CS3157: Advanced Programming

Lecture #7
June 18
Shlomo Hershkop
shlomo@cs.columbia.edu
Overview

- More C++
  - Overloading
  - Classes and hierarchies
  - Software engineering
Announcements

- Remember to submit the code from last Monday tonight as the lab
CPP classes

- A class if a collection of functions and variables
- In CPP we also have special functions called constructors and destructors
- Need to understand how to design and then code classes….think of objects first..
Counting

- Say you want to create a counting object, example a web counter of visitors..

- 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
#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;
}
idea

- Since local variable can be changed
- 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"<<endl;
    Fraction f1;
    cout<<"End"<<endl;
    return 0;
}
```

- As we did last week, printout out in the constructor and dest can show you how things are being manipulated in the memory space..
Remember from last time..

- 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

Counter a;
Counter b;
a.setCount(19);
b = a;
b.print();
Reminder Constructors

- In C++ a class has a default constructor only if you don’t define any

```cpp
class Example1 { 
    public: 
        int a, b, c; 
        void multiply (int n, int m) 
            { a=n; b=m; c=a*b; } ; 
    } ;
```
Example

- So can say
  Example1 ex;
- Which will create a constructor for you

- As soon as you define any constructor (say taking one arg) the above line stops working

- Blame the compiler
surprise

- By default
- Compiler will create
  - copy constructor
  - copy assignment operator
  - default destructor.
Differences

- Should understand the difference between
  - X
  - *X
  - &X
  - X.Y
  - X->Y
  - (*X).Y
friendship

- Functions (which you’ve seen) and classes which can access private class members

- So you’ve friend functions for overloading’

- Here is an example of friend class
// friend class
#include <iostream>
using namespace std;

class CSquare;

class CRectangle {
  int width, height;
  public:
    int area ()
    {return (width * height);} //area is public
    void convert (CSquare a);
};

class CSquare {
  private:
    int side;
  public:
    void set_side (int a)
    {side=a;}
    friend class CRectangle;
};

void CRectangle::convert (CSquare a) {
  width = a.side;
  height = a.side;
}
int main () {
    CSquare sqr;
    CRectangle rect;
    sqr.set_side(4);
    rect.convert(sqr);
    cout << rect.area();
    return 0;
}
C++ hierarchy

- **composition:**
  - creating objects with other objects as members

- **derivation:**
  - defining classes by expanding other classes
  - like “extends” in java

