

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)



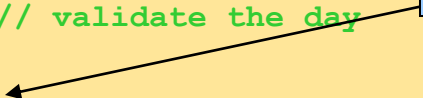
```
1 // Fig. 7.4: date1.h
2 // Declaration of the Date class.
3 // Member functions defined in date1.cpp
4 #ifndef DATE1_H
5 #define DATE1_H
6
7 class Date {
8 public:
9     Date( int = 1, int = 1, int = 1900 ); // default constructor
10    void print() const; // print date in month/day/year format
11    ~Date(); // provided to confirm destruction order
12 private:
13     int month; // 1-12
14     int day; // 1-31 based on month
15     int year; // any year
16
17    // utility function to test proper day for month and year
18    int checkDay( int );
19 };
20
21 #endif
```

```

22 // Fig. 7.4: date1.cpp
23 // Member function definitions for Date class.
24 #include <iostream>
25
26 using std::cout;
27 using std::endl;
28
29 #include "date1.h"
30
31 // Constructor: Confirm proper value for month;
32 // call utility function checkDay to confirm proper
33 // value for day.
34 Date::Date( int mn, int dy, int yr )
35 {
36     if ( mn > 0 && mn <= 12 )           // validate the month
37         month = mn;
38     else {
39         month = 1;
40         cout << "Month " << mn << " invalid. Set to month 1.\n";
41     }
42
43     year = yr;                          // should validate yr
44     day = checkDay( dy );               // validate the day
45
46     cout << "Date object constructor for date ";
47     print();                            // interesting: a print with no arguments
48     cout << endl;
49 }
50

```

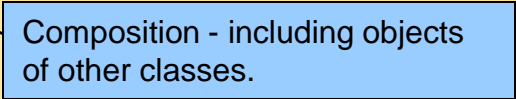
Constructor will print a line when called.



Destructor will print a line when called.

```
51 // Print Date object in form month/day/year
52 void Date::print() const
53     { cout << month << '/' << day << '/' << year; }
54
55 // Destructor: provided to confirm destr
56 Date::~Date()
57 {
58     cout << "Date object destructor for date ";
59     print();
60     cout << endl;
61 }
62
63 // Utility function to confirm proper day value
64 // based on month and year.
65 // Is the year 2000 a leap year?
66 int Date::checkDay( int testDay )
67 {
68     static const int daysPerMonth[ 13 ] =
69         {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
70
71     if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
72         return testDay;
73
74     if ( month == 2 && // February: Check for leap year
75         testDay == 29 &&
76         ( year % 400 == 0 ||
77           ( year % 4 == 0 && year % 100 != 0 ) ) )
78         return testDay;
79
80     cout << "Day " << testDay << " invalid. Set to day 1.\n";
81
82     return 1; // leave object in consistent state if bad value
83 }
```

```
84 // Fig. 7.4: employ1.h
85 // Declaration of the Employee class.
86 // Member functions defined in employ1.cpp
87 #ifndef EMPLOY1_H
88 #define EMPLOY1_H
89
90 #include "date1.h"
91
92 class Employee {
93 public:
94     Employee( char *, char *, int, int, int, int, int, int );
95     void print() const;
96     ~Employee(); // provided to confirm destruction order
97 private:
98     char firstName[ 25 ];
99     char lastName[ 25 ];
100     const Date birthDate;
101     const Date hireDate;
102 };
103
104 #endif
```




Composition - including objects
of other classes.

```

105// Fig. 7.4: emply1.cpp
106// Member function definitions for Employee class.
107#include <iostream>
108
109using std::cout;
110using std::endl;
111
112#include <cstring>
113#include "emply1.h"
114#include "date1.h"
115
116Employee::Employee( char *fname, char *lname,
117                   int bmonth, int bday, int byear,
118                   int hmonth, int hday, int hyear )
119    : birthDate( bmonth, bday, byear ),
120      hireDate( hmonth, hday, hyear )
121{
122    // copy fname into firstName and be sure that it fits
123    int length = strlen( fname );
124    length = ( length < 25 ? length : 24 );
125    strncpy( firstName, fname, length );
126    firstName[ length ] = '\0';
127
128    // copy lname into lastName and be sure that it fits
129    length = strlen( lname );
130    length = ( length < 25 ? length : 24 );
131    strncpy( lastName, lname, length );
132    lastName[ length ] = '\0';
133
134    cout << "Employee object constructor: "
135          << firstName << ' ' << lastName << endl;
136}

```

Constructor will print a line when called.



