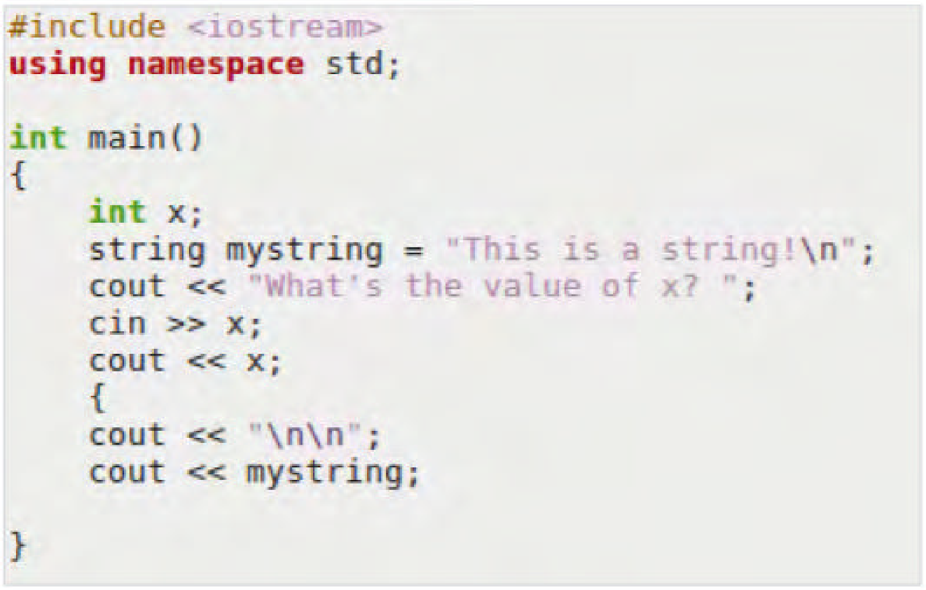
If it sounds confusing don't worry, the compiler lets you know where you went wrong.



```
// Program to print positive number entered by the user
// If user enters negative number, it is skipped

#include <iostream>
using namespace std;

int main()
{
     int number;
     cout << "Enter an integer: ";
     cin >> number;

     // checks if the number is positive
     if ( number > 0)
     {
          cout << "You entered a positive integer: " << number << endl;
     }

     cout << "This statement is always executed.";
     return 0;

}
```

TOO MANY BRACES

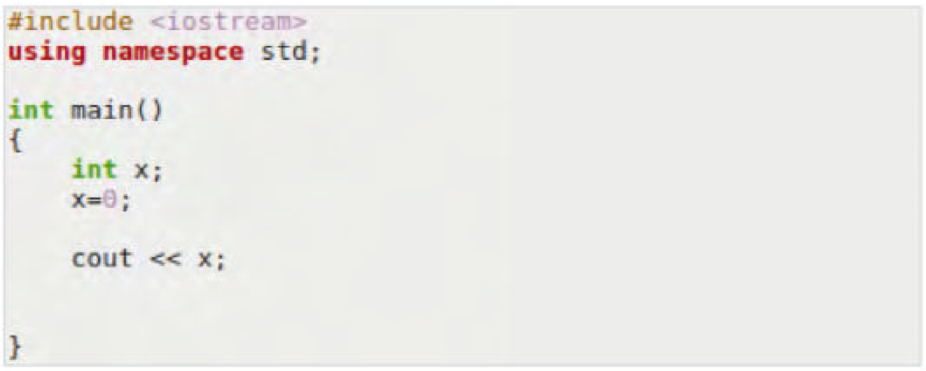
The braces, or curly brackets, are beginning and ending markers around blocks of code. So for every { you must have a }. Often it's easy to include or miss out one or the other facing brace when writing code; usually when writing in a text editor, as an IDE adds them for you.



```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    {
    cout << "\n\n";
    cout << mystring;

}
```

INITIALISE VARIABLES

In C++ variables aren't initialised to zero by default. This means if you create a variable called x then, potentially, it is given a random number from 0 to 18,446,744,073,709,551,616, which can be difficult to include in an equation. When creating a variable, give it the value of zero to begin with: `x=0`.



```
#include <iostream>
using namespace std;

int main()
{
    int x;
    x=0;

    cout << x;

}
```

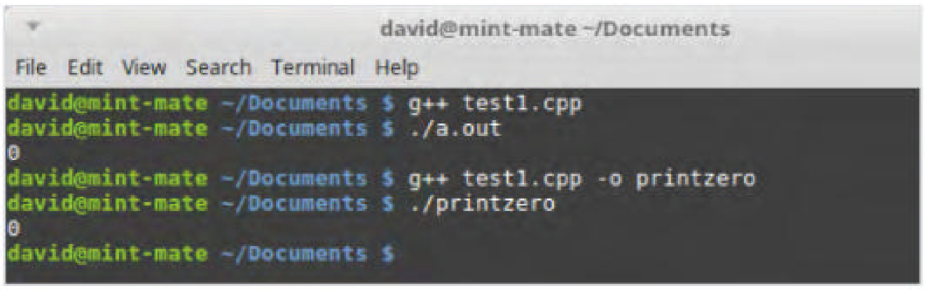
A.OUT

A common mistake when compiling in Linux is forgetting to name your C++ code post compiling. When you compile from the Terminal, you enter:

```
g++ code.cpp
```

This compiles the code in the file code.cpp and create an a.out file that can be executed with ./a.out. However, if you already have code in a.out then it's overwritten. Use:

```
g++ code.cpp -o nameofprogram
```


david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ g++ test1.cpp
david@mint-mate ~/Documents $ ./a.out
0
david@mint-mate ~/Documents $ g++ test1.cpp -o printzero
david@mint-mate ~/Documents $ ./printzero
0
david@mint-mate ~/Documents $
```