```cpp
class SortedIntArray : public IntArray {  
public:
    void sort();
private:
    int *sortBuf;
}; // end of class SortedIntArray
```

- “base class” (IntArray) and “derived class” (SortIntArray)
- derived class can only access public or protected members of base class
public derivation means that users of the derived class can access the public portions of the base class

private derivation means that all of the base class is inaccessible to anything outside the derived class

private is the default
Functions can be member or non-member, your choice!
  - Non-member as friends if need private data
- If its member, can use the *this* pointer

Exception: operators (), [], -> or any assignments must be class members

When overloading need to follow set function signature
cout

- cout << yourclass

- left operand is ostream &
- so non member functions (belongs to ostream)
- friend if you would like

- lets code something
String class

- let's define a simple string class

- put output in its const and dest so we can follow

- constructor should take `const char *`
- would like to have following defined:
  ```
  int length();
  int hash();
  ```
- any ideas on how to do it?
overload printing

friend ostream & operator <<(ostream &, const String &);

ostream &operator<<(ostream &output, String &str) {
    output << "'", str, "'");
    return output;
}
note

- when you call:
  `cout << s1 << s2;`
- it is first:
  `operator<<(cout, s1)`
- and then
  `operator<<(cout, s2)`
Next

- want to overload the unary operator !

- test if a string is blank

- int operator!() const;
- or
- friend int operator(const String &);

- !s1
- s.operator!() or operator![](s)
same idea

- const String operator+=(const String &)

- vs

- friend const String &operator+=(String &, const String &)

- what will s1 += s2 produce?
Array Class

- Arrays are hard to work with directly since there is no support for out of bounds

- Let's look at 18.4 from the book
extending

- any ideas on how to extend the base class ??
so how can we tell the difference between 
++s1 and s1++
signatures

- s1++

- s1.operator++(0)

- operator++(s1,0)
+++s1;

s1.operator++();

operator++(s1)
reuse

- one of the powers to OOP is the idea of reuseability

- if I spend 5 billion hours working on my code, I probably want to get some use out of it outside of the specific task
  - design issues
  - extension issues
inheritance

- idea: allow a new class to inherit data members and functions from a base class
  - can add members and functions
  - represents a more specific idea
- vehicle -> minivan
- you can access protected members of parent

- can not access private members of parent
  - can still use public accessors and modifiers
code

class IntArray: public Array {

  ■ simplest type of inheritance
  ■ private members not inherited
  ■ public/protected inherited accordingly
code

- create a point class
  - setPoint
  - <<

- derive Square
  - getArea()
overriding

- we can redefine a base class function in the derived class and have C++ call the correct one
Question

- can
- Point *pp1;
- Square *sp1;

- given
- Point p = Point(3,4);
- Square s = Square(..

- can we say:
- pp1 = s  ??????
- sp1 = p  ??????
private inheritance

- we have used public inheritance

- private inheritance makes everyone from the base class come in as private members of the derived class
base class constructors

- need to launch base class constructor in derived class if you don’t want the default to be called

- destructors are reversed

- lets see this in action
is a vs has a

- one important design decision is to know when to derive and when to use member variable
issue

- one issue with overriding, is that if the derived class doesn’t provide a function, we will use the base class definition

- this doesn’t always make sense

- Example I want a function MPG for any type of vehicle, but doesn’t make sense of base class
virtual functions

- solution:

- declare the function to be virtual

- virtual double MPG();

- allow you to use a base class pointer to call at runtime the correct function (polymorphism)
abstract class

- sometimes it's even useful to have a base class which can't be instantiated.
- if any virtual function is declared pure virtual:
- virtual int MPG() = 0;
note

- constructors can not be virtual

- need virtual destructors to make everything work if you are going to have destructors in any of your classes (do it anyway)
lets look at 20.1 code
Linkage directions

- If you want to call a function in another programming language, the compiler must be told that different rules apply

- Linkage directive
  - Single statement
  - Compound form

- Declared outside of functions
Single form

- extern "C" void something(int);

- Keyword
- String
- Function
- Compiler will type check any function calls
Compound form

- `extern “C” {
  int printf(const char * ...);
  int scanf (const char * ...);
}

- `extern “C” {
  #include <cmath>
  
  }


Other languages

- Depends on the compiler
- For example many support FORTRAN
Dynamic allocation

- Local variables have local life and scope
- If you want to dynamically create and manage memory, use the new and delete
- Using pointers

- Have to be careful from dangling pointers…
- Ideas?
Reality check

- int *p = new int (1024);
- int *q = new int [1024];
- int (*r)[1024] = new int [4][1024];
Abstraction and member functions

- How are objects internally manipulated by CPP....let's take a look at a complex example
Rect

class Rect {

    // ...
    private:
    int top, left;
    int width, height;

    ..
};
Color

class Color{
    // ..
    private:
    int data;
};
class TextBox: public Rect{
    //...
    private:
    Color txtColor;
    int frameThick;
    char *text;
};
main

main() {
    TextBox source, dest;

    //...

    dest = source;

    How to get this to work?
Overloading operator =

class TextBox : public Rect{
    public:
    void operator=(TextBox &source);
    ..
Equivalent

main() {
    TextBox source, dest;

    //...

    dest.operator=(source);
}
Inside

void TextBox::operator=(TextBox &source) {

    if(this == &source)
        return;

    Rect::operator=(source);

    txtColor = source.txtColor;

