C++ is fun – Part Four

at Turbine/Warner Bros.!

Russell Hanson
Brief Overview

• We will do some lab examples using simple classes with private data and access functions like get() and set().

• Opening and reading Excel files and using/installing external libraries/API’s

• Introduction to typedef, struct, and enum

• Some first gameplay classes
Typedef, struct, and enum

3 Ways to declare struct student:
1. struct student {
   char id_num[5];
   char name[10];
   char gender;
   int age;
} studno_1, studno_2;
2. struct { // no tag
   char id_num[5];
   char name[10];
   char gender;
   int age;
} studno_1, studno_2;
3. struct student{
   char id_num[5];
   char name[10];
   char gender;
   int age;
};

Here, struct is a keyword that tells the compiler that a structure template is being declared and student is a tag that identifies its data structure. Tag is not a variable; it is a label for the structure’s template. Note that there is a semicolon after the closing curly brace.

In the program declare as:
struct student studno_1, studno_2;

In (2) there is no structure tag, this means we cannot declare structure variables of this type elsewhere in the program instead we have to use the structure variables studno_1 and studno_2 directly.
The **structure pointer operator** (→), consisting of a minus (−) sign and a greater than (>) sign with no intervening spaces, accesses a structure member via a **pointer to the structure**.

```c
// The packet length
#define PCKT_LEN 8192

// Can create separate header file (.h) for all headers' structure
// The IP header's structure
struct ipheader {
    unsigned char    iph_ihl:5, iph_ver:4;
    unsigned char    iph_tos;
    unsigned short int iph_len;
    unsigned short int iph_id;
    unsigned char    iph_flag;
    unsigned short int iph_offset;
    unsigned char    iph_ttl;
    unsigned char    iph_proto;
    unsigned short int iph_chcksum;
    unsigned int     iph_sourceip;
    unsigned int     iph_destip;
};

// UDP header's structure
struct udphheader {
    unsigned short int udph_srcport;
    unsigned short int udph_destport;
    unsigned short int udph_len;
    unsigned short int udph_chcksum;
};

// total udph header length: 8 bytes
```

```c
// Initialize struct
// Use the (->) structure pointer operator
struct ipheader *ip = (struct ipheader *) buffer;
struct udphheader *udp = (struct udphheader *) (buffer + sizeof(struct ipheader));

// Fabricate the IP header or we can use the standard header structures but assign our own values.
ip->iph_ihl = 5;
ip->iph_ver = 4;
ip->iph_tos = 16; // Low delay
ip->iph_len = sizeof(struct ipheader) + sizeof(struct udphheader);
ip->iph_ident = htons(54321);
ip->iph_ttl = 64; // hops
ip->iph_proto = 17; // UDP

// Source IP address, can use spoofed address here!!!
ip->iph_sourceip = inet_addr(argv[1]);
// The destination IP address
ip->iph_destip = inet_addr(argv[3]);

// Fabricate the UDP header. Source port number, redundant
udp->udph_srcport = htons(atoi(argv[2]));

// Destination port number
udp->udph_destport = htons(atoi(argv[4]));
udp->udph_len = htons(sizeof(struct udphheader));
```
In class example! Typing is good for character(s)

//accessing structure element
#include <iostream.h>
#include <stdlib.h>
struct Card
{
    char *face; //pointer to char type
    char *suit; //pointer to char type
};
int main()
{
    //declare the struct type variables
    struct Card p;
    struct Card *SPtr;
    p.face = "Ace";
    p.suit = "Spades";
    SPtr = &p;
    cout<<"Accessing structure element:\n";
    cout<<"\n\nSPtr->suit\' = "<<SPtr->suit<<endl;
    cout<<"\nSPtr->face\' = "<<SPtr->face<<endl;

    // cout<<"\n\nSPtr->suit\' = "<<p->suit<<endl; // ERROR
    // cout<<"\n\nSPtr->face\' = "<<p->face<<endl; // ERROR

    // structtest.cpp:23: error: base operand of ‘->’ has non-pointer type ‘Card’
    // structtest.cpp:24: error: base operand of ‘->’ has non-pointer type ‘Card’
}

// Running the program:
// ./structtest
// Accessing structure element:

// 'SPtr->suit' = Spades
// 'SPtr->face' = Ace
typedef type-declaration the_synonym;

You cannot use the typedef specifier inside a function definition. When declaring a local-scope identifier by the same name as a typedef, or when declaring a member of a structure or union in the same scope or in an inner scope, the type specifier must be specified. For example:

    typedef float TestType;

    int main()
    {
    
    //function scope...
    int MyFunct(int)
    {
        //same name with typedef, it is OK
        int TestType;
    }
Names for structure types are often defined with `typedef` to create shorter and simpler type names. For example, the following statements:

```c
typedef struct Card Card;
typedef unsigned short USHORT;
```

