6.1 Introduction

- Object-oriented programming (OOP)
  - Encapsulates data (attributes) and functions (behavior) into packages called classes

- Information hiding
  - Implementation details are hidden within the classes themselves

- Classes
  - Classes are the standard unit of programming
  - A class is like a blueprint – reusable
  - Objects are instantiated (created) from the class
  - For example, a house is an instance of a “blueprint class”
6.2 Structure Definitions

• Structures
  – Aggregate data types built using elements of other types

```c
struct Time {
    int hour;
    int minute;
    int second;
};
```
  – Members of the same structure must have unique names
  – Two different structures may contain members of the same name
  – Each structure definition must end with a semicolon

• Self-referential structure
  – Contains a member that is a pointer to the same structure type
  – Used for linked lists, queues, stacks and trees

• `struct`
  – Creates a new data type that is used to declare variables
  – Structure variables are declared like variables of other types
  – Example:

```c
Time timeObject, timeArray[10],
    *timePtr, &timeRef = timeObject;
```
6.3 Accessing Members of Structures

- Member access operators:
  - Dot operator (.) for structures and objects
  - Arrow operator (->) for pointers
  - Print member hour of timeObject:
    
    ```
    cout << timeObject.hour;
    ```
    OR
    
    ```
    timePtr = &timeObject;
    cout << timePtr->hour;
    ```
  - `timePtr->hour` is the same as `( *timePtr ).hour`
  - Parentheses required: `*` has lower precedence than `. `

```cpp
// Fig. 6.1: fig06_01.cpp
// Create a structure, set its members, and print it.

#include <iostream>

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

struct Time {    // structure definition
    int hour;     // 0-23
    int minute;   // 0-59
    int second;   // 0-59
};

void printMilitary( const Time & );  // prototype
void printStandard( const Time & );  // prototype

int main()
{
    Time dinnerTime;    // variable of new type Time
    // set members to valid values
    dinnerTime.hour = 18;
    dinnerTime.minute = 30;
    dinnerTime.second = 0;
    
    cout << "Dinner will be held at ";
    printMilitary( dinnerTime );
    cout << " military time, \nwhich is ";
    printStandard( dinnerTime );
    cout << " standard time.\n";
}
```

Dinner will be held at 18:30 military time, which is 6:30:00 PM standard time.

Creates the user-defined structure type Time with three integer members: hour, minute and second.
2.2 Set the time to an invalid hour, then print it

3. Define the functions printMilitary and printStandard

Dinner will be held at 18:30 military time, which is 6:30:00 PM standard time.

Time with invalid values: 29:73
6.5 Implementing a Time Abstract Data Type with a Class

- Classes
  - Model objects that have attributes (data members) and behaviors (member functions)
  - Defined using keyword `class`
  - Have a body delineated with braces (`{` and `}`)
  - Class definitions terminate with a semicolon
  - Example:

```cpp
class Time {
public:
    Time();
    void setTime(int, int, int);
    void printMilitary();
    void printStandard();

private:
    int hour;     // 0 - 23
    int minute;   // 0 - 59
    int second;   // 0 - 59
};
```

- Member access specifiers
  - Classes can limit the access to their member functions and data
  - The three types of access a class can grant are:
    - **Public** — Accessible wherever the program has access to an object of the class
    - **Private** — Accessible only to member functions of the class
    - **Protected** — Similar to private and discussed later

- Constructor
  - Special member function that initializes the data members of a class object
  - Cannot return values
  - Have the same name as the class
6.5 Implementing a Time Abstract Data Type with a Class

- Class definition and declaration
  - Once a class has been defined, it can be used as a type in object, array and pointer declarations
  - Example:

```cpp
Time sunset,                  // object of type Time
arrayOfTimes[5],       // array of Time objects
*pointerToTime,          // pointer to a Time object
&dinnerTime = sunset;    // reference to a Time object
```

Note: The class name becomes the new type specifier.

```
1 // Fig. 6.3: fig06_03.cpp
2 // Time class.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 // Time abstract data type (ADT) definition
8 class Time {
9 public:
10    Time();                        // constructor
11    void setTime( int, int, int ); // set hour, minute, second
12    void printMilitary();          // print military time format
13    void printStandard();          // print standard time format
14 private:
15    int hour;     // 0 – 23
16    int minute;   // 0 – 59
17    int second;   // 0 – 59
18   
19   }
20
21 // Time constructor initializes each data member to zero.
22 // Ensures all Time objects start in a consistent state.
23 Time::Time() { hour = minute = second = 0; }
24
25 // Set a new Time value using military time. Perform validity
26 // checks on the data values. Set invalid values to zero.
27 void Time::setTime( int h, int m, int s )
28 {
29    hour = ( h >= 0 && h < 24 ) ? h : 0;
30    minute = ( m >= 0 && m < 60 ) ? m : 0;
31    second = ( s >= 0 && s < 60 ) ? s : 0;
32 }
```

