CS 415
Operating Systems Practicum

Oliver Kennedy

Who is this yutz?

- *Oliver Kennedy, grad student by day*
  - *4124 Upson Hall*
  - *Office Hours: M/W 1:30-3:00*

415: The Good

- *See what goes on "under the hood"*
- *Get to experience the whole thing*
  - *Design the core of an OS and build an app on top of it*
- *Play around with fun toys*

415: The Bad

- *You need to know C++/Assembly*
- *You need a good architecture background*
- *You'll be spending a lot of time in the Systems lab*

415: The Ugly

- *Grading*
  - *5 Projects, 20% each*
    - *Meetings with me*
  - *Groups of 2: The group gets the grade*
- *Cheating*
  - *Don't*

Should you choose to accept it...

- *Threads 1: Thread Basics*
- *Threads 2: Preemption*
- *Networking 1: UDP*
- *Networking 2: TCP*
- *Final Project: Simulated Ad-hoc*

Where to go?

- *C/C++*
  - *Kernighan & Ritchie: The C Programming Language*
  - *Oualline: Practical C Programming*
  - *Visual Studio's Help Section*
- *x86 ASM*
  - *http://www.scs.stanford.edu/nyu/04fa/lab/reference.html*
- *Review Sessions*
- *Me*

Final Words

- *CMS*
  - *http://cms.csuglab.cornell.edu/*
- *Review Sessions*
  - *C++ for Java programmers*
    - *Now (PH 203)*
  - *Computer Architecture*
    - *Thursday: 3:15 (PH 203?)*

C++ for Java Programmers

Oliver Kennedy

based on lecture slides by Tom Roeder

Why use C++?

- *A pretty face on assembly*
  - *Fast/Compiles to native machine code*
  - *Grants access to hardware*
- *Simple*
  - *Most commonly used languages are based on C*
- *OO features available*

Why don’t more people use C++?

- *Explicit memory management*
  - *Leaks, Accessing freed memory...*
- *Language features dependent on platform*
  - *Size of primitives, Library availability*
- *Limited typechecking*
- *Header Files*

C++ Files

Header File

```
int myFunc(int myVar)
class myClass extends mySuperclass {
  public:
    int myClassVar;
    myClass(int myVar);
    int myMethod(int myVar);
  private:
    int myPrivateMethod(int myVar);
}
```

Source File

```
int globalVar;

int myFunc(int myVar){ ... }
int myClass:myClassVar(int myVar){
  myClassVar = myVar;
  myClassVar++;
  this.myClassVar++;
}
int myClass:myMethod(int myVar){ ... }
int myClass:myPrivateMethod...
```

Primitives

- *Integer Types: int, short, long*
  - *short(2) <= int(4/8) <= long(8/16)*
- *Floating Point Types: float, double*
  - *float(16) <= double(32)*
- *Character Type: char*
  - *Windows uses WCHAR*

Control Flow

- *if(...) { ... } else { ... }*
- *while(...) { ... }*
- *for(... ; ... ; ...) { ... }*
- *Functions*
  - *int myFunc(int myVar) { return myVar; }*
  - *myVar = myFunc(4);*
- *Programs start at int main()*

Examples: main()/arg

```
static void main(String args[]){
      TrackPoint myTrack = new LocatePoint();
      myTrack.updatePoint();
      System.println(myTrack.getColor() + args[0]);
}
```

```
int main(int argc, char **argv){
      TrackPoint myTrack = new TrackPoint();
      myTrack.updatePoint();
      cout << myTrack.getColor() << argv[0] << endl;
      delete myTrack;
}
```

The Enum/Typedef

- *enum maps text in the code to an integer*
  - *enum foo { bar, baz, bat };*
  - *enum foo myVar = bar;*
  - *enum color { blue = 7, green = 137};*
- *typedef creates an abbreviation for a type*
  - *typedef int foo;*
  - *foo myVar = 3;*

The Struct

- *Structures are like mini-classes*
  - *No methods, no superclass, just variables*
- *struct foo { int bar; int baz; };*
  - *struct foo myVar;*
  - *myVar.bar = 2*
- *typedef struct foo {int bar;} baz;*
  - *baz myVar;*

Classes

- *No interfaces*
  - *... but we have multiple inheritance*
- *class myClass extends mySuperClass { ... }*
- *Classes are separated into 3 sections denoted by public: private: protected:*
- *'virtual' denotes a function meant to be overridden.*

