Announcement

- Due to thanksgiving I will post lab for those who want to leave early
- I will be around as usual
- Labs have been graded, please take a look and meet the TAs about any issues, sooner rather than later
Announcements

- Homework Project 3
  - Group work

- MP3 online organizer/library
- Perl / 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
- “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

code

```c
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

- ---- ? ----- : -------

- condition ? then : else

- (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 is 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 *c2;
    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:

```cpp
int main(void) {
    cout<"start"
    Fraction f1;
    cout<"End"
    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

```cpp
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

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
background

- 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

```cpp
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 CPP

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 C++
- Treated as functions
- But it's important to understand how they really work

+  ~  -  !  =  *  /=  +=  <<

>>  &&  ++  []  ()  new  delete  new[]  ->  >>=

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 \textit{this} \\
\textit{(), [], ->} or any assignments must be class members

When overloading need to follow set function signature

Code from fig18_03 (c book)
Phone Numbers

- Say you want to code a phone number class

```cpp
class PhoneNumber {

private:
    char areaCode[4];  // 3-digit area code and null
    char exchange[4];  // 3-digit exchange and null
    char line[5];      // 4-digit line and null
};
```

Notice: how would you add a new phone number (or change data)
class PhoneNumber {
private:
    char areaCode[4]; // 3-digit area code and null
    char exchange[4]; // 3-digit exchange and null
    char line[5];    // 4-digit line and null

friend ostream &operator<<( ostream&, const PhoneNumber & );
friend istream &operator>>( istream&, PhoneNumber & );

// Overloaded stream-insertion operator (cannot be
// a member function if we would like to invoke it with
// cout << somePhoneNumber;).
ostream &operator<<( ostream &output, const PhoneNumber &num )
{
    output << "(" << num.areaCode << ") "
          << num.exchange << "-" << num.line;
    return output; // enables cout << a << b << c;
}
}
```cpp
istream &operator>>( istream &input, PhoneNumber &num )
{
    input.ignore(); // skip ( 
    input >> setw( 4 ) >> num.areaCode; // input area code 
    input.ignore( 2 ); // skip ) and space 
    input >> setw( 4 ) >> num.exchange; // input exchange 
    input.ignore(); // skip dash (-) 
    input >> setw( 5 ) >> num.line; // input line 
    return input; // enables cin >> a >> b >> c; 
}
```

```cpp
int main()
{
    PhoneNumber phone; // create object phone 
    cout << "Enter phone number in the form (123) 456-7890:\n";
    cin >> phone;
    cout << "The phone number entered was: " << phone << endl;
    return 0;
}
```
unary

- Y += Z
- Y.operator+=( Z )

- ++D
- member
  - D.operator++()
- Non member
  - operator++(D)

development question

- is there a different if you code in visual cpp vs gcc in cygwin if you are working on windows ??

- when would it matter ?
virtual functions

- in C++ virtual functions allow you to define a specific function in the base class, which is undefined, and each of the subclasses need to override (implement a definition)

- virtual char * md5sum();

- so if we use a base class pointer at a derived class object, calling md5sum will call the correct one

- compile time resolution
  - static binding
Instantiation

- Remember as soon as you declare a class you are instantiating it
- String s;

- sometimes would like to have classes which can not be instantiated

- abstract classes
  - a class is made abstract by having a pure virtual function.

Abstract

- virtual char * md5sum() =0;

- any ideas on what error will be thrown if you instantiate it?
non virtual base functions

- if you have a parent class A.foo()
- derived class B defines B.foo()

- A *a_ptr = B_object

- a_ptr.foo()
  - which foo will be triggered?
  - why ?

abstract classes II

- remember that making a pointer doesn’t instantiate anything
- can create pointers of type abstract classes
- used to enable polymorphic behavior

- Example: Operating system device
  - read/write behavior
destructors

- when creating and manipulating objects in a polymorphic context, destructors will only be called on base class

solution

- define a virtual base class destructor
- will correct destructor will be called
Example 20_1