122COM: Introduction to C++

David Croft

Coventry University

david.croft@coventry.ac.uk

2017

Overview

Expectations

- All courses expected to be aware of different languages.
  - Advantages and disadvantages.
- BIT & MC are allowed to do most of 122COM in Python3.
  - Can choose C++14 if they wish.
- Everyone else is expected to move to C++14 for the remainder of 122COM.

Expectations

- All students are expected to learn some C++.
- In future weeks will mostly be looking at generic programming concepts.
  - Will be taught in Python and C++.
- BIT & MC students.
  - Python or C++ unless task says otherwise.
  - Will not be tested on C++ code.
    - May be tested on language differences.
    - High/low languages.
    - Compiling.
    - Static/dynamic typing.
- Everyone else.
  - C++ unless task says otherwise.

C++14 in Codio

A

New projects on Codio will not have a C++ or C++14 compiler installed.

Three options if you want to create your own Codio projects:

1. Use the 122COM stack.
   - Has everything pre-installed.
2. Install it manually.

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt update
sudo apt install build-essential gcc-7 g++-7
sudo update-alternatives -install /usr/bin/gcc gcc /usr/bin/gcc-7 60
sudo update-alternatives -install /usr/bin/g++ g++ /usr/bin/g++-7 60
```

3. Use the auto install script.

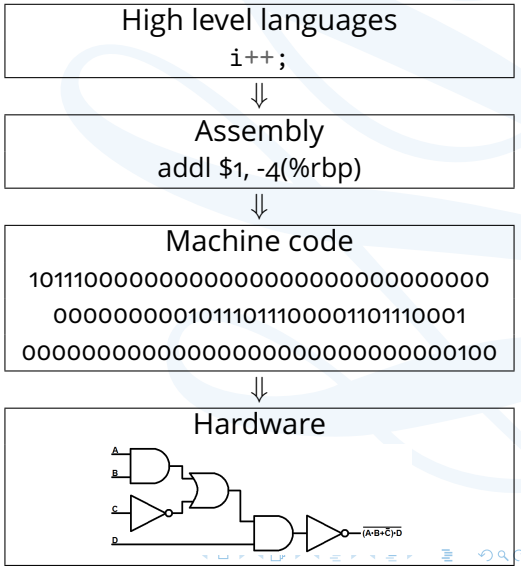
Running random scripts off the internet is generally a bad idea!

```
wget https://git.io/vF0E1 -O - | sudo bash
```

Language levels

Highs and lows

- Programming languages split into levels.
- Low level languages are machine code, assembly language.
- High level languages are Python, C++, Java etc.
  - Not a binary classification, e.g. C++ is lower level than Python.



Low level

Machine code

- 1st generation.
- Really hard to understand.
- Really hard to write.
- The actual instructions to the hardware.

```
11100101110001110100010111111100
00101010000000000000000000000000
10000011010001011111110000000001
10111000000000000000000000000000
00000000010111011100001101110001
00000000000000000000000000000100
```

Low level

Machine code

- 1st generation.
- Really hard to understand.
- Really hard to write.
- The actual instructions to the hardware.

```
11100101110001110100010111111100
00101010000000000000000000000000
10000011010001011111110000000001
10111000000000000000000000000000
00000000010111011100001101110001
00000000000000000000000000000100
```

Assembly

- 2nd generation.
- Hard for humans to understand.
- Hard for humans to write.
- 1-to-1 correspondence with what is run.

