

# CS 215 Manual

# **Spring 2009**

Sonoma State University  
Computer Science Department  
Instructor: Dr. Tia Watts  
[tia.watts@sonoma.edu](mailto:tia.watts@sonoma.edu)

# Table of Contents

Course Syllabus – Spring 2009 .....	3
Proposed Course Schedule – Spring 2009.....	5
Policy on Collaboration.....	6
Policy on Incomplete Grade .....	6
Laboratory Assignment and Project Submission Instructions.....	9
Laboratory Exercise 1 (C++ Review).....	11
Laboratory Exercise 2 (Algorithm Development).....	23
Laboratory Exercise 3 (C++ structs) .....	27
Laboratory Exercise 4 (C++ classes).....	31
Laboratory Exercise 5 (The STL).....	35
Laboratory Exercise 6 (Operator Overloading).....	39
Laboratory Exercise 7 (Makefiles and Templates).....	43
Laboratory Exercise 8 (Dynamic memory allocation) .....	49
Laboratory Exercise 9 (Template classes).....	61
Laboratory Exercise 10 (Doubly Linked Lists).....	67
Laboratory Exercise 11 (Templated Lists) .....	75
Laboratory Exercise 12 (Exception Handling).....	83
Laboratory Exercise 13 (Intro to MFC).....	89
Laboratory Exercise 14 (Inheritance and Polymorphism).....	97
Laboratory Exercise 15 (Ordered Linked Lists).....	107
Laboratory Exercise 16 (More MFC).....	115
Exam 1 Review Questions.....	129
Exam 2 Review Questions.....	130
Final Review Questions.....	132
Code Analysis Homework Assignment.....	135
Summary of Essential Linux Commands .....	144
Summary of Essential emacs Commands.....	147
Summary of Essential vi Commands.....	148
GDB QUICK REFERENCE GDB Version 4 .....	151

# Course Syllabus – Spring 2009

## Catalog Description

Lecture, 2 hours; laboratory, 3 hours. Pointers and dynamic allocation of storage; linked lists; an introduction to the object oriented programming (OOP) paradigm; classes and objects; encapsulation; member variables and member functions; inheritance and polymorphism; scoping; templates; iterators; error handling techniques. Prerequisite: Completion of CS 110 and 115 with a C- or better, or consent of instructor.

This course is currently taught using C++.

## Prerequisites

CS 115 (CS 150) or instructor consent. Basic C/C++ programming skills are required. Students lacking a background in C/C++ will need to familiarize themselves with the language within the first two weeks of the course. Students who have not acquired a strong foundation in the fundamentals of procedural programming (i.e., earned a grade below a C- in CS 115) must retake CS 115 or its equivalent before taking this course.

## Instructor

Dr. Tia Watts Darwin 116E (707) 644-2807  
e-mail: [tiawatts@cs.sonoma.edu](mailto:tiawatts@cs.sonoma.edu) (Message subject MUST begin with **CS 215**)  
Office Hours: Tuesday 12:00 - 1:00 pm Office or Lab  
Thursday 2:00 - 3:00 pm Office or Lab

## Class Meetings

Lecture: Monday/Wednesday 2:30 – 3:45 pm Darwin 31  
Lab Section: Wednesday 10:00 am – 12:50 pm Darwin 28

## Course Objectives

Upon successful completion of this course, the student will be able to:

- Use predefined classes to declare objects in a C++ program.
- Design, implement, and use new C++ classes.
- Employ operator overloading in C++ classes.
- Design, implement, and use template classes.
- Use single and multi-dimensional arrays in a C++ class/program.
- Use static and dynamic arrays in a C++ class/program.
- Develop and implement recursive algorithms
- Design, implement, and use linked lists in a C++ class/program.
- Use container classes from the Standard Template Library (STL).
- Employ inheritance and polymorphism in C++ classes.
- Design and implement applications using a library of application development classes

## Important Dates

26 January 2009	First day of classes
6 February 2009	Last day to ADD or DROP courses
16 February 2009	President's Day Observed (No classes – University closed)
20 February 2009	Last day to WITHDRAW from courses online
4 March 2009	Midterm Exam 1
31 March 2009	Cesar Chavez Day Observed (No classes – University closed)
8 April 2009	Midterm Exam 2
13-17 April 2009	Spring Break (No classes – University open)
15 May 2009	Last day of classes
18 May 2009	Final Exam: 2:00 - 3:50 pm (tentative)

## Course Materials

### Text

Absolute C++; (3rd Edition)

by Walter Savitch; Addison Wesley Publishers; 200?