    frameThick = source.frameThick;

    delete []text;
    if(source.text != 0) {
        text = new char[strlen(source.text+1)];
        strcpy(text,source.text);
    }
    else
        text = 0;
}
Implicit assignment

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

- TextBox 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
  TextBox(TextBox &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)
code

TextBox::TextBox(TextBox &source) {

    Rect::operator=(source);

    frameThick = source.frameThick;
textColor = source.textColor;

    etc
Chaining

- If you want to be able to say
  
  `Textbox a, b, c;
  //...
  a = b = c ;`

- how would the operator overloaded be different ??
Exception

- Like in java, CPP allows you to throw and catch exceptions

- Compiler time exceptions
- Run time exceptions
Template programming

- Allows you to specify a type to pass in to your class, so can create a collection class to handle many different types, without having the problem if limited casting in the code

- Allows you to move errors from run time to compiler time
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
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
- the correct destructor will be called
Virtual functions

- Allows you to declare a function in the base class without a definition
- Each of the derived class provide a definition unique to their implementation
- At runtime will allow all derived class object instances to be manipulated uniformly
Next

- Software engineering
  - Will cover most in class, you are responsible for understanding high level overview
What is Software Engineering?

Stephen Schach: “Software engineering is a discipline whose aim is the production of fault-free software, delivered on time and within budget, that satisfies the user’s needs.”

includes:
- requirements analysis
- human factors
- functional specification
- software architecture
- design methods
- programming for reliability
- programming for maintainability
- team programming methods
- testing methods
- configuration management
People

- you can’t do everything yourself
- e.g., your assignment: “write an operating system”
- where do you start?
- what do you need to write?
- do you know how to write a device driver?
- do you know what a device driver is?
- should you integrate a browser into your operating system?
- how do you know if it’s working?
Why

- in school, you learn the mechanics of programming
- you are given the specifications
- you know that it is possible to write the specified program in the time allotted
- but not so in the real world...
  - what if the specifications are not possible?
  - what if the time frame is not realistic?
  - what if you had to write a program that would last for 10 years?
- in the real world:
  - software is usually late, over budget and broken
  - software usually lasts longer than employees or hardware
- the real world is cruel and software is fundamentally brittle
Who

- the average manager has no idea how software needs to be implemented
- the average customer says: “build me a system to do X”
- the average layperson thinks software can do anything (or nothing)
- most software ends up being used in very different ways than how it was designed to be used
Time

- you never have enough time
- software is often under budgeted
- the marketing department always wants it tomorrow
- even though they don’t know how long it will take to write it and test it
- “Why can’t you add feature X? It seems so simple...”
- “I thought it would take a week...”
- “We’ve got to get it out next week. Hire 5 more programmers...”
Complexity

- software is complex!
- or it becomes that way
  - feature bloat
  - patching

- e.g., the evolution of Windows NT
  - NT 3.1 had 6,000,000 lines of code
  - NT 3.5 had 9,000,000
  - NT 4.0 had 16,000,000
  - Windows 2000 has 30-60 million
  - Windows XP has 40-45 million...
  - Vista 50-55 million
Necessity

- you will need these skills!

- risks of faulty software include
  - loss of money
  - loss of job
  - loss of equipment
  - loss of life
Therac-25


- therac-25 was a linear accelerator released in 1982 for cancer treatment by releasing limited doses of radiation

- it was software-controlled as opposed to hardware-controlled (previous versions of the equipment were hardware-controlled)

- it was controlled by a PDP-11; software controlled safety

- in case of error, software was designed to prevent harmful effects
BUT
in case of software error, cryptic codes were displayed to the operator, such as:
“MALFUNCTION xx”
Where $1 < xx < 64$

operators became insensitive to these cryptic codes
they thought it was impossible to overdose a patient
however, from 1985-1987, six patients received massive overdoses of radiation and several died
**main cause:**

- a race condition often happened when operators entered data quickly, then hit the up-arrow key to correct the data and the values were not reset properly.

- the manufacturing company never tested quick data entry— their testers weren’t that fast since they didn’t do data entry on a daily basis.

- apparently the problem had existed on earlier models, but a hardware interlock mechanism prevented the software race condition from occurring.

- in this version, they took out the hardware interlock mechanism because they trusted the software.
Example 2: Ariane 501

- next-generation launch vehicle, after Ariane 4
- prestigious project for ESA
- maiden flight: June 4, 1996
- inertial reference system (IRS), written in ADA
  - computed position, velocity, acceleration
  - dual redundancy
  - calibrated on launch pad
  - relibration routine runs after launch (active but not used)
- one step in recalibration converted floating point value of horizontal velocity to integer
- ADA automatically throws out of bounds exception if data conversion is out of bounds
- if exception isn’t handled... IRS returns diagnostic data instead of position, velocity, acceleration
- perfect launch

- ariane 501 flies much faster than ariane 4

- horizontal velocity component goes out of bounds

- IRS in both main and redundant systems go into diagnostic mode

- control system receives diagnostic data but interprets it as weird position data

