C++ Notes

Joseph Bentley

DeAnza College
Computer Information System
November 2015

Table of Contents

Course Information

Course Objectives

At the completion of the course, you should be able to write basic C++ programs which make use of the following:

- reference variables
- default arguments
- dynamic memory allocation
- classes
- constructors and destructors
- static data members and static member functions
- function overloading and operator overloading
- inheritance
- polymorphism
- C++ input/output classes and file I/O

Course Outline

1. Intro to C++
2. Difference between C & C++: reference variables, default arguments, new and delete
3. Introduction to Classes
4. Constructors and Destructors
5. More class features: this pointer, static members, friend functions
6. Function and Operator Overloading
7. Inheritance and Polymorphism
8. Input/Output and File I/O
9. C++ Applications & Review
10. Final

Preface

These notes are not intended to be a textbook. These pages represent numerous revisions of examples and notes of C++ concepts that I used for myself to gain an understanding of the language over the last dozen, or so, years. Since I learn best by looking at examples, I decided long ago to use these notes as a teaching tool. Every time I teach a class in C++ using these notes, I find mistakes, shortcomings, inaccuracies, and explanations needed. I make a list of corrections and notes to myself to rewrite this or that. And, even though I update the notes almost every time I teach the class, I never get it right. I do believe, however, that this makes me a better teacher – not being satisfied. I think that if I ever got it right, I'd have to quit (by then the language would be totally obsolete).

To make effective use of these notes, you have to learn to read examples. Reading an example of code, is not like reading anything else. It's like, you read a line of code, then you ask yourself:
"What does this mean?"
"Why did the author do it this way?"
"What's that function?" (time out while you go look it up)
"What does that syntax mean?" "Who's doing what to whom?"
"Is there another way of doing this?"

Step back …

"What's the point?" (do I understand the concept(s) that Joe is trying to demonstrate)

This is a time-consuming and tedious process. (I can't read very much of someone else's code without getting antsy and distracted). As you become more experienced, you will be able to skip over "obvious" lines of code and concentrate on the gist of the example. (After you've seen *#include <iostream>* dozens of times, you won't even think about it). To be successful in reading examples, I recommend that you don't try to spend a lot of time doing it. Reading one or two examples and really getting it is better than trying to read six or ten examples and kinda", "sorta", getting it.

Reading an example and getting it means that you "own the code". It's yours now. It doesn't mean memorizing it. It doesn't even mean that you don't have to look back see how to do that. It means that you understand how it works and you can reproduce, when needed, the concept or the logic (and take another look if you need to). After all, when you're cooking lasagna, you may have made it dozens of times, but it doesn't hurt to have the recipe next to you when you are making it for the fifty-first time.

Joe
January 2009

Some thoughts on the class

The following notes represent some on my thoughts about this course.

Why learn C++?

C++ is programming language that is very much in demand today, probably the language that is in most demand currently. It will definitely be a plus to have it on your resume.

What does the class cover?

CIS27 is a basic C++ class. Upon successful completion, you should be able to write C++ code, to read it, to use C++ reference manuals, and to step into an entry-level C++ programming situation. This is not an advanced class. It does not cover templates, exception handling, the Standard Template Library (STL), RTTI, writing your own manipulators, binary trees, and object oriented programming concepts. It is the basics, the language syntax, as well as language concepts. There are separate courses for the advanced concepts and object oriented programming.

What are the prerequisites?

Successful completion of a C programming class. That means a grade of A, B or C in such a class. You do not need any significant C programming experience, but you do need to be familiar with the basics, such as, variables, data types, for loops, while loops, input and output, file I/O, pointers, arrays, string functions, and basic ANSI standard functions. Do not expect to successfully complete this class without some C experience.

How can you be successful in this class?

"Successful" probably means an "A" in the class. An "A" means you can put it on your resume. It means that you could step into an entry-level C++ programming position and produce code within a short time. A "B" means that you missed a little, but with a little study and work you can be right there with the "A" types. A "C" means that there's hope, but you'll need to put in some time to catch up to the "A"s. Any other grade should repeat the course, probably after some C programming review. You're may be taking this class to help you get (or keep) a job. It's the "A"s that stand the best chance.

Now, back to the question. This class is 12 weeks long. It is a fairly short time commitment. You can be successful by doing the following:

- **Meet the prerequisites**. Make sure you're comfortable with C. During the course, if you hear of a C concept that you are not familiar with, research it, or ask about it and get it.

- **Commit the time** required for this course. You will need approximately 8 to 12 hours per week outside of class to complete the assignments and do the suggested reading. If you don't have the time, or don't commit to it, don't plan on an A or B in the class (and there isn't that many Cs received).

- **Come to class and be on time**. Students who are on the road to "success", but have to miss a class, usually get behind. This probably translates into a full letter grade. If you "have to" miss a class session, plan on doing lots or reading and studying to make it up. Punctuality is especially important during the final and midterm. The tests have time limits. Final grades have dropped a full letter, because students showed up late for a test and didn't have enough time.

- **Get and read a textbook.** The course notes are not a text book. They do not contain detailed explanations of C++ concepts. You should acquire a source for explanations beyond the examples in the notes. Check Appendix B for some possible books.

- **Do every assignment**. Start early. Learn to break up the problem into small parts. Do one part at a time. Test your code as you proceed. This is particularly relevant when you start writing classes and member functions. If the assignment involves writing 2 classes, then do one at a time. Write one member function at a time, and test it. Make sure you understand what each part is supposed to accomplish. It not, ask. When you get stuck, try to solve the problem yourself. When you still can't get it, ask for help (see below).

- **Study for the tests**. The tests are open book, but that's not the time to start reading. You should know exactly what topics are on the test. Do you know each one, or not? It's fair to ask for an explanation of some topic before the morning of the test. To study for a test shouldn't take long if you've kept up. You should be able to look at a list of topics, think about them, and consider whether or not you thoroughly understand the topic. Is there some syntax or notation that you do not understand about that topic?

- **Ask questions**. In class and out of class. It's difficult to concentrate for 2+ hours of lecture (no matter how brilliant and entertaining the instructor is). You need to psyche yourself up to endure this. A good night's sleep the night before, some Starbuck's coffee, whatever it takes…

- **Use the lab time for some extra help**, hand holding, showing you how to do it, explaining some concept in detail, giving you a hint on an assignment.

- **Talk to other students in the class about assignments, problems or various topics**. This does not mean copying. This is not a course to practice typing or copying files. You are encouraged to discuss problems with other students, but not to copy their solution. This is just like a programming job. You will probably consult with coworkers regarding problems, but you cannot expect a coworker to do your work for you. If you need a hint on an assignment, ask the instructor. Remember, you will be taking the tests by yourself.

- Asking Email Questions

You will need to ask a question (probably several) during the course. Do not hesitate to send me an email question. You should get a reply with 24 hours. Make sure you do get a reply, if not, send the question again, and if necessary, call me up. It's important that you understand what is a "fair" email question, a what is not.

It's fair to ask me to explain line 19 of example 2-5 or to explain what is meant by the 4th paragraph on page 317. It is not fair to ask to explain all 450 lines of example 2-7 or describe in an email note how constructors work or explain chapter 7 in the text. It is fair to ask what a copy constructor is.

The following are "fair" questions to ask about an assignment:

- What is causing this error message: "Call to undefined function: strcopy()" on line 54 of my classX::funk() function? (The code is included)
- What is the purpose of the goo() function in the XYZ class?
- Can you give me a little hint about how to start writing the moo() function for ABC class?
- Did we have an example similar to the poo() function?

These are "unfair" questions to ask about an assignment:

- I don't understand what to do?
- How do I get started?
- I don't understand classes?
- How do I write the moo() function for ABC class?
- What is causing this error message: "Call to undefined function: strcopy()" on line 54 of my classX::funk() function? (The code is **not** included)
- I've got a whole bunch of error messages, what do they mean?
- What do these 3 error messages mean (messages included)? (2 is the limit)
- Can I turn the assignment late?

When you submit an email question about an assignment, make sure you include all relevant parts of your code. If you're not sure whether or not to include part of your program, then include it. I would prefer that your code is included in the body of the note. If you want to use an attachment, my first preference is one file attachment (even though you may have a multi-file application).

Introduction to C++

What is C++ ?

- it's "a better C"

- it supports data abstraction

- it's an Object-Oriented Programming Language

  OOPLs have the following characteristics:

  - **Encapsulation** is the combining the data structure with actions. The data structure is used to represent the properties, the state, or characteristics of objects. The actions represent permissible behaviors of objects. These are controlled through the member functions that are attached to objects. It is through encapsulation that an object may be much more realistically represented - by including both the properties and the behavior in its definition. Further, the class designer can control the user interface much easier with this approach.

  - **Inheritance** is the ability to define a hierarchical relationship between objects that permits objects of a more specific class to inherit the properties (data) and behaviors (functions) of a more general class. For example, you might have dog objects with associated properties and behaviors and also beagle objects that will have all the properties and behaviors of dogs with a few more specific properties and behaviors of their own.

  - **Polymorphism** is the ability for different objects to interpret messages (functions) differently. For example, asking different shape objects (circle, square, rectangle, trapezoid) to return their area will result in different implementations. This is accomplished with virtual functions using "late (or dynamic) binding".

  OOPLs support modular programming, ease of development and maintainability.

What is Object Oriented Programming? *February 2009 blog*

C++ is an object oriented programming language. This means that the language is oriented toward programming with objects. Well, duh! Object oriented programming is a different approach than you learned in C. In C++, you create classes, after thinking about the design and what you want to accomplish, and, by the way, you usually don't get that part right on the first try. The class consists of members - data members or member functions. Member functions are also called methods. They're also called behaviors. Once you've created a class, you can declare (or define) a variable of that class type. That variable can be referred to as an object, or an instance (of the class type). You can then call a function using the object. Another way of saying that is, "you can apply the member function to the object". The member function is, well, first of all, it's just a function, meaning, it does something. Sometimes, the member function does something to the object. Sometimes, the member function does something for the object. Sometimes, the member function tells you something about the object. Sometimes, the member function is used to create the object (that's called a constructor). Sometimes, the member functions are used to destroy, or get rid of, the object (that's called a destructor). Some member functions look like operators (think + - / * -> % ! ~) when you called. C++ is all about accomplishing your task using objects and using the member functions that belong to the class(es). You can't really do anything with C++ that can't do with C, but once you get the hang of it, it's, in my opinion, a more natural way of getting the job done. Don't expect to immediately start coding using an object oriented approach, but may by the time you have completed this course, and have been exposed to dozens of examples in which classes have been used in many way, something should maybe ought to rub off.

C++ is "a better C", supports data abstraction, and is an effective OOPL primarily due to it use of classes. In C++, the class is the cornerstone of the language. Yes, C++ includes new syntax, new operators, new functions, and enhanced libraries, but it's really the implementation of classes, that give the language its identity.

Classes are:

- a more powerful type of struct
- data (properties, characteristics, state) and behaviors (methods)

For example, a dog class may contain data members such as breed, color, height, weight, number of feet (usually 4), eye color, etc. The class may also contain the behaviors that the dog can exhibit (the actions that it can perform, or the methods that it can apply). The dog may be able to sit, to run, to eat (that might affect its weight), etc. Sit, run, and eat would be member functions of the dog class. The class itself is not data.

An object is an instance (or occurrence) of a class. For example, you might have a dog, called Spot. Spot is an object. (Don't try to use this analogy on spouses!) Spot will possess all the properties of dog and can perform the behaviors of a dog. You might want Spot to sit down, so you'd say to spot: spot.sit(); That's, of course, assuming that Spot knows a little C++. The following C++ code illustrates these concepts.

```
class dog
{
  private:
    char breed[25];
    char color[20];
    int height;
    float weight;
    int feet;
    char eye_color[10];
  public:
    void sit(void);
    void eat(void);
    void run(void);
};

int main(void)
{
  dog spot;
  spot.sit();
  spot.eat();
  spot.run();
  return 0;
}
```

Getting Started

To get started with C++, we should dive right in, start writing code and see what's new. This section will address four differences between C and C++.

- Writing comments in a C++ program
- Using cin and cout for input and output to a C++ program (and #include <iostream>)
- Declaring variables almost anywhere
- Using type bool for true or false

Comments in a C++ program

In C, you learned to use the /* … */ style comment. Since C++ includes the C language syntax, you can, of course, use the same style comments, but C++ also includes its own style of writing a program comment. This is accomplished using //. The // can be used any place on a program line to mean that anything to the right of the // is intended to be a comment and not part of the program – not compiled. An entire line may be commented, like this:

```
// see, this is a comment consisting of an entire line
```

or part of a line like this:

```
if (b * b – 4 * a * c < 0) {          // make sure the determinant is not negative
```

You can use either C-style comments, /* … */, or C++ style comments or both in your program. The advantage of the C-style comment is that the comment can be long and span many lines, and the advantage of the C++ comment is that it's easier, two keystrokes instead of four. Besides, this is all about C++ anyway. There is one word of caution. Be careful of nesting the two styles. For example, this is OK.

```
/* this is a one-line comment */

/* this is
    a comment that
    goes on and on,
    and ends on this line */

/* Here's another comment // and what's this?
    Oh, who cares?
    That's all folks */
```

Now, here's the rub:

```
// this is OK

/* nothing wrong with
    this comment
*/
```

```
// This is a C++ style comment /* and now let's do a
    C-style comment */

/* Here's another
// screw up */
```

Do you see the problem? Just be careful.

cin and cout

In learning C, you probably started with scanf() and printf() for input and output. In C++, we will do the same thing with cin and cout. cin will be used for input from the keyboard and cout for output to the screen.

In C, it looks like this:

```
printf("Enter some number\n");
scanf("%d",&i);
```

In C++, like this:

```
cout << "Enter some number\n";
cin >> i;
```

scanf() and printf() are functions. cin and cout are not function they are things, or more precisely, objects. You can think of them as the keyboard and monitor (or screen). To be more precise, cin is an object of type istream and cout is an object of type ostream. And what about << and >>? These guys (*guys* is a technical term, more precisely, a male technical term) are operators, just like the + in $x + y$. And in case you wanted to know, operators, in C++ can be the same thing as functions.

So, of course, our two lines of code, displays *Enter some number* on the screen and the program stops, waiting for the user to enter a number. It acts just like the C code.

One more point, cin and cout are not free, just like printf() and scanf() are not free. To use printf() and scanf() you must include the header file, <*stdio.h*>. Similarly, in C++, for cin and cout, you will need to include the header file, <*iostream*>. Note that it is <*iostream*>, not <*iostream.h*>. We'll get to that later.

What exactly is cin and cout?

Consider the statements,

```
int age;
cin >> age;
cout << "I am " << age << " years old." << endl;
```

In C we would write,

```
int age;
scanf("%d",&age);
printf("I am %d years old.\n",39);
```

These two statements do the same thing. **cin** and **cout** are **objects**. This means that they are variables of a certain type (specially they are variables of type istream and ostream). They are not functions, like printf() and scanf(). The function part of the statements is the >> and << operators. And, while we're at it, endl is almost the same as **\n** .

Declaring variables where you want

In C you learned to declare variables at the beginning of a function and in C++ you can do the same thing. That, of course, would be too boring. So, there's another way. You can declare variables where you need them. Variables do not have to be declared at the beginning of your program, or the beginning of a function, or at the top of a block. They can be declared just before you use them. Of course, you can't declare them after you use them. For example, in C you would:

```
int x;
printf("Enter some number\n");
scanf("%d",&x);
```

And is C++, you can do this:

```
printf("Enter some number\n");
int x;
scanf("%d",&x);
```

There is the advantage or declaring variables at the top of a function. The reader of the code (that might be you in six months), knows where to look for variable declarations. On the other hand, being able to declare variable any old time allows you to write code without a lot of up front planning. That's a good thing, right?

Type bool

ANSI/ISO C++ includes a type called bool to store true-false values. This concept has been around forever in programming and in C and even C++. I guess the only issue was settling on the name of the type. An obvious application might look like this:

```
…
bool rich;
rich = money > 1000000;
if (rich) {
  cout << "Whoopee!";
}
```

…

Here is the first example that demonstrates the some initial C++ concepts and some differences between C and C++.

Example 1-1

```
1  // File:  Ex1-1.cpp
2
3  // Illustrates some of the basic differences between C and C++:
4  // Comments
5  // cin and cout for input and output
6  // declarations of variables almost anywhere
7  // use of type bool
8
9  #include <iostream>      // instead of <stdio.h> or <iostream.h>
10  using namespace std;
11
12  /* You can still use the old comment, */
13
14  /* but you must be // very careful
15  about mixing them */
16
17  // Your best bet is to use this style for 1 line or a partial line
18  /* And use this style when your comment
19  consists of multiple lines */
20
21  int main (void)
22  {
23    cout << "hey";              // Why won't printf or puts work here?
24    //printf("hey");
25    //puts("hey");     // Can you use printf or puts in a C++ program?
26
27    for (int k = 1; k < 5; k++)// declare a variable when you need it
28    {
29          cout << k;
30    }
31    //cout << k;
32    cout << endl;              // print a carriage return (newline)
33
34    cout << "Please enter your name => ";
35
36    char name[10];             // I feel like declaring a variable
37
38    cin >> name;
39
40    cout << "Hey " << name << ", nice name." << endl;
41
42    cout << endl;              // blank line
43
44    cout << "Hey " << name << ", how old are you? ";
```

```
45
46    int age;              // Declare another variable
47    cin >> age;
48
49    bool IsOld = age > 35;
50    bool IsYoung = !IsOld;
51    cout << IsOld << ' ' << IsYoung << endl;
52
53    if (IsOld) cout << name << ", you don't really look that old!\n;
54
55    char dogs_name[10];
56    int cats;
57
58    cout << "What's your dog's name and how many cats do you have? "
59         << endl;
60
61    cin >> dogs_name >> cats;
62
63    cout << "I'll bet " << dogs_name << " is a good dog and your "
64        << cats << " cat" << (cats>1?"s are":" is") << " nice too\n";
65
66    {
67         // This is a block
68         int x = 5; // x is local to this block
69         cout << x;
70    }
71
72    // cout << x;    What would happen if you tried to print x now?
73
74    return 0;
75  }
```

```
*****  Sample Run  *****

hey1234
Please enter your name => Joe
Hey Joe, nice name.

Hey Joe, how old are you? 34
0 1
What's your dog's name and how many cats do you have?
Bart 2
I'll bet Bart is a good dog and your 2 cats are nice too
5
```

Note: **cin** is similar to scanf(), but does not require conversion specifiers and whitespace is a separator for multiple variables. **cin** does not require the address operator (&), like scanf().

<< ("left-shift" in C) is called the insertion operator. >> ("right shift") is called the extraction operator.

Note on ***for loops*** with MS Visual C++ 6.0: First of all, you should not be using MS Visual C++, it's too old. But, if you insist, the following code does not work according to the C++ "standard". The "standard" specifies that **k** will only "have scope" for the ***for loop***, and after completion of the ***for loop***, **k** will be undefined. MS Visual C++ 6.0 doesn't see it that way. You've been warned!

```
for (int k = 1; k < 5; k++)
{
   cout << k;
}
```

Namespace std and the new Header filenames

The namespace keyword of C++ is used to group related data and functions. For example, if two sources provide a function called strcpy(). You may distinguish between them by prefacing strcpy() with the namespace, like VendorA::strcpy() or std::strcpy(). The namespace std is used to identify the standard ANSI/ISO symbols (functions, classes, and variables). The ANSI/ISO standards committee stipulated that standard header files would not have a filename extension. So, the header file, **iostream.h** will be identified as just **iostream**. The standard C header filenames, such as math.h, string.h, etc. will be prefaced with a **c** and the extension is dropped. Hence, **math.h** and **string.h** become **cmath** and **cstring**.

Example 1-2 namespace std and ANSI/ISO standard header files

```
1  // File: ex1-2.cpp - namespace std and the new header filenames
2
3  #include <iostream>
4  #include <cmath>
5  #include <cstring>
6  #include <cstdlib>
7  #include <cctype>
8  using namespace std;
9
10  // Create a namespace
11  namespace mystuff
12  {
13    int cout = 5;
14    double sqrt(double x)
15    {
16          return x / 2.0;
17    }
18  }
19
20  int main(void)
21  {
22    char cout[32] = "This is a bad idea";
23    char temp[80];
24    std::cout << "hey\n";
25    std::cout << "the square root of 2 is " << sqrt(2.) << endl;
26    strcpy(temp,"hello");
27    strcat(temp," there");
28    std::cout << strlen(temp) << temp << endl;
29    std::cout << atoi("4") << endl;
30    std::cout << toupper('a') << endl;
31    std::cout << (char)toupper('a') << endl;
32
33    std::cout << mystuff::cout << ' ' << cout << endl;
34
35    std::cout << sqrt(5.75) << ' ' << mystuff::sqrt(5.75)[1] << endl;
```

[1] Digital Mars C++ compiler (ver 8.42) produces a compile error on line 35 (suspect this is a bug).

```
36    return 0;
37  }
******  Program Output  ******

hey
the square root of 2 is 1.41421
11hello there
4
65
A
5 This is a bad idea
2.39792 2.875
```

Note that symbols default to their local definitions first, then to std definitions.

main() and the return type

The C++ standard specifies that main() must return an int. That is, you must define main() like

int main()
{
…
}

or

int main (void)
{
…
}

or

int main(int argc, char* argv[])
{
…
}

You may **not** define main() as

void main()
{
…
}

or

```
main()
{
…
}
```

You do not, however, have to end main() with a return statement. If the end of main() is reached without a return statement, a return of 0 is assumed.

The using directive and declaration

The keyword using is used as both a compiler directive and as a declaration. Throughout this text, the "using namespace std" directive directs the compiler to make available all of the "*std*" names. So, for example,

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

tells the compiler to recognize the names: cin, cout, endl, and others in whatever scope these two lines appear. Without the "**using namespace std;"** directive, the user would still have to qualify the the cin, cout, and endl identifiers as std::cin, std::cout, and std::endl;

Another approach is the using declaration, like this:

```
#include <iostream>
…
using std::cout;
using std::endl;

…

cout << …
```

Now, the user can use the identifiers cout and endl without the std namespace, but only those std identifiers. The using declaration adds an identifier to the current scope.

Introductory C++ Concepts

Reference Variables

A reference variable is an alias for another variable. It is similar to a pointer in that it contains the address of a variable, but unlike a pointer, you do not need to perform any dereferencing yourself.

Reference variables must be initialized when they are declared, and they cannot be reassigned to refer to another variable.

Examples:

```
float pie = 3.14;
float& apple = pie;
```

Here's a little piece of code that shows how a reference works, but it's not very realistic, because it doesn't make sense to use a reference for another variable in the same function.

```
int x = 5;
int &z = x;                          // z is another name for x

cout << x << endl;      -> prints 5
cout << z << endl;      -> prints 5


z = 9;                                          // same as x = 9;

cout << x << endl;      -> prints 9

cout << z << endl;      -> prints 9
```

Important rule: **You must initialize references**.

You may not declare a reference variable, like this:

```
int& ri;
```

Don't confuse it with the legal declaration of a pointer:

```
int* pi;
```

Guideline: **References are used as function arguments (or parameters) or return types.**

Example 2-1

This example compares two swap functions. p_swap is written as the traditional swap function (that you would write in C). r_swap is the equivalent version which uses references. Note the advantage of using references:

1. you don't have to pass the address of a variable
2. you don't have to dereference the variable inside the called function.

```
1  // File:  ex2-1.cpp Reference Variables - swap functions
2
3  #include <iostream>
4  using namespace std;
5
6  // Function prototypes (required in C++)
7  void pointerSwap(int*, int*);
8  void referenceSwap(int& i1, int& i2);
9
10  int main (void)
11  {
12    int x = 5;
13    int y = 7;
14
15    cout << x << ' ' << y << endl;
16
17    pointerSwap(&x,&y);
18
19    cout << x << ' ' << y << endl;
20
21    referenceSwap(x,y);
22
23    cout << x << ' ' << y << endl;
24
25    return 0;
26  }
27
28  void pointerSwap(int *a, int *b)
29  {
30    int temp;
31    temp = *a;
32    *a = *b;
33    *b = temp;
34  }
35
36  void referenceSwap(int &a, int &b)
37  {
38    int temp;
39    temp = a;
40    a = b;
41    b = temp;
42  }
```

```
*****  Output  *****

5 7
7 5                    <- line 19 output
5 7                    <- line 23 output
```

Program Analysis

This is an extensive analysis of this simple program. Probably more than you want to read about, but it should give you some ideas about how you should analyze an example. Some of the really obvious details were skipped, but you still thought about them.

Line 7 This, of course, is a prototype for a function that takes two pointer to int arguments. And, by the way what if the function prototype was written like?

```
void pointerSwap(int* n1, int* n2);                  or
void pointerSwap(int* a, int* b);                  or
void pointerSwap(int *n1, int *n2);                  or
void pointerSwap(int* n1, int *n2);                  or
void pointerSwap(int *, int *);                  or
```

It doesn't matter, they all mean the same thing. The * can be next to the int or next to a variable, or the variable doesn't even have to be there. This is just a style consideration.

Line 8 The int& function arguments mean that the arguments are **references** to int.

Line 17 Call the pointerSwap() function are pass in the addresses of two ints. Notice, that the was described (in the prototype discussion) as taking two pointer to int arguments, but when the function call is made here, you say that "the addresses of two ints" are passed in. You are, of course, familiar with this terminology from you C education.

Line 19 Here we see evidence that the pointer swap worked.

Line 21 In the call to the referenceSwap() function, note that the variable a just passed in (no addresses), just like a "pass by value". You know what that means, right?

Line 28 In the function definition heading, the function argument variable names do not have to match the variable names used in the function prototype. In the function argument, int *a, the * means that a is a pointer to an int, or the address of an int.

Line 31 *a, here means dereference a,or take the value stored at the pointer address.

Line 32 Notice, here, that *a can be used as an "L value". In the line above, it is used as an "R value".

Line 36 The argument, int &a (or int& a), means that **a** is a refererce to an int, or **a** refers to an int that exists elsewhere.

Line 39 Notice than when you use the reference, you don't need to defererence it. Herre's a secret – C++ accomplishes the reference utilization using pointers, but that none of your business!

It's often desirable to use a reference to a constant type to prevent changes to the referenced variable.

Example 2-2

This example shows function parameters passed as a reference and passed as a reference to const.

```
1  // File:  ex2-2.cpp     Reference Variables - function parameters
2
3  #include <iostream>
4  using namespace std;
5
6  void update_salary(double& sal)
7  {
8    sal *= 1.1;
9    return;
10  }
11
12  void display_salary(const double &sal)
13  {
14    cout << sal << endl;
15    return;
16  }
17
18
19  int main (void)
20  {
21    double salary = 50000.;
22
23    display_salary(salary);
24
25    update_salary(salary);
26
27    display_salary(salary);
28
29    return 0;
30  }
```

```
*****  Sample Run  *****

50000
55000
```

With reference variables, the & may be attached to either the variable type or the variable name as demonstrated in this example.

Example 2-3

This example illustrates the use of references as function parameters in an example that is a little more sophisticated. The purpose of the example is determine won, lost and tied statistics for some teams. The example is a little shallow, but it does make use of references to structs. Before you read the example, take a look at the sample program run at the end, so you get the gist of what it's trying to accomplish.

```
1  // File:  ex2-3.cpp   Reference Variables - function parameters
2
3  #include <iostream>
4  #include <cstring>
5  using namespace std;
6
7  const int     NumTeams = 5;    // better than #define NumTeams 5
8  const int     NumScores = 11;
9
10  // a team struct contains a team name and its W-L-T totals
11  struct team {
12    char       name[10];
13    unsigned   won;
14    unsigned   lost;
15    unsigned   tied;
16  };
17
18  // the league struct contains an array of team structs
19  struct league {
20    team       teams[NumTeams];
21  };
22
23  // this struct hold 2 team names and their points scored in a game
24  struct score {
25    char      teamname[2][10];
26    unsigned   points[2];
27  };
28
29  // function prototypes
30  void   initializeLeague(league& L);
31  void   enterScores(score* S);
32  int    getTeamNumFromName(league& L, const char* Name);
33  void   updateWonLostTied(league& L, score* S);
34  void   printLeagueStats(league& L);
35
36  int main (void) {
37    league     Birds;
38    score      Scores[NumScores];
39
40    initializeLeague(Birds);
41    enterScores(Scores);
42    updateWonLostTied(Birds,Scores);
```

```
43    printLeagueStats(Birds);
44
45    return 0;
46  }
47
48  // Assign team names and zero out won, lost, tied
49  void initializeLeague(league& L) {
50    for (int i = 0; i < NumTeams; i++)
51    {
52      cout << "Enter team name => ";
53      cin >> L.teams[i].name;
54      L.teams[i].won = 0;
55      L.teams[i].lost = 0;
56      L.teams[i].tied = 0;
57    }
58    cout << endl << NumTeams << " teams initialized\n\n";
59  }
60
61  void enterScores(score* S) {
62    cout << "Enter " << NumScores << " scores:\n";
63    for (int i = 0; i < NumScores; i++)
64    {
65      cout << "<team #1> <score #1> <team #2> <score #2> => ";
66      cin >> S[i].teamname[0] >> S[i].points[0]
67          >> S[i].teamname[1] >> S[i].points[1];
68    }
69    cout << endl << NumScores << " scores entered\n\n";
70  }
71
72  // getTeamNumFromName() returns the index of the teams array
73  // (in the league struct) in which Name matchs the value
74  // of the league.teams[i].name.  If no match is found, the
75  // function return -1
76  int getTeamNumFromName(league& L, const char* Name) {
77    for (int i = 0; i < NumTeams; i++)
78    {
79      if (strcmp(L.teams[i].name,Name) == 0) return i;
80    }
81    cerr << "Error: unable to find team: " << Name << endl;
82    return -1;       // team name not found
83  }
84
85  void updateWonLostTied(league& L, score* S) {
86    int        i,team0,team1;
87
88  for (i = 0; i < NumScores; i++) {
89      team0 = getTeamNumFromName(L,S[i].teamname[0]);
90      team1 = getTeamNumFromName(L,S[i].teamname[1]);
91
92  // if team name is bad, don't use the score
93      if (team0 == -1 || team1 == -1) continue;
```

```
94
95      if (S[i].points[0] > S[i].points[1]) {    // team0 won
96        L.teams[team0].won++;
97        L.teams[team1].lost++;
98      }
99      if (S[i].points[0] < S[i].points[1]) {    // team1 won
100       L.teams[team1].won++;
101       L.teams[team0].lost++;
102     }
103     if (S[i].points[0] == S[i].points[1]) {  // tie game
104       L.teams[team0].tied++;
105       L.teams[team1].tied++;
106     }
107   }
108 }
109
110 void  printLeagueStats(league& L) {
111   int  i;
112   cout << "\nName\tWon\tLost\tTied\n";
113   for (i = 0; i < NumTeams; i++) {
114     cout << L.teams[i].name << '\t'
115          << L.teams[i].won << '\t'
116          << L.teams[i].lost << '\t'
117          << L.teams[i].tied << '\n';
118   }
119 }
```

```
/************************  Sample Run  ****************************

Enter team name => coots
Enter team name => ducks
Enter team name => eagles
Enter team name => finches
Enter team name => geese

5 teams initialized

Enter 11 scores:
<team #1> <score #1> <team #2> <score #2> => coots 5 ducks 2
<team #1> <score #1> <team #2> <score #2> => coots 3 eagles 7
<team #1> <score #1> <team #2> <score #2> => coots 1 finches 0
<team #1> <score #1> <team #2> <score #2> => coots 0 geese 0
<team #1> <score #1> <team #2> <score #2> => ducks 1 eagles 4
<team #1> <score #1> <team #2> <score #2> => ducks 5 finches 2
<team #1> <score #1> <team #2> <score #2> => ducks 2 geese 1
<team #1> <score #1> <team #2> <score #2> => eagles 7 finches 7
<team #1> <score #1> <team #2> <score #2> => eagles 8 geese 5
<team #1> <score #1> <team #2> <score #2> => finches 3 geese 2
<team #1> <score #1> <team #2> <score #2> => gooses 9 ducks 2

11 scores entered
```

```
Error: unable to find team: gooses

Name    Won     Lost    Tied
coots   2       1       1
ducks   2       2       0
eagles  3       0       1
finches 1       2       1
geese   0       3       1

**********************************************************************/
```

Default Arguments

Default arguments is a shortcut that is not available in C. It allows function arguments to automatically be provided in the function call. Default arguments are commonly used when a function is called repeatedly using the same argument value(s)

Here's an example of a function, power(), that has a default argument.

Example 2-4

```
1  // File:  ex2-4.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  long power(int,int = 2); // function prototype with default argument
7
8  int main(void)
9  {
10    cout << power(5) << endl;         // use default argument
11    cout << power(2,10) << endl;      // don't use default argument
12    cout << power(12345) << endl;     // use default argument
13
14    return 0;
15  }
16
17
18  long power(int x, int y)
19  {
20    long num = 1;
21    for (int i = 1; i <= y; i++) num *= x;
22    return num;
23  }
```

```
*****  Sample Output  *****

25
1024
152399025
```

A function may possess several default arguments. Such as ...

```
void funk1 (int, double, int = 5, double = 3.14);

or

int funk2 (int = 1, int = 2, int = 3, int = 4);
```

Call's to funk1 could look like:

```
funk1(2,3.14,6,1.23)  // all arguments are supplied

or

funk1(2,3.14,6)            // the same as funk1(2,3.14,6,3.14)

or

funk1(2,3.14)              // the same as funk1(2,3.14,5,3.14)
```

Calls to funk2 could look like:

```
funk2(2,4,6,8)        // all arguments are supplied

or

funk2(2,4,6)          // the same as funk2(2,4,6,4)

or

funk2(2,4)            // the same as funk2(2,4,3,4)

or

funk2(2)              // the same as funk2(2,2,3,4)

or

funk2()               // all arguments are default
```

Notes

- In a function argument list, mandatory arguments may never follow default arguments. For example, funk(int,int,int=2,int=5) is OK, but funk(int,int=3,int,int=6) is not OK. Default arguments must come at the end of the argument list.

- Default arguments should be placed in the function prototype, not the function heading. This (strong) recommendation should be followed, even if your compiler permits the default argument in the function heading. The exception to this is the situation where the function is defined before it is called and, in this case, the prototype is not necessary.

- Default arguments may not be repeated in both the function prototype and the heading of the function definition.

Dynamic Memory Allocation

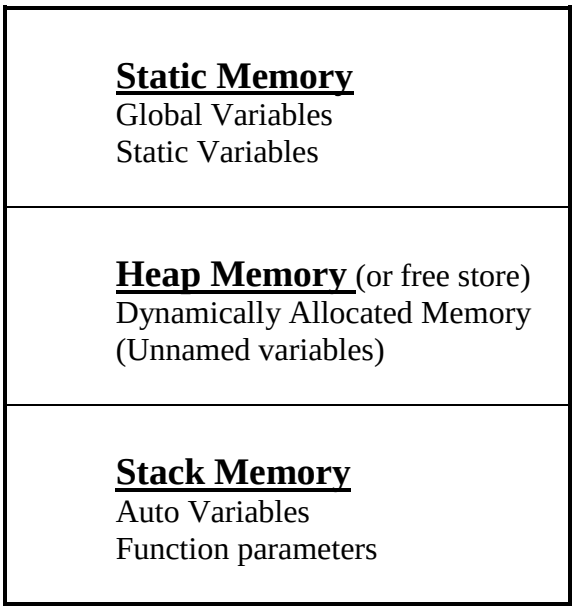
In C and C++ three types of memory are used by programs:

Static memory - where global and static variables live

Stack memory - "scratch pad" memory that is used by automatic variables.

Heap memory - (or free store memory) memory that may be dynamically allocated at execution time. This memory must be "managed". This memory is accessed using pointers.

Computer Memory



In C, the malloc(), calloc(), and realloc() functions are used to dynamically allocate memory from the **Heap**.

In C++, this is accomplished using the **new** and **delete** operators.

Dynamic memory allocation permits the user to create "variable-length" arrays, since only the memory that is needed may be allocated.

The new operator

new is used to allocate memory during execution time. **new** returns a pointer to the address where the object is to be stored. **new** always returns a pointer to the type that follows the **new.**

Example: allocate memory for 1 int

```
int *p;          // declare a pointer to int
p = new int;     // p points to the heap space allocated for the int
```

Example: allocate memory for a float value

```
float *f = new float;       // f points to a float in the heap space
```

More examples:

```
char* ptr_char = new char;

double *trouble = new double;

int** ptr_ptr_int = new int*;
```

```
struct employee_record
{
  char empno[7];
  char name[26];
  char orgn[5];
  float salary;
  ...
};
```

```
employee_record* harry = new employee_record;
```

- What is harry?

```
int *p = new int(6);        // allocated and assigns
```

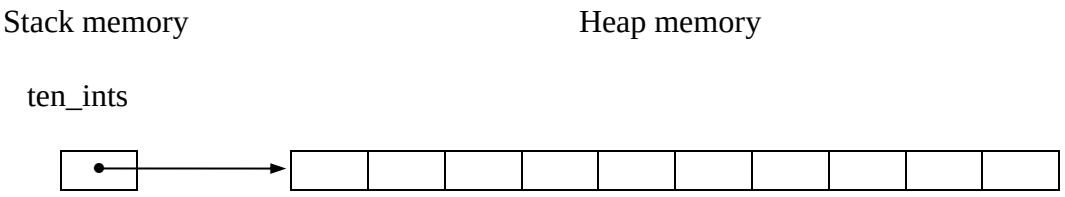
Dynamic Memory Allocation for Arrays

Example - allocate memory for 10 ints

```
int* ten_ints = new int[10];
```

ten_ints is a pointer to the first of 10 ints. They will be stored in contiguous memory, so that you can access the memory like an array. For example, ten_ints[0] is the address of the first int in heap memory, ten_ints[1] is the address of the second int and so on …

It sort of looks like this:



Type* pType = new Type[25];

Note: Even though you allocate memory for an array of Type with new, it always returns a pointer to the Type.

Example - allocate memory for a two-dimensional array

```
int (*p2d)[4] = new int[3][4];
```

Example - allocate memory for a string

```
char* text = new char[4];
strcpy(text,"hey");
```

If you attempt to dynamically allocate memory and it is not available, new will throw a **bad_alloc exception**. In pre-standard C++ **new** would return a value of 0 (or a null pointer), like malloc() in C, and most C++ programmers would use a test for 0 to check for failure of the allocation. Even though compiler manufacturers were slow to adopt this policy, most now conform to this standard. In this age of vast memory sizes, the failure of **new** is uncommon and more often than not, indicates a problem from a different source. Programmers are advised to adopt exception handling techniques (not covered in this course) for identification of this situation.

Note: **you may not initialize a dynamically allocated array as you do a single value**. Specifically,

```
int* pi = new int[5](0);          // this is illegal
```

The delete operator

The **delete** operator is used to release the memory that was previously allocated with new. The **delete** operator does not clear the released memory, nor does it change the value of the pointer that holds the address of the allocated memory. It is probably a good idea to set the pointer to the released memory to 0. To release memory for an array that was allocated dynamically, use [] (empty braces) after the **delete** operator.

Examples:

```
int *pi = new int;
…
delete pi;
double *pd = new double[100];
…
delete [] pd;
```

Example 2-5 - Dynamic memory allocation

```
1  // File:  ex2-5.cpp
2
3  #include <iostream>
4  #include <cstdlib>
5  #include <new>
6  using namespace std;
7
8  int main()
9  {
10
11    int i;
12    int* pint;
13    try {
14          pint = new int[99999];
15          cout << "memory is cheap\n";
16    }
17    // if the dynamic memory allocation fails, new throws a bad_alloc
18    catch (bad_alloc& uhoh) {
19          cerr << uhoh.what() << endl;    //displays "bad allocation"
20    }
21
22    for (i = 0; i < 99999; i++) pint[i] = 0;
23
24    delete [] pint;
25
26    pint = 0;
27  }
```

```
******  Output  ******
```

```
memory is cheap
```

Example 2-6 - Dynamic Memory Allocation for char arrays

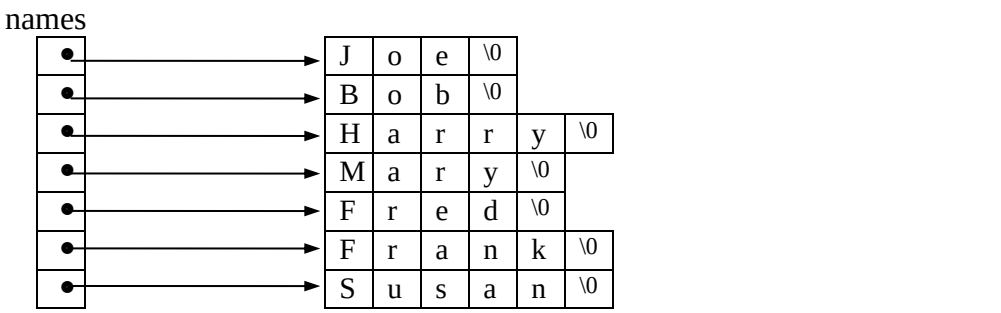
This example illustrates dynamically allocating memory to store char arrays. Storage for an array of pointers to the char arrays is not (but could be) allocated dynamically. Note each char array (name) can have a different length. Only the space required for each char array is allocated.

```
1  // File: ex2-6.cpp
2
3  #include <iostream>
4  #include <cstring>
5  using namespace std;
6
7  int main(void)
8  {
9     int i;
10    char * names[7];        // declare array of pointers to char
11    char temp[16];
12
13    // read in 7 names and dynamically allocate storage for each
14    for (i = 0; i < 7; i++)
15    {
16          cout << "Enter a name => ";
17          cin >> temp;
18          names[i] = new char[strlen(temp) + 1];
19
20          // copy the name to the newly allocated address
21          strcpy(names[i],temp);
22    }
23
24    // print out the names
25    for (i = 0; i < 7; i ++) cout << names[i] << endl;
26
27    // return the allocated memory for each name
28    for (i = 0; i < 7; i++) delete [] names[i];
29    return 0;
30  }
```

```
******  Sample Run  ******

Enter a name => Joe
Enter a name => Bob
Enter a name => Harry
Enter a name => Mary
Enter a name => Fred
Enter a name => Frank
Enter a name => Susan
Joe
Bob
Harry
```

```
Mary
Fred
Frank
Susan
```

The following illustrates the memory used in the last example:



Here's another solution for the last problem:

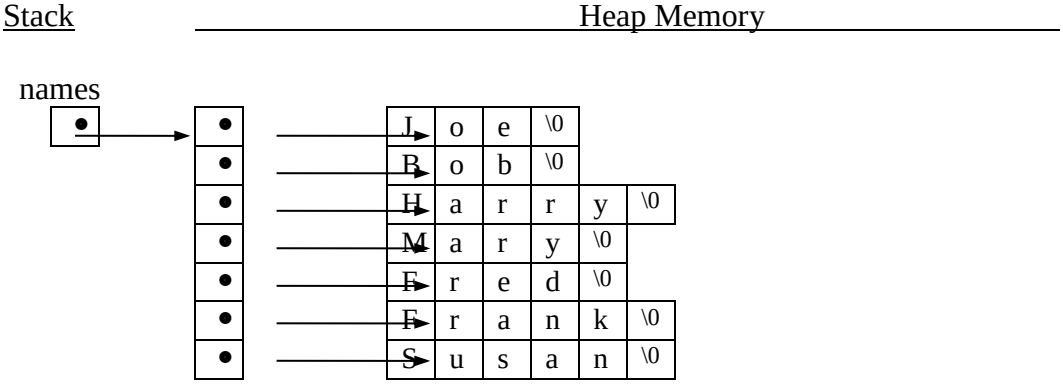
Example 2-7 - Dynamic Memory Allocation for char arrays

```
1  // File: ex2-7.cpp
2
3  #include <iostream>
4  #include <cstring>
5  using namespace std;
6
7  int main()
8  {
9     int i;
10    char ** names;        // declare pointer to pointer to char
11    char temp[16];
12    int NumberOfNames = 7;
13
14    names = new char*[NumberOfNames];
15
16    // read in 7 names and dynamically allocate storage for each
17    for (i = 0; i < NumberOfNames; i++)
18    {
19          cout << "Enter a name => ";
20          cin >> temp;
21          names[i] = new char[strlen(temp) + 1];
22
23          // copy the name to the newly allocated address
24          strcpy(names[i],temp);
25    }
26
27    // print out the names
```

```
28    for (i = 0; i < NumberOfNames; i ++) cout << names[i] << endl;
29
30    // return the allocated memory for each name
31    for (i = 0; i < NumberOfNames; i++) delete [] names[i];
32
33    delete [] names;
34  }
```

```
******  Sample Run  ******

Enter a name => Joe
Enter a name => Bob
Enter a name => Harry
Enter a name => Mary
Enter a name => Fred
Enter a name => Frank
Enter a name => Susan
Joe
Bob
Harry
Mary
Fred
Frank
Susan
```

Here is what memory looks like for this example:



- ✔ What happens on line 20 when the user enters a name longer than 16 characters?

Introduction to Classes

It is C++ classes that make possible encapsulation, data hiding, inheritance and what C++ is all about.

Classes are an extension of structures. A class is a user-defined type. Classes include both data members and member functions.

