the gamedesigninitiative
at cornell university

Lecture 9

C++: 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

C++ Overview

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 li~~...~~ ~~...cou~~rse

- Java ~~...~~
  - Ma~~...~~
  - Simi~~...~~ ~~...some~~ important differences

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

- **This Lecture**: an overview of the differences

  - If you are a C++ expert, will be review

C++ Overview

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;

}
```

the gamedesigninitiative
at cornell university

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
#include

// Appli                              TION
int main(){

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

    // Must return something
    return 0;

}
```
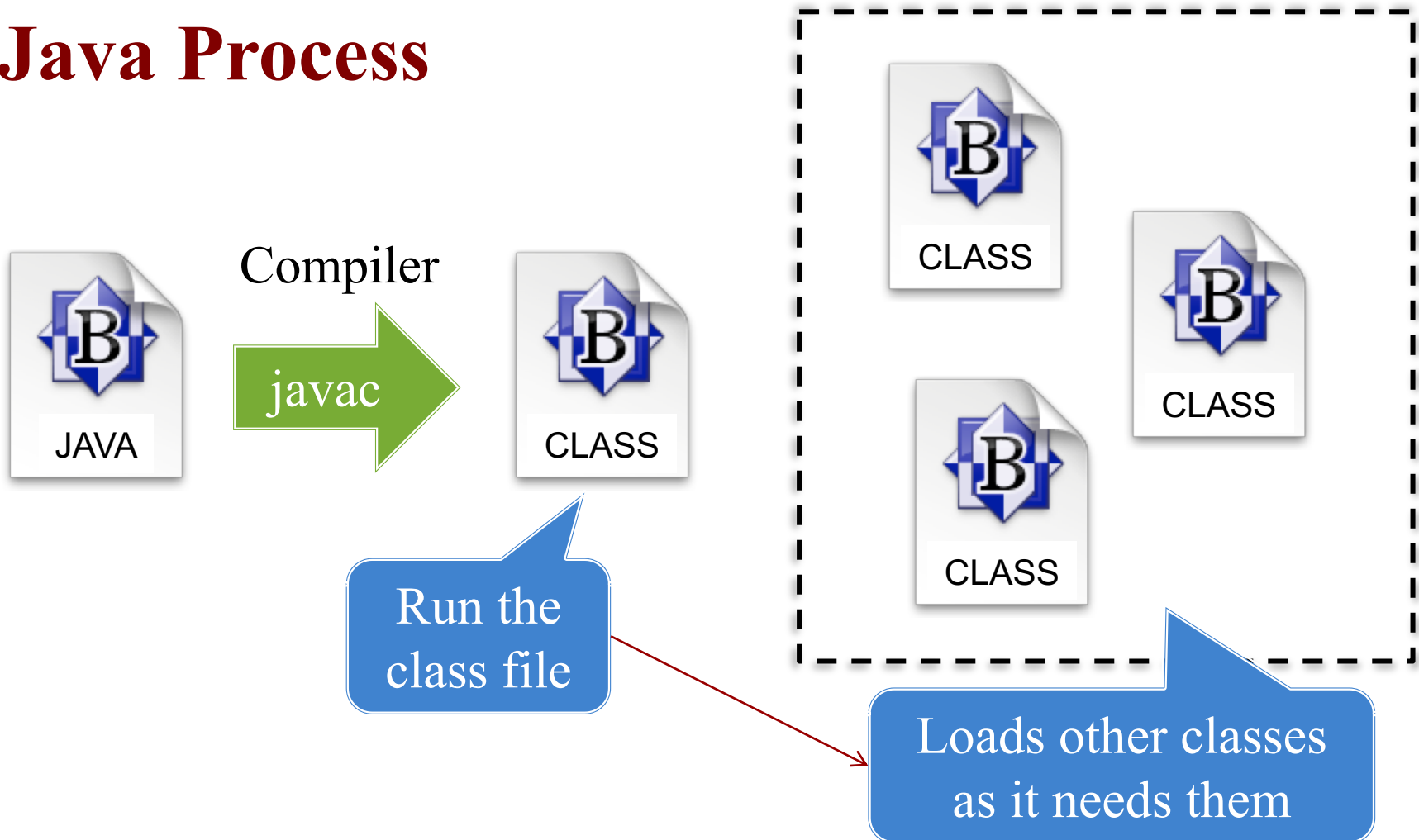
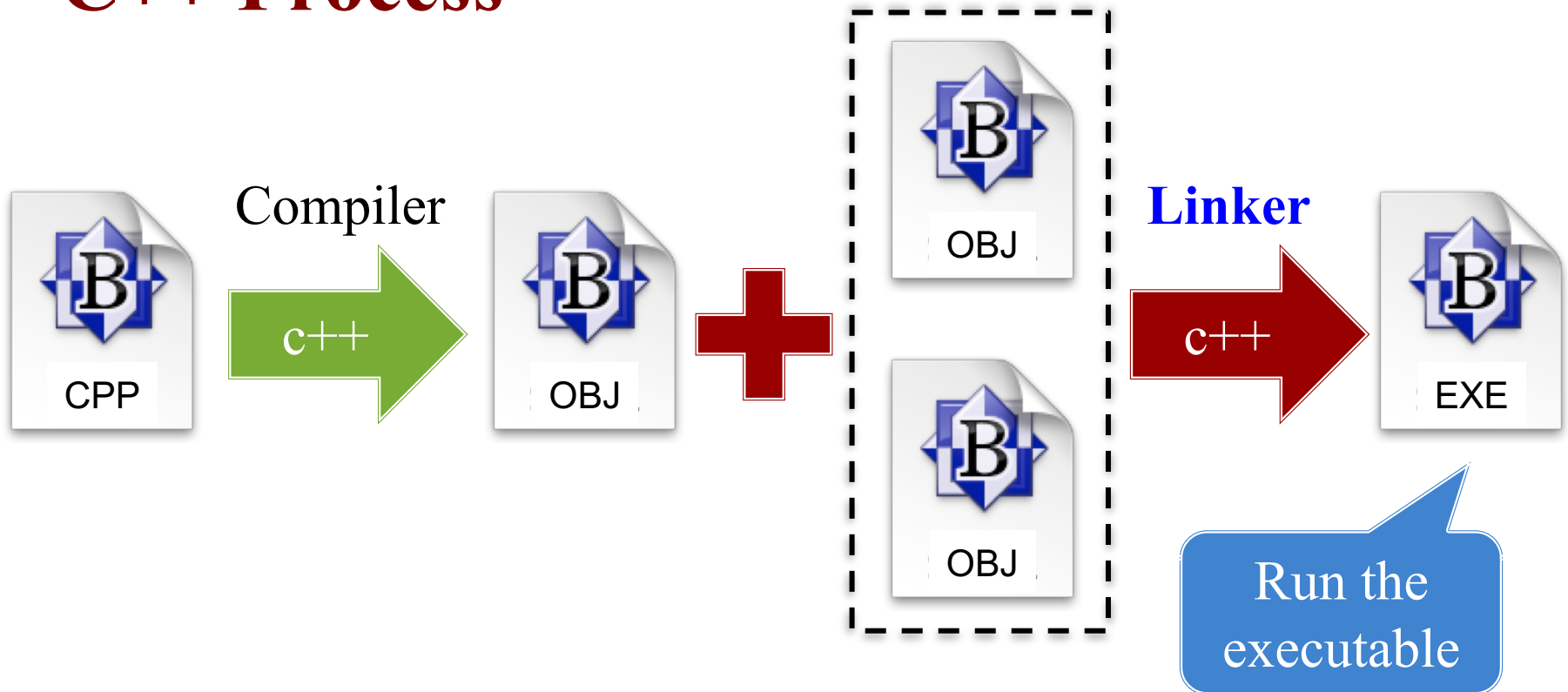> C-style console.
> In CUGL, use
> CULog instead.

