COSC 220: Data Structures in C++

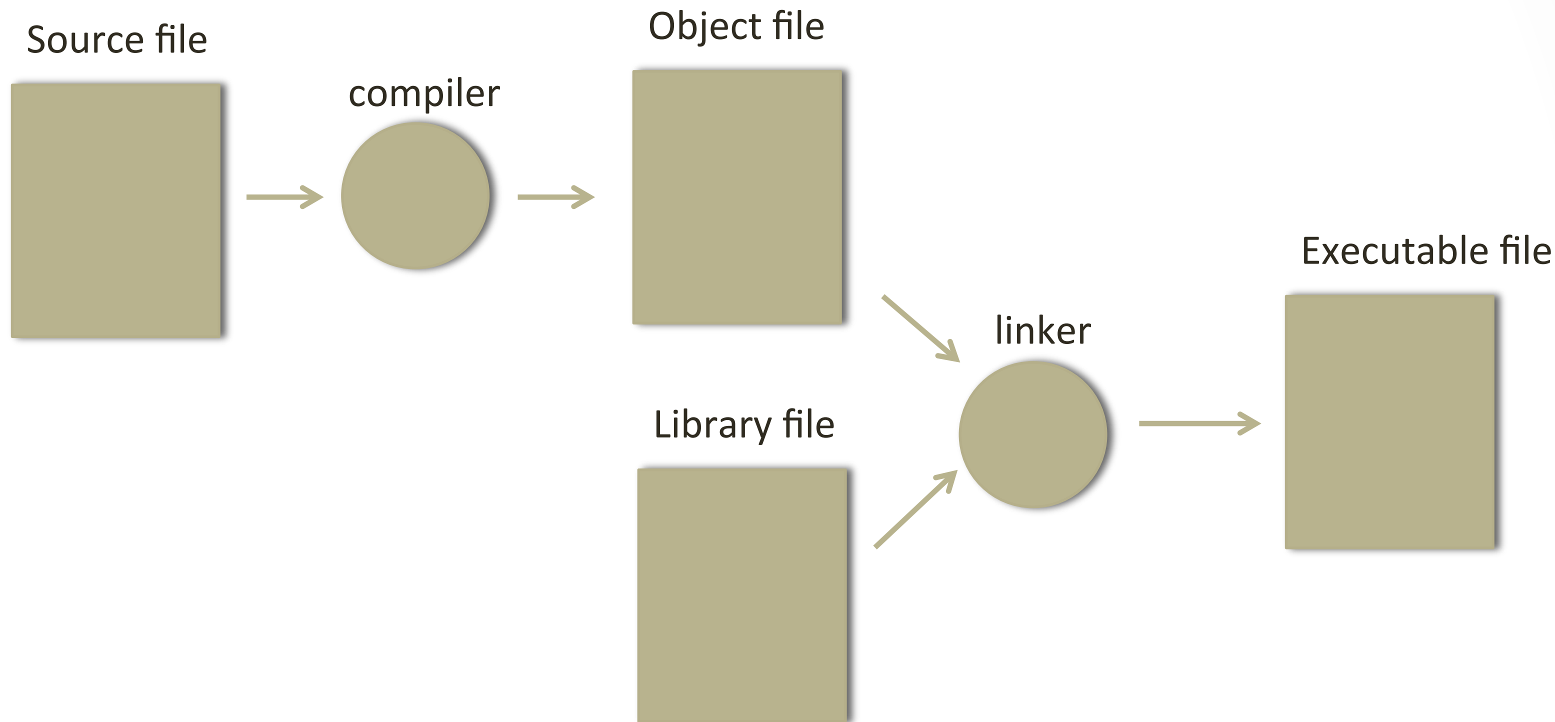
Data Structures: Queue Class

This week (week 9):
Designing Classes: Object Oriented Programming

HW#3 online later today

COSC 220: Data Structures in C++

Review of the Compilation Process:



1. **Source file**: text of your code
2. **Compiler** → translate source to **object file**
3. **Object file**: machine-language instructs
4. Object file combined with other object files via Linker → **Executable file**
5. Other object files: predefined obj files called **libraries**
6. **Libraries**: contain machine-lang instructs for various operations req'd by program
7. **Linking**: combining object files into an executable

