CSE 333 – SECTION 7

C++ practice

Mix C and C++ (HW4)

- **Including C Headers in C++ Code**
- Inform the C++ compiler that some methods you'll be calling are C methods

```
extern "C" {
    #include “Array2d.h"
}
```

Mix C and C++ (HW4)

- You can't get GTK+ to invoke a method on an object
- Basic Approach: provide an adapter function (See hw4 spec)

Mix C and C++ (HW4)

- Another good idea for you to think about:
- Write a wrapper class for your C code, e.g. Array2d.c becomes a Array2d class

C++

- **Object-oriented**
- A major addition of C++ is its support for classes & objects

C++ classes

- **Encapsulation and Abstraction**
  - encapsulation: hiding implementation details
  - abstraction: associating behavior with encapsulated state (invoking a method on an instance)

- **Access specifiers:**
  - Public: anything outside the class can access it
  - Protected: only this class and derived classes can access it
  - Private: only this class can access it

Polymorphism

- **static polymorphism:**
  - Multiple functions or methods with the same name, but different argument types (overloading)
  - Works for all functions, not just class members
- **dynamic polymorphism:**
  - Derived classes can override methods of parents

Operator Overloading

- A form of polymorphism.
- Give special meanings to operators in user-defined classes
- Special member functions in classes with a particular naming convention
- For E.g., for overloading the ‘=’ operator, define a member function named operator=

Common operators

- The most commonly overloaded operators are
  - = (assignment operator)
  - + - * (binary arithmetic operators)
  - += -= *= (compound assignment operators)
  - == != (comparison operators)

Example: Class Point

Point.h

Defined in the class,
so inline by default

```
#ifndef _POINT_H_
#define _POINT_H_
class Point {
 public:
  Point(const int x, const int y);  // constructor
  int get_x() const { return x_; }  // inline member function
  int get_y() const { return y_; }  // inline member function
  double Distance(const Point &p) const;  // member function
  void SetLocation(const int x, const int y);  // member functn
 private:
  int x_;  // data member
  int y_;  // data member
};  // class Point
#endif  // _POINT_H_
```

Point.cc

```
#include <cmath>
#include "Point.h"
Point::Point(const int x, const int y) {
  x_ = x;
  this->y_ = y;  // "this->" is optional, unless names conflict
}
double Point::Distance(const Point &p) const {
  // We can access p's x_ and y_ variables either through the
  // get_x(), get_y() accessor functions, or the x_, y_ private
  // member variables directly, since we're in a member
  // function of the same class.
  double distance = (x_ - p.get_x()) * (x_ - p.get_x());
  distance += (y_ - p.y_) * (y_ - p.y_);
  return sqrt(distance);
}
void Point::SetLocation(const int x, const int y) {
  x_ = x;
  y_ = y;
}
```

Overloading Operator =

```
Point &Point::operator=(const Point &pt) {
  if (this != &pt) {
    x_ = pt.x_;
    y_ = pt.y_;
  }
  return *this;
}
```

Sec7 exercise

- Write a C++ program that:
- has a class representing a rectangle
  - Use Point class to store the coordinates of the top-left corner
  - Has width, height

- has the following methods:
  - return the area of a rectangle
  - test if a point is inside a rectangle
  - overload “=”,