

HOW DO C++ AND JAVA COMPARE WITH EACH OTHER

- Syntax is mostly the same.
- Different library functions.

`System.out.println("Hello");` *versus* `cout << "Hello";`

- No automatic memory management in C++.
 - You have to delete what you create dynamically (i.e., using `new`).
- C++ does have classes (and thus methods) but it also have functions, which are actually the fundamental building blocks.
 - In particular, `main` is a **function**, not a method.
- C++ uses **header files** containing function, variable, and class declarations.
 - Pain at the beginning, but you will come to appreciate them.
- In C++, the type `int` is stored in a word of memory whose size is machine dependent.
- One word: pointers (in C++).

A FIRST C++ PROGRAM

Java (Foo.java)

```
import java.io.*;

//We must declare a class to contain main
class Foo {
    public static void main(String[] args){
        for (int i = 0; i < 10; i++) {
            System.out.println(i + "*" + i +
                               " = " + i*i);
        }
    }
} // end Foo
```

C++ (foo.cc)

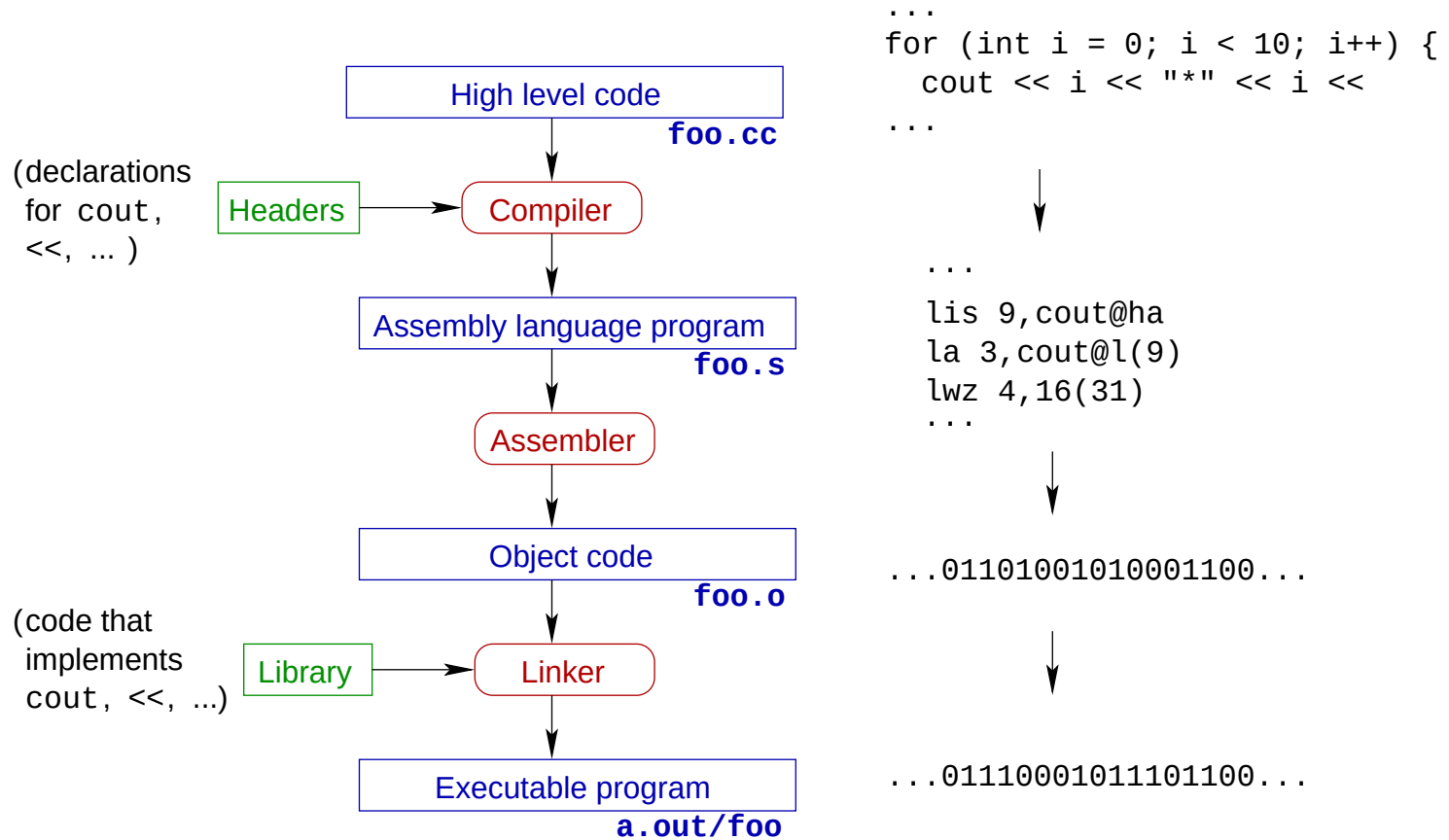
```
#include <iostream>
using namespace std;

//main pertains to no class
//no need for any definition
int main(int argc, char** argv) {
    for (int i = 0; i < 10; i++) {
        cout << i << "*" << i <<
             " = " << i*i << "\n";
    }
    return 0; // optional!
}
// nothing to end here
```

