Feedback

- Comments on project 2?
- Any feedback/horror stories?
- Anyone get a buyout offer?
Announcements

- Homework Project 3 posted online
  - Group work

- MP3 online organizer/library

- Perl / C++ / C
  - Ability to upload, sort, search, preview (10 sec)
  - Ability to play random, ordered, history
  - User accounts with shared songs etc
    - Can upload songs for personal use
    - Can upload songs for anyone to use
Outline

- C++
  - class abstractions
  - class inheritance
  - creating classes
  - long example

- Reading:
  - chapters 16, 17, 18 (C++, classes, overloading)
Random stuff

- Anyone know how to grab errors?

- Can overload stderr to grab it using shell instructions

  ```bash
  "ls -la 2>&1"
  ```
Stuff II

- c: manipulates variables by reference through the use of pointers

- c++ introduces another way of manipulating references variables
  - reference parameters
  - &
  - easier to use than pointers
example

```c
void addTo(int a, int b) {
    a = a+b;
}

void addToRef(int &a, int b) {
    a = a+b;
}
```
Stuff III

- in theory could return referenced variables
- but if not declared static, will return dangling pointers (since will be freed from stack)
- like pointers, need to assign a ref variable to make sense or else etc

```c
int sum = 129;
int &refint = sum;
refint = 2;
```
Variable scope:
- We mentioned that variables can live outside of functions
- CPP allows you to specify scope through unary scope operator (::)
- So can differentiate between local and global variables
int count = 10;

int main(){
    int count;
    count = 5;

    //who are we talking about and how
    //to refer to each
count is your local
::count is global

std::count is the same as ::count if we are in std namespace
Stuff V

- \( a == 5 \) ? \( a++ \) : \( a = 0; \)
OOP

- As mentioned before C++ was mainly introduced to organize C in an OO fashion

- when you use OOD you end up working with objects and states

- Lets say you need to manipulate fractions, how would you do this in basic c?
  - do you have a more elegant solution?

- Lets talk about it in C++
Functions

- **Accessor**
  - get some state information from the object

- **Mutator**
  - change information

- **Helper**
  - internal functions to accomplish tasks cleanly

- **Predicate**
  - help answer simple yes/no questions
CPP classes

- A class if a collection of functions and variables
- In CPP we have constructors and destructors
- In C++ how to define a constructor?
- destructor?
- when are they invoked?
Counting

- Say you want to create a counting object

- What do you need?
Simple version I

class Counter {
public:
    int x;
    void print() { cout << x << endl; }
};
Simple version I b

class Counter {
public:
    int x;
    void print();
};

void Counter::print() {
    cout << x << endl;
}
accessing variables

- Count mycounter;
- mycounter.x = 7;
- mycounter.print();

- Counter *countPTR;
- countPTR = new Count;
- counterPTR->print();
abstraction

- Anyone can program (well almost)
- Important to use planning when writing code
- When you define a class separate how to use the class and how to represent the information in the class
- i.e. what belongs in private/public
```cpp
#include<iostream>

using namespace std;

int globalcounter = 0;

class Counter{
    private:
        int x;
    public:
        Counter (){
            x = globalcounter++;
            cout<<"in the cnst "<<x<<endl;
        }
        void print() { cout << x <<endl; }
        ~Counter() { cout<<"in dest for "<<x<<endl; }
    
};

int main(int argc, int *argv){
    cout<<"this is a test of counter class"<<endl;
    Counter c1;
    Counter *c1;
    c2 = new Counter; //notice new
    return 0;
}
```
One issue

- being careful not return private references
Practice

- After class: code the counter class
  - How would you add an ID to differentiate between counter instances??
idea

- add a static member ID
  - Instead of global variable

- int Foo::ID = 0;
  - in global scope of class
Hands on Coding

- For lab will be coding a fraction class
- main will look something like:

```c
int main(void) {
  cout<<“start”<<endl;
  Fraction f1;
  cout<<“End”<<endl;
  return 0;
}
```

- We will be printing out when things are called, will get interesting
- add constructor/destructor
- add print to them and see what it outputs
- add a global fraction
- now add a pointer to a fraction
  - what happened to the destructor?
Question on coding

- what if we wanted to keep roman numerals as a counter?

- How robust is your code to this??

- What would the class functions look like?
Example II

class Counter {
private:
    char * x;
    char * convertInt(int number);
public:
    Counter() { ... }
    ~Counter { ... }
    void setCount(int newcnt) {
        x = convertInt(x);
    }
    void print() { cout << x << endl; }
}
Implicit assignment

- If you don’t define an assignment operator
  - Will try to figure out how do to it
  - By looking at each field member variable
  - Works with primitives
  - Pointers will get shallow copied

```cpp
Counter a;
Counter b;
a.setCount(19);
b = a;
b.print();
```
Copy constructor

- Counter t2 = t1;
- Looks like assignment
- Really a constructor call with object as argument
- Called *copy constructor*
- Combination of constructor and assignment
Defining it

- Just overload the constructor
  
  `Counter(Counter &source);`

- Be careful:
  - When you overload the copy constructor you throw out a default constructor
  - Which means you need to explicitly define a default constructor (no arg)
question

- so how do you do this with a roman numeral counter?

- RomanCounter A,B;
  //do some stuff to A
  //want to say
  // B = A
C++ allows you to do this by redefining the = sign

Let's talk about some background material in order to write this code
- **const** keyword
  - there are times when we want to ensure that our program will not change a specific variables value.
  - variables can be declared const
    - const int x = 3;
    - const Count c(13);
  - functions need to be declared const when dealing with const variable members
**const class members**

- **const class members are assigned at construction time using the : notation**

```cpp
class Worker {
public:
    Worker(int id, int job);
    int getID() const;
private:
    const int _ID;
    int _job;
};
```
constructor

Worker(int id, int job) : _ID(id) {
    _job = job;
}

Classes within classes

- class member variables can be other classes
  - Example can create a String class (lab)
  - Counter can have a String instance

- Important theory: member constructors are actually called before main class constructors
  - does this make sense?
Stuff VI

- this is a keyword

- represents a pointer to the class itself

- this->x

- or (*this).x
static

- static members have instance wide scope and livability

- great for shared variable

- have to be careful how used
assert

- special macro runs a test
- if true continues
- if false
  - dies without calling destructors
Order of running program

- In C we saw that the program always starts from main.

- As mentioned in class, this is different in C++. 
What can go wrong

- The good thing about cpp is that your program can now crash many times even before reaching main 😊
Ordering and where to look for problems

- Global variables
  - Assignments and constructors
  - What else ??
- Main
- Local variables
- End local variables
- End main
- Global destructors
friends

- A class can declare a function to be a friend
- allows access to private member of the class
- not scoped during definition, can use undefined classes
Operator overloading

- Most operators can be overloaded in cpp
- Treated as functions
- But its important to understand how they really work
Look up list
Operators which can not be overloaded

- .
- .*
- ::
- ?:
- sizeof
- $X = X + Y$
- Need to overload
  
  +  
  
  =  

- But this doesn’t overload $+= $
- Functions can be member or non-member
- Non-member as friends
- If its member, can use `this`
- `()`, `[]`, `->` or any assignments must be class members

- When overloading need to follow set function signature
Code from fig18_03 (c book)