Defines the new type name `Card` as a synonym for type `struct Card` and `USHORT` as a synonym for type `unsigned short`. Programmers usually use `typedef` to define a structure type so a structure tag is not required. For example, the following definition:

```c
typedef struct{
    char *face;
    char *suit;
} Card;
```

Creates the structure type `Card` without the need for a separate `typedef` statement. Then `Card` can now be used to declare variables of type `struct Card`. For example, the following declaration:

```c
Card deck[50];
```

Declares an array of 50 `Card` structures. `typedef` simply creates a new type name which may be used as an alias for an existing type name.
// typedef specifier
#include <stdio.h>

typedef struct mystructtag
{
    int   x;
    double y;
    char* z;
} mystruct;

int main()
{
    mystruct Test1, *Test2;
    Test1.x = 111;
    Test1.y = 1.111;
    printf("Test1.x = %d\nTest1.y = %f\n", Test1.x, Test1.y);

    Test1.z = "This is a string";
    Test2 = &Test1;
    printf("Test1->z = %s\n", Test2->z);
    return 0;
}

$ ./structexample
Test1.x = 111
Test1.y = 1.111000
Test1->z = This is a string
//accessing structure element
#include <iostream.h>
#include <stdlib.h>
typedef struct Card
{
  char *face; //pointer to char type
  char *suit; //pointer to char type
};

int main()
{
  //declare the struct type variables
  Card p;
  Card *SPtr;
  p.face = "Ace";
  p.suit = "Spades";
  SPtr = &p;
  cout<<"Accessing structure element:\n";
  cout<<"\n\n'SPtr->suit' = "<<SPtr->suit<<endl;
  cout<<"'SPtr->face' = "<<SPtr->face<<endl;

  // cout<<"\n'SPtr->suit' = "<<p->suit<<endl; // ERROR
  // cout<<"'SPtr->face' = "<<p->face<<endl; // ERROR

  // structtest.cpp:23: error: base operand of ‘->’ has non-pointer type ‘Card’
  // structtest.cpp:24: error: base operand of ‘->’ has non-pointer type ‘Card’

}

// Running the program:
// ./structtest
// Accessing structure element:

// 'SPtr->suit' = Spades
// 'SPtr->face' = Ace
Arrays of Structures

For example, to store and manipulate the information contained in 100 student records, we use the following statement:

```c
struct student{
    char id[5];
    char name[80];
    char gender; int age;
}stud[100];
```

Or something like the following statements:

```c
struct student{
    char id[5];
    char name[80];
    char gender;
};
struct student stud[100];
```

These statements declare 100 variables of type `student` (a structure). As in arrays, we can use a subscript to reference a particular student structure or record. For example, to print the name of the seventh student, we could write the following statement:

```c
cout<<stud[6].name;
```
enum - Enumeration Constants

- This is another user-defined type consisting of a set of named constants called enumerators.
- Using a keyword `enum`, it is a set of **integer constants** represented by identifiers.
- The syntax is shown below:

```c
// for definition of enumerated type
enum [tag]
{
    enum-list
}
[declarator];
```

- And

```c
// for declaration of variable of type tag
enum tag declarator;
```

- These enumeration constants are, in effect, **symbolic constants** whose values can be set automatically. The values in an `enum` start with 0, unless specified otherwise, and are incremented by 1. For example, the following enumeration:

```c
denum days {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
```

- Creates a new data type, `enum days`, in which the identifiers are set automatically to the integers 0 to 6. To number the `days` 1 to 7, use the following enumeration:

```c
denum days {Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun};
```
int rollDice(); // rolls dice, calculates and displays sum

int main()
{
    // enumeration with constants that represent the game status
    enum Status { CONTINUE, WON, LOST }; // all caps in constants

    int myPoint; // point if no win or loss on first roll
    Status gameStatus; // can contain CONTINUE, WON or LOST

    // randomize random number generator using current time
    srand( time(0) );

    int sumOfDice = rollDice(); // first roll of the dice

    (...) switch ( sumOfDice )
    {
        case 7: // win with 7 on first roll
        case 11: // win with 11 on first roll
            gameStatus = WON;
            break;
        case 2: // lose with 2 on first roll
        case 3: // lose with 3 on first roll
        case 12: // lose with 12 on first roll
            gameStatus = LOST;
            break;
        default: // did not win or lose, so remember point
            gameStatus = CONTINUE; // game is not over
            myPoint = sumOfDice; // remember the point
            cout << "Point is " << myPoint << endl;
            break; // optional at end of switch
How to download an API off the internet and use it! 😊

- API’s or “Application Programming Interfaces” are tools programmers use to add functionality to their programs and essentially these contain 1) header files (.h) and 2) library files (.dll or .lib)

- Examples include: physics engines, GUI libraries, game engines, Excel file reading engines, etc. ... you name it.
Using the Excel Library “libxl”

http://www.libxl.com/download.html

**Download**

It will write banner in first row of each spreadsheet and it will be able to read only 100 cells (first row is unavailable). [Buy](http://www.libxl.com/download.html) a license key to remove banner and reading restriction.

