122COM: C++ introduction

David Croft
david.croft@coventry.ac.uk

January 26, 2016

Abstract

This week we are going to be looking at C++. We will be looking at the differences between it and Python and how to compile C++ code.

1 Introduction

As with Python there are multiple versions of C++. You have been learning Python 3, for C++ you will be learning C++11 (approved in 2011) as this version introduced a large number of new features and some features that make C++11 code more Pythonesque and makes things a bit easier when moving from Python to C++. We will not be using the latest version (C++14) as there is limited support for it at the moment. C++ is mostly backwards compatible, code written in older versions (C++03, C++98) will work in newer versions so there will be no problems in moving to C++14 and C++17 in the future.

The source code files for this lab are available at:

```
https://github.com/dscroft/122COM_cpp_intro.git
```

```
git@github.com:dscroft/122COM_cpp_intro.git
```

2 C++

2.1 Compiling

Before it is run C++ code has to be compiled into an executable. This is the process that converts the high level C++ code that the programmer wrote into the low level machine instructions which the computer can actually run. There are a wide range of different compilers out there but we will be using the GNU C compiler or g++ compiler which has versions available on all platforms.

On the university machines g++ is installed as part of Cygwin so once you have opened up the command line you will will need to run...

```
set path=%path%;c:/cygwin/bin
```

...or the setup.bat file from the github repo from the command line before it will work. You will need to do this each time you open the command line.

Alternatively you can sign up for a Cloud9 account at `https://c9.io/` (which is really nice) and do it through your web browser by creating a Custom or C++ workspace but be aware that you will be responsible for backing up your work.

In order to compile your C++ programs:

1. Open the command line
2. Enter
   `set path=%path%;c:/cygwin/bin`
3. Navigate to the folder containing your source code.
   - If your source code is located on your H: drive you can get there by entering
     `cd /d h:`
4. Compile your code using the following command
   `g++ --std=c++11 SOURCE_FILENAME.cpp -o PROGRAM_NAME`
5. Run your program by just entering the name of your program
   `PROGRAM_NAME`

Pre-lab work: Compile

1. Compile and run pre_compile.cpp

When you were writing Python code all your files had a .py extension, with C++ you files should have a .cpp extension. The C++ file extensions are more complicated than Python and there will be situations when you will encounter .h, .hpp or .c files but we'll cover those when we need to.

2.1.1 Integrated Development Environments (IDEs)

This week you will be compiling your code manually using g++. While it is possible to compile any program in this way it's a bit cumbersome and difficult when dealing with larger projects. Once you have gotten the hand of using g++ manually I recommend that you explore some of the many C++ IDEs, these will offer various features like project management tools, auto complete, intellisense, code highlighting and all of them will offer to compile your C++ for you.

Some IDEs I can recommend are Cloud9, CodeBlocks, Microsoft Visual Studio and Sublime Text 3 but there are many others. This module is not going to require that you use a specify IDE so just find one that you like.

2.2 Errors

Compiling is one of the first steps is debugging your program. Syntax errors will be detected by the compiler. When an error is detected the compiler will generate an error message describing what and where the error is in the code. Your code will not compile if there are any errors.

Depending on their settings compilers can also generate warnings. A compiler warning does not suggest a definite problem, but can highlight areas that should be double checked. Your code will still compile is there are warnings but may not work correctly.

Just because you code generates no compiler errors or warnings that doesn't mean that your code actually works. There is a difference between the syntax of your code being correct and the logic of your code being correct. There are a wide, wide, wide range of problems that the compiler cannot detect.

Pre-lab work: Compile

2. Compile and run the C++ version pre_error.
   - You'll need to fix the syntax and logic errors.
   - A Python version of the program has been provided to show you what the output should be.

3 Static and Dynamic typing

C++ is a high level, 3rd generation, compiled language. It is normally considered more difficult to use than Python but is more capable and significantly faster. There are many difference between C++ and Python but from a programming standpoint, the most significant difference is their type systems. C++ is statically typed and Python is dynamically typed.

