Handouts/cse687/Code/DarkCorners

C++ Dark Corners

Jim Fawcett
CSE687 – Object Oriented Design
Spring 2014

C++ is a large capable language. Like most large languages it has some dark corners – places that you need to treat carefully. Here's a list of all the dark areas I can think of:

1. Compiler generated functions
2. Constructor initialization
3. Overriding errors
4. Overloading out of scope
5. Using default parameters with virtual functions
6. Virtual destructors
7. Multiple Inheritance

Handouts/cse687/Code/DarkCorners

The compiler will generate certain member functions:

1. If, and only if, no constructors are declared the compiler will generate, if needed, a default constructor that does default construction of each of the class bases and member data.
2. If no copy constructor, assignment operator, or destructor is declared in a class the compiler will generate one, if needed, which does copy, assignment, or destruction of each of the class bases and member data. This is only correct if each of the bases and data members has correct copy, assignment, or destruction semantics.
3. So, for each class you design you should decide to let the compiler generate these functions if correct. Otherwise you must either implement them or make them inaccessible by making private or declare =delete on each signature.

Each constructor may be initialized with list:

1. When a constructor is called, as its first action, it calls a constructor for each of its bases and data members. For non-default constructors you should always specify how each of the bases and data members is to be constructed using an initialization list. If the compiler chooses which constructor to call it may not choose correctly.
2. Constant and reference members must be initialized with an initialization list since they cannot be reset.

Overriding:

1. Avoid redefining, in derived classes, non-virtual base class functions
    a. Non-virtual member functions do not have vtable entries and so the function called is the type of the pointer or reference, not the type of object attached to the pointer or reference, breaking polymorphism.
    b. So it is possible for a base class function to be called on a derived class object, with possibly disastrous results.

Handouts/cse687/Code/DarkCorners

Overloading:

1. Avoid overloading non-virtual base class functions in derived classes
    a. Overloads work only within a single scope, not across both base and derived class scopes.
    b. The result may be hiding of base class member functions that are inherited by the derived class.
2. Avoid overloading virtual functions
    a. If a derived class redefines a base class virtual function which has overloads in the base that will hide the base class overloads that are inherited.

Avoid using default parameters in virtual functions:

1. Parameters don't have vtable entries, so they are bound based on the type of pointer or reference to an object, not of the object type.
2. This results in a derived class using base class defaults even though the derived class defined different values for the defaulted parameters.

Always provide a virtual destructor if your class may be used as the base class for a derivation:

1. If you don't do that, and a client creates an instance of a class derived from your base class on the heap, bound to a base pointer, then when the client calls delete on that pointer, the destructor called is based on the type of pointer not the type of object, so only the base destructor will be called.

Multiple virtual Inheritance of implementation:

1. Virtual multiple inheritance merges multiple copies of an inherited base into one unique shared base type. This type is initialized by the most derived class in the inheritance chain. That means that one or more based of the most derived may not be initialized as they specified in their constructors. So correct code can break correct code under multiple virtual inheritance.
2. The best solution is to design your classes so that virtual inheritance is not needed.
3. Note that in virtual inheritance the keyword virtual qualifies classes not functions.

Handouts/cse687/Code/DarkCorners

Definitions:

1. **Overriding:**
   Provide, in a derived class, a declaration and definition of a virtual base class function, using exactly the same function signature and the same or covariant return type.
   a. A covariant return type is a pointer or reference of the derived type, when the base virtual function returns a pointer or reference of the base type.
   b. It is optional, but desirable, to declare the derived function as virtual.

2. **Overloading:**
   Provide, in the same class, or in the same global scope, a function definition that uses the function identifier of another existing function with a different sequence of formal parameter types.
   a. Note that you cannot overload on return type, because a client is not compelled to use the return type, so the compiler cannot figure out which function to bind to.