Structures

```
struct thing
{
  int a;
  char b;
  float c;
};
     .
     .
     .
int main (void)
{
  int i;
  thing x;
  thing y;
  thing stuff[10];
  thing * ptr_to_a_thing;
     .
     .
     .
  x.a = 5;
  y.c = 3.14;
     .
     .
     .
  ptr_to_a_thing = &y;

  i = (*ptr_to_a_thing).a;          // same as i = ptr_to_a_thing->a;
.
.
.
```

The Class Definition

Class members, functions and data, are identified with an access-specifier, **private**, **public**, or **protected**. Private and protected members may only be accessed by member functions of the class (except for friend functions). Public members may be accessed wherever they are "in scope".

Class member functions are defined using the class name and the **scope resolution operator :: .**

A class definition describes what a class is and what it does, or maybe what you can do to it. It consists of the keyword class, followed by a class name, opening and closing braces and a semicolon at the end. It may contain members, similar to a struct. The members are of two types, data and functions. It may also contain access specifiers, type definitions, nested classes, and possibly the specifications of friend functions.

```
class thing {
  private:                          // it's most common for data members
    int a;                          // to be private or protected
    char b;
    float c;
  protected:                    // protected members used with inheritance
    (some data and functions)
  public:
    void funk1(void);                   // member function prototypes
    void funk2(int);
    int  funk3(float,char,int);
};

// member function definitions

void thing::funk1(void)
{
     ...
}

void thing::funk2(int x)
{
     ...
}

int thing::funk3(float f, char ch, int i)
{
     ...
}
     .
int main (void)
{
  thing one, two, three;
  thing array[10];

  one.funk1();                                          // What is this?
  // one.a = 5;                                         // Why won't this work?
.
```

- What are one, two, three and array?

Class Examples

Example 3-1 - The circle class

```
1  // File:  ex3-1.cpp - the circle class
2
3  #include <iostream>
4  using namespace std;
5
6  class circle
7  {
8    private:
9      double radius;
10    public:
11     void store(double);
12     double area(void);
13     void display(void);
14 };
15
16 // member function definitions
17 void circle::store(double r)
18 {
19   radius = r;
20   return;
21 }
22
23 double circle::area(void)
24 {
25   return 3.14 * radius * radius;
26 }
27
28 void circle::display(void)
29 {
30   cout << radius << endl;
31   return;
32 }
33
34 int main(void)
35 {
36   circle c;   // instance (object) of circle class
37   c.store(5.0);
38   cout << "The area of circle c is " << c.area() << endl;
39   cout << "Circle c has radius ";
40   c.display();
41   return 0;
42 }
```

****** Output ******

```
The area of circle c is 78.5
Circle c has radius 5
```

Class Definition Notes

- The default access specifier is private, so if an access specifier does not appear before class members, then the members are private.

- Members, public, private, or protected, may appear in any order in the class definition. Further, access specifiers may be repeated within the class definition.

- Private members are not accessible (visible) to functions that are not members of the the class. This principle is referred to as data hiding.

- Member functions are defined after the class definition, or in a separate file, that "includes" the class definition.

- A semicolon is required at the end of each declaration, function or data member, within the class definition. An exception to this rule involves *implicit inline functions*. Further, it is possible to declare multiple data members of the same type using a comma to separate them.

- Data members are usually private or protected. A public data member does not have the protection of the class. Functions are may be public, private or protected. Public member functions are part of the class interface. Clients (other non-class functions) may only access class objects using the public member functions. Member functions may be private. A private member function may only be accessed by another member function of the same class. The exception to this is a friend function (to be discussed later).

- Classes and structs are the same in C++ with one exception. In classes, the default access specifier is private, in structs, it's public.

- Member functions are also called methods, behaviors or messages

- Variables or instances of a classes that are declared in a program are called objects. Member functions may be executed using a class object or by dereferencing a pointer to a class object. For example:

```
class X
{
  public:
    int funk();      // a member function
};
…
X ray;
X* pX;
pX = &ray;

ray.funk();    // execution of a member function using an object
pX->funk();    // execution of a member function using a pointer
               // to a class object
```

Example 3-2 - The triangle class

```
1  // File ex3-2.cpp - the triangle class
2
3  #include <iostream>
4  using namespace std;
5
6  class triangle
7  {
8    private:
9      double base;
10      double height;
11      double area;
12    public:
13      void store(double, double);
14      void calc_area(void);
15      void show(void);
16 };
17
18 // member function definitions
19 void triangle::store(double b, double h) {
20   base = b;
21   height = h;
22   return;          // "return" is optional
23 }
24
25 void triangle::calc_area(void) {
26   area = .5 * base * height;
27   return;
28 }
29
30 void triangle::show(void) {
31   cout << "base = " << base;
32   cout << "   height = " << height;
33   cout << "   area = " << area << endl;
34   return;
35 }
36
37 int main(void)
38 {
39   triangle t;           // an instance of the triangle class
40   t.store(1.23,4.55);   // initialize triangle t
41   t.calc_area();        // calculate the area of triangle t
42   t.show();             // display triangle t data
43   return 0;
44 }
```

```
******  Output  ******

base = 1.23   height = 4.55   area = 2.79825
```

Example 3-3 - The fraction Class

```
1  // File: ex3-3.cpp - the fraction class
2
3  #include <iostream>
4  #include <cstdlib>                                  // for abs()
5  using namespace std;
6
7  class fraction  {
8    public:
9      void set(int n, int d);
10      void display(void);
11      void add(fraction&, fraction&);
12      void subtract(fraction&, fraction&);
13      void multiply(fraction&, fraction&);
14      void divide(fraction&, fraction&);
15      void reduce(void);
16    private:
17      int numer;                                      // numerator
18      int denom;                                     // denominator
19 };
20
21 void fraction::set(int n, int d)
22 {
23   numer = n;
24   denom = d;
25   return;
26 }
27
28 void fraction::display(void)
29 {
30   cout << numer << '/' << denom << endl;
31   return;
32 }
33
34 void fraction::add(fraction& f1, fraction& f2)
35 {
36   numer = f1.numer * f2.denom + f2.numer * f1.denom;
37   denom = f1.denom * f2.denom;
38   return;
39 }
40
41 void fraction::multiply(fraction& f1, fraction& f2)
42 {
43   numer = f1.numer * f2.numer;
44   denom = f1.denom * f2.denom;
45 }
46
47 void fraction::reduce()
48 {
49   int min;                  // the minimum of the denom and numer
50   min = abs(denom) < abs(numer) ? abs(denom) : abs(numer);
```

```
51    for (int i = 2; i <= min; i++)
52    {
53      while ((abs(numer) % i == 0) && (abs(denom) % i == 0))
54      {
55        numer /= i;
56        denom /= i;
57      }
58    }
59    return;
60  }
61
62  int main(void)
63  {
64    fraction f,g,h;       // declare fractions f, g, and h
65    f.set(3,4);           // initialize fraction f & g
66    g.set(7,20);
67    f.display();          // display fraction f
68    g.display();
69    h.add(f,g);           // h = f + g
70    h.display();
71    h.reduce();           // reduce h
72    h.display();
73    h.multiply(f,g);      // h = f * g
74    h.display();
75    int i,j;
76    cout << "Enter a fraction numerator and denominator => ";
77    cin >> i >> j;
78    h.set(i,j);
79    h.multiply(h,h);
80    cout << i << '/' << j << " squared is ";
81    h.display();
82    return 0;
83  }
```

```
******  Sample Run  ******

3/4
7/20
88/80
11/10
21/80
Enter a fraction numerator and denominator => 2 3
2/3 squared is 4/9
```

- ✔ Why use fraction& arguments instead of fraction?

- ✔ Should add's arguments be const fraction& instead of fraction&?

What if you tried to add a fraction to itself? f1.add(f1,f2) ?

How would you write the subtract and divide member functions?

Here's another implementation of the reduce() function for example 3-3. It implements the Euclidean Algorithm for determining the greatest common divisor of the fraction's numberator and denominator.

```
1  void swap(int& x, int& y)
2  {
3     int temp;
4     temp = x;
5     x = y;
6     y = temp;
7  }
8
9  int gcd (int x, int y)
10  {
11    if (x < y) swap(x,y);
12    int rem = x % y;
13    while (rem > 0) {
14          return gcd(y,rem);          // Note recursive call to gcd()
15    }
16    return y;
17  }
18
19  void fraction::reduce(void)
20  {
21    int divisor = gcd(numer,denom);
22    numer /= divisor;
23    denom /= divisor;
24    return;
25  }
```

Inline Functions

Functions, either class members or non-class member functions, may be defined as inline. An inline function is one in which the function code replaces the function call directly. Specifying that a function be inline is a request to the compiler that it be inline, but the compiler may choose not to make it an inline function. This is transparent to the user. Inline functions should be short (preferable one-liners).

Inline class member functions may be implicit, if they are defined as part of the class definition, or explicit if they are defined outside of the class definition using the keyword, inline.

Inline functions have only internal linkage, that is, they are local to the file in which they are defined.

Example

```
class xyz
{
  private:
     .
     .
     .
  public:
    void funk1(void) { cout  << "Have a nice day\n"; return; }
    void funk2(void);
     .
     .
};

inline void xyz::funk2(void)
{
  cout << "Ok, don't have a nice day\n";
  return;
}
     .
     .

int main(void)
{
  xyz a,b;
  a.funk1();
  b.funk2();
     .
     .
```

funk1 is an implicit inline function. funk2 is an explicit inline function.

Example 3-4 - The Clock class

```
1  // File: ex3-4.cpp - the Clock class
2
3  #include <iostream>
4  using namespace std;
5
6  class Clock
7  {
8    private:
9      int hours;
10     int minutes;
11     int seconds;
12   public:
13     void set(int h, int m, int s)
14       {hours = h; minutes = m; seconds = s; return;}     // inline
15     void increment(void);
16     void display(int=0) const;
17 };
18
19
20 void Clock::increment (void)
21 {
22   seconds++;
23   minutes += seconds/60;
24   hours += minutes/60;
25   seconds %= 60;
26   minutes %= 60;
27   hours %= 24;
28   return;
29 }
30
31
32 void Clock::display(int format) const
33 {
34   if (format) {              // use format hh:mm:ss AM/PM
35     cout << (hours % 12 ? hours % 12:12) << ':'
36          << (minutes < 10 ? "0" :"") << minutes << ':'
37          << (seconds < 10 ? "0" :"") << seconds
38          << (hours < 12 ? " AM" : " PM") << endl;
39   }
40   else {                     // use format hh:mm:ss (24 hour time)
41     cout << (hours < 10 ? "0" :"") << hours << ':'
42          << (minutes < 10 ? "0" :"") << minutes << ':'
43          << (seconds < 10 ? "0" :"") << seconds << endl;
44   }
45 }
46
47
48 int main(void)
49 {
```

```
50    Clock c;
51    c.set(23,59,55);
52    for (int i = 0; i < 10; i++) {
53      c.increment();
54      c.display();
55      c.display(1);
56      cout << endl;
57    }
58    return 0;
}
```

```
****** Output  ******

23:59:56
11:59:56 PM

23:59:57
11:59:57 PM

23:59:58
11:59:58 PM

23:59:59
11:59:59 PM

00:00:00
12:00:00 AM

00:00:01
12:00:01 AM

00:00:02
12:00:02 AM

00:00:03
12:00:03 AM

00:00:04
12:00:04 AM

00:00:05
12:00:05 AM
```

Example 3-5 - The Date class

```
1  // File:  ex3-5.cpp - The Date class
2
3  #include <iostream>
4  #include <cstring>
5  #include <cstdlib>
6  using namespace std;
7
8  const unsigned DaysPerMonth[] =
9     {31,28,31,30,31,30,31,31,30,31,30,31};
10
11  class Date
12  {
13    unsigned day;
14    unsigned month;
15    unsigned year;
16    void errmsg(const char* msg);
17  public:
18    void set(const char* mmddyy);
19    void increment(void);
20    void display(void) const;
21  };
22
23  void Date::set(const char* mm_dd_yy)
24  {
25    char* temp;
26    char copy[9];
27
28    // assume user enters date as mm/dd/yy
29    if (strlen(mm_dd_yy) != 8) errmsg(mm_dd_yy);
30
31    // use a copy of mm_dd_yy  What is the impact to the function?
32    strcpy(copy,mm_dd_yy);
33
34    // parse the date and get the month
35    temp = strtok(copy,"/");    // strtok() replaces "/" with a NULL
36    if (temp != NULL) month = atoi(temp);
37    else errmsg(copy);
38
39    // parse the date and get the day
40    temp = strtok(NULL,"/");    // strtok() finds the next "/"
41    if (temp != NULL) day = atoi(temp);
42    else errmsg(copy);
43
44    // parse the date and get the year
45    temp = strtok(NULL,"/");
46    if (temp != NULL) year = atoi(temp);
47    else errmsg(copy);
48
49    // Make a Y2K correction for a 2-digit year
```

```
50    if (year < 50) year += 2000;
51    else if (year < 100) year += 1900;
52    else ;      // assume the year is right
53  }
54
55  void Date::increment (void)
56  {
57    // increment the day
58    day++;
59
60    // check for the end of the month
61    if (day > DaysPerMonth[month - 1])  // past end of current month?
62    {
63          month ++;
64          day = 1;
65    }
66
67    // check for the end of the year
68    if (month > 12)
69    {
70          year ++;
71          month = 1;
72    }
73
74    return;
75  }
76
77  void Date::display(void) const
78  {
79    cout << "The date is " << month << '/' << day << '/'
80          << (year%100< 10?"0":"") << year%100 << endl;
81    if (day % DaysPerMonth[month-1] == 0) cout << endl;
82    return;
83  }
84
85  void Date::errmsg(const char* msg)
86  {
87    cerr << "Invalid date format: " << msg << endl;
88    exit(EXIT_FAILURE);
89  }
90
91  int main(void)
92  {
93    Date d;
94    char mmddyy[9];
95    cout << "Enter the starting date <mm/dd/yy> => ";
96    cin >> mmddyy;
97    d.set(mmddyy);
98    for (int i = 0; i < 375; i++)
99    {
100         d.display();
```

```
101         d.increment();
102    }
103    return 0;
104   }
```

```
******  Sample Output #1  ******

Enter the starting date <mm/dd/yy> => 4/20/09
Invalid date format: 1/20/09


******  Sample Output #2  ******

Enter the starting date <mm/dd/yy> => 04/20/09
The date is 4/20/09
The date is 4/21/09
The date is 4/22/09
The date is 4/23/09
The date is 4/24/09
The date is 4/25/09
The date is 4/26/09
The date is 4/27/09
The date is 4/28/09
The date is 4/29/09
The date is 4/30/09

The date is 5/1/09
The date is 5/2/09
The date is 5/3/09
The date is 5/4/09
The date is 5/5/09
The date is 5/6/09
The date is 5/7/09
The date is 5/8/09
The date is 5/9/09
The date is 5/10/09
…
The date is 12/29/09
The date is 12/30/09
The date is 12/31/09

The date is 1/1/10
The date is 1/2/10
The date is 1/3/10
The date is 1/4/10
The date is 1/5/10
The date is 1/6/10
…
The date is 4/24/10
```

```
The date is 4/25/10
The date is 4/26/10
The date is 4/27/10
The date is 4/28/10
The date is 4/29/10
```

- What is the effect of making errmsg() private in the class?

- How does strtok() work?

  Is there a difference if line 40 is coded like this? *temp = strtok(NULL,"/");*
  No, not really, in C++ the macro NULL is defined as (integer) 0.

- How would you make this program work for leap years?

- How would you change display() to print the date with a two-digit month and day?

- What is atoi() and how does it work? Why is it not considered "safe"?

Practice problem: Modify this program to print the date as "m/dd/yy" and to handle leap years?

Example 3-6 - The quadratic (Equation) class

```
1  // File:  ex3-6.cpp - the quadratic class
2
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6
7  class quadratic
8  {
9  private:
10      float a, b, c;
11  public:
12      void set(float f1, float f2, float f3)
13                           {a = f1; b = f2; c = f3; return;}
14      void solve(void) const;
15      void display(void) const;
16  };
17
18  void quadratic::solve (void) const
19  {
20    bool complex_roots;
21    float radical_stuff;
22    if (a == 0)
23    {
24          cout <<
25            "This is not a quadratic equation.  It has only one root "
26               << -c/b << endl;
27          return;
28    }
29    complex_roots = (radical_stuff = b * b - 4 * a * c) < 0;
30    cout << "The roots of the equation are ";
31    if (!complex_roots)
32          cout << ((-b+sqrt(radical_stuff))/(2.*a))
33          <<" and " << ((-b-sqrt(radical_stuff))/(2.*a)) << endl;
34    else
35          cout << (-b/(2.*a)) <<'+'<< sqrt(-
   radical_stuff)/(2.*a)<<'i'
36          <<
37          " and "
38          << (-b/(2.*a)) << '-' << sqrt(-radical_stuff)/(2.*a) << 'i'
39          <<endl;
40    return;
41  }
42
43  void quadratic::display(void) const
44  {
45    cout << "The coefficients of the quadratic equations are: "
46          << a << ", " << b << ", and " << c <<endl;
47    return;
```

```
48  }
49
```

```
50  int main(void)
51  {
52    quadratic equation;
53    float a,b,c;
54    while (1)
55    {
56          cout <<
57          "Enter 3 coeffients for a quadratic equation (or quit) => ";
58          if (!(cin >> a >> b >> c)) break;
59          equation.set(a,b,c);
60          equation.display();
61          equation.solve();
62          cout << endl;
63    }
64    return 0;
65  }
```

```
******  Sample Run  ******

Enter 3 coefficients for an equation (or quit) => 1 2 1
The coefficients of the quadratic equations are: 1, 2, and 1
The roots of the equation are -1 and -1

Enter 3 coefficients for an equation (or quit) => 1 0 -1
The coefficients of the quadratic equations are: 1, 0, and -1
The roots of the equation are 1 and -1

Enter 3 coefficients for an equation (or quit) => 1 2 3
The coefficients of the quadratic equations are: 1, 2, and 3
The roots of the equation are -1+1.414214i and -1-1.414214i

Enter 3 coefficients for an equation (or quit) => 1 1 1
The coefficients of the quadratic equations are: 1, 1, and 1
The roots of the equation are -0.5+0.866025i and -0.5-0.866025i

Enter 3 coefficients for an equation (or quit) => 0 3 6
The coefficients of the quadratic equations are: 0, 3, and 6
This is not a quadratic equation. It has only one root -2
```

Enter 3 coefficients for an equation (or quit) => **quit**

- ✔ How does the line: ***if (!(cin >> a >> b >> c)) break;*** work?

Example 3-7 - The card and deck classes

This example demonstrates a container relationship between classes. This is also called containment.

```
1  // File: ex3-7.cpp - card and deck classes
2
3  #include <iostream>
4  #include <cstdlib>                  // needed for rand() function
5  using namespace std;
6
7  const char* value_name[13] = {"two","three","four","five","six",
8    "seven","eight","nine","ten","jack","queen","king","ace"};
9
10 const char* suit_name[4] = {"clubs","diamonds","hearts","spades"};
11
12 class card
13 {
14   private:
15       int value;
16       int suit;
17   public:
18       void assign(int);
19       int get_value(void) const;             // accessor function
20       int get_suit(void) const;              // accessor function
21       void print(void) const;
22 };
23
24 void card::assign(int x)
25 {
26   value = x % 13;
27   suit = x / 13;
28   return;
29 }
30
31 int card::get_value(void) const
32 {
33   return value;
34 }
35
36 int card::get_suit(void) const
37 {
38   return suit;
39 }
40
41 void card::print(void) const
42 {
43   cout << (value_name[value]) << " of "
44       << (suit_name[suit]) << endl;
45   return;
46 }
```

```
47  class deck
48  {
49    private:
50        card d[52];
51        int next_card;
52    public:
53        void create_deck(void);
54        void shuffle(void);
55        void deal(int=5);
56        void print(void) const;
57  };
58
59
60  void deck::create_deck(void)
61  {
62    for (int i = 0; i < 52; i++) d[i].assign(i);
63    next_card = 0;
64    return;
65  }
66
67
68  void deck::shuffle(void)
69  {
70    int i, k;
71    cout << "I am shuffling the deck\n";
72    for (i = 0; i < 52; i++)
73    {
74        k = rand() % 52;
75        card temp = d[i];
76        d[i] = d[k];
77        d[k] = temp;
78    }
79    return;
80  }
81
82
83  void deck::print(void) const
84  {
85    cout << "\nHere's the deck:\n";
86    for (int i = 0; i < 52; i++) d[i].print();
87    return;
88  }
89
90  void deck::deal(int no_of_cards)
91  {
92    cout <<"\nOk, I will deal you "<<no_of_cards<<" cards:\n";
93    for (int i=0; i<no_of_cards; i++) d[next_card++].print();
94    return;
95  }
```

```
96  int main(void) {
97    deck poker;
98    poker.create_deck();
99    poker.print();
100    poker.shuffle();
101    poker.print();
102    poker.deal();
103    poker.deal(3);
104    return 0;
105  }
```

```
******  Output  ******

Here's the deck:
two of clubs
three of clubs
four of clubs
five of clubs
six of clubs
seven of clubs
eight of clubs
nine of clubs
     .    .
     .    .
ace of spades
I am shuffling the deck

Here's the deck:
ten of hearts
ace of diamonds
queen of clubs
three of diamonds
four of spades
eight of spades
eight of diamonds
     .    .
     .    .
Ok, I will deal you 5 cards:
ten of hearts
ace of diamonds
queen of clubs
three of diamonds
four of spades

Ok, I will deal you 3 cards:
eight of spades
eight of diamonds
eight of clubs
```

const member functions

A **const member function** is a class member function that may not make any changes to any of the data members of the class. A const member function is identified by appending the keyword, **const**, to **both** the prototype and the heading of the function definition.

The following example demonstrates const member functions. It constains several compilation errors.

```
1  // File: ex3-8.cpp - const member functions
2
3  void makeit6(int& I)
4  {
5     I = 6;
6  }
7
8
9  class ABC
10  {
11    int x;
12  public:
13    void funk();
14    void gunk() const;               // const member function
15    void hunk(int&);
16    void junk(int&) const;           // const member function
17    void lunk(const int&);
18    void munk(const int&) const;     // const member function
19  };
20
21  void ABC::funk()
22  {
23    x = 6;           // ok
24    makeit6(x);      // ok
25    makeit6(6);      // error: cannot convert const int to int&
26  }
27
28  void ABC::gunk() const
29  {
30    x = 6;           // error: cannot change data member in CMF
31    makeit6(x);      // error: cannot pass const int as int&
32  }
33
34  void ABC::hunk(int &I)
35  {
36    x = 6;           // ok
37    makeit6(I);      // ok
38    makeit6(x);      // ok
39  }
40
41  void ABC::junk(int &I) const
```

```
42  {
43    x = I;           // error: cannot change data member in CMF
44    makeit6(I);      // ok
45    makeit6(x);      // error: cannot pass const int as int&
46  }
47
48
49  void ABC::lunk(const int &I)
50  {
51    x = I;           // ok
52    makeit6(I);      // error: cannot pass const int& as int&
53    makeit6(x);      // ok
54  }
55
56  void ABC::munk(const int &I) const
57  {
58    x = I;           // error: cannot change data member in CMF
59    makeit6(I);      // error: cannot pass const int& as int&
60    makeit6(x);      // error: cannot pass const int as int&
61  }
62
63
64  int main()
65  {
66    ABC object;
67    int i = 3;
68    object.funk();
69    object.gunk();
70    object.hunk(i);
71    object.junk(i);
72    object.lunk(i);
73    object.munk(i);
74
75    return 0;
76  }
```

```
No output – compile errors
```

- ✔ Make sure that you are clear on the difference between a const member function and one with a reference to const argument. Note: in example 3-8 above, lunk() is not a const member function. It is a member function with a reference to const argument.

mutable

You haven't heard the entire truth about const member functions. The definition stated earlier was, "A **const member function** is a class member function that may not make any changes to any of the data members of the class". Actually, there is an exception to this. The keyword, **mutable**, allows the user to supersede the const intension. A class member that is defined using the storage specifier, mutable, may be changed in a const member function. Here is example 3-8 again with a mutable class member:

```
1  // File: ex3-8m.cpp - const member function with a mutable member
2
3  void makeit6(int& I)
4  {
5     I = 6;
6  }
7
8
9  class ABC
10  {
11    mutable int x;
12  public:
13    void funk();
14    void gunk() const;               // const member function
15    void hunk(int&);
16    void junk(int&) const;           // const member function
17    void lunk(const int&);
18    void munk(const int&) const;     // const member function
19  };
20
21  void ABC::funk()
22  {
23    x = 6;                 // ok
24    makeit6(x);            // ok
25    makeit6(6);            // error: cannot convert const int to int&
26  }
27
28  void ABC::gunk() const
29  {
30    x = 6;                 // ok
31    makeit6(x);            // ok
32  }
33
34  void ABC::hunk(int &I)
35  {
36    x = 6;                 // ok
37    makeit6(I);            // ok
38    makeit6(x);            // ok
39  }
40
41  void ABC::junk(int &I) const
```

```
42  {
43    x = I;                // ok
44    makeit6(I);           // ok
45    makeit6(x);           // ok
46  }
47
48
49  void ABC::lunk(const int &I)
50  {
51    x = I;                // ok
52    makeit6(I);           // error: cannot pass const int& as int&
53    makeit6(x);           // ok
54  }
55
56  void ABC::munk(const int &I) const
57  {
58    x = I;                // ok
59    makeit6(I);           // error: cannot pass const int& as int&
60    makeit6(x);           // ok
61  }
62
63
64  int main()
65  {
66    ABC object;
67    int i = 3;
68    object.funk();
69    object.gunk();
70    object.hunk(i);
71    object.junk(i);
72    object.lunk(i);
73    object.munk(i);
74
75    return 0;
76  }
```

```
No output – compile errors
```

mutable should be used in a situation where you want a member function to only be able to change a few of the class members, not all of them.

Classes Containing Enumerated Types

Example 3-9 - Enums and classes

```
1  // File: ex3-9.cpp - enums and classes
2
3  #include <iostream>
4  using namespace std;
5
6  enum Size {small, medium, large};        // global enum
7
8  class Thing
9  {
10   public:
11     enum Color {red, white, blue};
12     enum {FALSE, TRUE };                 // anonymous enum
13     void setBigness(Size=small);
14     void setHue(Color=red);
15     Size getBigness() const;
16     Color getHue() const;
17     int amIBlue() const
18      { if (hue == blue) return TRUE; else return FALSE; }
19   private:
20     Size bigness;
21     Color hue;
22 };
23
24 void Thing::setBigness(Size s)
25 {
26   bigness = s;
27 }
28
29 void Thing::setHue(Color c)
30 {
31   hue = c;
32 }
33
34 Size Thing::getBigness() const {
35  return bigness;
36 }
37
38 Thing::Color Thing::getHue() const {
39   return hue;
40 }
41
42
43 Thing::Color nonMember(const Thing&); // function prototype
44
45 int main()
46 {
47   Size S = large;
```

```
48    Thing::Color C = Thing::white;
49
50    Thing bigRedThing;
51    bigRedThing.setBigness(S);
52    bigRedThing.setBigness(medium);
53    bigRedThing.setHue();
54    cout << "I am"
55         << (bigRedThing.amIBlue() ? " " : " not ")
56         << "blue\n";
57
58    Thing littleBlueThing;
59    littleBlueThing.setBigness();
60  //  littleBlueThing.setHue(blue);
61    littleBlueThing.setHue(Thing::blue);
62    cout << "I am"
63         << (littleBlueThing.amIBlue() ? " " : " not ")
64         << "blue\n";
65    littleBlueThing.setHue(C);
66    cout << "I am"
67         << (littleBlueThing.amIBlue() ? " " : " not ")
68         << "blue\n";
69
70    nonMember(bigRedThing);
71  }
72
73  Thing::Color nonMember(const Thing& T)
74  {
75     return T.getHue();
76  }
```

```
******  Output  ******

I am not blue
I am blue
I am not blue
```

Note: An enumerated type defined within a class is subject to access specifiers, just like data members or member functions. In other words, if the color type in the thing class of this example is specified as private, then that type is not visible except to class member functions.

Nested Classes

This example illustrates nested classes. Note the use of the scope resolution operator in the member function definitions. This example is logically the same as example 3-7.

Example 3-10 - Nested Classes

```
1  // File: ex3-10.cpp - the card and deck example with nested classes
2
3  #include <iostream>
4  #include <stdlib>
5  using namespace std;
6
7  const char* value_name[13] =  {"two","three","four","five","six",
8     "seven","eight","nine","ten","jack","queen","king","ace"};
9  const char* suit_name[4] =
10     {"clubs","diamonds","hearts","spades"};
11
12
13  class deck
14  {
15    public:                                  // Why is this public?
16      class card
17      {
18        private:
19          int value;
20          int suit;
21        public:
22          void assign(int);
23          int get_value(void) const;
24          int get_suit(void) const;
25          void print(void) const;
26      };
27
28    private:
29      card d[52];
30      int next_card;
31    public:
32      void create_deck(void);
33      void shuffle(void);
34      void deal(int=5);
35      void print(void) const;
36  };
37
38
39  int deck::card::get_value(void) const
40  {
41    return value;
42  }
43
44  int deck::card::get_suit(void) const
```

```
45  {
46    return suit;
47  }
48
49  void deck::card::assign(int x)
50  {
51    value = x % 13;
52    suit = x / 13;
53  }
54
55  void deck::card::print(void) const
56  {
57    cout << (value_name[value]) << " of "
58         << (suit_name[suit]) << endl;
59    return;
60  }
61
62  void deck::create_deck(void)
63  {
64    for (int i = 0; i < 52; i++) d[i].assign(i);
65    next_card = 0;
66  }
67
68  void deck::shuffle(void)
69  {
70    int i, k;
71    card temp;
72    cout << "I am shuffling the deck\n";
73    for (i = 0; i < 52; i++)
74    {
75      k = rand() % 52;
76      temp = d[i];
77      d[i] = d[k];
78      d[k] = temp;
79    }
80  }
81
82  void deck::print(void) const
83  {
84    cout << "\nHere's the deck:\n";
85    for (int i = 0; i < 52; i++) d[i].print();
86  }
87
88  void deck::deal(int no_of_cards)
89  {
90    cout <<"\nOk, I will deal you "<<no_of_cards<<" cards:\n";
91    for (int i=0;i<no_of_cards; i++)  d[next_card++].print();
92    return;
93  }
```

```
94  int main (void)
95  {
96  //  card C;                  // Why is this commented out?
97    deck::card C;              // Instatiate a card object
98    C.assign(17);
99    C.print();                 // prints six of diamonds
100
101    deck poker;
102    poker.create_deck();
103    poker.print();
104    poker.shuffle();
105    poker.print();
106    poker.deal();
107    poker.deal(3);
108    return 0;
109  }
```

```
******  Output same as Example 3-7 ******
```

Note: A nested class, like the card class of this example, is subject to access specifiers, just like data members or member functions. In other words, if the card class within the deck class of this example is specified as private, then the card type is not visible except to deck class member functions.

Multi-File C++ Programs

It is common practice in larger C++ programs to separate the application into multiple files. The files typically consist of one or more header files, function definition files, and a file to hold main(). The header files generally contain class definitions, constants, enumerated types, structures, typedefs, and inline function definitions. Header files have the same name as the class or a name related to the application and have the extension .h (.hpp on some compilers). Class member function definitions are contained in another file(s). The filename is often the same name as the class name and extension .cpp (or whatever the requirement for your compiler). This file(s) should "include" the header file(s) and may be compiled separately. The file containing main() will usually have the name of the application and the usual C++ extension. main() will need to include the header file(s) and after compilation will link to the compiled class member definition file(s). This process is managed for you in many of the PC and Mac compilers with projects.

This example is logically the same as Example 3-7.

Example 3-11 - A Multi-file program

```
1  // File: ex3_11c.h - card class definition
2
3  #ifndef EX3_11C_H
4  #define EX3_11C_H
5
6  class card
7  {
8    private:
9      int value;
10      int suit;
11    public:
12      void assign(int);
13      int get_value(void) const;
14      int get_suit(void) const;
15      void print(void) const;
16  };
17
18  #endif
```

Note: some compilers do not support header files with a hyphen in the file name.

```
1  // File: ex3_11d.h - deck class definition
2  #ifndef EX3_11D_H
3  #define EX3_11D_H
4  #include "ex3_11c.h"
5
6  class deck
7  {
8    private:
9        card d[52];
10        int next_card;
11    public:
12        void create_deck(void);
13        void shuffle(void);
14        void deal(int=5);
15        void print(void) const;
16  };
17  #endif
```

```
1  // File:  ex3-11c.cpp  card class member function definitions
2
3  #include <iostream>
4  using namespace std;
5
6  #include "ex3_11c.h"
7
8  const char* value_name[13] =
9   {"two","three","four","five","six",
10    "seven","eight","nine","ten","jack","queen","king","ace"};
11  const char* suit_name[4] =
12   {"clubs","diamonds","hearts","spades"};
13
14  card::get_value(void) const {
15    return value;
16  }
17
18  int card::get_suit(void) const {
19    return suit;
20  }
21
22  void card::assign(int x) {
23    value = x % 13;
24    suit = x / 13;
25    return;
26  }
27
28  void card::print(void) const {
29    cout << (value_name[value]) << " of "
30        << (suit_name[suit]) << endl;
31    return;
32  }
```

```
1  // File: ex3-11d.cpp - deck class member function definitions
2
3  #include <iostream>
4  #include <cstdlib>            // needed for rand() function
5  using namespace std;
6
7  #include "ex3_11d.h"
8
9  void deck::create_deck(void) {
10    for (int i = 0; i < 52; i++) d[i].assign(i);
11    next_card = 0;
12  }
13
14  void deck::shuffle(void) {
15    int i, k;
16    card temp;
17    cout << "I am shuffling the deck\n";
18    for (i = 0; i < 52; i++)   {
19        k = rand() % 52;
20        temp = d[i];
21        d[i] = d[k];
22        d[k] = temp;
23    }
24  }
25
26  void deck::print(void) const {
27    cout << "\nHere's the deck:\n";
28    for (int i = 0; i < 52; i++) d[i].print();
29  }

30  void deck::deal(int no_of_cards) {
31    cout <<"\nOk, I will deal you "<<no_of_cards<<" cards:\n";
32    for (int i = 0;i<no_of_cards; i++) d[next_card++].print();
33  }
```

```
1  // File: ex3-11.cpp - main()
2
3  #include "ex3_11d.h"
4
5  int main (void) {
6    deck poker;
7    poker.create_deck();
8    poker.print();
9    poker.shuffle();
10    poker.print();
11    poker.deal();
12    poker.deal(3);
13    return 0;
14  }
```

The output is the same as examples 3-7.

Command-line Compilation

Microsoft Visual C++ 2008 compiler

Before you can perform a command-line compile, you must run **vcvars32.bat**. This program and the **cl.exe** for the command-line compile are found in the directory: \Program Files\Microsoft Visual Studio 9.0\VC\bin

```
C:\deanza\cis27\examples>vcvars32

C:\deanza\cis27\examples>"c:\Program Files\Microsoft Visual Studio
9.0\Common7\Tools\vsvars32.bat"
Setting environment for using Microsoft Visual Studio 2008 x86 tools.

C:\deanza\cis27\examples>cl ex3-11.cpp ex3-11c.cpp ex3-11d.cpp /EHsc
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.21022.08 for
80x86
Copyright (C) Microsoft Corporation.  All rights reserved.

ex3-11.cpp
ex3-11c.cpp
ex3-11d.cpp
Generating Code...
Microsoft (R) Incremental Linker Version 9.00.21022.08
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:ex3-11.exe
ex3-11.obj
ex3-11c.obj
ex3-11d.obj
```

GNU compiler

```
g++ ex3-11.cpp ex3-11c.cpp ex3-11d.cpp -Wall
```

Constructors and Destructors

Constructors are special class member functions that are used to create class objects. They execute automatically when an instance of a class is created. Constructors are used to initialize class data members. They are also used to allocate memory for a class. A constructor's name is the same as the class name.

Destructors are functions that also execute automatically when the object goes out of scope (existence). Destructors, too, have a special name. It is the class name preceded by a ~ (tilde). Destructors are used to release dynamically allocated memory and to perform other "cleanup" activities.

Example

```
class xyz
{
  private:
      .
      .
  public:
    xyz();          // constructor prototype
    ~xyz();         // destructor prototype
      .
      .
};

xyz::xyz()                                // constructor definition
{
      .
      .
}

xyz::~xyz()                                  // destructor definition
{
      .
      .
}
      .
      .
      .
int main(void)
{
  xyz thing;       // the construction is called now
      .
      .
      .
  return 0;         // the destructor is called now
}
```

Constructor/Destructor Notes

- Constructors and destructors are usually placed in the public part of class definition.
- Both the constructor and the destructor have no return type, nor a return statement.
- Destructors cannot have arguments. Constructors can. They can have several arguments, including default arguments.
- Constructors are not usually called explicitly. They are called automatically. Destructors are not usually called explicitly.
- A class may have several constructors. If a class has multiple constructors, the argument list, including default arguments, must be unique. (see box below)
- Ctor and Dtor are abbreviations for constructor and destructor.
- Every object must have a constructor. If you do not provide one, the compiler will create one for you. This constructor is a default constructor. Default constructor also refers to a constructor without arguments.
- Destructors are automatically called when a class object is **delete**d.

Overloaded Functions

Overloaded functions are functions with the same name, but different arguments. The following function prototypes illustrate overloaded functions:

```
int funk();

int funk(int x);

int funk(double d);

int funk(char*, int);

int funk(xyz&, abc&);

void funk(int a, int b, int c = 1);
```

Example 4-1 - The Circle Class with a constructor and destructor

```
1  // File: ex4-1.cpp - the circle class with ctor and dtor
2
3  #include <iostream>
4  using namespace std;
5
6  class circle
7  {
8    private:
9        double radius;
10    public:
11       circle(double);
12       ~circle();
13       double area(void) const;
14       void display(void) const;
15 };
16
17 circle::circle(double r)                          // constructor
18 {
19   radius = r;
20 }
21
22 circle::~circle()                                 // destructor
23 {
24   cout << "The destructor is called now\n";
25 }
26
27 double circle::area(void) const
28 {
29   return 3.14 * radius * radius;
30 }
31
32 void circle::display(void) const
33 {
34   cout << radius << endl;
35   return;
36 }
37
38 int main(void)
39 {
40   circle c(5.);       // an instance (object) of circle class
41   cout << "The area of circle c is " << c.area() << endl;
42   cout << "Circle c has radius ";
43   c.display();
44   return 0;
45 }
```

```
******  Output  ******

The area of circle c is 78.5
Circle c has radius 5
The destructor is called now
```

Example 4-2 - Constructor/Destructor Execution with respect to Scope

```
1  // File: ex4-2.cpp
2
3  #include <iostream>
4  using namespace std;
5
6
7  class test
8  {
9      char ch;
10   public:
11     test(char c);                                  // constructor
12     ~test();                                      // destructor
13 };
14
15 test::test(char c)
16 {
17   ch = c;
18   cout << "*** constructor called for object " << ch <<endl;
19 }
20
21 test::~test()
22 {
23   cout << "destructor called for object " << ch << endl;
24 }
25
26 int main(void)
27 {
28   test a('a');
29   {
30       test b('b');
31   }
32   {
33       test c('c');
34       {
35           test d('d');
36       }
37   }
38   return 0;
39 }
```

```
******  Output  ******

*** constructor called for object a
*** constructor called for object b
destructor called for object b
*** constructor called for object c
*** constructor called for object d
destructor called for object d
destructor called for object c
destructor called for object a
```

Example 4-3 - Constructor and Destructor

```
1  // File: ex4-3.cpp - the person class with ctor & dtor
2
3  #include <iostream>
4  #include <cstring>
5  #include <cstdlib>
6  using namespace std;
7
8  class person
9  {
10   private:
11     char* name;
12   public:
13     person(const char *);                    // constructor
14     ~person();                                // destructor
15     void print(void) const;          // display person's name
16 };
17
18 person::person(const char* n)
19 {
20   name = new char[strlen(n)+1];
21   strcpy(name,n);
22 }
23
24 person::~person(void)
25 {
26   delete[] name;
27 }
28
29 void person::print(void) const
30 {
31   cout << name << endl;
32   return;
33 }
34
35 int main(void)
36 {
37   person mary("Mary");
38   person joe("Joe");
39   mary.print();
40   joe.print();
41   return 0;
42 }
```

```
******  Output  ******
Mary
Joe
```

Example 4-4 - The card and deck Classes

This example illustrates a containment relationship between classes.

```
1  // File: ex4-4.cpp
2
3  #include <iostream>
4  #include <cstdlib>                  // needed for rand() function
5  using namespace std;
6
7  const char* value_name[13] = {"two","three","four","five","six",
8     "seven","eight","nine","ten","jack","queen","king","ace"};
9
10  const char* suit_name[4] = {"clubs","diamonds","hearts","spades"};
11
12  class card
13  {
14    private:
15     int value;
16     int suit;
17    public:
18      card() {}
19     card(int);
20     int get_value(void) { return value;} // accessor function
21     int get_suit(void) { return suit;}   // accessor function
22     void print(void);
23  };
24
25  card::card(int x)              // constructor
26  {
27    value = x % 13;
28    suit = x / 13;
29  }
30
31  void card::print(void)
32  {
33    cout << (value_name[value]) << " of "
34     << (suit_name[suit]) << endl;
35    return;
36  }
37
38
39  class deck
40  {
41    private:
42     card d[52];
43     int next_card;
44    public:
45     deck(void);
46     void shuffle(void);
47     void deal(int=5);
48     void print(void);
```

```
49  };
50
51  deck::deck(void)
52  {
53    for (int i = 0; i < 52; i++) d[i] = card(i);
54    next_card = 0;
55  }
56
57
58  void deck::shuffle(void)
59  {
60    int i, k;
61    card temp;
62    cout << "I am shuffling the deck\n";
63    for (i = 0; i < 52; i++)
64    {
65     k = rand() % 52;
66     temp = d[i];
67     d[i] = d[k];
68     d[k] = temp;
69    }
70    return;
71  }
72
73  void deck::print(void)
74  {
75    for (int i = 0; i < 52; i++)
76     d[i].print();
77    return;
78  }
79
80  void deck::deal(int no_of_cards)
81  {
82    cout << "\nOk, I will deal you " << no_of_cards << " cards:\n";
83    for (int i = 0; i < no_of_cards; i++)
84     d[next_card++].print();
85    return;
86  }
87
88
89  int main (void)
90  {
91    deck poker;
92    poker.shuffle();
93    poker.print();
94    poker.deal();
95    poker.deal(3);
96    return 0;
97  }
```

Output similar to Example 3-7

- How does line 53 work? What does ***d[i] = card(i)*** mean?
- How many constructor calls result from the deck instantiation on line 91?

Example 4-5 – The card and deck Classes Again

This example illustrations a more sophisticated approach to the card and deck classes and more interesting constructors and a destructor.

```
1  // File: ex4-5.cpp
2
3  #include <iostream>
4  #include <cstdlib>            // needed for rand() function
5  using namespace std;
6
7  const char* value_name[13] = {"two","three","four","five","six",
8     "seven","eight","nine","ten","jack","queen","king","ace"};
9
10  const char* suit_name[4] = {"clubs","diamonds","hearts","spades"};
11
12  class card
13  {
14    private:
15     int value;
16     int suit;
17    public:
18     card(int=0);
19     int get_value(void) { return value;} // accessor function
20     int get_suit(void) { return suit;}   // accessor function
21     void print(void);
22  };
23
24  card::card(int x)
25  {
26    x = abs(x)%52;              // make sure x is between 0 and 51
27    value = x % 13;
28    suit = x / 13;
29  }
30
31  void card::print(void)
32  {
33    cout << (value_name[value]) << " of "
34     << (suit_name[suit]) << endl;
35    return;
36  }
37
38
39  class deck
40  {
41    private:
42     card**  d;
43     int size;
44     int next_card;
45    public:
46     deck(int s = 52);
```

```
47     ~deck(void);
48     void shuffle(void);
49     void deal(int=5);
50     void print(void);
51  };
52
53  deck::deck(int s)
54  {
55    size = s;
56    d = new card*[size];
57    for (int i = 0; i < size; i++) d[i] = new card(i);
58    next_card = 0;
59  }
60
61  deck::~deck(void)
62  {
63    for (int i = 0; i < size; i++) delete d[i];
64    delete [] d;
65    cout << "The deck is gone" << endl;
66  }
67
68  void deck::shuffle(void)
69  {
70    int i, k;
71    card* temp;
72    cout << "I am shuffling the deck\n";
73    for (i = 0; i < size; i++)
74    {
75     k = rand() % size;
76     temp = d[i];
77     d[i] = d[k];
78     d[k] = temp;
79    }
80    return;
81  }
82
83  void deck::print(void)
84  {
85    for (int i = 0; i < size; i++)
86     d[i]->print();       // same as (*d[i]).print()
87    return;
88  }
89
90  void deck::deal(int no_of_cards)
91  {
92    cout << "\nOk, I will deal you " << no_of_cards << " cards:\n";
93    for (int i = 0; i < no_of_cards; i++)
94     d[next_card++]->print();
95    return;
96  }
97
```

```
99  int main (void)
100  {
101    deck poker;
102    poker.shuffle();
103    poker.print();
104    poker.deal();
105    poker.deal(3);
106    return 0;
107  }
```

Output similar to Example 3-7

- In line 57, how does the expression ***d[i] = new card(i)*** work ?