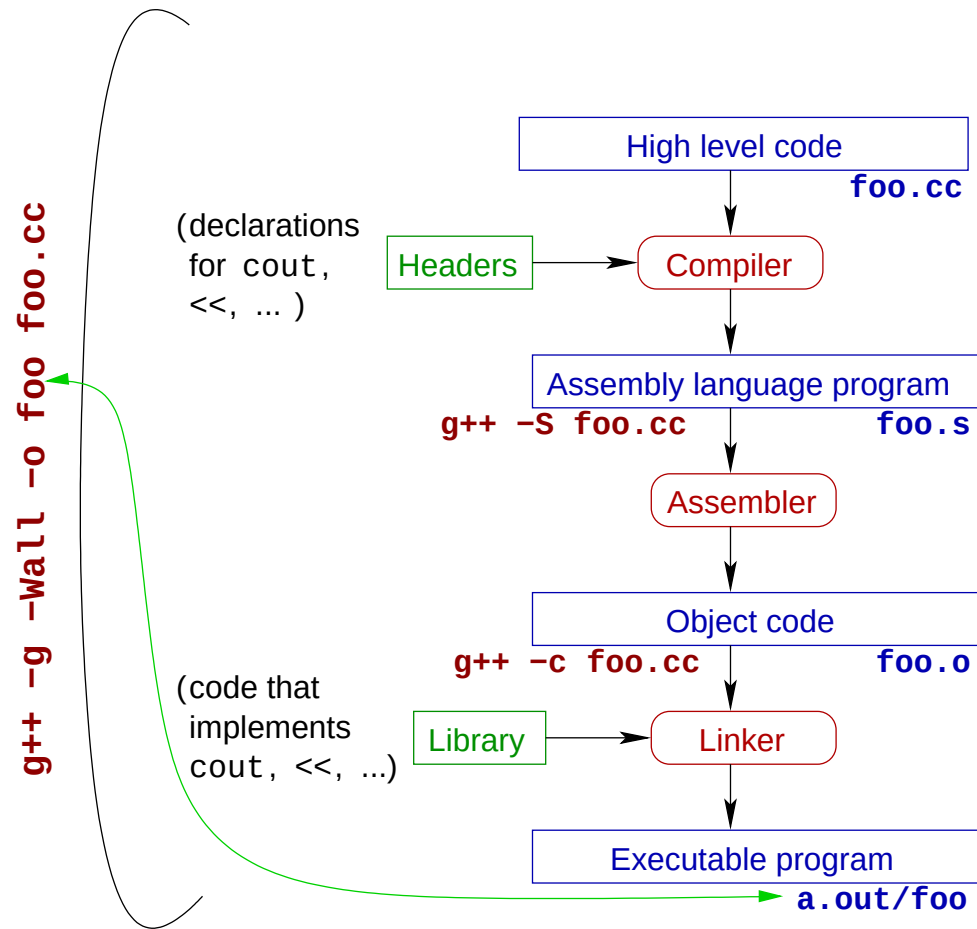
Notice:

- Inclusion of header files.
- The for loop.
- Output (cout and the << operator).
- String literals, and the escape sequence \n.
- Integer variables, literals and integer arithmetic.
- Assignments, incrementing integers.
- Comparisons.

Now what?



Now what?



```
...  
for (int i = 0; i < 10; i++) {  
    cout << i << "*" << i <<  
...  
...
```



```
...  
lis 9,cout@ha  
la 3,cout@l(9)  
lwz 4,16(31)  
...
```



...01101001010001100...



...01110001011101100...

ONE FUNCTION PROGRAMS

- Let us consider for a starter programs that look like this:

```
int main(int argc, char** argv) {  
    code that does something useful  
}
```

- The **code** looks similar to what you would see inside a Java method.

- contains **data declaration** and **actual code** (or instructions)

- **data** looks just like in Java, with some minor differences

```
int total;                float balance[size];  
const int size = 100;
```

- **code** looks again similar to Java code

- * Notable exception: there is no difference between booleans and integers (0 = false).