<table>
<thead>
<tr>
<th>LibXL for Windows 3.4.1</th>
<th>(2013-01-04)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.libxl.com/download.html">Download</a></td>
<td></td>
</tr>
<tr>
<td>Size: 4 607 154 bytes</td>
<td></td>
</tr>
<tr>
<td>MD5: B208F9C22A46562D67E09FC56B53707</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LibXL for Linux 3.4.1</th>
<th>(2013-01-04)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.libxl.com/download.html">Download</a></td>
<td></td>
</tr>
<tr>
<td>Size: 6 933 809 bytes</td>
<td></td>
</tr>
<tr>
<td>MD5: DF17E4DFCBFE2525336F495CAACE4878</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LibXL for Mac 3.4.1</th>
<th>(2013-01-04)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.libxl.com/download.html">Download</a></td>
<td></td>
</tr>
<tr>
<td>Size: 11 708 099 bytes</td>
<td></td>
</tr>
<tr>
<td>MD5: 200C71DFF60E6E5F5D97207AE0ADA</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LibXL for iOS 3.4.1</th>
<th>(2013-01-04)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.libxl.com/download.html">Download</a></td>
<td></td>
</tr>
<tr>
<td>Size: 10 169 391 bytes</td>
<td></td>
</tr>
<tr>
<td>MD5: 30D3E9A8EF4CC1A02A93238A7BBF4DA3</td>
<td></td>
</tr>
</tbody>
</table>

**What's new in v.3.4.1:**

- added xllsm format support
- added UTF-8 support
- fixed some bugs

Join to our newsletter:

```plaintext
your@email
```

[Submit]
Using the Excel Library “libxl”

LibXL is a library for direct reading and writing Excel files.

Package contents:

- `bin` 32-bit dynamic library
- `bin64` 64-bit dynamic library
- `doc` C++ documentation
- `examples` C, C++, C#, Delphi and Fortran examples (MinGW, Visual Studio, Qt, Code::Blocks)
- `include_c` headers for C
- `include_cpp` headers for C++
- `lib` 32-bit import library for Microsoft Visual C++
- `lib64` 64-bit import library for Microsoft Visual C++
- `net` .NET wrapper (assembly)
- `changelog.txt` change log
- `libxl.url` link to home page
- `license.txt` end-user license agreement
- `readme.txt` this file
Using the Excel Library “libxl”

Using library:

1. Microsoft Visual C++

   - add include directory in your project, for example: c:\libxl\include_cpp

     Project -> Properties -> C/C++ -> General -> Additional Include Directories

   - add library directory in your project, for example: c:\libxl\lib

     Project -> Properties -> Linker -> General -> Additional Library Directories

   - add libxl.lib in project dependencies:

     Project -> Properties -> Linker -> Input -> Additional Dependencies

   - copy bin\libxl.dll to directory of your project
Using the Excel Library “libxl”

```cpp
#include <iostream>
#include <conio.h>
#include "libxl.h"

using namespace libxl;

int main()
{
    Book* book = xlCreateBook();
    if(book)
        if(book->load(L"..\generate\example.xls"))
        {
            Sheet* sheet = book->getSheet(0);
            if(sheet)
            {
                const wchar_t* s = sheet->readStr(2, 1);
                if(s) std::wcout << s << std::endl << std::endl;

                std::cout << sheet->readNum(4, 1) << std::endl;
                std::cout << sheet->readNum(5, 1) << std::endl;
                const wchar_t* f = sheet->readFormula(6, 1);
            }
        }
    else
    {
        std::cout << "At first run generate!" << std::endl;
    }
    book->release();
}

_`getch();`
return 0;
```
Using the Excel Library “libxl”

```cpp
class book {
 sheets* sheet;
public:
  book() { sheet = libxl::open (L"book.xls"); }
  ~book() { close(); }
private:
  void close() { delete sheet; sheet = nullptr; }
};

const wchar_t* f = sheet->readFormula(6, 1);
if(f)
  std::wcout << f << std::endl;
else
  std::cout << L"At first run generate !" << std::endl;

int year, month, day;
book->dateUnpack(sheet->readNum(8, 1), &year, &month, &day);
std::cout << year << L"-" << month << L"-" << day << std::endl;

book->release();
}

std::cout << L"\nPress any key to exit...";
_getch();
return 0;
}
LibXL is pay software, ExcelFormat is another FREE option

It will write banner in first row of each spreadsheet and it will be able to read only 100 cells (first row is unavailable). Buy a license key to remove banner and reading restriction. – your friends at LibXL

http://www.codeproject.com/Articles/42504/ExcelFormat-Library

Using the Code

To use the new formatting functionality, first create an `XLSFormatManager` object, like in the `example1()` function, and attach it to an existing `BasicExcel` object:

```cpp
void example1(const char* path)
{
    BasicExcel xls;

    // create sheet 1 and get the associated BasicExcelWorksheet pointer
    xls.New(1);
    BasicExcelWorksheet* sheet = xls.GetWorksheet(0);

    XLSFormatManager fmt_mgr(xls);
}
```

You can find all the examples of this article in the source code file `Examples.cpp`.

To define a custom font, create an `ExcelFont` object and set any needed properties, for example, the font weight for a bold font:

```cpp
ExcelFont font_bold;
font_bold._weight = FW_BOLD;  // 700=bold, 400=normal
```

The format of an Excel cell can be defined by a `CellFormat` object, which holds the chosen font and some more properties:

```cpp
CellFormat fmt_bold(fmt_mgr);
fmt_bold.set_font(font_bold);
```

After you have prepared the `CellFormat`, you can choose the font and display settings of Excel cells by calling `SetFormat()`: 
// Create a table containing a header row in bold and four rows below.
int col=0, row = 0;

for(col=0; col<10; ++col) {
    BasicExcelCell* cell = sheet->Cell(row, col);
    cell->Set("TITLE");
    cell->SetFormat(fmt_bold);
}

while(++row < 4) {
    for(int col=0; col<10; ++col)
        sheet->Cell(row, col)->Set("text");
}

++row;

Text color is specified by setting color indices in ExcelFont, for example:

ExcelFont font_red_bold;
font_red_bold._weight = FK_BOLD;
font_red_bold._color_index = EGA_RED;

CellFormat fmt_red_bold( fmt_mgr, font_red_bold);
fmt_red_bold.set_color1(COLOR1_PAT_SOLID); // solid background
fmt_red_bold.set_color2(MAKE_COLOR2(EGA_BLUE,0)); // blue background

CellFormat fmt_green( fmt_mgr, ExcelFont().set_color_index(EGA_GREEN));

for(col=0; col<10; ++col) {
    BasicExcelCell* cell = sheet->Cell(row, col);
    cell->Set("xxx");
    cell->SetFormat(fmt_red_bold);
    cell = sheet->Cell(row, ++col);
    cell->Set("yyy");
    cell->SetFormat(fmt_green);
}

ExcelFormat.h contains constants to define basic palette colors in the enumeration EXCEL_COLORS, you can use in
calls to ExcelFont::set_color_index(). The macro MAKE_COLOR2 accepts two color indices to specify the
pattern line and background colors. As a shortcut to calling CellFormat::set_color1() and
CellFormat::set_color2(), you can also use CellFormat::set_background() to define cells with solid
background colors or colorize patterns.

After creating and formatting the Excel cells in memory, all you have to do is to save the new Excel sheet as a file:

xls.SaveAs(path);
What these Excel interfaces are actually doing....

```c
struct CODE {
    enum {
        FORMULA=0x0006,
        YEOF=0x000A,
        CALCCOUNT=0x000C,
        CALCMODE=0x000D,
        PRECISION=0x000E,
        REFMODE=0x000F,
        DELTA=0x0010,
        ITERATION=0x0011,
        PROTECT=0x0012,
        PASSWORD=0x0013,
        HEADER=0x0014,
        FOOTER=0x0015,
        EXTERNSHEET=0x0017,
        NAME=0x0018,
        ... //Token array and the result of a formula cell.
        //End of a record block with leading BOF record.
        //Maximum number of times the formulas should be iteratively calculated
        //Calculate formulas manually, automatically, or automatically except for multiple table operations
        //Whether formulas use the real cell values for calculation or the values displayed on the screen.
        //Method used to show cell addresses in formulas.
        //Maximum change of the result to exit an iteration.
        //Whether iterations are allowed while calculating recursive formulas.
        //Whether worksheet or a workbook is protected against modification.
        //16-bit hash value, calculated from the worksheet or workbook protection password.
        //Page header string for the current worksheet.
        //Page footer string for the current worksheet.
        //List with indexes to SUPBOOK records
        //Name and token array of an internal defined name.
```
You can download the class example programs from here:

A simple class GradeBook

```cpp
// GradeBook class definition
class GradeBook {
    public:
        // function that displays a welcome message to the GradeBook user
        void displayMessage()
        {
            cout << "Welcome to the Grade Book!" << endl;
        }
    } // end function displayMessage
} // end class GradeBook