- What is better about this approach over the last example?

Example 4-6 - When is a Constructor called?

```
1  File:  ex4-6.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  class Z {
7    public:
8      Z(void) { cout << "Z's constructor is called now\n";}
9  };
10
11  int main(void) {
12    cout << "\n1. Is the constructor called?\n";
13    Z z;                                        // declare a Z
14    cout << "\n2. Is the constructor called?\n";
15    Z bunch[3];                          // declare a bunch of Zs
16    cout << "\n3. Is the constructor called?\n";
17    Z* ptrZ;                            // declare a pointer to Z
18    cout << "\n4. Is the constructor called?\n";
19    Z* a_new_prtZ = new Z;           // allocate memory for a Z
20    cout << "\n5. Is the constructor called?\n";
21    Z* threeZ = new Z[3];           // allocate memory for 3 Zs
22    cout << "\n6. Is the constructor called?\n";
23    Z** ptr_ptr_Z;                // declare a ptr to ptr to a Z
24    cout << "\n7. Is the constructor called?\n";
25    Z** ptr_ptr_newZ = new Z*;   // alloc mem for a ptr to a Z
26    return 0;
27  }
```

```
******  Output  ******

1. Is the constructor called?
Z's constructor is called now

2. Is the constructor called?
Z's constructor is called now
Z's constructor is called now
Z's constructor is called now

3. Is the constructor called?

4. Is the constructor called?
Z's constructor is called now

5. Is the constructor called?
Z's constructor is called now
Z's constructor is called now
Z's constructor is called now

6. Is the constructor called?

7. Is the constructor called?
```

Example 4-7 - When is a Constructor called?

```
1  // File:  ex4-7.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  class Z
7  {
8    public:
9        Z(void)                                    // constructor
10       {
11        cout << "Z's constructor is called now" << endl;
12       }
13       ~Z()                                        // destructor
14       {
15           cout << "Z's destructor is called now" << endl;
16       }
17 };
18
19 Z funk1(Z hey)
20 {
21   cout << "This is funk1\n";
22   return hey;
23 }
24
25 int main(void)
26 {
27   Z temp;
28   funk1(temp);
29   return 0;
30 }
```

```
******  Output  ******

Z's constructor is called now
This is funk1
Z's destructor is called now
Z's destructor is called now
Z's destructor is called now
```

- What is going on?

✔ Example 4-8 - When is a Constructor called?

```
1  // File: ex4-8.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  class Z
7  {
8    public:
9        Z(void)
10        {
11         cout << "Z's default constructor is called now\n";
12        }
13        Z(const Z& zed)
14        {
15         cout << "Z's copy constructor is called now" << endl;
16        }
17        ~Z()
18        {
19            cout << "Z's destructor is called now" << endl;
20        }
21 };
22
23 Z funk1(Z hey)
24 {
25   cout << "This is funk1\n";
26   return hey;
27 }
28
29 int main(void)
30 {
31   Z temp;
32   funk1(temp);
33   return 0;
34 }
```

```
******  Output  ******

Z's default constructor is called now
Z's copy constructor is called now
This is funk1
Z's copy constructor is called now
Z's destructor is called now
Z's destructor is called now
Z's destructor is called now
```

✔ What happened?

Example 4-9 - The Person and People Classes

```
1  // File: ex4-9.cpp
2
3  #include <iostream>
4  #include <cstring>
5  #include <cstdlib>
6  using namespace std;
7
8  class Person
9  {
10  private:
11     char* name;
12  public:
13     Person(const char *);       // constructor
14     ~Person();
15     void print(void) const;           // display Person's name
16  };
17
18
19  Person::Person(const char* n)
20  {
21    name = new char[strlen(n)+1];
22    if (name == 0)
23    {
24     cerr << "Insufficent memory to store " << n << endl;
25     exit(1);
26    }
27    strcpy(name,n);
28  }
29
30
31  Person::~Person()
32  {
33    cout << "Person destructor call for " << name << endl;
34    delete [] name;
35  }
36
37
38  void Person::print(void) const
39  {
40    cout << name << endl;
41    return;
42  }
43
44
45  class People
46  {
47    private:
48     Person** array;
49     int Person_index;    // Person index
50     int Max_People; // number of People in array
```

```
51    public:
52     People(int);
53     ~People();
54     void addPerson(void);
55     void print(void) const;
56  };
57
58
59  People::People(int nope)
60  {
61    Max_People = nope;
62    array = new Person*[Max_People];
63    Person_index = 0;
64  }
65
66
67  People::~People()
68  {
69    cout << "\nPeople destructor called" << endl;
70    for (int i = 0; i < Max_People; i++)
71    {
72      cout << "deleting pointer to Person[" << i << "]\n";
73      delete array[i];
74    }
75    delete [] array;
76  }
77
78
79  void People::addPerson(void)
80  {
81    char temp[20];
82    cout<<"Enter the Persons name => ";
83    cin >> temp;
84    array[Person_index++] = new Person(temp);
85    return;
86  }
87
88
89  void People::print(void) const
90  {
91    cout << "\nHere's the People:\n";
92    for (int i = 0; i < Person_index; i++) array[i]->print();
93    return;
94  }
95
96
97  int main (void)
98  {
99    cout << "How many friends do you have? ";
100    int no_friends;
101    cin >> no_friends;
102    People friends(no_friends);
```

```
103     for (int i = 0; i < no_friends; i++) friends.addPerson();
104     friends.print();
105     return 0;
106   }
```

```
*****  Sample Run  ******

How many friends do you have? 5
Enter the persons name => Henry
Enter the persons name => Hubert
Enter the persons name => Hank
Enter the persons name => Hilbert
Enter the persons name => Hellbert

Here's the people:
Henry
Hubert
Hank
Hilbert
Hellbert

People destructor called
deleting pointer to person[0]
Person destructor call for Henry
deleting pointer to person[1]
Person destructor call for Hubert
deleting pointer to person[2]
Person destructor call for Hank
deleting pointer to person[3]
Person destructor call for Hilbert
deleting pointer to person[4]
Person destructor call for Hellbert
```

- What does a People object look like?

- What caused each destructor call to Person?

- No copy constructors were provided for either the Person class or the People class. Is that a good idea? How would you write them?

The Default Constructor

A default constructor is the constructor that is executed when no arguments are provided in the declaration. There are three possible situations for this:

1. If you do not provide any constructor for a class, the compiler-provided one is considered the default constructor. It, of course, does not do anything other than allocating memory for the class object.

   This looks like:

```
class x
{
…            // no constructors defined for the class
};
```

2. If you provide a constructor without any arguments (void), then that is the class default constructor.

   This looks like:

```
class x
{
…
  x();       // or x(void);
…
};
```

3. If you provide a constructor with all default arguments, then, that, too, may be considered the default constructor. Warning: you may not have a class with both a void-argument constructor and one with all default arguments.

   This looks like:

```
class x
{
…
  x(int x=5);            // Note:  the only argument is a default argument
…
};
```

   or maybe like this:

```
class x
{
…
  x(double d = 0.0, unsigned short s=0);              // Two default
arguments
…
};
```

Instantiation of an Object Using the Default Constructor

Objects are declared using the default constructor without parentheses, not even empty parentheses.

For example, to declare (instantiate an object) using any one of the x classes from the previous page, you would write it as:

```
x object;
```

or

```
…      new x;
```

or

```
…      new x();
```

not

```
x object();
```

✔ Why can't you declare a class object with parentheses using the default constructor?

Answer:

Suppose you write a function like this:

```
void funk()
{
…
       x object();
…
}
```

The problem is that the statement, ***x object();***, can take on two meanings. It looks just like a function prototype (a function called object with a void argument and an x return) and now you want to use it to instantiate an x object? I don't think so. You compiler refuses to be confused.

Overloading Constructors and Copy Constructors

It is common practice to overload constructors, providing them with different argument types for different situations. The default constructor is one that has no arguments. A copy constructor is one that copies an existing instance of a class. It is common practice to use reference to a const class object as the argument of a copy constructor.

Example 4-10 - The text class

```
1  File:  ex4-10.cpp
2
3  #include <iostream>
4  #include <cstring>
5  using namespace std;
6
7  class text
8  {
9    private:
10        char* s;
11        int length;
12    public:
13        text(void);                    // default constructor
14        text(const char *);
15        text(const text&);              // copy constructor
16        text(int);
17        text(char);
18        ~text() { delete [] s; }        // inline destructor
19        void print(void) const;
20 };
21
22 text::text(void) {
23   s = new char[1];
24   s[0] = '\0';
25   length = 0;
26 }
27
28 text::text(const char* str)
29 {
30   s = new char[strlen(str) + 1];
31   length = strlen(str);
32   strcpy(s,str);
33 }
34
35 text::text(const text& str)
36 {
37   length = str.length;
38   s = new char[length + 1];
39   strcpy(s,str.s);
40 }
```

```
41  text::text(int len)
42  {
43    length = len;
44    s = new char[length + 1];
45  }
46
47  text::text(char ch)
48  {
49    length = 1;
50    s = new char[2];
51    s[0] = ch;
52    s[1] = '\0';
53  }
54
55  void text::print(void) const
56  {
57    cout << s << endl;
58  }
59
60  int main(void)
61  {
62    text a("Have a nice day");
63    a.print();
64
65    text b(a);
66    b.print();
67
68    text c;                    // Note: do not use text c();
69    c.print();
70
71    text d(15);
72    d.print();
73
74    text e('x');
75    e.print();
76
77    return 0;
78  }
```

```
******  Output  ******

Have a nice day
Have a nice day

----------------²²²²                    ← What's this?
x
```

More constructor questions and answers

Should you write a Constructor?
Yes

What if you don't write a constructor?
Your compiler will write one for you

Why won't this compile?

```
1  class xyz {
2      int x;
3    public:
4     xyz(int _x) { x = _x; }
5  };
6
7  int main() {
8    xyz Object;
9    return 0;
10  }
```

The xyz declaration in main() would use a default constructor. If you write any constructor in your class, your compiler will not write a default constructor for you.

Why should you write a copy constructor?
It is often desirable to create an object by copying an existing object. And even if you don't do that, anytime an object is passed or returned by value, the copy constructor is used.

What happens if you don't write a copy constructor?
If you don't your compiler will write one for you when needed. It may not do what you want.

What's the general advice about writing constructors?
Write a default constructor, a copy constructor and any others you may need. Even if you don't need the default or copy constructor now, you may need it later.

What if a class contains another class object, will the contained object's constructor be called when a container class object is declared?
Yes

Constructor Initialization List

C++ has a special syntax used to initialize class data members in the constructor function. Initializing values may be placed in a comma-separated list after the constructor argument list and before the function body. A colon precedes the list of initializers. For example:

Example 4-11 - Constructor with Initialization List

```
1  // File: ex4-11.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  class xyz
7  {
8    private:
9      int a, b, c;
10   public:
11     xyz() : a(1), b(2), c(3) { }           // default constructor
12     xyz(int, const xyz&);             // constructor #2 prototype
13     void print(void) { cout << a << b << c << endl; }
14 };
15
16
17 xyz::xyz(int aa, const xyz& hey) : a(aa), b(5), c(hey.c) {}
18
19
20 int main(void)
21 {
22   xyz yo;                          // uses default constructor
23
24   xyz yoyo(2,yo);                       // uses constructor #2
25
26   yo.print();
27
28   yoyo.print();
29
30   return 0;
31 }
```

```
******  Output  ******

123
253
```

There are three situations where constructor initialization lists are required:

- when a class contains a const data member
- when a class contains a reference data member
- when you wish to make reference to a specific, not-default constructor (this is especially useful with inheritance and containment)

The following example show initialization of a class with const and reference data members:

Example

```
class xyz
{
  private:
    int a;
    const int b;
    int& c;
  public:
    xyz(int,int&);
     .
     .
     .
};

xyz::xyz(int i, int& j) : b(i+1), c(j)
{
  a = 0;
}
     .
     .
     .

int main(void)
{
  int z = 3,
      q = 5;
  xyz hey(z,q);
     .
     .
   .
```

✔ After hey is declared, what are the values assigned to hey's a, b, and c?

Example 4-11a – Another example using constructor initializers

```
1  #include <string>
2
3  class tail
4  {
5     unsigned short length;
6  public:
7     tail(unsigned short len = 6);
8  };
9
10  tail::tail(unsigned short len)
11  : length(len)
12  {}
13
14
15  class dog
16  {
17    tail             tail_;
18    std::string      name_;
19  public:
20    dog(std::string name);
21  };
22
23  dog::dog(std::string name)
24  : tail_(tail()), name_(name)
25  {}
26
27  int main()
28  {
29    dog beagle("Emily");
30    return 0;
31  }
```

Notes:

Line 7

Line 10

Line 11

Line 18

Line 20

Line 24 – What if you do not include ***tail_(tail(),*** here?

Line 29

The following example illustrates some initialization lists. This example also shows how multiple constructors may be very applicable to a real-world situation. If you are not into algebra and geometry, skip the arithmetic.

Example 4-12 - The Point and Line classes.

```
1  // File: ex4-12p.h - Point class header file
2
3  #ifndef POINT_H
4  #define POINT_H
5
6  inline double square(double d) { return d*d; }
7
8  class Line;         // forward declaration
9
10  class Point
11  {
12  private:
13    double x;
14    double y;
15  public:
16
17    // Constructors
18
19    // Create Point at origin
20    Point(void);
21
22    // Create Point using x-y coordinates
23    Point(double x1,double y1);
24
25    // Create Point as midpoint of two Points
26    Point(const Point& p, const Point& q);
27
28    // Create Point as intersection of two lines
29    Point(const Line& l,const Line &m);
30
31    // Copy constructor
32    Point(const Point& p);
33
34    // print a point as (x,y)
35    void print(void) const;
36
37    // accessor functions
38    double get_x(void) const;
39    double get_y(void) const;
40
41    // assign value to x member
42    void set_x(double x1);
43
44    // assign value to y member
45    void set_y(double y1);
46
```

```
47    // determine the distance to another point
48    double distance_to_Point(const Point& p) const;
49  };
50
51  #endif
```

```
1   // File: ex4-12p.cpp - Point class source file
2
3   #include <cmath>
4   #include <iostream>
5   using namespace std;
6
7   #include "ex4-12p.h"
8   #include "ex4-12l.h"
9
10   Point::Point(void)
11   {
12     x = 0.0; y = 0.0;
13   }
14
15
16   Point::Point(double x1,double y1)
17   {
18     x = x1; y = y1;
19   }
20
21
22   Point::Point(const Point& p)
23   {
24     x= p.x; y = p.y;
25   }
26
27
28   Point::Point(const Point& p, const Point& q)
29   {
30     x = (p.x+q.x)/2.; y = (p.y+q.y)/2.;
31   }
32
33
34   Point::Point(const Line& l,const Line &m)
35   {
36     if (l.slope() == m.slope())   // parallel or coincident Lines
37     {
38           x = HUGE_VAL;
39           y = HUGE_VAL;
40     }
41     else
42     {
43           x = (m.get_b()*l.get_c()-m.get_c()*l.get_b())/
44                 (l.get_b()*m.get_a()-m.get_b()*l.get_a());
45           y = (l.get_a()*m.get_c()-m.get_a()*l.get_c())/
```

```
46                (l.get_b()*m.get_a()-m.get_b()*l.get_a());
47    }
48  }
49
50
51  void Point::print(void) const
52  {
53    cout << '(' << x << ',' << y << ')';
54  }
55
56
57  void Point::set_x(double x1)
58  {
59    x = x1;
60  }
61
62
63  void Point::set_y(double y1)
64  {
65    y = y1;
66  }
67
68
69  double Point::get_x(void) const
70  {
71    return x;
72  }
73
74
75  double Point::get_y(void) const
76  {
77    return y;
78  }
79
80  double Point::distance_to_Point(const Point& p) const
81  {
82    return sqrt(square(p.x-x)+square(p.y-y));
83  }
```

```
1  // File: ex4-12l.h - Line class header file
2
3  #ifndef LINE_H
4  #define LINE_H
5
6  #include "ex4-12p.h"     // include Point header file
7
8  class Line
9  {
10  private:
11    Point p1;
12    Point p2;
```

```
13    double a,b,c;    // coefficients for equation of a Line
14  public:
15    // Constructors
16    // Create Line using two Points
17    Line(const Point& pp1,const Point& pp2);
18
19    // Create Line using coefficients for the equation of a Line
20    Line(double c1, double c2, double c3);
21
22    // Create Line parallel/perpendicular to a Line through a Point
23    Line(const Point& p,const Line& l, const char* Line_type);
24
25    // Create horizontal/vertical Line through a Point
26    Line(const Point& p, const char* Line_type);     // horiz/vert
   through pt
27
28    // Create an offset Line from a given Line
29    Line(const Line& l,double offset);
30
31    // Create Line though a Point with a given lenght and and angle
32    Line(const Point& p,double length, double angle);
33
34    // Print Line: equation, Points, and slope
35    void print(void) const;
36
37    // Accessor functions
38    double get_a(void) const { return a; }
39    double get_b(void) const { return b; }
40    double get_c(void) const { return c; }
41    const Point& get_p1() const { return p1; }
42    const Point& get_p2() const { return p2; }
43
44    // Length of a Line (distance between two Points)
45    double length(void) const;
46
47    // Midpoint of a Line
48    Point midpoint() const;
49
50    // Slope of a Line
51    double slope(void) const;
52
53    // distance to a parallel Line
54    double distance_to_Line(const Line&) const;
55
56    // distance to a Point
57    double distance_to_Point(const Point&) const;
58
59    // x-intercept of a Line
60    double x_intercept(void) const;
61
62    // y-intercept of a Line
63    double y_intercept(void) const;
```

```
64  };
65
66  #endif
```

```
1  // File: ex4-12l.cpp – Line class source file
2
3  #include <cmath>
4  #include <cstring>
5  #include <iostream>
6  using namespace std;
7
8  #include "ex4-12l.h"
9
10  Line::Line(const Point& pp1,const Point& pp2) : p1(pp1), p2(pp2)
11  {
12    if (slope() == HUGE_VAL)
13    {
14          a = 1.0;
15          b = 0.0;
16          c = -pp1.get_x();
17    }
18    else
19    {
20          a = -slope();
21          b = 1.0;
22          c = slope()*pp1.get_x()-pp1.get_y();
23    }
24  }
25
26
27  // create a Line using equation coefficients
28  Line::Line(double c1, double c2, double c3)
29  : a(c1), b(c2), c(c3)
30  {
31    if (c1 != 0.0) p1 = Point(-c3/c1,0.0);
32    else if (c2 != 0.0) p1 = Point(1.0,-c3/c2);
33    else p1 = Point(0.0,0.0);
34    if (c2 != 0.0) p2 = Point(0.0,-c3/c2);
35    else if (c1 != 0.0) p2 = Point(-c3/c1,1.0);
36    else p2 = Point(0.0,0.0);
37  }
38
39
40  // create a Line through a Point parallel or perpendicular to a
    Line
41  Line::Line(const Point& p,const Line& l, const char* Line_type) :
    p1(p)
42  {
```

```
43    double m;      // slope
44    if (strcmp(Line_type,"parallel")==0.0) m = l.slope();
45    else if (l.slope() == 0.0) m = HUGE_VAL;
46    else if (l.slope() == HUGE_VAL) m = 0.0;
47    else m = -1./l.slope();
48    if (m == HUGE_VAL) {
49          a = 1.0;
50          b = 0.0;
51          c = -p.get_x();
52          p2.set_x(p1.get_x());
53          p2.set_y(p1.get_y()+1.0);
54    }
55    else {
56          a = m;
57          b = -1.0;
58          c = -m*p.get_x()+p.get_y();
59          p2.set_x(0.0);
60          p2.set_y(c);
61    }
62  }
63
64
65  // create a vertical/horizontal Line through a Point
66  Line::Line(const Point& p, const char* Line_type) : p1(p)
67  {
68    if (strcmp(Line_type,"vertical") == 0)
69    {
70          p2.set_x(p1.get_x());
71          p2.set_y(p1.get_y()+1);
72          a = 1.0;
73          b = 0.0;
74          c = -p1.get_x();
75    }
76    else
77    {
78          p2.set_x(p1.get_x()+1);
79          p2.set_y(p1.get_y());
80          a = 0.0;
81          b = 1.0;
82          c = -p1.get_y();
83    }
84  }
85
86
87  Line::Line(const Line& l, double offset)
88  :p1(l.p1.get_x(),l.p1.get_y()+offset/fabs(sin(atan(l.slope())))),
89  p2(l.p2.get_x(),l.p2.get_y()+offset/fabs(sin(atan(l.slope()))))
90  {
91    if (slope() == HUGE_VAL)
92    {
93          a = 1.0;
94          b = 0.0;
```

```
95          c = -p1.get_x();
96     }
97     else
98     {
99          a = -slope();
100         b = 1.0;
101         c = slope()*p1.get_x()-p1.get_y();
102    }
103   }
104
105
106   // create an angled-length Line
107   Line::Line(const Point& p,double length, double angle)
108   :p1(p),p2(p1.get_x()+length*cos(angle),p1.get_y()+length*sin(angle
    ))
109   {
110    if (slope() == HUGE_VAL)
111    {
112         a = 1.0;
113         b = 0.0;
114         c = -p1.get_x();
115    }
116    else
117    {
118         a = -slope();
119         b = 1.0;
120         c = slope()*p1.get_x()-p1.get_y();
121    }
122   }
123
124
125   void Line::print(void) const
126   {
127    cout << "Line  eqn: ";
128    if (a == 1.0) cout << 'x';
129    else if (a == -1.0) cout << "-x";
130    else if (a != 0.0) cout << a << 'x';
131    if ((a != 0.0) && (b != 0.0)) cout << " + ";
132    if (b == 1.0) cout << 'y';
133    else if (b == -1.0) cout << "-y";
134    else if (b != 0.0) cout << b << 'y';
135    if (c != 0.0) cout << " + " << c;
136    cout << " = 0";
137    cout << "  pts: ";
138    p1.print();
139    cout << ',';
140    p2.print();
141    cout << "  slope: " << slope() << endl;
142   }
143
144
145   double Line::length(void) const
```

```
146  {
147   return sqrt(square(p2.get_x()-p1.get_x())+square((p2.get_y()-
   p1.get_y())));
148  }
149
150
151  Point Line::midpoint() const
152  {
153   return
   Point((p1.get_x()+p2.get_x())/2,(p1.get_y()+p2.get_y())/2);
154  }
155
156
157  double Line::slope(void) const
158  {
159   if (!(p2.get_x()-p1.get_x())) return HUGE_VAL;
160   else return ((p2.get_y()-p1.get_y())/(p2.get_x()-p1.get_x()));
161  }
162
163
164  // returns distance from a Line to a Point
165  double Line::distance_to_Point(const Point& p) const
166  {
167   return (fabs(a*p.get_x()+b*p.get_y()+c)/
168         sqrt(square(a)+square(b)));
169  }
170
171
172  // returns distance between two parallel Lines
173  double Line::distance_to_Line(const Line& l) const
174  {
175   if (slope() != l.slope()) return 0.0;
176   else return distance_to_Point(l.p1);
177  }
178
179
180  double Line::x_intercept(void) const
181  {
182   if (a != 0.0) return -c/a;
183   else return HUGE_VAL;
184  }
185
186
187  double Line::y_intercept(void) const
188  {
189   if (b != 0.0) return -c/b;
190   else return HUGE_VAL;
191  }
```

```
1  // File: ex4-12m.cpp – main()
2
```

```
3  #include <iostream>
4  #include <iomanip>       // for setprecision
5  using namespace std;
6
7  #include "ex4-12p.h"
8  #include "ex4-12l.h"
9
10
11  const double pi = 3.14159265;
12
13
14  int main(void)
15  {
16    cout << setprecision(3) << endl;      // print with 3 decimal
   place accuracy
17
18    Point origin;
19    Point p1(1.,2.);
20    Point p2(3.0,4.0);
21    Point p3(3.0,5.0);
22    Point p4(4.0,5.0);
23    Point p5(0.0,5.0);
24    Point p6(-2.0,3.0);
25
26    Line l1(p1,p2);
27    Line l2(p2,p3);
28    Point p7(l1,l2);
29    Line l3(p3,p4);
30    Line l4(p5,p6);
31    Line l5(1.,2.,3.);
32    Line l6(p1,l4,"parallel");
33    Line l7(p1,l4,"perpendicular");
34    Line l8(p6,"vertical");
35    Line l9(p7,"horizontal");
36    Line l10(l4,1.0);
37    Line l11(l5,-2.0);
38    Line l12(origin,4.0,pi/3.0);
39    Line l13(p1,5.0,-pi/4.0);
40
41    cout << "origin="; origin.print(); cout << endl;
42    cout << "p1="; p1.print(); cout << endl;
43    cout << "p2="; p2.print(); cout << endl;
44    cout << "p3="; p3.print(); cout << endl;
45    cout << "p4="; p4.print(); cout << endl;
46    cout << "p5="; p5.print(); cout << endl;
47    cout << "p6="; p6.print(); cout << endl;
48    cout << "p7="; p7.print(); cout << endl;
49    cout << "l1-"; l1.print();
50    cout << "l1 (x-intercept): " << l1.x_intercept()
51         << "   (y-intercept): " << l1.y_intercept() << endl;
52    cout << "l2-";l2.print();
53    cout << "l3-";l3.print();
```

```
54    cout << "l4-";l4.print();
55    cout << "l5-";l5.print();
56    cout << "l6-";l6.print();
57    cout << "l7-";l7.print();
58    cout << "l8-";l8.print();
59    cout << "l9-";l9.print();
60    cout << "l10-";l10.print();
61    cout << "l11-";l11.print();
62    cout << "l12-";l12.print();
63    cout << "l13-";l13.print();
64
65    cout << "length of l1 = " << l1.length() << endl;
66    cout << "length of l2 = " << l2.length() << endl;
67    cout << "midpoint of l1 = "; l1.midpoint().print(); cout << endl;
68    cout << "distance p1 to p2 = " << p1.distance_to_Point(p2) <<
   endl;
69    cout << "distance p1 to p3 = " << p1.distance_to_Point(p3) <<
   endl;
70    cout << "distance p1 to p4 = " << p1.distance_to_Point(p4) <<
   endl;
71    cout << "distance p2 to p3 = " << p2.distance_to_Point(p3) <<
   endl;
72    cout << "distance p1 to p1 = " << p1.distance_to_Point(p1) <<
   endl;
73    cout << "distance l1 to p3 = " << l1.distance_to_Point(p3) <<
   endl;
74    cout << "distance l1 to p4 = " << l1.distance_to_Point(p4) <<
   endl;
75    cout << "distance l2 to p5 = " << l2.distance_to_Point(p5) <<
   endl;
76    cout << "distance l3 to p6 = " << l3.distance_to_Point(p6) <<
   endl;
77    cout << "distance l1 to l4 = " << l1.distance_to_Line(l4) <<
   endl;
78    return 0;
79  }
```

```
****** Output (MS Visual C++ 2008) ******

origin=(0,0)
p1=(1,2)
p2=(3,4)
p3=(3,5)
p4=(4,5)
p5=(0,5)
p6=(-2,3)
p7=(3,4)
l1-Line  eqn: -x + y + -1 = 0  pts: (1,2),(3,4)  slope: 1
l1 (x-intercept): -1   (y-intercept): 1
l2-Line  eqn: x + -3 = 0  pts: (3,4),(3,5)  slope: 1.#J
l3-Line  eqn: y + -5 = 0  pts: (3,5),(4,5)  slope: 0
l4-Line  eqn: -x + y + -5 = 0  pts: (0,5),(-2,3)  slope: 1
```

```
l5-Line  eqn: x + 2y + 3 = 0  pts: (-3,0),(0,-1.5)  slope: -0.5
l6-Line  eqn: x + -y + 1 = 0  pts: (1,2),(0,1)  slope: 1
l7-Line  eqn: -x + -y + 3 = 0  pts: (1,2),(0,3)  slope: -1
l8-Line  eqn: x + 2 = 0  pts: (-2,3),(-2,4)  slope: 1.#J
l9-Line  eqn: y + -4 = 0  pts: (3,4),(4,4)  slope: 0
l10-Line  eqn: -x + y + -6.41 = 0  pts: (0,6.41),(-2,4.41)  slope: 1
l11-Line  eqn: 0.5x + y + 5.97 = 0  pts: (-3,-4.47),(0,-5.97)  slope:
-0.5
l12-Line  eqn: -1.73x + y = 0  pts: (0,0),(2,3.46)  slope: 1.73
l13-Line  eqn: 1x + y + -3 = 0  pts: (1,2),(4.54,-1.54)  slope: -1
length of l1 = 2.83
length of l2 = 1
midpoint of l1 = (2,3)
distance p1 to p2 = 2.83
distance p1 to p3 = 3.61
distance p1 to p4 = 4.24
distance p2 to p3 = 1
distance p1 to p1 = 0
distance l1 to p3 = 0.707
distance l1 to p4 = 0
distance l2 to p5 = 3
distance l3 to p6 = 2
distance l1 to l4 = 2.83
```

****** Output (`gcc version 4.3.2 (GCC) undex Linux`) ******

`Compile command:` **`g++ ex4-12*.cpp -Wall`**

****** Output ******

```
origin=(0,0)
p1=(1,2)
p2=(3,4)
p3=(3,5)
p4=(4,5)
p5=(0,5)
p6=(-2,3)
p7=(3,4)
l1-Line  eqn: -x + y + -1 = 0  pts: (1,2),(3,4)  slope: 1
l1 (x-intercept): -1   (y-intercept): 1
l2-Line  eqn: x + -3 = 0  pts: (3,4),(3,5)  slope: inf
l3-Line  eqn: y + -5 = 0  pts: (3,5),(4,5)  slope: 0
l4-Line  eqn: -x + y + -5 = 0  pts: (0,5),(-2,3)  slope: 1
l5-Line  eqn: x + 2y + 3 = 0  pts: (-3,0),(0,-1.5)  slope: -0.5
l6-Line  eqn: x + -y + 1 = 0  pts: (1,2),(0,1)  slope: 1
l7-Line  eqn: -x + -y + 3 = 0  pts: (1,2),(0,3)  slope: -1
l8-Line  eqn: x + 2 = 0  pts: (-2,3),(-2,4)  slope: inf
l9-Line  eqn: y + -4 = 0  pts: (3,4),(4,4)  slope: 0
l10-Line  eqn: -x + y + -6.41 = 0  pts: (0,6.41),(-2,4.41)  slope: 1
l11-Line  eqn: 0.5x + y + 5.97 = 0  pts: (-3,-4.47),(0,-5.97)  slope: -0.5
l12-Line  eqn: -1.73x + y = 0  pts: (0,0),(2,3.46)  slope: 1.73
l13-Line  eqn: 1x + y + -3 = 0  pts: (1,2),(4.54,-1.54)  slope: -1
length of l1 = 2.83
length of l2 = 1
```

```
midpoint of l1 = (2,3)
distance p1 to p2 = 2.83
distance p1 to p3 = 3.61
distance p1 to p4 = 4.24
distance p2 to p3 = 1
distance p1 to p1 = 0
distance l1 to p3 = 0.707
distance l1 to p4 = 0
distance l2 to p5 = 3
distance l3 to p6 = 2
distance l1 to l4 = 2.83
```

Copy Constructor Notes

Why write a copy constructor? The compiler will provide one for you. Why bother, do you really need more practice writing constructors? The answer depends on your class. The compiler provided copy constructor performs a "shallow copy". This means that each data member is "cloned" for the new copy. For example, consider this class:

```
class X {
int         a;
double            b;
char        c[10];
…
};
…
X     Object1;
X     Object2(Object1);       // this uses the copy ctor
```

The compiler provided copy constructor would copy each of Object1's members to Object2, so two object would be identical. In this case, you do not need to write a copy constructor.

What about this class?

```
class Y {
…
char* ptr;
…
};
…
Y     Object3;
Y     Object4(Object3);       // this uses the copy ctor
```

Again, the compiler provided copy constructor would perform a shallow copy. This means that Object4's ptr would contain the same address as Object3's ptr. Is that what you want? This means that if you change the value that ptr points to for Object 3, then you also changed it for Object 4. Hence, the two objects and not autonomous. So, you should write a copy constructor in this case, so that ptr points to its own value. The typical copy constructor for this case should look something like this:

```
Y::Y(const Y& _Y) {
…
  ptr = new char[strlen(_Y.ptr)+1];
  strcpy(ptr,_Y.ptr);
…
}
```

Example 4-13 – This should demonstrate why you might want to write a copy constructor.

```
1  // File: ex4-13.cpp - Why write a copy constructor?
2
3  #include <iostream>
4
5  class ABC
6  {
7     int * p;
8  public:
9           ABC(int P=0) { p = new int(P); }
10          ~ABC() { delete p; }
11          int value() const { return *p; }
12  };
13
14  void print (ABC object)
15  {
16    std::cout << object.value() << std:: endl;
17  }
18
19  int main()
20  {
21    ABC x(2);
22    print (x);
23    ABC y(3);
24    print(y);
25    print (x);
26  }
27
```

****** Output ******

MS Visual Studio Enterprise 2015

```
2
3
-572662307
!!!!!!  Runtime error  !!!!!!
```

gnu g++ compiler, version 4.3.2 on Linux

```
2
3
0
Program crash!!!!!
```

Note: the gnu c++ compiler has an option to compile with warnings that violate the style guidelines for Scott Meyers' **Effective C++** book, ***-Weffc++***. This option would have produced the following warnings:

```
/home/joe/deanza/examples> g++ ex4-13.cpp -Wall -Weffc++
ex4-13.cpp:6: warning: `class ABC' has pointer data members
ex4-13.cpp:6: warning:   but does not override `ABC(const ABC&)'
ex4-13.cpp:6: warning:   or `operator=(const ABC&)'
ex4-13.cpp: In constructor `ABC::ABC(int)':
ex4-13.cpp:9: warning: `ABC::p' should be initialized in the member
   initialization list
```

What should the signature of a copy constructor look like?

```
class Cls
{
…
      Cls(Cls cls);           // 1. argument passed by value?
      Cls(Cls& cls);          // 2. argument passed by reference?
      Cls(const Cls& cls);    // 3. argument passed by reference to const?
…
};
```

1. won't work. If it did, it would be an infinitely recursive call.
2. Works, but it's not as "safe" as 3. (do you know why?)
3. This is the best. You may not have 2 and 3 present together.

Static Class Objects

How does a static object behave in C++? If you declare an object as static, is it instantiated only one time, like C? Can you return a static object by reference?

```
// File: ex4-14.cpp - Static objects
1
2  #include <iostream>
3  using namespace std;
4
5  class thing {
6    public:
7      thing() { cout<<"thing constructor called for "<<this<<endl; }
8      ~thing() { cout<<"thing destructor called for "<<this<<endl; }
9  };
10
11  thing& funk() {
12    static thing T;
13    cout << "funk() called: &T=" << &T << endl;
14    return T;
15   }
16
17   int main() {
18     cout << &(funk()) << endl;
19     cout << &(funk()) << endl;
20     cout << &(funk()) << endl;
21     return 0;
22   }
23
```

```
******* Program output  *******

thing constructor called for 0x22640
funk() called: &T=0x22640
0x22640
funk() called: &T=0x22640
0x22640
funk() called: &T=0x22640
0x22640
thing destructor called for 0x22640
```

- How many different objects were created in this program?

The delete operator and destructors

- When you delete a class object, the destructor is called for the class.
- If you delete an array of class objects, without using delete [], the destructor may only be called once, instead of once for each array object.

Example 4-15 - delete and destructors

```
1  // File: eTest4-15.cpp - delete or delete []
2
3  #include <iostream>
4  using namespace std;
5
6  class Test
7  {
8     int n;
9  public:
10    Test() { n = 0; cout << "Test constructor called\n"; }
11    ~Test() { cout << "Test destructor called\n"; }
12 };
13
14 int main(void)
15 {
16   Test* ptrTest;
17
18   cout << "allocate space for 1 Test\n";
19   ptrTest = new Test;
20   delete ptrTest;
21
22   cout << "allocate space for 3 Tests\n";
23   ptrTest = new Test[3];
24   delete ptrTest;
25   // MS Visual C++ 2008 calls the destructor once, then errors
26   // Digital Mars C++ Compiler (ver 8.42) calls the destructor once
27   // gnu compiler (ver 4.3.1) calls destructor once, then crashes
28   return 0;
29 }
```

```
******  Output  ******

allocate space for 1 X
X constructor called
X destructor called
allocate space for 3 Xs
X constructor called
X constructor called
X constructor called
X destructor called
```

✔ What is the problem here?

Containment, Initializers, and Default Constructors

The following example demonstrates how constructors work in a container relationship. It also shows how you can control which constructor is called in the contained class by using constructor initialization list syntax.

Example 4-16 – Containment and constructors

```
1  // File: ex4-16.cpp - containment and constructors
2
3  #include <iostream>
4  using namespace std;
5
6
7  class One
8  {
9     int member;
10  public:
11    One() { cout << "One default ctor called\n";}
12    One(int j) : member(j)
13      { cout << "One second ctor called" << endl;}
14  };
15
16  class Two
17  {
18    One member;
19  public:
20    Two() { cout << "Two default ctor called" << endl; }
21    Two(int k) : member(k)
22      { cout << "Two second ctor called" << endl;}
23  };
24
25  int main()
26  {
27    cout << "declare object1" << endl;
28    Two object1;
29    cout << "declare object2" << endl;
30    Two object2(5);
31    return 0;
32  }
```

```
******  Output  ******

declare object1
One default ctor called
Two default ctor called
declare object2
One second ctor called
Two second ctor called
```

Containment vs. Nested classes and constructor calls

The following example illustrates the difference in constructor calls between a container relationship and a nested organization

```
1  // Example 4-14a.cpp -
2  // Containment vs. Nested classes and constructor calls
3  // Contributed by Raymond White 5/2009
4
5  #include <iostream>
6  using namespace std;
7
8  class Innermost
9  {
10   int i;
11 public:
12   Innermost() { cout << "ctor of Innermost\n"; }
13   ~Innermost() { cout << "dtor of Innermost\n"; }
14 };
15
16 class Inner
17 {
18   Innermost inm;
19   int i;
20 public:
21   Inner() { cout << "ctor of Inner\n"; }
22   ~Inner() { cout << "dtor of Inner\n"; }
23 };
24
25 class Outer
26 {
27   Inner in;
28   int i;
29 public:
30   Outer() { cout << "ctor of Outer\n"; }
31   ~Outer() { cout << "dtor of Outer\n"; }
32 };
33
34 class Outermost
35 {
36   Outer o;
37   int i;
38 public:
39   Outermost() { cout << "ctor of Outermost\n"; }
40   ~Outermost() { cout << "dtor of Outermost\n"; }
41 };
42
43 class NestedOutermost
44 {
45   int i;
46 public:
47   class NestedOuter
```

```
48    {
49      int i;
50    public:
51      class NestedInner
52    {
53        int i;
54      public:
55        class NestedInnermost
56        {
57          int i;
58        public:
59          NestedInnermost() { cout << "ctor of NestedInnermost\n"; }
60          ~NestedInnermost() { cout << "dtor of NestedInnermost\n"; }
61        };
62
63        NestedInner() { cout << "ctor of NestedInner\n"; }
64        ~NestedInner() { cout << "dtor of NestedInner\n"; }
65      };
66
67      NestedOuter() { cout << "ctor of NestedOuter\n"; }
68      ~NestedOuter() { cout << "dtor of NestedOuter\n"; }
69    };
70
71    NestedOutermost() { cout << "ctor of NestedOutermost\n"; }
72    ~NestedOutermost() { cout << "dtor of NestedOutermost\n"; }
73  };
74
75
76  int main()
77  {
78    Outermost myOutermost;
79    NestedOutermost myNestedOutermost;
80    cout << "sizeof(myOutermost)=" << sizeof(myOutermost) << endl;
81    cout << "sizeof(myNestedOutermost)=" =<<sizeof(myNestedOutermost)
82      << endl;
83    return 0;
84  }
```

******* Output *******

```
ctor of Innermost
ctor of Inner
ctor of Outer
ctor of Outermost
ctor of NestedOutermost
sizeof(myOutermost)=16
sizeof(myNestedOutermost)=4
dtor of NestedOutermost
dtor of Outermost
dtor of Outer
dtor of Inner
dtor of Innermost
```

As you can see from the output, the creation of the (container relationship) Outermost object results in four constructor calls whereas the NestedOutmost results in only one constructor call.

Explicit Constructors

The keyword ***explicit*** is used to specify that a constructor may only be used for object instantiation and not for automatic conversion. explicit should only be applied to single-argument constructors, or contructors that can be invoked with one argument, since the idea is to avoid automatic conversion of one type to another (class) type.

Here's an example that demonstrates the effect.

Example 4-17 – Explicit constructors

```
1  // File: ex4-17.cpp - explicit constructors
2
3  #include <iostream>
4  using namespace std;
5
6  class A
7  {
8  public:
9     A(int);                    // non-explicit ctor
10  };
11
12
13  class B
14  {
15  public:
16    explicit B(int);           // explicit ctor
17  };
18
19  A::A(int) {
20    cout << "A ctor called\n";
21  }
22
23  B::B(int) {                  // do not repeat keyword explicit
24    cout << "B ctor called\n";
25  }
26
27  void funkA(A object) {
28    cout << "funkA called\n";
29  }
30
31  void funkB(B object) {
32    cout << "funkB called\n";
33  }
34
35  void funkAB(A obj)
36  {
37    cout << "funkAB(A) called\n";
38  }
39
40  void funkAB(B obj)
```

```
41  {
42    cout << "funkAB(B) called\n";
43  }
44  int main()
45  {
46    A objA(2);       // instantiate an A object
47    B objB(3);       // instantiate a B object
48
49    funkA(objA);     // call funkA() with an exact argument match
50
51    funkA(9);        // call funkA() with an non-exact argument match
52
53    funkB(objB);     // call funkB() with an exact argument match
54
55  //  funkB(16);  // compile error: cannot convert int to a B object
56
57    funkAB(6);       // compile error if B(int) is not explicit
58
59    return 0;
60  }
```

```
******  Output  ******

A ctor called
B ctor called
funkA called
A ctor called
funkA called
funkB called
funkAB(A) called
```

More Class Concepts

The this Pointer

this is a special pointer used inside member functions to point to the object itself. *this (this dereferenced) represents the "*current*" object. **this** is the address of the object. **this** is most commonly used to return by reference.

Example 5-1 - The **this** Pointer

```
1  // File: ex5-1.cpp - the this pointer
2  #include <iostream>
3  using namespace std;
4
5  class Thing
6  {
7    private:
8      int x;
9      double y;
10    public:
11     Thing(int arg1 = 0, double arg2 = 0.0);          // constructor
12     void copyThing(Thing&);
13     void printThing(void) { cout << x << ' ' << y << endl; }
14 };
15
16 Thing::Thing(int arg1, double arg2) : x(arg1), y(arg2)
17 { }
18
19 void Thing::copyThing(Thing& z)
20 {
21   if (this == &z)
22   {
23    cout << "Don't copy me to myself\n";
24    return;
25   }
26   x = z.x;
27   y = z.y;
28 }
29
30 int main(void)
31 {
32   Thing a(5,3.14);
33   Thing b(1);
34   Thing c;
35   a.printThing();
36   b.printThing();
37   c.printThing();
38   c.copyThing(a); // copy Thing-a to c
39   c.printThing();
40   b.copyThing(b); // copy Thing-b to b
```

```
41    b.printThing();
42  }
```

Chaining Functions

Functions may be "chained" by returning a reference to the class type.

Example 5-2 - Chaining Functions

```
1  // File: ex5-2.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  class Circle
7  {
8  private:
9      double radius;
10  public:
11      Circle (double r) : radius(r) {}          // constructor
12      Circle& area() ;
13      Circle& circumference() ;
14  };
15
16  Circle& Circle::area()
17  {
18      cout<<"The area of the Circle is "
19          << 3.14 * radius * radius << endl;
20      return (*this);
21  }
22  Circle& Circle::circumference()
23  {
24      cout<<"The circumference of the Circle is "
25          << 2. * 3.14 * radius << endl;
26      return (*this);
27  }
28
29  int main()
30  {
31      Circle c1(5);
32      c1.area().circumference();
33
34      Circle c2(4.45);
35      c2.circumference().area();
36  }
```

```
******  Output  ******

The area of the circle is 78.5
The circumference of the circle is 31.4
The circumference of the circle is 27.946
The area of the circle is 62.1799
```

- ✔ Why are the area() and circumference() functions not defined as const?

Static Data Members

A static data member is a data member that is shared by all instances of the class. There is only one occurrence of the static data member regardless of how many class objects exist. Static data members are still access controlled (private vs. public). Private static data members may not be accessed by non-member functions. **Non-const** static data members must be defined (initialized) outside of the class definition and outside of member function definitions.