Example: Public/Private

Java

```
public class TrackPo
    protected MyPoi

    public TrackPoi
        lastPoint =
    }
```

C++

```
class TrackPoint
    public:
        TrackPoi
        ~TrackPo
        point ge
        virtual
    protected:
        virtual
```

Example: Virtual

Java

```java
//this method should be overridden by subclasses
public void updatePoint(){
    lastPoint.moveY(1);
}
```

C++

```cpp
//this method should be overridden by subclasses
virtual void TrackPoint::updatePoint(){
    lastPoint->y += 1;
}
```

Classes (cont)

- *Classes are broken down into 2 parts*
- *Definition (Header File)*
  - *Describes the class*
- *Implementation (Source)*
  - *The methods*
  - *method names preceded by `ClassName::`*

Example: Classes

Definition

```
class TrackPoint extends GetColor {
    public:
        TrackPoint();
        ~TrackPoint();
        point getPoint();
        virtual color getColor();
    protected:
        void updatePoint();
        point *lastPoint;
}
```

Implementation

```
TrackPoint::TrackPoint() {
    lastPoint = malloc(sizeof(point));
    (* lastPoint).x = 0;
    lastPoint->y = 0;
}

TrackPoint::~TrackPoint() {
    free(lastPoint);
}

point TrackPoint::getPoint() {
    return *lastPoint;
}
```

Arrays

- *Arrays work like they do in java*
  - *... if you know how big the array will be in advance*
  - *and no .length variable*
- *Static Array Sizes: int myArray[20]*
- *Dynamic Array sizes: see pointers*

Strings

- *Just an array of characters*
- *char *myString, or char myString[20];*
- *Terminated with '\0'*

Pointers

- *& gets a variable's address*
- ** dereferences or declares a pointer*
  - *int *myPointer = &myIntVar;*
  - **myPointer++;*
- *myPointer = (int *)malloc(sizeof(int))*
- *free(myPointer)*

Pointers (continued)

- *You must call free() on each pointer you malloc, but only after you're done!*
- *You can allocate arrays with malloc(*
  - *malloc(sizeof(int) * n)*
  - *These work like normal arrays.*
- *Classes use new and delete instead of malloc and free()*

Example: Memory

C++

```
TrackPoint::TrackPoint() {
    lastPoint = malloc(sizeof(point));
    (* lastPoint).x = 0;
    lastPoint->y = 0;
}

TrackPoint::~TrackPoint() {
    free(lastPoint);
}
```

```
TrackPoint myTrack = new TrackPoint();
myTrack.updatePoint();
cout << myTrack.getColor() << argv[0] << endl;
delete myTrack;
```

Java

```
public TrackPoint(){
    lastPoint = new MyPoint(0, 0);
}
```

Example: Pointer Usage

```
TrackPoint::TrackPoint() {
    lastPoint = malloc(sizeof(point));
    (* lastPoint).x = 0;
    lastPoint->y = 0;
}

TrackPoint::~TrackPoint() {
    free(lastPoint);
}
```

Special Pointers

- *Anonymous pointers*
  - *void **
  - *Analogous to Java's Object*
- *Function pointers*
  - *int call_me(float a) { return (int)a; }*
  - *int (*fp)(float) = &call_me*
  - *(*fp)(3.0)*

Parameter Passing

- *Consider: b = 3; foo(b); cout << b;*
- *void foo(int a) { a += 2; } // outputs 3*
- *void foo(int *a) { (*a) += 2; } //outputs 5*
- *In Java Objects/Arrays behave like case 2*
- *In C Pointers/Arrays behave like case 2*

Careful...

- *No garbage collection, free what you take*
- *Arrays aren't bounds checked (and no .length)*
- *Variables may not be cleared after allocation. (Set pointers to NULL)*
- *Check for NULL pointers before each use!*
- *Packages like Purify exist to help*

The Preprocessor

- *#define foo 42*
- *#define foo(a, b) a+b*
- *#include*
- *#ifdef / #else / #end*
  - *#ifdef foo means that if foo is not defined, everything between that and #else will be treated as if it were commented out*

Example: Precompiler

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

```
//comment the following line out to use #defines for colors
#define USE_ENUM

#ifdef USE_ENUM
enum e_color { red = 0xf00, green = 0x0f0, blue = 0x00f };
typedef enum e_color color;
#else
#define red 0xf00
#define green 0x0f0
#define blue 0x00f
typedef int color;
#end
```