// function main begins program execution
int main()
{
    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
    myGradeBook.displayMessage(); // call object's displayMessage function

    Welcome to the Grade Book!
} // end main
```
Let’s compile & run the program!

// Define class GradeBook with a member function displayMessage;
// Create a GradeBook object and call its displayMessage function.
#include <iostream>
using namespace std;

// GradeBook class definition
class GradeBook
{
 public:
   // function that displays a welcome message to the GradeBook user
   void displayMessage()
   {
      cout << "Welcome to the Grade Book!" << endl;
   } // end function displayMessage
}; // end class GradeBook

// function main begins program execution
int main()
{
   GradeBook myGradeBook; // create a GradeBook object named
   myGradeBook
   myGradeBook.displayMessage(); // call object's displayMessage function

} // end main
Add set() and get() functions to the class

```cpp
// Define class GradeBook that contains a courseName data member
// and member functions to set and get its value;
// Create and manipulate a GradeBook object with these functions.
#include <iostream>
#include <string> // program uses C++ standard string class
using namespace std;

// GradeBook class definition
class GradeBook
{
    public:
        // function that sets the course name
        void setCourseName( string name )
        {
            courseName = name; // store the course name in the object
        } // end function setCourseName

        // function that gets the course name
        string getCourseName()
        {
            return courseName; // return the object's courseName
        } // end function getCourseName

        // function that displays a welcome message
        void displayMessage()
        {
            // this statement calls getCourseName to get the
            // name of the course this GradeBook represents
            cout << "Welcome to the grade book for\n" << getCourseName() << "!"
            << endl;
        } // end function displayMessage
```
private:
    string courseName; // course name for this GradeBook
}; // end class GradeBook

// function main begins program execution
int main()
{
    string nameOfCourse; // string of characters to store the course name
    GradeBook myGradeBook; // create a GradeBook object named myGradeBook

    // display initial value of courseName
    cout << "Initial course name is: " << myGradeBook.getCourseName() << endl;

    // prompt for, input and set course name
    cout << "Please enter the course name:" << endl;
    getline( cin, nameOfCourse ); // read a course name with blanks
    myGradeBook.setCourseName( nameOfCourse ); // set the course name

    cout << endl; // outputs a blank line
    myGradeBook.displayMessage(); // display message with new course name
} // end main

Initial course name is:

Please enter the course name:
CS101 Introduction to C++ Programming

Welcome to the grade book for
CS101 Introduction to C++ Programming!
Dear class, let’s try, and fail, to access a private member of a class (without a `get()` function)

```cpp
// function main begins program execution
int main()
{
    string nameOfCourse; // string of characters to store the course name
    GradeBook myGradeBook; // create a GradeBook object named myGradeBook

    // display initial value of courseName
    cout << "Initial course name is: " << myGradeBook.getCourseName() << endl;

    /***************************************************************************/
    cout << "Initial course name is: " << myGradeBook.courseName; // ERROR
    cout << endl;

    $ g++ Deitel-ch3-cpp.cpp -o Deitel-ch3-cpp
    Deitel-ch3-cpp.cpp: In function ‘int main()’::
    Deitel-ch3-cpp.cpp:34: error: ‘std::string GradeBook::courseName’ is private
    Deitel-ch3-cpp.cpp:47: error: within this context */

    // prompt for, input and set course name
    cout << "\nPlease enter the course name:" << endl;
    getline( cin, nameOfCourse ); // read a course name with blanks
    myGradeBook.setCourseName( nameOfCourse ); // set the course name

    cout << endl; // outputs a blank line
    myGradeBook.displayMessage(); // display message with new course name
} // end main
```
// Define class GradeBook that contains a courseName data member
// and member functions to set and get its value;
// Create and manipulate a GradeBook object.
#include <iostream>
#include <string> // program uses C++ standard string class
using namespace std;

// GradeBook class definition
class GradeBook
 {
  public:
    // function that sets the course name
    void setCourseName(string name)
    {
      courseName = name; // store the course name in the object
    } // end function setCourseName
    
    // function that gets the course name
    string getCourseName()
    {
      return courseName; // return the object's courseName
    } // end function getCourseName
    
    // function that displays a welcome message
    void displayMessage()
    {
      // this statement calls getCourseName to get the
      // name of the course this GradeBook represents
      cout << "Welcome to the grade book for" << getCourseName() << "!
      << endl;
    } // end function displayMessage
  private:
    string courseName; // course name for this GradeBook
  }; // end class GradeBook

// main begins program execution
int main()
{
  string courseNameOfCourse; // string of characters to store the course name
  GradeBook myGradeBook; // create a GradeBook object named myGradeBook

  // display initial value of courseName
  cout << "Initial course name is: " << myGradeBook.getCourseName() << endl;

  // prompt for, input and set course name
  cout << "Please enter the course name:" << endl;
  getline(cin, courseNameOfCourse); // read a course name with blanks
  myGradeBook.setCourseName(courseNameOfCourse); // set the course name

  cout << endl; // outputs a blank line
  myGradeBook.displayMessage(); // display message with new course name
} // end main

This is the copy-paste-able source code for the class program with get() and set() parameters
However, with a friend function we can access the private members

