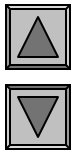
Lesson 2 - Outline

- **Separating Interface from Implementation**
- **Controlling Access to Members**
- **Access-Functions and Utility-Functions**
- **Constructors**
- **Using Member Functions**
- **Using Data Member**

Separating Interface from Implementation

- Makes it easier to modify programs
- Header files (*.h)
  - Contains class definitions and function prototypes
- Source-code files (*.cpp)
  - Contains member function definitions



Outline

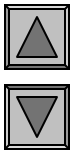
1. Using the same `Time` class as before, create a header file

```
1  // Fig. 6.5: time1.h
2  // Declaration of the Time class.
3  // Member functions are defined in time1.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME1_H
7  #define TIME1_H
8
9  // Time abstract data type definition
10 class Time {
11 public:
12    Time();                          // constructor
13    void setTime( int, int, int ); // set hour, minu
14    void printMilitary();          // print military
15    void printStandard();          // print standard time format
16 private:
17    int hour;     // 0 - 23
18    int minute;   // 0 - 59
19    int second;   // 0 - 59
20 };
21
22 #endif
```

Dot (**.**) replaced with underscore (**_**) in file name.

If **`time1.h`** (**`TIME1_H`**) is not defined (**`#ifndef`**) then it is loaded (**`#define TIME1_H`**). If **`TIME1_H`** *is* already defined, then everything up to **`#endif`** is ignored.

This prevents loading a header file multiple times.



Outline

2. Create a source code file

2.1 Load the header file to get the class definitions

2.2. Define the member functions of the class

```
// Fig. 6.5: time1.cpp
// Member function definitions for Time class.
#include <iostream>

using std::cout;

#include "time1.h"

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time() { hour = minute = second = 0; }

// Set a new Time value using military time. Perform validity
// checks on the data values. Set invalid values to zero.
void Time::setTime( int h, int m, int s )
{
   hour   = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
}

// Print Time in military format
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
}

// Print time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
}
```

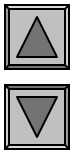
Source file uses **#include** to load the header file

Source file contains function definitions

Controlling Access to Members

- **`public`**
  - Presents clients with a view of the services the class provides (interface)
  - Data and member functions are accessible

- **`private`**
  - Default access mode
  - **`private`** members only accessible through the **`public`** class interface using **`public`** member functions



Outline

1. Load header file for `Time` class

2. Create an object of class `Time`

2.1 Attempt to set a `private` variable

2.2 Attempt to access a `private` variable

```
1  // Fig. 6.6: fig06_06.cpp
2  // Demonstrate errors resulting from attempts
3  // to access private class members.
4  #include <iostream>
5
6  using std::cout;
7
8  #include "time1.h"
9
10 int main()
11 {
12    Time t;
13
14    // Error: 'Time::hour' is not accessible
15    t.hour = 7;
16
17    // Error: 'Time::minute' is not accessible
18    cout << "minute = " << t.minute;
19
20    return 0;
21 }
```

Attempt to modify **`private`** member variable **`hour`**.

Attempt to access **`private`** member variable **`minute`**.

Program Output

```
Compiling...
Fig06_06.cpp
D:\Fig06_06.cpp(15) : error C2248: 'hour' : cannot access private
member declared in class 'Time'
D:\Fig6_06\time1.h(18) : see declaration of 'hour'
D:\Fig06_06.cpp(18) : error C2248: 'minute' : cannot access private
member declared in class 'Time'
D:\time1.h(19) : see declaration of 'minute'
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```

Access-Functions and Utility-Functions

- Utility-functions
  - **`private`** functions that support the operation of public functions
  - Not intended to be used directly by clients

- Access-functions
  - **`public`** functions that read/display data or check conditions
  - Allow **`public`** functions to check **`private`** data

- Following example
  - Program to take in monthly sales and output the total
  - Implementation not shown, only access functions

```
1.   // SalesPerson class definition
2.
3.   #ifndef SALESP_H
4.   #define SALESP_H
5.
6.   class SalesPerson
7.   {
8.      public:
9.         SalesPerson();                    // constructor
10.
11.        void getSalesFromUser();          // get sales figures from keyboard
12.
13.        void setSales( int, double );     // User supplies one month's
14.                                          // sales figures.
15.        void printAnnualSales();
16.
17.     private:
18.        double totalAnnualSales();        // utility function
19.
20.        double sales[ 12 ];               // 12 monthly sales figures
21.  };
22.
23.  #endif
```


```
// Fig. 6.7: fig06_07.cpp
// Demonstrating a utility function
// Compile with salesp.cpp
#include "salesp.h"

int main()
{
   SalesPerson s;          // create SalesPerson object s

   s.getSalesFromUser();  // note simple sequential code
   s.printAnnualSales();  // no control structures in main
   return 0;
}
```

Create object **s**, an instance of class **SalesPerson**

Outline

1. Load header file and compile with the file that contains the function definitions

2. Create an object

2.1 Use the object's member functions to get and print sales