```
movl $42, -4(%rbp)
addl $1, -4(%rbp)
```

High level

Python, C, C++, Java, PHP, Perl etc.

High level

Python, C, C++, Java, PHP, Perl etc.

- 3rd generation.

High level

Python, C, C++, Java, PHP, Perl etc.

- 3rd generation.
- Favour programmer, not machine.

High level

Python, C, C++, Java, PHP, Perl etc.

- 3rd generation.
- Favour programmer, not machine.
- Easy for humans to understand...compared to the alternatives.
- Easy for humans to write...compared to the alternatives.

High level

Python, C, C++, Java, PHP, Perl etc.

- 3rd generation.
- Favour programmer, not machine.
- Easy for humans to understand...compared to the alternatives.
- Easy for humans to write...compared to the alternatives.
- Portable.

High level

Python, C, C++, Java, PHP, Perl etc.

- 3rd generation.
- Favour programmer, not machine.
- Easy for humans to understand...compared to the alternatives.
- Easy for humans to write...compared to the alternatives.
- Portable.
  - Different machine/processor/OS == different compiler.
  - Same C/Python/C++/Java code.

High level

Python, C, C++, Java, PHP, Perl etc.

- 3rd generation.
- Favour programmer, not machine.
- Easy for humans to understand...compared to the alternatives.
- Easy for humans to write...compared to the alternatives.
- Portable.
  - Different machine/processor/OS == different compiler.
  - Same C/Python/C++/Java code.

```
int i=42;
i++;
```

C++

History of C++

So far you have used Python.
Now going to learn C++.

- Created somewhere in 1979-1983.
- Based on C (created 1972).
- Going to be learning C++14 (approved 2014).
  - Almost identical to C++11 with bugfixes.
- C++17 has been approved (2017).
  - Limited support so far.
- 99.9% backwards compatible.
  - All the way to C.

History of C++

So far you have used Python.
Now going to learn C++.

- Created somewhere in 1979-1983.
- Based on C (created 1972).
- Going to be learning C++14 (approved 2014).
  - Almost identical to C++11 with bugfixes.
- C++17 has been approved (2017).
  - Limited support so far.
- 99.9% backwards compatible.
  - All the way to C.

- Supports the same paradigms as Python.
  - Objected oriented, functional, declarative etc.

About C++

Most significant difference...

- C++ is statically typed.
  - Python is dynamically typed.

- In Python variables keep track of values AND type.

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

- In C++ variables have one type forever.
  - Have to specify type when creating.

```
int    var1 = 42;
string var2 = "foo";
float  var3 = 0.123;
```

Data types

In C++ have to specify a variable's type.

- So what types are available?
- Thousands (at least).
  - You can create your own.
- Few standard ones.
- Most basic data types are called primitives.

Primitive types

- Knowing what the different variables are.
- Knowing all the primitives and the variations.
- Knowing ranges/sizes.

- Most of these should be familiar from Python.

Type	Size (bytes)	Values
bool	1	true/false
char	1	'a', 'Z', '6', '+'
int	4	-2147483647 $\rightarrow$ 2147483647
unsigned int	4	0 $\rightarrow$ 4294967295
float	4	1.234, -0.0001
double	8	1.23456789, -0.000000001
void		

Sizes are correct for a 32bit machine.

Syntax

Syntax

Moving from Python to C++.

- Not as bad/scary as it seems.
- Same basic structure.
- Slightly different syntax.

Hello World!

Basic Python.

```
print('Hello World!')
```

Hello World!

Basic Python.

```
print('Hello World!')
```

More complete Python

```
import sys

def main():
    print('Hello World!')

if __name__ == '__main__':
    sys.exit(main())
```

lec_hello.py

Hello World!

Basic Python.

```
print('Hello World!')
```

More complete Python

```
import sys

def main():
    print('Hello World!')

if __name__ == '__main__':
    sys.exit(main())
```

lec_hello.py

C++.

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

int main()
{
    cout « "Hello World!" « endl;

    return 0;
}
```

lec_hello.cpp

Hello World!

Basic Python.

```
print('Hello World!')
```

More complete Python

```
import sys

def main():
    print('Hello World!')

if __name__ == '__main__':
    sys.exit(main())
```

lec_hello.py

C++.