Note the :: preceding the function names.
```cpp
// 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" );
}

// Driver to test simple class Time
int main()
{
    Time t;  // instantiate object t of class Time

cout << "The initial military time is ";
t.printMilitary();
cout << "The initial standard time is ";
t.printStandard();

    t.setTime( 13, 27, 6 );
    cout << "Military time after setTime is ";
t.printMilitary();
    cout << "Standard time after setTime is ";
t.printStandard();

t.setTime( 99, 99, 99 );  // attempt invalid settings
    cout << "After attempting invalid settings:
    cout << "Military time: ";
t.printMilitary();
    cout << "Standard time: ";
t.printStandard();
cout << endl;
    return 0;
}
```

The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM

Notice how functions are called using the dot (.) operator.
6.5 Implementing a Time Abstract Data Type with a Class

• Destructors
  – Functions with the same name as the class but preceded with a tilde character (~)
  – Cannot take arguments and cannot be overloaded
  – Performs “termination housekeeping”

• Binary scope resolution operator (::)
  – Combines the class name with the member function name
  – Different classes can have member functions with the same name

• Format for defining member functions
  
  ```cpp
  ReturnType ClassName::MemberFunctionName() {
  ...
  }
  ```

• If a member function is defined inside the class
  – Scope resolution operator and class name are not needed
  – Defining a function outside a class does not change it being `public` or `private`

• Classes encourage software reuse
  – Inheritance allows new classes to be derived from old ones
6.6 Class Scope and Accessing Class Members

• Class scope
  – Data members and member functions

• File scope
  – Non member functions

• Inside a scope
  – Members accessible by all member functions
    • Referenced by name

• Outside a scope
  – Members are referenced through handles
    • An object name, a reference to an object or a pointer to an object

• Function scope
  – Variables only known to function they are defined in
  – Variables are destroyed after function completion

• Accessing class members
  – Same as structs
  – Dot (.) for objects and arrow (->) for pointers
  – Example:
    • \texttt{t.hour} is the \texttt{hour} element of \texttt{t}
    • \texttt{TimePtr->hour} is the \texttt{hour} element
// Fig. 6.4: fig06_04.cpp
// Demonstrating the class member access operators . and ->
// CAUTION: IN FUTURE EXAMPLES WE AVOID PUBLIC DATA!

#include <iostream>

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

// Simple class Count
class Count {
public:
    int x;
    void print() { cout << x << endl; }
};

int main() {
    Count counter,    // create counter object
        *counterPtr = &counter, // pointer to counter
        &counterRef = counter;  // reference to counter

cout << "Assign 7 to x and print using the object's name: ";
counter.x = 7;   // assign 7 to data member x
counter.print(); // call member function print

cout << "Assign 8 to x and print using a reference: ";
counterRef.x = 8; // assign 8 to data member x
counterRef.print(); // call member function print

cout << "Assign 10 to x and print using a pointer: ";
counterPtr->x = 10;  // assign 10 to data member x
counterPtr->print(); // call member function print

    return 0;
}

Assign 7 to x and print using the object's name: 7
Assign 8 to x and print using a reference: 8
Assign 10 to x and print using a pointer: 10

It is rare to have **public** member variables. Usually only member functions are **public**; this keeps as much information hidden as possible.
6.7 Separating Interface from Implementation

- Separating interface from implementation
  - Makes it easier to modify programs
  - Header files
    - Contains class definitions and function prototypes
  - Source-code files
    - Contains member function definitions

```cpp
// Fig. 6.5: time1.h
// Declaration of the Time class.
// Member functions are defined in time1.cpp

// prevent multiple inclusions of header file
#ifndef TIME1_H
#define TIME1_H

// Time abstract data type definition
class Time {
    public:
        Time();                        // constructor
        void setTime( int, int, int ); // set hour, minute, second
        void printMilitary();          // print military time format
        void printStandard();          // print standard time format

    private:
        int hour;     // 0 - 23
        int minute;   // 0 - 59
        int second;   // 0 - 59

};
#endif
```

Outline

1. Using the same Time class as before.

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.
6.8 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
  - Data only accessible to member functions and friends
  - **private** members only accessible through the **public** class interface using **public** member functions
### 6.9 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
6.10 Initializing Class Objects: Constructors

- **Constructors**
  - Initialize class members
  - 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

```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;
}
```

**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

Create object `s`, an instance of class `SalesPerson`.
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    int hour;     // 0 - 23
22    int minute;   // 0 - 59
23    int second;   // 0 - 59
24
25 #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.

