Introduction to Classes

Classes - Terminology

- **Class definition** – where the structure of the class is defined (the name of the class, the members and their types, etc). This is normally contained in a header file.
- **Class implementation** – where the member functions are defined. This is normally contained in a source file.
- **Member** – the variables and functions that are a part of the class

Classes

Members of a class can be declared as

- **Public**
- **Private**
- **Protected**

Classes

Members of a class can be declared as

- **Public** – directly accessible by objects and pointers
- **Private**
- **Protected**

Classes

Members of a class can be declared as

- **Public** – directly accessible by objects and pointers
- **Private** – directly accessible by members and “friends” of the class
- **Protected**

Classes

Members of a class can be declared as

- **Public** – directly accessible by objects and pointers
- **Private** – directly accessible by members and “friends” of the class
- **Protected** – directly accessible by members and friends and derived classes

Class Definitions

In general:

```
class ClassName{
 public:
  Members (variables, functions);
 private:
  Members (variables, unctions);
};
```

Example:

```
class School{
 public:
  School(); // constructor
  void SetName(char *);
  void SetEnrollment(int);
  void SetTuition(double);
 private:
  char name_[30];
  int enrollment_;
  double tuition_;
};
```

Classes - Terminology

- **Object** – a variable of the class type
- **Instantiate** – create an object of the class
- **Constructor** – function that shares the same name as the class.
  - It can take parameters.
  - It can **not** return a value.
  - It is called whenever an object of the class is instantiated

Classes - Terminology

- **Default Constructor** – constructor that can be called with no arguments.
- **Copy Constructor** – constructor to create a copy of an existing object. Needed when the class has pointers as data members

Constructors..

- A constructor that can be called with a single argument should be marked explicit to avoid implicit casting.

Classes - Terminology

- **Destructor** – function that shares the same name as the class preceded by a tilde ~
  - It can not take any parameters or return a value.
  - It is called to clean house once the program leaves the scope of an object (with the exception of static objects).

Classes - Terminology

- **Accessor Function (Get Function)** – function that returns the value of a private data member
- **Mutator Function (Set Function)** – function that modifies the value of a private data member
- **Utility Function** – private member function to perform tasks for other member functions

Constructors and Destructors

- The constructor for an object is called as soon as the object is instantiated. The constructor for global objects are called first.
- The destructor for an object is called as soon as the program leaves the scope of the object UNLESS – the object is of the static storage class, then its destructor is called once the program leaves main.
- In general, the objects follow the stack model – Last In, First Out.

Writing Member Functions

- When writing the implementation of a member function you must tie it to the class with the scope resolution operator (::)

  Ex
```
void School::PrintSchool()
                { body }
```

Objects

- Declare an object just like any other variable.

```
School s1;
```

- Use the dot operator to access members of an object.

```
s1.SetName(“USC”);
```

- Use the arrow operator to access members when using a pointer to an object.

```
School * sptr;
sptr->SetEnrollment( 35000 );
```

Miscellaneous

- Assignment Statements – defaults to member-wise copy
- Members can not be initialized in the class definition.
- Constructors can be overloaded. Destructors can not be overloaded.

Miscellaneous

- Returning a reference to a private data member. It’s possible but be very careful!

Definition vs. Implementation

It is a good idea to place the class definition and class implementation in separate files – definition in a header file, implementation in a source file.