137

138void 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

148Employee::~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.


```

153// Fig. 7.4: fig07_04.cpp
154// Demonstrating composition: an object with member objects.
155#include <iostream>
156
157using std::cout;
158using std::endl;
159
160#include "empty1.h"
161
162int main()
163{
164    Employee e( "Bob", "Jones", 7, 24, 1949, 3, 12, 1988 );
165
166    cout << '\n';
167    e.print();
168
169    cout << "\nTest Date constructor with invalid values:\n";
170    Date d( 14, 35, 1994 ); // invalid Date values
171    cout << endl;
172    return 0;
173}

```

Only `empty.h` has to be loaded; that file has the command to load `date.h`.

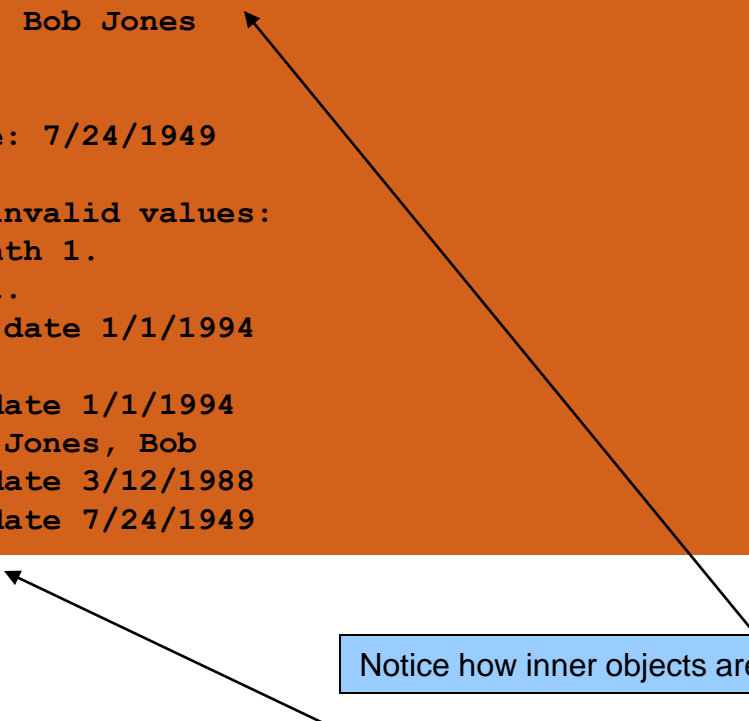


```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Employee object constructor: Bob Jones
```

```
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



FRIEND 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
    . ... ..
}
```



```

1 // Fig. 7.6: fig07_06.cpp
2 // Non-friend/non-member functions cannot access
3 // private data of a class.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 // Modified Count class
10 class Count {
11 public:
12     Count() { x = 0; } // constructor
13     void print() const { cout << x << endl; } // output
14 private:
15     int x; // data member
16 };
17
18 // Function tries to modify private data of Count,
19 // but cannot because it is not a friend of Count.
20 void SetX( Count &c, int val )
21 {
22     c.x = val; // ERROR: 'Count::x' is not accessible
23 }
24
25 int main()
26 {
27     Count counter;
28
29     SetX( counter, 3 ); // cannotSetX is not a friend
30     return 0;
31 }

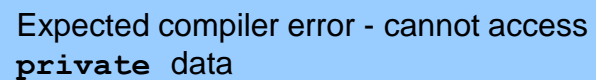
```

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\cpphttp3\examples\Ch07\Fig07_06\Fig07_06.cpp(22) :
    error C2248: 'x' : cannot access private member declared in
    class 'Count'
        D:\books\2000\cpphttp3\examples\Ch07\Fig07_06\
        Fig07_06.cpp(15) : see declaration of 'x'
Error executing cl.exe.

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



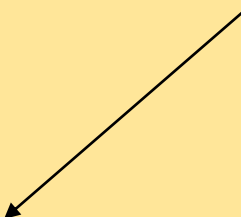
Expected compiler error - cannot access
private data

```

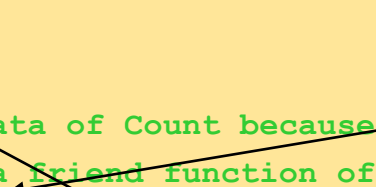
1 // Fig. 7.5: fig07_05.cpp
2 // Friends can access private members of a class.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 // Modified Count class
9 class Count {
10 public:
11     Count() { x = 0; } // constructor
12     friend void setX( Count &, int ); // friend declaration
13     void print() const { cout << x << endl; } // output
14 private:
15     int x; // data member
16 };
17
18 // Can modify private data of Count because
19 // setX is declared as a friend function of Count
20 void setX( Count &c, int val )
21 {
22     c.x = val; // legal: setX is a friend of Count
23 }
24
25 int main()
26 {
27     Count counter;
28
29     cout << "counter.x after instantiation: ";
30     counter.print();

```

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



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



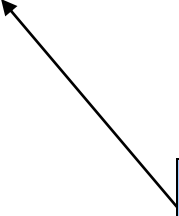
Changing private variables allowed.




```
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.



```
#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);
}
```

```
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