61 // Fig. 6.8: fig06_08.cpp
62 // Demonstrating a default constructor
63 // Function for class Time:
64 #include <iostream>
65 using std::cout;
66 using std::endl;
67
68 #include "time2.h"
69
70 int main()
71 {    // all arguments defaulted
72    Time t1,
73        t2(2),          // minute and second defaulted
74        t3(21, 34),    // second defaulted
75        t4(12, 25, 42), // all values specified
76        t5(27, 74, 99); // all bad values specified
77    cout << "Constructed with:\n"
78        << "all arguments defaulted:\n   ";
79    t1.printMilitary();
80    cout << "\n ";
81    t1.printStandard();
82
83    cout << "\nhour specified; minute and second defaulted:";
84    << "\n ";
85    t2.printMilitary();
86    cout << "\n ";
87    t2.printStandard();
88    cout << "\nhour and minute specified; second defaulted:";
89    << "\n ";
90    t3.printMilitary();
91
Notice how objects are initialized:
ConstructorObjectName (value1,value2...);
If not enough values are specified, the rightmost values are set to their defaults.
6.12 Using Destructors

- Destructors
  - Are member function of class
  - Perform termination housekeeping before the system reclaims the object’s memory
  - Complement of the constructor
  - Name is tilde (~) followed by the class name (i.e., ~Time)
    - Recall that the constructor’s name is the class name
  - Receives no parameters, returns no value
  - One destructor per class
    - No overloading allowed
6.13 When Constructors and Destructors Are Called

- Constructors and destructors called automatically
  - Order depends on scope of objects

- Global scope objects
  - Constructors called before any other function (including `main`)
  - Destructors called when `main` terminates (or `exit` function called)
  - Destructors not called if program terminates with `abort`

- Automatic local objects
  - Constructors called when objects are defined
  - Destructors called when objects leave scope
    - i.e., when the block in which they are defined is exited
  - Destructors not called if the program ends with `exit` or `abort`

- Static local objects
  - Constructors called when execution reaches the point where the objects are defined
  - Destructors called when `main` terminates or the `exit` function is called
  - Destructors not called if the program ends with `abort`
1 // Fig. 6.9: create.h
2 // Definition of class CreateAndDestroy.
3 // Member functions defined in create.cpp.
4 #ifndef CREATE_H
5 #define CREATE_H
6
7 class CreateAndDestroy {
8 public:
9    CreateAndDestroy( int );  // constructor
10    ~CreateAndDestroy();      // destructor
11 private:
12    int data;
13};
14
15 #endif

16 // Fig. 6.9: create.cpp
17 // Member function definitions for class CreateAndDestroy
18 #include <iostream>
19
20 using std::cout;
21 using std::endl;
22
23 #include "create.h"
24
25 CreateAndDestroy::CreateAndDestroy( int value )
26 {
27    data = value;
28    cout << "Object " << data << "   constructor";
29 }
30
31 CreateAndDestroy::~CreateAndDestroy()
32 { cout << "Object " << data << "   destructor" << endl; }

1. Create a header file
1.1 Include function prototypes for the destructor and constructor

2. Load the header file
2.1 Modify the constructor and destructor

Constructor and Destructor changed to print when they are called.

© 2000 Prentice Hall, Inc. All rights reserved.
33 // Fig. 6.9: fig06_09.cpp
34 // Demonstrating the order in which constructors and
destructors are called.
35 #include <iostream>
36
37 using std::cout;
38 using std::endl;
39
40 // include "create.h"
41
42 void create( void );  // prototype
43
44 CreateAndDestroy first( 1 ); // global object
45
46 int main()
47 {
48    cout << " (global created before main)" << endl;
49
50    CreateAndDestroy second( 2 );  // local object
51    cout << " (local automatic in main)" << endl;
52
53    static CreateAndDestroy third( 3 );  // local object
54    cout << " (local static in main)" << endl;
55
56    create(); // call function to create objects
57
58    CreateAndDestroy fourth( 4 );  // local object
59    cout << " (local automatic in main)" << endl;
60    return 0;
61
62 }
63
64 // Function to create objects
65 void create( void )
66 {
67    CreateAndDestroy fifth( 5 );
68    cout << " (local automatic in create)" << endl;
69
70    static CreateAndDestroy sixth( 6 );
71    cout << " (local static in create)" << endl;
72
73    CreateAndDestroy seventh( 7 );
74    cout << " (local automatic in create)" << endl;
75 }

