CS 161
Intro to CS I

Classes and Objects
Object Oriented Programming

• **Procedural programming** uses variables to store data, and focuses on the processes/functions that occur in a program. Data and functions are separate and distinct.

• **Object-oriented programming** is based on objects that encapsulate the data and the functions that operate on it.
Jargon

- **object**: software entity that combines data and functions that act on the data in a single unit

- **attributes**: the data items of an object, stored in member variables or data members

- **member functions (methods)**: procedures/functions that act on the attributes of the class

- **encapsulation**: the bundling of an object’s data and procedures into a single entity
Class vs Object

• **Class**: a programmer-defined data type used to define objects
  – It is a pattern for creating objects

• **Object**: the instantiation of the pattern in the runtime environment

```java
string fName, lName;
creates two objects of the string class
```
class Square
{
  private:
    int side;
  public:
    void setSide(int s)
    { side = s; }
    int getSide()
    { return side; }
};
Access Specifiers

- Used to control access to members of the class.
- Each member is declared to be either:
  - `public`: can be accessed by functions outside of the class
  - `private`: can only be called by or accessed by functions that are members of the class
- Can be listed in any order in a class
- Can appear multiple times in a class
- If not specified, the default is `private`
Objects

• An **object** is an instance of a class
• It is defined as other identifiers
  
  ```
  Square sq1, sq2;
  ```

• It can access members using dot operator
  
  ```
  sq1.setSide(5);
  cout << sq1.getSside();
  ```
Special Member Functions

• **Accessor, get, getter** function: uses but does not modify a member variable
  
  ex: `getSide`

• **Mutator, set, setter** function: modifies a member variable
  
  ex: `setSide`

• **Constructor:** more on this later

• **Destructor:** more on this later
Member Function Definition

• Member functions are part of a class declaration
  – Can place entire function definition inside the class declaration
    or
  – Can place just the prototype inside the class declaration and write the function definition after the class
  – Member functions defined inside the class declaration are called **inline functions**
  – Only very short functions should be inline functions
class Square
{
private:
    int side;

public:
    void setSide(int s)
    {
        side = s;
    }

    int getSide()
    {
        return side;
    }
};
Member Function Definition

• Put a function prototype in the class declaration
• In the function definition, precede the function name with the class name and **scope resolution operator (::)**

```cpp
int Square::getSide() {
    return side;
}
```
Conventions and a Suggestion

Conventions:
• Member variables are usually **private**
• Accessor and mutator functions are usually **public**
• Use ‘get’ in the name of accessor functions, ‘set’ in the name of mutator functions

Suggestion: calculate values to be returned in accessor functions when possible, to minimize the potential for stale data
Inline vs Regular Functions

• When a regular function is called, control passes to the called function
  – the compiler stores return address of call, allocates memory for local variables, etc.

• Code for an inline function is copied into the program in place of the call when the program is compiled
  – This makes a larger executable program, but
  – There is less function call overhead, and possibly faster execution
Review

• Class is a pattern or description, say Dog
• Object is an instance, i.e. specific item of that pattern-
  – Rover
  – Fido
  – Spot
  – Dodger
class Circle
{
  private:
    double radius;

  public:
    void setRadius(double r)  // Mutator
    {
      radius = r;
    }

    double getArea()  // Accessor
    {
      return 3.14 * pow(radius, 2);
    }
};

int main()
{
  Circle circle1, circle2;

  circle1.setRadius(1);  // This sets circle1's radius to 1
  circle2.setRadius(2.5);  // This sets circle2's radius to 2.5

  cout << "The area of circle1 is " << circle1.getArea() << endl;
  cout << "The area of circle2 is " << circle2.getArea() << endl;

  return 0;
}

Example from book pages 409-416