COSC 220: Data Structures in C++

Designing Classes

We used several data structures that are implemented via Classes in C++
Similar to OO programming in Python
Will go over methods to implement classes
Later: deeper design, memory management, class inheritance

Simplest Class: Representing Points
point: x-y value rep'd in Euclidean Space
unified pair will be called a ***point*** so use name Point as type

COSC 220: Data Structures in C++

Designing Classes

Consider C structure type
we can define Point as one:

```
struct Point {
  int x;
  int y;
};
```

defines Point as a trad'l structure with 2 components called: *fields* or *members*
Point struct contains fields named x and y both of type int

declare a type:
Point p;
→ compiler reserves space in stack frame for internal fields int x and int y as well

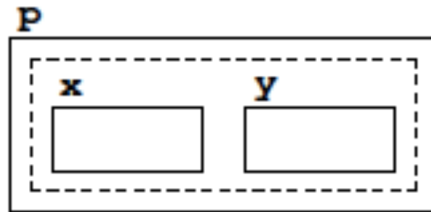
COSC 220: Data Structures in C++

Designing Classes

declare a type:

Point p;

→ compiler reserves space in stack frame for internal fields int x and int y as well



access individual fields using dot operator:

var.name

var: variable containing structured value

name: desired field

e.g. use p.x and p.y → individ coord. values of Point struct p

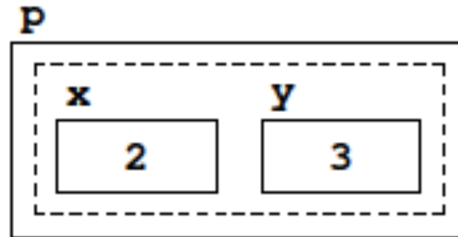
COSC 220: Data Structures in C++

Designing Classes

init’ze components of Point p to represent (2,3):

p.x = 2;

p.y = 3;



C++: given a point value, can assign that value to a variable, pass it as a parameter to a function, or return it as a result.

COSC 220: Data Structures in C++

Designing Classes

Defining a point as a C++ class:

```
class Point {
public:
  int x;
  int y;
};
```

fields of a class: instance variables
same syntax as fields in a structure

Key difference, C++ can define a public and private section
→ what parts of program have access to those fields

COSC 220: Data Structures in C++

Designing Classes

```
class Point {
public:
  int x;
  int y;
};
```

public: fields avail to anyone using the defining class
private: visible only to the defining class and not any of its clients

Modern C++: struct → entries public,
class → entries private

class: use methods to access private fields

COSC 220: Data Structures in C++

Designing Classes

```
class Point {
public:
  int x;
  int y;
};
```

point class: select x field by: pt.x
in general: public instance variables shunned upon.
Modern C++ → all instance variables private.
Use methods exported by class to access any info the class contains

separating implement'n details from client: simplicity, flexibility, and security. This are main points of encapsulation.

COSC 220: Data Structures in C++

Designing Classes

```
class Point {
private:
  int x;
  int y;
};
```

above now has instance variables x and y private. But what is the issue now?

COSC 220: Data Structures in C++

Designing Classes

```
class Point {
private:
  int x;
  int y;
};
```

above now has instance variables x and y private. But what is the issue now?
No way to access them!!!

Now: need way to create Point objects and to obtain x and y coord's from an existing Point.

Creating objects: **constructor**
always same name as the class

COSC 220: Data Structures in C++

Designing Classes

Creating objects: **constructor**
always same name as the class

Constructor that takes no args = default constructor

e.g. in Point class, default constructor → create a pair of coord values

Methods that retrieve instance variables: **getters (accessors)**
getters start with prefix: get
followed by name of field after capitalizing first letter
e.g. **getX** and **getY**

COSC 220: Data Structures in C++

Designing Classes

simple ver of point class:

why xc and yc params instead of x and y?