Example 5-3 - Static Data Member

```
1  // File: ex5-3.cpp - static data member
2
3  #include <iostream>
4  using namespace std;
5
6  // function prototype
7  void funk();
8
9  class Circle
10  {
11  private:
12    double radius;
13    static unsigned numCircles;
14  public:
15    Circle (double r = 1.0) : radius(r) { numCircles++;}
16    Circle (const Circle& C) : radius(C.radius) { numCircles++;}
17    ~Circle () {numCircles--;}
18    void printCircleCount();
19  };
20
21  unsigned Circle::numCircles = 0;     // static member definition
22
23  void Circle::printCircleCount()
24  {
25    cout << "Number of Circles = " << numCircles << endl;
26  }
27
28
29  int main()
30  {
31    Circle c1(5.);
32    c1.printCircleCount();
33    Circle c2;
34    c2.printCircleCount();
35    c1.printCircleCount();
36    {
37          Circle c3(1.5);
38          c3.printCircleCount();
39    }
40    c1.printCircleCount();
41    Circle c4(c1);
```

```
42    c1.printCircleCount();
43
44    funk();
45    c1.printCircleCount();
46  }
47
48  void funk()
49  {
50    Circle tempLocal;
51    tempLocal.printCircleCount();
52  }
```

```
*****  Output *****

Number of Circles = 1
Number of Circles = 2
Number of Circles = 2
Number of Circles = 3
Number of Circles = 2
Number of Circles = 3
Number of Circles = 4
Number of Circles = 3
```

Static Member Functions

A static member function is a member function that cannot access the this pointer for an object. This means that the static member function cannot access the non-static data members of a class. It is used to access the static data members of a class. Static member functions are called using the class name and scope resolution operator, providing it has public access, as illustrated in the example below. Static member functions may not be const member functions.

Example 5-4 - Static Member Function

```
1  // File: ex5-4.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  class Circle
7  {
8  private:
9      double radius;
10      static unsigned numCircles;
11   public:
12      Circle(double r = 1.0);
13      ~Circle();
14      static void print_numCircles();
```

```
15      static void resetNumCircles();
16  };
17
18  unsigned Circle::numCircles = 0 ;
19
20  Circle::Circle(double r) : radius(r)
21  {
22      numCircles++;
23  }
24
25  Circle::~Circle()
26  {
27      numCircles--;
28  }
29
30  void Circle::print_numCircles()
31  {
32      cout << "number of Circles = " << numCircles << endl;
33  }
34
35  void Circle::resetNumCircles()
36  {
37      numCircles = 0;
38  }
39
40  int main()
41  {
42      Circle c1(5.);
43      Circle::print_numCircles();
44      Circle c2(4.);
45      Circle::print_numCircles();
46      Circle::resetNumCircles();
47      Circle::resetNumCircles();
48      Circle::print_numCircles();
49      Circle c3(1.);
50      Circle::print_numCircles();
51  }
```

```
*****  Output  *****

number of Circles = 1
number of Circles = 2
number of Circles = 0
number of Circles = 1
```

Friend Functions

A friend function is a non-member function that has access to the private parts of a class. Friendship can be granted in three ways:

- to an independent (non-class member) function
- to a class member function of another class
- to another class (to all functions in that class)

A friend function is always a non-class member. A function outside of a class cannot "seek friendship". Friendship is only granted by a class to another function (or class). A friend has access to all private members.

It is common practice for friend functions to have arguments which include references to the (friendship-granting) class.

Example 5-5 - An independent friend

```
1  // File: ex5-5.cpp - a friend to the Circle class
2
3  #include <iostream>
4  using namespace std;
5
6  class Circle
7  {
8    private:
9      double radius;
10    public:
11     Circle (double r = 1.0) : radius(r) { }
12    friend void print(const Circle&);
13 };
14
15
16 int main(void)
17 {
18   Circle c1(5.);
19   print(c1);
20   Circle c2;
21   print(c2);
22 }
23
24 void print(const Circle& c)
25 {
26   cout << "This Circle has radius " << c.radius<< endl;
27 }
```

Friendly advice

- Friend functions are not affected by their location in a class definition or any access specifiers.
- Granting friendship to another function or class is not reciprocal. If class xyz declares that class abc is a friend, then class xyz is not necessarily a friend to class abc.
- Friendship is not transitive. If class xyz grants friendship to class abc, and class abc grants friendship to class def, then the friendship from xyz is not automatically granted to def.
- Friendship is not inherited. The friend of a base class is not a friend to a class derived from the base. Further, if a base class, B, is a friend to another class, C, classes derived from B are not friends of C. (My friends are not necessarily my children's friends, and my children's friends are not my friends.)
- Class member functions operate on the object that invokes the function. Friend functions operate on objects that are passed as arguments.

Granting friendship to another class

- If class dog grants friendship to class cat, then any function of the cat class can access any member of the dog class.
- The word class is optional in the grant of friendship to another class.

Granting friendship to a function of another class

- To grant friendship to a member of another class, you must indicate the class name and function name using the scope resolution operator.
- If you want the dog class to grant friendship to the meow function of the cat class, you must:

  1) forward declare the dog class.

  2) define the cat class, declaring the meow function, but not defining it.

  3) define the dog class, identifying the friend function, cat::meow().

  4) define the cat member functions.

Example 5-6 - A friend to the card and deck classes

```
1  // File: ex5-6.cpp - a friend to the card and deck classes
2
3  #include <iostream>
4  #include <cstdlib> // needed for rand() function
5  #include <string>
6  using namespace std;
7
8  const string value_name[13] =
9  "two","three","four","five","six","seven","eight","nine","ten","jack
   ","queen","king","ace"};
10
11  const string suit_name[4] =
12  {"clubs","diamonds","hearts","spades"};
13
14  const int HandSize = 5;
15  const int DeckSize = 52;
16
17  class Deck;  // forward declare the Deck class
18
19  class Hand
20  {
21  private:
22      int card[HandSize];
23  public:
24      Hand() { }
25      void dealMe(Deck&);
26      void print(const Deck&) const;
27  };
28
29  class Card
30  {
31  private:
32      int value;
33      int suit;
34  public:
35      Card(int arg = 0) : value(arg%13), suit(arg%4) { }
36      int get_value() const
37      {
38          return value;
39      }
40      int get_suit() const
41      {
42          return suit;
43      }
44      void print(void) const;
45      friend void Hand::print(const Deck&) const;
46  };
47
48  void Card::print() const
49  {
50      cout << value_name[value] << " of " << suit_name[suit] << endl;
```

```
51  }
52
53
54  class Deck
55  {
56      friend class Hand;
57  private:
58      Card d[DeckSize];
59      int next_Card;
60  public:
61      Deck();
62      void shuffle();
63      void deal(int=HandSize);
64      void print() const;
65  };
66
67  Deck::Deck()
68  {
69      for (int i = 0; i < DeckSize; i++) d[i] = Card(i);
70      next_Card = 0;
71  }
72
73  void Deck::shuffle()
74  {
75      int i, k;
76      Card temp;
77      cout << "I am shuffling the Deck\n";
78      for (i = 0; i < DeckSize; i++)
79      {
80          k = rand() % DeckSize;
81          temp = d[i];
82          d[i] = d[k];
83          d[k] = temp;
84      }
85  }
86
87  void Deck::print() const
88  {
89      for (int i = 0; i < DeckSize; i++) d[i].print();
90  }
91
92  void Hand::dealMe(Deck& deck)
93  {
94      for (int i = 0; i < HandSize; i++) card[i] = deck.next_Card++;
95  }
96
97  void Hand::print(const Deck& deck) const
98  {
99      cout << "here is your hand:\n";
100      for (int i = 0; i < HandSize; i++) deck.d[card[i]].print();
101  }
102
```

```
103  int main (void)
104  {
105      Deck poker;
106      poker.shuffle();
107      poker.print();
108      Hand Joe;
109      Hand Mary;
110      Joe.dealMe(poker);
111      Mary.dealMe(poker);
112      cout << "\nOk, Joe ";
113      Joe.print(poker);
114      cout << "\nOk, Mary ";
115      Mary.print(poker);
116  }
```

****** Output ******

```
I am shuffling the deck
ten of hearts
ace of diamonds
queen of hearts
three of hearts
four of diamonds
eight of diamonds
eight of spades
eight of hearts
seven of spades
six of hearts
     .
     .
     .

Ok, Joe here is your hand:
ten of hearts
ace of diamonds
queen of hearts
three of hearts
four of diamonds

Ok, Mary here is your hand:
eight of diamonds
eight of spades
eight of hearts
seven of spades
six of hearts
```

- ✔ Does Hand::print() have to be declared as a friend of the Card class?
- ✔ How can you change the code to eliminate all friend functions?

Example 5-7 - More friendly poker

```
1  // File: ex5-7.cpp
2
3  #include <iostream>
4  #include <cstdlib>            // needed for rand() function
5  #include <string>
6  #include <cassert>
7  #include <ctime>
8  using namespace std;
9
10  const string value_name[13] = {"two","three","four","five","six",
11
   "seven","eight","nine","ten","jack","queen","king","ace"
12                                  };
13  const string suit_name[4] = {"clubs","diamonds","hearts","spades"};
14
15  const int HandSize = 5;
16  const int DeckSize = 52;
17
18  class Deck;
19
20  class Hand
21  {
22      friend void threeOrFourOfAKind(const Hand&);
23
24  public:
25      Hand(const string&, Deck&);
26      void print() const;
27      string getName() const
28      {
29          return name;
30      }
31      const Deck& getDeck() const
32      {
33          return deck;
34      }
35  private:
36      string name;
37      int Card_no[HandSize];
38      Deck& deck;
39      void dealMe(Deck&);
40  };
41
42
43  Hand::Hand(const string& n, Deck& d) : name(n), deck(d)
44  {
45      dealMe(deck);
46  }
47
48  class Card
49  {
50  private:
```

```
51      int value;
52      int suit;
53  public:
54      Card (int x = 0) : value(x%13), suit(x%4) { }
55      int get_value(void) const
56      {
57          return value;
58      }
59      int get_suit() const
60      {
61          return suit;
62      }
63      void print(void) const;
64  };
65
66  void Card::print() const
67  {
68      cout << (value_name[value]) << " of "  << (suit_name[suit]) <<
    endl;
69  }
70
71
72  class Deck
73  {
74      friend void threeOrFourOfAKind(const Hand&);
75      friend class Hand;
76  public:
77      Deck();
78      void print(void) const;
79
80  private:
81      Card d[DeckSize];
82      int nextCard;
83      void shuffle(void);
84  };
85
86
87  Deck::Deck() : nextCard(0)
88  {
89      for (int i = 0; i < DeckSize; i++) d[i] = Card(i);
90      nextCard = 0;
91      shuffle();
92  }
93
94  void Deck::shuffle(void)
95  {
96      int k;
97      Card temp;
98      cout << "I am shuffling the Deck\n";
99      for (int i = 0; i < DeckSize; i++)
100      {
101          k = rand() % DeckSize;
```

```
102          temp = d[i];
103          d[i] = d[k];
104          d[k] = temp;
105      }
106  }
107
108  void Deck::print(void) const
109  {
110      for (int i = 0; i < DeckSize; i++) d[i].print();
111  }
112
113  void Hand::dealMe(Deck& deck)
114  {
115      assert(deck.nextCard < DeckSize-4);
116      for (int i = 0; i < HandSize; i++) Card_no[i] =
   deck.nextCard++;
117  }
118
119  void Hand::print() const
120  {
121      cout << "Ok " << name << ", here is your hand:" << endl;
122      for (int i = 0; i < HandSize; i++) deck.d[Card_no[i]].print();
123      threeOrFourOfAKind(*this);
124      cout << endl;
125
126  }
127
128  int main (void)
129  {
130      srand(time(0));
131      Deck poker;
132
133      Hand curly("Curly",poker);
134      Hand larry("Larry",poker);
135      Hand moe("Moe",poker);
136
137      curly.print();
138      larry.print();
139      moe.print();
140  }
141
142  void threeOrFourOfAKind(const Hand& who)
143  {
144      int temp;
145      int Card_count;
146      for (int i = 0; i < 3; i++)
147      {
148          Card_count = 1;
149          temp = (who.getDeck().d[who.Card_no[i]]).get_value();
150          for (int j = i + 1; j < HandSize; j++)
151              if (temp ==
   who.getDeck().d[who.Card_no[j]].get_value()) Card_count++;
```

```
152          if (Card_count > 2)
153          {
154              cout << "Hey, you have " << Card_count << ' ' <<
   value_name[temp] << "s.\n";
155          }
156      }
157  }
158
```

```
****** Sample Run ******

I am shuffling the Deck
Ok Curly, here is your hand:
jack of diamonds
six of hearts
eight of diamonds
nine of spades
six of clubs

Ok Larry, here is your hand:
ten of hearts
six of spades
seven of hearts
five of spades
eight of spades

Ok Moe, here is your hand:
three of hearts
ten of clubs
three of spades
queen of clubs
three of clubs
Hey, you have 3 threes.
```

Mutual Friendship

What if you want a function of one class to be a friend of a second class, and a function of the second class to be a friend of the first class? How do you do it?

Make the bark() function of the dog class a friend of the cat class and the meow() function a friend of the dog class.

Example 5-8 - Mutual friends

```
1  // File: ex5-8.cpp
2  // File: ex5-8.cpp
3
4  #include <iostream>
5  #include <string>
6  using namespace std;
7
8  class Cat;          // forward declaration
9  class Dog
10  {
11      string name;
12  public:
13      void bark(const Cat&) const;
14      Dog(const string& n) : name(n) { }
15      friend class Cat;
16  };
17  class Cat
18  {
19      string name;
20  public:
21      void meow(const Dog&) const;
22      Cat(const string& n) : name(n) { }
23      friend void Dog::bark(const Cat&) const;
24  };
25
26
27  void Dog::bark(const Cat& c) const
28  {
29      for (size_t i = 0; i < name.size(); i++) cout << " woof ";
30      cout << endl;
31  }
32  void Cat::meow(const Dog& d) const
33  {
34      for (size_t i = 0; i < name.size(); i++) cout << " meow ";
35      cout << endl;
36  }
```

```
37  int main()
38  {
39      Dog bart("Bart");
40      Cat socks("Socks");
41      bart.bark(socks);
42      socks.meow(bart);
43  }
```

```
****** Output ******

 woof  woof  woof  woof  woof
 meow  meow  meow  meow
```

Linked List

A linked list is a data storage technique that allows for variable size container. Linked lists typically consists "connected" nodes containing both data and one or more pointers. The pointer(s) perform the connection to the "next" data item. The most common type of linked lists are single-ended (containing data and one *next* pointer) and double-ended (containing data and two pointers, *next* and *previous*).

The following example illustrates a single ended linked list used to store int data.

Example 5-9 – Linked List

```
1  // file: ex5-9node.h
2
3  #ifndef NODE_H
4  #define NODE_H
5
6  class List;         // forward declaration
7
8  class Node
9  {
10    int         data_;
11    Node*       next_;
12    Node();               // disable the default ctor
13  public:
14    Node(int d,Node* n) { data_ = d; next_ = n; }
15  friend class List;
16  };
17
18  #endif
```

```
1  // file: ex5-9list.h
2
3  #ifndef LIST_H
4  #define LIST_H
5
6  #include "ex5-9node.h"
7
8  class List
9  {
10    Node*       top_;
11  public:
12    List();
13    ~List();
14    void push(int item);
15    int pop();
16    int top() const;
17    void print() const;
18    bool remove(int item);
19    Node* find(int item);
```

```
20  };
21
22  #endif
```

```
1   // file: ex5-9list.cpp
2
3   #include <iostream>
4   #include <cstdlib>
5   using namespace std;
6
7   #include "ex5-9l.h"
8
9   List::List()
10   {
11     top_ = 0;
12   }
13
14
15   List::~List()
16   {
17     Node* temp = top_;
18     while (temp != 0) {
19           top_ = top_ -> next_;
20           delete temp;
21           temp = top_;
22     }
23   }


24   void List::push(int item)
25   {
26     Node* temp = new Node(item,top_);
27     if (!temp) {
28           cerr<<"Unable to allocate memory for a Node. Exiting
     ...\n";
29           exit(-1);
30     }
31     else {
32           top_ = temp;
33     }
34   }
35
36
37   int List::pop()
38   {
39     Node* temp = top_;
40     top_ = top_->next_;
41     int value = temp->data_;
42     delete temp;
43     return value;
44   }
```

```
45
46
47  int List::top() const
48  {
49    return top_ -> data_;
50  }
51
52
53  void List::print() const
54  {
55    Node* temp = top_;
56    while (temp != 0) {
57          cout << temp -> data_ << ' ';
58          temp = temp -> next_;
59    }
60    cout << endl;
61  }
62
63
64  Node*  List::find(int item)
65  {
66    Node* temp = top_;
67    while (temp != 0) {
68          if (temp->data_ == item) return temp;
69          temp = temp -> next_;
70    }
71    return 0;
72  }
73
74
75  bool List::remove(int item)
76  {
77    if (!find(item)) {
78          cerr << item << " is not in the List\n";
79          return false;
80    }
81    Node* temp1 = top_;
82    Node* temp2;
83    if (top_->data_ == item) {
84          top_ = top_ -> next_;
85          delete temp1;
86          return true;
87    }
88    while (temp1->next_->data_ != item) {
89          temp2 = temp1;
90          temp1 = temp1 -> next_;
91    }
92    temp2 = temp1 -> next_;
93    temp1->next_ = temp2 -> next_;
94    delete temp2;
95    return true;
96  }
```

```
1  // file: ex5-9main.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  #include "ex5-9list.h"
7
8  int main (void)
9  {
10    List L;
11    L.push(2);
12    L.push(4);
13    L.push(6);
14    L.push(8);
15    L.push(10);
16    L.print();
17    cout << "top=" << L.top() << endl;
18    if (L.find(2)) cout << 2 << " is in the list\n";
19    if (L.find(5)) cout << 5 << " is in the list\n";
20    if (L.find(6)) cout << 6 << " is in the list\n";
21    if (L.find(10)) cout << 10 << " is in the list\n";
22    L.remove(3);
23    L.remove(6);
24    L.print();
25    L.remove(2);
26    L.remove(10);
27    L.print();
28
29    return 0;
30  }
```

```
****** Output ******

10 8 6 4 2
top=10
2 is in the list
6 is in the list
10 is in the list
3 is not in the List
10 8 4 2
8 4
```

Example 5-10 – Standard Template Library Solution for Example 5-9

```
1  // File ex5-10.cpp - STL linked list example
2
3  #include <iostream>
4  #include <list>
5  #include <algorithm>          // for copy and find algorithms
6  #include <iterator>           // for ostream_iterator
7  using namespace std;
8
9  void print(list<int>& lint)
10  {
11    copy(lint.begin(),lint.end(),ostream_iterator<int>(cout," "));
12    cout << endl;
13  }
14
15  bool find(list<int>& lint, int value)
16  {
17    return find(lint.begin(),lint.end(),value)!=lint.end();
18  }
19
20  int main (void)
21  {
22    list<int> L;
23    L.push_front(2);
24    L.push_front(4);
25    L.push_front(6);
26    L.push_front(8);
27    L.push_front(10);
28    print(L);
29    cout << "top=" << *L.begin() << endl;
30
31    if (find(L,2)) cout << 2 << " is in the list\n";
32    if (find(L,5)) cout << 5 << " is in the list\n";
33    if (find(L,6)) cout << 6 << " is in the list\n";
34    if (find(L,10)) cout << 10 << " is in the list\n";
35    L.remove(3);
36    L.remove(6);
37    print(L);
38    L.remove(2);
39    L.remove(10);
40    print(L);
41    return 0;
42  }
```

```
******  Output  ******

10 8 6 4 2
top=10
2 is in the list
6 is in the list
10 is in the list
```

```
10 8 4 2
8 4
```

Function and Operator Overloading

Function Overloading

An overloaded function is one with different signatures. A function's signature is its argument list. For example,

```
void funk(int);
int funk(float);
int funk(int,float,char*);
```

These three functions have the same name, but different signatures. A signature represents a function's name and its argument list, not the return type. Consider,

```
void funky(int);
int funky(int);
```

These two functions have the same signature and could not be defined in the same program scope. In order to overload functions, they must have different signatures. Your compiler would not allow this.

Furthermore, functions with default arguments do not constitute different signatures, even though their function calls appear different. The functions

```
int flunk(int);
int flunk(int = 5);
```

may be called as: flunk(1) or flunk();

So, even though the function calls are unique, your compiler would disallow these definitions.

For overloaded functions, the compiler selects a function with the specified name and matching the argument list according to a “best match” criteria. The “best match” is made using the following order precedence:

1) exact matches or trivial conversions (array names to pointers, a type to a const type)
2) promotions - char to int, short to int, bool to int and float to double (not int to long)
3) standard conversions – “demotions” , integral types to floating types, floating to integral
4) user-defined conversion functions - constructors, conversion operators
5) ellipsis - similar to the way printf and scanf work with a variable number of arguments. Example: void printf(...).

Example 6-1 - Function overloading – non-exact matches

This example demonstrates some function overloading and how the compiler handles non-exact matches.

```
1  // File: ex6-1.cpp - function overloading
2
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6
7  class Circle
8  {
9      double radius;
10  public:
11      Circle(double r = 0) : radius(r) { }
12      static const double PI;
13      double getRadius() const
14      {
15          return radius;
16      }
17  };
18
19  class Rectangle
20  {
21      double length, width;
22  public:
23      Rectangle(double len, double wid) : length(len), width(wid) { }
24      double getLength() const
25      {
26          return length;
27      }
28      double getWidth() const
29      {
30          return width;
31      }
32  };
33
34  class Triangle
35  {
36      double a, b, c;
37  public:
38      Triangle(double arg1, double arg2, double arg3) : a(arg1),
    b(arg2), c (arg3) { }
39      double geta() const
40      {
41          return a;
42      }
43      double getb() const
44      {
45          return b;
46      }
```

```
47      double getc() const
48      {
49          return c;
50      }
51      double area() const;
52  };
53
54  // Overloaded area function declarations
55  double area(const Circle& C);
56  double area(const Rectangle& R);
57  double area(const Triangle& T);
58
59
60  int main()
61  {
62      Circle c(10.0);
63      Rectangle r(4.0,5.0);
64      Triangle t(3.0,4.0,5.0);
65
66      cout << "Circle c has area " << area(c) << endl;
67      cout << "Rectangle r has area " << area(r) << endl;
68      cout << "Triangle t has area " << area(t) << endl;
69  }
70
71  const double Circle::PI = 3.1415926535897;
72
73  double Triangle::area() const
74  {
75      double s = .5 * (a + b + c);
76      return sqrt(s*(s-a)*(s-b)*(s-c));
77  }
78
79
80  double area(const Circle& C)
81  {
82      return Circle::PI * C.getRadius() * C.getRadius();
83  }
84
85  double area(const Rectangle& R)
86  {
87      return R.getLength() * R.getWidth();
88  }
89
90  double area(const Triangle& T)
91  {
92      return T.area();
93  }
```

```
******  Output  ******

Circle c has area 314.159
Rectangle r has area 20
```

```
Triangle t has area 6
```

Example 6-2 - copyFile
This example makes use of function overloading. This purpose of this program is to build a copyfile command that may be used at the operating system level. That is, it will emulate the DOS copy command or the UNIX cp command. Following the example code are sample command-line compile commands that demonstrate compilation and linking for the command-line environment. The program also makes use of command-line arguments.

The program allows the user to:

1) copy one file to another
2) append one file to another
3) convert a file to uppercase
4) convert a file to lowercase
5) copy one file to another only if the target file does not exist
6) copy part of one file to another (from line# to line#)
7) copy part of one file to another (from line# to end of file)
8) copy part of one file to another (from beginning of file to line#)

```
1  // File: ex6-2.cpp - overloaded functions
2
3  #include <cctype>
4  #include <cstdlib>
5  #include <string>
6  #include <iostream>
7  #include <fstream>
8  #include <sstream>  // for ostringstream
9  using namespace std;
10
11  #ifdef __GNUG__          // gnu compilers
12  #include <unistd.h>
13  #else                    // windows compilers
14  #include <io.h>         // for access() function
15  #endif
16
17  // Overloaded function prototypes
18
19  void copyFile(const string& fn1, const string& fn2);
20  void copyFile(const string& fn1, const string& fn2, const string&
   option);
21  void copyFile(const string& fn1, const string& fn2, const string&
   fromto, int lineno);
22  void copyFile(const string& fn1, const string& fn2, const string&
   from, int line1,
23                const string& to, int line2);
24
25  void errorMessage(const string& message);
26  void errorMessage(const string& message, const string& filename);
27
28  void displayCmdSyntax()
29  {
```

```
30      fprintf(stderr,"Usage:\n");
31      fprintf(stderr,"\tcopyFile <file1> <file2>\n");
32      fprintf(stderr,"\tcopyFile <file1> <file2> -append\n");
33      fprintf(stderr,"\tcopyFile <file1> <file2> -upper\n");
34      fprintf(stderr,"\tcopyFile <file1> <file2> -lower\n");
35      fprintf(stderr,"\tcopyFile <file1> <file2> -noreplace\n");
36      fprintf(stderr,"\tcopyFile <file1> <file2> -from <line#> -to
   <line#>\n");
37      fprintf(stderr,"\tcopyFile <file1> <file2> -from <line#>\n");
38      fprintf(stderr,"\tcopyFile <file1> <file2> -to <line#>\n");
39  }
40
41  int main(int argc, char* argv[])
42  {
43      if (argc<3 || *argv[1] == '?')
44      {
45          displayCmdSyntax();
46          exit(0);
47      }
48
49      switch (argc)
50      {
51      case 3:
52          copyFile(argv[1],argv[2]);
53          break;
54      case 4:
55          copyFile(argv[1],argv[2],argv[3]);
56          break;
57      case 5:
58          copyFile(argv[1],argv[2],argv[3],atoi(argv[4]));
59          break;
60      case 7:
61          copyFile(argv[1],argv[2],argv[3],atoi(argv[4]),
62                   argv[5],atoi(argv[6]));
63          break;
64      default:
65          errorMessage("Error: Invalid syntax\n");
66      }
67      printf("Ok\n");
68      return 0;
69  }
70
71  // copies file fn1 to file fn2
72  void copyFile(const string& fn1, const string& fn2)
73  {
74      char     buffer[1024];
75      ifstream fin(fn1.c_str());
76      ofstream fout(fn2.c_str());
77      if (!fin) errorMessage("Unable to open input file",fn1);;
78      if (!fout) errorMessage("Unable to open input file",fn2);
79      while (fin.getline(buffer,sizeof(buffer))) fout << buffer <<
   endl;
```

```
80  }
81
82  // copies file fn1 to file fn2 using append, upper, lower or
   noreplace
83  void copyFile(const string& fn1, const string& fn2, const string&
   option)
84  {
85      char     buffer[1024];
86      // check for valid option
87      if (option == "-upper" || option =="-lower" || option =="-
   append" || option =="-noreplace") /* keep going */ ;
88      else errorMessage("Invalid option");
89
90      ifstream fin(fn1.c_str());
91      ofstream fout;
92      if (!fin) errorMessage("Unable to open input file",fn1);;
93
94      if (option == "-upper" || option =="-lower")
95      {
96          fout.open(fn2.c_str());
97          if (!fout) errorMessage("Unable to open input file",fn2);
98      }
99
100      // append option
101      if (option == "-append")
102      {
103          fout.open(fn2.c_str(),ios_base::app);
104          while (fin.getline(buffer,sizeof(buffer)))
105          {
106              fout << buffer << endl;
107          }
108          return;
109      }
110      // upper & lower case options
111      if (option=="-upper" || option=="-lower")
112      {
113          char ch;
114          while ((ch = fin.get()) != EOF)
115          {
116              if (option =="-upper") fout.put(toupper(ch));
117              else fout.put(tolower(ch));
118          }
119      }
120
121      // noreplace option
122      if (option == "-noreplace")
123      {
124          // check for the existance of file fn2
125          // Note: The access() function is non-ANSI, but available
   on Borland/UNIX compilers
126          // MS compilers use _access
```

```
127          if (access(fn2.c_str(),0)==0) errorMessage("Noreplace
   error for output file",fn1);
128          else copyFile(fn1,fn2);
129      }
130  }
131
132  // copy part of fn1 to fn2 (from linto or to lineno)
133  void copyFile(const string& fn1, const string& fn2, const string&
   fromto, int lineno)
134  {
135      char    buffer[1024];
136      ifstream fin(fn1.c_str());
137      ofstream fout(fn2.c_str());
138      if (!fin) errorMessage("Unable to open input file",fn1);;
139      if (!fout) errorMessage("Unable to open input file",fn2);
140
141      // copy "from" lineno to the end of file
142      if (fromto =="-from")
143      {
144          // read records up to "from"
145          for (int i = 1; i<lineno; i++)
146          {
147              fin.getline(buffer,sizeof(buffer));
148              if (!fin.good())           // make sure the records
   are read up to "from"
149              {
150                  ostringstream sout;
151                  sout << "Unable to read past record " << i;
152                  errorMessage(sout.str());
153              }
154          }
155          // read the rest of the file & write it out
156          while (fin.getline(buffer,sizeof(buffer))) fout << buffer
   << endl;
157      }
158      else if (fromto =="-to")
159      {
160          // read records up to "to" & write them out
161          for (int i = 0; i<lineno; i++)
162          {
163              fin.getline(buffer,sizeof(buffer));
164              if (!fin.good())           // make sure the records
   are read up to "from"
165              {
166                  ostringstream sout;
167                  sout << "Unable to read past record " << i;
168                  errorMessage(sout.str());
169              }
170              fout << buffer << endl;
171          }
172      }
173      else errorMessage("Invalid command syntax");
```

```
174  }
175
176  void copyFile(const string& fn1, const string& fn2, const string&
   from, int line1,
177                const string& to, int line2)
178  {
179      char    buffer[256];
180      int     i;
181      // check the syntax
182      if (from != "-from"||to != "-to")
183          errorMessage("Invalid from/to syntax\n");
184      if (line1>line2) errorMessage("Invalid 'from' > 'to'\n");
185
186      ifstream fin(fn1.c_str());
187      ofstream fout(fn2.c_str());
188      if (!fin) errorMessage("Unable to open input file",fn1);;
189      if (!fout) errorMessage("Unable to open input file",fn2);
190
191      // read records up to "from" line1
192      for (i = 1; i<line1; i++)
193      {
194          fin.getline(buffer,sizeof(buffer));
195          if (!fin.good())          // make sure the records are
   read up to "from"
196          {
197              ostringstream sout;
198              sout << "Unable to read past record " << i;
199              errorMessage(sout.str());
200          }
201      }
202      for (i = line1; i<=line2; i++)
203      {
204          fin.getline(buffer,sizeof(buffer));
205          if (!fin.good())          // make sure the records are
   read up to "from"
206          {
207              ostringstream sout;
208              sout << "Unable to read past record " << i;
209              errorMessage(sout.str());
210          }
211
212          fout << buffer << endl;
213      }
214  }
215
216  void errorMessage(const string& msg)
217  {
218      cerr << msg << endl;
219      exit (-1);
220  }
221
222  void errorMessage(const string& message, const string& filename)
```

```
223  {
224       cerr << message << ' ' << filename << endl;
225       exit(-1);
226  }
```

Compile command for DOS using MS Visual Studio Express 2012:

cl ex6-2.cpp

Note**:** Before you can perform a command-line compile, you must run **vcvars32.bat**. This program and the **cl.exe** for the command-line compile are found in the directory: \Program Files\Microsoft Visual Studio 11.0\VC\bin

Compile command for GNU (for UNIX/Linux) compiler:

g++ ex6-2.cpp -o cf

```
******  Sample Program Run   ******

C:\td124a>cf ?
Usage:
        cf <file1> <file2>
        cf <file1> <file2> -append
        cf <file1> <file2> -upper
        cf <file1> <file2> -lower
        cf <file1> <file2> -noreplace
        cf <file1> <file2> -from <line#> -to <line#>
        cf <file1> <file2> -from <line#>
        cf <file1> <file2> -to <line#>

C:\td124a>cf ex6-2.inp ex6-2.out
Ok

C:\td124a>cf ex6-2.inp ex6-2.out -from 2 -to 3
Ok

C:\td124a>cf ex6-2.inp ex6-2.out -noreplace
Noreplace error for output file

C:\td124a>cf ex6-2.inp ex6-2.out -form 2 -to 3
Invalid from/to syntax
```

Operator Overloading

Operators in C++ may be overloaded in the same way that functions are overloaded. In C, the + (plus) operator is "overloaded" to work for int or float values. In C++, this concept is extended to include class types.

Notes:

- You may overload the following operators:

  ```
  +   -   *   /   %   ^   &   |
  ~   !   ,   =   <   >   <=  >=
  ++  --  <<  >>  ==  !=  &&  ||
  +=  -+  /=  %=  ^=  &=  |=  *=
  <<= >>= []  ()  ->  ->* new  delete
  ```

- Most operators may be overloaded, both binary and unary operators. The following operators may not be overloaded:
  `.` direct member
  `.*` direct pointer to member
  `::` scope resolution
  `? :` ternary

- To overload an operator, create a function called operator@ where @ is the operator symbol you wish to overload.

- Operator precedence is still in effect for overloaded operators and may not be changed.

- Default arguments are not allowed in overloaded operator functions.

- For an expression involving binary operators, A + B means:
  `A.operator+(B)` if operator+() is a class member function
  `operator+(A,B)` if operator+() is a non-class member function

- An overloaded operator function may be defined as a class member function, a friend function, or even a non-friend function.

- You may not overload an operator (redefine) for the built-in primitive types. In other words, if a and b are ints, then a+b will always be (int) a+b.

- You may not create any new operator symbols

Example 6-3 - Fraction class with overloaded + and ! operators

```
1  // File: ex6-3.cpp overloaded + and ! operator for fraction class
2
3  #include <iostream>
4  using namespace std;
5
6  class fraction
7  {
8    private:
9        int numer;
10       int denom;
11   public:
12       fraction(int n = 0, int d = 1);
13       void operator!(void) const;
14       fraction operator+(const fraction&);
15 };
16
17 fraction::fraction(int n, int d)
18 {
19   numer = n;
20   denom = d;
21 }
22
23 void fraction::operator!(void) const
24 {
25   cout << numer << '/' << denom << endl;
26   return;
27 }
28
29 fraction fraction::operator+(const fraction& f2)
30 {
31   fraction temp(0,0);
32   temp.numer = numer * f2.denom + f2.numer * denom;
33   temp.denom = denom * f2.denom;
34   return temp;
35 }
36
37 int main(void)
38 {
39   fraction f(3,4);
40   fraction g(2,3);
41   fraction h = f + g;         // Do you need a default ctor here?
42   !h;                                                  // prints 17/12
43
44   return 0;
45 }
```

✔ In this example operator+ returns a fraction by value. Is it possible or appropriate to have the function return by reference or have a void return?

✔ Line 41: What is the difference between ***fraction h = f + g;*** and ***fraction h(f+g);***

In this example, operator+ is defined as a friend function.

Example 6-4 - A friendly overloaded +

```
1  #include <iostream>
2  using namespace std;
3
4  class fraction {
5    private:
6       int numer, denom;
7    public:
8      fraction(int n = 0, int d = 1);
9      void operator!(void) const;
10    friend  fraction operator+(const fraction&,const fraction&);
11 };
12
13
14 fraction::fraction(int n, int d) {
15   numer = n;
16   denom = d;
17 }
18
19 void fraction::operator!(void) const {
20   cout << numer << '/' << denom << endl;
21 }
22
23 // fraction friend function
24 fraction operator+(const fraction& f1,const fraction& f2) {
25   fraction temp(f1.numer *f2.denom + f2.numer * f1.denom,
26      f1.denom * f2.denom);
27   return temp;
28 }
29
30
31 int main(void) {
32   fraction f(3,4);
33   fraction g(2,3);
34   fraction h = f + g;
35   !f;
36   !g;
37   !h;
38   return 0;
39 }
```

```
******  Output  ******

3/4
2/3
17/12
```

✔ What's the better approach, example 6-3 or example 6-4?

This example demonstrates a more "complete" set of overloaded operators for the fraction class. Notice that all operators are specified as member functions

Example 6-5 - The Overloaded fraction class

```
1  // File: ex6-5.cpp
2  #include <iostream>
3  #include <cassert>
4  using namespace std;
5
6  class fraction {
7        int numer, denom;
8    public:
9        fraction(int = 0, int = 1);
10       void operator!(void) const;       // print the fraction
11       fraction& operator~(void);        // reduce the fraction
12       fraction operator-(void) const;   // negative of fraction
13       fraction operator*(void) const;   // reciprocal of fraction
14       fraction& operator+=(const fraction&);
15       fraction& operator-=(const fraction&);
16       fraction& operator*=(const fraction&);
17       fraction& operator/=(const fraction&);
18       fraction operator+(int) const;
19       fraction operator-(int) const;
20       fraction operator*(int) const;
21       fraction operator/(int) const;
22       bool operator>(const fraction&) const;
23       bool operator<(const fraction&) const;
24       bool operator>=(const fraction&) const;
25       bool operator<=(const fraction&) const;
26       bool operator==(const fraction&) const;
27       bool operator!=(const fraction&) const;
28       fraction operator+(const fraction&) const;
29       fraction operator-(const fraction&) const;
30       fraction operator*(const fraction&) const;
31       fraction operator/(const fraction&) const;
32       fraction& operator++();  // prefix op returns by ref
33       fraction operator++(int);      // postix op returns by value
34 };
35
36 // member function definitions
37 fraction::fraction(int n, int d) {
38   assert(d != 0);
39   numer = n;
40   denom = d;
41 }
42
43 // print the fraction
44 void fraction::operator!(void) const {
45   cout << numer << '/' << denom << endl;
46 }
47
```

```
48  // reduce the fraction
49  fraction& fraction::operator~(void) {
50  int min;
51  // find the minimum of the denom and numer
52    min = denom < numer ? denom : numer;
53    for (int i = 2; i <= min; i++) {
54        while ((numer % i == 0) && (denom % i == 0)) {
55            numer /= i;
56            denom /= i;
57        }
58    }
59    return *this;
60  }
61
62  // negate the fraction
63  fraction fraction::operator-(void) const {
64    return fraction(-numer,denom);
65  }
66
67  // fraction reciprocal
68  fraction fraction::operator*(void) const {
69    return fraction(denom,numer);
70  }
71
72  fraction& fraction::operator+=(const fraction& f) {
73    numer = numer*f.denom+denom*f.numer;
74    denom = denom*f.denom;
75    return *this;
76  }
77
78  fraction& fraction::operator-=(const fraction& f) {
79    *this += (-f);
80    return *this;
81  }
82
83  fraction& fraction::operator*=(const fraction& f) {
84    numer = numer*f.numer;
85    denom = denom*f.denom;
86    return *this;
87  }
88
89  fraction& fraction::operator/=(const fraction& f) {
90    *this *= (*f);
91    return *this;
92  }
93
94  bool fraction::operator>(const fraction& f) const {
95    return (float) numer/denom > (float) f.numer/f.denom;
96  }
97
98  bool fraction::operator<(const fraction& f) const {
99    return f>*this;
```

```
100  }
101
102  bool fraction::operator==(const fraction& f) const {
103    return numer*f.denom == denom*f.numer;
104  }
105
106  bool fraction::operator!=(const fraction& f) const {
107    return !(*this == f);
108  }
109
110  bool fraction::operator<=(const fraction& f) const {
111    return !(*this > f);
112  }
113
114  bool fraction::operator>=(const fraction& f) const {
115    return !(*this<f);
116  }
117
118  fraction fraction::operator+(const fraction& f) const {
119    return fraction(numer*f.denom+denom*f.numer,denom*f.denom);
120  }
121
122  fraction fraction::operator-(const fraction& f) const {
123    return fraction(numer*f.denom-denom*f.numer,denom*f.denom);
124  }
125
126  fraction fraction::operator*(const fraction& f) const {
127    return fraction(numer*f.numer,denom*f.denom);
128  }
129
130  fraction fraction::operator/(const fraction& f) const {
131    return (*this) * (*f);
132  }
133
134  fraction fraction::operator+(int i) const {
135    return fraction(numer+i*denom,denom);
136  }
137
138  fraction fraction::operator-(int i) const {
139    return (*this) + -i;
140  }
141
142  fraction fraction::operator*(int i) const {
143    return fraction(numer*i,denom);
144  }
145
146  fraction fraction::operator/(int i) const {
147    return fraction(numer,i*denom);
148  }
149
150  // prefix increment operator
151  fraction& fraction::operator++() {
```

```
152   numer += denom;
153   return *this;
154  }
155
156  // postfix increment operator
157  fraction fraction::operator++(int) {   // Note dummy int argument
158    fraction temp = *this;
159    ++(*this);                          // call the prefix operator
160    return temp;
161  }
162
163
164  int main(void)
165  {
166    fraction f(3,4);           // initialize fraction f & g
167    fraction g(1,2);
168    cout << "!f ";  !f;
169    cout << "!g ";  !g;
170    cout << endl;
171    cout << "-g ";  !-g;
172    cout << "*g ";  !*g;
173    fraction h = g + f;
174    cout << endl;
175    cout << "h=g+f " << " !h ";  !h;
176    cout << "!~h ";  !~h;
177    cout << endl;
178    cout << "f+g ";  ! (f + g);
179    cout << "f-g ";  ! (f - g);
180    cout << "f*g ";  ! (f * g);
181    cout << "f/g ";  ! (f / g);
182    cout << endl;
183    cout << "f+=g "; !~(f+=g);
184    cout << "f-=g "; !~(f-=g);
185    cout << "f*=g "; !~(f*=g);
186    cout << "f/=g "; !~(f/=g);
187    cout << endl;
188    cout << "f<g " << (f<g) << endl;
189    cout << "f>g " << (f>g) << endl;
190    cout << "f==g " << (f==g) << endl;
191    cout << "f!=g " << (f!=g) << endl;
192    cout << "f<=g " << (f<=g) << endl;
193    cout << "f>=g " << (f>=g) << endl;
194    cout << endl;
195    cout << "f+5 ";  !(f+5);
196    cout << "f-5 ";  !(f-5);
197    cout << "f*5 ";  !(f*5);
198    cout << "f/5 ";  !(f/5);
199    cout << endl;
200    cout << "f+=5 "; f+=5; cout << "!~f "; !~f;     // What's this?
201    cout << "++f "; !++f; cout << "f="; !f;
202    cout << "f++ "; !f++; cout << "f="; !f;
203
```

```
204     return 0;
205   }
```

```
******  Output  ******

!f 3/4
!g 1/2

-g -1/2
*g 2/1

h=g+f  !h 10/8
!~h 5/4

f+g 10/8
f-g 2/8
f*g 3/8
f/g 6/4

f+=g 5/4
f-=g 3/4
f*=g 3/8
f/=g 3/4

f<g 0
f>g 1
f==g 0
f!=g 1
f<=g 0
f>=g 1

f+5 23/4
f-5 -17/4
f*5 15/4
f/5 3/20

f+=5 !~f 23/4
++f 27/4
f=27/4
f++ 27/4
f=31/4
```

Should any of these member functions be specified as friend functions?

Why do operator~ and unary operator- have different return types?

How do the increment operators work?

Example 6-6 - "More power"

```
1  // File: ex6-6.cpp
2
3  #include <iostream>
4  #include <cstdlib>
5  #include <cmath>
6  using namespace std;
7
8  class Integer
9  {
10   private:
11    long x;
12   public:
13    Integer(long i) { x = i;}
14    long operator^(int);
15 };
16
17 long Integer::operator^(int power)
18 {
19   if (power == 0) return 1;
20   long temp = x;
21   for (int i = 1; i < power; i++) temp *= x;
22   return temp;
23 }
24
25
26 class Real
27 {
28    double d;
29   public:
30    Real(double arg) { d = arg;}
31    double operator^(double);
32 };
33
34 double Real::operator^(double power)
35 {
36   if (d == 0 && power == 0)
37   {
38     cout << "0 ^ 0 is undefined\n";
39     exit (1);
40   }
41
42   if (d < 0 && power != floor(power))
43   {
44     cout <<
45         "You may only take integer powers of negative numbers\n";
46     exit (1);
47   }
48
49  return pow(d,power);
50 }
```

```
51  int main (void)
52  {
53    Integer z(2), y(3);
54    cout << (z^5) << endl;
55    cout << (y^0) << endl;
56    Real r1(3.14), r2(6.02e23), r3(1.2345);
57    cout<< (r1^2) << endl;
58    cout<< (r2^3) << endl;
59    cout<< (r3^0) << endl;
60    cout<< (r1^3.14) << endl;
61    Real r4(-1.4);
62    cout << (r4^3) << endl;
63    cout << (r4^1.3) << endl;
64
65    return 0;
66  }
```

```
******  Output  ******

32
1
9.8596
2.18167e+71
1
36.3378
-2.744
```

You may only take integer powers of negative numbers

What if you want to evaluate an expression like πr^2 ? Is this correct?

```
3.141592654*r^2
```

Unary vs Binary, Member vs.Non-Member

Only two types of overloaded operator functions may exist, unary or binary, and they may be defined as member or non-member functions. So, there are only four ways to define these functions. The following table summarizes these possibilities. Assume @ represents an overloaded operator. There is, of course, no such operator available in C++.

