the gamedesigninitiative
at cornell university

C++:
The Basics

So You Think You Know C++

- Most of you are experienced Java programmers
  - Both in 2110 and several upper-level courses
  - If you saw C++, was likely in a systems course
- Java was based on C++ syntax
  - Marketed as "C++ done right"
  - Similar with some important differences
- **This Lecture**: an overview of the differences
  - If you are a C++ expert, will be review

So You Think You Know C++

- Most of you are experienced Java programmers
  - Both in 2110 and several upper-level ces
  - If you saw C++, was li rse
- Java
  - Ma
  - Sim some important differences
- **This Lecture**: an overview of the differences
  - If you are a C++ expert, will be review

All the sample code is online.
Download and **play with it**.

Comparing Hello World

Java

```
/* Comments are single or multiline
 */

// Everything must be in a class
public class HelloWorld {

  // Application needs a main METHOD
  public static void main(String arg[]){

    System.out.println("Hello World");

  }

}
```

C++

```
/*Comments are single or multiline
 */

// Nothing is imported by default
#include <stdio.h>

// Application needs a main FUNCTION
int main(){

    printf("Hello World");
    printf("\n");    // Must add newline

    // Must return something
    return 0;
}
```

Comparing Hello World

Java

```
/* Comments are single or multiline
 */

// Everything must be in a class
public class HelloWorld {

   // Application needs a main METHOD
   public static void main(String arg[]){

      System.out.println("Hello World");

   }

}
```

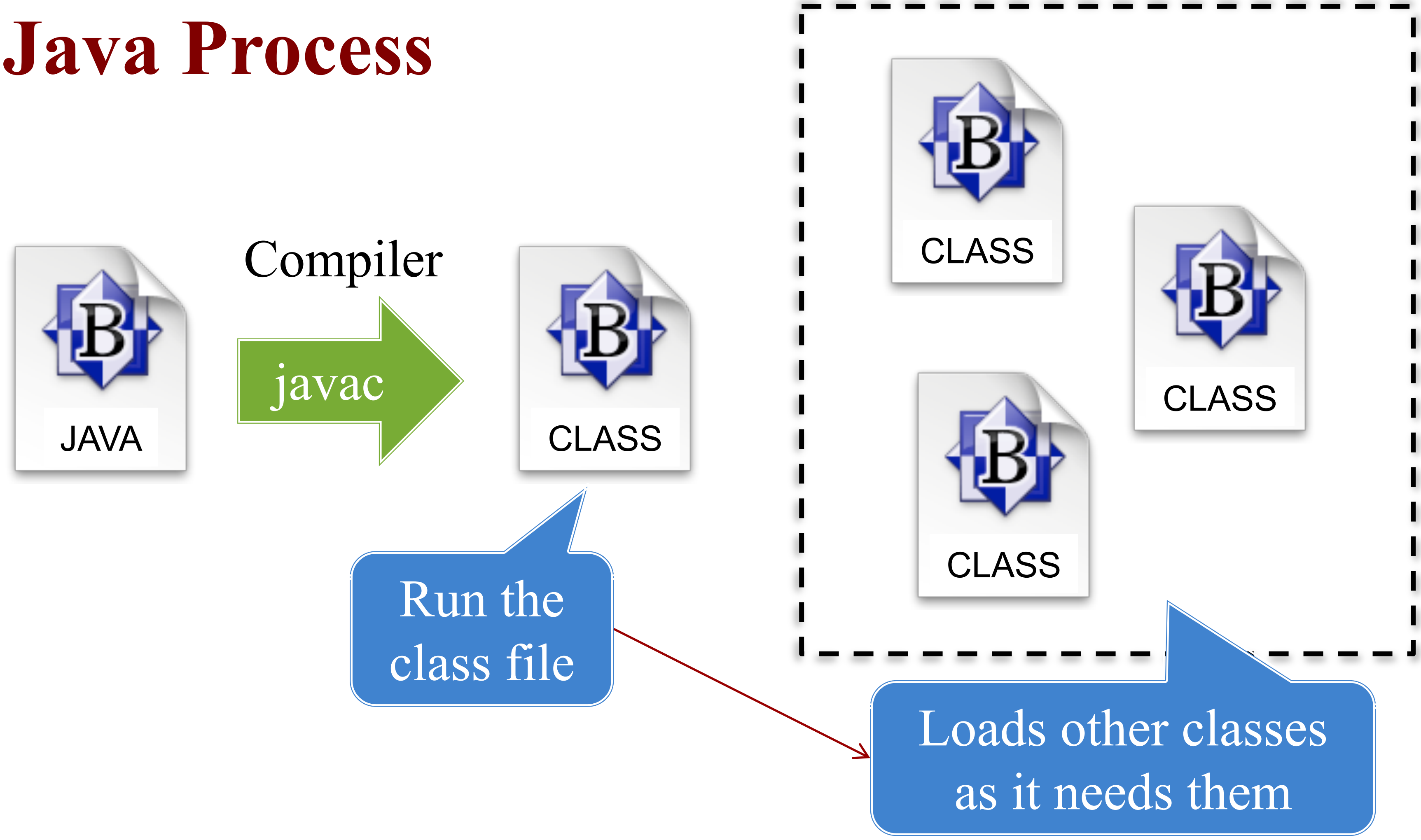
C++

```
/*Comments are single or multiline
 */

// Nothi
#includ

// Appli                                    TION
int main(){

   printf("Hello World");
   printf("\n");    // Must add newline

   // Must return something
   return 0;
}
```