OUTPUT

Object 1 constructor (global created before main)
Object 2 constructor (local automatic in main)
Object 3 constructor (local static in main)
Object 5 constructor (local automatic in create)
Object 6 constructor (local static in create)
Object 7 constructor (local automatic in create)
Object 7 destructor
Object 5 destructor
Object 4 destructor
Object 4 destructor
Object 2 destructor
Object 6 destructor
Object 3 destructor
Object 1 destructor

Program Output

Notice how the order of the constructor and destructor call depends on the types of variables (automatic, global and static) they are associated with.
6.14 Using Data Members and Member Functions

- 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

6.15 A Subtle Trap: Returning a Reference to a Private 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
  - public member functions can return non-const references to private data members
    • Should be avoided, breaks encapsulation
// Fig. 6.11: time4.h
// Declaration of the Time class.
// Member functions defined in time4.cpp

// Preprocessor directives that prevent multiple inclusions of header file
#ifndef TIME4_H
#define TIME4_H

class Time {
public:
    Time( int = 0, int = 0, int = 0 );
    void setTime( int, int, int );
    int getHour();
    int &badSetHour( int ); // DANGEROUS reference return
private:
    int hour;
    int minute;
    int second;
};
#endif

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

#include "time4.h"

// Constructor function to initialize private data.
// Calls member function setTime to set variables.
// Default values are 0 (see class definition).
Time::Time( int hr, int min, int sec )
{ setTime( hr, min, sec ); }

// Set the values of hour, minute, and second.
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;
}

// Get the hour value
int Time::getHour() { return hour; }

// POOR PROGRAMMING PRACTICE:
// Returning a reference to a private data member.
int &Time::badSetHour( int hh )
{
    hour = ( hh >= 0 && hh < 24 ) ? hh : 0;

    return hour; // DANGEROUS reference return
}
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 #include "time4.h"
58 using std::cout;
59 using std::endl;
60 int main()
61 {
62   Time t;
63   int &hourRef = t.badSetHour(20);
64   cout << "Hour before modification: " << hourRef;
65   hourRef = 30; // modification with invalid value
66   cout << "Hour after modification: " << t.getHour() << endl;
67)
68   // Dangerous: Function call that returns
69   // a reference can be used as an lvalue!
70   t.badSetHour(12) = 74;
71   cout << "POOR PROGRAMMING PRACTICE!!!!!!!\n"
72   << "badSetHour as an lvalue, Hour: " << t.getHour() << endl;
73   return 0;
74 }
```

Declare **Time** object `t` and reference `hourRef` that is assigned the reference returned by the call `t.badSetHour(20)`. Alias used to set the value of `hour` to 30 (an invalid value).

```
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.

```
*****************************************
POOR PROGRAMMING PRACTICE!!!!!!!
badSetHour as an lvalue, Hour: 74
*****************************************
```
6.16 Assignment by Default Memberwise Copy

• Assigning objects
  – An object can be assigned to another object of the same type using the assignment operator ( = )
  – Member by member copy

• Objects may be
  – Passed as function arguments
  – Returned from functions (call-by-value default)

```cpp
1 // Fig. 6.12: fig06_12.cpp
2 // Demonstrating that class objects can be assigned
3 // to each other using default memberwise copy
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 // Simple Date class
9 class Date {
10 public:
11    Date( int = 1, int = 1, int = 1990 ); // default constructor
12    void print();
13 private:
14    int month;
15    int day;
16    int year;
17 };
18
19 // Simple Date constructor with no range checking
20 Date::Date( int m, int d, int y )
21    { month = m;
22      day = d;
23      year = y;
24    }
25
26 // Print the Date in the form mm-dd-yyyy
27 void Date::print()
28    { cout << month << '-' << day << '-' << year; }
```
### 6.17 Software Reusability

- Software resusability
  - Implementation of useful classes
  - Class libraries exist to promote reusability
    - Allows for construction of programs from existing, well-defined, carefully tested, well-documented, portable, widely available components
  - Speeds development of powerful, high-quality software