		Unary Operator	**Binary Operator**
Member function	prototype	**? operator@();**	**? operator@(Arg);**
	Functional notation call	Arg1.operator@() [2]	Arg1.operator@(Arg2)[1]
	Infix notation call	@Arg1[1]	Arg1 @ Arg2[1]
Non-member function	prototype	**? operator@(Arg1);** [1]	**? operator@(Arg1,Arg2);**[3]
	Functional notation call	operator@(Arg1) [1]	operator@(Arg1,Arg2) [2]
	Infix notation call	@Arg1 [1]	Arg1 @ Arg2 [2]

[2] Arg1 would have to be a class object
[3] Either Arg1 or Arg2 or both would have to be a class object

Example 6-7 - Matrix Arithmetic

The following example is an implementation of Matrix addition. It is meant to demonstrate the overloaded + and = operators. This example also uses the this operator in member functions, so that objects can be followed in the program using the program output.

```
1  // File: ex6-7.cpp
2
3  #include <iostream>
4  #include <cstdlib>
5  using namespace std;
6
7  class Matrix
8  {
9  private:
10      int** element;
11      int rows;
12      int cols;
13      void alloc(void);
14      void release(void);
15  public:
16      Matrix(int = 0, int = 0);  // also default constructor
17      Matrix(const Matrix&); // copy constructor
18      ~Matrix();
19      void print(void) const;
20      Matrix operator+(const Matrix&) const;
21      Matrix& operator=(const Matrix&);
22  };
23
24  Matrix::Matrix(int r, int c) : rows(r), cols(c)
25  {
26      cout << "Constructor called for object " << this <<endl;
27      alloc();
28
29      // initialize Matrix elements with random numbers 0-9
30      for (int i = 0; i < rows; i++)
31          for (int j = 0; j < cols; j++)
32              element[i][j] = rand()%10;
33  }
34
35  Matrix::Matrix(const Matrix& arg) : rows(arg.rows), cols(arg.cols)
36  {
37      cout << "\nIn copy constructor for object " << this;
38      cout << ", argument: " << &arg << endl;
39
40      alloc();
41      for (int i = 0; i < rows; i++)
42          for (int j = 0; j < cols; j++)
43              element[i][j] = arg.element[i][j];
44  }
45
46  Matrix::~Matrix(void)
```

```
47  {
48      cout << "\n~~ Destructor called for object: " << this << endl;
49
50      release();
51  }
52
53  void Matrix::alloc(void)       // allocate heap memory for elements
54  {
55      cout << "Allocate heap memory for Matrix " << this << "
   elements\n";
56
57      element = new int*[rows];
58      for (int i = 0; i < rows; i++)
59          element[i] = new int[cols];
60  }
61
62  void Matrix::release(void)
63  {
64      cout << "I got rid of Matrix " << this << "'s elements\n";
65
66      for (int i = 0; i < rows; i++)
67          delete [] element[i];
68      delete [] element;
69  }
70
71  void Matrix::print(void) const
72  {
73      cout << "\nMatrix values for object: "<< this << endl;
74
75      for (int i = 0; i < rows; i++)
76      {
77          for (int j = 0; j < cols; j++)
78              cout << element[i][j] << '\t';
79          cout << endl;
80      }
81  }
82
83  Matrix Matrix::operator+(const Matrix& arg) const
84  {
85      cout << "\nExecuting operator+ for object: " << this;
86      cout << ", argument: " << &arg << endl;
87
88      if (rows != arg.rows || cols != arg.cols)
89      {
90          cerr << "Invalid Matrix addition\n";
91          return (*this);
92      }
93
94      Matrix temp(rows,cols);
95
96      for (int i = 0; i < rows; i++)
97          for (int j = 0; j < cols; j++)
```

```
98              temp.element[i][j] = element[i][j] + arg.element[i][j];
99
100     temp.print();
101     return temp;
102 }
103
104 Matrix& Matrix::operator=(const Matrix& arg)
105 {
106     cout << "\nExecuting operator= for object: " << this;
107     cout << ", argument: " << &arg << endl;
108
109     // Make sure rows and cols match the argument
110     if (rows != arg.rows || cols != arg.cols)
111     {
112         release();
113         rows = arg.rows;
114         cols = arg.cols;
115         alloc();
116     }
117
118     for (int i = 0; i < arg.rows; i++)
119         for (int j = 0; j < arg.cols; j++)
120             element[i][j] = arg.element[i][j];
121
122     return *this;
123 }
124
125 int main(void)
126 {
127     Matrix A(3,4), B(3,4), C;
128     A.print();
129     B.print();
130     C.print();
131     C = A + B;
132     C.print();
133 }
```

****** OUTPUT ******

```
Constructor called for object 0x28fee8
Allocate heap memory for Matrix 0x28fee8 elements
Constructor called for object 0x28fedc
Allocate heap memory for Matrix 0x28fedc elements
Constructor called for object 0x28fed0
Allocate heap memory for Matrix 0x28fed0 elements

Matrix values for object: 0x28fee8
1       7       4       0
9       4       8       8
2       4       5       5

Matrix values for object: 0x28fedc
```

```
1       7       1       1
5       2       7       6
1       4       2       3

Matrix values for object: 0x28fed0

Executing operator+ for object: 0x28fee8, argument: 0x28fedc
Constructor called for object 0x28fe3c
Allocate heap memory for Matrix 0x28fe3c elements

Matrix values for object: 0x28fe3c
2       14      5       1
14      6       15      14
3       8       7       8

In copy constructor for object 0x28fef4, argument: 0x28fe3c
Allocate heap memory for Matrix 0x28fef4 elements

~~ Destructor called for object: 0x28fe3c
I got rid of Matrix 0x28fe3c's elements

Executing operator= for object: 0x28fed0, argument: 0x28fef4
I got rid of Matrix 0x28fed0's elements
Allocate heap memory for Matrix 0x28fed0 elements

~~ Destructor called for object: 0x28fef4
I got rid of Matrix 0x28fef4's elements

Matrix values for object: 0x28fed0
2       14      5       1
14      6       15      14
3       8       7       8

~~ Destructor called for object: 0x28fed0
I got rid of Matrix 0x28fed0's elements

~~ Destructor called for object: 0x28fedc
I got rid of Matrix 0x28fedc's elements

~~ Destructor called for object: 0x28fee8
I got rid of Matrix 0x28fee8's elements
```

✔ How would you change the operator=() function so that you could assign a matrix with a different number of rows or columns?

Is it a Copy Constructor or a Default Constructor and an Assignment Operator?

If you instantiate a class object x using an existing object y, such as,

Test x(y);

It is assumed that the copy constructor is called to perform the creation of the object. Further, if you use the syntax,

Test x = y;

Then, the same copy constructor is called. Is that correct, or is the default constructor called, then the assignment operator? Consider the following example.

Example 6-7a - Copy Constructor or a Default Constructor and an Assignment Operator?

```
// File: Ex6-7a.cpp -
// Copy Constructor or a Default Constructor and Assignment Operator

#include <iostream>
using namespace std;

class Test
{
public:
 Test() { cout << "default ctor: " << this << endl; }
 Test(const Test& arg) { cout << "copy ctor: " << this << " argument=" << &arg << endl; }
 Test& operator=(const Test& arg) { cout << "operator=: " << this << " argument=" << &arg << endl; return *this; }
 Test operator+(const Test& arg) { cout << "operator+: " << this << " argument=" << &arg << endl; return *this; }
 Test operator-(const Test& arg) { cout << "operator-: " << this << " argument=" << &arg << endl; Test temp;  return temp; }
};

int main()
{
 Test one;
 Test two(one);
 one = two;
 Test three = one;
 Test four = one + two;
 Test five = one - two;
 cout << "&five=" << &five << endl;

 return 0;
}
```

******* Output *******

MS Visual C++ 2008

```
default ctor: 002BFABB                                          (19)
copy ctor: 002BFAAF argument=002BFABB                           (20)
operator=: 002BFABB argument=002BFAAF                           (21)
copy ctor: 002BFAA3 argument=002BFABB                           (22)
operator+: 002BFABB argument=002BFAAF                           (23)
copy ctor: 002BFA97 argument=002BFABB                           (23)
operator-: 002BFABB argument=002BFAAF                           (24/14)
default ctor: 002BF997                                          (14)
copy ctor: 002BFA8B argument=002BF997                           (24)
&five=002BFA8B                                                  (25)
```

g++ 4.4.0 on Linux

```
default ctor: 0x7fff43e46cff                                    (19)
copy ctor: 0x7fff43e46cfe argument=0x7fff43e46cff               (20)
operator=: 0x7fff43e46cff argument=0x7fff43e46cfe               (21)
copy ctor: 0x7fff43e46cfd argument=0x7fff43e46cff               (22)
operator+: 0x7fff43e46cff argument=0x7fff43e46cfe               (23)
copy ctor: 0x7fff43e46cfc argument=0x7fff43e46cff               (23)
operator-: 0x7fff43e46cff argument=0x7fff43e46cfe               (24/14)
default ctor: 0x7fff43e46cfb                                    (14)
&five=0x7fff43e46cfb                                            (25)
```

Comments

Line 19: default constructor call
Line 20: copy constructor call
Line 21: assignment operator call
Line 22: copy constructor call. Is this the answer to our question? Not yet, let's look further.
Line 23: operator+ is called, then the copy constructor (same result on both compilers)
Line 24: Now here's where it gets interesting. The call to operator-() invokes a default Test constructor call in line 14. Then the MS compiler copies the temporary object using the copy constructor, since the return is "by value". The g++ compiler appears to by "optimizing this onstructor call out". So, it appear that th g++ compiler does not use the copy compiler in this case to perform the "return by value" copy.
Line 25: This illustrates that the five object was created by the copy constructor for the MS compiler and by the default constructor for the g++ compiler.

Who ever said this was going to be easy? Haven't we got better things to do with our time?

Type Conversions

Operator overloading comes into play in converting one type to other. You already have experience using two methods of conversions. First, you can use a cast to convert one primitive type to another. The second method simply involves constructors. A non-default constructor takes the arguments provided and creates a new-user defined object.

The next type of conversion to consider to converting a user-defined object into either a primitive or another user-defined object. This is accomplished using an operator () function, as illustrated in the next two examples.

Example 6-8 - Conversion of a user-defined type to a primitive type

```
1  // File: ex6-8.cpp
2  #include <iostream>
3  using namespace std;
4
5  class B {
6      int b;
7    public:
8      B(int i) : b(i) {}
9      operator int() const;
10 };
11
12 B::operator int() const {
13   cout << "* B:: operator int() called\n";
14   return b;
15 }
16
17 int main() {
18   B eight(8);
19   cout << eight << endl;
20   cout << eight + 5 << endl;
21   cout << 5 + eight << endl;
22   cout << (eight > 3) << endl;
23   return 0;
24 }
```

```
******  Output  ******
* B:: operator int() called
8
* B:: operator int() called
13
* B:: operator int() called
13
* B:: operator int() called
1
```

- What would happen if operator int() was not defined?

Example 6-9 - More Conversions of a user-defined type

```
1  // File: ex6-9.cpp - More Type Conversions
2
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7
8  class Day;    // forward declaration
9
10  class Number
11  {
12    int n;
13  public:
14    // Constructor
15    Number(int i = 0) : n(i) { cout << "Number(int) ctor called\n"; }
16
17    // Conversion operators
18    operator int() const;
19    operator Day() const;
20  };
21
22  Number::operator int() const
23  {
24    cout << "* Number::operator int() called\n";
25    return n;
26  }
27
28  const string Days[7] =
29     {"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday",
30      "Saturday"};
31
32
33  class Day
34  {
35    string dow;
36  public:
37    // Constructor
38    Day(int n = 0);
39
40    // Conversion operator
41    operator Number() const;    // convert Day to Number
42
43    // !operator prints dow
44    void operator!() { cout << "dow = " << dow << endl; }
45  };
46
47  Day::Day(int index)
48  : dow(Days[index % 7])
49  {
```

```
50    cout << "Day(int) ctor called\n";
51  }
52  Day::operator Number() const
53  {
54    cout << "** Day:: operator Number() called\n";
55    for (int i = 0; i < 7; i++) if (dow == Days[i]) return Number(i);
56    return Number(0);
57  }
58
59  Number::operator Day() const
60  {
61    cout << "* Number::operator Day() called\n";
62    return Day(n);
63  }
64
65
66  int main()
67  {
68    Number N1(65);
69    cout << "N1 = " << N1 << endl;
70
71    Day d1(1);
72    !d1;
73
74    //  Day d2(N1);       Why is this an ambiguity?
75
76    Number N2(d1);
77    cout << "N2 = " << N2 << endl;
78
79    !Day(Number(d1)+2);
80
81    return 0;
82  }
```

```
******  Output  ******

Number(int) ctor called
* Number::operator int() called
N1 = 65
Day(int) ctor called
dow = Monday
** Day:: operator Number() called
Number(int) ctor called
* Number::operator int() called
N2 = 1
** Day:: operator Number() called
Number(int) ctor called
* Number::operator int() called
Day(int) ctor called
dow = Wednesday
```

Program Analysis

Line 8: Why is there a Day forward declaration?
Line 15: Describe the Number constructor.
Line 25: What are the options for a return from Number::operator int()?
Line 28: Why is Days const? How are the Days array elements created?
Line 48: Why “index % 7” ?
Line 56: Why is return Number(0) there? (Is it a good idea?)
Line 74: Why is this line commented out?

Inheritance and Polymorphism

Inheritance

Inheritance is a relationship between two classes such that one class takes on (inherits) the properties and behaviors (types, data members and member functions) of another class. The *derived* class inherits from a *base* class. This process facilitates code reuse and is a formal method of expressing natural relationships between types.

A *derived* class may be the *base* for another class. Several classes may inherit from one class. A *derived* class may inherit from several classes. Just like people! This is called *multiple inheritance.*

The following example illustrates some of the basic inheritance concepts.

Example 7-1 - First inheritance example

```
1  // File: ex7-1.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  class Base
7  {
8  protected:
9      int b;
10  public:
11      Base(int n);
12      void print() const
13      {
14          cout << "Base data is " << b << endl;
15      }
16  };
17
18  Base::Base(int n) : b(n)
19  {
20      cout << "created Base object: " << this << endl;
21  }
22
23  class Derived : public Base
24  {
25  private:
26      int d;
27  public:
28      Derived(int x,int y);
29      void print() const;
```

```
30      void printBase() const
31      {
32          cout << this << "'s Base is " << b << endl;
33      }
34  };
35
36  Derived::Derived(int x, int y) : Base(x), d(y)
37  {
38      cout << "created Derived object: " << this << endl;
39  }
40
41  void Derived::print(void) const
42  {
43      cout << "Derived data is " << d << endl;
44      Base::print();
45  }
46
47  int main()
48  {
49      Base b1(5);
50
51      // print base object
52      b1.print();
53      cout << endl;
54      Derived d1(3,4);
55
56      // print derived object
57      d1.print();
58      cout << endl;
59
60      d1.printBase();
61
62      // call base class print() from derived class object
63      d1.Base::print();
64      cout << endl;
65
66      cout << "how big is an int? " << sizeof(int) << endl;
67      cout << "how big is a Base? " << sizeof b1 << endl;
68      cout << "how big is a Derived? " << sizeof d1 << endl;
69  }
```

******** Output ********

```
created Base object: 0x69fefc
Base data is 5

created Base object: 0x69fef4
created Derived object: 0x69fef4
Derived data is 4
Base data is 3

0x69fef4's Base is 3
```

```
Base data is 3

how big is an int? 4
how big is a Base? 4
how big is a Derived? 8
```

Inheritance Notes

- The base class data members are usually protected. Thus, they may be accessible in the derived class.
- Public inheritance is the most common type of inheritance. In public inheritance, the protected base members are accessible and are also protected in the derived class. Also, the public base members remain public in the derived class. In any type of inheritance, private base members are not accessible in any "place" except in base class member functions. Access in derived classes to the base members by inheritance type is summarized in the following table:

Access to base class members in a derived class Base

Base Class Members	**Public Inheritance**	**Private Inheritance**	**Protected Inheritance**
Private	not accessible	not accessible	not accessible
Protected	protected	private	protected
Public	public	private	protected

- The derived class constructor automatically makes a call to the base class constructor. You can cause a certain base class constructor to be called by using constructor initialization list syntax. If you don't, then the default base class constructor is called (and it had better be there).

- The base class constructor executes before the derived class constructor, and the derived destructor will execute before the base destructor.

- The derived class will use the accessible member functions of the base class unless it has a function of the same *signature*.

- The following members are not inherited by the derived class:
  constructors
  destructors
  friend functions

- Static data members may be inherited and hence, are shared among the base and derived class objects, providing they have public or protected access. Further, static member functions may also be inherited.

- Derived classes are also called subclasses, base classes are also called superclasses.

Inheritance Examples

The following example illustrates a typical inheritance situation. Suppose you have a number class in which addition with a plus sign is defined. This class works well, but you would also like to be able to use it for subtraction. To do so, define your "own" class and inherit the number class. Add a subtraction function to your class.

Example 7-2 - Adding functionality to a class using inheritance

```
1  // File: ex7-2.cpp - Adding functionality to a class using
   inheritance
2
3  #include <iostream>
4  using namespace std;
5
6  class Number
7  {
8  protected:
9      int x;
10  public:
11      Number() {}
12      Number(int n) : x(n) { }
13      Number(const Number& n) : x(n.x) { }
14      int get_x() const
15      {
16          return x;
17      }
18      Number& operator=(const Number& z)
19      {
20          x = z.x;
21          return *this;
22      }
23      Number  operator+(const Number& y) const
24      {
25          return x + y.x;
26      }
27  };
28
29  ostream& operator<<(ostream& out, const Number& obj)
30  {
31      out << obj.get_x();
32      return out;
33  }
34
35  class MyNumber : public Number
36  {
37  public:
38      MyNumber() {}
39      MyNumber(int n) : Number(n) { }
40      MyNumber(const Number& n) : Number(n) {}
41      MyNumber(const MyNumber& m) : Number(m) {}
```

```
42      MyNumber  operator-(const MyNumber& y) const
43      {
44          return x - y.x;
45      }
46  };
47
48  int main(void)
49  {
50      Number n1(4), n2(5);
51      Number n3;
52      cout << "n1=" << n1 << endl;
53      cout << "n2=" << n2 << endl;
54      n3 = n1 + n2;
55      cout << "n3=" << n3 << endl;
56      cout << endl;
57
58      MyNumber mn1(7), mn2(4);
59      MyNumber mn3;
60      cout << "mn1=" << mn1 << endl;
61      cout << "mn2=" << mn2 << endl;
62
63      mn3 = mn1 + mn2;
64      cout << "mn3=" << mn3 << endl;
65
66      mn3 = mn1 - mn2;
67      cout << "mn3=" << mn3 << endl;
68
69      MyNumber mn4(n1);
70      cout << "mn4=" << mn4 << endl;
71
72      MyNumber mn5(mn1);
73      cout << "mn5=" << mn5 << endl;
74  }
```

```
******  Output  ******

n1=4
n2=5
n3=9

mn1=7
mn2=4
mn3=11
mn3=3
mn4=4
mn5=7
```

- What is the purpose of the default constructors in both classes?

- How can the MyNumber copy constructor pass a MyNumber& to the Number copy constructor?

- What is returned from the operator+ and operator- functions?

Example 7-3 - Inherit the deck class

```
1  // File: ex7-3.cpp - Inherit the deck class
2
3  #include <iostream>
4  #include <cstdlib>
5  using namespace std;
6
7  class Card
8  {
9  private:
10      int value;
11      int suit;
12  public:
13      Card(int n = 0);
14      Card(int val, int s);
15      int get_value() const
16      {
17          return value;
18      }
19      int get_suit() const
20      {
21          return suit;
22      }
23  };
24
25  Card::Card(int n) : value(n % 13), suit(n / 13)
26  { }
27
28  Card::Card(int val, int s) :value (val), suit(s)
29  { }
30
31  ostream& operator<<(ostream& out, const Card& crd)
32  {
33      const string valueStr[13] =
34      {
35          "two","three","four","five","six","seven",
36          "eight","nine","ten","jack","queen","king","ace"
37      };
38      const string suitStr[4] =
   {"clubs","diamonds","hearts","spades"};
39      out << valueStr[crd.get_value()] << " of " <<
   suitStr[crd.get_suit()];
40      return out;
41  }
42
43  class Deck
44  {
45  protected:
46      const int DeckSize;
47      Card* ptrCard;
48
49  public:
```

```
50      Deck(int = 0);
51      ~Deck();
52      Card* get_ptrCard() const
53      {
54          return ptrCard;
55      }
56      int getDeckSize() const
57      {
58          return DeckSize;
59      }
60      void shuffle();
61  };
62
63  Deck::Deck(int n) :     DeckSize(n), ptrCard(new Card[n])
64  { }
65
66  Deck::~Deck()
67  {
68      delete [] ptrCard;
69      ptrCard = nullptr;
70  }
71
72  ostream& operator<<(ostream& out, const Deck& deck)
73  {
74      for (int i = 0; i < deck.getDeckSize(); i++)
75          out << deck.get_ptrCard()[i] << endl;
76      return out;
77  }
78
79  void Deck::shuffle()
80  {
81      cout << "I am shuffling the Deck\n";
82      Card temp;
83      for (int i = 0; i < DeckSize; i++)
84      {
85          int k = rand() % DeckSize;
86          temp = ptrCard[i];
87          ptrCard[i] = ptrCard[k];
88          ptrCard[k] = temp;
89      }
90  }
91
92
93  class PokerDeck : public Deck
94  {
95  public:
96      PokerDeck();
97  };
98
99  PokerDeck::PokerDeck() : Deck(52)
100  {
101      for (int i = 0; i < DeckSize; i++) ptrCard[i] = Card(i);
```

```
102  }
103
104  class PinocleDeck : public Deck
105  {
106  public:
107      PinocleDeck();
108  };
109
110  PinocleDeck::PinocleDeck() : Deck(48)
111  {
112      for (int i = 0; i < DeckSize; i++) ptrCard[i] =
   Card(i%6+7,i/2%4);
113  }
114
115  int main()
116  {
117      PokerDeck pokerD;
118      cout << "This is a poker deck\n" << pokerD << endl;
119      PinocleDeck pinocleD;
120      cout << "This is a pinocle deck\n"<< pinocleD << endl;
121      pokerD.shuffle();
122      pinocleD.shuffle();
123  }
```

```
******  Output  ******

two of clubs                          <= the poker deck starts here
three of clubs
four of clubs
five of clubs
six of clubs
seven of clubs
eight of clubs
 ..
 ..
 ..
queen of spades
king of spades
ace of spades

nine of clubs                         <= the pinocle deck starts here
ten of clubs
jack of diamonds
queen of diamonds
king of hearts
ace of hearts
nine of spades
ten of spades
jack of clubs
queen of clubs
king of diamonds
ace of diamonds
nine of hearts
ten of hearts
 ..
 ..
 ..
ace of diamonds
nine of hearts
ten of hearts
jack of spades
queen of spades
king of clubs
ace of clubs
nine of diamonds
ten of diamonds
jack of hearts
queen of hearts
king of spades
ace of spades
…
```

Example 7-4 - Account classes

```
1  // File: ex7-4.cpp - Derive Savings and Checking from Account
2
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  class Account
8  {
9  protected:
10      unsigned long long accountNum;
11      double balance;
12      double intRate;         // annual interest rate
13  public:
14      Account(unsigned long long num = 0,double bal = 0,double = 0);
15      void deposit(double amount);
16      void withdraw(double amount);
17      void month_end();
18      friend ostream& operator<<(ostream& out, const Account&
    account);
19  };
20
21  ostream& operator<<(ostream& out, const Account& account)
22  {
23      out << fixed << setprecision(2);
24      out << "Account: " << account.accountNum << "  balance = $" <<
    account.balance << endl;
25      return out;
26  }
27
28  Account::Account(unsigned long long acc_no,double init_bal,double
    i_rate)
29      : accountNum(acc_no), balance(init_bal), intRate(i_rate)
30  {
31      cout << "* New Account\t";
32      cout << *this << endl;
33  }
34
35  void Account::deposit(double amount)
36  {
37      cout << "Account: " << accountNum << "  deposit = $" << amount
    << endl;
38      balance += amount;
39  }
40
41  void Account::withdraw(double amount)
42  {
43      cout << "Account: " << accountNum << "  withdraw = $" << amount
    << endl;
44      balance -= amount;
45  }
```

```
46
47  void Account::month_end()
48  {
49      cout << "Account month-end processing: " << accountNum << endl;
50      balance *= (1.+intRate/12.);
51      cout << *this << endl;
52  }
53
54  class SavingsAccount : public Account
55  {
56  public:
57      SavingsAccount(long acc_no, double init_bal = 50., double
    i_rate = .02)
58          : Account(acc_no,init_bal,i_rate) { }
59  };
60
61
62  class CheckingAccount : public Account
63  {
64  private:
65      double min_balance;
66      double service_charge;
67  public:
68      CheckingAccount(unsigned long long, double, double =
    300.,double = 3.,double = .01);
69      void process_check(double amt)
70      {
71          withdraw(amt);
72      }
73      void month_end(void);
74  };
75
76  CheckingAccount::CheckingAccount(unsigned long long acc_no, double
    init_bal,
77                                   double min_bal, double
    service_chg, double i_rate)
78      : Account(acc_no,init_bal,i_rate),
79        min_balance(min_bal),
80        service_charge(service_chg)
81  { }
82
83  void CheckingAccount::month_end()
84  {
85      cout << "Checking Account month-end processing: " << accountNum
    << endl;
86      balance *= (1.+intRate/12.);
87      if (balance < min_balance) balance -= service_charge;
88      cout << *this << endl;
89  }
90
91  int main()
92  {
```

```
93      SavingsAccount Mysavings(1234560ULL,500.);
94      CheckingAccount Mychecking(1234561ULL,1000.);
95      Mysavings.deposit(100.);
96      cout << Mysavings << endl;
97      Mysavings.withdraw(200.);
98      cout << Mysavings << endl;
99      Mychecking.deposit(100.);
100      cout << Mysavings << endl;
101      Mychecking.process_check(200.);
102      cout << Mychecking << endl;
103      Mysavings.month_end();
104      Mychecking.month_end();
105  }
```

```
******  Output  ******

* New account  account: 1234560  balance = 500

* New account  account: 1234561  balance = 1000

account: 1234560  deposit = 100
account: 1234560  balance = 600

account: 1234560  withdraw = 200
account: 1234560  balance = 400

account: 1234561  deposit = 100
account: 1234561  balance = 1100

account: 1234561  withdraw = 200
account: 1234561  balance = 900

account month-end processing: 1234560
account: 1234560  balance = 401.666656

checking account month-end processing: 1234561
account: 1234561  balance = 903
```

Example 7-5 - Triangle classes
This example demonstates two levels of inheritance.

```
1  // File: ex7-5.cpp - Triangle classes
2
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6
7  class Triangle
8  {
9  protected:
10      double a,b,c;
11  public:
12      Triangle(double s1,double s2,double s3) : a(s1), b(s2), c(s3)
   {}
13      double area()const;
14      double perimeter() const
15      {
16          return a + b + c;
17      }
18      friend ostream& operator<<(ostream&, const Triangle&);
19  };
20
21  double Triangle::area() const
22  {
23      double s = perimeter()/2.0;    //    s = semiperimater
24      return sqrt(s*(s-a)*(s-b)*(s-c));
25  }
26
27  ostream& operator<<(ostream& out, const Triangle& triangle)
28  {
29      out << &triangle << ": sides "
30          << triangle.a << ' ' << triangle.b << ' ' << triangle.c;
31      return out;
32  }
33
34  class Isosceles : public Triangle
35  {
36  public:
37      Isosceles(double base, double leg) : Triangle(base,leg,leg) {}
38  };
39
40  class Equilateral : public Isosceles
41  {
42  public:
43      Equilateral(double side) : Isosceles(side,side) {}
44  };
45
46
47  int main()
48  {
49      Triangle t1(3,4,5);
```

```
50      cout << t1 << endl;
51      cout << "perimeter=" << t1.perimeter() << "  area=" <<
   t1.area() << endl;
52
53      Isosceles t2(2,4);
54      cout << t2 << endl;
55      cout << "perimeter=" << t2.perimeter() << "  area=" <<
   t2.area() << endl;
56
57      Equilateral t3(5);
58      cout << t3 << endl;
59      cout << "perimeter=" << t3.perimeter() << "  area=" <<
   t3.area() << endl;
60
61  }
```

```
******  Output  ******

0x6afee8: sides 3 4 5
perimeter=12  area=6
0x6afed0: sides 2 4 4
perimeter=10  area=3.87298
0x6afeb8: sides 5 5 5
perimeter=15  area=10.8253
```

Private Inheritance

Private inheritance may be used to represent a "has-a" relationship between two classes. (Fortunately) this type of inheritance is not all that common. Private inheritance is more commonly replaced by containment, or a container relationship. Instead of a "has-a" relationship between classes, private inheritance is more commonly used to express an "in terms of" relationship. Here are some notes regarding private inheritance:

- Private inheritance is the default inheritance type, even though public inheritance is by far the more common type of inheritance. This is what you get if you leave off the "public" after the colon in the class definition.

- Private inheritance is used to indicate that one class "contains" another class, but the containment is limited to exactly one instance of the base class.

- The (privately) derived class inherits the base class public and protected members, but does not "pass them on". That is, the derived class must provide it's own public interface to any base class members desired.

- Private inheritance is used when you want to make use of the base class, but you wish to hide the base class public interface or you wish to provide your own public interface.

Example 7-6 – Private inheritance

The following example demonstrates private inheritance. The objective is to create a name class that is defined in terms of the "standard" string class. To keep the class simple, the name class has a simple user interface, thus hiding the complexity of the string class.

```
1  // Example 7-6 - private inheritance
2
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  class name : private string
8  {
9  public:
```

```
10    name(const char *);
11    void print() const;
12    string first_last() const;
13    string initials() const;
14    void change_last(const string& new_last);
15  };
16
17  name::name(const char* n) : string(n) {}
18
19  void name::print() const {
20    cout << c_str() << ".\n";
21  }
22
23  string name::first_last() const {
24    size_t comma_pos = find(',');
25    size_t second_space = find_last_of(' ');
26    return substr(comma_pos+2,second_space-comma_pos-2) +
27           ' ' + substr(0,comma_pos);
28  }
29
30  string name::initials() const {
31    string inits;
32    inits = data()[find(',')+2];
33    return inits + data()[length()-1] + *data();
34  }
35
36  void name::change_last(const string& new_last) {
37    replace(0,find(','),new_last);
38  }
39
40  int main() {
41    name joe("Bentley, Joseph E");
42    joe.print();
43    cout << joe.first_last() << endl;
44    cout << joe.initials() << endl;
45    joe.change_last("Smith");
46    joe.print();
47    return 0;
48  }
```

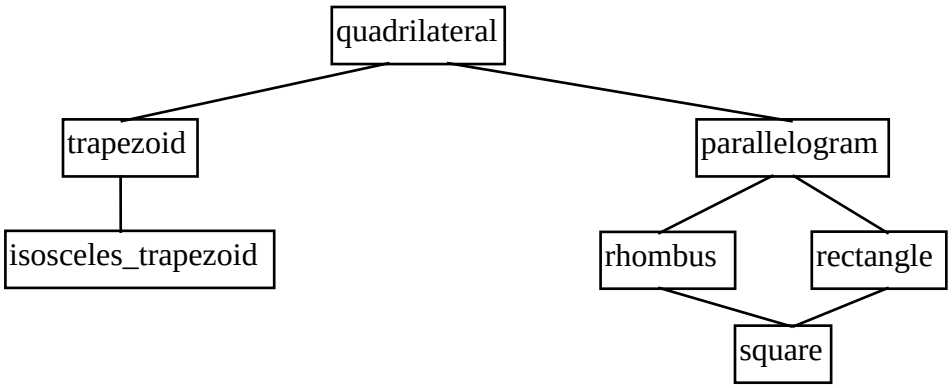
```
******  Output  ******

Bentley, Joseph E.
Joseph Bentley
JEB
Smith, Joseph E.
```

Multiple Inheritance

Example 7-7 - First Multiple Inheritance Example

```
File: ex7-7.cpp - multiple inheritance

#include <iostream>
using namespace std;

class one {
  protected:
    int a,b;
  public:
     one(int z,int y) { a = z; b = y; }
     void show(void) const { cout << a << ' ' << b << endl; }
};

class two {
  protected:
    int c,d;
  public:
    two(int z,int y) { c = z; d = y; }
    void show(void) const { cout << c << ' ' << d << endl; }
};

class three : public one, public two
{
  private:
    int e;
  public:
    three(int,int,int,int,int);
    void show(void) const
    { cout <<a<< ' ' <<b<< ' ' <<c<< ' ' <<d<< ' ' <<e<< endl;}
};

three::three(int a1, int a2, int a3, int a4, int a5)
: one(a1,a2),two(a3,a4)
{
  e = a5;
}

int main(void)
{
  one abc(5,7);
  abc.show();                                   // prints 5 7
  two def(8,9);
  def.show();                                   // prints 8 9
  three ghi(2,4,6,8,10);
  ghi.show();                                   // prints 2 4 6 8 10
  return 0;
}
```

The next example illustrates a more complicated inheritance situation. It models the relationship between types of quadrilaterals. This relationship is shown in the following figure:



Note that the parallelogram class will be derived from the quadrilateral class, both the rhombus and rectangle classes will be derived from the parallelogram class. And the square is derived from both the rhombus and the rectangle classes. It's the square class that makes this multiple inheritance.

Example 7-8 - Quadrilaterals

```
1  // File: ex7-8.cpp
2
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6
7  class quadrilateral
8  {
9    protected:
10      double a,b,c,d;
11    public:
12      quadrilateral(double s1,double s2,double s3,double s4)
13      : a(s1), b(s2), c(s3), d(s4) {}
14      quadrilateral() {}
15      void show() const
16      {
17         cout << "quadrilateral: " << this << "  sides "
18              << a << ' ' << b << ' ' << c << ' ' << d << endl;
19      }
20 };
```

```
21  class trapezoid : public quadrilateral
22  {
23    public:
24      trapezoid(double base1,double base2,double leg1,double leg2)
25      : quadrilateral(base1,leg1,base2,leg2) {}
26  };
27
28  class isosceles_trapezoid : public trapezoid
29  {
30    public:
31      isosceles_trapezoid(double base1,double base2,double leg)
32      : trapezoid(base1,leg,base2,leg) {}
33  };
34
35  class parallelogram : public quadrilateral
36  {
37    protected:
38      int angle;
39    public:
40      parallelogram(double s1,double s2, int ang)
41      : quadrilateral(s1,s2,s1,s2) { angle = ang; }
42      parallelogram() { }
43      void show_angles(void) const
44      {
45        cout << "angles = " << angle << ' ' << (180-angle) << endl;
46      }
47  };
48
49  class rectangle : virtual public parallelogram
50  {
51    public:
52      rectangle(double base, double height)
53      : parallelogram(base,height,90) {}
54      rectangle() {}
55  };
56
57  class rhombus: virtual public parallelogram
58  {
59    public:
60      rhombus(double side,int ang) : parallelogram(side,side,ang){}
61      rhombus() {}
62  };
63
64  class square : public rhombus,public rectangle
65  {
66    public:
67      square(double side) : parallelogram(side,side,90) {}
68  };
```

```
69  int main(void)
70  {
71    quadrilateral q1(1,2,3,4);
72    q1.show();
73
74    trapezoid q2(22,13,8,15);
75    q2.show();
76
77    isosceles_trapezoid q3(18,8,13);
78    q3.show();
79
80    parallelogram q4(4,3,45);
81    q4.show();
82    q4.show_angles();
83
84    rectangle q5(4,3);
85    q5.show();
86    q5.show_angles();
87
88    rhombus q6(5,45);
89    q6.show();
90    q6.show_angles();
91    cout << endl;
92
93    square q7(5);
94    q7.show();
95    q7.show_angles();
96
97    return 0;
98  }
```

```
******  Output  ******

quadrilateral: 0x3dc9ffd6  sides 1 2 3 4
quadrilateral: 0x3dc9ffb6  sides 22 8 13 15
quadrilateral: 0x3dc9ff96  sides 18 8 13 13
quadrilateral: 0x3dc9ff74  sides 4 3 4 3
angles = 45 135
quadrilateral: 0x3dc9ff52  sides 4 3 4 3
angles = 90 90
quadrilateral: 0x3dc9ff2e  sides 5 5 5 5
angles = 45 135

quadrilateral: 0x3dc9ff0a  sides 5 5 5 5
angles = 90 90
```

Note: The rectangle and rhombus classes both inherit the parallelogram class. Their inheritance is designated virtual, so that if a class is derived from both of the them, the parallelogram data will not be repeated in the class.

Polymorphism

Polymorphism is implemented when you have (a) derived class(es) containing a member function with the same signature as a base class. A function invoked through a pointer or a reference to the base class, will execute the correct implementation regardless of whether the pointer is pointing at a base class object or a derived class object. Functions that behave in this way are called virtual functions. The determination of which function to call is not known at compile-time, so the correct function is selected during execution. This process is called late binding, or dynamic binding. The usual call of a function through an object, is known to the compiler, hence, early binding or static binding.

Non-virtual vs. Virtual Functions

Example 7-9 - Non virtual Functions

This example and the next one deomonstrate the difference between a virtual and a non-virtual function.

```
1  // File: ex7-9.cpp - Inheritance with a non-virtual function
2
3  #include <iostream>
4  using namespace std;
5
6  class B
7  {
8  public:
9      B()
10     {
11         cout << "B ctor called for " << this << endl;
12     }
13     void funk1()
14     {
15         cout << "B::funk1() called for " << this << endl;
16     }
17     void funk2()
18     {
19         cout << "B::funk2() called for " << this << endl;
20     }
21 };
22
23 class D : public B
24 {
25 public:
26     D()
27     {
28         cout << "D ctor called for " << this << endl;
29     }
30     // Override funk1()
```

```
31      void funk1()
32      {
33          cout << "D::funk1() called for " << this << endl;
34      }
35  };
36
37  int main()
38  {
39      B b;
40      D d;
41      cout << endl;
42
43      b.funk1();
44      d.funk1();
45      cout << endl;
46
47      b.funk2();
48      d.funk2();
49      cout << endl;
50
51      B* pB;
52      pB = &b;
53      pB->funk1();
54      cout << endl;
55
56      pB = &d;
57      pB->funk1();
58      cout << endl;
59
60      cout << "size of b = " << sizeof b << endl;
61      cout << "size of d = " << sizeof d << endl;
62  }
```

```
******  Output  ******

B ctor called for 0x69fefb
B ctor called for 0x69fefa
D ctor called for 0x69fefa

B::funk1() called for 0x69fefb
D::funk1() called for 0x69fefa

B::funk2() called for 0x69fefb
B::funk2() called for 0x69fefa

B::funk1() called for 0x69fefb

B::funk1() called for 0x69fefa

size of b = 1
size of d = 1
```

- Why does a B and a D object have a size of 1?

Example 7-10 - Virtual Functions

This example is the same as the last one, except that funk1() is declared a virtual function. Hence, this program implements polymorphism.

```
1  // File: ex7-10.cpp - Inheritance with a virtual function
2
3  #include <iostream>
4  using namespace std;
5
6  class B
7  {
8  public:
9     B() { cout << "B ctor called for " << this << endl;}
10    void funk1() { cout << "B::funk1() called for " << this << endl;
   }
11    virtual void funk2() { cout << "B::funk2() called for " << this
   << endl; }
12  };
13
14  class D : public B
15  {
16  public:
17    D() { cout << "D ctor called for " << this << endl;}
18    void funk1() { cout << "D::funk1() called for " << this << endl;
   }
19    virtual void funk2() { cout << "D::funk2() called for " << this
   << endl; }
20  };
21
22  int main()
23  {
24    B b;
25    D d;
26    cout << endl;
27
28    b.funk1();
29    d.funk1();
30    cout << endl;
31
32    b.funk2();
33    d.funk2();
34    cout << endl;
35
36    B* pB;
37    pB = &b;
38    pB->funk1();
39    pB->funk2();
40    cout << endl;
41
42    pB = &d;
43    pB->funk1();
```

```
44    pB->funk2();
45
46    cout << endl;
47
48    cout << "size of b = " << sizeof b << endl;
49    cout << "size of d = " << sizeof d << endl;
50  }
```

```
******  Output  ******

B ctor called for 0x69fef8
B ctor called for 0x69fef4
D ctor called for 0x69fef4

B::funk1() called for 0x69fef8
D::funk1() called for 0x69fef4

B::funk2() called for 0x69fef8
D::funk2() called for 0x69fef4

B::funk1() called for 0x69fef8
B::funk2() called for 0x69fef8

B::funk1() called for 0x69fef4
D::funk2() called for 0x69fef4

size of b = 4
size of d = 4
```

Example 7-11 - Virtual Functions

This example illustrates that
1) a virtual function does not have to be overridden in the derived class and
2) also that you may not execute a derived class function that is not defined in the base class through a base class pointer even if the pointer is pointing at a derived class object.

```
1  // File: ex7-11.cpp
2
3  #include <iostream>
4  using namespace std;
5
6  class B
7  {
8  protected:
9      int b;
10  public:
11      B()
12      {
13          cout << "B ctor called for " << this << endl;
14          b = 0;
15      }
16      virtual void virt()
17      {
18          cout << "B::virt() called for " << this << endl;
19      }
20  };
21
22  class D : public B
23  {
24  protected:
25      int d;
26  public:
27      D()
28      {
29          cout << "D ctor called for " << this << endl;
30          d = 0;
31      }
32      void non_virt2()
33      {
34          cout << "D::non_virt2() called for " << this << endl;
35      }
36  };
37
38  int main()
39  {
40      B b;                  // declare a base object
41      D d;                  // declare a derived object
42
43      b.virt();             // invoke virt() through a base object
44      d.virt();             // invoke virt() through a derived object
45
46      B* pb;                // pb is a pointer to a base class object
```

```
47      pb = &b;        // pb points to b
48
49      pb->virt();          // invoke virt() through a base pointer
50      //   to a base object
51
52      pb = &d;        // pb points to d
53      pb->virt();          // invoke virt() through a base pointer
54      //   to a derived object
55
56      cout << "size of b = " << sizeof b << endl;
57      cout << "size of d = " << sizeof d << endl;
58      d.non_virt2(); // invoke non_virt2() through derived object
59  //  pb->non_virt2(); Error:  non_virt2() is not a member of B
60  }
61
```

```
******  Output  ******

B ctor called for 0x69fef4
B ctor called for 0x69fee8
D ctor called for 0x69fee8
B::virt() called for 0x69fef4
B::virt() called for 0x69fee8
B::virt() called for 0x69fef4
B::virt() called for 0x69fee8
size of b = 8
size of d = 12
D::non_virt2() called for 0x69fee8
```

Example 7-12 - Virtual Functions

This example shows that
1) "virtualness" is passed down to derived classes even if the immediate "parent" class does not name a function as virtual and
2) polymorphism may be implemented through references instead of pointers to base objects.

```
1  // File: ex7-12.cpp
2
3  // This example shows that "virtualness" is passed down to derived
   classes
4  // even if the immediate "parent" class does not name a function as
   virtual.
5  // It also illustrates polymorphism implemented through references
   instead
6  // of pointers to base objects.
7
8  #include <iostream>
9  #include <string>
10  using namespace std;
11
```

```
12  class person
13  {
14  public:
15      virtual string who_am_i() const
16      {
17          return "person";
18      }
19      string non_virtual_who_am_i() const
20      {
21          return "non_virtual person";
22      }
23  };
24
25  class child : public person
26  {
27  public:
28      string who_am_i() const
29      {
30          return "child";
31      }
32      string non_virtual_who_am_i() const
33      {
34          return "non_virtual child";
35      }
36  };
37  class grand_child : public child
38  {
39  public:
40      string who_am_i() const
41      {
42          return "grand_child";
43      }
44      string non_virtual_who_am_i() const
45      {
46          return "non_virtual grand_child";
47      }
48  };
49
50  void identify_yourself(const person& p)
51  {
52      cout << "I am a " << (p.who_am_i()) << endl;
53      cout << "I am a " << (p.non_virtual_who_am_i()) << endl;
54  }
55
56  int main()
57  {
58      person P;
59      child C;
60      grand_child G;
61      person* pp;
62      pp = &P;
63      cout << (pp->who_am_i()) << endl;
```

```
64      cout << (pp->non_virtual_who_am_i()) << endl;
65      pp = &C;
66      cout << (pp->who_am_i()) << endl;
67      cout << (pp->non_virtual_who_am_i()) << endl;
68      pp = &G;
69      cout << (pp->who_am_i()) << endl;
70      cout << (pp->non_virtual_who_am_i()) << endl;
71      cout << "sizeof(person) = " << sizeof(person) << endl;
72      cout << "sizeof(child) = " << sizeof(child) << endl;
73      cout << "sizeof(grand_child) = " << sizeof(grand_child) <<
   endl;
74      identify_yourself(P);
75      identify_yourself(C);
76      identify_yourself(G);
77  }
```

```
******  Output  ******

person
non_virtual person
child
non_virtual person
grand_child
non_virtual person
sizeof(person) = 4
sizeof(child) = 4
sizeof(grand_child) = 4
I am a person
I am a non_virtual person
I am a child
I am a non_virtual person
I am a grand_child
I am a non_virtual person
```

Why write a Virtual destructor?

Example 7-13
This example illustrates why you might want to write a virtual destructor.

```
1  // File: ex7-13.cpp - Why a Virtual destructor?
2
3  #include <iostream>
4  using namespace std;
5
6  class X
7  {
8  public:
9      X()
10      {
11          cout << "X constructor called\n";
12      }
13      ~X()
14      {
15          cout << "X destructor called\n";
16      }
17  };
18
19
20  class A : public X
21  {
22  public:
23      A()
24      {
25          cout << "A constructor called\n";
26      }
27      ~A()
28      {
29          cout << "A destructor called\n";
30      }
31  };
32
33
34  int main()
35  {
36      X* ptrX;
37
38      ptrX = new X;
39      delete ptrX;
40
41      cout << endl;
42
43      ptrX = new A;
44      delete ptrX;
45  }
```

```
******  Output  ******

X constructor called
X destructor called

X constructor called
A constructor called
X destructor called
```

- What's the problem?