C-style console.
In CUGL, use
CULog instead.

Biggest Difference: **Compilation**

Java Process

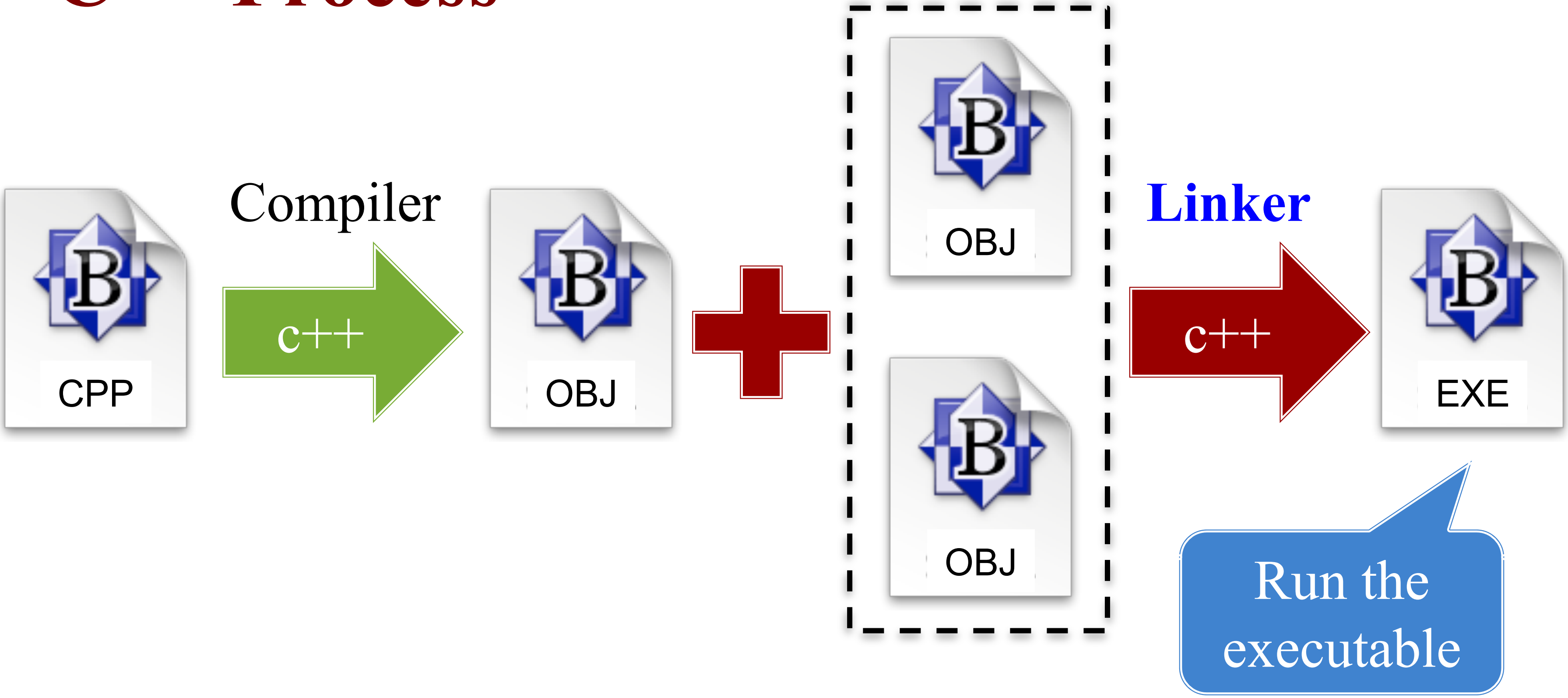


Run the class file

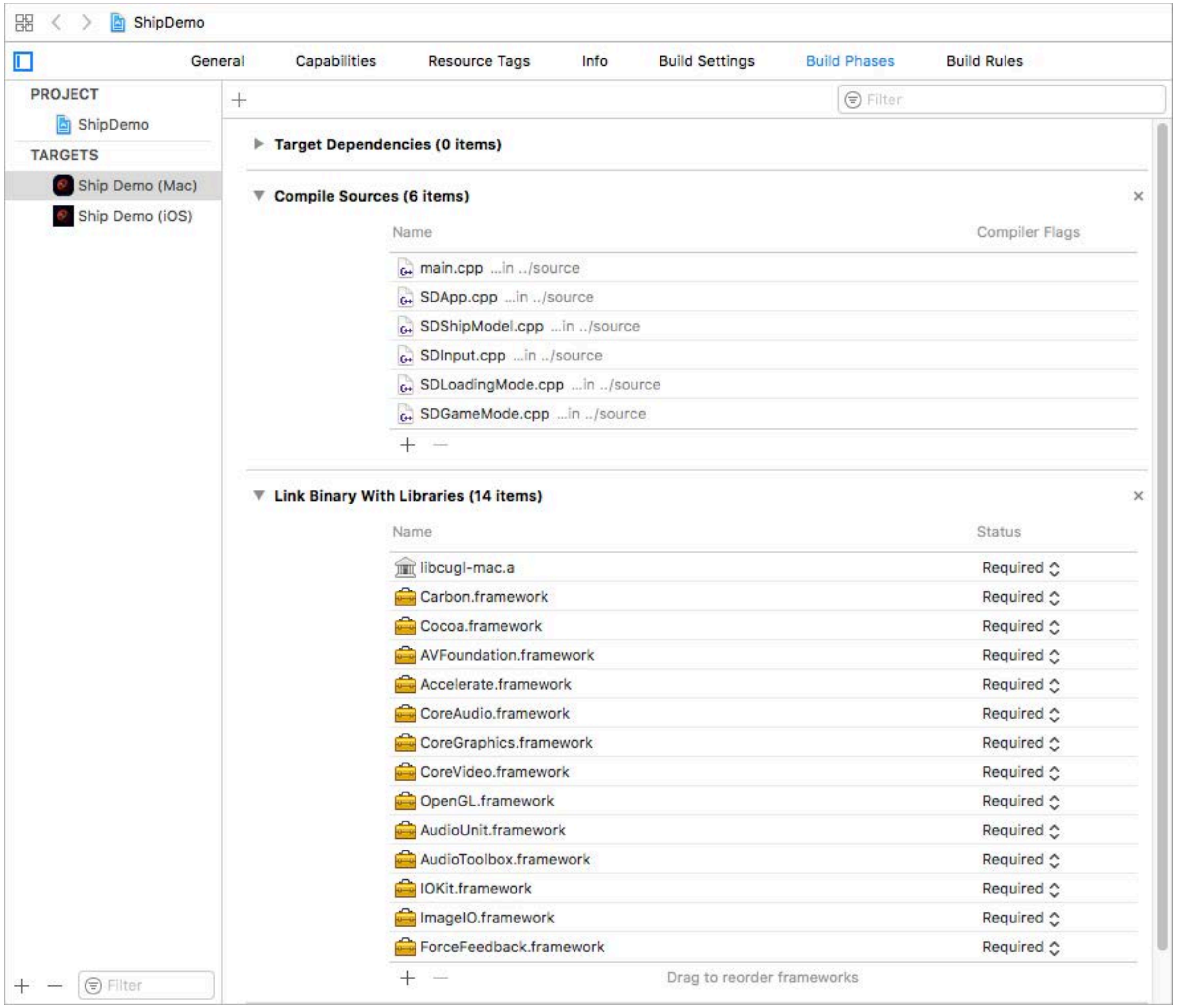