Getting Started with Microsoft Visual C++ 6 with an Introduction to MFC (2nd Edition)

by Harvey M. Deitel, Paul J. Deitel, Tem Nieto, and Edward Strassberger

### Other Materials

CS 215 Lab Manual

Loose Leaf Binder; Hole Punch; Small Stapler

## Coursework

### Lecture

The proposed outline of the topics to be covered appears in the course schedule. Students are expected to attend all lectures and to read the relevant sections of the text prior to lecture. Students are responsible for making up the missed work if they are absent. *All electronic devices must be turned off during lecture.*

### Labs

The lab meetings provide an opportunity for students in the class to interact with each other and with the instructor, to get immediate solutions to coding problems, and to get advice on the process of programming. The labs for this course are designed to give the student an opportunity to work with a variety of programming concepts. Lab attendance is essentially mandatory. Lab exercises are due at the end of the lab period; an extension (until the beginning of the next lab) will automatically be given *only* to students who have not completed the assignment but have attended the full lab.

### Programming Projects

Several programming projects will be assigned during the semester. Each project will require that you write one or more C++ program modules. Project due dates and times will be stated on the project assignment handout. Projects must be submitted by the due date and time. Submission times will be based on the UNIX date/time stamp on the file(s). No projects will be accepted after the due date and time.

### Midterms and Final Exam

The examinations are based on the assumption that each student is responsible for knowing the material covered in class (lecture and lab) and also material covered in relevant sections of the course text. All exams are closed book; however, one 8 ½ by 11 *handwritten* sheet of notes (front and back) will be allowed. The final exam will be comprehensive. There may be one or more unannounced quizzes given during the semester. Exams cannot be made up unless you have made arrangements with the instructor prior to the date of the exam. Quizzes cannot be made up.

## Grading

Homework, Lab Attendance and Exercises	20%
Programming Project	30%
Exams and Quizzes	50%

### Grading Scale:

100	...	93	...	90	...	87	...	83	...	80	...	77	...	73	...	70	...	67	...	63	...	60	...	0
	A		A-		B+		B		B-		C+		C		C-		D+		D		D-		F	

Note: You must separately earn a passing grade on the programming projects and the exams in order to pass the course.

CS Majors must take this course for a letter grade. University guidelines regarding the grade of Incomplete will be strictly adhered to. Incomplete grades will only be given for circumstances beyond a student's control; inability to keep up with the work due to an excessive course load, for example, is insufficient to warrant an Incomplete. The University also requires that, to be a candidate for an Incomplete grade, the student must currently be doing C (or better) work in the course.

## Proposed Course Schedule – Spring 2009

Week	Monday	Lecture/Lab Topics	Lab	Reading	Activities
1	26 January	C++ Review	1	Ch 1-4, 9	
2	2 February	Arrays	2	Ch 5	Project 1 assigned
3	9 February	Structs, arrays of structs, sorting	3	Ch 6	
4	16 February	Class design and implementation	4	Ch 6, 7	Project 1 due; Project 2 assigned
5	23 February	Operator overloading, friend functions	5	Ch 8	
6	2 March	Arrays of objects, STL vector class	6	Ch 7, 19	Exam 1 (4 March)
7	9 March	Modular programming, “make” files, recursion	7	Ch 11	Project 2 due; Project 3 assigned
8	13 March	Template classes	8	Ch 16	
9	23 March	Pointers, linked lists	9	Ch 10, 17	
10	30 March	Linked list class	10	Ch 10, 17	Project 3 due; Project 4 assigned
11	6 April	Defining iterators Random access operator, dereferencing operator	11	Ch 17	Exam 2 (8 April)
	13 April	Spring Break			
12	20 April	Exception Handling	12	Ch 18	Project 4 due; Project 5 assigned
13	27 April	Inheritance, polymorphism	13	Ch 10, 17	
14	4 May	MFC (Microsoft Foundation Classes)	14	Ch 14	
15	11 May	Inheritance, polymorphism	15	Ch 14	Project 5 due
	18 May	Final Exam – Monday – 18 May – 2:00 – 3:50 pm		Ch 1-19	Final Exam

## Policy on Collaboration

You are encouraged to discuss course material with other students. Don't be shy about consulting with anyone, but please understand that you, and only you, bear the responsibility for solving the problems associated with producing a successful project or solving a lab assignment. Please read the CS Department policy on plagiarism and keep the following in mind.

All material turned in for credit must be your own work (team assignments are an exception to this). You may discuss ideas and approaches but you should work out all details and write up all solutions on your own. Copying part or all of another student's assignment, with or without the student's knowledge, is prohibited. Similarly, copying old or published solutions is prohibited.

