Friend, Nesting and Static Members

Sukhbir Tatla

sukhbir.tatla@senecacollege.ca

Friends

- Remember that with data hiding and encapsulation, we force clients to manipulate private data through public member functions. By declaring friends, we allow non-member functions or member functions of other classes access to private data.

- A class can declare another class to be a friend.

- When we do this, the first class gives all member functions of the second class permission to access all of its protected and private information (not the other way around; a class cannot declare itself a friend of another class).

Friend Classes

- When a class (declaring class) declares another class (friend class) to be a friend, the friend class' member functions can invoke the protected or private member functions or use the protected or private data members of the declaring class.

- To do so, the friend class must have access to an object of the declaring class' type.

- This can be done by having an object of the friend class as a data member or a pointer to an object of the friend class.

Non-Member Friend Functions

- A non-member function can be declared a friend by placing the following statement in the declaring class. The declaration of a friend function takes the form of a function prototype statement, preceded by the keyword friend.

- Typically, friend functions are designed with formal arguments, where clients pass either an object, the address of an object, or a reference to an object as an argument.

```
class declaring_class_name {
  friend return_type function_name(arg_list);
};
```

Member Function Friends

- The declaration of a friend member function takes the form of a member function prototype statement, preceded by the keyword friend.
- Member functions are declared using their class name followed by the scope resolution operator.

MEMBER FUNCTION FRIENDS

- The friend member function must have an object of the class to which it is a friend -- from a formal argument, as a local object in the member function's implementation, as a data member in the member function's class, or as a global object.

```
class declaring_class_name {
   friend return_type
  class_name::function_name(arg_list);
   };
```

Member Function Friends

- But… A member function cannot be declared a friend before it is first declared as a member of its own class.
- Unlike a friend class, where we can have an incomplete declaration, friend member function declarations (in our declaring class) cannot take place until after the member functions have been declared (in their own class).
- If two or more classes share the same friend class or function, those classes share a mutual friend. This means that more than one class gives a single friend class or function permission to access their members.

FRIEND FUNCTIONS AND FRIEND CLASSES

- **`friend`** function and **`friend`** classes
  - Can access **`private`** and **`protected`** (more later) members of another class
  - **`friend`** functions are not member functions of class
    - Defined outside of class scope
- Properties
  - Friendship is granted, not taken
  - NOT symmetric (if B a **`friend`** of A, A not necessarily a **`friend`** of B)
  - NOT transitive (if A a **`friend`** of B, B a **`friend`** of C, A not necessarily a **`friend`** of C)

Practice

- REALITY CHECK (Week 4)
- Questions #1 (Word Problem)
- Questions #2 (Word Problem)

Nesting

Nesting

- Nesting restricts the scope of a class' name and can reduce the global namespace pollution.

- A class' definition can be placed inside the public, protected, or private sections of another class. The purpose of this is to hide the nested class' name inside another class, restricting access to that name.

- It does not give either the outer class or the nested class any special privileges to access protected or private members of either class.

- A nested class does not allow the outer class access to its hidden members....the name of the nested class is hidden.

Nesting

- If a class is defined within another class in the public section, the nested class is available for use the same as objects defined for the outer class.
- To define objects of a public nested class, the name of the nested class must be qualified by the outer class' name (i.e., outer_class::nested_class).

```
class outer_class {
  public:
    class nested_class {
  public:
      nested_class(); //nested class'
constructor
    ...
};
```

- outer_class::nested_class object; //define an object

Nesting

- If a class is defined within another class in the private section, the nested class is available for use only by the member functions of the outer class and by friends of that class. Clients cannot create objects of the nested class type.
- In the implementation of a nested class' member functions, where the interface is separate from the implementation, an additional class name and scope resolution operator is required.

```
//nested class' constructor
  implementation
outer_class::nested_class::nested_class
  () {
   ...
}
```

Practice

- REALITY CHECK (Week 4)
- Questions #3 (Word Problem)
- Questions #4 (Word Problem)

Static Members

Static Class Members

`static` class members

- Shared by all objects of a class
  - Normally, each object gets its own copy of each variable
- Efficient when a single copy of data is enough
  - Only the **`static`** variable has to be updated
- May seem like global variables, but have *class scope*
  - Only accessible to objects of same class
- Initialized at file scope
- Exist even if no instances (objects) of the class exist
- Can be variables or functions
  - **`public`**, **`private`**, or **`protected`**

Static Members

- Shared properties are represented by static class members. Static class members can be either static data members or static member functions. These members can be declared in either the public, protected, or private sections of a class interface.
- When a member function is declared as static, we are not limited to using this function through an object of that class. This function can be used anywhere it is legal to use the class itself. This is because static member functions are invoked independent of the objects and do not contain a this pointer.
- It can be called via: `class::function(args)`

STATIC MEMBERS

- To specify static member functions, we must declare the functions to be static in the class definition (interface file) using the keyword static .
- Then, define those static member functions in the class implementation file.
- The definition of our static member functions should look identical to the definition of non-static member functions.
- The scope of a static member function is the same as that of a non-static member function.
- Access to a static member function from outside the class is the same as access to a non-static member function and depends on whether the member is declared as public, protected, or private.

Static Class Members (II)

- Accessing **`static`** members
  - **`public static`** variables: accessible through any object of the class
    - Or use class name and (: :)

      `Employee::count`
  - **`private static`** variables: a **`public static`** member function must be used.
    - Prefix with class name and (: :)

      `Employee::getCount()`
  - **`static`** member functions cannot access non-**`static`** data or functions

Static Data Members

- When we use the keyword static in the declaration a data member, only one occurrence of that data member exists for all instances of the class. Such data members belong to the class and not to an individual object and are called static data members. Static data represents class data rather than object data.

- To specify a static data member, we must supply the declaration inside the class interface and the definition outside the class interface. Unlike a non-static data member, a static data member's declaration does not cause memory to be allocated when an object is defined.

Static Data Members

```
//static data member declaration (class
  interface)
class class_name {
   ...
   static data_type static_data_member;
   ...
};
```

- A static data member's definition must be supplied only once and is usually placed in the class implementation file. A static data member's definition must be preceded by the class name and the scope resolution operator before the static data member's identifier.

```
//static data member definition (class
  implementation)
data_type class_name::static_data_member =
  initial_value;
```

Static Data Members

- It is important to realize that memory is allocated for static data members when we explicitly define those static data members, not when we declare the static data members as part of a class definition. Think of the static data member declarations within a class as external references to data members defined elsewhere.

- Clients can access a public static data member by saying `class_name::static_data_member`

- It is also possible for clients to access a public static data member once an object is defined for that class by saying object_name.static_data_member.

PRACTICE

- REALITY CHECK (Week 4)
- Questions #5 (Word Problem)

THANK YOU!

Any Questions Please?