```cpp
// Fig. 10.13: fig10_13.cpp
// Friends can access private members of a class.
#include <iostream>
using namespace std;

// Count class definition
class Count
{
    friend void setX( Count &, int ); // friend declaration

    public:
    // constructor
    Count() : x( 0 ) // initialize x to 0
    {
        // empty body
    } // end constructor Count

    // output x
    void print() const
    {
        cout << x << endl;
    } // end function print

    private:
    int x; // data member
}; // end class Count

// function setX can modify private data of Count
// because setX is declared as a friend of Count (line 9)
void setX( Count &c, int val )
{
    c.x = val; // allowed because setX is a friend of Count
} // end function setX

int main()
{
    Count counter; // create Count object
    cout << "counter.x after instantiation: ";
    counter.print();
    setX( counter, 8 ); // set x using a friend function
    cout << "counter.x after call to setX friend function: ";
    counter.print();
    return 0;
} // end main
```

counter.x after instantiation: 0
counter.x after call to setX friend function: 8
This is the copy-paste-able source code for using friends to access private parts.
Without *friend* – compiler error

```bash
$ g++ friend.cpp -o friendcpp
friend.cpp: In function ‘void setX(Count&, int)’:
friend.cpp:25: error: ‘int Count::x’ is private
friend.cpp:32: error: within this context
```

With *friend*

```bash
$ g++ friend.cpp -o friendcpp
Russells-MacBook-Pro:TurbineWarnerBros
russell$ ./friendcpp
counter.x after instantiation: 0
counter.x after call to setX friend function: 8
```
**Protected** base-class data can be accessed from derived classes

protected base-class data can be accessed from derived class.

**Notes on Using protected Data**
In this example, we declared base-class data members as protected, so derived classes can modify the data directly. Inheriting protected data members slightly improves performance, because we can directly access the members without incurring the overhead of calls to *set* or *get* member functions.
Member initializer list, separated by commas

Employee Constructor’s Member Initializer List
The colon (:) following the constructor’s header (Fig. 10.11, line 12) begins the member initializer list. The member initializers specify the Employee constructor parameters being passed to the constructors of the string and Date data members. Parameters firstName, lastName, dateOfBirth and dateOfHire are passed to the constructors for objects firstName’s (Fig. 10.11, line 12), lastName (Fig. 10.11, line 13), birthDate (Fig. 10.11, line 14) and hireDate (Fig. 10.11, line 15), respectively. Again, member initializers are separated by commas.

Default and copy constructors

A copy constructor is a special constructor in the C++ programming language for creating a new object as a copy of an existing object. The first argument of such a constructor is a reference to an object of the same type as is being constructed (const or non-const), which might be followed by parameters of any type (all having default values).

According to the standard, a class’s default constructor is automatically created if no constructor is defined (Fig. 10.11, line 9). The default constructor creates a default-constructed object. 

Normally the compiler automatically creates a copy constructor for each class (known as a default copy constructor) but for special cases the programmer creates the copy constructor, known as a user-defined copy constructor. In such cases, the compiler does not create one. Hence, there is always one copy constructor that is either defined by the user or by the system.
Let’s do an example with a constructor and destructor, these are quite common.

```cpp
// CreateAndDestroy class definition.
// Member functions defined in CreateAndDestroy.cpp.
#include <string>
using namespace std;

#ifdef CREATE_H
#define CREATE_H

class CreateAndDestroy
{
  public:
    CreateAndDestroy( int, string ); // constructor
    ~CreateAndDestroy(); // destructor
  private:
    int objectID; // ID number for object
    string message; // message describing object
}; // end class CreateAndDestroy

#endif

// CreateAndDestroy class member-function definitions.
#include <iostream>
#include "CreateAndDestroy.h" // include CreateAndDestroy class definition

// constructor
CreateAndDestroy::CreateAndDestroy( int ID, string messageString )
{
  objectID = ID; // set object's ID number
  message = messageString; // set object's descriptive message
  cout << "Object " << objectID << " constructor runs "
    << message << endl;
} // end CreateAndDestroy constructor

