Lecture 7

C++ Overview
Lecture 7

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 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
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:
- 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

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. Similar to CCLog, used by Cocos2d-x
Biggest Difference: **Compilation**

**Java Process**

- **Compiler**
  - javac

- **Run the class file**

- **Loads other classes as it needs them**

---

**Compilation Process**

1. **Java Process**
   - **Compiler** (javac)
     - **CLASS**
     - **CLASS**
     - **CLASS**

2. **Run the class file**

3. **Loads other classes as it needs them**

---

**Biggest Difference: Compilation**
Biggest Difference: Compilation

C++ Process

Compiler

```
CPP
```

```
c++
```

```
OBJ
```

Linker

```
c++
```

```
OBJ
```

```
OBJ
```

```
EXE
```

Run the executable
All Handled by the IDE

- Copy Bundle Resources (3 items)
- Compile Sources (7 items)
  - FilmStrip.cpp ...
  - GameController.cpp ...
  - InputController.cpp ...
  - ResourceLoader.cpp ...
  - Ship.cpp ...
  - AppDelegate.cpp ...
  - main.cpp ...
- Link Binary With Libraries (11 items)
  - libcocos2d Mac.a
  - libcurl.dylib
  - libbz.dylib
  - IOKit.framework
  - OpenGL.framework
  - AppKit.framework
  - Foundation.framework
  - QuartzCore.framework
  - OpenAL.framework
  - AVFoundation.framework
  - Audio Toolbox.framework

Drag to reorder frameworks
All Handled by the IDE

Command line work requires a **Makefile**
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`
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`

```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
```

Type not built-in
# 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

---

**C++ Overview**

---

**game design initiative at cornell university**
# 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>

**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

```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>0x7ed508</th>
<th>0x7ed548</th>
<th>0x7ed568</th>
<th>0x7ed588</th>
<th>0x7ed5a8</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

```
return
```

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

- How do we manage these in C++?
  - For char*, we don’t. Operation + is illegal.
  - For string, it is complicated. Later in lecture

- **Idea**: Functions to remove string memory worries
  - Formatters like printf/CCLLog for direct output
  - Stream buffers to cut down on extra allocations
Managing 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 (CCLog)
  - 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
# Classes in C++

## Declaration
- Like a Java interface
  - Fields, method prototypes
  - Put in the header file

```c++
class AClass {
private: // All privates in group
  int field;
  void helper();

public: // All publics in group
  AClass(int field); // constructor
  ~AClass(); // destructor
}; // SEMICOLON!
```

## Implementation
- Body of all of the methods
  - Preface method w/ class
  - Put in the cpp file

```c++
void AClass::helper() {
  field = field+1;
}
AClass::AClass(int field) {
  this->field = field;
}
AClass::~AClass() {
  // Topic of later lecture
}
```
## Stack-Based vs. Heap Based

<table>
<thead>
<tr>
<th>Stack-Based</th>
<th>Heap-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Object assigned to local var</td>
<td>- Object assigned to pointer</td>
</tr>
<tr>
<td>- Variable is NOT a pointer</td>
<td>- Object variable is a pointer</td>
</tr>
<tr>
<td>- Deleted when variable deleted</td>
<td>- Must be manually deleted</td>
</tr>
<tr>
<td>- Methods/fields with period (.)</td>
<td>- Methods/fields with arrow (-&gt;)</td>
</tr>
</tbody>
</table>

- **Example:**
  ```cpp
  void foo() {
    Point p(1,2,3); // constructor
    ...
    // Deleted automatically
  }
  ```

- **Example:**
  ```cpp
  void foo() {
    Point* p = new Point(1,2,3);
    ...
    delete p;
  }
  ```
## Stack-Based vs. Heap Based

### Stack-Based
- **Object assigned to local var**
  - Variable is NOT a pointer
  - Deleted when variable deleted
  - Methods/fields with period (.)
- **Example:**
  ```c++
  void foo() {
      Point p(1,2,3); // constructor
      ...
      // Deleted automatically
  }
  ```

### Heap-Based
- **Object assigned to pointer**
- **Object variable is a pointer**
- **Must be manually deleted**
- **Methods/fields with arrow (->)**
- **Example:**
  ```c++
  void foo() {
      Point* p = new Point(1,2,3);
      ...
      delete p;
  }
  ```
Do not need heap to return
- Can move to calling stack
- But this must \textit{copy} object

Need a special constructor
- Called \textit{copy constructor}
- Takes \textit{reference} to object
- C++ calls automatically

Is this a good thing?
- Performance cost to copy
- Cheaper than heap if small

```cpp
Point foo_point(float x) {
    Point p(x, x);
    return p; // Not an error
}
```

```
Point::Point(const Point& p) {
    x = p.x;
    y = p.y;
    z = p.z;
}
```

Calls
Returning a Stack-Based Object

- Do not need heap to return
  - Can move to calling stack
  - But this must \textit{copy} object

- Need a special constructor
  - Called \textit{copy constructor}
  - Takes reference to object
  - \texttt{C++} calls automatically

- Is this a good thing?
  - Performance cost to copy
  - Cheaper than heap if small

\begin{verbatim}
Point foo_point(float x) {
  Point p(x, x);
  return p; // Not an error
}
Point::Point(const Point& p) {
  x = p.x;
  y = p.y;
  z = p.z;
}
\end{verbatim}
The Many Meanings of `const`

- In C++, it is common to see something like:
  ```cpp
  const Point& foo(const Point& p) const;
  ```
The Many Meanings of `const`

- In C++, it is common to see something like:
  
  ```
  const Point& foo(const Point& p) const;
  ```

  Caller cannot modify the object returned.
The Many Meanings of `const`

- In C++, it is common to see something like:

  ```cpp
  const Point& foo(const Point& p) const;
  ```

  **Caller cannot modify the object returned**
  
  **Method cannot modify the object passed**
In C++, it is common to see something like:

```cpp
const Point& foo(const Point& p) const;
```

- Caller cannot modify the object returned.
- Method cannot modify the object passed.
- Method cannot modify any object fields.
The Many Meanings of `const`

- In C++, it is common to see something like:
  ```c++
  const Point& foo(const Point& p) const;
  ```
  - Caller cannot modify the object returned
  - Method cannot modify the object passed
  - Method cannot modify any object fields

- Believe it or not, these are not the only `consts`
  - But these are generally the only ones to use
  - See online tutorials for more
Inlining Method Definitions

- Can implement in `.h` file
  - Define methods Java-style
  - Will **inline** the methods
- Less important these days
  - Good compilers inline
  - Function overhead is low
- Only two good applications
  - Getters and setters
  - Overloaded operators
  - Use this sparingly

```cpp
class Point {
  private:
    float x;
    float y;
  
  public:
    Point(float x, float y, float z);
    float getX() const { return x; }
    void setX(float x) {
        this->x = x;
    }
    ...
};
```
Operator Overloading

- Change operator meaning
  - Great for math objects: +, *
  - But can do any symbol: ->

- Method w/ “operator” prefix
  - Object is always on the left
  - Other primitive or const &

- Right op w/ friend function
  - Function, not a method
  - Object explicit 2nd argument
  - Has full access to privates

```cpp
Point& operator*=(float rhs) {
    x *= rhs; y *= rhs; z *= rhs;
    return *this;
}

Point operator*(const float &rhs) const {
    return (Point(*this)*=rhs);
}

friend Point operator* (float lhs,
                         const Point& p) {
    return p*lhs;
}
```
Subclasses

- Subclassing similar to Java
  - Inherits methods, fields
  - Protected limits to subclass

- Minor important issues
  - Header must import subclass
  - `super()` syntax very different
  - See tutorials for more details

- Weird C++ things to avoid
  - No **multiple inheritance**!
  - No **private subclasses**

```cpp
class A {
  public:
    float x;

  A(float x) { this->x = x; }

  ...
};

class B : public A {
  public:
    float y;

  B(float x, float y) : A(x) {
    this->y = y;
  }

  ...
};
```
Subclasses

- Subclassing similar to Java
  - Inherits methods, fields
  - Protected limits to subclass
- Minor important issues
  - Header must import subclass
  - `super()` syntax very different
  - See tutorials for more details
- Weird C++ things to avoid
  - No **multiple inheritance**!
  - No **private subclasses**

```cpp
class A {
public:
  float x;
  A(float x) {
    this->x = x;
  }
  ...
};

class B { public A {
public:
  float y;
  B(float x, float y) : A(x) {
    this->y = y;
  }
  ...
};
```
C++ and Polymorphism

• Polymorphism was a major topic in CS 2110
  • Variable is reference to interface or base class
  • Object itself is instance of a specific subclass
  • Calls to methods are those implemented in subclass

• Example:
  • List<int> list = new LinkedList<int>();
  • list.add(10); // Uses LinkedList implementation

• This is a major reason for using Java in CS 2110
  • C++ does not quite work this way
C++ and Polymorphism

- Cannot change stack object
  - Variable assignment copies
  - Will lose all info in subclass

- Only relevant for pointers
  - C++ uses static pointer type
  - Goes to method for type

- What the hell?
  - No methods in object data
  - Reduces memory lookup
  - But was it worth it?

```cpp
class A {
public:
    int foo() {return 42;}
};

class B : public A {
public:
    int foo() {return 9000; }
};

B* bee = new B();
x = b->foo();  // x is 9000
A* aay = (A*)bee;
y = a->foo();  // y is 42!!!
```
Fixing C++ Polymorphism

- Purpose of **virtual** keyword
  - Add to method in base class
  - Says “will be overridden”

- Use optional in subclass
  - Needed if have subsubclass
  - Or if not further overridden

- Hard core C++ users hate
  - Causes a performance hit
  - Both look-up and storage
  - But not a big deal for you

```cpp
class A {
public:
    virtual int foo() {return 42;}
};

class B : public A {
public:
    int foo() {return 9000; }
};

B* bee = new B();
x = b->foo();  // x is 9000

A* aay = (A*)bee;
y = a->foo();  // y is 9000
```
Is There Anything Else?

- C++ has a lot of features not covered in lecture
  - **Templates** are the biggest topic skipped
  - **Preprocessor directives** and macros (like `#ifndef`)
  - **Namespaces** (e.g. packages)

- But you can survive this class without them
  - Need to use templates, but not write them
  - Using templates is close to a Java generic

- Or just look at some tutorials online
Summary

• C++ has a lot of similarities to Java
  • Java borrowed much of its syntax, but “cleaned it up”

• Memory in C++ is a lot trickier
  • Anything allocated with new must be deleted
  • C++ provides many alternatives to avoid use of new

• Classes in C++ have some important differences
  • Can be copied between stacks if written correctly
  • C++ supports operator overloading for math types
  • C++ needs special keywords to support polymorphism