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

int main()
{
    cout << "Hello World!" << endl;

    return 0;
}
```

lec_hello.cpp

- All programs in C++ **MUST** have exactly one `main()` function.
- C++ uses { and } instead of indentation.
  - You should still have indentation in C++ but is aesthetic only.
- Semi-colons at the end of lines.

if statements

Same rules as Python.

- Slightly different syntax.
- and is now &&
- or is now ||
- equivalence is still ==
- The conditional statement/s must be in brackets.
- Instead of indenting we use braces {}.

if statements

Same rules as Python.

- Slightly different syntax.
- and is now &&
- or is now ||
- equivalence is still ==
- The conditional statement/s must be in brackets.
- Instead of indenting we use braces {}.

```
a = 1
b = 2

if a == b and b > 0:
    print('Hello World' )
```

if statements

Same rules as Python.

- Slightly different syntax.
- and is now &&
- or is now ||
- equivalence is still ==
- The conditional statement/s must be in brackets.
- Instead of indenting we use braces {}.

```
a = 1
b = 2

if a == b and b > 0:
    print('Hello World' )
```

```
int a = 1;
int b = 2;

if( a == b && b > 0 )
{
    cout « "Hello World!" « endl;
}
```

Arrays

Similar to Python lists.

- Can't be resized.

```
sequence = [1, 2, 42, 69, 8]
sum = 0

for i in range(len(sequence)):
    sum += sequence[i]
```

```
array<int,5> sequence = {1, 2, 42, 69, 8};
int sum = 0;

for( int i=0; i<sequence.size(); i++ )
{
    sum += sequence[i];
}
```

Vectors

Problem, C++ arrays have a set size.

- Saw we had to provide a size when declaring arrays.

Vectors

Problem, C++ arrays have a set size.

- Saw we had to provide a size when declaring arrays.

C++ does have 'arrays' that can be resized.

- Called vectors.
- Use arrays inside.

Vectors

Problem, C++ arrays have a set size.

- Saw we had to provide a size when declaring arrays.

C++ does have 'arrays' that can be resized.

- Called vectors.
- Use arrays inside.

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

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

    myVector.emplace_back(5);

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

lec_vector.cpp

Vectors II

C++ vectors are the closest thing to Python lists.

- If you are moving to C++ from Python easier to use vectors?

- append() → push_back() or emplace_back()
- pop() → pop_back()
- slicing → resize()

while loops

Same rules as Python.

- Slightly different syntax.
- Brackets `()`.
- Braces `{}`.
- Semicolons `;`.

```
counter = 0
while counter < 10:
    print('Hello World!')
    counter += 1
```

```
int counter = 0;
while( counter < 10 )
{
    cout « "Hello World!" « endl;
    counter += 1;
}
```

for loops

C++ has two kinds of for loops.

- One type similar to Python for loops.
  - Actually a range-based loop.
  - Will be covered later.
- One type similar to a while loop.

for loops

The original C++ for loop.

```
for( int counter=0; counter<10; counter+=1 )
{
    cout « "Hello World!" « endl;
}
```

for loops


```python
for counter in range(10):
    print('Hello World!')
```

The original C++ for loop.

- Seems very different to the python loop.

```cpp
for( int counter=0; counter<10; counter+=1 )
{
    cout << "Hello World!" << endl;
}
```

for loops

The original C++ for loop.

- Seems very different to the python loop.
- Lots of commonalities.

```
for counter in range(10):
    print('Hello World!')
```

```
for counter in range(0,10,1):
    print('Hello World!')
```

```
for( int counter=0; counter<10; counter+=1 )
{
    cout « "Hello World!" « endl;
}
```

for loops

The original C++ for loop.

- Seems very different to the python loop.
- Lots of commonalities.
- Also to while loops.