FIGURE 6-1 A simple version of the Point class

```
/*
 * Class: Point
 * ------------
 * This class represents an x-y coordinate point on a two-dimensional
 * integer grid.
 */

#include <string>
#include "strlib.h"
using namespace std;

class Point {
public:

   Point() {
      x = 0;
      y = 0;
   }

   Point(int xc, int yc) {
      x = xc;
      y = yc;
   }

   int getX() {
      return x;
   }

   int getY() {
      return y;
   }

   string toString() {
      return "(" + integerToString(x) + ","
                 + integerToString(y) + ")";
   }

private:
   int x;
   int y;
};
```

Constructors

Getter methods

Public section

Instance variables

Private section

COSC 220: Data Structures in C++

Designing Classes

simple ver of point class:

why xc and yc params instead of x and y?

x, y: instance var's
xc, yc: param's

can do this later: called Shadowing not there yet need more…

FIGURE 6-1 A simple version of the Point class

```
/*
 * Class: Point
 * ------------
 * This class represents an x-y coordinate point on a two-dimensional
 * integer grid.
 */

#include <string>
#include "strlib.h"
using namespace std;

class Point {
public:

   Point() {
      x = 0;
      y = 0;
   }

   Point(int xc, int yc) {
      x = xc;
      y = yc;
   }

   int getX() {
      return x;
   }

   int getY() {
      return y;
   }

   string toString() {
      return "(" + integerToString(x) + ","
                 + integerToString(y) + ")";
   }

private:
   int x;
   int y;
};
```

Constructors

Getter methods

Public section

Instance variables

Private section

COSC 220: Data Structures in C++

Designing Classes

What else is missing?

FIGURE 6-1 A simple version of the Point class

```
/*
 * Class: Point
 * ------------
 * This class represents an x-y coordinate point on a two-dimensional
 * integer grid.
 */

#include <string>
#include "strlib.h"
using namespace std;

class Point {

public:

   Point() {
      x = 0;
      y = 0;
   }

   Point(int xc, int yc) {
      x = xc;
      y = yc;
   }

   int getX() {
      return x;
   }

   int getY() {
      return y;
   }

   string toString() {
      return "(" + integerToString(x) + ","
                 + integerToString(y) + ")";
   }

private:
   int x;
   int y;

};
```

COSC 220: Data Structures in C++

Designing Classes

What else is missing?
no means of changing
values of fields

export methods that
set values of certain
instance variables
called: **setters**
set methods also called:
mutators

replace pt.x = value;
with pt.setX(value);

Doing this: instance var's
no longer private
you can change them

setter methods rare in
OO design

FIGURE 6-1 A simple version of the Point class

```
/*
 * Class: Point
 * ------------
 * This class represents an x-y coordinate point on a two-dimensional
 * integer grid.
 */

#include <string>
#include "strlib.h"
using namespace std;

class Point {

public:

   Point() {
      x = 0;
      y = 0;
   }

   Point(int xc, int yc) {
      x = xc;
      y = yc;
   }

   int getX() {
      return x;
   }

   int getY() {
      return y;
   }

   string toString() {
      return "(" + integerToString(x) + ","
                 + integerToString(y) + ")";
   }

private:
   int x;
   int y;

};
```

Constructors

Getter methods

Public section

Instance variables

Private section

COSC 220: Data Structures in C++

Designing Classes

Modern C++
Classes designed in a way where impossible to change val's of instance variables after instantiation of the class. i.e. after the object is created.

Classes designed this way: **immutable**
(in context of C++)

FIGURE 6-1 A simple version of the Point class

```
/*
 * Class: Point
 * ------------
 * This class represents an x-y coordinate point on a two-dimensional
 * integer grid.
 */

#include <string>
#include "strlib.h"
using namespace std;

class Point {
public:

   Point() {
      x = 0;
      y = 0;
   }

   Point(int xc, int yc) {
      x = xc;
      y = yc;
   }

   int getX() {
      return x;
   }

   int getY() {
      return y;
   }

   string toString() {
      return "(" + integerToString(x) + ","
                 + integerToString(y) + ")";
   }
private:
   int x;
   int y;
};
```

COSC 220: Data Structures in C++

Designing Classes

C++ Class: recall implementation and interface sep'd

Class def'n occurs in header file (.h file)
Corresp'g code occurs in cpp file (.cpp)

Corresponding code in .cpp file occurs as indep. method definitions not nested within a class def'n.
→ need to specify class to which they belong
→ Add the class name as a **qualifier** before the method name with a double colon between
→ Thus fully qualified name of **getX** method for the point class is: **Point::getX**