// destructor
CreateAndDestroy::~CreateAndDestroy()
{
  // output newline for certain objects; helps readability
  cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
  cout << "Object " << objectID << " destructor runs "
    << message << endl;
} // end ~CreateAndDestroy destructor
```
// Demonstrating the order in which constructors and
// destructors are called.
#include <iostream>
#include "CreateAndDestroy.h" // include CreateAndDestroy class definition
using namespace std;

void create( void ); // prototype

CreateAndDestroy first( 1, "(global before main)" ); // global object

int main()
{
    cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
    CreateAndDestroy second( 2, "(local automatic in main)" );
    static CreateAndDestroy third( 3, "(local static in main)" );
    create(); // call function to create objects
    cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
    CreateAndDestroy fourth( 4, "(local automatic in main)" );
    cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
} // end main

// function to create objects
void create( void )
{
    cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
    CreateAndDestroy fifth( 5, "(local automatic in create)" );
    static CreateAndDestroy sixth( 6, "(local static in create)" );
    CreateAndDestroy seventh( 7, "(local automatic in create)" );
    cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;
} // end function create
This is the copy-paste-able source code

// CreateAndDestroy class definition.
// Member functions defined in CreateAndDestroy.cpp.
#include <string>
using namespace std;

#ifndef CREATE_H
#define CREATE_H

class CreateAndDestroy
{
    public:
        CreateAndDestroy( int, string ); // constructor
        ~CreateAndDestroy(); // destructor
    
    private:
        int objectID; // ID number for object
        string message; // message describing object
    }
#endif

// CreateAndDestroy class member-function definitions.
#include <iostream>
#ifdef CREATE_H
#include "CreateAndDestroy.h" // include CreateAndDestroy class definition using namespace std;

void create( void ); // prototype

CreateAndDestroy first( 1, "(global before main)" ); // global object
int main()
{
    cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
    CreateAndDestroy second( 2, "(local automatic in main)" );
    static CreateAndDestroy third( 3, "(local static in main)" );
    create(); // call function to create objects
    cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
    CreateAndDestroy fourth( 4, "(local automatic in main)" );
    cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
    }
    // end main

    // function to create objects
    void create( void )
    {
        cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
        CreateAndDestroy fifth( 5, "(local automatic in create)" );
        static CreateAndDestroy sixth( 6, "(local static in create)" );
        CreateAndDestroy seventh( 7, "(local automatic in create)" );
        cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;
        }
    // end function create

void create( void ); // prototype

CreateAndDestroy first( 1, "(global before main)" ); // global object
int main()
{
    cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
    CreateAndDestroy second( 2, "(local automatic in main)" );
    static CreateAndDestroy third( 3, "(local static in main)" );
    create(); // call function to create objects
    cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
    CreateAndDestroy fourth( 4, "(local automatic in main)" );
    cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
    }
    // end main

    // function to create objects
    void create( void )
    {
        cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
        CreateAndDestroy fifth( 5, "(local automatic in create)" );
        static CreateAndDestroy sixth( 6, "(local static in create)" );
        CreateAndDestroy seventh( 7, "(local automatic in create)" );
        cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;
        }
    // end function create
Definition for a complex number class with real and imaginary parts

```cpp
// Complex class definition.
#ifndef COMPLEX_H
#define COMPLEX_H

class Complex {
    public:
        Complex( double = 0.0, double = 0.0 ); // constructor
        Complex operator+( const Complex & ) const; // addition
        Complex operator-( const Complex & ) const; // subtraction
        void print() const; // output

    private:
        double real; // real part
        double imaginary; // imaginary part
}; // end class Complex

#endif
```
// Complex class member-function definitions.
#include <iostream>
#include "Complex.h" // Complex class definition
using namespace std;

// Constructor
Complex::Complex( double realPart, double imaginaryPart ) :
    real( realPart ),
    imaginary( imaginaryPart )
{
    // empty body
} // end Complex constructor

// addition operator
Complex Complex::operator+( const Complex &operand2 ) const
{
    return Complex( real + operand2.real,
                    imaginary + operand2.imaginary );
} // end function operator+

// subtraction operator
Complex Complex::operator-( const Complex &operand2 ) const
{
    return Complex( real - operand2.real,
                    imaginary - operand2.imaginary );
} // end function operator-
To randomize without having to enter a seed each time, we may use a statement like

```
srand( time( 0 ) );
```

This causes the computer to read its clock to obtain the value for the seed. Function `time` (with the argument 0 as written in the preceding statement) typically returns the current time as the number of seconds since January 1, 1970, at midnight Greenwich Mean Time (GMT). This value is converted to an unsigned integer and used as the seed to the random number generator. The function prototype for `time` is in `<ctime>`.

**Generalized Scaling and Shifting of Random Numbers**

Previously, we simulated the rolling of a six-sided die with the statement