- * Thus the following sucks but is nonetheless correct:

```
int i = 5;  
while (i) { i--; }
```

“ONE CLASS” PROGRAMS

- If your **main** functions grows too long or too complicated, you should split the program into multiple **functions**.

```
public static int square(int x) {  
    return x * x;  
}
```

- You will get something like this:

```
#include <iostream>  
using namespace std;  
  
int square(int x) {  
    return x * x;  
}  
  
int main(int argc, char** argv) {  
    cout << "Five squared is " << square(5) << "\n";  
}
```

C++ BASIC TYPES

- In a nutshell:

Type	Declaration	Literals	What it really means
boolean	<code>bool a;</code>	<code>true false</code>	Could also use integers
characters	<code>char a;</code>	<code>'b' '\n' '\\'</code>	Really a one-byte integer
integers	<code>int a;</code>	<code>15</code>	
floating	<code>float a;</code>	<code>3.1415</code>	3.4×10^{-38} to 3.4×10^{38} , 7 digits
point	<code>double a;</code>		1.7×10^{-308} to 1.7×10^{308} , 15 digits

- A whole bunch of integers:

Keyword	Min	Max	Bytes
<code>char</code>	-128	127	1
<code>unsigned char</code>	0	255	1
<code>short</code>	-32,768	32,767	2
<code>unsigned short</code>	0	65,535	2
<code>int</code>	-2,147,483,648	2,147,483,647	4
<code>unsigned int</code>	0	4,294,967,295	4

COMPOUND STATEMENTS

- (Compound) statement: { first-statement second-statement ... }

- Conditional:

if (condition)
 statement

if (condition)
 statement
else
 statement

- Loops:

while (condition)
 statement

for (init-statement; condition; update-statement)
 statement



```
{  
  init-statement  
  while (condition) {  
    statement  
    update-statement;  
  }  
}
```

break
continue

remember
them?

INPUT

- We have an object `cout` that takes care of output. Similarly, the object that takes care of input is `cin`.
- The operator that sends data to the output is `<<`. Similarly, the operator that takes data from the input is `>>`.

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

int main(int argc, char** argv) {
    double radius = -1, area;
    const double PI = 3.14159;

    while (radius != 0) {
        cout << "Enter radius: ";
        cin >> radius;
        if (radius == 0) { cout << "Bye\n"; }
        else {
            area = PI * radius * radius;
            cout << "Area is " << area << '\n';
        }
    }
}
```

INPUT (CONT'D)

- Text input may be a bit tricky:

```
int main(int argc, char** argv) {  
    char aString[100];  
  
    cout << "Type something: ";  
    cin >> aString;  
    cout << "You typed \"" << aString << "\"\n";  
}
```

INPUT (CONT'D)

- Text input may be a bit tricky:

```
int main(int argc, char** argv) {
    char aString[100];

    cout << "Type something: ";
    cin >> aString;
    cout << "You typed \"" << aString << "\"\n";
}
```

- So we do instead:

```
int main(int argc, char** argv) {
    char aString[100];

    cout << "Type something: ";
    cin.getline(aString, 100);
    cout << "You typed \"" << aString << "\"\n";
}
```

INPUT (CONT'D)

- The following works as expected...

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

int main () {
    int i, j;
    char input[20];

    cin >> i;
    cin >> j;

    cin >> input;

    cout << "=====\n";
    cout << "i = " << i <<"; j = " << j
        << "\ninput = " << input << "\n";
}
```

INPUT (CONT'D)

- ...but the following does not (why?)

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

```
int main () {
    int i, j;
    char input[20];
```

```
    cin >> i;
    cin >> j;
```

```
    cin.getline( input, 20 );
```

```
    cout << "=====\n";
    cout << "i = " << i <<"; j = " << j
        << "\ninput = " << input << "\n";
```

```
}
```

INPUT (CONT'D)

- ...but the following **does** (why?)

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

int main () {
    int i, j;
    char input[20];

    cin >> i;
    cin >> j;

    cin.ignore();

    cin.getline( input, 20 );

    cout << "=====\n";
    cout << "i = " << i <<"; j = " << j
        << "\ninput = " << input << "\n";
}
```

MORE OUTPUT

- Output has its own trickyness:

```
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
    cout << "This line should show up before hanging";
    while (1);
}
```

MORE OUTPUT

- Output has its own trickyness:

```
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
    cout << "This line should show up before hanging";
    while (1);
}
```

- So we do instead:

```
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
    cout << "This line should show up before hanging";
    cout.flush();
    while (1);
}
```