In a dynamically typed language identifiers (variable names) are bound to an object (variable). In Python we can use the same identifiers to store ints, strings and floats just by binding that identifier to a new object. When the program runs the interpreter figures out the type of each identifier only when it reaches that point in the code. Below for example we see the identifier `var` being bound to an int, string and a float. A Python variable's type can be found using the `type()` function by calling the function with the variable of interest as a parameter (e.g. `type(variableName)`).

```
var = 42        # type(var) = <type 'int'>
var = 'foo'     # <type 'str'>
var = 0.123     # <type 'float'>
```

Listing 1: Example of Python variable reuse.

In a statically typed language identifiers are bound not just to an object, but also to an object type. An explicit type declaration is required for every single identifier before it can be used. Once an identifier has been bound to a type, it cannot be changed.

```cpp
int main()
{
    int var = 42;
    string var = "foo";
    float var = 0.123;

    return 0;
}
```

Listing 2: Example of C++ variable reuse that won't work.

```cpp
int main()
{
    int varA = 42;
    string varB = "foo";
    float varC = 0.123;

    return 0;
}
```

Listing 3: Will work.

3.1 Primitive Variables

In C++ the are several built in types also known as C++ primitives. These primitives form the basis for all other variable types.

The exact characteristics of the primitives changes depending on the compiler. For example, the values that an integer can hold change dramatically depending on if you using a 16, 32 or 64 bit machine. All the examples shown below are for a 32 bit machine.

- `int` - Stores whole numbers in the range -2,147,483,647 to 2,147,483,647.
  - `unsigned int` - Stores whole numbers in the range 0 to 4,294,967,295.
- `float` - Floating point (decimal numbers).
- `double` - Like `float` but uses twice as much memory, has a greater precision than float.
- `char` - A single character e.g. 'A', '9' or '@'.
  - Only ASCII characters are allowed, i.e. '' is not possible.
- `bool` - Can be either `true` or `false`.

This is not an exhaustive list, variable types such as `short` and `long double` exist but these are relatively rare and you don't need to understand them at the moment.

4 Structure

4.1 `main()`

The first difference between the structures of Python and C++ programs is the main function. In Python, programs often have a main function but in C++ it is mandatory.

Every C++ program must have a single `main()` function. The `main()` function is the entry point to a C++ program.

4.2 Modules/libraries

Python has already introduced us to the concept of modules which extend the capabilities of our code. For example we import the Tkinter module when we want to create Graphical User Interfaces (GUIs) or the random module when we want to generate random numbers.

C++ provides something similar with *##include* statements and libraries. As in Python these lines go at the top of your code and specify additional code that should be included in your program. The library that you are going to need most often is iostream, this provides the C++ version of Python's `print()`. In C++ `print()` is replaced with `cout` (C++ OUTput) and if you want a newline you need to say `endl` (END Line). Python's `input()` is replaced with `cin` (C++ INput), also part of iostream.

With C++ there is an added complication that `cout` and `endl` belong to the `std` (standard) namespace. Exactly what this means isn't important at the moment, just be aware that you either need to put `std::` in front of `cout` etc as shown in listing 4 or put the line `using namespace std;` at the top of your code as shown in listing 5.

```
#include <iostream>

int main()
{
    std::cout << "Hello World!"
        << std::endl;
    return 0;
}
```

Listing 4: Hello world! in C++.

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

int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Listing 5: Hello world! using namespace std.

4.3 Layout

Unlike Python where it controlled the structure of our code, C++ doesn't care about indentation. A C++ program can be written on a single line and it will still work. Instead of indentation, C++ uses { and } (braces) to define the structure. Instead of sections of code being indented to the same level, C++ contains sections in matching { and } brackets.

```python
def main():
    for a in range(10):
        for b in range(10)
            for c in range(10):
                print('hi')
```

```cpp
int main()
{
    for(int a=0; a<10; a++)
    {
        for(int b=0; b<10; b++)
        {
            for(int c=0; c<10; c++)
            {
                cout << "hi" << endl;
            }
        }
    }

    return 0;
}
```

HOWEVER, using whitespace and properly indenting your code is still considered good behaviour for C++ programmers. Indentation makes it much easier to understand the structure of a program, not just for the original programmer but also for anyone that has to work with their code in the future. Listing 7 and 6 show exactly the same program. Just because you can write code like listing 6 doesn't mean you should[1].

```cpp
#include <iostream>
using namespace std;
int main(){for(int i=0;i<10;i++){cout<<"Hello World!"<<endl;}return 0;}
```

Listing 6: Badly formatted C++.