- attempts to correct it...

- ka-boom!

- failure at altitude of 2.5 miles

- 25 tons of hydrogen, 130 tons of liquid oxygen, 500 tons of solid propellant
expensive failure:
- ten years
- $7 billion

horizontal velocity conversion was deliberately left unchecked

who is to blame?

“mistakes were made”

software had never been tested with actual flight parameters

problem was easily reproduced in simulation, after the fact
Mythical man-month

- Fred Brooks (1975)

- book written after his experiences in the OS/360 design

- major themes:
  - Brooks’ Law: “Adding manpower to a late software project makes it later.”
  - the “black hole” of large project design: getting stuck and getting out
  - organizing large team projects and communication
  - documentation!!!
  - when to keep code; when to throw code away
  - dealing with limited machine resources

- most are supplemented with practical experience
No silver bullet

- paper written in 1986 (Brooks)

- “There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade of productivity, in reliability, in simplicity.”

- why? software is inherently complex

- lots of people disagreed, but there is no proof of a counter-argument

- Brooks’ point: there is no revolution, but there is evolution when it comes to software development
SE Mechanics

- well-established techniques and methodologies:
  - team structures
  - software lifecycle / waterfall model
  - cost and complexity planning / estimation
  - reusability, portability, interoperability, scalability
  - UML, design patterns
Team Structures

- why Brooks’ Law?
  - training time
  - increased communications

- while people/work grows by
  - how to divide software? this is not task sharing

- types of teams
  - democratic
  - “chief programmer”
  - synchronize-and-stabilize teams
  - eXtreme Programming teams
Lifecycles

- software is not a build-one-and-throw-away process
- that’s far too expensive
- so software has a lifecycle
- we need to implement a process so that software is maintained correctly

- examples:
  - build-and-fix
  - waterfall
Software lifestyle cycle

- 7 basic phases (Schach):
  - requirements (2%)
  - specification/analysis (5%)
  - design (6%)
  - implementation (module coding and testing) (12%)
  - integration (8%)
  - maintenance (67%)
  - retirement

- percentages in ()’s are average cost of each task during 1976-1981
- testing and documentation should occur throughout each phase
- note which is the most expensive!
Requirements

- what are we doing, and why?

- need to determine what the client needs, not what the client wants or thinks they need

- worse— requirements are a moving target!

- common ways of building requirements include:
  - prototyping
  - natural-language requirements document

- use interviews to get information (not easy!)

- example: your online store
Specifications

- the “contract”— frequently a legal document
- what the product will do, not how to do it
- should NOT be:
  - ambiguous, e.g., “optimal”
  - incomplete, e.g., omitting modules
  - contradictory
- detailed, to allow cost and duration estimation
- classical vs object-oriented (OO) specification
  - classical: flow chart, data-flow diagram
  - object-oriented: UML
- example: your online store
Design Phase

- the “how” of the project
- fills in the underlying aspects of the specification
- design decisions last a long time!
- even after the finished product
  - maintenance documentation
  - try to leave it open-ended
- architectural design: decompose project into modules
- detailed design: each module (data structures, algorithms)
- UML can also be useful for design
- example: your online store
Implementation

- implement the design in programming language(s)
- observe standardized programming mechanisms
- testing: code review, unit testing
- documentation: commented code, test cases
- integration considerations
  - combine modules and check the whole product
  - top-down vs bottom-up?
  - testing: product and acceptance testing; code review
  - documentation: commented code, test cases
  - done continually with implementation (can’t wait until the last minute!)
- example: your online store
Maintenance Phase

- defined by Schach as any change
- by far the most expensive phase
- poor (or lost) documentation often makes the situation even worse
- programmers hate it

- several types:
  - corrective (bugs)
  - perfective (additions to improve)
  - adaptive (system or other underlying changes)

- testing maintenance: regression testing (will it still work now that I’ve fixed it?)
- documentation: record all the changes made and why, as well as new test cases
- example: your on-line store—how might the system change once it’s been implemented?
Retirement phase

- the last phase, of course

- why retire?
  - changes too drastic (e.g., redesign)
  - too many dependencies ("house of cards")
  - no documentation
  - hardware obsolete

- true retirement rate: product no longer useful
Planning and Estimation

we still need to deal with the bottom line
- how much will it cost?
- can you stick to your estimate?
- how long will it take?
- can you stick to your estimate?

how do you measure the product (size, complexity)?
Reusability

- impediments:
  - lack of trust
  - logistics of reuse
  - loss of knowledge base
  - mismatch of features

- how to:
  - libraries
  - APIs
  - system calls
  - objects (OOP)
  - frameworks (a generic body into which you add your particular code)
Portability

- Java and C#
  - Java: uses a JVM
    - write once, run anywhere (sorta, kinda)
  - C#: also uses a JVM
    - emphasizes mobile data rather than code
- winner?
  - betting against Microsoft is historically a losing proposition...
interoperability

- e.g., CORBA
- define abstract services
- allow programs in any language to access services in any language in any location
- object-ish
Scalability

- something to keep in mind

- don’t worry about scaling beyond the abilities of the machine

- avoid unnecessary barriers

- from single connection to forking processes to threads...
homework

- Next phase of online project

- What if you don’t have enough time 😊

- Software engineering solution ??