Example 7-14
This example shows how to write a virtual destructor. Compare the output with the last example.

```
1  // File: ex7-14.cpp - Why a Virtual destructor? Here's why!
2
3  #include <iostream>
4  using namespace std;
5
6
7  class X
8  {
9  public:
10      X()
11      {
12          cout << "X constructor called\n";
13      }
14      virtual ~X()
15      {
16          cout << "X destructor called\n";
17      }
18  };
19
20
21  class A : public X
22  {
23  public:
24      A()
25      {
26          cout << "A constructor called\n";
27      }
28      ~A()
29      {
30          cout << "A destructor called\n";
31      }
32  };
33
34
35  int main()
36  {
37      X* ptrX;
38
39      ptrX = new X;
40      delete ptrX;
41
42      cout << endl;
43
44      ptrX = new A;
45      delete ptrX;
46  }
```

```
******  Output  ******
```

```
X constructor called
X destructor called

X constructor called
A constructor called
A destructor called
X destructor called
```

Note: it is not necessary to repeat the **virtual** for the destructor in the derived class.

Non-Virtual, Virtual, and Pure Virtual Functions

The following notes differentiate these three types of class member functions:

Non-Virtual

- This is the default type of class member function. The keyword ***virtual*** does not appear in the function prototype.
- Non-virtual functions, as a rule, are not usually overridden in the derived class.

Virtual

- The keyword ***virtual*** appears at the beginning of the function prototype in the base class. It doesn't have to be used in derived class function prototypes, but it's not a bad idea to use it.
- Virtual functions, as a rule, are usually overridden in the derived class.
- Virtual functions make polymorphism possible.

Pure Virtual

- The keyword ***virtual*** appears at the beginning and **= 0** at the end of the function prototype in the base class. The **= 0** is not repeated in derived classes unless that class is intended to serve as a base class for other derived classes.
- Pure virtual functions must be overridden in the derived class, unless, that class is also a base class for other classes.
- Pure virtual functions are not defined in the class in which they are declared as pure vitual.
- The presence of a pure virtual function in a class makes it an abstract class. Abstract classes may not be instantiated.

Abstract Classes and Pure Virtual Functions

The following example is the traditional shape class example, illustrating the abstract base class, shape, with pure vitual functions.

Example 7-15 - Abstract classes and pure virtual functions

```
1  // File: ex7-15.cpp - Abstract classes
2
3  #include <iostream>
4  #include <cmath>
5  #include <cstdlib>
6  using namespace std;
7
8  const double pi = 3.141592654;
9
10  class Shape
11  {
12  protected:
13      double x;
14      double y;
15  public:
16      Shape(double = 0,double = 0);
17      double get_x() const
18      {
19          return x;
20      }
21      double get_y() const
22      {
23          return y;
24      }
25      virtual double area() const = 0;    // pure virtual function
26      virtual double girth() const = 0;   // pure virtual function
27  };
28
29  Shape::Shape(double c_x, double c_y) : x(c_x),y(c_y) {}
30
31  ostream& operator<<(ostream& out, const Shape& object)
32  {
33      cout << '(' << object.get_x() << ',' << object.get_y() << ')';
34      return out;
35  }
36
37  class Square : public Shape
38  {
39  private:
40      double side;
41  public:
42      Square(double c_x,double c_y, double s);
43      double get_side()
44      {
```

```
45          return side;
46      }
47      double area() const;
48      double girth() const;
49  };
50
51  Square::Square(double c_x, double c_y, double s) : Shape(c_x,c_y),
    side(s)
52  { }
53
54  double Square::area() const
55  {
56      return side * side;
57  }
58
59  double Square::girth() const
60  {
61      return 4.*side;
62  }
63
64  class Triangle : public Shape
65  {
66  private:
67      double a,b,c; // length of 3 sides
68  public:
69      Triangle(double c_x,double c_y, double s1, double s2, double
    s3);
70      void print_sides();
71      double area() const;
72      double girth() const;
73  };
74
75  Triangle::Triangle(double c_x, double c_y, double s1, double s2,
    double s3)
76      : Shape(c_x,c_y), a(s1), b(s2), c(s3)
77  { }
78
79  void Triangle::print_sides()
80  {
81      cout << a << ' ' << b << ' ' << c;
82  }
83
84  double Triangle::area() const
85  {
86      double s = (a + b + c) / 2.;   // semiperimeter
87      return sqrt(s*(s-a)*(s-b)*(s-c));
88  }
89
90  double Triangle::girth() const
91  {
92      return a+b+c;
93  }
```

```
94
95  class Circle : public Shape
96  {
97  private:
98      double radius;
99  public:
100     Circle(double c_x, double c_y, double r);
101     double get_radius()
102     {
103         return radius;
104     }
105     double area() const;
106     double girth() const;
107 };
108
109 Circle::Circle(double c_x, double c_y, double r) : Shape(c_x,c_y),
   radius(r)
110 { }
111
112 double Circle::area() const
113 {
114     return pi*radius*radius;
115 }
116
117 double Circle::girth() const
118 {
119     return 2.*pi*radius;
120 }
121
122 int main()
123 {
124     //  Shape sh(6,7); can't create instance of abstract class
125     Circle c(3,4,5);
126     cout << "Circle c - center: ";
127     cout << c << endl;
128     cout << "  radius = " << c.get_radius();
129     cout << "  area = " << c.area();
130     cout << "  circumference = " << c.girth() << endl;
131
132     Square s(5.,2.,1.);
133     cout << "Square s - center: ";
134     cout << s << endl;
135     cout << "  side = " << s.get_side();
136     cout << "  area = " << s.area();
137     cout << "  perimeter = " << s.girth() << endl;
138
139     Triangle t(0,0,3,4,5);
140     cout << "Triangle t - center: ";
141     cout << t << endl;
142     cout << "  sides = ";
143     t.print_sides();
144     cout << "  area = " << t.area();
```

```
145      cout << "  perimeter = " << t.girth() << endl;
146
147      cout << "sizeof(double)=" << sizeof(double) << endl;
148      cout << "sizeof(Shape)=" << sizeof(Shape) << endl;
149      cout << "sizeof(Square)=" << sizeof(Square) << endl;
150      cout << "sizeof(Triangle)=" << sizeof(Triangle) << endl;
151      cout << "sizeof(Circle)=" << sizeof(Circle) << endl;
152  }
```

```
******  Output  ******

circle c - center: (3,4)  radius = 5  area = 78.5398  circumference = 31.4159
square s - center: (5,2)  side = 1  area = 1  perimeter = 4
triangle t - center: (0,0)  sides = 3 4 5  area = 6  perimeter = 12
sizeof(double)=8
sizeof(shape)=24
sizeof(square)=32
sizeof(triangle)=48
sizeof(circle)=32
```

Example 7-16 - Life
The following example is a practical application which make use of polymorphism and an abstract class.

```
1  // File: ex7-16.cpp - Life and polymorphism
2
3  #include <iostream>
4  #include <cstdlib>
5  using namespace std;
6
7  enum Bool { FALSE, TRUE};
8  enum LifeForm {VACANT, WEED, RABBIT, HAWK};
9
10  const int GridSize = 10;
11  const int Cycles = 10;
12  const int NumberLifeForms = 4;
13  const int HawkLifeExpectancy = 8;
14  const int HawkOvercrowdingLimit = 3;
15  const int RabbitLifeExpectancy = 3;
16
17  class Grid;
18
19  class LivingThing
20  {
21  protected:
22      int x,y;
23      void AssessNeighborhood(const Grid& G, int sm[]);
24  public:
25      LivingThing(int _x, int _y): x(_x), y(_y) {}
26      virtual ~LivingThing() {}
27      virtual LifeForm WhoAmI() const = 0;
28      virtual LivingThing* next(const Grid& G) = 0;
29  };
30
31  class Grid
32  {
33  private:
34      LivingThing* cell[GridSize][GridSize];
35  public:
36      Grid();
37      ~Grid()
38      {
39          if (cell[1][1]) release();
40      }
41      void update(Grid&);
42      void release();
43      void print();
44      LivingThing* get_cell(int row, int col) const;
45  };
46
47  /* This function counts the number of each LivingThing thing in
48    the neighborhood.  A neighborhood is a square and the 8
```

```
49    adjacent squares on each side of it */
50  void LivingThing::AssessNeighborhood(const Grid& G, int count[])
51  {
52      int i, j;
53      count[VACANT] = count[WEED] = count[RABBIT] = count[HAWK] = 0;
54      for (i = -1; i <= 1; ++i)
55          for (j = -1; j <= 1; ++j)
56              count[G.get_cell(x+i,y+j) -> WhoAmI()]++;
57  }
58
59  LivingThing* Grid::get_cell(int row, int col) const
60  {
61      return cell[row][col];
62  }
63
64  class Vacant : public LivingThing
65  {
66  public:
67      Vacant(int _x, int _y):LivingThing(_x,_y) {}
68      LifeForm WhoAmI() const
69      {
70          return (VACANT);
71      }
72      LivingThing* next(const Grid& G);
73  };
74
75  class Weed : public LivingThing
76  {
77  public:
78      Weed(int _x, int _y): LivingThing(_x,_y) {}
79      LifeForm WhoAmI() const
80      {
81          return (WEED);
82      }
83      LivingThing* next(const Grid& G);
84  };
85
86  class Rabbit : public LivingThing
87  {
88  protected:
89      int age;
90  public:
91      Rabbit(int x, int y, int a = 0) : LivingThing(x,y), age(a) {}
92      LifeForm WhoAmI() const
93      {
94          return (RABBIT);
95      }
96      LivingThing* next(const Grid& G);
97  };
98
99  class Hawk : public LivingThing
100  {
```

```
101  protected:
102      int age;
103  public:
104      Hawk(int x, int y, int a = 0): LivingThing(x,y), age(a) {}
105      LifeForm WhoAmI() const
106      {
107          return (HAWK);
108      }
109      LivingThing* next(const Grid& G);
110  };
111
112  // This function determines what will be in an Vacant square in
   the next cycle
113  LivingThing* Vacant::next(const Grid& G)
114  {
115      int count[NumberLifeForms];
116      AssessNeighborhood(G,count);
117
118  // If there is more than one Rabbit in the neighborhood, a new
   Rabbit
119  //     is born.
120      if (count[RABBIT] > 1)   return (new Rabbit(x,y));
121
122  // otherwise, if there is more than one Hawk, a Hawk will be born
123      else if (count[HAWK] > 1) return (new Hawk(x, y));
124
125  // otherwise, if there is Weed in the neighborhood, Weed will grow
126      else if (count[WEED])   return (new Weed(x, y));
127
128  // otherwise the square will remain Vacant
129      else return (new Vacant(x, y));
130  }
131
132  // if there is more Weeds than Rabbits, then new Weed will grow,
133  // otherwise Vacant
134  LivingThing* Weed::next(const Grid& G)
135  {
136      int count[NumberLifeForms];
137      AssessNeighborhood(G, count);
138      if (count[WEED] > count[RABBIT])   return (new Weed(x, y));
139      else return (new Vacant(x, y));
140  }
141
142  /* The Rabbit dies if:
143   there's more Hawks in the neighborhood than Rabbits
144   not enough to eat
145   or if it's too old
146   otherwise a new Rabbit is born */
147  LivingThing* Rabbit::next(const Grid& G)
148  {
149      int count[NumberLifeForms];
150      AssessNeighborhood(G, count);
```

```
151      if (count[HAWK] >= count[RABBIT] ) return (new Vacant(x, y));
152      else if (count[RABBIT] > count[WEED]) return (new Vacant(x,
   y));
153      else if (age > RabbitLifeExpectancy) return (new Vacant(x,
   y));
154      else return (new Rabbit(x,y, age + 1));
155  }
156
157  // Hawk die of overcrowding, starvation, or old age
158  LivingThing* Hawk::next(const Grid& G)
159  {
160      int count[NumberLifeForms];
161      AssessNeighborhood(G, count);
162      if (count[HAWK] > HawkOvercrowdingLimit) return (new Vacant(x,
   y));
163      else if (count[RABBIT] < 1) return (new Vacant(x,y));
164      else if (age > HawkLifeExpectancy) return (new Vacant (x, y));
165      else return (new Hawk(x, y, age + 1));
166  }
167
168  Grid::Grid()
169  {
170      LifeForm creature;
171      int i, j;
172      for (i = 0; i < GridSize; i++)
173          for (j = 0; j < GridSize; j++)
174          {
175              if (i == 0 || i == GridSize - 1 || j ==0 || j ==
   GridSize - 1)
176                  creature = VACANT;
177              else
178                  creature = LifeForm(rand() % NumberLifeForms);
179              switch (creature)
180              {
181              case HAWK:
182                  cell[i][j] = new Hawk(i,j);
183                  break;
184              case RABBIT:
185                  cell[i][j] = new Rabbit(i,j);
186                  break;
187              case WEED:
188                  cell[i][j] = new Weed(i,j);
189                  break;
190              case VACANT:
191                  cell[i][j] = new Vacant(i,j);
192              }
193          }
194  }
195
196  void Grid::release()
197  {
198      int i, j;
```

```
199      for (i = 1; i < GridSize - 1; ++i)
200          for (j = 1; j < GridSize - 1; ++j) delete cell[i][j];
201      cell[1][1] = 0;
202  }
203
204  void Grid::update(Grid& old)
205  {
206      int i, j;
207      for (i = 1; i < GridSize - 1; ++i)
208          for (j = 1; j < GridSize - 1; ++j)
209              cell[i][j] = old.cell[i][j] -> next(old);
210  }
211
212  void Grid::print()
213  {
214      LifeForm creature;
215      int i, j;
216      for (i = 1; i < GridSize - 1; i++)
217      {
218          for (j = 1; j < GridSize - 1; j++)
219          {
220              creature = cell[i][j]->WhoAmI();
221              switch (creature)
222              {
223              case HAWK:
224                  cout << "H";
225                  break;
226              case RABBIT:
227                  cout << "R";
228                  break;
229              case WEED:
230                  cout << "W";
231                  break;
232              case VACANT:
233                  cout << "0";
234              }
235          }
236          cout << endl;
237      }
238      cout << endl;
239  }
240
241  int main()
242  {
243      Grid G1, G2;
244      G1.print();
245
246      for (int i = 1; i <= Cycles; i++)
247      {
248          cout << "Cycle " << i << endl;
249          if (i % 2)
250          {
```

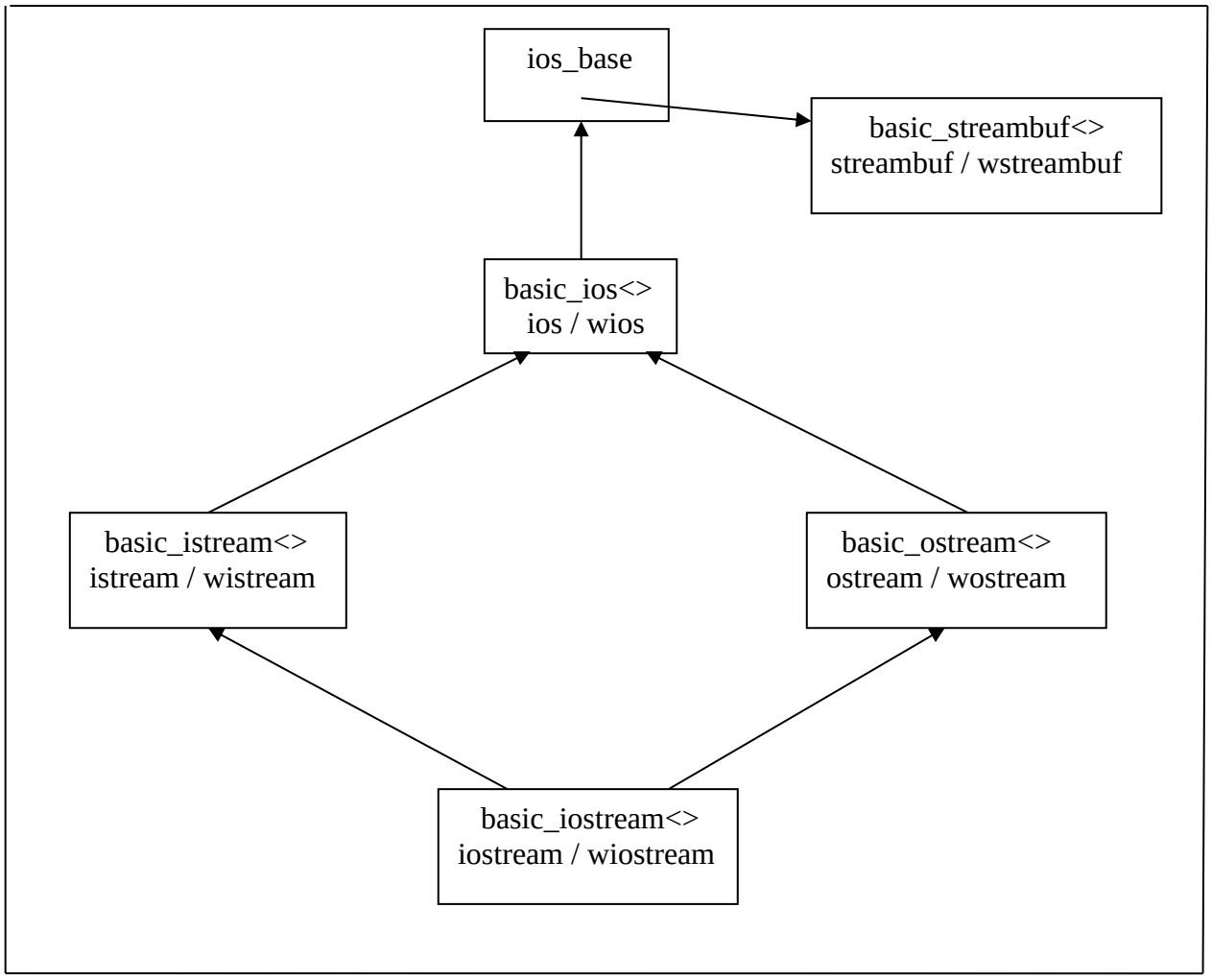
```
251                 G2.update(G1);
252                 G2.print();
253                 G1.release();
254             }
255             else
256             {
257                 G1.update(G2);
258                 G1.print();
259                 G2.release();
260             }
261         }
262     }
```

```
******  Output  ******

WHR0W0RR                                00WWWW0H
R0WWWHWH                                HHHWW00H
HRH0H0RW                                H00HHH0H
0RW0RHHR                                H000H0HH
HRW0HHHR                                000R0W0W
HHRWW0WH                                H00RRWWW
W0H0RWWH                                WHH0RWWW
HWWR0HRR                                WWHRRHWW

Cycle 1                                 Cycle 6
0H0WWW00                                HHWWWWH0
0R0WWH0H                                000WWHH0
H0HHHR00                                0HH000H0
R00H0000                                0H0H0H00
H00RH000                                HHR0RWHW
H00WWHWH                                0HR00WWW
WHHRRWWH                                W0HR0WWW
0WW0RH00                                WWH00HWW

Cycle 2                                 Cycle 7
0HWWWWH0                                00WWWW0H
H0HWWHH0                                HHHWW00H
HRH000H0                                H00HHH0H
0HH0RH00                                H0HHHHHH
HHW0HHHW                                000R0W0W
0HR00HW0                                H00RRWWW
W0HRRWW0                                WHH0WWWW
WWWR0HHW                                WWHHW0WW

Cycle 3                                 Cycle 8
H0WWWW0H                                HHWWWWH0
HHHWW00H                                000WWHH0
H00HHH0H                                0HH000H0
H00H00HH                                0HH00000
000R000W                                HHR0RWHW
H00RR0WW                                0HR0RWWW
WHH00WWW                                W00RWWWW
WW00RH0W                                WW00WWWW

Cycle 4                                 Cycle 9
0HWWWWH0                                00WWWW0H
000WWHH0                                HHHWW00H
0HH000H0                                H00HWH0H
0HHHHH00                                H00RWHHH
HHR0RWHW                                000RRW0W
0HR00WWW                                HH0RRWWW
W0HRRWWW                                WWR0WWWW
WWH00HWW                                WWWWWWWW

Cycle 5                                 Cycle 10
```

```
HHWWWWH0                    HHR00WHW
000WWWH0                    0HR0RWWW
0HHHW0H0                    WWRRWWWW
0HR00H00                    WWWWWWWW
```

C++ Input/Output & File I/O

Input / Output Classes



Class/Template Descriptions

ios_base	class that serves as a base for all stream classes. Contains stream state and format flags.
basic_ios<>	class template derived from ios_base<>. Virtual base for stream classes. Contains a pointer to the stream buffer, functions for stream state and error indications.
ios	basic_ios class for char type
wios	basic_ios class for wchar type
basic_istream<>	class template derived from basic_ios<>. Defines streams used for input.
istream	basic_istream class for char type
wistream	basic_istream class for wchar type
basic_ostream<>	class template derived from basic_ios<>. Defines streams used for output.
ostream	basic_ostream class for char type
wostream	basic_ostream class for wchar type
basic_iostream<>	class template derived from both basic_istream<> and basic_ostream<>. Defines streams used for both input and output.
ostream	basic_iostream class for char type
wostream	basic_iostream class for wchar type
basic_streambuf<>	class template used to define interface to all stream types. Performs the actual reading and writing to/from streams.
streambuf	basic_streambuf class used to handle char type data.
wstreambuf	basic_streambuf class used to handle wchar type data.

ios_base class

typedefs

typedef T1 fmtflags; T1, T2 are integer types (int, enum, …)
typedef T2 iostate;

constants

Format flag constants

These constants are used to assign a value to a fmtflags value. They represent formatting/parsing specifications for a stream.

boolalpha	reads or displays “true” or “false” for bool values instead of “1” or “0”.
dec	reads or displays integer input/output as a decimal number.
fixed	displays floating point numbers in decimal format.
hex	reads or displays integer input/output as a hexadecimal number.
internal	displays octal, hexadecimal, and scientific numbers in internal format.
left	left justifies output in a field.
oct	reads or displays integer input/output as an octal number.
right	right justifies output in a field.
scientific	displays floating point numbers in scientific (exponential) format.
showbase	displays a number base prefix for integer output (0x or 0X for hex and 0 for octal)
showpoint	displays a decimal point in floating point numeric output.
showpos	displays a + sign for positive numeric output.
skipws	skips leading whitespace before >> operations.
unitbuf	flushes output stream after each insertion.
uppercase	displays hexadecimal output and the e of scientific format in uppercase.

adjustfield	group (bitwise or) of three flags : left, right, and internal.
basefield	group (bitwise or) of three flags : hex, oct, and dec.
floatfield	group (bitwise or) of two flags : fixed and scientific.

Stream state constants

These constants are used to assign a value to an iostate value. They represent the state of a stream.

badbit	is set if a stream is corrupted.
eofbit	is set if the EOF has been read.
failbit	is set if an I/O operation fails.
goodbit	is set if a stream is OK, no other bits set.

Some ios_base member functions

fmtflags flags() const;	returns the stream's fmtflags settings
fmtflags flags(fmtflags value);	sets the streams fmtflags. Note: clears any other flags already set.
fmtflags setf(fmtflags value);	sets the streams fmtflags. Note: does **not** clear other flags already set.
fmtflags setf(fmtflags val, fmtflags mask);	first clears the mask settings, then sets the stream's fmtflags. Note: does **not** clear other flags already set.
void unsetf(fmtflags value);	clears the fmtflags values(s).
streamsize precision() const;	returns the stream's precision setting for floating point values. Note: precision setting is the number of decimal places if fixed or scientific is set. If neither is set, the precision represents the number of significant digits. In either case, the value is automatically rounded to the precision setting.
streamsize precision(streamsize size);	sets the stream's precision. See note above.
streamsize width() const;	returns the stream's field width that applies to the next output value.
streamsize width(streamsize size);	sets the stream's *minimum* field width that applies **only** to the next output value. The width applies to both numeric and char data.

Some basic_ios member functions

The **ios** class is a typedef for the instantiation of the basic_ios template of type char.

char_type fill() const;	returns the fill character assigned to a stream. This character is used to pad a field whose width is more than the number of characters needed to display a value.
char_type fill(char_type ch);	sets the fill characted assigned to a stream.
bool operator!() const;	returns fail().
bool bad() const;	returns true if the stream's badbit is set. The bad() function indicates a corrupted stream.
void clear(iostream state=goodbit);	sets the iostate (by default to goodbit).
bool eof() const;	returns true if the eofbit is set (the stream's EOF has been read).
bool fail() const;	returns true if the stream's failbit **or** badbit is set. Use the fail() function to check to see if a file is successfully opened.
bool good() const;	returns true if the stream's goodbit is set.
iostate rdstate() const;	returns a streams iostate value.

Some basic_istream member functions

The **istream** class is a typedef for the instantiation of the basic_istream template of type char.

streamsize gcount() const;	returns the number of characters read by the last input operation
int get();	reads and returns the next character available in a stream. Returns EOF if no character is available.

istream specific functions

istream& get(char& ch);	reads the next char and assignes it to ch. Returns the "current" istream.
istream& get(char* buf, streamsize n);	reads n-1 bytes and stores them in buf. A newline(\n) will terminate the read. In that case, the newline **is not read and not stored**. The char data in buf is null-terminated.
istream& get(char* buf, streamsize n, char deliminator);	reads at most n-1 bytes and stores them in buf. If the deliminator is encountered, the read ends. The deliminator **is not read and not stored**. The char data in buf is null-terminated.
istream& getline(char* buf, streamsize n);	reads n-1 bytes and stores them in buf. A newline(\n) will terminate the read. In that case, the newline **is read, but not stored**. The char data in buf is null-terminated. **The newline (\n) is considered the delimiter for this function.** See warning in next paragraph.
istream& getline(char* buf,streamsize n,char deliminator);	reads at most n-1 bytes and stores them in buf. If the deliminator is encountered, the read ends. The deliminator **is read and not stored**. The char data in buf is null-terminated. **Warning**: **If the delimiter is not read, it is considered an error. The failbit is set, even though the stream data is stored in buf. This warning does not apply to the get() function.**

istream& ignore(streamsize n=1, int_type deliminator=EOF);

	extracts and discards n char or extracts until deliminator is read.
int_type peek();	returns the next available char. Does not increment the current get pointer.
istream& putback(char ch);	inserts the char ch into the input stream at the current get pointer position-1. Moves the current get pointer position back 1 byte.
istream& read(char* buf, streamsize n);	reads n bytes into buf. buf is not null terminated.
streamsize readsome(char* buf, streamsize n);	reads n bytes into buf. buf is not null terminated. Returns the number of characters read. The difference between read() and readsome() is that readsome() does not set the failbit if it cannot read n characters (if it encounters EOF before the read is complete).
istream& unget();	inserts the last char read into the input stream at its original position.

Some basic_ostream member functions

The **ostream** class is a typedef for the instantiation of the basic_ostream template of type char.

ostream& flush();	writes any data in the output buffer to the output stream.
ostream specific functions	
ostream& put(char ch);	writes ch to the output stream.
ostream& write(char* buf, streamsize n);	writes n bytes of char starting from the address buf into the output stream.

Example 8-1 – ios_base fmtflags

This example demonstrates the ios_base member, fmtflags.

The following header file, ***fmtflags.h***, will be used in the next few examples:

```
1  // File: fmtflags.h
2
3  #ifndef FMTFLAGS_H
4  #define FMTFLAGS_H
5
6  #include <iostream>
7  using namespace std;
8
9  void show_fmtflags(ios_base& stream) {
10    if (&stream == &cout) cout << "cout ";
11    if (&stream == &cerr) cout << "cerr ";
12    if (&stream == &clog) cout << "clog ";
13    if (&stream == &cin) cout << "cin ";
14    cout << "ios_base::fmtflags set: ";
15    if (stream.flags() & ios::boolalpha) cout << "boolalpha ";
16    if (stream.flags() & ios::dec) cout << "dec ";
17    if (stream.flags() & ios::fixed) cout << "fixed ";
18    if (stream.flags() & ios::hex) cout << "hex ";
19    if (stream.flags() & ios::internal) cout << "internal ";
20    if (stream.flags() & ios::left) cout << "left ";
21    if (stream.flags() & ios::oct) cout << "oct ";
22    if (stream.flags() & ios::right) cout << "right ";
23    if (stream.flags() & ios::scientific) cout << "scientific ";
24    if (stream.flags() & ios::showbase) cout << "showbase ";
25    if (stream.flags() & ios::showpoint) cout << "showpoint ";
26    if (stream.flags() & ios::showpos) cout << "showpos ";
27    if (stream.flags() & ios::skipws) cout << "skipws ";
28    if (stream.flags() & ios::unitbuf) cout << "unitbuf ";
29    if (stream.flags() & ios::uppercase) cout << "uppercase ";
30    cout << endl;
31 }
32
33 #endif
```

```
1  // File: ex8-1.cpp - ios::fmtflags
2
3  #include "fmtflags.h"
4
5  int main()
6  {
7     // save the default fmtflags settings for cout
8     ios::fmtflags cout_flags = cout.flags();
9
10    // print the default cout fmtflags settings value
11    cout << cout_flags <<endl;
12
13    // display the default fmtflags values for cout, cin, cerr, clog
14    show_fmtflags(cout);
15    show_fmtflags(cin);
16    show_fmtflags(cerr);
17    show_fmtflags(clog);
18
19    // turn on hex for cout
20    cout.flags(ios::hex);
21    // display the fmtflags settings for cout
22    show_fmtflags(cout);
23    // print some numbers
24    cout << 12 << ' ' << 123 << ' ' << 1234 << ' ' << 12345 << endl;
25
26    // turn on hex and showbase for cout
27    cout.flags(ios::hex|ios::showbase);
28    show_fmtflags(cout);
29    cout << 12 << ' ' << 123 << ' ' << 1234 << ' ' << 12345 << endl;
30
31    // turn on hex, showbase, and uppercase for cout
32    cout.flags(ios::hex|ios::showbase|ios::uppercase);
33    show_fmtflags(cout);
34    cout << 12 << ' ' << 123 << ' ' << 1234 << ' ' << 12345 << endl;
35
36    // turn on octal for cout
37    cout.flags(ios::oct);
38    show_fmtflags(cout);
39    cout << 12 << ' ' << 123 << ' ' << 1234 << ' ' << 12345 << endl;
40
41    // turn on octal and showbase for cout
42    cout.flags(ios::oct|ios::showbase);
43    show_fmtflags(cout);
44    cout << 12 << ' ' << 123 << ' ' << 1234 << ' ' << 12345 << endl;
45
46    // reset cout's flags back to the default settings
47    cout.flags(cout_flags);
48    show_fmtflags(cout);
49    cout << 12 << ' ' << 123 << ' ' << 1234 << ' ' << 12345 << endl;
50
51    // print out the value for each of the fmtflags constants
52    cout << "ios::boolalpha=" <<ios::boolalpha << endl;
```

```
53    cout << "ios::dec=" << ios::dec << endl;
54    cout << "ios::fixed=" << ios::fixed << endl;
55    cout << "ios::hex=" << ios::hex << endl;
56    cout << "ios::internal=" << ios::internal << endl;
57    cout << "ios::left=" << ios::left << endl;
58    cout << "ios::oct=" << ios::oct << endl;
59    cout << "ios::right=" << ios::right << endl;
60    cout << "ios::scientific=" << ios::scientific << endl;
61    cout << "ios::showbase=" << ios::showbase << endl;
62    cout << "ios::showpoint=" << ios::showpoint << endl;
63    cout << "ios::showpos=" << ios::showpos << endl;
64    cout << "ios::skipws=" << ios::skipws << endl;
65    cout << "ios::unitbuf=" << ios::unitbuf << endl;
66    cout << "ios::uppercase=" << ios::uppercase << endl;
67    return 0;
68  }
```

```
******  Output (GNU ver. 4.32) ******

4098                                                                    (11)
cout ios_base::fmtflags set: dec skipws                                 (14)
cin ios_base::fmtflags set: dec skipws                                  (15)
cerr ios_base::fmtflags set: unitbuf                                    (16)
clog ios_base::fmtflags set: dec skipws                                 (17)
cout ios_base::fmtflags set: hex                                        (22)
c 7b 4d2 3039                                                           (24)
cout ios_base::fmtflags set: hex showbase                               (28)
0xc 0x7b 0x4d2 0x3039                                                   (29)
cout ios_base::fmtflags set: hex showbase uppercase                     (33)
0XC 0X7B 0X4D2 0X3039                                                   (34)
cout ios_base::fmtflags set: oct                                        (38)
14 173 2322 30071                                                       (39)
cout ios_base::fmtflags set: oct showbase                               (43)
014 0173 02322 030071                                                   (44)
cout ios_base::fmtflags set: dec skipws                                 (48)
12 123 1234 12345                                                       (49)
ios::boolalpha=1                                                        (52)
ios::dec=2                                                              (53)
ios::fixed=4                                                            (54
ios::hex=8                                                              (55)
ios::internal=16                                                        (56)
ios::left=32                                                            (57)
ios::oct=64                                                             (58)
ios::right=128                                                          (59)
ios::scientific=256                                                     (60)
ios::showbase=512                                                       (61)
ios::showpoint=1024                                                     (62)
ios::showpos=2048                                                       (63)
ios::skipws=4096                                                        (64)
ios::unitbuf=8192                                                       (65)
ios::uppercase=16384                                                    (66)


******  Output (MS Visual C++ 2008) ******
```

```
513                                                                    (11)
cout ios_base::fmtflags set: dec skipws                                (14)
cin ios_base::fmtflags set: dec skipws                                 (15)
cerr ios_base::fmtflags set: dec skipws unitbuf                        (16)
clog ios_base::fmtflags set: dec skipws                                (17)
cout ios_base::fmtflags set: hex                                       (22)
c 7b 4d2 3039                                                          (24)
cout ios_base::fmtflags set: hex showbase                              (28)
0xc 0x7b 0x4d2 0x3039                                                  (29)
cout ios_base::fmtflags set: hex showbase uppercase                    (33)
0XC 0X7B 0X4D2 0X3039                                                  (34)
cout ios_base::fmtflags set: oct                                       (38)
14 173 2322 30071                                                      (39)
cout ios_base::fmtflags set: oct showbase                              (43)
014 0173 02322 030071                                                  (44)
cout ios_base::fmtflags set: dec skipws                                (48)
12 123 1234 12345                                                      (49)
ios::boolalpha=16384                                                   (52)
ios::dec=512                                                           (53)
ios::fixed=8192                                                        (54)
ios::hex=2048                                                          (55)
ios::internal=256                                                      (56)
ios::left=64                                                           (57)
ios::oct=1024                                                          (58)
ios::right=128                                                         (59)
ios::scientific=4096                                                   (60)
ios::showbase=8                                                        (61)
ios::showpoint=16                                                      (62)
ios::showpos=32                                                        (63)
ios::skipws=1                                                          (64)
ios::unitbuf=2                                                         (65)
ios::uppercase=4                                                       (66)
```

Example 8-2 – ios_base member functions

This examples demonstrates some of the ios_base member functions.

```
1  // File: ex8-2.cpp - ios_base member functions
2
3  #include "fmtflags.h"
4
5  int main()
6  {
7     // Display cout's default width, fill, and precision settings
8     cout << "cout.width()=" << cout.width() <<endl;
9     cout << "cout.fill()=" << cout.fill() << ' ' << (int) cout.fill()
   << endl;
10    cout << "cout.precision()=" << cout.precision() <<endl;
11
12    // Display cin's default width, fill, and precision settings
13    cout << "cin.width()=" << cin.width() <<endl;
14    cout << "cin.fill()=" << cin.fill() << ' ' << (int) cin.fill() <<
   endl;
15    cout << "cin.precision()=" << cin.precision() <<endl;
16
17    // Display cerr's default width, fill, and precision settings
18    cout << "cerr.width()=" << cerr.width() <<endl;
19    cout << "cerr.fill()=" << cerr.fill() << ' ' << (int) cerr.fill()
   << endl;
20    cout << "cerr.precision()=" << cerr.precision() <<endl;
21
22    // Demonstrate the ios_base::width() function
23    cout << 1 << 2 << 3 << endl;
24    cout.width(5);
25    cout << 1 << 2 << 3 << endl;
26
27    // Demonstrate the width() function for char data
28    cout.width(5);
29    cout << 'a' << 'b' << 'c' << endl;
30    cout.width(5);
31    cout << "a" << "b" << "c" << endl;
32
33    // What happens when a value's length exceeds the width setting
34    cout.width(3);
35    cout << 123456789 << '|' <<"hey\n";
36
37    // width and left justification
38    cout.setf(ios::left,ios::adjustfield);
39    cout.width(5);
40    cout << 1 << 2 << 3 << endl;
41
42    // Demonstrate fill()
43    cout.fill('$');
44    cout.width(10);
45    cout << 1 << '|' << 1 << endl;
```

```
46
47    // Set precision to 4
48    cout.precision(4);
49    cout.width(10);
50    cout << 123.45678 << '|' << 1234567.8 << '|' << 1.2345678 << '|'
   << 12345678 << endl;
51
52    // precision set to 4 and fixed flag set
53    cout.setf(ios::fixed,ios::floatfield);
54    cout.width(10);
55    cout << 123.45678 << '|' << 1234567.8 << '|' << 1.2345678 << '|'
   << 12345678 << endl;
56
57    // Any difference between float and double?
58    float f = 314.f;
59    cout << f << '|' << 314. << endl;
60
61    // Turn off fixed setting
62    cout.unsetf(ios::fixed);
63    cout << f << '|' << 314. << endl;
64
65    // Turn on showpoint
66    cout.setf(ios::showpoint);
67    cout << f << '|' << 314. << endl;
68
69    // Turn on fixed
70    cout.setf(ios::fixed,ios::floatfield);
71    cout << 123.45678 << '|' << 1234567.8 << '|' << 1.2345678 << '|'
   << 12345678 << endl;
72
73    // Clear the flags for cout and turn on hex for cin
74    cout.flags(ios_base::fmtflags(0));
75    cin.setf(ios::hex|ios::basefield);
76    cout.width(35);
77    cout << "Enter a hexadecimal number => ";
78    int Hex;
79    cin >> Hex;
80
81    // Display the Hex value in decimal and hex
82    cout << "Hex=" << Hex << '|' << hex << Hex << endl;
83
84    // Check the format flags for cin and cout
85    show_fmtflags(cin);
86    show_fmtflags(cout);
87
88    // Turn on hex, the right way
89    cin.setf(ios::hex,ios::basefield);
90    cout << "Try again, dummy => ";
91    cin >> Hex;
92    cout << Hex << endl;
93
94    // Is there a problem with cin
```

```
95    cout << "cin.rdstate=" << cin.rdstate() << endl;
96
97    // What are the format flag settings for cin and cout
98    show_fmtflags(cin);
99    show_fmtflags(cout);
100
101   // Fix the problem with cin
102   cin.clear();
103   cout << "cin.rdstate=" << cin.rdstate() << endl;
104
105   // Clear the input buffer
106   cin.ignore(50,'\n');
107
108   // Try again for the Hex input
109   cout << "Ok, get it right this time => ";
110   cin >> Hex;
111   cout << Hex << endl;
112
113   return 0;
114  }
```

```
******  Output (MS Visual C++ 2008)  ******

cout.width()=0
cout.fill()=  32
cout.precision()=6
cin.width()=0
cin.fill()=  32
cin.precision()=6
cerr.width()=0
cerr.fill()=  32
cerr.precision()=6
123
    123
    abc
    abc
123456789|hey
1    23
1$$$$$$$$|1
123.5$$$$$|1.235e+006|1.235|12345678
123.4568$$|1234567.8000|1.2346|12345678
314.0000|314.0000
314|314
314.0|314.0
123.4568|1234567.8000|1.2346|12345678
$$$$$Enter a hexadecimal number => abc
Hex=-858993460|cccccccc
cin ios_base::fmtflags set: dec hex oct skipws
cout ios_base::fmtflags set: hex
Try again, dummy => cccccccc
cin.rdstate=2
cin ios_base::fmtflags set: hex skipws
```

```
cout ios_base::fmtflags set: hex
cin.rdstate=0
Ok, get it right this time => abc
abc

******  Output (GNU ver. 4.32b) ******

cout.width()=0
cout.fill()=  32
cout.precision()=6
cin.width()=0
cin.fill()=  32
cin.precision()=6
cerr.width()=0
cerr.fill()=  32
cerr.precision()=6
123
    123
    abc
    abc
123456789|hey
1    23
1$$$$$$$$$|1
123.5$$$$$|1.235e+06|1.235|12345678
123.4568$$|1234567.8000|1.2346|12345678
314.0000|314.0000
314|314
314.0|314.0
123.4568|1234567.8000|1.2346|12345678
$$$$$Enter a hexadecimal number => abc
Hex=4197696|400d40
cin ios_base::fmtflags set: dec hex oct skipws
cout ios_base::fmtflags set: hex
Try again, dummy => 400d40
cin.rdstate=4
cin ios_base::fmtflags set: hex skipws
cout ios_base::fmtflags set: hex
cin.rdstate=0
Ok, get it right this time => abc
abc
```

3 What happened to the output on Line 88?

The error was actually made in line 81, ***cin.setf(ios::hex|ios::basefield);.*** We want ***cin.setf(ios::hex,ios::basefield)*** here. By using the | we turn on the hex, dec, and oct fmtflags.

Example 8-3 – istream member functions

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6     char temp[80], ch;
7
8     cout << "Enter something => ";
9
10    // Get the first 6 bytes of the input buffer
11    cin.get(temp,7);
12    cout << "temp=" << temp << endl;
13
14    // Get the next 7 bytes of the input buffer
15    cin.get(temp,8);
16    cout << "temp=" << temp << endl;
17
18    // Get the next bytes from the input buffer
19    cin.get(ch);
20    cout << "ch=" << ch << endl;
21
22    // Get the rest of the input buffer until '\n' is read
23    while (cin.get(ch) && ch != '\n') {
24          cout << ch;
25    }
26    cout << endl;
27
28    cout << "Enter something else => ";
29    // Get some more data from the input buffer up to 'v'
30    cin.getline(temp,sizeof(temp),'v');
31    cout << "temp=" << temp << endl;
32
33    // Read 6 more bytes from the input buffer
34    cin.getline(temp,7);
35    cout << "temp=" << temp << endl;
36
37    // Read 1 more byte from the input buffer
38    cin.get(ch);
39    cout << "ch=" << ch << endl;           // What happened here?
40
41    // Look at the next byte in the input buffer
42    cout << "cin.peek()=" << cin.peek() << endl;
43
44    // What is the state of cin
45    cout << "cin.rdstate()=" << cin.rdstate() << endl;
46
47    // Clear the state of cin
48    cin.clear();
49
```

```
50    // Read 1 byte from the input buffer
51    cin.get(ch);
52    cout << "ch=" << ch << endl;
53
54    // Write the last byte read back into the input buffer
55    cin.unget();
56
57    cin.putback('X');
58    cin.getline(temp,sizeof(temp));
59    cout << "temp=" << temp << endl;
60
61    cout << "Enter some more => ";
62    cin.read(temp,6);
63    cout << "cin.gcount()=" << cin.gcount() << endl;
64    cout << "temp=" << temp << endl;
65    cin.ignore(5);
66    cin >> temp;
67    cout << "temp=" << temp << endl;
68    cin.ignore(80,'\n');
69
70    cout << "Enter one last time => ";
71
72    // Read the first word from the input buffer
73    cin >> temp;
74    cout << "temp=" << temp << endl;
75
76    // Read 7 more bytes into temp
77    cout << "cin.readsome(temp,7)=" << cin.readsome(temp,7) << endl;
78    cout << "temp=" << temp << endl;
79    cout << "cin.readsome(temp,10)=" <<cin.readsome(temp,10) << endl;
80    cout << "temp=" << temp << endl;
81    return 0;
82  }
```

```
******  Output (MS Visual C++ 2008) ******

Enter something => Have a nice day.
temp=Have a
temp= nice d
ch=a
y.
Enter something else => Have a very nice day.
temp=Ha
temp=e a ve
ch=

cin.peek()=-1
cin.rdstate()=2
ch=r
temp=Xry nice day.
Enter some more => Have a totally excellent day.
cin.gcount()=6
```

```
temp=Have ace day.
temp=lly
Enter one last time => That's enough now.
temp=That's
cin.readsome(temp,7)=7
temp= enoughe day.
cin.readsome(temp,10)=6
temp= now.
he day.

******  Output (GNU ver. 4.32b) ******

Enter something => Have a nice day.
temp=Have a
temp= nice d
ch=a
y.
Enter something else => Have a very nice day.
temp=Ha
temp=e a ve
ch=

cin.peek()=-1
cin.rdstate()=4
ch=r
temp=Xry nice day.
Enter some more => Have a totally excellent day.
cin.gcount()=6
temp=Have ace day.
temp=lly
Enter one last time => That's enough now.
temp=That's
cin.readsome(temp,7)=0
temp=That's
cin.readsome(temp,10)=0
temp=That's
```

What happened on line 41?

The problem really occurred on line 36, ***cin.getline(temp,7);***. The default delimiter for this version of getline() is '\n' (or more precisely, *widen('\n')*). The getline() function expects to read the delimiter. If the delimiter is not encounter in the "n-1" characters, the the failbit is set, That is, the read in not successful, even though the n-1 characters were read and stored in the buffer. Advice, be careful with this function and check the stream state if there is any possibility that the delimiter may not be present.

Example 8-4 - ostream member functions: put() and write()
The following examples illustrates the use of ostream put() and write() functions.