4.4 Comments

In C++ comments are specified in one of two ways. There is the single line comment which works the same way as Python comments by putting `//` in front of the comment (as opposed to # in Python). And there is the multiline comment which would by putting `/*` at the start of the comment and `*/` at the end. As the name suggests, the multiline comment can span multiple lines but does not have to.

```cpp
# python comment          // c++ comment          /* multiline c++
                                                    comment */
```

[1]Unless you're taking part in the International Obfuscated C Code Contest. Look it up, those programmers are insane.

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

int main()
{
    // loop round ten times
    for(int i=0; i<10; i++)
    {
        cout << "Hello World!" << endl; // print Hello World!
    }

    return 0;
}
```

Listing 7: Well formatted C++.

5 if statements

C++ has very similar `if` statments to Python. They flow the same logical structure (i.e. `if`, `elif`, `else`) and each if statement follows the same conditional rules as Python. The differences a purely syntactical.

Firstly the conditionals, in C++ `and` is replaced with `&&` and `or` is replaced with `||`. `==` and `!=` remain the same across both, as do `<`, `>`, `<=` etc. Python's `elif` becomes `else if`.

```
a = 42

if a > 10:
    print( 'Option 1' )
elif a % 2 == 0 and a > 0:
    print( 'Option 2' )
else:
    print( 'Option 3' )
```

Listing 8: Python if statements.

```
int main()
{
    int a = 42;

    if( a > 10 )
    {
        cout << "Option 1" << endl;
    }
    else if( a % 2 == 0 && a > 0 )
    {
        cout << "Option 2" << endl;
    }
    else
    {
        cout << "Option 3" << endl;
    }

    return  0;
}
```

Listing 9: C++ if statements.

Pre-lab work:

3. Using everything you have learnt so far, convert the lab_age Python program into C++.
4. Compile your program and make sure it works correctly.

6 Iteration

6.1 `while` loops

C++ while loops work in exactly the same way as Python ones, see listings 13 and 14 on page 10.

6.2 `for` loops

C++ has two kinds of for loop. In Python for loops are always to iterate over a sequence

```
for i in range(10):
```

This creates a list containing `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]` and then steps through the list one element at a time. C++ had similar range based for loops introduced in C++11.

```python
s = 'foobar'
for c in s:
    print(c)
```

Listing 10: Python for loop.

```cpp
string s = "foobar";
for( char c : s )
{
    cout << c << endl;
}
```

Listing 11: C++11 for loop.

But C++ also has an older form of for loops which were carried over from C and which you will encounter and need to use on a regular basis. This for loop has three parts, the initialisation, the end condition and the iteration instruction.

```cpp
for( /*initialisation*/; /*end condition*/; /*iteration instruction*/ )
{
}
```

Listing 12: C++ for loop.

The initialisation configures the starting state of the for loop, it sets the starting values of the variables that are used in the for loop. The end condition controls when the for loop should exit. The for loop will continue while the end condition is `true`. The iteration instruction is an action to perform every time that we go through the loop. The iteration instruction is run at the end of each pass through the loop.
This style of for loop is much closer to a Python `while` loop than a Python `for` loop.

```python
count = 0
while count < 10:
    # do stuff
    count += 1
```

Listing 13: Python while loop.

```cpp
int count = 0;
while( count < 10 )
{
    // do stuff
    count++;
}
```

Listing 14: C++ while loop.

```cpp
for( int count=0; count<10; count++ )
{
    // do stuff
}
```

Listing 15: Traditional C++ for loop.

Lab work:

5. lab_fibonacci is a very simple Python program that calculates the first 40 digits in the Fibonacci sequence.
   - Google Fibonacci sequence if you don't know what it is.
   - Run the program so that you know what the output is.
6. Rewrite this code in C++.
   - Compile the code and make sure it runs correctly.

7 Exceptions

Like Python, C++ can (and does) raise exceptions when it encounters a problem. C++ exceptions are almost identical to those in Python except for some minor changes in syntax.

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

float fraction_to_float( int n, int d )
{
    if( d == 0 )
    {
        // raise an exception
        throw runtime_error("Can't divide by zero");
    }

    return float(n) / float(d);
}

int main()
{
    int numerator;
    int denominator;

    cout << "Enter the top of the fraction ";
    cin >> numerator;

    cout << "Enter the bottom of the fraction ";
    cin >> denominator;

    try
    {
        cout << fraction_to_float( numerator, denominator ) << endl;
    }
    // handle any runtime_error exceptions
    catch( runtime_error& e )
    {
        cerr << e.what() << endl;
    }

    return 0;
}
```

8 Arrays

Arrays are used in C++ to hold multiple variables. The closest equivalent in Python would be lists, however unlike lists, arrays cannot change their size.