Receive help with care. Avoid working too closely with another student. Otherwise, you can unwittingly become dependent on that student's help and fool yourself into thinking that you understand things better than you really do. Always attempt to do as much as you can on your own. Then, after you do seek help, be sure to work through similar problems on your own. Also, don't forget other sources of programming help such as the your textbook, <http://www.cplusplus.com>, the debugger, and CodeWarrior and Visual C++ documentation.

Give help with care. Don't help too much. When you understand something, you may be tempted to show someone the complete solution. However, if you do this, you will rob them of the learning experience of reaching the solution on their own. Try giving a hint that will help them get "unstuck" Although you are allowed to help other students, you are never under any obligation to do so.

Violations of these restrictions carry severe penalties. Remember that you are ultimately (i.e., during an exam or quiz) responsible for understanding the material.

## Policy on Incomplete Grade

It is the policy of the Computer Science Department that a grade of Incomplete (I) shall be assigned only when the instructor concludes that a clearly identifiable portion of course requirements cannot be met within the academic term for unforeseen, but fully justified, reasons; and that there is still a possibility of earning credit.

An incomplete shall NOT be assigned when:

- the request is made before the thirteenth week of instruction
- it is necessary for the student to attend a major portion of the class when it is next offered (i.e., if a student needs to repeat a class, an incomplete should not be given)
- the student is not passing the course with a C- or better at the time of the request
- the student is unable to keep up with course work due to an excessive course load

The condition for removal of the Incomplete shall be entered on the "Request for Incomplete" form and a copy filed in the department office prior to listing an "I" on the Grade Roster. The student must retain the grades for any coursework that was due prior to the incomplete being assigned. The incomplete cannot be removed on the basis of work taken at another institution nor by re-enrolling in the course.

An incomplete must be made up within one calendar year immediately following the end of the term in which it was assigned. This limitation prevails whether or not the student maintains continuous enrollment. Failure to complete the assigned work will result in an incomplete "I" being converted to a "NC" which will affect the grade point average.

# Spring 2009 University Calendar

January 2009						
SU	MO	TU	WE	TH	FR	SA
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

- 1 Spring 2009 Registration Fees Due
- 12-16 Second Registration (fees due February 6th )
- 15 Scholarship Application deadline for Fall 2009
- **19 Martin Luther King Jr. Birthday (campus closed)**
- 20-21 No Registration or Add/Drop activity (Financial Aid Processing)
- 22 University Convocation
- 22 Faculty Development Day
- 22-23 Open Registration (no appointment needed) - for New and Continuing Students. FEES due Feb. 6th.
- 23 Orientation and Advising
- 23 Last day to cancel registration with full refund of fees
- **26 Instruction Begins**
- 26-Feb 6 Late Registration, Add/Drop continues (fees due Feb 6th )  
*Note: Contract Courses submitted by February 6th will be added to schedule by Census*

February 2009						
SU	MO	TU	WE	TH	FR	SA
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

- 6 Last day to Add or Register Late
- 6 Last day to Drop
- 6 Last day to submit Contract Course
- 7-20 Drop with a 'W' - done on-line
- 9-20 Petition to Add with \$20 Administrative Fee class (*adding classes permitted because of serious and compelling reasons only*)
- 15 Final deadline for graduation applications to be submitted for Spring 2009
- 15 Graduation Application Priority Filing date for December 2009
- **16 Presidents Day - campus closed**
- 20 Last day to add Special Studies/Independent Studies/Internships. *Contract courses submitted after February 8th will appear on the student schedule in mid March.*
- 20 Last day to petition to add
- 20 Last day to change grade mode - *done on-line*
- 20 Last day to drop with a 'W' - *done on-line*
- 23 Census date
- 23-March 13 REGISTRATION FREEZE - *no processing*

- 23-April 17 Petition to late withdraw from a class with \$20 administrative fee begins  
(*dropping classes permitted because of serious and compelling reasons only*)

March 2009						
SU	MO	TU	WE	TH	FR	SA
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

- 31 ERD due for First Registration eligibility Fall 2009
- **31 Cesar Chavez Birthday (campus closed)**

April 2009						
SU	MO	TU	WE	TH	FR	SA
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

- 1 Final deadline for graduation applications to be submitted for August 2009
- 1 - June 2 Summer 2009 Registration\*
- 2 Last day to withdraw and receive pro-rated cancellation of fees
- **13 - 17 Spring Break (no classes, campus open)**
- 17 Last day to withdraw (no refund)
- 18-May 15 Withdrawing from the semester or University at this point follows same requirements as retroactive withdrawal (*requires Petition form and documentation*)
- 27 - May 1 Registration for Fall 2009 - by appointment only

May 2009						
SU	MO	TU	WE	TH	FR	SA
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

- 15 Last day of instruction
- 18-22 Final Exams
- 23 Commencement

# Laboratory Assignment and Project Submission Instructions

All laboratory assignments and projects will be submitted electronically using the linux platform. Naming standards are important; if you do not follow the standards, I will not receive your submission and you will not receive credit for the submission.

