CSE 333 – SECTION 3

References, const and classes
This or that?

• Consider the following code:

**Pointers:**
```c
int i;
int *pi = &i;
```

**References:**
```c
int i;
int &ri = i;
```

In both cases,

```
pi
addr
```

```
ri
addr
```

The difference lies in how they are used in expressions:

```c
*pi = 4;
ri = 4;
```
Pointers and References

• Once a reference is created, it cannot be later made to reference another object. This is often done with pointers.
• References cannot be null, whereas pointers can.
• References cannot be uninitialized. It is impossible to reinitialize a reference.
C++ const declaration

- As a declaration specifier, `const` is a type specifier that makes objects unmodifiable.
  
  ```
  const int m = 255;
  ```

- Reference to constant integer:
  ```
  int n = 100;
  const int &ri = n;  //ri becomes read only
  ```
When to use?

- Function parameter types and return types and functions that declare overloaded operators.
- **Pointers**: may point to many different objects during its lifetime. Pointer arithmetic (++ or --) enables moving from one address to another. (Arrays, for e.g.)
- **References**: can refer to only one object during its lifetime.

**Style Guide Tip:**
- use const reference parameters to pass input
- use pointers to pass output parameters
- input parameters first, then output parameters last
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 function

private:
    int x_; // data member
    int y_; // data member
}; // class Point
Section Exercise – Part I

• Define a class Rectangle whose instance variables are a pair of Point objects (upper left, lower right).

• Include at least one constructor. Make sure you get const right in the right places.

• Methods:
  • `getul()`, `getlr()` - returns upper and lower points.
  • `cornerPoints()` – to obtain the corner points.
  • `area()` - returns the Rectangle's area.
  • `contains(Point &p)` - returns true or false depending on whether point p is inside the rectangle.
Part II

- Add a second constructor that takes 4 coordinates (minx, maxx, miny, maxy) and creates the upper left/lower right Point instance variables.
- Make the first constructor smart enough so the points can be any two diagonal corners and the constructor figures out what the top/bottom/left/right coordinates are and constructs upper left/lower right instance Point instance variables accordingly.
- Additional Methods:
  - `Intersects(Rectangle &other)` - returns true if this rectangle intersects the other one.
  - `BoundingBox(Rectangle &other)` - returns a new rectangle that tightly encloses both this rectangle and other.