Common programming Error

- Forgetting the **semicolon** at the end of a class (or structure) definition is a syntax error.
- Specifying a **return type** and/or a return value for a **constructor** is a syntax error.
- Attempting to initialize a data member of a class explicitly in the class definition is a syntax error.
- When defining a class's member functions outside that class, omitting the class name and scope resolution operator on the function name is a syntax error.
- An attempt by a function which is not a member of a particular class (or a **friend** of that class) to access a **private** member of that class is a syntax error.
- Data members of a class cannot be initialized in the class definition.
- Attempting to declare a return type for a constructor and/or attempting to return a value from a constructor are syntax errors.
- Specifying default initializers for the same member function in both a header file and in the member function definition.
- It is a syntax error to attempt to pass arguments to a destructor, to specify a return type for a destructor (even **void** cannot be specified).to return values from a destructor, or to overload destructor.
- A constructor can call other member functions of the class such as set or get functions, but because the constructor is initializing the object, the data members may not yet be in consistent state. Using data members before they have been properly initialized can cause logic errors.

Performance Tip

- Structures are ordinarily passed call-by –value. To avoid the overhead of copying a structure, pass the class object by-reference.
- Defining a small member function inside the class definition automatically inlines the member function (if the compiler chooses to do so). This can improve performance, but it does not promote the best software engineering because clients of the class will be able to see the implementation of the function.
- Actually objects contain only data, so objects are much smaller than if they also contained functions. Applying operator **sizeof** to a class name or to an object of that class will report only the size of the class's data. The compiler creates one copy (only) of the class's member functions separate from all objects of the class. All objects of the class share this one copy of the member functions. Each object, of course, needs its own copy of the class's data because this data can vary among the objects. The function code is nonmodifiable (also called reentrant code or pure procedure) and hence can be shared among all objects of one class.
- Passing an object call-by-value is good from a security stand pointer because the called function has no access to the original object, but call-by-value can degrade performance when making a copy of a large object. An object can be passed by-reference by passing either a pointer or a reference to the object. Call-by-reference offers good performance but is weaker from a security

standpoint because the called function is given access to the original object. Call-by-**const**-reference is a safe, good –performing alternative.

Software Engineering observation

- To avoid the overhead of call-by-value yet still gain the benefit that the caller's original data is protected from modification, pass large-size arguments as const references.
- It is important to write programs that are understandable and easy to maintain. Change is the rule rather than the exception. Programmers should anticipate that their code will be modified. As we will see, classes can facilitate program modifiability.
- Clients of a class use the class without knowing the internal details of how the class is implemented. If the class implementation is changed (to improve performance, for example), provided the class's interface remains constant, the class's client source code need not change (although the client may need to be recompiled). This makes it much easier to modify systems.
- Member functions are usually shorter than functions in non-object-oriented programs because the data stored in the data m embers has ideally been validated by a constructor and/or by member functions that store new data. Because the data is already inside the object, the member function calls often have no arguments or at least have fewer arguments than typical function calls in non-object-oriented languages. Thus, the calls are shorter, the function definitions are shorter and the function prototypes are shorter.
- Clients have access to a class's interface but should not have access to a class's implementation.

- Declaring member functions inside a class definition (via their function prototypes) and defining those member functions outside that class definition separates the interface of a class from its implementation of that class's member functions.

- Only the simplest member functions should be defined in the class header.

- Using an object –oriented programming approach can often simplify function calls by reducing the number of parameters to be passed. This benefit of object-oriented programming derives from the fact that encapsulation of data members and member function within an object gives the member functions the right to access the data members.

- A central theme of this course is "reuse, reuse, and reuse". We will carefully discuss a number of techniques for "polishing" classes to encourage reuse. We focus on "crafting valuable classes" and creating valuable "software assets."

- Place the class declaration in a header file to be included by any client that wants to use the class. This forms the class's **public** interface (and provides the clients with the function prototypes it needs to be able to call the class's member functions. Place the definitions of the class member functions in a source file. This forms the implementation of the class.

- Clients of a class do not need access to the class's source code in order to use the class. The clients do, however, need to be able to link to the class's object code .This encourages independent software vendors (ISVs) to provide class libraries for sale or license. The ISVs provide in their products only the header files and the object modules. No proprietary information is revealed - as would be the case if source code were provided. The C++ user community benefits by having more ISV – produced class libraries available.

- Information important to the interface to a class should be included in the header file. Information that will be used only internally in the class and will not be needed by clients of the class should be included in the unpublished source file. This is yet another example of the principle of least privilege.
- C++encourages programs to be implementation independent. When the implementation of a class used by implementation –independent code changes. That code need not be modified, but it may need to be recompiled.
- Keep all the data members of a class **private**. Provide **public** member functions to set the values of **private** data members and to get the values of **private** data members. This architecture helps hide the implementation of a class from its clients, which reduces bugs and improves program modifiability.
- Class designers use **private**, **protected**, and **public** members to enforce the nation of information hiding and the principle of least privilege.
- The class designer need not provide set and/or get functions for each **private** data item; these capabilities should be provided only when appropriate.
- Member functions tend to fall into a number of different categories: functions that read and return the value of private data members: functions that set the value of private data members; functions that implement the features of the class; and functions that perform various mechanical chores for the class such as initializing class objects, assigning class objects, converting between classes and built-in types or between classes and other classes, and handling memory for class objects.
- A phenomenon of object-oriented programming is that once a class is defined, creating and manipulating objects of that class usually involves issuing only a simple sequence of member function calls-few, if any ,control structure are needed .By contrast, it is common to have control structure in the implementation of a class's member functions.
- If a member function of a class already provides all or part of the functionality required by a constructor (or other member function) of the class, call that member function from the constructor (or other member function). This simplifies the maintenance of the code and reduces the likelihood of an error if the implementation of the code is modified. As a general rule: Avoid repeating code.
- It is possible for a class not to have a default constructor.
- As we will see (throughout the remainder of the book), constructors and destructors have much greater prominence in C++ and object-oriented programming than is possible to convey after only our brief introduction here.

- Making data members **private** and controlling access, especially write access, to those data members through **public member functions** helps ensure data integrity.
- Accessing **private** data through set and get member functions not only protects the data members from receiving invalid values, but it also insulates clients of the class from the representation of the data members. Thus, if the representation of the data changes for some reason (typically to reduce the amount of storage required or to improve performance), only the member functions need to change –the clients need not change as long as the interface provided by the member functions remains the same. The clients may, however, need to be recompiled.

Good programming practice

- Use each member access specifier only once only once in a class definition for clarity and readability. Place **public** members first where they are easy to locate.
- Use the name of the header file with the period replaced by an underscore in the **# ifndef** and **#define** preprocessor directives of a header file.
- If you choose to list the **private** members first in class definition, explicitly use the **private** label despite the fact that **private** is assumed by default. This improves program clarity. Our preference is to list the **public** members of a class first to emphasize the class's interface.
- Despite the fact that the **public** and **private** labels may be repeated and intermixed, list all the **public** members of a class first in one group and then list all the **private** members in another group. This focuses the client's attention on the class's **public** interface, rather than on the class's implementation.

- When appropriate (almost always), provide a constructor to ensure that every object is properly initialized with meaningful values. Pointer data members, in particular, should be initialized to some legitimate pointer value or to 0.

- Declare default function argument values only in the function prototype within the class definition in the header file.

- Member functions that set the values of private data should verify that the intended new values are proper; if they are not, the set functions should place the **private** data members into an appropriate consistent state.

- Never have a public member function return a non-**const** reference (or a pointer) to a **private** data member. Returning such a reference violates the encapsulation of the class.