```
 OUTPUT
Enter sales amount for month 1: 5314.76
Enter sales amount for month 2: 4292.38
Enter sales amount for month 3: 4589.83
Enter sales amount for month 4: 5534.03
Enter sales amount for month 5: 4376.34
Enter sales amount for month 6: 5698.45
Enter sales amount for month 7: 4439.22
Enter sales amount for month 8: 5893.57
Enter sales amount for month 9: 4909.67
Enter sales amount for month 10: 5123.45
Enter sales amount for month 11: 4024.97
Enter sales amount for month 12: 5923.92

The total annual sales are: $60120.59
```

Use access functions to gather and print data (**getSalesFromUser** and **printAnnualSales**). Utility functions actually calculate the total sales, but the user is not aware of these function calls.

Notice how simple **main()** is – there are no control structures, only function calls. This hides the implementation of the program.

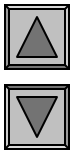
Constructors

- Initialize class members/objects
- Same name as the class
- No return type
- Member variables can be initialized by the constructor or set afterwards

- Passing arguments to a constructor
  - When an object of a class is declared, initializers can be provided
  - Format of declaration with initializers:

```
Class-type ObjectName( value1,value2,…);
```

  - Default arguments may also be specified in the constructor prototype

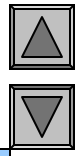


Outline

1. Define class `Time` and its default values

```
1  // Fig. 6.8: time2.h
2  // Declaration of the Time class.
3  // Member functions are defined in time2.cpp
4
5  // preprocessor directives that
6  // prevent multiple inclusions of header file
7  #ifndef TIME2_H
8  #define TIME2_H
9
10 // Time abstract data type definition
11 class Time {
12 public:
13    Time( int = 0, int = 0, int = 0 );  // default constructor
14    void setTime( int, int, int ); // set hour, minute, second
15    void printMilitary();          // print military time format
16    void printStandard();          // print standard time format
17 private:
18    int hour;     // 0 - 23
19    int minute;   // 0 - 59
20    int second;   // 0 - 59
21 };
22
23 #endif
```

Notice that default settings for the three member variables are set in constructor prototype. No names are needed; the defaults are applied in the order the member variables are declared.



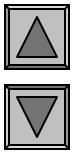
Outline

2. Create objects using default arguments

2.1 Print the objects

Notice how objects are initialized:
Constructor ObjectName`(`*value1*,*value2*…`);`
If not enough values are specified, the rightmost values are set to their defaults.

```
61 // Fig. 6.8: fig06_08.cpp
62 // Demonstrating a default constructor
63 // function for class Time.
64 #include <iostream>
65
66 using std::cout;
67 using std::endl;
68
69 #include "time2.h"
70
71 int main()
72 {
73    Time t1,             // all arguments defaulted
74         t2(2),          // minute and second defaulted
75         t3(21, 34),     // second defaulted
76         t4(12, 25, 42), // all values specified
77         t5(27, 74, 99); // all bad values specified
78
79    cout << "Constructed with:\n"
80         << "all arguments defaulted:\n   ";
81    t1.printMilitary();
82    cout << "\n   ";
83    t1.printStandard();
84
85    cout << "\nhour specified; minute and second defaulted:"
86         << "\n   ";
87    t2.printMilitary();
88    cout << "\n   ";
89    t2.printStandard();
90
91    cout << "\nhour and minute specified; second defaulted:"
92         << "\n   ";
93    t3.printMilitary();
```



Outline

2.1 (continued) Print the objects.

```
94    cout << "\n   ";
95    t3.printStandard();
96
97    cout << "\nhour, minute, and second specified:"
98         << "\n   ";
99    t4.printMilitary();
100   cout << "\n   ";
101   t4.printStandard();
102
103   cout << "\nall invalid values specified:"
104        << "\n   ";
105   t5.printMilitary();
106   cout << "\n   ";
107   t5.printStandard();
108   cout << endl;
109
110   return 0;
111}
```

Program Output