Loads other classes as it needs them

Biggest Difference: **Compilation**

C++ Process



All Handled by the IDE



▶ Target Dependencies (0 items)

▼ Compile Sources (6 items)

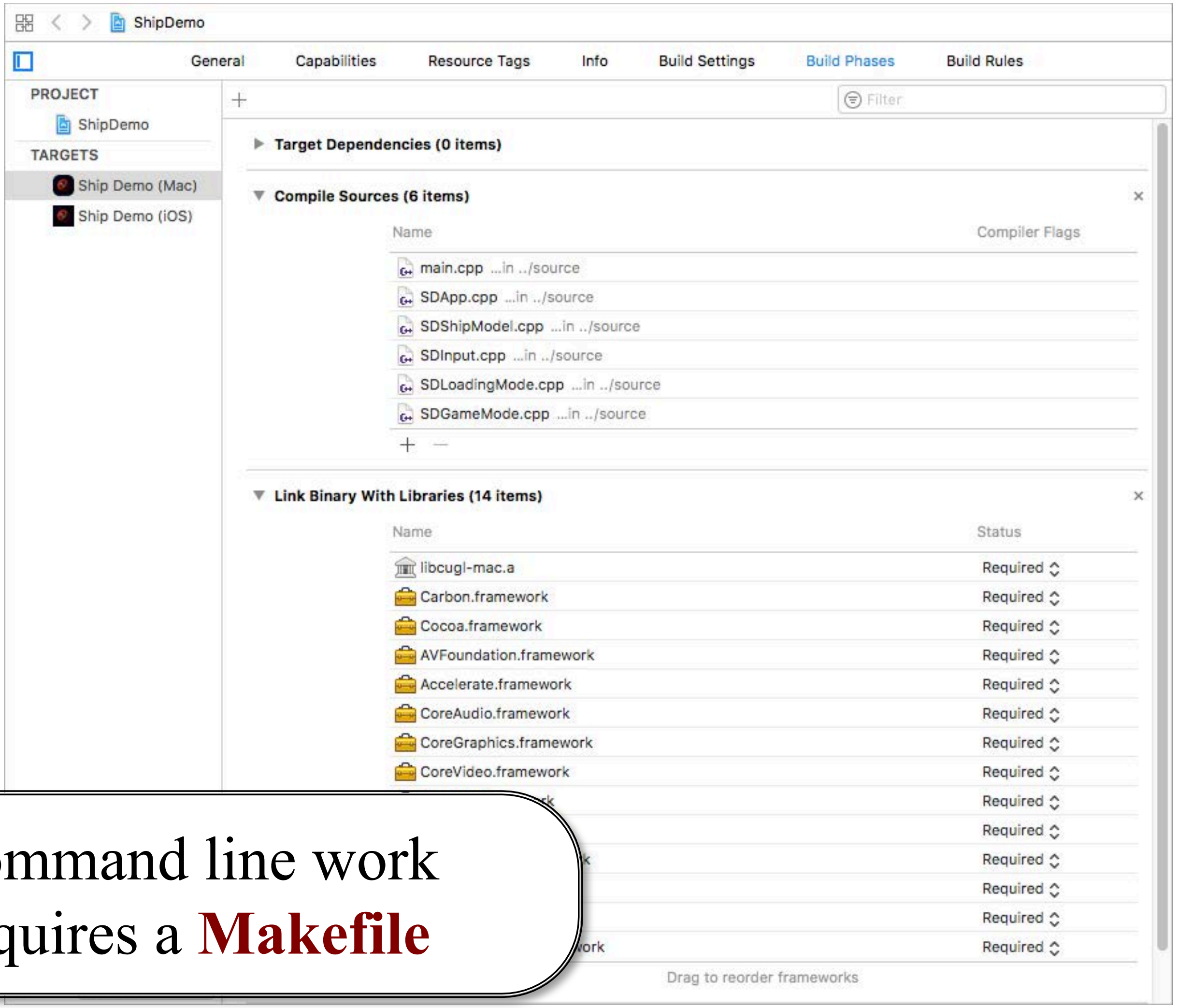
Name	Compiler Flags
main.cpp ...in ../source	
SDApp.cpp ...in ../source	
SDShipModel.cpp ...in ../source	
SDInput.cpp ...in ../source	
SDLoadingMode.cpp ...in ../source	
SDGameMode.cpp ...in ../source	