COSC 220: Data Structures in C++

FIGURE 6-2 Preliminary interface for the Point class

```
/*
 * File: point.h
 * -------------
 * This interface exports the Point class, which represents a point on
 * a two-dimensional integer grid.
 */

#ifndef _point_h
#define _point_h

#include <string>

class Point {

public:

/*
 * Constructor: Point
 * Usage: Point origin;
 *        Point pt(xc, yc);
 * ------------------------
 * Creates a Point object.  The default constructor sets the coordinates
 * to 0; the second form sets the coordinates to xc and yc.
 */

   Point();
   Point(int xc, int yc);

/*
 * Methods: getX, getY
 * Usage: int x = pt.getX();
 *        int y = pt.getY();
 * -------------------------
 * These methods returns the x and y coordinates of the point.
 */

   int getX();
   int getY();

/*
 * Method: toString
 * Usage: string str = pt.toString();
 * ----------------------------------
 * Returns a string representation of the Point in the form "(x,y)".
 */

   std::string toString();

private:

   int x;                      /* The x-coordinate */
   int y;                      /* The y-coordinate */

};

#endif
```

COSC 220: Data Structures in C++

FIGURE 6-3 Preliminary implementation of the Point class

```
/*
 * File: point.cpp
 * ---------------
 * This file implements the point.h interface.
 */

#include <string>
#include "point.h"
#include "strlib.h"
using namespace std;

/*
 * Implementation notes: Constructors
 * ----------------------------------
 * The constructors initialize the instance variables x and y.  In the
 * second form of the constructor, the parameter names are xc and yc
 * to avoid the problem of shadowing the instance variables.
 */

Point::Point() {
   x = 0;
   y = 0;
}

Point::Point(int xc, int yc) {
   x = xc;
   y = yc;
}

/*
 * Implementation notes: Getters
 * -----------------------------
 * The getters return the value of the corresponding instance variable.
 * No setters are provided to ensure that Point objects are immutable.
 */

int Point::getX() {
   return x;
}

int Point::getY() {
   return y;
}

/*
 * Implementation notes: toString
 * ------------------------------
 * The implementation of toString uses the integerToString function
 * from the strlib.h interface.
 */

string Point::toString() {
   return "(" + integerToString(x) + "," + integerToString(y) + ")";
}
```

COSC 220: Data Structures in C++

Designing Classes

Code the point class interface and implementation

Create instances of the Point class and output to the console

Create another simple data structure class of your choice and create instances of it and output to the console.

COSC 220: Data Structures in C++

Designing Classes

Load in the following files and rename them using refractor:
my_point_starterCode.h → my_point.h
my_point_starterCode.cpp → my_point.cpp
point_test_starterCode.cpp → point_test.cpp

my_point.h and .cpp define the 'Point' class in a slightly different way than the one in Stanford libraries.
point_test.cpp → uses the point class

add this to Cmake:
set(SOURCE_FILES my_point.cpp point_test.cpp)

COSC 220: Data Structures in C++

Designing Classes

Operator Overloading:
Much as we overloaded existing operators in Python to be used with User defined data types
we can do the same in C++

Load in the following files and rename them using refractor:
my_point_starterCode.h → my_point.h
my_point_starterCode.cpp → my_point.cpp
point_test_starterCode.cpp → point_test.cpp

my_point.h and .cpp define the 'Point' class in a slightly different way than the one in Stanford libraries.
point_test.cpp → uses the point class

COSC 220: Data Structures in C++

Designing Classes

In your Cmake file:
add the following line:
set(SOURCE_FILES my_point.cpp point_test.cpp)

Note, we can still use Stanford libraries and also incorporate our own class and program that uses that class

In the point class, we have a method:
'toString()'
so if we have a **Point** called **pt** we can output it as follows:
cout << "pt = " << pt.toString() << endl;
Methods are used via the dot notation

COSC 220: Data Structures in C++

Designing Classes

so if we have a **Point** called **pt** we can output it as follows:
cout << “pt = ” << pt.toString() << endl;
Methods are used via the dot notation

We can overload the insertion operator “<<” in a way that we can output the Point pt directly:
cout << “pt = ” << pt << endl;

overloading operators → use keyword **operator** in front of symbol. e.g.
to redefine the + operator:
you def. a function named: **operator+**

COSC 220: Data Structures in C++

Designing Classes

so for insertion operator:
you def. a function named: **operator<<**
Note: the way the insertion operator works:
left operand of << is the output stream
right operand of << is what you insert into the stream

from before: **<iostream>** defines an entire hierarchy of output streams (more on class hierarchy in a bit).
ostream is the most general output stream (files, strings, chars, etc.)