```
 OUTPUT
Constructed with:
all arguments defaulted:
   00:00
   12:00:00 AM
hour specified; minute and second defaulted:
   02:00
   2:00:00 AM
hour and minute specified; second defaulted:
   21:34
   9:34:00 PM
hour, minute, and second specified:
   12:25
   12:25:42 PM
all invalid values specified:
   00:00
   12:00:00 AM
```

When only **hour** is specified, **minute** and **second** are set to their default values of **0**.

Using Member Functions

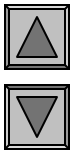
- Allow clients of the class to *set* (i.e., write) or *get* (i.e., read) the values of private data members
- Example:

  Adjusting a customer's bank balance

  - **`private`** data member **`balance`** of a class **`BankAccount`** could be modified through the use of member function **`computeInterest`**
  - A member function that sets data member **`interestRate`** could be called **`setInterestRate`**, and a member function that returns the **`interestRate`** could be called **`getInterestRate`**
- Providing *set* and *get* functions does not make **`private`** variables **`public`**
- A set function should ensure that the new value is valid

Using Data Member

- Reference to an object
  - Alias for the name of the object
  - May be used on the left side of an assignment statement
  - Reference can receive a value, which changes the original object as well
- Returning references: a subtle “trap”
  - **`public`** member functions can return non-**`const`** references to **`private`** data members
    - Should be avoided, breaks encapsulation



Outline

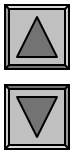
1. Define class

1.1 Function prototypes

1.2 Member variables

```
1  // Fig. 6.11: time4.h
2  // Declaration of the Time class.
3  // Member functions defined in time4.cpp
4
5  // preprocessor directives that
6  // prevent multiple inclusions of header file
7  #ifndef TIME4_H
8  #define TIME4_H
9
10 class Time {
11 public:
12    Time( int = 0, int = 0, int = 0 );
13    void setTime( int, int, int );
14    int getHour();
15    int &badSetHour( int );  // DANGEROUS reference return
16 private:
17    int hour;
18    int minute;
19    int second;
20 };
21
22 #endif
```

Notice how member function **badSetHour** returns a reference (**int &** is the return type).



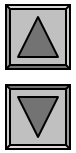
Outline

1. Load header

1.1 Function definitions

```
23 // Fig. 6.11: time4.cpp
24 // Member function definitions for Time class.
25 #include "time4.h"
26
27 // Constructor function to initialize private data.
28 // Calls member function setTime to set variables.
29 // Default values are 0 (see class definition).
30 Time::Time( int hr, int min, int sec )
31    { setTime( hr, min, sec ); }
32
33 // Set the values of hour, minute, and second.
34 void Time::setTime( int h, int m, int s )
35 {
36    hour   = ( h >= 0 && h < 24 ) ? h : 0;
37    minute = ( m >= 0 && m < 60 ) ? m : 0;
38    second = ( s >= 0 && s < 60 ) ? s : 0;
39 }
40
41 // Get the hour value
42 int Time::getHour() { return hour; }
43
44 // POOR PROGRAMMING PRACTICE:
45 // Returning a reference to a private data member.
46 int &Time::badSetHour( int hh )
47 {
48    hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
49
50    return hour;  // DANGEROUS reference return
51 }
```

`badSetHour` returns a reference to the **`private`** member variable **`hour`**. Changing this reference will alter **`hour`** as well.



Outline

1.2 Declare reference

2. Change data using a reference

3. Output results

```
52 // Fig. 6.11: fig06_11.cpp
53 // Demonstrating a public member function that
54 // returns a reference to a private data member.
55 // Time class has been trimmed for this example.
56 #include <iostream>
57
58 using std::cout;
59 using std::endl;
60
61 #include "time4.h"
62
63 int main()
64 {
65    Time t;
66    int &hourRef = t.badSetHour( 20 );
67
68    cout << "Hour before modification: " << hourRef;
69    hourRef = 30;  // modification with invalid value
70    cout << "\nHour after modification: " << t.getHour();
71
72    // Dangerous: Function call that returns
73    // a reference can be used as an lvalue!
74    t.badSetHour(12) = 74;
75    cout << "\n\n*********************************\n"
76         << "POOR PROGRAMMING PRACTICE!!!!!!!\n"
77         << "badSetHour as an lvalue, Hour: "
78         << t.getHour()
79         << "\n*********************************" << endl;
80
81    return 0;
82 }
```

Declare **Time** object **t** and reference **hourRef** that is assigned the reference returned by the call **t.badSetHour(20)**.

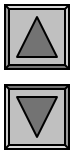
```
Hour before modification: 20
```

Alias used to set the value of **hour** to 30 (an invalid value).

```
Hour after modification: 30
```

Function call used as an *lvalue* and assigned the value **74** (another invalid value).

```
*********************************
POOR PROGRAMMING PRACTICE!!!!!!!
badSetHour as an lvalue, Hour: 74
*********************************
```



Outline

Program Output

```
Hour before modification: 20
Hour after modification: 30

*********************************
POOR PROGRAMMING PRACTICE!!!!!!!
badSetHour as an lvalue, Hour: 74
*********************************
```

`HourRef` used to change **`hour`** to an invalid value. Normally, the function **`setbadSetHour`** would not have allowed this. However, because it returned a reference, **`hour`** was changed directly.