# CSc 256 Survival Guide: C++ for Java programmers

Why C/C++?

Interpretation vs. direct execution

Variables, input, output

Arrays

Functions/methods

Examples:

http://unixlab.sfsu.edu/~whsu/csc256/C4Java/



# C/C++

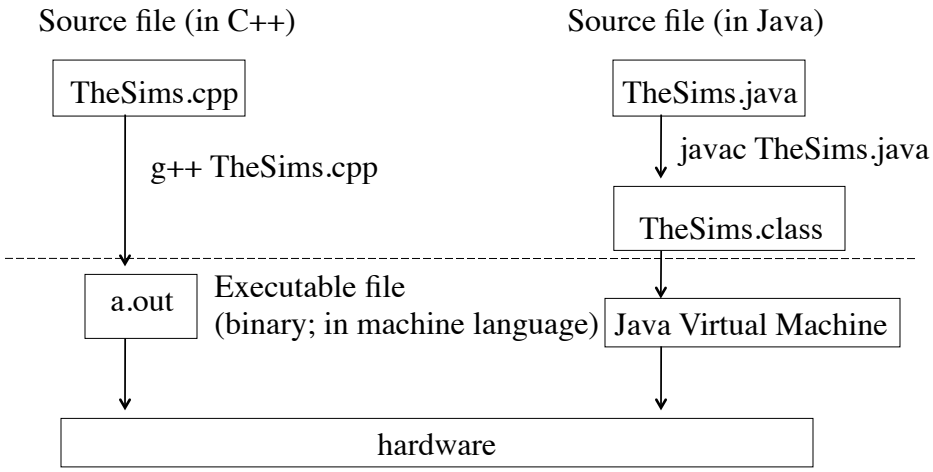
C (early 70s):

- Portable replacement for assembly language programs
- Simple to compile, easy to map to machine language
- Low-level access to memory, weak typing
- Pointers and pointer arithmetic
- Minimal run-time support

C++ (mid 80s):

- C with objected-oriented features

# Native execution vs. Interpretation






C/C++ is compiled to *executable files* in machine language. The executables run directly on hardware.

Java is compiled to *class files* containing *Java bytecodes*. A class file has to be interpreted by a Java Virtual Machine, another application.

Comparisons:

- Each machine architecture has a different machine language.

  All architectures have the same Java bytecodes.

  Hence, Java programs "compile once, run everywhere".

  C/C++ programs must be recompiled for each architecture.
- But interpretation is *slower* than direct execution.

# Compile/run C/C++ and Java

All examples are in
http://unixlab.sfsu.edu/~whsu/csc256/C4Java/

[C++ source files have extension .cc or .cpp]

Compile and run Bye.java:

```
unixlab% javac Bye.java    [this creates Bye.class]
unixlab% java Bye
Goodbye cruel world...
unixlab%
```



Compile and run Bye.cc with g++ compiler:

```
unixlab% g++ Bye.cc -o Bye [this creates Bye]
unixlab% ./Bye
Goodbye cruel world...
unixlab%
```

-o Bye means "generate output executable named Bye".
If omitted, will generate executable file named a.out.

```
unixlab% g++ Bye.cc [this creates a.out]
unixlab% ./a.out
Goodbye cruel world...
unixlab%
```

# Basic syntax differences

```
public class Bye {
    public static void main(String[] args){
        System.out.println("Goodbye cruel world...");
    }
}
```

```
using namespace::std;
#include <iostream>
int main() {
  cout << "Goodbye cruel world..." << endl;
}
```



Main differences:

- Java and C/C++ have different libraries
  C++:
  ```
  using namespace::std;
  #include <iostream>
  ```
- Different output statements
- Java main method code is wrapped in public class.
  C/C++ main is outermost level (no public class).
- Java main returns void
  C++ main returns int
- Java main and C++ main have different arguments
- Java main is *public* and *static*
  No such modifiers for C++ main

