CSE 230 Intermediate Programming in C and C++

Classes, Objects and Strings

Fall 2017

Stony Brook University

Instructor: Shebuti Rayana

http://www3.cs.stonybrook.edu/~cse230/

Ref. Book: C How to Program, 8th edition by Deitel and Deitel

Object Oriented Programming: Classes

- Classes encapsulate data (attribute) and functions (behavior); the data and functions of a class are intimately tied together.
- A class can be reused many times to make many objects of the same class.
- Class objects communicate with one another with well-defined *member functions*, and their *data members* are hidden within themselves.
- The unit of OOP is the class from which objects are eventually instantiated.
- Groups of actions that perform some task are formed into these member functions.

Object Oriented Programming: Classes

- The focus of attention in OOP is on classes rather than functions.
- In C, a `struct` is a collection of related variables (data), whereas in C++, a class contains *data members* and *member functions.*
  - In the C++ community, the terms *data members* and *member functions* refer to instance variables and methods respectively.
- The data members keep the current state of an object, and member functions allow a user of the object to query the object (find out its state) or modify the object (alter its state).
- In C++, a class definition is considered as a user-defined (or programmer-defined) type.

Classes in C++

- A class definition begins with the keyword **`class`** and terminates with a semicolon (**;**). It is considered as a user-defined type.
  - Class name may differ from the filename.
- The **`public:`** and **`private:`** are called member access specifier.
- Any member defined after **`public`** (and before the next access specifiers) is accessible wherever the program has access to the object of that class.
- Any member defined after **`private`** (and up to the next access specifiers) is accessible only to member functions of the class. Also, default mode in a class is **`private`**.
- Access specifiers are always followed by a colon(**:**), and can appear multiple times in any order in class.

Example:

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

// definition of class Square
class Square
{
        private:
                int len;          // restricted access
         public:                  // universal access
                Square() { }    // can be omitted
                int getLength()  { return len; }
                int  getArea() { return len*len ; }
                 void setLength(int L)  { len = L; }
};       // end class Square

int main(void)
{
        int side;
        cout << " Enter the side of the square: ";
        cin  >> side;
        Square s;             // instantiate object s of class Square
        s.setLength(side);
        cout << "The area of the square of length "
                        <<  s.getLength() << " is "
                        <<  s.getArea()    << endl;
        return 0;
}
```

Separating the Interface

- Declaring member functions inside a class (via their prototypes) and defining those members outside the class separates the interface of a class from its implementation.
- This promotes good software engineering.

Example:

```
// Time abstract data type (ADT)
#include <iostream>
using namespace std;

class Time {
        public:
                Time();                                  // constructor
                void setTime(int h, int m, int s) ;  // 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
}; // end class Time (interface)
```

Defining the Implementation

- Member functions may be defined in the same file as the class definition.
- Function definition must be preceded by the class name followed by the scope resolution operator (::).
  - Informs the compiler in what class this member function belongs to

Implementation

```
//Time constructor:Ensures objects start in a consistent state.
Time::Time( ) { hour = minute = second = 0; }

void Time::setTime( int h, int m, int s ) //use military format
{
   hour   = ( h >= 0  && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60) ? m : 0;
   second = ( s >= 0  &&  s < 60 ) ? s : 0;
} // end setTime

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

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

Class Type

- Once the class is defined, it can be used as a type in object, array and pointer definitions as follows:
- Time sunset; // object of class Time
- Time timeArray[10]; // array of Time objects
- Time *pointerToTime; // pointer to a Time object
- Time &refTime = sunset;// reference to a Time object

```
int main()
{
   Time t;  // instantiate object t of class Time
   cout << "The initial military time is ";
   t.printMilitary();
   cout << "\nThe initial standard time is ";
   t.printStandard();
   t.setTime( 13, 27, 6 );
   cout << "\n\nMilitary time after setTime is ";
   t.printMilitary();
   cout << "\nStandard time after setTime is ";
   t.printStandard();
       t.setTime( 99, 99, 99 );  // attempt invalid settings
   cout << "\n\nAfter attempting invalid settings:"
       << "\nMilitary time: ";
   t.printMilitary();
   cout << "\nStandard time: ";
   t.printStandard();
   cout << endl;
   return 0;
} // end function main
```