Arrays are declared in a similar way to variables, we have to specify the variable name and type of the variable (in this case an array). However, we also have to let the compiler know the type and number of variables we want to store in the array. Because arrays cannot change their size it is important that you specify the correct array size.

Just like Python lists, arrays are 0 indexed (i.e. the 1st element in an array/list is in position 0).

Setting or getting a value from an array has identical syntax to getting/setting from a Python list except that you cannot use negative numbers to select from the back of the array.

Unlike lists, slicing (selecting subsections of a list e.g. `someArray[1:10]`) is not possible in C++ natively.

Listing 16 shows a simple C++ program that creates two arrays and sets the values of the elements in those arrays using `for` loops. Both kinds of C++ `for` loops are shown so that you can see how each works.

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

int main()
{
    array<int,10> arrayOfInts;
    array<char,20> arrayOfChars;

    /* setting the contents of an
       array using an old style
       for loop */
    for( int i=0; i<10; ++i )
    {
        arrayOfInts[i] = 42;
    }

    /* setting array contents using
       a new c++11 for loop */
    for( char& c : arrayOfChars )
    {
        c = 'A';
    }

    return 0;
}
```

Listing 16: C++ array example showing the two ways to iterate over an array.

8.1 Vectors

As already mentioned, C++ arrays cannot be resized. This can, sometimes, limit their usefulness. The closest C++ equivalent to the Python `list` is the `vector`. A vector is effectively just a resizeable array.

Vectors support most of the features of Python lists however the commands used are different. For example, the `.append()` function is replaced with `.push_back()`. `len()` is replaced with `.size()` etc. For a complete list of the available functions look at the documentation online (i.e. `http://www.cplusplus.com/reference/vector/vector/`).

Listing 17 shows how to define a vector in comparison to an array, how to add additional elements and how to access elements from a vector and array.

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

int main()
{
    array<int,5> myArray = {1,2,3,4,5};
    vector<int> myVector = {1,2,3,4};

    myVector.push_back(5);

    cout << myArray[0] << endl;
    cout << myVector[0] << endl;
}
```

Listing 17: C++ vector example showing how to access and add additional elements.

Lab work:

7. Using everything you have learnt for far, convert the lab_array Python program into C++.
   - Note that I have provided you with a C++ version of the Python `input()` function. So you don't need to worry about how to read in C++ user input just at the moment.
8. Compile your program and make sure it works correctly.

9 Functions

As with variables, functions in C++ must have an explicitly stated type. The function type describes the type of the variable that the function returns. Even when the function isn't returning anything we still have to provide a return type. In these situations the return type should be set as `void`.

```python
def sum( a, b ):
    return a + b

def return_nothing():
    print( 'Nothing' )

c = sum( 42, 69 )
print( sum( 100, 1 ) )

return_nothing()
```

Listing 18: Examples of a Python functions.

```cpp
int sum( int a, int b )
{
    return a + b;
}

void return_nothing()
{
    cout << "Nothing" << endl;
}

int main()
{
    int c = sum( 42, 69 );
    cout << sum( 100, 1 ) << endl;

    return_nothing();

    return 0;
}
```

Listing 19: Examples of a C++ functions.

Lab work:

9. Complete lab_function. You need to write a function called `is_leap()` that identifies if a given year is, or is not, a leap year.
   - `is_leap()` should return `true` or `false`.
   - `is_leap()` should take one argument, the year to be tested.
   - Think carefully about the return and argument variable types.
   - Those of you that did the sync lab will be familiar with the correct leap year logic, otherwise you may want to research this further.

Extended work:

10. Investigate C++ classes.
    - Pay particular attention to the C++ protected state as this isn't really supported in Python.
11. If you want additional reading I recommend "A Transition Guide from Python 2.x to C++" by Michael H. Goldwasser and David Letscher (available online).
    - The examples are in Python2 as opposed to Python3 but the differences are minor.