12.2 A `private` member is inaccessible from anywhere outside its class definition. A `protected` member is inaccessible from anywhere outside its class definition, with the exception that it is accessible from the definitions of derived classes.

12.3 In an inheritance hierarchy, each default constructor invokes its parent's default constructor before it executes itself, and each destructor invokes its parent's destructor after it executes itself. The effect is that all the parent default constructors execute in top-down order, and all the parent destructors execute in bottom-up order.

12.4 A `virtual` member function is a member function that can be overridden in a subclass.

12.5 A pure `virtual` function is a `virtual` member function that cannot be called directly; only its overridden functions in derived classes can be called. A pure `virtual` function is identified by the initializer `=0` at the end of its declaration.

12.6 A *memory leak* is the loss of access to memory in a program due to the wrong destructor being invoked. See Example 12.12 on page 285.

12.7 By declaring a base class destructor `virtual`, memory leaks as in Example 12.12 on page 285 can be prevented because after it is invoked its indicated subclass destructor(s) will also be invoked.

12.8 An abstract base class is a base class which includes at least one pure `virtual` function. Abstract base classes cannot be instantiated.

12.9 A concrete derived class is a subclass of an abstract base class that can be instantiated; *i.e.*, one which contains no pure `virtual` functions.

12.10 Static binding refers to the linking of a member function call to the function itself during compile time, in contrast to dynamic binding which postpones that linking until run time. Dynamic is possible in C++ by using virtual functions and by passing pointers to objects.

12.11 *Polymorphism* refers to the run-time binding that occurs when pointers to objects are used in classes that have `virtual` functions. The expressions `p->f()` will invoke the functions `f()` that is defined in the object to which `p` points. However, that object could belong to any one of a series of subclasses, and the selection of subclass could be made at run time. If the base-class function is `virtual`, then the selection (the "binding") of which `f()` to invoke is made at run time. So the expression `p->f()` can take "many forms."

12.12 Polymorphism promotes extensibility by allowing new subclasses and methods to be added to a class hierarchy without having to modify application programs that already use the hierarchy's interface.

12.13 The `protected` data member `a` can be accessed from the derived `Y` only if it is the member of the current object (*i.e.* only if it is `this->a`). `Y` cannot access `x.a` for any other object `x`.

Solutions to Problems

12.1 First we implement a `Card` class:

```
enum Rank {TWO=2, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN
           JACK, QUEEN, KING, ACE};
enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES };

class Card
{   friend class Hand;
    friend class Deck;
    friend ostream& operator<<(ostream&, const Card&);
  public:
    char rank() { return rank_; }
    char suit() { return suit_; }
  private:
    Card() { };
    Card(Rank rank, Suit suit) : rank_(rank), suit_(suit) { };
```