```
1  // File: ex8-4.cpp - ostream member functions: put() and write()
2
3  #include <iostream>
4  using namespace std;
5
6  struct stuff {
7       int a;
8       short b;
9       long c;
10      float d;
11      double e;
12      char f;
13      char* g;
14      char h[16];
15  };
16
17
18  int main(void)
19  {
20    char text[] = "The quick brown fox jumped over the lazy poodle";
21    for (int i = 0; i< 20; i++) {
22          cout.put(text[i]);
23    }
24    cout.put('\n');
25
26    cout.write(text,sizeof(text));
27    cout << endl;
28
29    stuff thing;
30    thing.a = 57;
31    thing.b = 98;
32    thing.c = 123456789;
33    thing.d = 1.2;
34    thing.e = 2.7;
35    thing.f = '*';
36    thing.g = text;
37    strcpy(thing.h,"bet the farm");
38
39    cout.write((char*)&thing,sizeof(thing));
40    cout << endl;
41
42    return 0;
43  }
```

```
******  Output  ******

The quick brown fox
The quick brown fox jumped over the lazy poodle
9   b   ╩═[ÜÖÖ?ÜÖÖÖÖÖ♣@*♫☺ ╪5♀ bet the farm ↕
```

Input/Output Manipulators

Manipulators are functions or function-like operators that change the state of the I/O stream. Those manipulators with arguments require the <iomanip> header file.

Manipulator	**I/O**	**Purpose**
boolalpha	I/O	sets boolalpha flag
dec	I/O	sets dec flag for i/o of integers, clears oct,hex
endl	O	inserts a newline **and flushes output stream**
ends	O	inserts a null
fixed	O	sets fixed flag
flush	O	flushes stream
hex	I/O	sets hex flag for i/o of integers, clears dec,oct
internal	O	sets internal flag
left	O	sets left flag
noboolalpha	I/O	clears boolalpha flag
noshowbase	O	clears showbase flag
noshowpoint	O	clears showpoint flag
noshowpos	O	clears showpos flag
noskipws	I	clears skipws flag
nounitbuf	O	clears unitbuf flag
nouppercase	O	clears uppercase flag
oct	I/O	sets oct flag for i/o of integers, clears dec,hex
resetiosflags(ios_base::fmtflags mask)	I/O	clears format flags specified by mask
right	O	sets right flag
scientific	O	sets scientific flag
setbase(int base)	I/O	sets integer base (8, 10, or 16)
setfill(char_type ch)	O	sets the fill character to ch
setiosflags(ios::base::fmtflags mask)	I/O	sets fomat flags to mask value
setprecision(int p)	O	sets precision of floating point numbers
setw(int w)	O	sets output field width to w
showbase	O	sets showbase flag
showpoint	O	sets showpoint flag
showpos	O	sets showpos flag
skipws	I	sets skipws flag
unitbuf	O	sets unitbuf flag
uppercase	O	sets uppercase flag
ws	I	extracts whitespace

Example 8-5 – Input/Output manipulators

The following examples illustrates the use of standard input/output manipulators.

```
1  // File: ex8-5.cpp - I/O Manipulators
2
3  #include <iomanip>
4  using namespace std;
5
6  #include "fmtflags.h"
7
8  int main() {
9     // save the initial cout flags settings
10
11    ios_base::fmtflags cout_fmtflags = cout.flags();
12
13    // Display the cout flags
14    show_fmtflags(cout);
15
16    // hex, oct, & dec manipulators
17    cout << hex << 123 << ' ' << oct << 123 << ' ' <<dec<<123<<endl;
18    show_fmtflags(cout);
19
20    // Turn on showpos, uppercase, showpoint, left, hex, (dec on)
21    cout << setiosflags(ios::showpos|ios::uppercase|ios::showpoint|
22                        ios::showbase|ios::left|ios::hex)
23         << 123 << endl;
24    show_fmtflags(cout);
25
26    // Clear the dec flag
27    cout << resetiosflags (ios::dec) << 123 << endl;
28    show_fmtflags(cout);
29
30    // Demonstrate the setfill and setw manipulators
31    cout << setfill('$') << setw(10) << 123 << endl;
32    cout << 123 << endl;
33
34    // Reset cout's flags back to the original settings
35    cout.flags(cout_fmtflags);
36
37    // Turn on hex
38    cout << hex << 123 << endl;
39    show_fmtflags(cout);
40
41    // Turn on octal
42    cout << oct << 123 << endl;
43    show_fmtflags(cout);
44
45    // Demonstrate setprecision
46    cout << setprecision(3)
47        << 1.2 << ' '<< 3.14 << ' ' << 35 << ' ` << 3.14159 << endl;
48
```

```
49    // Demonstrate setprecision with showpoint
50    cout << showpoint
51         << 1.2 << ' '<< 3.14 << ' ' << 35 << ' ` << 3.14159 << endl;
52
53    // Demonstrate showpos
54    cout << showpos
55         << 1.2 << ' '<< 3.14 << ' ' << 35 << ' ` << 3.14159 << endl;
56
57    show_fmtflags(cout);
58
59    // Back to decimal
60    cout << dec
61         << 1.2 << ' '<< 3.14 << ' ' << 35 << ' ` << 3.14159 << endl;
62    show_fmtflags(cout);
63
64    // What is truth?
65    cout << true << ' ' << boolalpha << true << endl;
66    show_fmtflags(cout);
67
68    return 0;
69  }
```

****** Output ******

```
cout ios_base::fmtflags set: dec skipws
7b 173 123
cout ios_base::fmtflags set: dec skipws
123
cout ios_base::fmtflags set: dec hex left showbase showpoint showpos skipws uppercase
0X7B
cout ios_base::fmtflags set: hex left showbase showpoint showpos skipws uppercase
0X7B$$$$$$
0X7B
7b
cout ios_base::fmtflags set: hex skipws
173
cout ios_base::fmtflags set: oct skipws
1.2 3.14 43 3.14
1.20 3.14 43 3.14
+1.20 +3.14 43 +3.14
cout ios_base::fmtflags set: oct showpoint showpos skipws
+1.20 +3.14 +35 +3.14
cout ios_base::fmtflags set: dec showpoint showpos skipws
1 true
cout ios_base::fmtflags set: boolalpha dec showpoint showpos skipws
```

Overloading the Insertion and Extraction Operators

Example 8-6 - Overloading the insertion operator

```
1  // File: ex8-6.cpp - Overloading the insertion operator
2
3  #include <iostream>
4  using namespace std;
5
6  const char* suit_name[4] = {"clubs","diamonds","hearts","spades"};
7  const char* value_name[13] = {"two","three","four","five","six",
8   "seven","eight","nine","ten","jack","queen","king","ace"};
9
10  class card {
11    private:
12      int suit;
13      int value;
14    public:
15      card(int=0);
16    friend ostream& operator<< (ostream&,const card&);
17  };
18
19  card::card(int x) {
20    suit = x / 13;
21    value = x % 13;
22  }
23
24
25  ostream& operator<<(ostream& s,const card& c) {
26    s << value_name[c.value] << " of " << suit_name[c.suit] << endl;
27    return s;
28  }
29
30  int main(void)
31  {
32    card c1(47);
33    card c2;
34    cout << c1;
35    cout << c2;
36    cout << card(3) << card(4);
37
38    return 0;
39  }
```

```
******  Output  ******

ten of spades
two of clubs
five of clubs
six of clubs
```

Example 8-7 - Overloading Insertion and Extraction

```
1  // File: ex8-7.cpp
2
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6
7  class fraction
8  {
9    private:
10     int numer;
11     int denom;
12   public:
13     fraction() {}
14     fraction(int n, int d) : numer(n), denom(d) {}
15     int get_numer(void) const { return numer; }
16     int get_denom(void) const { return denom; }
17     void reduce(void);
18   friend istream& operator>>(istream& s, fraction& f);
19 };
20
21 void fraction::reduce(void)
22 {
23   int min;
24 // find the minimum of the denom and numer
25   min = abs(denom) < abs(numer) ? abs(denom) : abs(numer);
26   for (int i = 2; i <= min; i++)
27   {
28     while ((abs(numer) % i == 0) && (abs(denom) % i == 0))
29     {
30       numer /= i;
31       denom /= i;
32     }
33   }
34   return;
35 }
36
37 // extraction's a friend
38 istream& operator>>(istream& s, fraction& f)
39 {
40   s >> f.numer >> f.denom;
41   return s;
42 }
43
44 // insertion's not a friend
45 ostream& operator<<(ostream& s, fraction f)
46 {
47   f.reduce();
48   s << f.get_numer() << '/' << f.get_denom();
49   return s;
50 }
```

```
51  int main(void)
52  {
53    fraction f(3,4);
54    cout << f << endl;
55    fraction g(2,4);
56    cout << g << endl;
57    cout << "Enter a fraction: numerator denominator => ";
58    fraction h;
59    cin >> h;
60    cout << h << endl;
61    return 0;
62  }
```

```
******  Sample Run  ******

3/4
1/2
Enter a fraction: numerator denominator => 6 8
3/4
```

- ✔ Why the the fraction passed by value in the overloaded insertion operator function?

Example 8-8 - Printing a deck

Here's one more example of overloading the insertion operator.

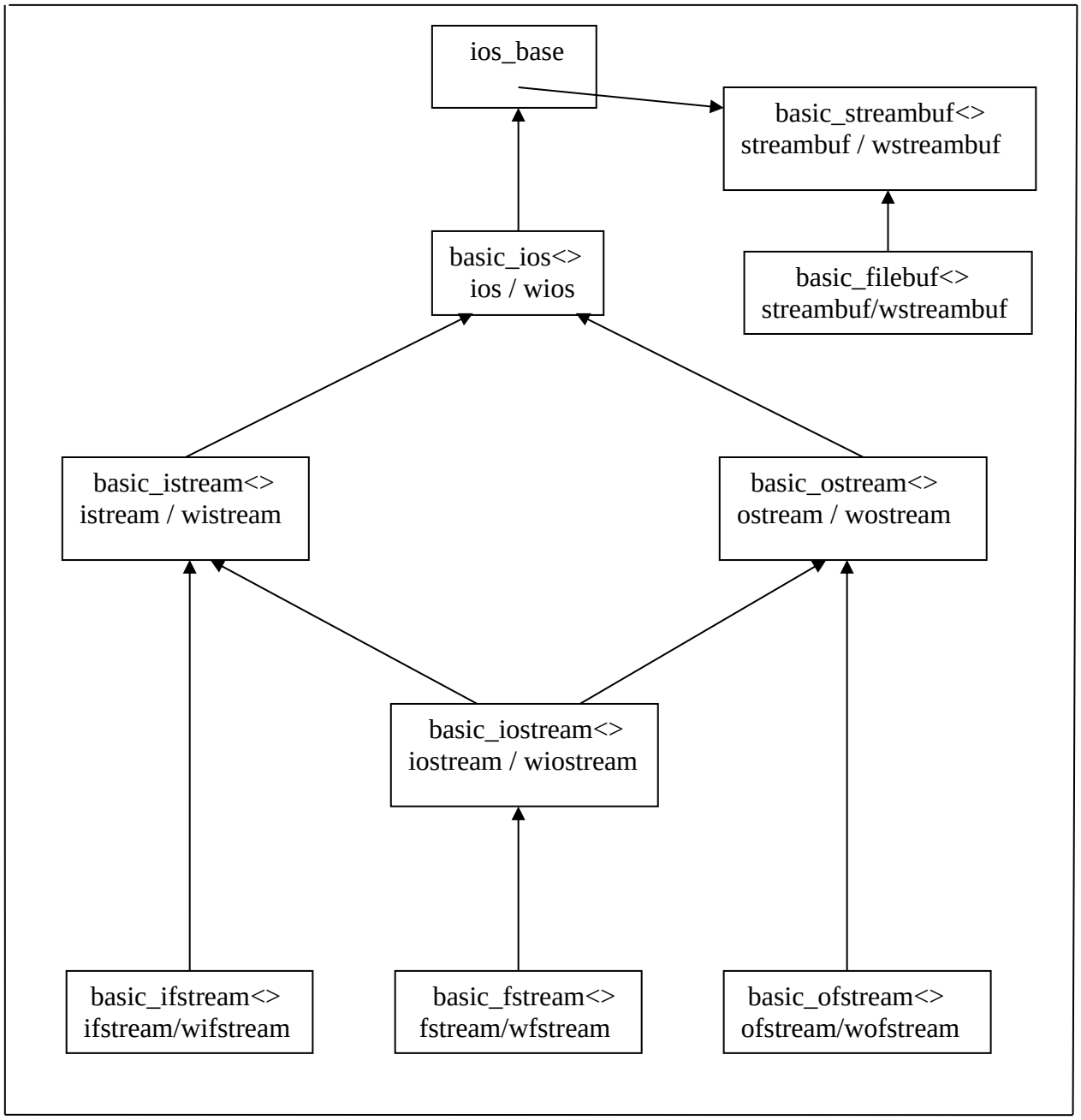
```
1  // File: ex8-8.cpp - overloading the insertion operator
2
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6
7  const char suit_char[5] = "CDHS";
8  const char value_char[14] = "23456789TJQKA";
9
10  class deck;                          // forward declare the deck
11
12  class card
13  {
14    private:
15      int suit;
16      int value;
17    public:
18      card(int);
19    friend ostream& operator<<(ostream&,const deck&);
20  };
21
22  card::card(int x)
23  {
24    suit = x / 13;
25    value = x % 13;
26  }
27
28
29  class deck
30  {
31    private:
32      card* d[52];
33    public:
34      deck();
35      ~deck();
36    friend ostream& operator<<(ostream&,const deck&);
37  };
38
39  deck::deck()
40  {
41    for (int i = 0; i < 52; i++) d[i] = new card(i);
42  }
43
44  deck::~deck()
45  {
46    for (int i = 0; i < 52; i++) delete d[i];
47  }
```

```
48  int main(void)
49  {
50    deck cards;
51    cout << cards;
52    return 0;
53  }
54
55  ostream& operator<<(ostream& s,const deck& dk)
56  {
57    for (int i = 0; i < 52; i++)
58    {
59      s << value_char[dk.d[i]->value] << suit_char[dk.d[i]->suit];
60      if (i % 13 == 12) s << endl;
61     else s << ' ';
62    }
63    return s;
64  }
```

```
******  Output  ******

2C 3C 4C 5C 6C 7C 8C 9C TC JC QC KC AC
2D 3D 4D 5D 6D 7D 8D 9D TD JD QD KD AD
2H 3H 4H 5H 6H 7H 8H 9H TH JH QH KH AH
2S 3S 4S 5S 6S 7S 8S 9S TS JS QS KS AS
```

- ✔ Why does operator<< need to be a friend of both the card and the deck classes?

C++ File I/O


ios_base
basic_streambuf<>
streambuf / wstreambuf
basic_ios<>
ios / wios
basic_filebuf<>
streambuf/wstreambuf
basic_istream<>
istream / wistream
basic_ostream<>
ostream / wostream
basic_iostream<>
iostream / wiostream
basic_ifstream<>
ifstream/wifstream
basic_fstream<>
fstream/wfstream
basic_ofstream<>
ofstream/wofstream

Class/Template Descriptions

basic_ifstream<>	class template derived from basic_istream<>. Defines file streams used for input.
ifstream	basic_ifstream class for char type
wifstream	basic_ifstream class for wchar type
basic_ofstream<>	class template derived from basic_ostream<>. Defines file streams used for output.
ofstream	basic_ofstream class for char type
wofstream	basic_ofstream class for wchar type
basic_fstream<>	class template derived from basic_iostream<>. Defines file streams used for both input and output.
fstream	basic_fstream class for char type
wfstream	basic_fstream class for wchar type

basic_ifstream<> members

basic_ifstream();

explicit basic_ifstream(const char* filename, ios_base::openmode mode = ios_base::in);[1]

void close();

bool is_open();

void open(const char* filename, ios_base::openmode mode = ios_base::in);

basic_ofstream<> members

basic_ofstream();

explicit basic_ofstream(const char* filename, ios_base::openmode mode = ios_base::out);

void close();

bool is_open();

void open(const char* filename, ios_base::openmode mode = ios_base::out);

basic_fstream<> members

basic_fstream();

explicit
basic_fstream(const char* filename, ios_base::openmode mode = ios_base::in | ios_base::out);

void close();

bool is_open();

void open(const char* filename, ios_base::openmode mode = ios_base::in | ios_base::out);

[1] explicit means that an object of this type can only be create by an explicit declaration of the object, and not by a conversion

Example 8-9 - Simple File I/O

```
1  // File: ex8-9.cpp
2
3  #include <iostream>
4  #include <fstream>
5  #include <cstdlib>
6  using namespace std;
7
8  int main(void)
9  {
10    ifstream f1("ex8-9.cpp");
11    ifstream f2("nofile");
12
13    ofstream f3("file.one");
14    fstream f4("iofile");
15
16    char buff[80];
17
18    if (!f1) {
19          cout << "Hey, I can't find the \"ex8-9.cpp\" file\n";
20          exit(1);
21    }
22
23    cout << boolalpha;  // turn on "true"/"false" for cout's bools
24
25    cout << "f2.rdstate()=" << f2.rdstate() << endl;
26    cout << "f2.fail()=" << f2.fail() << endl;
27    cout << "f2.bad()=" << f2.bad() << endl;
28    if (f2.fail())
29          cout << "Hey, I can't find \"nofile\", but who cares\n";
30    if (f3)
31          cout << "Hey, I've decided to create a \"file.one\"
   file\n";
32
33    f1 >> buff;
34    cout << buff << endl;
35    f3 << "Have a nice day\n" << endl;
36
37    // did f4 get opened?
38    cout << "f4.is_open()=" << f4.is_open()<<endl;
39    cout << "f4.good()=" << f4.good()<<endl;
40    cout << "f4.bad()=" << f4.bad()<<endl;
41    cout << "f4.fail()=" << f4.fail()<<endl;
42
43    // try to write using the f4 stream
44    f4 << buff << endl;
45
46    // recheck f4's status
47    cout << "f4.is_open()=" << f4.is_open()<<endl;
48    cout << "f4.good()=" << f4.good()<<endl;
49    cout << "f4.bad()=" << f4.bad()<<endl;
```

```
50    cout << "f4.fail()=" << f4.fail()<<endl;
51    return 0;
52  }
```

```
******  Output  - MS Visual C++ 2008 ******

f2.rdstate()=2
f2.fail()=true
f2.bad()=false
Hey, I can't find "nofile", but who cares
Hey, I've decided to create a "file.one" file
//
f4.is_open()=false
f4.good()=false
f4.bad()=false
f4.fail()=true
f4.is_open()=false
f4.good()=false
f4.bad()=true
f4.fail()=true


******  Output  - gnu version 4.32b ******

f2.rdstate()=4
f2.fail()=true
f2.bad()=false
Hey, I can't find "nofile", but who cares
Hey, I've decided to create a "file.one" file
//
f4.is_open()=false
f4.good()=false
f4.bad()=false
f4.fail()=true
f4.is_open()=false
f4.good()=false
f4.bad()=false
f4.fail()=true
```

More I/O Members and Types

ios_base class

typedefs

typedef T3 openmode;

constants

Open mode constants

These constants are used to assign a value to an openmode value. They represent the mode for opening a stream.

app	position to the end of the stream before each write operation.
ate	position to the end of the stream when the stream is opened.
binary	open the stream in binary mode (newlines are 1 byte).
in	open for input.
out	open for output
trunc	delete an existing file when opening.

Positioning constants

These constants are used to assign a value to a seekdir value. They used for relative positioning in a file stream with the seekg() and seekp() functions.

beg	position is relative to the beginning of a file stream.
cur	position is relative to the current position in a file stream.
end	position is relative to the end of a file stream.

More basic_istream members

istream& seekg(ios_base::pos_type pos);	positions to the location indicated by pos in a file stream. pos_type is the type returned by the tellg() function.
istream& seekg(ios_base::pos_type pos, ios_base::seekdir dir);	seeks to the position pos characters from dir in a file stream.
pos_type tellg();	returns the character position in the file stream. If the stream state is non-zero, then the function returns pos_type (-1).

More basic_ostream members

ostream& seekp(ios_base::pos_type pos);	positions to the location indicated by pos in a file stream. pos_type is the type returned by the tellp() function.
ostream& seekp(ios_base::pos_type pos, ios_base::seekdir dir);	seeks to the position pos characters from dir in a file stream.
pos_type tellp();	returns the character position in the file stream. If the stream state is non-zero, then the function returns pos_type (-1).

Example 8-10 – File I/O – positioning in a file

```
1  // File ex8-10.cpp - file I/O
2
3  #include <fstream>
4  #include <iostream>
5  #include <cstdlib>
6  #include <cstring>
7  using namespace std;
8
9  int main()
10  {
11    int ch, i;
12
13    // Open a file for output
14    ofstream fout("da_file");
15
16    // Check file open
17    if (!fout) {
18          cerr << "I can't open \"da_file\"\n";
19          exit(EXIT_FAILURE);       // EXIT_FAILURE signifies failure
20    }
21
22    // Write 4 lines into the file
23    fout << "Have a nice day\n";
24    fout << 7 << endl;
25    fout << 3.14159 << endl;
26    fout << hex << 123 << ' ' << oct << 123 << endl;
27
28    // Close the file
29    fout.close();
30
31    // Open the file for input
32    ifstream fin("da_file");
33
34    // Check input file open
35    if (!fin) {
36          cerr << "I can't open \"da_file\"\n";
37          exit(EXIT_FAILURE);
38    }
39
40    char buff[80];
41
42    // Read each line from the file
43    while (!fin.getline(buff,80).eof()) {
44          // Print the length of the line and its contents
45          cout << strlen(buff) << '\t' << buff << endl;
46    }
47
48    // Print the current position in the file
49    cout << fin.tellg() << endl;
50
```

```
51    // Move to byte 7 (the 8th byte) in the file
52    fin.seekg(7,ios_base::beg);
53
54    // Print the current position in the file
55    cout << fin.tellg() << endl;
56
57    // Set the boooalpha flag for cout
58    cout.setf(ios_base::boolalpha);
59
60    // Print the file's stream state and state bits
61    cout << "fin.rdstate()=" << fin.rdstate();
62    cout << " fin.bad()=" << fin.bad();
63    cout << " fin.fail()=" << fin.fail();
64    cout << " fin.eof()=" << fin.eof() <<endl;
65
66    // Clear the stream state
67    fin.clear();
68
69    // Print the file's stream state and state bits
70    cout << "fin.rdstate()=" << fin.rdstate();
71    cout << " fin.bad()=" << fin.bad();
72    cout << " fin.fail()=" << fin.fail();
73    cout << " fin.eof()=" << fin.eof() <<endl;
74
75    // Move to byte 7 (the 8th byte) in the file
76    fin.seekg(7,ios_base::beg);
77
78    // Print the current position in the file
79    cout << fin.tellg() << endl;
80
81    // Read the next "word" from the file
82    fin >> buff;
83
84    // Print the "word"
85    cout << buff << endl;
86
87    // Print the current position in the file
88    cout << fin.tellg() << endl;
89
90    // Read the next 10 characters
91    for (i = 0; i < 10; i++) {
92          ch = fin.get();
93          // Print counter, char (as int), char, and stream position
94          cout << i << '\t' << ch << '\t' << (char) ch << '\t'
95               << fin.tellg() << endl;
96    }
97
98    return 0;
99  }
```

```
******  Output (MS Visual C++ 2008)  ******

15      Have a nice day
1       7
7       3.14159
6       7b 173
-1
37
fin.rdstate()=0 fin.bad()=false fin.fail()=false fin.eof()=false[1]
fin.rdstate()=0 fin.bad()=false fin.fail()=false fin.eof()=false
7
nice
11
0       32              12
1       100     d       13
2       97      a       14
3       121     y       15
4       10
        17
5       55      7       18
6       10
        20
7       51      3       21
8       46      .       22
9       49      1       23


******  Output (GNU 4.32b)  ******

15    Have a nice day
1     7
7     3.14159
6     7b 173
-1
-1
fin.rdstate()=6 fin.bad()=false fin.fail()=true fin.eof()=true
fin.rdstate()=0 fin.bad()=false fin.fail()=false fin.eof()=false
7
nice
11
0     32          12
1     100   d     13
2     97    a     14
3     121   y     15
4     10
      16
5     55    7     17
6     10
      18
7     51    3     19
8     46    .     20
9     49    1     21
```

[1] The fin.eof()=false is unexpected, but is was found that the fin.seekg() on line 52 clears the stream state, even though the seekg() does not succeed.

- What's the difference between the two outputs? Why?

Example 8-11 - File I/O – positioning, modes, and stream state

```
1  // File ex8-11.cpp – positioning, modes, and stream state
2
3  #include <fstream>
4  #include <iostream>
5  #include <cstdlib>
6  using namespace std;
7
8  void print_file(istream&);
9
10  int main() {
11
12    // Declare and open an output file stream
13    ofstream fout("da_file");
14
15    // Check the file open
16    if (!fout) {
17          cerr << "I can't open \"da_file\"\n";
18          exit (EXIT_FAILURE);
19    }
20
21    // write 3 lines into a new file
22    fout << "Have a nice day.\n";
23    fout << "Have a great day.\n";
24    fout << "Have a totally excellent day.\n";
25
26    // close the file                        Why?
27    fout.close();
28
29    // re-open the file as an fstream object for input and output
30    fstream finout("da_file",ios_base::in|ios_base::out);
31
32    print_file(finout);
33
34    // clear the EOF bit
35    finout.clear();
36
37    // position to byte 7 in the file
38    finout.seekp(7,ios::beg);
39
40    // Are the get and put pointers the same?
41    cout<< "finout.tellg()=" << finout.tellg()<< endl;
42    cout<< "finout.tellp()=" << finout.tellp()<< endl;
43
44    // replace "nice" with "fine"
45    finout<< "fine";
46
47    // Are the get and put pointers still the same?
48    cout<< "finout.tellg()="<< finout.tellg()<< endl;
49    cout << "finout.tellp()=" << finout.tellp () << endl;
50
```

```
51    print_file(finout);
52
53    // close the file
54    finout.close();
55
56    // reopen the file in input/output/binary mode
57    finout.open("da_file",ios_base::in|ios_base::out|ios::binary);
58
59    // write hey into the file
60    finout<< "hey";
61
62    print_file(finout);
63
64    // try again to write hey into the file
65    finout.seekp(0,ios::beg);
66    finout<< "hey";
67    print_file(finout);
68
69    // try app mode
70    finout.clear();
71    finout.close();
72    finout.open("da_file",ios_base::in|ios_base::out|ios::app);
73    finout<< "hey";
74
75    print_file(finout);
76
77    return 0;
78  }
79
80
81  void print_file(istream& file)
82  {
83    cout<< "file.rdstate="<< file.rdstate() << endl;
84    char buffer[80];
85    file.seekg(0,ios::beg);
86    while (file.getline(buffer,sizeof(buffer))) {
87          cout<< buffer << endl;
88    }
89    cout << endl;
90  }
```

```
******  Output  (MS Visual C++ 2008)  ******

file.rdstate=0                              (32)
Have a nice day.
Have a great day.
Have a totally excellent day.

finout.tellg()=7                            (41)
finout.tellp()=7                            (42)
finout.tellg()=11                           (48)
finout.tellp()=11                           (49)
```

```
file.rdstate=0                                  (51)
Have a fine day.
Have a great day.
Have a totally excellent day.

file.rdstate=7                                  (62)
Have a fine day.
Have a great day.
Have a totally excellent day.

file.rdstate=7                                  (67)

file.rdstate=0                                  (75)
Have a fine day.
Have a great day.
Have a totally excellent day.
hey


******  Output  (GNU g++ 4.32)  ******

file.rdstate=0
Have a nice day.
Have a great day.
Have a totally excellent day.

finout.tellg()=7
finout.tellp()=7
finout.tellg()=11
finout.tellp()=11
file.rdstate=0                                  (51)
Have a fine day.
Have a great day.
Have a totally excellent day.

file.rdstate=0                                  (62)
heye a fine day.
Have a great day.
Have a totally excellent day.

file.rdstate=6                                  (67)

file.rdstate=0                                  (75)
heye a fine day.
Have a great day.
Have a totally excellent day.
hey
```

The following example illustrates binary file I/O. The example is a bit hokey, but it demonstrates input-output techniques that might be useful in a database application.

Example 8-12 – File I/O – read() and write()

```
1  // File: ex8-12.cpp - File I/O read() and write()
2
3  #include <fstream>
4  #include <iostream>
5  #include <iomanip>
6  #include <cstdlib>
7  #include <cstring>
8  #include <string>
9  using namespace std;
10
11  class Employee
12  {
13  public:
14    Employee() : age(0), salary(0.f) {}
15  private:
16    char empno[8];
17    char name[32];
18    unsigned short age;
19    float salary;
20    friend istream& operator>>(istream&, Employee&);
21    friend ostream& operator<<(ostream&, const Employee&);
22  };
23
24  istream& operator>>(istream& in, Employee& E)
25  {
26    in >> E.empno >> E.name >> E.age >> E.salary;
27    return in;
28  }
29
30  ostream& operator<<(ostream& out, const Employee& E)
31  {
32    out << setprecision(2) << fixed << showpoint << left;
33    out << setw(8) << E.empno
34          << setw(13) << E.name
35          << right << setw(3) << E.age
36          << setw(10) << E.salary;
37    return out;
38  }
```

```
39  class EmployeeFile
40  {
41  public:
42    EmployeeFile(string filename = "empfile");
43    void print();                          // Why isn't this const?
44    void open_for_read_write();
45    void close() { File.close(); }
46    bool operator!() const { return !File.rdstate(); }
47  private:
48    string Filename;
49    fstream File;
50    friend EmployeeFile& operator>>(EmployeeFile&, Employee&);
51    friend EmployeeFile& operator<<(EmployeeFile&, const Employee&);
52  };
53
54  // constructor opens file in output/binary mode
55  EmployeeFile::EmployeeFile(string filename)
56  : Filename(filename), File(filename.c_str(),ios::out|ios::binary)
57  {
58  }
59
60  void EmployeeFile::print()
61  {
62    Employee temp;
63
64    // position to the beginning of the file
65    File.seekg(0,ios_base::beg);
66
67    // read data from the file and print out each Employee record
68    while (!(*this>>temp)) {
69          cout << temp << endl;
70    }
71
72    // clear the File stream state (after reading EOF)
73    File.clear();
74  }
75
76  void EmployeeFile::open_for_read_write()
77  {
78    File.open(Filename.c_str(),ios::in | ios::out | ios::binary);
79
80    if (File.fail() ) {
81      cerr << "Unable to open Employee file: " << Filename << endl;
82      exit(-1);
83    }
84  }
85
86  EmployeeFile& operator>>(EmployeeFile& EF, Employee& E)
87  {
88    EF.File.read((char*) &E, sizeof E);
89    return EF;
90  }
```

```
91  EmployeeFile& operator<<(EmployeeFile& EF, const Employee& E)
92  {
93    EF.File.write((char*) &E, sizeof E);
94    return EF;
95  }
96
97  int main()
98  {
99    Employee temp;
100   EmployeeFile EmpFile("employee.dat");
101
102   for (int i = 0; i<4; ++i) {
103         cout << "Enter empno name age salary\n";
104         cin >> temp;
105         EmpFile << temp;
106   }
107
108   // close the file and reopen it in read-write mode
109   EmpFile.close();
110   EmpFile.open_for_read_write();
111
112   EmpFile.print();
113   return 0;
114  }
```

****** Sample Run ******

```
Enter empno name age salary
654321 Joe 35 80000
Enter empno name age salary
642731 Jim 55 85000
Enter empno name age salary
615787 Helen 60 90000
Enter empno name age salary
J00787 Susan 47 50000
654321  Joe           35  80000.00
642731  Jim           55  85000.00
615787  Helen         60  90000.00
J00787  Susan         47  50000.00
```

✔ What is the purpose of the char* cast in the file.read() above?

✔ Why is the EmployeeFile File stream opened by the constructor in write mode, then later closed and re-opened in read-write mode?

Example 8-13 - A DOS grep command

The following example is used to create a grep command for DOS. The UNIX grep command is used to search a file, or a list of files for the existence of a desired target string. This example was written for DOS, not UNIX. It demonstrates conditional compilation to allow for compiler differences.

After the program listing, sample compile commands are demonstrated and there is a note about executing the program under Windows XP.

```
1  // File: ex8-13.cpp - A grep command for DOS
2
3  #include <fstream>
4  #include <iostream>
5  #include <cstdlib>
6  #include <cstring>
7  using namespace std;
8
9  // The macro __GNUG__ is set for GNU C++
10  #ifdef __GNUG__
11  #include <unistd.h>          // for access()
12  #else
13  #include <io.h>         // for _access()
14  #endif
15
16  int main(int argc, char* argv[])
17  {
18    char       filename[64],
19                    buffer[1024],
20                    command[128],
21                    tempFilename[9];
22    int             lineno,
23                    hits = 0,
24                    files = 0,
25                    system_command_status;
26    ifstream   fin1,
27                    fin2;
28
29    // Check command-line syntax
30    if (argc != 3) {
31          cerr << "Syntax error\ngrep [target text] [target file(s)]
32               \n";
33          exit (-1);
34    }
35
36    // create a temporary file to hold the filenames to be searched
37    strcpy(tempFilename,"tempa");         // first possible filename
38
39    // The access() or _access() functions are used to status the
40    // existence of a file
```

```
41
42    // If you're not using a GNU C++ compiler, use _access()
43  #ifdef __GNUG__
44    while (access(tempFilename,0) == 0 && tempFilename[4] <= 'z')
45    // If you're not using a GNU C++ compiler, use _access()
46  #else
47    while (_access(tempFilename,0) == 0 && tempFilename[4] <= 'z')
48  #endif
49          tempFilename[4]++;
50
51    // If you've tried all filenames from "tempa" to "tempz",
52    // then give up
53    if (tempFilename[4] > 'z') {
54          cerr << "Unable to create a temp* file.  Please cleanup\n";
55          exit (-2);
56    }
57
58    // Create the command:  "dir /b [filename(s)] > [tempfilename]
59    strcpy(command,"dir /b ");
60    strcat(command,argv[2]);
61    strcat(command," > ");
62    strcat(command,tempFilename);
63
64    // system() allows you to issue operating system commands
65    system_command_status = system(command);
66
67    // check the status of the system command
68    if (system_command_status == -1) {
69          cerr << "Error with system command: " << command << endl;
70          exit(-3);
71    }
72
73    // open temporary file (containing names of files to be searched)
74    fin1.open(tempFilename);
75
76    // Make sure the tempFilename is not empty
77    if (fin1.peek() == EOF) {
78          cerr << "Error: target file(s) do(es) not exist, dummy\n";
79          exit (-4);
80    }
81
82    // for each file in tempFilename, search for the target string
83    while (fin1.getline(filename,sizeof(filename))) {
84
85          // open the next file to be searched
86          fin2.open(filename);
87
88          // increment file count
89          files++;
90
91          // initialize line counter
92          lineno = 0;
```

```
93
94          // clear errors in fin2 stream
95          fin2.clear();
96
97          // position at the beginning of the file to be searched
98          fin2.seekg(0L, ios:: beg);
99
100         // read each line from the file into buffer
101         while (fin2.getline(buffer,sizeof(buffer))) {
102
103               // increment line counter
104               lineno++;
105
106               // does buffer contains the target string?
107               if (strstr(buffer,argv[1])) {
108
109                     // If so, increment hit counter
110                     hits++;
111
112                     // display filename, line count, contents of line
113                     cout << filename << '[' << lineno << "] "
114                          << buffer << '\n';
115               }
116         }
117
118         // close the file to be searched
119         fin2.close();
120   }
121
122   // close the temporary file
123   fin1.close();
124
125   // Print summary information
126   cout << "Found " << hits << " occurrence(s) in " << files
127        << " file(s)\n";
128
129   // create the DOS command to erase the tempFilename
130   strcpy(command,"erase ");
131   strcat(command,tempFilename);
132
133   // Issue system erase command
134   system_command_status = system(command);
135
136   // status system command
137   if (system_command_status == -1) {
138         cerr << "Error with system command: " << command << endl;
139         exit(-5);
140   }
141
142   return 0;
143  }
```

Compile Commands

Microsoft Visual C++ 2008

```
cl ex8-13.cpp –EHsc
```

Note: executable name is ex8-13.exe

GNU gxx version 3.10b (for DOS)

```
g++ ex8-13.cpp -o grep.exe –Wall
```

```
******  Sample Program Execution  ******

C:\deanza\examples>grep system *.cpp
ex8-13.cpp[25]                          system_command_status;
ex8-13.cpp[61]  // system() allows you to issue operating system commands
ex8-13.cpp[62]  system_command_status = system(command);
ex8-13.cpp[64]  // check the status of the system command
ex8-13.cpp[65]  if (system_command_status == -1) {
ex8-13.cpp[66]          cerr << "Error with system command: " << command <<
endl
;
ex8-13.cpp[128]         // Issue system erase command
ex8-13.cpp[129]         system_command_status = system(command);
ex8-13.cpp[131]         // status system command
ex8-13.cpp[132]         if (system_command_status == -1) {
ex8-13.cpp[133]                 cerr << "Error with system command: " <<
command
 << endl;
all2htm.cpp[65]                 status = system(command);
Found 12 occurrence(s) in 96 file(s)


C:\djgpp\include> \deanza\examples\grep access *.h
dpmi.h[148] int __dpmi_get_descriptor_access_rights(int _selector);
                        /* LAR instruction  */
dpmi.h[149] int __dpmi_set_descriptor_access_rights(int _selector, int
_rights);
                        /* DPMI 0.9 AX=0009 */
io.h[37] #define sopen(path, access, shflag, mode) \
io.h[38]        open((path), (access)|(shflag), (mode))
unistd.h[76] int                access(const char *_path, int _amode);
unistd.h[125] /* additional access() checks */
Found 6 occurrence(s) in 60 file(s)
```

Note: this next sample run has quoted arguments for the command. The first argument allows you to search for more than one word in the file. The second argument is quoted, because the GNU executable version wanted wildcard arguments quoted.

```
C:\deanza\examples>grep "virtual function" "*.*"
```

```
EX7-15.CPP[33]        virtual double area(void) const = 0;           // pure
vir
tual function
EX7-15.CPP[34]   virtual double girth(void) const = 0;  // pure virtual
function

Found 2 occurrence(s) in 124 file(s)
```

Appendix A: Exercises

Exercise #1

Use the Date struct, the MonthName array, the function prototypes, the main() and the output below to complete the following program. Make sure that you use C++ ANSI standard header files. Try to match the output exactly. Turn in your entire program and the output.

```
struct Date
{
      unsigned short month;
      unsigned short day;
      unsigned short year;
};

const char* MonthName[12] = {"January","February","March","April","May",
      "June","July","August","September","October","November","December"};

void InputDate(Date*);
void Print1(Date);
void Print2(Date);
void Print3(Date);

int main()
{
      Date FirstDay, FinalDay, HoliDay;
      InputDate(&FirstDay);
      InputDate(&FinalDay);
      InputDate(&HoliDay);
      Print1(FirstDay);
      Print2(FirstDay);
      Print3(FirstDay);
      Print1(FinalDay);
      Print2(FinalDay);
      Print3(FinalDay);
      Print1(HoliDay);
      Print2(HoliDay);
      Print3(HoliDay);
      return 0;
}
```

****** Program Output ******

```
Enter month day year (separated by spaces) => 1 13 2005
Enter month day year (separated by spaces) => 1 1 2005
Enter month day year (separated by spaces) => 12 14 2005
01/13/05
January 13, 2005
13JAN05
01/01/05
January 1, 2005
01JAN05
12/14/05
December 14, 2005
14DEC05
```

Exercise #2

Write a complete C++ program that makes use of the student struct and main() shown below. The program should read in the student data from the keyboard, calculate the final grade and print the student data for two (or more) test cases.

Your program should meet the following requirements:

- Use the student struct with the indicated members.
- You should write at least four functions, getStudentData(), checkStudentData(), calculateFinalGrade(), and printStudentData(). Each of these functions should take a student struct argument, passed by reference.
- Your getStudentData() function should allocate memory dynamically for the last name. This memory should be released in main() before your program ends.
- checkStudentData() should be called by getStudentData(). It should verify the accuracy of the entered data. The ssn should be 9 numeric digits, the labs grades[6] must have a value between 0 and 25, the midterm between 0 and ??[1], and the final between 0 and ???[1]. If the entered data is incorrect, you may exit the program or ask the user to re-enter the data.
- The calculateFinalGrade() function should reflect the policies used to determine points and the final grade for the course. Remember to discard the lowest lab grade, but not the last lab.
- Your program should produce output similar to that shown on the next page. **<u>You must submit your output along with a program listing</u>**.
- Do not include a disk or email your solution.

```
struct student
{
  char* lastname;
  char ssn[10];
  int  lab_grade[NumLabGrades[1]];
  int  midterm;
  int  final;
  int total_points;
  char final_grade;
};

…
int main(void)
{
       student Me, You;
       getStudentData(Me);
       calculateFinalGrade(Me);

       getStudentData(You);
       calculateFinalGrade(You);
       printStudentData(Me);
       printStudentData(You);

…            <- something goes here
```

[6] Use the values stated on the course syllabus

```
            return 0;
      }


************************  Sample Run #1  ****************************

Enter the last name => Smith
Enter social security number => 123456789
Enter 9 lab grades (separated by a space) => 25 20 18 20 19 18 16 22 19
Enter midterm grade => 65
Enter final grade => 114
Thanks!

Enter the last name => Nguyen
Enter social security number => 987654321
Enter 9 lab grades (separated by a space) => 19 18 17 16 15 14 13 12 11
Enter midterm grade => 55
Enter final grade => 77
Thanks!

Name: Smith
SSN: 123456789
Lab grades: 25 20 18 20 19 18 16 22 19
Midterm: 65
Final: 114
Total Points: ???       <- you figure this out
Final grade: ?          <- you figure this out

Name: Nguyen
SSN: 987654321
Lab grades: 19 18 17 16 15 14 13 12 11
Midterm: 55
Final: 77
Total Points: ???       <- you figure this out
Final grade: ?          <- you figure this out

************************  Sample Run #2  ****************************

Enter the last name => Doe
Enter social security number => 345678901
Enter #[1] lab grades (separated by a space) => 11 12 11 12 11 12 …[1]
Enter midterm grade => 81
Enter final grade => 88
Thanks!

Invalid midterm 81.  Exiting...
```

Exercise #3

This assignment is a continuation of the exercise 2. The requirements for this assignment are the same as exercise 2. You are to convert the student struct to a class and make the four functions, getStudentData(), checkStudentData(), calculateFinalGrade(), and printStudentData(), members of your student class.

Make sure you complete the following steps for this assignment:

- Convert **student** to a class. The data members should be private. The three functions, getStudentData(), calculateFinalGrade(), and printStudentData(), should be public members functions. checkStudentData() should be private.

- Add a fifth function to the class - ***void DeleteName(void)***. This function should handle the delete for the lastname member.

- printStudentData() should be a const member function.

- Make DeleteName() an implicit inline function.

- In main(), prompt the user for the number of students to process. Dynamically allocate the memory for the student objects.

- Run your code with at least 4 student objects that demonstrate adequate testing of your program.

Exercise #4

Create a date class consisting of:

- 3 private unsigned int data members: month, day, and year.
- 5 constructors, as described below
- a destructor, as described below
- a print() function
- an increment() function

Use the following global variables:

```
const char* const Months[12] = {"January","February","March","April","May",
"June","July","August","September","October","November","December"};
const unsigned CurrentYear = 2009;
// DaysPerMonth - non-const so that changes can be made for leap year
unsigned DaysPerMonth[13] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
```

Constructors:

- The default constructor should look something like this:

```
// The default constructor returns the current system date
date::date() {
      time_t timer = time(0);
      tm* NOW = localtime(&timer);
      month = NOW->tm_mon+1;  // NOW->tm_mon is current month#-1
      day = NOW->tm_mday;
      year = NOW->tm_year + 1900;
}
```

- The time_t and tm types and the time() and localtime() functions are defined in the ANSI C header file, ***ctime***. Review the documentation for these types and functions.
- The second constructor should have 3 unsigned arguments, the third argument is ***default*** = CurrentYear. The 3 arguments should be assigned to the month, day, and year members respectively. Observe the assumption listed below concerning 2-digit years. This constructor should be able to take a 2-digit or a 4-digit year.
- The third constructor should take a char* argument of the form “mmddyy” or “mm/dd/yy”. The argument should be parsed and respective values assigned to the month, day, and year members. Observe the assumption listed below concerning 2-digit years.
- The fourth constructor should take a single unsigned int argument, the day of the year for the current year. For example, date(25) is January 25, 2009 and date(364) is December 30, 2009.
- The fifth constructor is the copy constructor.

The destructor should call the print() function to display the date that is being destructed.

The print() function, with a default int argument, should display dates with a format ***mm/dd/yy*** or ***Month d, yyyy*** or ***ddMONyy***, depending on the value of the argument. Note the uppercase in the 3rd format. Note: ***mm/dd/yy*** is the default format**.**

The increment() function should add 1 day to any date. This function must work for leap years. The rules for leap years are:

- A leap year occurs in every year that can be divided evenly by four, except the years that mark the even hundreds, such as 1500.
- The only century years that are leap years are those that can be divided evenly by 400, such as 1600 and 2000.

Assumptions:

- For the constructors, any 2-digit reference to a year is to be interpreted as follows: 1) assume any two-digit year reference < 50 refers to years between 2000 and 2049. 2) assume any two-digit year reference >= 50 refers to years between 1950 and 1999. For example, 02/04/94 means February 4, 1994, and 02/04/49 means February 4, 2049.

- Assume date values for month, day, year, and day of the year are correct. You do not have to perform error checking on these values.

Use this main() to test your program and redirect your output to a file. Turn in the output file with your program listing.

