C++ Classes

access specifiers
private:  // only class access
protected:  // only class and subclasses can access
public:  // anyone can access

friend
	Used to allow access to private or protected members by another
	specified function or class
	Often used for operator< and operator<< etc.
friend class X;  // class X is a friend
friend int f(int);  // function f is friend
struct and class

struct defaults everything to public
class defaults everything to private
   including both members and base classes

Typical class declaration and definition

// X.h
class X {
public:
   void f(int);
}; // X

// X.cpp
void X::f(int i) {
   cout << i;
}

Definition and declaration at same time

This is required when declaring a templated class using g++

// X.h
class X {
public:
   void f(int i) {
      cout << i;
   }
}; // X

This way "inlines" the function definitions
Subclassing

access on base classes specifies access by users of the subclass to superclass members

must use virtual member function declaration to allow member to be overridden in subclass

pure virtual member functions can be overridden but do not need a definition in the base class

Any class that has a pure virtual member function cannot be instantiated

but a subclass can be instantiated if it has no pure virtual members

Such a class is an abstract class

class Super { // abstract class - pure virtual member
    int f(int i); // not overridable
    virtual int g(int i); // virtual - can override
    virtual int h(int i) = 0; // pure virtual
};

class Sub : public Super {
    virtual int f(int i); // not overriding
    virtual int g(int i); // overriding
    virtual int h(int i); // overriding
};
Special class members

  default constructor
  default copy constructor
  default operator=
  default destructor

  all classes with subclasses should declare their destructor as virtual

Conversions

  Conversions
  function arguments
  initializers
  return values
  statement expressions

  conversion constructors
  explicit keyword

  conversion member function

class X {
public:
X(); // default constructor
X(X &x); // copy constructor
X(const X &x); // const copy constructor
X & operator=(const X &x) // assignment
virtual ~X(); // overridable destructor

X(const int &i); // conversion from int
explicit X(const Person &p); // not automatic
operator double(); // auto from X to double
};
const
   "not writable"
In
class X {
const1 int * f(const2 int *p) const3;
   const1 = the return value points to a const
   const2 = the argument points to a const
   const3 = the *this object is const
this pointer
this is a pseudo variable in all nonstatic member functions
It points to the object the member function was called on
the object is *this
operators such as operator= operator<< operator>>
generally return their left-hand-side or *this
static members
  static data members
    like a java class variable
    not a part of a class instance - only one class value
static member functions
  not invoked on a class instance
  no this pseudo variable
g++ template instantiation

g++ requires templated classes to be defined in the .h file
A corresponding .cpp file is not needed
This was the original way classes were defined in C++
The loader will discard duplicate definitions
For templated classes this is necessary (with g++) to allow instantiation of the templated class with the specified types