OOP/Computer Programming

By : Dr. Danish Shehzad

Composition: Objects as Members of Classes

- Composition
  - Class has objects of other classes as members
- Construction of objects
  - Member objects constructed in order declared
    - Not in order of constructor's member initializer list
  - Constructed before their enclosing class objects (host objects)

```cpp
// Fig. 7.4: date1.h
// Declaration of the Date class.
// Member functions defined in date1.cpp
#ifndef DATE1_H
#define DATE1_H

class Date {
public:
   Date( int = 1, int = 1, int = 1900 ); // default constructor
   void print() const;  // print date in month/day/year format
   ~Date();  // provided to confirm destruction order
private:
   int month;  // 1-12
   int day;    // 1-31 based on month
   int year;   // any year

   // utility function to test proper day for month and year
   int checkDay( int );
};

#endif
```

```cpp
// Fig. 7.4: date1.cpp
// Member function definitions for Date class.
#include <iostream>

using std::cout;
using std::endl;

#include "date1.h"

// Constructor: Confirm proper value for month;
// call utility function checkDay to confirm proper
// value for day.
Date::Date( int mn, int dy, int yr )
{
   if ( mn > 0 && mn <= 12 )       // validate the month
      month = mn;
   else {
      month = 1;
      cout << "Month " << mn << " invalid. Set to month 1.\n";
   }

   year = yr;                      // should validate yr
   day = checkDay( dy );           // validate the day

   cout << "Date object constructor for date ";
   print();         // interesting: a print with no arguments
   cout << endl;
}

```

Constructor will print a line when called.

```
// Print Date object in form  month/day/year
void Date::print() const
   { cout << month << '/' << day << '/' << year; }

// Destructor: provided to confirm destru
Date::~Date()
{
   cout << "Date object destructor for date ";
   print();
   cout << endl;
}

// Utility function to confirm proper day value
// based on month and year.
// Is the year 2000 a leap year?
int Date::checkDay( int testDay )
{
   static const int daysPerMonth[ 13 ] =
      {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

   if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
      return testDay;

   if ( month == 2 &&      // February: Check for leap year
        testDay == 29 &&
        ( year % 400 == 0 ||
         ( year % 4 == 0 && year % 100 != 0 ) ) )
      return testDay;

   cout << "Day " << testDay << " invalid. Set to day 1.\n";

   return 1;  // leave object in consistent state if bad value
}
```

Destructor will print a line when called.

```
// Fig. 7.4: emply1.h
// Declaration of the Employee class.
// Member functions defined in emply1.cpp
#ifndef EMPLY1_H
#define EMPLY1_H

#include "date1.h"

class Employee {
public:
   Employee( char *, char *, int, int, int, int, int, int );
   void print() const;
   ~Employee();  // provided to confirm destruction order
private:
   char firstName[ 25 ];
   char lastName[ 25 ];
   const Date birthDate;
   const Date hireDate;
};

#endif
```

Composition - including objects of other classes.

```
// Fig. 7.4: emply1.cpp
// Member function definitions for Employee class.
#include <iostream>

using std::cout;
using std::endl;

#include <cstring>
#include "emply1.h"
#include "date1.h"

Employee::Employee( char *fname, char *lname,
                    int bmonth, int bday, int byear,
                    int hmonth, int hday, int hyear )
   : birthDate( bmonth, bday, byear ),
     hireDate( hmonth, hday, hyear )
{
   // copy fname into firstName and be sure that it fits
   int length = strlen( fname );
   length = ( length < 25 ? length : 24 );
   strncpy( firstName, fname, length );
   firstName[ length ] = '\0';

   // copy lname into lastName and be sure that it fits
   length = strlen( lname );
   length = ( length < 25 ? length : 24 );
   strncpy( lastName, lname, length );
   lastName[ length ] = '\0';

   cout << "Employee object constructor: "
        << firstName << ' ' << lastName << endl;
}
```

Constructor will print a line when called.

```
137
138 void Employee::print() const
139 {
140    cout << lastName << ", " << firstName << "\nHired: ";
141    hireDate.print();
142    cout << "  Birth date: ";
143    birthDate.print();
144    cout << endl;
145 }
146
147 // Destructor: provided to confirm destruction order
148 Employee::~Employee()
149 {
150    cout << "Employee object destructor: "
151          << lastName << ", " << firstName << endl;
152 }
```

The **print** function is **const** and will print whenever a **Date** object is created or destroyed. It can print **const** objects because it is a **const** function.
Print requires no arguments, it is linked implicitly to the object that calls it.