# Variables, input/output

Compare IO.java and IO.cc; top half:

```
// IO.java: variables, input, output
import java.util.Scanner;
public class IO {
    public static void main(String[] args) {
1     int x = 13; float y = -4.3f;
2     System.out.println("x = " + x + " y = " + y);
```

```
// IO.cc: variables, input, output
using namespace::std;
#include <iostream>
int main() {
1  int x = 13; float y = -4.3f;
2  cout << "x = " << x << " y = " << y << endl;
```



Compare IO.java and IO.cc; bottom half:

```
        Scanner input = new Scanner(System.in);
1       String prompt = "Enter x: ";
2       System.out.print(prompt);
3       x = input.nextInt();
4       System.out.println("x = " + x);
    }
}
```

```
1  char prompt[] = "Enter x: ";
2  cout << prompt;
3  cin >> x;
4  cout << "x = " << x << endl;
}
```

# Arrays

Char array: declaration, initialization and output
Compare Arrays.java and Arrays.cc, top half:

```
1 char [] truth = {'C', 'r', 'u', 'e', 'l', '!'};
2 System.out.println(truth);
3 for (int i=0; i<truth.length; i++) {
4     System.out.println(i + " " + truth[i]);
  }
```

```
1 char truth[] = {'C', 'r', 'u', 'e', 'l', '!'};
2 cout << truth << endl;
3 for (int i=0; i<6; i++) {
4   cout << i << " " << truth[i] << endl;
  }
```



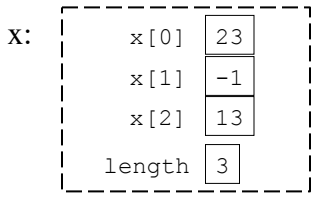
Int arrays: compare Arrays.java and Arrays.cc; bottom half:

```
1 int [] x = {23, -1, 13};

2 for (int i=0; i<x.length; i++) {
3     System.out.println(i + " " + x[i]);
4 }
```

```
1 int [] x = {23, -1, 13};

2 for (int i=0; i<3; i++) {
3    cout << i << " " << x[i] << endl;
4 }
```

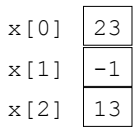
Arrays in Java are *objects*. The number of elements in an array is stored with the array data. Example:

```
int [] x = {23, -1, 13};
```




Arrays in C/C++ are bare data in memory. (More on this later). Example: `int x[] = {23, -1, 13};`

*This looks just like 3 integers in consecutive memory locations!*






In Java, this code will cause an exception / error:

```
int [] x = {23, -1, 13};
x[3] = 2;     // oops, write past end of array
```

In C/C++, this code will run most of the time!

```
int x[] = {23, -1, 13};
x[3] = 2;
```

Again, this is because in C/C++, x[] just looks like 3 integers grouped together.

x[3] is simply the integer after x[2].

What actually happens for x[3] = 2?

Some differences when working with arrays in C/C++ (vs. Java):

- Length of array has to be stored in separate variable
- When working with an array, must keep track of its length
- Access to element of array that was not allocated may not cause error (but looks like access to unrelated variables)

These are common sources of errors when programming in C/C++.

However, this means C/C++ executables do not have to check for array bounds on each array access; may run faster.

(These differences are more pronounced for multi-dimensional arrays.)



# Methods / functions

C/C++ functions/procedures are very similar to Java methods.

(Functions are not associated with classes/objects.)

Compare: Pow.java and 4.1.cpp. Main is almost identical.

```
public class Pow {
    public static void main(String [] args) {
        // main almost identical; not shown
    }
    public static int pow(int arg0, int arg1) {
        int product = 1;
        for (int i=0; i<arg1; i++) {
            product *= arg0;
        }
        return product;
    }
}
```

```
int pow(int, int);
int main() {
 // main almost identical; not shown
}
int pow(int arg0, int arg1) {
  int product = 1;
  for (int i=0; i<arg1; i++) {
    product *= arg0;
  }
  return product;
}
```

- Once we pull Java code for main() and pow() out of public class, almost identical to C++!
- Difference: in C/C++, need to specify *prototype* for pow() first: `int pow(int, int);`