the game design initiative
at cornell university

Biggest Difference: **Compilation**

Java Process

Compiler

javac

JAVA → CLASS

Run the class file

Loads other classes as it needs them

C++ Overview

Biggest Difference: **Compilation**

C++ Process
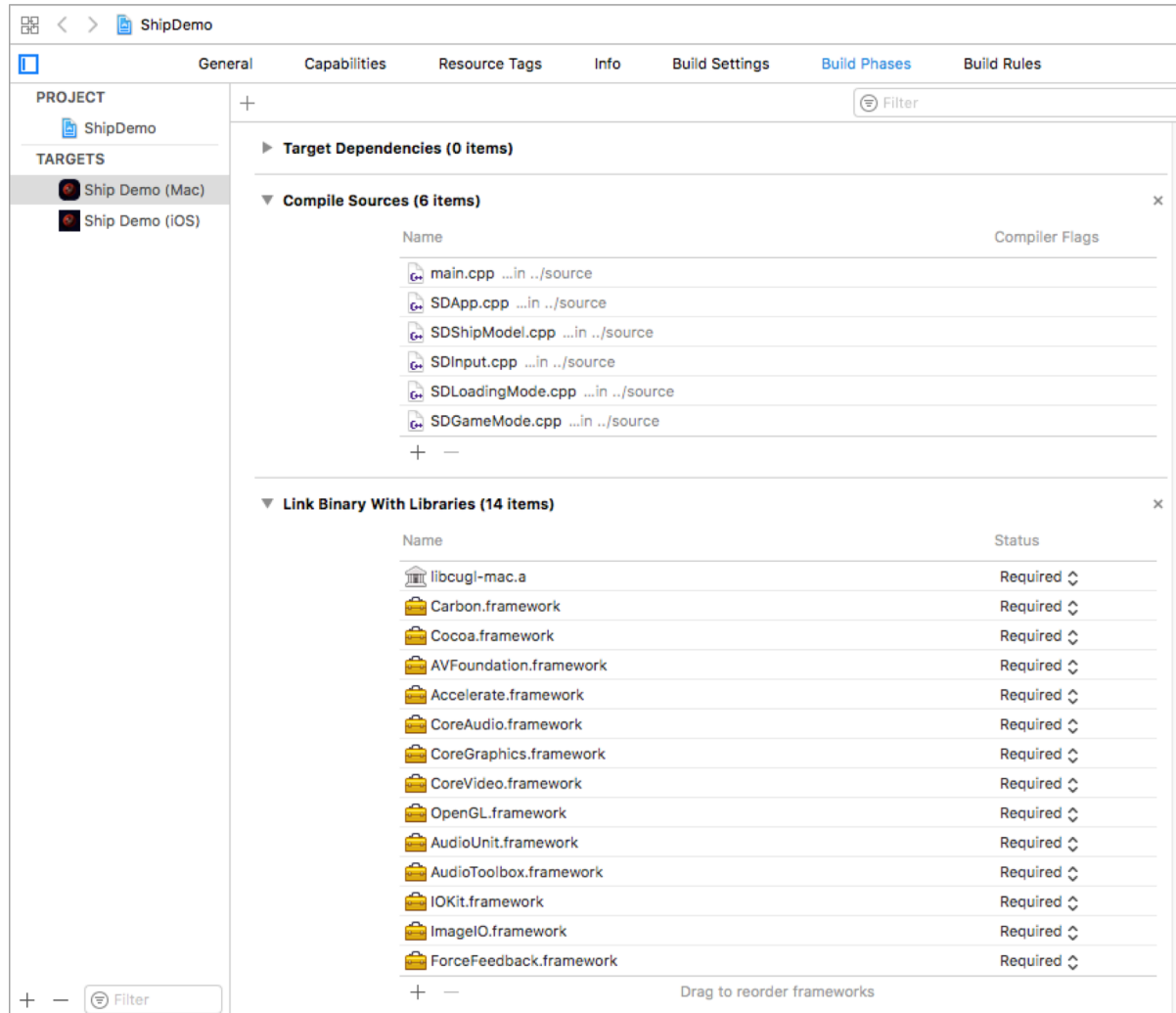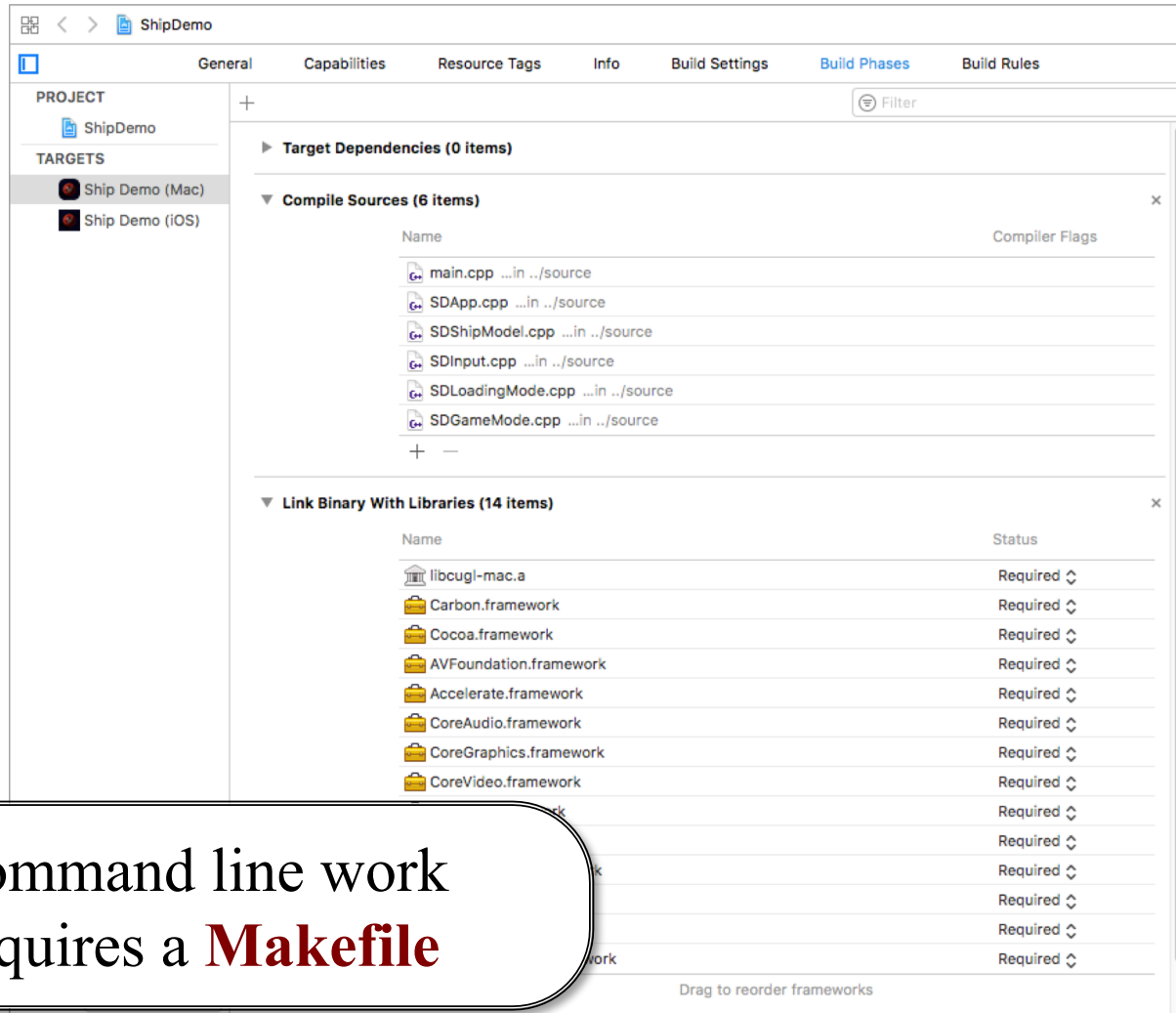
Compiler