```
face = 1 + rand() % 6;
```

which always assigns an integer (at random) to variable `face` in the range $1 \leq face \leq 6$. The width of this range (i.e., the number of consecutive integers in the range) is 6 and the starting number in the range is 1. Referring to the preceding statement, we see that the width of the range is determined by the number used to scale `rand` with the modulus operator (i.e., 6), and the starting number of the range is equal to the number (i.e., 1) that is added to the expression `rand % 6`. We can generalize this result as

```
number = shiftingValue + rand() % scalingFactor;
```

where `shiftingValue` is equal to the first number in the desired range of consecutive integers and `scalingFactor` is equal to the width of the desired range of consecutive integers.
A simple die-rolling program

```cpp
#include <iostream>
#include <iomanip>
#include <cstdlib> // contains prototypes for functions srand and rand
using namespace std;

int main()
{
    unsigned seed; // stores the seed entered by the user

    cout << "Enter seed: ";
    cin >> seed;
    srand( seed ); // seed random number generator

    // loop 10 times
    for ( int counter = 1; counter <= 10; ++counter )
    {
        // pick random number from 1 to 6 and output it
        cout << setw( 10 ) << ( 1 + rand() % 6 );

        // if counter is divisible by 5, start a new line of output
        if ( counter % 5 == 0 )
            cout << endl;
    } // end for
} // end main
```
// Randomizing die-rolling program.
#include <iostream>
#include <iomanip>
#include <cstdlib>  // contains prototypes for functions srand and rand
using namespace std;

int main()
{
    unsigned seed;  // stores the seed entered by the user
    cout << "Enter seed: ";
    cin >> seed;
    srand( seed );  // seed random number generator

    // loop 10 times
    for ( int counter = 1; counter <= 10; ++counter )
    {
        // pick random number from 1 to 6 and output it
        cout << setw( 10 ) << ( 1 + rand() % 6 );
        // setw is "set width" of output operations
        // if counter is divisible by 5, start a new line of output
        if ( counter % 5 == 0 )
            cout << endl;
    }  // end for
}  // end main
Here is a BankDatabase class

```cpp
#include "Account.h" // Account class definition

class BankDatabase
{
  public:
    BankDatabase(); // constructor initializes accounts

    // determine whether account number and PIN match those of an Account
    bool authenticateUser(int, int); // returns true if Account authentic

    double getAvailableBalance(int); // get an available balance
    double getTotalBalance(int); // get an Account's total balance
    void credit(int, double); // add amount to Account balance
    void debit(int, double); // subtract amount from Account balance

  private:
    vector<Account> accounts; // vector of the bank's Accounts

    // private utility function
    Account * getAccount(int); // get pointer to Account object
}; // end class BankDatabase
```
Here is a Paddle class

```cpp
// Paddle.h
// Paddle class definition (represents a paddle in the game).
#ifndef PADDLE_H
#define PADDLE_H

#include <Ogre.h> // Ogre class definitions
using namespace Ogre;

class Paddle
{
    public:
        // constructor
        Paddle(SceneManager *sceneManagerPtr, String paddleName,
                int positionX);
        ~Paddle(); // destructor
        void addToScene(); // add a Paddle to the scene
        void movePaddle( const Vector3 &direction ); // move a Paddle

    private:
        SceneManager* sceneManagerPtr; // pointer to theSceneManager
        SceneNode *nodePtr; // pointer to a SceneNode
        String name; // name of the Paddle
        int x; // x-coordinate of the Paddle
}; // end of class Paddle
#endif // PADDLE_H
```

**Fig. 27.10** | Paddle class definition (represents a paddle in the game).
Homework for Next Monday (pick two)

1) Write a class for a character in gameplay, with set() and get() methods. You should be able to get() or set() the player’s health, location, direction of movement, velocity, and any gear he/she has with him. What character data should be public or protected or private? What data structure should be used for the player’s direction or current location? What data structure should be used for his/her gear? Write how a driver class can interact with the character class, by updating, interacting with other characters, or interacting with the environment.

2) Design and implement a blackjack or twenty-one game using the randomization functions we have used for cards/suits. Allow playing against the computer as the “house.” Reveal all the cards at the end of a hand.

3) Write a function mysort() that takes an array of integers or floating point values and returns them sorted from least to greatest. Bonus 1: Pass the array by reference and change the original array. Bonus 2: Include an option to sort the array from greatest to least.


5) Outline how to implement the game Pong using pseudocode and/or class diagrams (http://en.wikipedia.org/wiki/Class_diagram). Bonus: Implement one of these classes.
Homework for Next Monday (pick two)

6) Write a concatenate function that takes a variable number of arguments (may or may not be of different data types) and returns their concatenation as a string.

7) Overload the ‘+’ operator to concatenate not only strings but also integers and/or floating point numbers.

8) Write a .cpp file that uses a function prototype and default parameters for a function.