122COM: Programming languages

David Croft

Coventry University

david.croft@coventry.ac.uk

2015

Overview

Highs and lows

- Programming languages split into levels.
- Low level languages are machine code, assembly language.
- High level languages are Python, C++, Java etc.

High level languages

```
i+=1
```

⇓

Assembly

```
GOTO 0x42
INCF 0x68 0x01
SLEEP
```

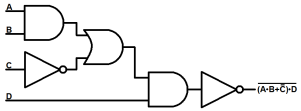
⇓

Machine code

01010011 01100101 01110010 01101001
01101111 01110101 01110011 01101100 01111001
00111111 01111001 00111111

⇓

Hardware



⇑

⇓

High level	Python, Ruby Java C++ C Forth, Basic	3rd generation
Low level	Assembly	2nd generation
	Machine code	1st generation
	Hardware	

Low level

Machine code

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

Assembly

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

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 == different compiler.
  - Same C/Python/C++/Java code.

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++11 (approved 2011).
- C++14 has been approved (2014).
  - No support yet.
- 99.9% backwards compatible.
  - All the way to C.

Expectations

- All students are expected to learn some C++.
- In future weeks we will be looking at generic programming concepts.
  - Sorting.
  - Searching.
  - Data structures.
- Those weeks will be taught in Python and C++.
  - Everyone else will have some mandatory C++ tasks.
  - BIT students can choose Python or C++ most tasks.
  - Will be specified at the time.
- BIT will not be examined on C++ code.
  - May be examined on language differences.
  - High/low languages.
  - Compiling.
  - Static/dynamic typing.
  - Stack/heap memory.

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 primitive types.

Primitive types

Type	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

Moving from Python to C++.

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

Hello World!

Python.

```python
import sys

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

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

C++.

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

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

    return 0;
}
```

if statements

Same rules as Python.

- Slightly different syntax.
- and is now &&.
- or is now ||.
- == is still ==.

```
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]
```

```
int sequence[5] = {1, 2, 42, 69, 8};
int sum = 0;

for( int i=0; i<5; i+=1 )
{
    sum += sequence[i];
}
```

Arrays II

Three ways to create a C array.

1. Just supply size

```
int   arrayOfInt[3];
char  arrayOfChars[5];
float arrayOfFloats[2];
```

2. Supply size and initialisation list

```
int   arrayOfInt[3]    = { 42, 69, 12 };
char  arrayOfChars[5]  = { 'A', 'z', '9' };
float arrayOfFloats[2] = { 1.23, 0.001, 8.1 };
```

3. Just initialisation list (will figure out the size)

```
int   arrayOfInt[]     = { 42, 69, 12 };
char  arrayOfChars[]   = { 'A', 'z', '9' };
float arrayOfFloats[]  = { 1.23, 0.001, 8.1 };
```

New and improved!

So far looked at the old style arrays.

- Carried forward from C.
- Still used today.
- C++03 introduced an alternative.
  - STL arrays.

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

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

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

    return 0;
}
```

There's two of them?

Two types of arrays.

- Old style arrays are still very common.
  - Legacy.
  - Want you to start off using 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.

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.
- Uses arrays inside.

```
#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.push_back(5);

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

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() and 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.

- 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.

Compiling


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

gcc & g++

GNU C Compiler (created 1987).

- Linux, Mac and Windows.

How to compile using g++.

- Demo
- g++ –std=c++11 hello.cpp -o hello
  - g++ - the compiler program.
  - –std=c++11 - we want to use the C++11 standard of C++.
  - hello.cpp - the file we want to compile.
  - -o hello - the name of the executable to create.

Debugging

What if your code is wrong?

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

```
int main()
{
    cout << "Hi" << endl;

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

    return 0;
}
```

The End