c++

CPP

OBJ

+

OBJ

OBJ

Linker

c++

EXE

Run the executable

C++ Overview

All Handled by the IDE

C++ Overview

All Handled by the IDE

Command line work requires a **Makefile**

C++ Overview

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
```

the gamedesigninitiative
at cornell university

Makefile Format

```
# Makefile comment (Python style)

# Variables.  In case we wanted to swap compilers

# 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 linker). Type "make main.o" to get this.
main.o: main.cpp 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

the game**design**initiative
at cornell university

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
```

Linker step

```
# 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

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

C++ Overview

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
```

C++ Overview

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)

the gamedesigninitiative
at cornell university

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

the **game**design**initiative**
at cornell university

Headers and Namepaces

- 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
```

the game design initiative
at cornell university

Headers and Namepaces

- 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);
}
```

C++ Overview

Headers and Namepaces

- 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);

}
```

C++ Overview

Headers and Namepaces

- 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

```cpp
/* stringfun.cpp */

#include "stringfun.h"

using namespace stringfun;

/* True if word a palindrome */
bool stringfun::isPalindrome(string w)
{
  if (s.          Prefix here
      return true;
  }

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

Prefix here

Not here

Pointers vs References

Pointer

- Variable with a * modifier
- Stores a memory location
- Can modify as a parameter
- Must dereference to use
- 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

Java's reference variables are a combination of the two

C++ Overview

the gamedesigninitiative
at cornell university

Pointers vs References

Pointer

- Variable with a * modifier
- Stores a ~~memory location~~
- Can m~~odify~~
- Must de~~reference~~
- 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

C++ Overview

the **gamedesigninitiative**
at cornell university

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…

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

return

0x7ed508	0x7ed548

address does not exist

```cpp
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!
}
```

C++ Overview

the gamedesigninitiative
at cornell university

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

C++ Overview

Strings are a Big Problem

- Java string operations allocate to the heap

allocate

s = "The point is ("+x+","+y+")"

allocate

- 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

24

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

the gamedesigninitiative
at cornell university

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

```cpp
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
}
```

C++ Overview

Next Time: Classes and Closures

C++ Overview