C++ is not Java

Plan for today

- Get you started with C++
  - Compilation environment
  - The very, very basics
  - Preparation for Lab 1

History of C++

- 1969 – UNIX
  - Created to play space wars
  - B (Basic Combined Programming Language)
- 1970
  - UNIX port to PDP-11
- 1972
  - C (rewrite of B)
- Early 1980s
  - C++ invented as a "superset of C"

History of C++

- Why bring this up
  - C is a "low level" system language
    - Program as manipulator of memory
    - All memory management done by hand
  - C++ is not only based on C but is a proper superset
    - If you can do it in C you can do it in C++
    - Added Object Oriented paradigm.
  - Take home message: C++ is not Java!!!!

C++ is not Java

10. Not classes -- just files

Not classes -- just files

- One class, two files
  - Header file (.h)
    - Contains class declaration (interface++)
  - Source file (.c, .cpp, .C, .cxx)
    - Contains class definition
      - implementation of methods

Not classes -- just files

- Cpp – The C Preprocessor
  - Reads all C code before compilation
  - Directives
    - Including text files
      - #include
    - Conditional compilation
      - #if / #ifdef / #ifndef / #else / #endif
    - Macros
      - #define

Not classes -- just files

- #include “filename.h”
  - Inserts text from one file into another before compilation
  - Contain info needed by other files to compile
    - Libraries – function signature
    - Classes – class interface (I.e. header file)

C++ is not Java

10. Not classes -- just files
9. Not packages -- just namespaces

Not packages -- just namespaces

- Like Java, C/C++ has a multitude of useful auxillary functions and classes in libraries
- Unlike Java, C++ does not have the notion of packages.
- C / C++ also doesn’t have nice javadocs
- Use `man` instead

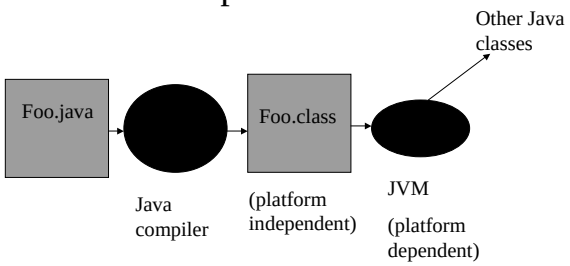
Not packages -- just namespaces

```
namespace foo {
  class bar { … };
}

foo::bar x;

using namespace foo;
bar x;
```

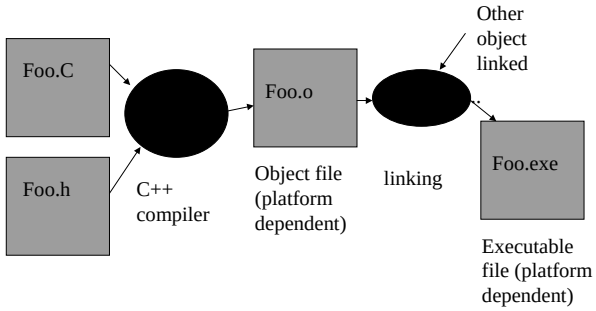
C++ is not Java

10. Not classes -- just files
9. Not packages -- just namespaces
8. We don’t need no stinkin’ VM

Compilation – Java



C++: We don't need no stinkin' VM



Compilation – Java vs C++

- Java
  - Compilation unit is the class
  - External classes located using a predefined path
  - JVM needed to execute
- C++
  - Compilation unit is the file
  - External classes require header and precompiled object file
  - Executable file need to execute.

Compilation – C++

- Use CC/g++ to compile individual files into object files
- Use CC/g++ to link object files into an executable file
  - Need to specify executable name, otherwise will be named a.out
- Run the executable file.

Using C Libraries

- Library header files must be #included for compilation
  - #include "mylib.h"
  - #include <math.h>
- Library object files must be linked when linking.
  - CC file file file –lm mylib.o

Managing C++ Projects

- `make`
  - Files to be compiled
  - Compiler options
  - Libraries
  - Dependencies
  - Make will build an executable
  - `gmakemake` – Makes Makefiles!

C++ is not Java

10. Not classes -- just files
9. Not packages -- just namespaces
8. We don't need no stinkin' VM
7. Universal Error Messages

Universal Error Messages

- If there is a problem…
  - `Bus error (core dumped)`
  - `Segmentation fault (core dumped)`

C++ is not Java

10. Not classes -- just files
9. Not packages -- just namespaces
8. We don't need no stinkin' VM
7. Universal Error Messages
6. The lonely function syndrome

The lonely function syndrome

- No top level "main" class
  - main() in its own file
  - main (int argc, char *argv[])
- Functions need not belong to a class

Functions

- Declaration vs definitions
  - Declaration is the function signature
    - Required before function can be used
    - Often included in a header file
    - Can exist in multiple files
    - `char *strcpy (char * to, char* from);`
  - Definition
    - Actual code
    - Must defined once and only once

Functions

- In C++ function arguments are pass by value:

```
int i = 7;
foo (i);
cout << i;
```

```
void foo (int j)
{
    cout << "Arg is " << j << endl;
    j = 12;
}
```

void – indicates that a function does not return a value

C++ is not Java

10. Not classes -- just files
9. Not packages -- just namespaces
8. We don't need no stinkin' VM
7. Universal Error Message
6. The lonely function syndrome
5. Sizing up a variable