Destructor will print a line when called.

```
// Fig. 7.4: fig07_04.cpp
// Demonstrating composition: an object with member objects.
#include <iostream>

using std::cout;
using std::endl;

#include "emply1.h"

int main()
{
   Employee e( "Bob", "Jones", 7, 24, 1949, 3, 12, 1988 );

   cout << '\n';
   e.print();

   cout << "\nTest Date constructor with invalid values:\n";
   Date d( 14, 35, 1994 );  // invalid Date values
   cout << endl;
   return 0;
}
```

Only **emply.h** has to be loaded; that file has the command to load **date.h**.

```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Employee object constructor: Bob Jones

Jones, Bob
Hired: 3/12/1988  Birth date: 7/24/1949

Test Date constructor with invalid values:
Month 14 invalid. Set to month 1.
Day 35 invalid. Set to day 1.
Date object constructor for date 1/1/1994

Date object destructor for date 1/1/1994
Employee object destructor: Jones, Bob
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949
```

Notice how inner objects are created first and destroyed last.

friend Functions and friend Classes

- **`friend`** function and **`friend`** classes
  - Can access **`private`** and **`protected`** members of another class
  - **`friend`** functions are not member functions of class
    - Defined outside of class scope
- Properties of friendship
  - 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`**)

Friend Functions Friend Classes

- **`friend`** declarations
  - To declare a **`friend`** function
    - Type **`friend`** before the function prototype in the class that is giving friendship

```
friend int myFunction( int x );
```

should appear in the class giving friendship

  - To declare a **`friend`** class
  - Type **`friend class Classname`** in the class that is giving friendship
  - if **`ClassOne`** is granting friendship to **`ClassTwo`**,

```
friend class ClassTwo;
```

  - should appear in **`ClassOne`**'s definition

Freind Function Syntax

```
class className
{
… .. …
        friend return_type functionName(argument/s);
 … .. …
 }

return_type functionName(argument/s)
{
… .. …
     // Private and protected data of className can be
    accessed from // this function because it is a friend
    function of className
. … .. …
}
```

```cpp
// Fig. 7.6: fig07_06.cpp
// Non-friend/non-member functions cannot access
// private data of a class.
#include <iostream>

using std::cout;
using std::endl;

// Modified Count class
class Count {
public:
   Count() { x = 0; }                          // constructor
   void print() const { cout << x << endl; }  // output
private:
   int x;  // data member
};

// Function tries to modify private data of Count,
// but cannot because it is not a friend of Count.
void SetX( Count &c, int val )
{
   c.x = val;  // ERROR: 'Count::x' is not accessible
}

int main()
{
   Count counter;

   SetX( counter, 3 ); // cannotSetX is not a friend
   return 0;
}
```

`cannotSetX` is not a **`friend`** of class **Count**. It cannot access **`private`** data.

`cannotSetX` tries to modify a **`private`** variable…

```
Compiling...
Fig07_06.cpp
D:\books\2000\cpphtp3\examples\Ch07\Fig07_06\Fig07_06.cpp(22) :
   error C2248: 'x' : cannot access private member declared in
   class 'Count'
        D:\books\2000\cpphtp3\examples\Ch07\Fig07_06\
        Fig07_06.cpp(15) : see declaration of 'x'
Error executing cl.exe.

test.exe - 1 error(s), 0 warning(s)
```

Expected compiler error - cannot access **`private`** data

```cpp
// Fig. 7.5: fig07 05.cpp
// Friends can access private members of a class.
#include <iostream>

using std::cout;
using std::endl;

// Modified Count class
class Count {
public:
 Count() { x = 0; } // constructor
 friend void setX( Count &, int ); // friend declaration
   void print() const { cout << x << endl; }  // output
private:
   int x;  // data member
};

// Can modify private data of Count because
// setX is declared as a friend function of Count
void setX( Count &c, int val )
{
   c.x = val;  // legal: setX is a friend of Count
}

int main()
{
   Count counter;

   cout << "counter.x after instantiation: ";
   counter.print();
```

setX a **friend** of class **Count** (can access **private** data).

setX is defined normally and is not a member function of **Count**.

Changing **private** variables allowed.

```
31     cout << "counter.x after call to setX friend function: ";

32     setX( counter, 8 );  // set x with a friend

33     counter.print();

34     return 0;

35 }
```

```
counter.x after instantiation: 0
counter.x after call to setX friend function: 8
```

private data was changed.

```cpp
#include <iostream>
using namespace std;

// forward declaration
class B;
class A {
     private:
        int numA;
     public:
        A(): numA(12) { }
        // friend function declaration
        friend int add(A, B);
};

class B {
     private:
         int numB;
     public:
         B(): numB(1) { }
         // friend function declaration
         friend int add(A , B);
};

// Function add() is the friend function of classes A and B
// that accesses the member variables numA and numB
int add(A objectA, B objectB)
{
    return (objectA.numA + objectB.numB);
}
```

```cpp
int main()
{
     A objectA;
     B objectB;
     cout<<"Sum: "<< add(objectA, objectB);
     return 0;
}
```

Friend Classes

- To declare a **`friend`** class
- Type **`friend class Classname`** in the class that is giving friendship
- if **`ClassOne`** is granting friendship to **`ClassTwo`**,

```
friend class ClassTwo;
```

- should appear in **`ClassOne`**'s definition