▼ Link Binary With Libraries (14 items)

Name	Status
libcugl-mac.a	Required
Carbon.framework	Required
Cocoa.framework	Required
AVFoundation.framework	Required
Accelerate.framework	Required
CoreAudio.framework	Required
CoreGraphics.framework	Required
CoreVideo.framework	Required
OpenGL.framework	Required
AudioUnit.framework	Required
AudioToolbox.framework	Required
IOKit.framework	Required
ImageIO.framework	Required
ForceFeedback.framework	Required

Drag to reorder frameworks

All Handled by the IDE



▶ Target Dependencies (0 items)

▼ Compile Sources (6 items)

Name	Compiler Flags
main.cpp ...in ../source	
SDApp.cpp ...in ../source	
SDShipModel.cpp ...in ../source	
SDInput.cpp ...in ../source	
SDLoadingMode.cpp ...in ../source	
SDGameMode.cpp ...in ../source	

▼ Link Binary With Libraries (14 items)

Name	Status
libcugl-mac.a	Required
Carbon.framework	Required
Cocoa.framework	Required
AVFoundation.framework	Required
Accelerate.framework	Required
CoreAudio.framework	Required
CoreGraphics.framework	Required
CoreVideo.framework	Required
k	Required
	Required
k	Required
	Required
	Required
ork	Required

Drag to reorder frameworks

Command line work requires a **Makefile**

Makefile Format

```
# Makefile comment (Python style)

# Variables.  In case we wanted to swap compilers
CC=c++

# Main application is first.  If you type "make" by itself, you get this.
app: main.o helper.o
    $(CC) -o app main.o helper.o

# The object files (pre-linker). Type "make main.o" to get this.
main.o: main.cpp main.h helper.h
    $(CC) -c main.cpp

helper.o: helper.cpp helper.h
    $(CC) -c helper.cpp
```

Makefile Format

```
# Makefile comment (Python style)

# Variables.  In case we wanted to swap compilers

# Main application first.  If you type "make" by itself, you get this.
app: main.o helper.o
	$(CC) -o app main.o helper.o

# T                  inker). Type "make main.o" to get this.
ma                   h helper.h
	$(CC) -c main.cpp

helper.o: helper.cpp helper.h
	$(CC) -c helper.cpp
```

Target

Dependencies

Do if target not there or older than dependencies

Evaluates variable

Makefile Format

```makefile
# Makefile comment (Python style)

# Variables.  In case we wanted to swap compilers
CC=c++

# Main application is first.  If you type "make" by itself, you get this.
app: main.o helper.o
	$(CC) -o app main.o helper.o
```

Linker step

```makefile
# The object files (pre-linker). Type "make main.o" to get this.
main.o: main.cpp main.h helper.h
	$(CC) -c main.cpp
```