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

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

The initial military time is 13:27
The initial standard time is 1:27:06 PM
```

Creating Header Files

- Each class definition is normally placed in a *header (.h) file,* and function definitions are placed in *source-code (.cpp) file* of the same base name.
- The header files are included in each file the class is used.
- The source-code (.cpp) file is eventually compiled and linked with the main program. Also include the .cpp files in each file the class is used.

time.h

```
#ifndef TIME_H // Prevent multiple inclusion of header
#define TIME_H
class Time {
public:
   Time( );                            // constructor
   void setTime(int h, int m, int s); // 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
}; // end class Time (interface)
#endif
```

time.cpp

```
#include  <iostream>
#include  "time.h"
using namespace std;

Time::Time( ) { hour = minute = second = 0; }
void Time::setTime( int h, int m, int s )//use military format
{
   hour     = ( h >= 0  && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60) ? m : 0;
   second = ( s >= 0  &&  s < 60 ) ? s : 0;
} // end setTime

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

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

test.cpp

```
#include  <iostream>
#include  "time.h"
#include  "time.cpp"
using namespace std;

int main()
{
   Time t;  // instantiate object t of class Time
   cout << "The initial military time is ";
   t.printMilitary();
   cout << "\nThe initial standard time is ";
   t.printStandard();
   t.setTime( 13, 27, 6 );
   cout << "\n\nMilitary time after setTime is ";
   t.printMilitary();
   cout << "\nStandard time after setTime is ";
   t.printStandard();
       t.setTime( 99, 99, 99 );  // attempt invalid settings
   cout << "\n\nAfter attempting invalid settings:"
       << "\nMilitary time: ";
   t.printMilitary();
   cout << "\nStandard time: ";
   t.printStandard();
   cout << endl;
   return 0;
} // end function main
```

Default Arguments with Constructors

- Constructors can contain default arguments.
- By providing default arguments to the constructor, even if no values are provided in a constructor call, the object is still guaranteed to be initialized to a consistent state.
- Parameter names can be omitted as usual, i.e. the type and its corresponding value separated by equal sign is sufficient. For example:

```
Time( int = 0, int = 0, int = 0);
```

Destructors

- A destructor is a special member function of a class.
- The name of the destructor is the tilde (~) character followed by the class name.
- In a sense, destructor is a complement of constructor.
- Destructor is called when an object is destroyed. For automatic objects, when program execution leaves the scope in which an object was instantiated.
- Destructors perform “termination housekeeping”, not actually release the objects memory.

Calling Constructors and Destructors

- Constructors and destructors are called automatically.
- The order depends on the order in which execution enters and leaves the scope in which objects are instantiated.
- Destructor calls are made in the reverse order of the constructor calls. However, the storage class of objects can alter the order in which destructors are called.
- For global objects, constructors are called before any other objects, and corresponding destructors are called when **main** terminates normally or **exit** is called.
- For automatic local objects, constructors are called when the execution reaches the point where object is defined. Destructors are called when objects leave scope normally.
- For static objects, constructors are called only once, and corresponding destructor is called after **main** terminates.

Assignment of Objects

- The assignment operator (=) can be used to assign an object to another object of the same type.
- Assigning objects is by default performed by *memberwise copy* – each member of one object is copied individually to the same member in another object.
- Memberwise copy can cause serious problems when used with a class whose data members contain dynamically allocated storage.

Using Data Members

- A class's **private** data members can be accessed only by member functions(and **friend**s).
- Classes often provide **public** member functions to allow clients of the class to set (i.e. write) or get (i.e. read) the values of **private** data members.

A Subtle Trap: reference

- A reference to a an object is an alias for the name of the object and hence may be used as a *object*.
- It is possible that a **public** member function of a class return a non-**const** reference to a **private** data member of that class.