C++ 3

- More on visibility of member functions and of inheritance
- More on virtual functions
- Iterators
- Breaking encapsulation - friend functions

Visibility Examples

protect.cc
protectPrivate.cc

- Show public inheritance
- Demonstrate that private member functions cannot be used in the derived class
- Demonstrate that protected member functions can be used in the derived class but not by a user of the base class
- Second version shows private inheritance
  – Can’t use public base class member functions on derived class objects EXCEPT inside derived class

Example

```
include <stream.h>
class A
{ int a;
  public: val_public()
      {cout << "in the A::val_public()  \n";}
  protected: val_protected()
      {cout << "in the A::val_protected()  \n";}
  private: val_private()
      {cout << "in the A::val_private()  \n";}
};
```

Example

```
class C : public A
{  public:
 void v1()
 {cout<< "in the C::v1---call---"; val_public(); }
 void v2()
 {cout << "in the C::v2---call---";val_protected();}
/*  the following declaration is illegal because
   val_private() is a private member of class A.
  void v3()
  {  val_private(); }  */
};
```

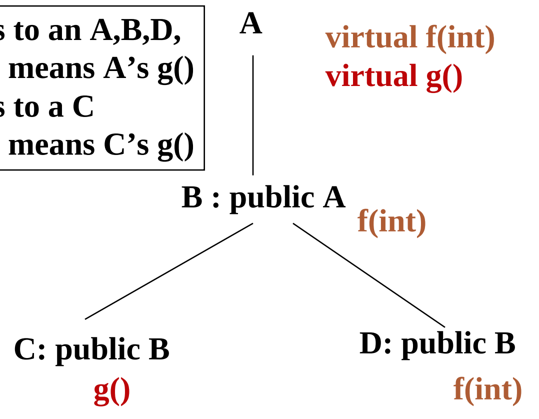
Example

```
main()
{    A a; C c;
     cout << "testing class C\n";
     c.val_public();
//   c.val_protected(); illegal because
       //val_protected() is protected in class A
//   c.val_private(); illegal because val_private()
       //is a private member of class A
     c.v1();
     c.v2();
}
```

```
58 remus!c++> a.out
testing class C
in the A::val_public()
in the C::v1---call---in the A::val_public()
in the C::v2---call---in the A::val_protected()
```

Virtual Member Functions

If a refers to an A,B,D,
then a.g() means A's g()
If a refers to a C
then a.g() means C's g()



If a refers to an A then a.f(0) is A"s f
If a refers to a B,C then a.f(1) is B's f
If a refers to a D then a.f(1) is D's f

Iterators

- Provide a way of examining a collection of objects, one by one
- In Java,
  - Form a new class which implements the Enumeration interface and provides standard `nextElement()` and `hasMoreElements()` functions
  - Constructor copies the container object (or keeps a local "pointer" into the container)

Iterators

- In C++, also need to create a new iterator class, but it needs to know the details of the collection implementation
  - Friend class
  - int nextElement(ele &) int 0 means no more elements; returns reference to the element in its argument

Example

```
#include<stdio.h>
#include<stream.h>
//stack class implemented as a vector
#define MAX 20
const int EMPTY = -1;
class stack
{     friend class stack_iterator;
      int s[MAX]; //private members
      int top;
 public: //public class interface
      stack() { top = EMPTY;}
      int isEmpty() {return (int) (top == EMPTY);}
      int isfull() {return (int) (top == MAX);}
      void push (int data) {s[++top] = data;} //no
  error check!
      void pop() {top--;} //no error check!
      int peek() {return s[top];}
};
```

Example

```
class stack_iterator
{      int current;
       stack *a;
  public:
     stack_iterator(stack *b)
     { a = b; //this iterator does not make a copy
              //of the stack object to save space;
             //uses a pointer to the original stack
         current = b->top;
     }
     int nextElement(int& j)
     {  if (current == EMPTY) {return 0;}
         else {j = a->s[current--]; return 1;}
};
```

Example

```
main()
{    stack q; stack * pq; int j; stack_iterator *iter;
     pq = &q;
     q.push(1); q.push(2); q.push(3); q.push(4);
     cout <<"top of stack is " << q.peek() << "\n";
     q.pop();
     cout <<"top of stack is "<< q.peek() << "\n";
     q.push(-3); q.push (-2);
     cout << "top of stack is "<< q.peek() << "\n";
}
//stack contains 1 2 3 -3 -2 here
//remember peek() does not pop the stack
```

```
 51 remus!c++> a.out
top of stack is 4
top of stack is 3
top of stack is -2
```

Example

```
  iter = new stack_iterator(pq);
  //   q.push(-1);//what will happen if we mutate the
  // stack after creating the iterator object; this
  // iterator won't see the new element
       cout << "printing entire stack ";
       while (iter->nextElement(j)) cout << " " << j ;
       cout << "\n";

   }
```

```
Output:
printing entire stack  -2 -3 3 2 1
```