Compiler step

```makefile
helper.o: helper.cpp helper.h
	$(CC) -c helper.cpp
```

Separation Requires Header Files

- Need #include for libs
  - But linker adds the libs
  - So what are we including?
- **Function Prototypes**
  - Declaration without body
  - Like an interface in Java
- Prototypes go in .h files
  - Also includes types, classes
  - May have own #includes

```
/* stringfun.h
 * Recursive string funcs in CS 1110
 */

#ifndef_STRINGFUN_H_
#define_STRINGFUN_H_

#include <string>

/* True if word a palindrome */
bool isPalindrome(string word);

/* True if palindrome ignore case */
bool isLoosePalindrome(string word);

#endif
```

Separation Requires Header Files

- Need #include for libs
  - But linker adds the libs
  - So what are we including?
- **Function Prototypes**
  - Declaration without body
  - Like an interface in Java
- Prototypes go in .h files
  - Also includes types, classes
  - May have own #includes

```
/* stringfun.h
 * Recursive string funcs in CS 1110
 */

#ifndef_STRINGFUN_H_
#define_STRINGFUN_H_

#include <

/* Tr
bool i

/* Tr
bool isLoosePalindrome(string word);

#endif
```

Prevents inclusion more than once (which is an error)

Separation Requires Header Files

- Need #include for libs
  - But linker adds the libs
  - So what are we including?
- **Function Prototypes**
  - Declaration without body
  - Like an interface in Java
- Prototypes go in .h files
  - Also includes types, classes
  - May have own #includes

```
/* stringfun.h
 * Recursive string funcs in CS 1110
 */

#ifndef _STRINGFUN
#define _STRINGFUN

#include <string>

/* True if word a palindrome */
bool isPalindrome(string word);

/* True if palindrome ignore case */
bool isLoosePalindrome(string word);

#endif
```

Type not built-in

Headers and Namespaces

- Headers are not packages!
  - Java import is very different
  - Packages prevent collisions
- C++ has **namespaces**
  - Define it in the header file
  - In-between curly braces
- Must add prefix when used
  - stringfun::isPalindrome(..)
  - *Even in implementation!*
- Unless have using command

```
/* stringfun.h */

#ifndef_STRINGFUN_H_
#define_STRINGFUN_H_

#include <string>

namespace stringfun {

  /* True if word a palindrome */
  bool isPalindrome(string word);

  /* True if palindrome ignore case */
  bool isLoosePalindrome(string word);

}

#endif
```

Headers and Namespaces

- Headers are not packages!
  - Java import is very different
  - Packages prevent collisions
- C++ has **namespaces**
  - Define it in the header file
  - In-between curly braces
- Must add prefix when used
  - stringfun::isPalindrome(..)
  - *Even in implementation!*
- Unless have using command

```
/* stringfun.cpp */

#include "stringfun.h"

/* True if word a palindrome */
bool stringfun::isPalindrome(string w)
{

  if (s.size() < 2) {
     return true;
  }

   string sub = s.substr(1,s.size()-2);
   return s[0] == s[s.size()-1] &&
      stringfun::isPalindrome(sub);
}
```

Headers and Namespaces

- Headers are not packages!
  - Java import is very different
  - Packages prevent collisions
- C++ has **namespaces**
  - Define it in the header file
  - In-between curly braces
- Must add prefix when used
  - stringfun::isPalindrome(..)
  - *Even in implementation!*
- Unless have using command

```
/* stringfun.cpp */

#include "stringfun.h”

using namespace stringfun;

/* True if word a palindrome */
bool stringfun::isPalindrome(string w)
{

  if (s.size() < 2) {
     return true;
  }

   string sub = s.substr(1,s.size()-2);
   return s[0] == s[s.size()-1] &&
      isPalindrome(sub);
}
```