Note: streams cannot be copied!!

So how do you pass a stream?

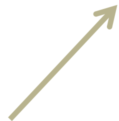
COSC 220: Data Structures in C++

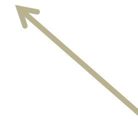
Designing Classes

Note: streams cannot be copied!!
So how do you pass a stream? Pass by reference.
Must also return the stream by reference as well.

Prototype looks like:

ostream & operator<<(ostream & os, Point pt);





return stream by ref pass stream by ref

COSC 220: Data Structures in C++

Designing Classes

Implementation looks like:

```
ostream & operator<<(ostream & os, Point pt) {
    os << pt.toString();
    return os;
}
```

Even simpler:

```
ostream & operator<<(ostream & os, Point pt) {
    return os << pt.toString();
}
```

COSC 220: Data Structures in C++

Designing Classes

Testing point for Equality:
Want to use the "==" operator.
Must overload it. But to compare points, we need to know the instance variables. In general, when overloading operators, 2 Methods:

1. Define operator as a *method* within the class. Here, left operand of the binary operator is receiver object and right operand is passed as a parameter.
2. Define operator as *free function* outside the class. Here, operands for a binary operator are both passed as parameters

COSC 220: Data Structures in C++

Designing Classes

1. Define operator as a *method* within the class. Here, left operand of the binary operator is receiver object and right operand is passed as a parameter.

Method 1: method based, put prototype in the Point class Public section.

```
bool operator==(Point rhs);
```

Implementation in my_point.cpp file

```
bool Point::operator==(Point rhs) {
    return x == rhs.x && y == rhs.y;
}
```

Since Class exported through interface, implementation of operator== you must specify that it's assoc'd to Point class. Use: "Point::"

```
bool Point::operator==(Point rhs) {
    return x == rhs.x && y == rhs.y;
}
```

if (pt == origin) ...

if **pt** and **origin** are of type Point, compiler designates **pt** as the receiver and then copies the value of **origin** to the parameter **rhs**.

unqualified ref's to **x** and **y** refer to fields of receiver **pt**
qualified express's **rhs.x** and **rhs.y** refer to fields in variable **origin**.

wild: operator== has access to private variables of both **pt** and **rhs** even though variables hold different objects. Def's in private section of class are private to the class and not objects.

COSC 220: Data Structures in C++

Designing Classes

2. Define operator as *free function* outside the class. Here, operands for a binary operator are both passed as parameters

Method 2: Free function outside the class

```
bool operator==(Point p1, Point p2);
```

prototype appears outside class definition

```
bool operator==(Point p1, Point p2) {
    return p1.x == p2.x && p1.y == p2.y
};
```

But doesn't work since do not have access to instance variables. How to resolve this?

COSC 220: Data Structures in C++

Designing Classes

Method 2: Free function outside the class

```
bool operator==(Point p1, Point p2) {
    return p1.x == p2.x && p1.y == p2.y
};
```

But doesn’t work since do not have access to private instance variables **x** and **y**. How to resolve this?

Designate operator function as a **friend**.

friend *prototype;*
friend bool operator==(Point p1, Point p2);
place in private section of class in the .h file.

can also do: **friend class** *name;*
This allows class to have access to private section of a class.

COSC 220: Data Structures in C++

Designing Classes

Add the “-” and “*” operators by overloading via method 1 and 2. Show that they both work by running point_test.cpp file. Then overload the == operator as a method.

Code a Vector2d class that has the following functions:
dot product which is overloaded * operator
isPerp() fctn
isParallel()
angleBet()
overload +, -, == operators
overload the * operator so that you can multiply a scalar by the vector

Extend to the Vector3d class and include the cross product. This returns a vector.