Sizing up a variable

- sizeof
  - Will return the size of a data item or object
    - sizeof (char) = 1
    - sizeof (bool) = 1

C++ is not Java

10. Not classes -- just files
9. Not packages -- just namespaces
8. We don't need no stinkin' VM
7. Universal Error Message
6. The lonely function syndrome
5. Sizing up a variable
4. Streaming I/O

Basic IO

- Two types of I/O
  - C-style (stdio)
    - `#include <stdio.h>`
    - `fprintf (FILE *f, const char *format, …);`
    - `fscanf (FILE *f, const char *format, …);`
    - `FILE *stdin;`
    - `FILE *stdout;`
    - `FILE *stderr;`
  - Only standard datatypes supported

  - `man –s 3C stdio`

Basic I/O

```
#include <stdio.h>
int a = 7;
float b = 6.4;
char *foo = "myString"
printf ("%d\t%f\t%s\n", a, b, foo);


7 6.4 myString
```

Basic I/O

```
#include <stdio.h>
int a;
float b;
char foo[10];
scanf ("%d\t%f\t%x\n", &a, &b, foo);


7 6.4  myString
```

Basic IO

- Two types of I/O
  - C++ style (I/O Streams)
    - << for output
    - >> for input
    - cin – standard input
    - cout – standard output
    - cerr – standard error

Basic I/O

```
#include <iostream>
using std

int a = 7;
float b = 6.4;
char *foo = “myString”;
cout << a << ‘\t’ << b << ‘\t’ << foo << endl;


7  6.4  myString
```

Basic I/O

```
#include <iostream>
using std

int a;
float b;
char foo[10];
cin >> a >> b >> foo(10);


7 6.4  myString
```

C++ is not Java

10. Not classes -- just files
9. Not packages -- just namespaces
8. We don’t need no stinkin’ VM
7. Universal Error Message
6. The lonely function syndrome
5. Sizing up a variable
4. Streaming I/O
3. Questionable statements

Statements

- `if` (condition) statement
- `if` (condition) statement `else` statement
- `switch` (condition) statement
- `while` (statement) statement
- `do` statement `while` (expression)
- `for` (; ;) statement
  - Statements can be nested blocks of code
    - I.e { … }

Statements

- Exiting a loop
  - break – exit the loop
  - continue – perform next iteration of a loop
  - goto – go anywhere

Statements

- Logical conditions
  - Are short cuircuited
  - `if ( ( a < b) && ( c > d)) { …}`
    - If (a > b), (c > d) will not get tested.

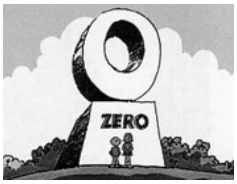
Statements

- Finally, my favorite C++ statement
  - var = (condition) ? statement1 : statement2
  - Same as:
    - `if` (condition)
      `var` = expression1
      else `var` = expression2
    - `min = (a <= b) ? a : b;`

C++ is not Java

10. Not classes -- just files
9. Not packages -- just namespaces
8. We don't need no stinkin' VM
7. Universal Error Message
6. The lonely function syndrome
5. Sizing up a variable
4. Streaming I/O
3. Questionable statements
2. My hero zero

My Hero Zero



Statements

- Logical conditions
  - False if condition evaluates to 0
  - True if condition evaluates to a non-zero value.
  - Can be interpreted as condition != 0

Statements

- Logical conditions
  - `int a;`
  - `if (a) { ... } // same as`
  - `if (a != 0) { ... }`
  - `int *b;`
  - `if (b) { ... } // same as`
  - `if (b != 0) { ... }`

Statements

- Logical conditions
  - Assignments and declarations can be made in a logical condition
  - The following is valid:
    - `if ( double d = somefunction (a)) { … }`
  - A most common mistake
    - `if ( a = b ) { … }  // is not the same as`
    - `if ( a == b) { … }`

C++ is not Java

10. Not classes -- just files
9. Not packages -- just namespaces
8. We don't need no stinkin' VM
7. Universal Error Message
6. The lonely function syndrome
5. Sizing up a variable
4. Streaming I/O
3. Questionable statements
2. My hero zero
1. They didn't write it for you!

Some code -- first.cpp

```
#include <iostream>
#include "volume.h"

int main (int argc, char *argv[])
{
  float ht, wd;
  long int dp;
  double v;

  // Get dimensions
  std::cout << "Enter height (x.xx), width (x.xx), depth (x)";
  std::cin >> ht >> wd >> dp;

  // Calculate
  v = Volume::calcVolume (ht, wd, dp);

  // Print results
  std::cout << "Volume is " << v << std::endl;
  std::cout << "Note that height is still " << ht << std::endl;

  return 0;
}
```

Some code -- volume.h

```
#ifndef VOLUME_INCLUDED
#define VOLUME_INCLUDED

namespace Volume {

// Prototype for calVolume function
double calcVolume (double, double, long);

}

#endif
```

Some code -- volume.cpp

```
namespace Volume {

double calcVolume (double height,
double width, long depth)
{
  height = 7.7;
  return height * width * depth;
}

}
```

Running

```
gmakemake > Makefile
make
first
Enter height (x.xx), width (x.xx), depth (x)
2.51 3.45 8
Volume is 69.276
Note that height is still 2.51
```

Questions?

- Next time
  - C++ Variables.