Headers and Namespaces

- Headers are not packages!
  - Java import is very different
  - Packages prevent collisions
- C++ has **namespaces**
  - Define it in the header file
  - In-between curly braces
- Must add prefix when used
  - stringfun::isPalindrome(..)
  - *Even in implementation!*
- Unless have using command

```
/* stringfun.cpp */

#include "stringfun.h"

using namespace stringfun;

/* True if word a palindrome */
bool stringfun::isPalindrome(string w)
{

   if (s.
      return true;
   }

   string s                            ze()-2);
   return s[0]      s[s.size()-1] &&
      isPalindrome(sub);
}
```

Prefix here

Not here

Pointers vs References

Pointer	Reference
Variable with a * modifier	Variable with a & modifier
Stores a memory location	Refers to another variable
Can modify as a parameter	Can modify as a parameter
Must dereference to use	No need to dereference
Can allocate in heap	Cannot allocate in heap

Java's reference variables are a combination of the two

Pointers vs References

Pointer

- Variable with a * modifier
- Stores a
- Can mo
- Must de
- Can allocate in heap

Reference

- Variable with a & modifier
- Refers to another variable
- Can modify as a parameter
- No need to dereference
- Cannot allocate in heap

Safer!

Preferred if do not need heap

Java's reference variables are a combination of the two

When Do We Need the Heap?

- To **return** a non-primitive
  - Return value is on the stack
  - Copied to stack of caller
  - Cannot copy if size variable
- Important for arrays, objects
  - But objects can cheat…

```
int* makearray(int size) {
    // Array on the stack
    int result[size];

    // Initialize contents
    for(int ii = 0; ii < size; ii++) {
        result[ii] = ii;
    }

    return result; // BAD!
}
```

0x7ed508	???
0x7ed528	4
0x7ed548	0
0x7ed568	1
0x7ed588	2
0x7ed5a8	3

return

0x7ed508	0x7ed548

address does not exist

Allocation and Deallocation

Not An Array

- Basic format:

```
type* var = new type(params);
…
delete var;
```

- Example:
  - `int* x = new int(4);`
  - `Point* p = new Point(1,2,3);`
- One you use the most

Arrays

- Basic format:

```
type* var = new type[size];
…
delete[] var; // Different
```

- Example:
  - `int* array = new int[5];`
  - `Point* p = new Point[7];`
- Forget [] == memory leak

Strings are a Big Problem

- Java string operations allocate to the heap
  - $s = \overbrace{\underbrace{\texttt{"The point is ("+x}}_{\text{allocate}}\texttt{+","}}^{\text{allocate}}\texttt{+y+")"}$
- How do we manage these in C++?
  - For char*, we don’t. Operation + is illegal.
  - For string, we can use + but it comes at a cost
- **Idea**: Functions to remove string memory worries
  - Formatters like `printf` and `CULog` for direct output
  - Stream buffers to cut down on extra allocations

Displaying Strings in C++

C-Style Formatters

- `printf(format,arg1,arg2,...)`
  - Substitute into % slots
  - Value after % indicates type
- Examples:
  - `printf("x = %d",3)`
  - `printf("String is %s","abc")`
- Primarily used for output
  - Logging/debug (`CULog`)
  - Very efficient for output

C++ Stream Buffers

- `strm << value << value << ...`
  - Easy to chain arguments
  - But exact formatting tricky
- Example:
  - `cout << "x = " << 3 << endl`
  - `stringstream s << "x = " << 3`
- Great if you need to **return**
  - More efficient than + op
  - Can concatenate non-strings

How Does Concatenation Work?

- String operations allocate
  - Each string needs memory
  - String ops are expensive
  - C++11 has optimized a lot
- Memory may be on **stack**
  - Almost never new strings
  - Return/parameters copied
  - Will see implications later
- What does this mean?
  - Simple operations are okay
  - Otherwise use stringstream

```
void foo() {
    string a = "Hello"; // Stack

    string b("Hello");  // Stack

    // THIS is on the heap
    string* c = new string("Hello");

    string d = a+" World";   // Stack

    string e = *c+" World"; // Stack

    // Copies to next frame in stack
    return e;

    // a, b, d, e are deleted
    //  c is still in heap
}
```

Next Time: Classes and Closures