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
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 some important differences

- Java
  - Marked as "C++ done right"

- 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;
}

C++ Overview
Comparing Hello World

**Java**

```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++**

```cpp
/* 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;
}
```

C-style console. In CUGL, use CULog instead.
Biggest Difference: **Compilation**

Java Process

1. **Compiler**
   - `javac`

2. **Run the class file**

3. **Loads other classes as it needs them**
Biggest Difference: Compilation

C++ Process

Compiler: CPP → OBJ

Linker: OBJ → EXE

Run the executable
All Handled by the IDE

C++ Overview
All Handled by the IDE

Command line work requires a 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

# 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 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

Target Dependencies

Do if target not there or older than dependencies

Evaluates variable
# 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
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`

```c++
#include <string>

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

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

```c++
#ifndef _STRINGFUN_H_
#define _STRINGFUN_H_

#include <string>

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

#ifndef STRINGFUN_H
#define STRINGFUN_H

#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

```c++
/* 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
```

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**

```cpp
/* 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
```

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

```c++
/* 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
```

C++ Overview
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

```cpp
/* 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
# Pointers vs References

<table>
<thead>
<tr>
<th>Pointer</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable with a * modifier</td>
<td>Variable with a &amp; modifier</td>
</tr>
<tr>
<td>Stores a memory location</td>
<td>Refers to another variable</td>
</tr>
<tr>
<td>Can modify as a parameter</td>
<td>Can modify as a parameter</td>
</tr>
<tr>
<td>Must dereference to use</td>
<td>No need to dereference</td>
</tr>
<tr>
<td>Can allocate in heap</td>
<td>Cannot allocate in heap</td>
</tr>
</tbody>
</table>

Java’s reference variables are a combination of the two
# Pointers vs References

<table>
<thead>
<tr>
<th>Pointer</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Variable with a * modifier</td>
<td>- Variable with a &amp; modifier</td>
</tr>
<tr>
<td>- Stores a memory location</td>
<td>- Refers to another variable</td>
</tr>
<tr>
<td>- Can modify as a parameter</td>
<td>- Can modify as a parameter</td>
</tr>
<tr>
<td>- Must dereference to use</td>
<td>- No need to dereference</td>
</tr>
<tr>
<td>- Can allocate in heap</td>
<td>- Cannot allocate in heap</td>
</tr>
</tbody>
</table>

Java’s reference variables are a combination of the two

**Safer!**
Preferred if do not need heap
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…

```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

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7ed508</td>
<td>???</td>
</tr>
<tr>
<td>0x7ed528</td>
<td>4</td>
</tr>
<tr>
<td>0x7ed548</td>
<td>0</td>
</tr>
<tr>
<td>0x7ed568</td>
<td>1</td>
</tr>
<tr>
<td>0x7ed588</td>
<td>2</td>
</tr>
<tr>
<td>0x7ed5a8</td>
<td>3</td>
</tr>
</tbody>
</table>

---

The game design initiative at Cornell University
## Allocation and Deallocation

### Not An Array

- Basic format:
  
  ```cpp
type* var = new type(params);
... delete var;
```

- Example:
  
  ```cpp
  int* x = new int(4);
  Point* p = new Point(1,2,3);
  ```

- One you use the most

### Arrays

- Basic format:
  
  ```cpp
type* var = new type[size];
... delete[] var; // Different
```

- Example:
  
  ```cpp
  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
- \( s = \text{"The point is (\"+x+\",\"+y+\")"} \)

- 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++

<table>
<thead>
<tr>
<th>C-Style Formatters</th>
<th>C++ Stream Buffers</th>
</tr>
</thead>
<tbody>
<tr>
<td>printf(format, arg1, arg2, ...)</td>
<td>strm &lt;&lt; value &lt;&lt; value &lt;&lt; ...</td>
</tr>
<tr>
<td>• Substitute into % slots</td>
<td>• Easy to chain arguments</td>
</tr>
<tr>
<td>• Value after % indicates type</td>
<td>• But exact formatting tricky</td>
</tr>
<tr>
<td>• Examples:</td>
<td>• Example:</td>
</tr>
<tr>
<td>• printf(&quot;x = %d&quot;, 3)</td>
<td>• cout &lt;&lt; &quot;x = &quot; &lt;&lt; 3 &lt;&lt; endl</td>
</tr>
<tr>
<td>• printf(&quot;String is %s&quot;, &quot;abc&quot;)</td>
<td>• stringstream s &lt;&lt; &quot;x = &quot; &lt;&lt; 3</td>
</tr>
<tr>
<td>• Primarily used for output</td>
<td>• Great if you need to return</td>
</tr>
<tr>
<td>• Logging/debug (CULog)</td>
<td>• More efficient than + op</td>
</tr>
<tr>
<td>• Very efficient for output</td>
<td>• Can concatenate non-strings</td>
</tr>
</tbody>
</table>
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
}
```
Next Time: Classes and Closures