```
for counter in range(10):
    print('Hello World!')
```

```
for counter in range(0,10,1):
    print('Hello World!')
```

```
for( int counter=0; counter<10; counter+=1 )
{
    cout « "Hello World!" « endl;
}
```

```
int counter = 0;
while( counter < 10 )
{
    cout « "Hello World!" « endl;
    counter += 1;
}
```

Ranged for loops

The new C++11 ranged for loop, for iterating over a sequence.

- Less powerful that the old style.
- Easier.
- while > for > ranged for

```
sequence = [1,2,3,4,5]
for i in sequence:
    print( i )
```

```
int main()
{
    array<int,5> sequence =
        { 1, 2, 3, 4, 5 };
    for( int i : sequence )
    {
        cout « i « endl;
    }

    return 0;
}
```

Compiling

C++ code has to be compiled before it is run.

- So does Python it just happens automatically.
- Compiler converts C++ code into machine code.
- Many IDEs handle compiling for you.
  - Visual Studio, Eclipse etc.

- Make you do it yourself in this module so you understand it.
  - Understand what IDE is doing.
  - Understand the configuration options in the IDE.
  - Understand the error messages you get.
  - Once understood then use IDEs.

Compiling


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

gcc & g++

In Codio we are using the GNU C Compiler (created 1987).

- Available for Linux, Mac and Windows.

gcc & g++

In Codio we are using the GNU C Compiler (created 1987).

- Available for Linux, Mac and Windows.

How to compile using g++. **Demo in Codio**

- g++ –std=c++14 hello.cpp -o hello
  - g++ - the compiler program.
  - –std=c++14 - we want to use the C++14 standard of C++.
  - hello.cpp - the file we want to compile.
  - -o hello - the name of the executable to create.

How to run the program. **Demo in Codio**

- ./hello
  - ./ - it's in the same directory we're in.
  - hello - the name of the executable to run.

Debugging

What if your code is wrong?

- Same as Python.
  - Syntax errors.
  - Runtime errors.
  - Logic errors.

Debugging

What if your code is wrong?

- Same as Python.
  - Syntax errors.
  - Runtime errors.
  - Logic errors.
  - Spot the errors.

```cpp
int main()
{
    cout « "Hi" « endl;

    for( int i=0; i>10; j+=1 )
    {
        cut « "Hello World!" « endl
    }

    return 0;
}
```

lec_error.cpp

Break

Functions

Have to specify the type for the return value and the parameters.

- Otherwise the same as Python.
- `void` if it doesn't return anything.

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

void nothing_function()
{
  cout « "Return nothing" « endl;
}
```

lec_function.cpp

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

def nothing_function():
    print( "Return nothing" )
```

lec_function.py

New and improved!

Important announcement - Two types of arrays in C++14.

- One is carried forward from C.
  - Still seen regularly.
- C++03 introduced an alternative.
  - STL arrays.

New and improved!

Important announcement - Two types of arrays in C++14.

- One is carried forward from C.
  - Still seen regularly.
- C++03 introduced an alternative.
  - STL arrays.

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

int main()
{
    int oldArray[5] = {1,2,3,4,5};
    array<int,5> newArray = {1,2,3,4,5};          // use me!

    cout << oldArray[0] << " " << newArray[0] << endl;
}
```

lec_arrays.cpp

There's two of them?

Two types of arrays.

- Old style arrays are still very common.
  - Legacy code.
  - Old tutorials.
  - Want you to use the new ones.
- What was wrong with the old ones?
- New arrays are safer.
  - Avoid overflows.
- Easier to use.
  - Sorting, searching, reversing, iterating etc.
- Are backwards compatible with old code.

Recap

Why do I care?

- Everyone
  - C++ is widely used, 4th on IEEE top language list 2016.
  - Knowledge of multiple languages can help you in understanding the underlying logic concepts.
- Computing - C++ provides more efficient code than Python.
- Computer Science - C++ provides direct memory access, allowing greater understanding of computer memory and important abilities such as concurrent programming.
- Ethical Hackers - C++ provides direct memory access, important in understanding many hacks.
- Games Tech - C++ is a requirement for many games companies, it is an absolute requirement for your 3rd year modules.

Recap

- C++ is a high level language.
- Compiled.
- Statically typed.
- Arrays cannot be resized.
  - Use new STL arrays.
- Vectors can be resized.
- Investigate C++ classes.
- Investigate STL Algorithm Library.

The End