```
int main()
{
  int i;
  date D1;
  date D2(9,19,00);
  date D3(7,15,49);
  date D4(1,10);
  date D5("010148");
  date D6("10/01/59");
  date D7(2,26,1936);
  date D8(2,26,2000);
  date D9(2,26,1900);
  date D10(2,26,1999);
  date D11(25);
  date D12(364);
  date D13(D10);
  D1.print(); D1.print(1); D1.print(2);
  D2.print(); D2.print(1); D2.print(2);
  D3.print(); D3.print(1); D3.print(2);
  D4.print(); D4.print(1); D4.print(2);
  D5.print(); D5.print(1); D5.print(2);
  D6.print(); D6.print(1); D6.print(2);
  D7.print(); D7.print(1); D7.print(2);
  D8.print(); D8.print(1); D8.print(2);
  D9.print(); D9.print(1); D9.print(2);
  D10.print(); D10.print(1); D10.print(2);
  D11.print(); D11.print(1); D11.print(2);
  D12.print(); D12.print(1); D12.print(2);
  D13.print(); D13.print(1); D13.print(2);
  cout << "5 days after "; D7.print(1);
  for (i = 1; i <= 5; i++) D7.increment(); D7.print(1);
  cout << "5 days after "; D8.print(1);
  for (i = 1; i <= 5; i++) D8.increment(); D8.print(1);
```

```
  cout << "5 days after "; D9.print(1);
  for (i = 1; i <= 5; i++) D9.increment(); D9.print(1);
  cout << "5 days after "; D10.print(1);
  for (i = 1; i <= 5; i++) D10.increment(); D10.print(1);
  cout << "35 days after "; D1.print(1);
  for (i = 1; i <= 35; i++) D1.increment(); D1.print(1);
  return 0;
}
```

******** Output ********

```
05/01/09                              <- today's date
May 1, 2009
01MAY09
09/19/00
September 19, 2000
19SEP00
07/15/49
July 15, 2049
15JUL49
01/10/02
January 10, 2002
10JAN02
01/01/48
January 1, 2048
01JAN48
10/01/59
October 1, 1959
01OCT59
02/26/36
February 26, 1936
26FEB36
02/26/00
February 26, 2000
26FEB00
02/26/00
February 26, 1900
26FEB00
02/26/99
February 26, 1999
26FEB99
01/25/02
January 25, 2002
25JAN02
12/30/02
December 30, 2002
30DEC02
02/26/99
February 26, 1999
26FEB99
5 days after February 26, 1936
March 2, 1936                         <- you figure it out
5 days after February 26, 2000
March ?, 2000                         <- you figure it out
5 days after February 26, 1900
March ?, 1900                         <- you figure it out
5 days after February 26, 1999
```

```
March ?, 1999                          <- you figure it out
35 days after May 1, 2009
? ?, 2009
date destructed: 02/26/99
date destructed: 12/30/02
date destructed: 01/25/02
date destructed: 03/03/99
…
```

Exercise #5

This assignment will give you practice writing constructors and working with a multiple file application. Use the Point and Line header and source files from example 4-12 to complete this assignment. You are to create a Circle class consisting of:

- Two data members, a Point representing the center of the Circle and a double for the radius.
- A print() member function that displays Circle data as shown in the output on the next page. Your output should have the exact format as shown.
- The following ten Circle constructor functions:
  1. A default constructor the creates a *unit* Circle at the origin. That is, center (0,0) and radius 1.
  2. A constructor with one double argument. The argument should be used for the radius. The center is assumed to be at the origin.
  3. A constructor with two double arguments. The arguments should be used for x-y coordinates of the center. The radius is assumed to be 1.
  4. A constructor with a Point and a double argument. The Point should be assigned to the center and the double to the radius. Pass the Point by reference to const.
  5. A constructor with three double arguments. The first two doubles should be used for the x-y coordinates of the center. The third double represents the radius.
  6. A constructor with two Point arguments. The first Point is the center, and the second represents a Point on the circle. Both points should be passed by reference (to const).
  7. A constructor with a Point and a Line argument. The Point is the center. The Circle is tangent to the line. Assume that the Point does not lie on the Line. Pass the arguments by reference.
  8. A constructor with one Line argument. The Line represents the diameter of the Circle.
  9. A constructor with a Circle argument and a double argument. The Circle will be concentric to the argument's Circle with the double radius.
  10. A copy constructor.

Additional requirements:

- Use the main() provided on the next page.
- Use the files ex4-12p.h, ex4-12p.cpp, ex4-12lh, and ex4-12l.cpp for the Point and Line class definitions and member functions.
- Create a multi-file application. Put your Circle class definition in a separate header file and the Circle member function definitions in a separate source file.
- Turn in only the Circle header file, the Circle member function source file, main() as a separate source file and the program output.
- Use constructor initializers in every constructor, even though it is not required. This will help you become familiar with the syntax.

Use this main() for the program.

```
int main(void)

{
      Point P(2.3,1.3), Q(3.4,4.5), R(3.1,4.9);
      Line L(P,Q);
      Circle C1;
      C1.print();
      Circle C2(3.5);
      C2.print();
      Circle C3(2.6,3.5);
      C3.print();
      Circle C4(P,5.5);
      C4.print();
      Circle C5(1.1,2.3,5.8);
      C5.print();
      Circle C6(P,Q);
      C6.print();
      Circle C7(R,L);
      C7.print();
      Circle C8(L);
      C8.print();
      Circle C9(C8,5.0);
      C9.print();
      Circle C10(C9);
      C10.print();

      return 0;
}
```

Your output should look like this:

```
center=(0,0) radius=1
center=(0,0) radius=3.5
center=(2.6,3.5) radius=1
center=(?,?) radius=?
center=(?,?) radius=?
center=(?,?) radius=?
center=(?,?) radius=?
center=(?,?) radius=?
center=(?,?) radius=?
center=(?,?) radius=?
```

Exercise #6

This assignment will give you practice writing constructors and destructors, static data members and static member functions, and friend functions. You are to create a Dictionary application to store words. You will allocate memory dynamically to store each word. Before you store each word, you need to check in the Dictionary to make sure that the word is not already in the Dictionary. Follow all directions listed below.

Create a **Word** class with the following:

1. The class should contain two data members:
   - a char* data member, **ptrWord**.
   - a static int member, **WordCount** that contains the number of words added to the Dictionary.
2. Three constructors:
   - A default constructor that allocates memory for a one element char array, initialized to '\0'. (this will not get used in your final program)
   - A constructor with a const char* argument. This constructor should allocate memory dynamically to store the argument. This is the constructor that will be used to store your words.
   - A copy constructor. (this will not get used in your final program)
3. A destructor to perform the necessary release of memory.
4. A **print()** function the displays the word. It should be a const member function and define it inline.
5. Another const member function, **GetWord()** that returns the char*, **ptrWord**.
6. A static member function, **GetWordCount()** that returns the **WordCount**.

Create a **Dictionary** class with the following:

1. One data member, **words**, that is a 100 element array of pointers to **Word**.
2. A default constructor that initializes the 100 **Word** pointers to 0.
3. A destructor that releases memory for each **Word** added to the Dictionary.
4. A const member function, **FindWord(char*)**, that returns a pointer to the **Word** if it is in the Dictionary, otherwise a null pointer.
5. A function, **AddWord(char*)**, that is used to add words to the Dictionary. **AddWord** should allocate memory dynamically for each word to be added. (Hint: a **Word** constructor should help you out here) Use the **FindWord()** function to make sure that you are not adding a word into the Dictionary that is already there. **AddWord** should return an int, 1, if the word is successfully added, otherwise 0.
6. A **print()** const member function that prints out all words stored in the Dictionary.
7. A friend function, **void print(const Dictionary&, int n)**, that prints out the nth word in the Dictionary.

Use the following main() to test your program:

```
int main (void)
{
  Dictionary Webster;
  int i;
  char temp[25];
```

```
  cout << "Enter 10 words separated by whitespace\n";
  for (i = 0; i < 10; i++) {
    cin >> temp;
    Webster.AddWord(temp);
  }
  Webster.print();
  cout << "The fifth word is " ; print(Webster,4);
  cout << "There are " <<Word::GetWordCount()<<" words in the Dictionary\n";
  return 0;
}
```

The output should look something like this:

```
Enter 10 words separated by whitespace
dog cat bird mouse goat horse dog pig fish Dog
* Error: duplicate word: dog
The Dictionary contains:
dog
cat
bird
mouse
goat
horse
pig
fish
Dog
The fifth word is goat
There are 9 words in the Dictionary
```

Extra Credit (1 point each)
Do not attempt this unless you first complete the required assignment and totally understand what you did.

1. Modify the **Dictionary print()** function to print the words out in sorted order.

2. Implement the Dictionary class as a linked list. The Word class will need to be modified to add a Word* member. You will have to modify/add other Word class member functions to treat it as a "node". The main() function should not have to be changed. The Dictionary class member function arguments should stay the same, but the code should change.
   Use Example 5-9 as a guide for this assignment.

Exercise #7

This assignment will give you practice working with classes, constructors, static data member, static member functions, and friend functions. The goal of this assignment is to create a partial model of a population system, emulating the aging and dying of a population, but unfortunately not the birth process. But, maybe that's fortunate for the programmer

Program requirements:

- Create the three classes described below.
- Use the main() function included.
- Your output should look like that shown below.
- Your program must be divided into multiple files. Each class should be defined in a separate header file and the member functions for each class should be in a separate source file. main() should also be in a separate file. Your program should consist of at least 7 files. Print each file on a separate page.

Class Descriptions

The **date** class is used to represent calendar dates. As a minimum, the class must contain:

- 3 unsigned int data members to represent month, day, and year
- a default constructor (use the same one from assignment 4)
- a constructor that takes 3 unsigned ints as arguments
- an increment function that adds 1 day to the **date** (you can use the same one from assignment 3. For determining ages, you can assume that a year is 365.25 days long.
- a **display()** function that displays the **date** in the mm/dd/yy format
- a **let_time_pass()** function that adds a random number of days to a **date**. The random number should be between 1 and 365. You will be using this function to add days to the (global) TODAY **date**. Hint: you might use the random number to call the **increment()** function repeatedly.
- Declare the **human** class as a friend of the **date** class.
- Declare the function, **int difference_between_2_dates(date,date)** as a friend of the **date** class.

The **human** class must contain at least:

- 3 data members:
  char name[32]
  date birthday
  bool alive
- 2 static data members
  static human* oldest_human
  static unsigned long number_of_living_humans
- a constructor: **human(const char* n,const date& b)**
- necessary accessor functions
- a function, **age()** that returns a **human**'s age in years
- a **die()** function (you know what that means)
- a **display()** function

- a static member function that assigns the appropriate **human*** to the **oldest_human**.
- a static member function that returns the **number_of_living_humans**
- Declare the function **void population::display() const** as a friend of the **human** class.

The **population** class must contain:

- Two data members:
  - **human**** people
  - const unsigned long original_size
- A private member function: **determine_oldest()** that "sets" the **oldest_human**
- A constructor
- A destructor
- A **display()** function
- An **examine_population()** function that takes a look at the population, calls the following **roll_the_dice()** function for each "living" **human**. If **roll_the_dice()** returns a number greater than .5, the **human** should "die".

```
float roll_the_dice(unsigned short age)
{
      return (float) age*(rand()%100)/10000.;
}
```

Assumptions

- All **human**s were "born" in the last century.
- Use the **difference_between_2_dates**() function so that it always returns a positive value. That is, you should always subtract the older date from the newer date.
- Use a population size of 20 for the final testing of your program.

Hints and Suggestions

- If you are new to working with multiple files, keep your program as one file until you get it working, then try to split it into multiple files.
- Declare TODAY as a global variable. To do this, enter:
  date TODAY;
  in your main() source file and declare it as an extern in the date header file, like this:
  extern date TODAY;
- Create a global array of names that you can use in the population constructor, like this:
  char* NAMES[] = {"Fred","Sam","Sally","George","Sue","Mary","Bill",...};
- Do not work with a population size of 20 until you are sure that your program is working. Use a small population, like 4 or 5.

The main() function

```
int main()
{
      srand(time(0));               // seed the random number generator
      population World(POPULATION_SIZE);
      cout << "Today is ";
      TODAY.display();
      cout << endl;
      World.display();

      // let time pass until half of the world's population dies
      do
      {
            TODAY.let_time_pass();
            World.examine_population();   // record deaths, find oldest
      } while (human::get_number_of_living_humans() > POPULATION_SIZE/2);

      cout << "Today is ";
      TODAY.display();
      cout << endl;
      World.display();
      return 0;
}
```

Sample Output

```
Today is 11/22/03

Allen was born on 9/15/04 is 99
Catherine was born on 11/1/62 is 41
Naihui was born on 10/16/56 is 47
Gayatri was born on 9/26/24 is 79
Bin was born on 5/12/28 is 75
Evan was born on 5/2/82 is 21
Sandy was born on 3/8/50 is 53
Sridevi was born on 4/26/85 is 18
Tanya was born on 3/26/32 is 71
Jing was born on 9/18/25 is 78
Haiying was born on 7/24/75 is 28
Rose was born on 1/21/25 is 78
Nisha was born on 5/8/72 is 31
Hnin was born on 3/14/68 is 35
Adeline was born on 8/17/57 is 46
Chen Wei was born on 8/8/19 is 84
Joe was born on 7/6/86 is 17
Bob was born on 11/26/33 is 69
Mary was born on 7/8/41 is 62
Sue was born on 4/7/80 is 23
The oldest living person, Allen is 99 years old.

4/9/04 Allen died at the age of 99
4/9/04 Chen Wei died at the age of 84
4/9/04 Bob died at the age of 70
9/16/04 Sandy died at the age of 54
9/16/04 Tanya died at the age of 72
```

```
9/16/04 Rose died at the age of 79
9/16/04 Mary died at the age of 63
1/10/05 Bin died at the age of 76
8/5/05 Gayatri died at the age of 80
8/5/05 Jing died at the age of 79
Today is 8/5/05

Catherine was born on 11/1/62 is 42
Naihui was born on 10/16/56 is 48
Evan was born on 5/2/82 is 23
Sridevi was born on 4/26/85 is 20
Haiying was born on 7/24/75 is 30
Nisha was born on 5/8/72 is 33
Hnin was born on 3/14/68 is 37
Adeline was born on 8/17/57 is 47
Joe was born on 7/6/86 is 19
Sue was born on 4/7/80 is 25
The oldest living person, Catherine is 42 years old.
```

Exercise #8

This assignment will give you practice writing constructors and destructors, static data members, static member functions, friend functions, and overloaded operator functions. The purpose of the program is to write a simple game involving dice. You are to create the three classes described below. Use the main() provided. The sample output should give you additional information about the program requirements.

Create a **Die** class with the following specifications:
1. One private data member, an unsigned short, **value**.
2. A private member functions, **roll()** that generates a random number and assigns it to value. You can use the expression, **rand**() % 6 + 1, to generate the random number.
3. Add an overloaded **operator <** private member function that can be used to compare the value of two **Die** objects.
4. Name the class **Dice** as a friend of the **Die** class. Remember to forward declare the **Dice** class.
5. Reminder: all members of the **Die** class are private. Access must be controlled through the **Dice** class.

Create a **Dice** class with the following specifications:
1. One data member, **dice**, a five element array of **Die**.
2. A **roll()** function that calls the **Die roll()** to assign values to the **dice** array. This function should call the following **sort()** function.
3. A private member function **sort()** that will sort the **dice** array.
4. A **min()** function that returns the minimum of the **dice** array.
5. A **sum()** function that returns the sum of the **dice** array.
6. A **max()** function that returns the maximum of the **dice** array.
7. A **average()** function that returns the average of the **dice** array**, rounded to the nearest integer**.
8. A **median()** function that returns the median of the **dice** array.
9. A **averageOfHighAndLow()** function that returns the average of the maximum and minimum values of the **dice** array. "Round up" this average.. For example, if the high is 6 and the low is 1, your function should return 4 (that's 3.5 rounded up).

   The **min()**, **sum()**, **max()**, **average()**, **median()** and **averageOfHighAndLow()** functions should return an unsigned short.

10. An **overloaded!** Operator that prints the **value** of the 5 **dice** and the points for the roll.
11. An **overloaded+** operator that can be used to determine the point value of the **dice** array. The points are equal to the average + median + averageOfHighAndLow for the roll of the dice.

Create a **Player** class with the following specifications:
1. Private data members, char* **name** and unsigned short **score**.
2. Two ***static*** unsigned short data members, **NumberOfPlayers** and **PlayerNumberWhoseTurnItIs**.
3. A constructor and destructor.

4. Two accessor functions, **getName()** and **getScore()** to return the **name** and the **score**.
5. A **roll()** function that calls the **Dice roll()**, prints the value of the dice (hint: use the **!** operator of the **Dice** class) and returns the points for the roll(hint: use the **Dice** overloaded **operator+).**
6. A **takeTurn()** function that calls the **roll()** function, adds that turns points to the **score** and returns the **score**.
7. A ***static*** member function, **whoseTurnIsIt(),** that returns the **PlayerNumberWhoseTurnItIs.**
8. A ***static*** member function, **nextPlayer()**, that manages the **PlayerNumberWhoseTurnItIs.**
9. A ***static*** member function, **HowManyPlayers()** that returns the **NumberOfPlayers.**

Write one non-class member function, **printScores()** that prints out the player names and scores as shown the program output. It should have a **Player*** argument.

Additional requirements:
1. The game ends when one player scores 100 points.
2. Try to match the program output, except for the random points and scores.
3. Your program should use every function and data member listed. You should "reuse" alot of code.

Use this main():

```
int main(void) {
  Dice Lucky;
  Player Beatles[4];
  unsigned short PlayerScore = 0;
  printScores(Beatles);
  while (PlayerScore < 100) {
    Player::nextPlayer();
    PlayerScore=Beatles[Player::whoseTurnIsIt()].takeTurn(Lucky);
    printScores(Beatles);
  }
  cout << "The winner is "
       << Beatles[Player::whoseTurnIsIt()].getName() << endl;
  return 0;
}
```

Here is the program output:

```
Enter player name => John
Enter player name => Paul
Enter player name => George
Enter player name => Ringo
Scores: John 0   Scores: Paul 0   Scores: George 0   Scores: Ringo 0
------------------------------------------------------------------------
John, you rolled 1 3 5 5 6  - that's 13 points
Scores: John 13   Scores: Paul 0   Scores: George 0   Scores: Ringo 0
------------------------------------------------------------------------
Paul, you rolled 1 4 5 5 5  - that's 12 points
Scores: John 13   Scores: Paul 12   Scores: George 0   Scores: Ringo 0
------------------------------------------------------------------------
George, you rolled 1 2 3 4 4  - that's 9 points
Scores: John 13   Scores: Paul 12   Scores: George 9   Scores: Ringo 0
```

```
------------------------------------------------------------------------
Ringo, you rolled 1 2 2 4 5  - that's 8 points
Scores: John 13   Scores: Paul 12   Scores: George 9   Scores: Ringo 8
------------------------------------------------------------------------
John, you rolled 1 2 3 3 4  - that's 9 points
Scores: John 22   Scores: Paul 12   Scores: George 9   Scores: Ringo 8
------------------------------------------------------------------------
Paul, you rolled 2 3 4 6 6  - that's 12 points
Scores: John 22   Scores: Paul 24   Scores: George 9   Scores: Ringo 8
------------------------------------------------------------------------
George, you rolled 2 3 3 4 6  - that's 11 points
Scores: John 22   Scores: Paul 24   Scores: George 20   Scores: Ringo 8
------------------------------------------------------------------------
Ringo, you rolled 1 2 3 5 6  - that's 10 points
Scores: John 22   Scores: Paul 24   Scores: George 20   Scores: Ringo 18
------------------------------------------------------------------------
John, you rolled 1 2 3 3 4  - that's 9 points
Scores: John 31   Scores: Paul 24   Scores: George 20   Scores: Ringo 18
------------------------------------------------------------------------
Paul, you rolled 2 2 3 3 3  - that's 9 points
Scores: John 31   Scores: Paul 33   Scores: George 20   Scores: Ringo 18
------------------------------------------------------------------------
George, you rolled 2 3 3 6 6  - that's 11 points
Scores: John 31   Scores: Paul 33   Scores: George 31   Scores: Ringo 18
------------------------------------------------------------------------
Ringo, you rolled 1 2 4 6 6  - that's 12 points
Scores: John 31   Scores: Paul 33   Scores: George 31   Scores: Ringo 30
------------------------------------------------------------------------
John, you rolled 1 2 3 4 5  - that's 9 points
Scores: John 40   Scores: Paul 33   Scores: George 31   Scores: Ringo 30
------------------------------------------------------------------------
Paul, you rolled 1 2 3 4 5  - that's 9 points
Scores: John 40   Scores: Paul 42   Scores: George 31   Scores: Ringo 30
…
```

```
…
George, you rolled 1 2 2 4 6  - that's 9 points
Scores: John 98   Scores: Paul 99   Scores: George 91   Scores: Ringo 87
------------------------------------------------------------------------
Ringo, you rolled 1 2 4 4 6  - that's 11 points
Scores: John 98   Scores: Paul 99   Scores: George 91   Scores: Ringo 98
------------------------------------------------------------------------
John, you rolled 2 2 3 4 5  - that's 10 points
Scores: John 108   Scores: Paul 99   Scores: George 91   Scores: Ringo 98
------------------------------------------------------------------------
The winner is John
```

Exercise #9

Create a **Money** class consisting of an unsigned int and an unsigned short data member, **dollars** and **cents**. You may assume that all money values are non-negative. Add the following member functions:

1. A constructor with two default arguments, an unsigned int and an unsigned short. The arguments should initialize the dollars and cents members. Since both arguments have default values (both 0), this constructor also serves as a default constructor.
2. A constructor with a *double* argument. The double must be used to initialize both the dollars and cents. For example, an argument of 1.75 should assign 1 to the dollars and 75 to the cents. Round off cents to the second decimal place, so 5.55555 would set the dollars to 5 and the cents to 56.
3. A copy constructor.
4. An overloaded !(unary) operator that serves as a print function. It should print the dollars and cents with a leading dollar sign and a decimal point separating the dollars and cents. Make sure you are able to print the following values: $1.03, $10.00, $0.01, $1234.56, and $0.00.
5. An overloaded + (unary) operator that "reduces" Money. For example, if you passed 5 and 150 into the first constructor, you would want to use this function to change the Money object to have dollars = 6 and cents = 50. This function should return Money by reference.
6. An overloaded < (binary) operator that tests two Money objects to see if the first is less than the second. This function should return a bool.
7. An overloaded == (binary) operator that tests two Money objects to see if the first is equal to the second. This function should return a bool.
8. An overloaded + (binary) operator that adds two Money objects and returns the sum by value(Money). Make sure your logic can handle $2.56+$5.67 and $5.63+$0.37.
9. An overloaded - (binary) operator that subtracts two Money objects and returns the difference. It should return a Money by value. If you try to subtract a larger Money from a smaller one, print an error message and have the function return $0.00. (0 dollars and 0 cents).
10. An overloaded * (binary) operator with a double argument. This permits you to multiple a Money object by a double, For example, $4.29*67.3. It should return a Money by value.
11. An overloaded += operator that adds more Money to a Money object and returns the result by reference. For example, M1 += M2; (this should change M1).

Divide your final program into three files: a header file for your Money class, a source file for your Money methods, and another source file for main(). Write you own main() and thoroughly test all functions. Make sure your main() demonstrates calls to each member function.

Exercise #10

This assignment will give you practice with overloaded operator functions.
Create a Date class to represent calendar dates.
The class should contain the following members:

- Three unsigned short data members to represent day, month, and year.
- A static const unsigned short 12-element array containing the number of days in each month.
- A constructor that initializes the three unsigned short members.

It should contain the following overloaded operator member functions:

1. A ! operator that prints a Date object using the format mm/dd/yy.
2. A prefix ++ operator that adds a day to a Date object. It should return the object by reference.
3. A postfix ++ operator that adds a day to a Date object. It should return the object by value.
4. A prefix -- operator that subtracts a day from a Date object. It should return the object by reference.
5. A binary + operator with an unsigned short argument. This function should add a number of days to a date object. Use call(s) to the prefix ++ operator in your function definition. It should return a Date object by value.
6. A binary - operator with an unsigned short argument. This function should subtract a number of days from a date object. Use call(s) to the prefix -- operator in your function definition. It should return a Date object by value.
7. A += operator with with an unsigned short argument. This function should add a number of days to a date object. Use call(s) to the prefix ++ operator in your function definition. It should return a Date object by reference.
8. A == operator that determines if two Date objects are equal. This function should return a bool.
9. A != operator that determines if two Date objects are not equal. This function should return a bool.
10. A > operator that determines if the "current" Date object is greater than another Date object. One Date object is greater than another Date object if it occurs after the other object. For example, "03/15/02" > "12/25/01". This function should return a bool.
11. A binary – operator with a const Date& argument. This function should determine the number of days between two Dates. The function should return an int. For example, "01/03/02" – "12/30/01" ➔ 4 and "12/30/01" – "01/03/02" ➔ -4.

Appropriate member functions should be defined as const member functions.

Assumptions

It is not necessary to make leap year corrections for this assignment. Assume that February always has 28 days and that a year is exactly 365 days.
A year value between 0 and 49 represents the years 2000 to 2049. A value between 50 and 99 represents the years 1950 to 1999.

Write a main() that demonstrates each of the overloaded member functions. Your output should demonstrate the validity of each function.

Exercise #11

This assignment will give you practice with inheritance and polymorphism.

Create the following classes:

GeometricObject: This is the base class for the other four classes. It should have two protected data members, x and y (coordinates). The class should have the following member functions:

```
GeometricObject()     // ctor
getX()          // accessor function
getY()          // accessor function
print()
describeYourself()
length()
area()
```

All member functions should be const member functions, except the constructor. describeYourself(), length(), and area() functions should be pure virtual functions. The print() function should display the line of output shown below, like
GeometricObject: 0x????? - location = (x,y)

point — Is derived from GeometricObject and has no data members. Its length() and area() functions should return 0.0.

line — Is also derived from GeometricObject and has two **point** data members, p1 and p2. Its constructor must initialize the x,y members of the GeometricObject class. The (x,y) coordinates of a line should be the midpoint of p1 and p2 (take the average of the x coordinates and the average of the y coordinates). Its area() function should return 0.0, but the length() function should return the distance between the two points. Use the formula:

$$d = \sqrt{(p1.x - p2.x)^2 + (p1.y - p2.y)^2}$$

circle — Is also derived from GeometricObject and has one data member, a double, radius. The constructor should have a point argument and a double argument. The point argument is used to initialize the (x,y) coordinates of the GeometricObject base and the double to initialize the radius. The length() function should return the circle's circumference and the area() function πr^2.

triangle — Is also derived from GeometricObject and has 3 point data members, p1, p2, and p3. Its constructor should have three (point) arguments, to initialize the data members. It also needs to initialize the GeometricObject's (x,y) coordinates. The (x,y) coordinate for a triangle

should be the average of its x coordinates and the average of its y coordinates. Add three private member functions, side1(), side2(), side3() to the class. They should each return a double length of a side. For example, side1() should return the length of the side that is opposite point p1. These functions will be useful for the length() and the area(). The length() should return the triangle's perimeter. Use the following formulas for the area of a triangle:

$$s = \tfrac{1}{2}(a + b + c) \qquad \text{(s is the semi-perimeter of a triangle)}$$

$$\text{Area} = \sqrt{s(s - a)(s - b)(s - c)}$$

Use the main() below to test your program. The output should give you an indication of what is expected.

```
int main(void)
{
  int i;
  Point P(0,0), Q(3,0), R(3,4);
  P.DescribeYourself();
  Line L(Q,R);
  L. DescribeYourself ();
  Circle C(P,3);
  C. DescribeYourself();
  Triangle T(P,Q,R);
  T. DescribeYourself();

// polymorphism testing
  GeometricObject* Obj[6];
  Obj[0] = new Point(2,1);
  Obj[1] = new Point(8,1);
  Obj[2] = new Point(5,5);
  Obj[3] = new Line(P,Q);
  Obj[4] = new Circle(P,2.0);
  Obj[5] = new Triangle(P,Q,R);

  for (i = 0; i < 6; i++) Obj[i]-> DescribeYourself();

  for (i = 0; i < 6; i++) delete Obj[i];

  return 0;
}
```

******** Program Output ********

```
GeometricObject: 0x6a0c4 - location = (0,0)
I am a point

GeometricObject: 0x6a060 - location = (3,2)
I am a line
Length=4

GeometricObject: 0x6a038 - location = (0,0)
```

```
I am a circle
Area=28.2743  circumference=18.8496

GeometricObject: 0x69fd4 - location = (2,1.33333)
I am a triangle
Area=6  permeter=12

GeometricObject: 0x6c160 - location = (2,1)
I am a point

GeometricObject: 0x6c180 - location = (8,1)
I am a point

GeometricObject: 0x6c1a0 - location = (5,5)
I am a point

GeometricObject: 0x6b100 - location = (?,?)
I am a line
Length=?

GeometricObject: 0x6c1c0 - location = (?,?)
I am a circle
Area=??????  circumference=??????

GeometricObject: 0x6e100 - location = (?,?)
I am a triangle
Area=?  permeter=??
```

Extra Credit (1 point0) Add a constructors to the circle class, so that you can create a circle: using a point and a line - the first point is the center, the circle is tangent to the line.

Exercise #12

This assignment will give you practice with inheritance and polymorphism.

1. Create an abstract Solid base class. It should consist of:
   - data members to represent the (x,y,z) coordinates of a solid in 3D space
   - at least one constructor
   - a function that displays the coordinates of a Solid object as (x,y,z)
   - pure virtual functions that:
     - return the volume of the object
     - return the surface area of the object
     - return the type of the solid
     - print "specialized" details about a Solid object (i.e. radius, height, width, …)
   - a non-virtual function that prints all information about a Solid object (it should call the 5 functions listed above)

2. Derive from the Solid class the following classes:
   RectangularSolid
   This class should have three members: length, width, and height

   Sphere
   This class should have one member, radius.

   Cylinder
   This class should have two data members, radius and height.

   Cone
   This class should have two data members, radius and height.

3. From the **RectangularSolid** class, derive a **Cube** class. It does not have any data members. It only needs a constructor, a type function, and a "print details" function.

You may need the following formulas for this assignment:

Solid	Volume	Surface Area
Rectangular Solid	$V = l\ w\ h$	$A = 2(lw + lh + wh)$
Sphere	$V = 4/3\pi r^3$	$A = 4\pi r^2$
Cone	$V = 1/3\pi r^2 h$	$A = \pi r\sqrt{r^2 + h^2} + \pi r^2$
Cylinder	$V = \pi r^2 h$	$A = 2\pi r(h + r)$
Cube	$V = s^3$	$A = 6s^2$

Use the following main() as a final test of your program:

```
int main() {
      RectangularSolid        Rec(1.,2.,3.,4.,5.,6.);
      Sphere                  Sph(1.,2.,3.,4.);
      Cylinder                Cyl(1.,2.,3.,4.,5.);
      Cone                    Con(1.,2.,3.,4.,5.);
      Cube                    Cub(1.,2.,3.,4.);
      Solid*                  ps;

      // Rectangular Solid test
      ps = &Rec;
      ps->print();

      // Sphere test
      ps = &Sph;
      ps->print();

      // Cylinder test
      ps = &Cyl ;
      ps->print();

      // Cone test
      ps = &Con ;
      ps->print();

      // Cube test
      ps = &Cub ;
      ps->print();
      return 0;
}

******  Program Output  ******

I am a rectangular solid located at (1,2,3)
length=4 width=5 height=6
volume=120 surface area=148

I am a sphere located at (1,2,3)
radius=4
volume=268.082 surface area=???

I am a cylinder located at (1,2,3)
radius=4 height=5
volume=??? surface area=???

I am a cone located at (1,2,3)
radius=4 height=5
volume=??? surface area=???

I am a cube located at (1,2,3)
side=4
volume=64 surface area=???
```

Exercise #13

This assignment will give you practice in producing formatted output using C++ input/output classes. Create a class consisting of an int, an unsigned, a long, a short, and a double. Generate random numbers to assign to the members. Print 20 lines of output using the class. Each line should be printed exactly as illustrated below. Of course, the numbers will not match since they are random. The output specifications are:

The first column contains the int member left justified displayed in octal.
The second column contains the unsigned member left justified displayed in hex.
The third column contains the long member left justified displayed in decimal.
The fourth column contains the short member left justified displayed in hex.
The fifth column contains the double member right justified.
The sixth column contains the double member right justified.
The seventh column contains the double member right justified.

The integer types should be printed in a field of width 9, the doubles using a width of 12. Make sure you match the base indicators and the precision of the doubles.

Note: for the scientific output shown in the last two columns, your compiler will print the exponent as either a 2-digit or 3-digit number, but not both. You cannot control this. You will have to figure out a way to massage the double value to produce one of the scientific outputs.

```
62436    0X22E8   2888     1889          426.0986  4.261e+002   4.26e+02
72062    0X29AF   7884     106c            5.1463  5.146e+000   5.15e+00
47262    0X68E8   19107    5a62            0.9017  9.017e-001   9.02e-01
31237    0X1A6C   29243    2cce           39.5479  3.955e+001   3.95e+01
13020    0X33FD   17339    459d            0.0789  7.890e-002   7.89e-02
74524    0X7E67   8056     5b9e            0.6746  6.746e-001   6.75e-01
45040    0X1842   21039    596             1.3779  1.378e+000   1.38e+00
14051    0X1995   1452     408c            1.8856  1.886e+000   1.89e+00
12706    0X24CE   17308    6bfe            0.0308  3.078e-002   3.08e-02
37534    0X66C5   21689    6e05            0.9871  9.871e-001   9.87e-01
15311    0X2AC5   15003    a87             3.0408  3.041e+000   3.04e+00
56020    0X8D1    3278     200b            0.0388  3.877e-002   3.88e-02
67351    0X23A1   12797    41d1            1.9757  1.976e+000   1.98e+00
76410    0X3908   21582    5a36            0.4780  4.780e-001   4.78e-01
13472    0X42C1   7242     7193            4.8966  4.897e+000   4.90e+00
35644    0X5EDA   16355    350a            2.4271  2.427e+000   2.43e+00
16706    0X7E32   23414    360d            1.0195  1.019e+000   1.02e+00
62433    0X3CC4   26845    667b            1.1618  1.162e+000   1.16e+00
16543    0X1ABC   3301     5eac            0.6685  6.685e-001   6.69e-01
57772    0X5E7E   28111    51be            9.0853  9.085e+000   9.09e+00
```

Exercise #14

This assignment will give you practice with C++ input/output classes and file I/O. Use the following program specifications and use the following main() to test your program. You should use the WordFile and the Dictionary classes described below, and any others you wish.

Here is the WordFile class. Write the 4 member functions. Add any others you desire. The getNextWord() function should place the "next word to be read" in the buffer argument and return it. If getNextWord() fails, it should return a null pointer.

```
class WordFile {
  private:
    fstream File;
  public:
    WordFile(const char* filename);
    void addWord(const char* word);
    void goToTopOfFile();
    char* getNextWord(char* buffer);
};
```

The Dictionary class should contain two members, an fstream object and an unsigned int, which stores the number of words in the Dictionary. The Dictionary constructor should use a WordFile& argument. This constructor should read the WordFile's file into memory. Use dynamic memory allocation to temporarily store the words, so they can be sorted. After sorting, write them out to the new Dictionary file and clean up.

Add four more members functions to the Dictionary class, getNumWords(), getDictionarySizeInBytes(), getMiddleWord() and find(). The function, getMiddleWord(), should return the word that's in the middle of the file. That is, if the file is 100 bytes, then the function should return the word that begins on or before byte 50.

Add a friend operator<<() to the Dictionary class. This function should print out the Dictionary.

Use the main() below for the final testing of your program. It, and the sample program run should give you more insight to the program. If you do not complete the program, turn in only the functions and the parts of the program that run, along with the output. Do not turn in a program that does not run.

Test your code thoroughly after you complete each function. Do not attempt the Dictionary class until you WordFile class is complete.

```
int main() {
  WordFile Words("wordfile.txt");
  char buffer[MaxWordSize];
// Read words into the Word file
  cout << "Enter words for the Word file (“quit” to stop)\n";
  while (cin.getline(buffer,MaxWordSize)&&strcmp(buffer,”quit”))
    Words.addWord(buffer);
  Dictionary Webster(Words);
```

```
// Print the Dictionary
  cout << Webster << endl;
// Print the Dictionary size
  cout << "Dictionary size = " << Webster.getDictionarySizeInBytes() << endl;
// Print the word in the middle of the dictionary
  cout << "The middle word is " << Webster.getMiddleWord(buffer) << endl;

// Search for words in the Dictionary
  cout << "Enter words to search for in the Dictionary (“\quit\” to stop)\n";
  cin.clear();
  while (cin.getline(buffer,MaxWordSize) && strcmp(buffer,"quit")) {
    cout << buffer << " is ";
    if (Webster.find(buffer)) cout << "definitely ";
    else cout << "NOT ";
    cout << "in the Dictionary\n";
  }
  return 0;
}
```

******** Sample Run ********

```
Enter words for the Word file (enter "quit" to stop)
chimpanzee
whale
bald eagle
tiger
zebra
mouse
horse fly
mountain goat
baboon
quit
Dictionary Words:
        baboon
        bald eagle
        chimpanzee
        horse fly
        mountain goat
        mouse
        …

Dictionary size = 86                <- Note this could be a different size
The middle word is ???              <- this may vary
Enter words to search for in the Dictionary ("quit" to stop)
elephant
elephant is NOT in the Dictionary
goat
goat is NOT in the Dictionary
baboon
baboon is definitely in the Dictionary
zebra
zebra is definitely in the Dictionary
mountain goat
mountain goat is definitely in the Dictionary
dog
dog is NOT in the Dictionary
quit
```

Exercise #15

Read in the input file below and produce the report shown.

Input file

```
John,Doe,123456789,20,21,22,23,16,19,16,50,75
Francisco,Washington,987654321,10,0,20,13,18,19,16,30,70
Tom,Nguyen,111111111,18,23,24,25,17,22,20,38,90
Victoria,Black,333333333,22,21,22,21,20,22,21,45,64
Sally,Seinfield,444444444,17,12,19,23,24,12,11,34,94
Sylvester,De La Rosa,555555555,25,25,24,20,25,25,21,44,80
George,O'Neill,666666666,21,12,3,14,21,14,17,45,99
Sylvia,Smart,777777777,20,21,22,23,24,20,25,44,78
Judy,Yang,888888888,16,19,22,24,25,20,25,45,100
Charles,Black,222222222,20,21,22,22,21,25,16,40,86
```

```
                          CIS27 Class Grades Report

Student Name          --- SSN ---  ---- Lab Grades ----  Mid  Fin  Pts   Perct  G
--------------------  -----------  -- -- -- -- -- -- --  ---  ---  ---   -----  -
Doe, John             123-45-6789  20 21 22 23 16 19 16   50   75  246   82.0%  B
Washington, Francisc  987-65-4321  10  0 20 13 18 19 16   30   70  196   65.3%  D
Nguyen, Tom           111-11-1111  18 23 24 25 17 22 20   38   90  260   86.7%  B
Black, Victoria       333-33-3333  22 21 22 21 20 22 21   45   64  238   79.3%  C
Seinfield, Sally      444-44-4444  17 12 19 23 24 12 11   34   94  234   78.0%  C
De La Rosa, Sylveste  555-55-5555  25 25 24 20 25 25 21   44   80  269   89.7%  B
O'Neill, George       666-66-6666  21 12  3 14 21 14 17   45   99  243   81.0%  B
Smart, Sylvia         777-77-7777  20 21 22 23 24 20 25   44   78  257   85.7%  B
Yang, Judy            888-88-8888  16 19 22 24 25 20 25   45  100  280   93.3%  A
Black, Charles        222-22-2222  20 21 22 22 21 25 16   40   86  253   84.3%  B
```

Program requirements

- Use C++ input/output techniques and file I/O, no stdio. You should declare one ifstream and one ofstream object.
- Create at least 3 classes:
  1. Name consists of a first and last name.
  2. StudentInfo consists of a Name, SSN, 7 lab grades, a midterm, final, and whatever else you want.
  3. Class consists of an array of StudentInfo or an array of StudentInfo pointers
- The rules for calculating points and grades are exactly like the 1st assignment or what was stated on the course syllabus. Remember to discard the lowest lab grade, but not the last one.
- Use the same input file shown. You may get a copy of the data file in the ATC if you do not want to type it in.
- Produce exactly the report, with the same spacing, formatting and text. Turn in a copy of the report file along with your program listing.
- If you are unsure of any program detail, ask the instructor for clarification.

Extra Credit (1 point each)

1. Sort the report by Name (major sort by last name, minor sort by first name)
2. Handle a missing lab score instead of a 0. Use this record for Francisco Washington: Francisco,Washington,987654321,10,,20,13,18,19,16,60,140
3. Change the lab grades in the input file to octal, but print them out as hexadecimal.

Exercise #16

Purpose

The purpose of this assignment is

- To give you practice in using the C++ input/output classes and file I/O.
- To give you practice in program planning, design and development.
- To solve a practical "real-world" problem.

Program Description

This program will track a portfolio consisting of three mutual funds over a ten year period. You will invest $10,000 exactly 10 years prior to the date that you run your program. You will use actual mutual fund historical data that you will download as input files to your program.

Requirements

Your must use a main() that is ***similar*** to the sample below. The "real work" should not be performed in main().
Your program output file should "logically" match that of the sample output below. You will probably use different mutual funds and run your program for different dates. You must display the initial investment data for your portfolio, the value of the portfolio 5 years ago, 3 years ago, 1 year ago, at the beginning of the year and the current value.
You should turn in the program listing and your output file.
Your solution must contain at least 3 classes, one of which is a "date" class. The "date" class must contain an overloaded insertion operator that "prints" a "date" in an "mm/dd/yy" format.
You are to use the actual downloaded mutual fund historical data. You may not edit this data.
You are to use the closing prices of the mutual on the date of interest. If that date is not a "market open" date, then you must backup and get the previous closing price of the fund.
You must invest at least $3000 in each fund.

References

Yahoo finance page: http://finance.yahoo.com/
You can look up mutual fund data by entering the "ticker" symbol next to the Get Quotes button at the top of the page. On the mutual fund "Summary" page, use the "Historical Prices" link on the left side of the page to get the history. On the "Historical Prices" page, use the "Download to Spreadsheet" link to download the mutual fund history to a file. Note, the download file is named table.csv, so you need to give it a unique name, since you'll need three different mutual fund history files.

You can find lots of good funds on these sites:
http://bloomberg.com/apps/data?pid=invest_mutualfunds
http://www.kiplinger.com/investing/funds/kip25/tables/index.php
http://www.smartmoney.com/top25funds/
http://www.morningstar.com/allanalyses/analysesLists.html?type=FO&fsection=all2000&lpos=Commentary
http://moneycentral.msn.com/investor/research/fundwelcome.asp?Funds=1

Assumptions

Assume that the input data is reliable, that dates and the mutual fund closing prices are valid. Assume that you do not have access to dividends or capital gains. There is no reinvestment. You give all that money to charity (or the teacher).

Suggestions

Allow 4-12 hours to solve this problem. You will need more time for program planning and analysis than the previous assignments.

Extra Credit

The student with the most valuable portfolio using "today's" closing prices will receive 2 extra credit points. In the event of a tie, only 1 point will be awarded. Remember, actual data must be used for this, but you are free to run the program on different days. By completing the program early, you can pick a day with a good market close.

Sample main()

```
int main()
{
    Date today;
    Portfolio myPortfolio(today.nYearsBefore(10));
    myPortfolio.addFund("VTSMX",3333.33f,"c:/deanza/data/vtsmx.csv");
    myPortfolio.addFund("VGTSX",3333.33f,"c:/deanza/data/vgtsx.csv");
    myPortfolio.addFund("VBMFX",3333.34f,"c:/deanza/data/vbmfx.csv");
    ofstream fout("c:/deanza/data/ass9.out");
    myPortfolio.report(today,fout);
    return 0;
}
```

Sample Program Output File

```
Mutual Funds                        VTSMX       VGTSX       VBMFX       Total

Initial Investment   06/06/99     3333.33     3333.33     3333.34    10000.00
Initial Shares                    132.380     358.808     572.739

Value 5 years ago    06/06/04     3243.31     3519.91     4547.55    11310.77
Value 3 years ago    06/06/06     3828.43     5098.67     4874.01    13801.11
Value 1 year ago     06/06/08     4314.27     6562.61     5544.11    16420.98
Value on January 1st 01/01/09     2953.40     3950.48     5715.93    12619.81
Value today          06/06/09     3060.63     4345.17     5738.84    13144.64
```

Index