## Submitting Laboratory Assignments

1. Make a copy of the program to put in the drop box. For example:  
`cp Lab1.cpp yourlastnameL1.cpp`
2. Copy your new copy to the drop box. For example:  
`cp yourlastnameL1.cpp ~tiawatts/cs215drop/.`

*Note the period at the end of each these statements.... it is important!*

3. Wait about 2 minutes and then check the web page to verify that your file has reached the drop box:  
<http://www.cs.sonoma.edu/~tiawatts/cs215/lab1sub.txt>

## Submitting projects

1. Make a copy of the program to put in the drop box. For example:  
`cp Project1.cpp yourlastnameP1.cpp`
2. Copy your new copy of the assignment to the drop box. For example:  
`cp yourlastnameP1.cpp ~tiawatts/cs215drop/.`

*Note the period at the end of this statement.... it is important!*

3. Wait about 2 minutes and then check the web page to verify that your file has reached the drop box:  
<http://www.cs.sonoma.edu/~tiawatts/cs215/proj1sub.txt>



# Laboratory Exercise 1 (C++ Review)

## Topics: C++ Review

- Compiling and executing C++ programs on a Linux platform
- C++ strings
- file input
- console display output
- loops
- decision statements
- simple arithmetic
- cin.fail() condition test
- using a debugger

## Goals: Upon successful completion of this lab you should be able to:

1. Start a linux session
2. Enter and edit a C++ program
3. Compile and run a C++ program
4. Rename and submit a C++ program to the drop box
5. Complete a linux session
6. Write a C++ input statement
7. Write a C++ output statement
8. Predict when the C++ cin.fail () test will return 1
9. Use the gdb debugger

## Related text sections:

- Chapter 2 (C++ program format)
- emacs reference
- linux reference

## Laboratory Exercise 1 Instructions

### Part 1

1. Boot the computer and select the Linux operating system.
2. Logon as “student”
3. Start a console session.
4. Use ssh to remotely connect to the mainframe:  
ssh [username@ewolf.cs.sonoma.edu](mailto:username@ewolf.cs.sonoma.edu)

**Note: steps 1 through 4 can also be done using the Windows operating system and TeraTerm SSH or Putty.**

5. You may need to change your password:  
passwd
6. Create and switch to a CS 215 subdirectory and a Lab1 directory within your CS 215 directory:  
mkdir cs215  
cd cs215  
mkdir Lab1  
cd Lab1
7. Using emacs, vi, or another text editor, enter the following C++ program (call the file Lab1a.cpp):

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

int main ()
{
    int v;
    cout << "Enter the data set input: ";
    cin >> v;
    cout << v << (cin.fail() ? " true" : " false") << endl;
    cin >> v;
    cout << v << (cin.fail() ? " true" : " false") << endl;
    cin >> v;
    cout << v << (cin.fail() ? " true" : " false") << endl;
    return 0;
}
```

8. Compile the program using the g++ compiler (call the executable lab1a):  
g++ Lab1a.cpp -o lab1a

9. Execute the program several times:  
./lab1a

Use each of the data sets in the following table as input. Each time the program asks you to enter the data set, enter all of the characters in the data set the first time the program pauses for input. Record the output.

9. (continued)

Case	Input data	Observed Output
1.	5 12 15	
2.	5 -12 15	
3.	5 a 12	
4.	5.13 -12 15	
5.	abc 5 12	

10. Write 1 to 3 sentences describing your observations of the rules used by the input operator (>>) when it is used to read an integer value.

11. Modify the program by changing the type of v from int to float.

12. Recompile and run the program with each of the data sets in the following table (Enter all of the input in the data set the first time the program pauses for input). Record the output.

Case	Input data	Observed Output
1.	5.1 12.05 15.23	
2.	5.1 -12.05 15.23	
3.	5 12 15.23	
4.	5.13 a 15	
5.	abc 5 12	

13. Write 1 to 3 sentences describing your observations of the rules used by the input operator (>>) when it is used to read a float value.

14. Modify the program by changing the type of v from float to char.

15. Recompile and run the program with each of the data sets in the following table (Enter all of the input in the data set the first time the program pauses for input). Record the output.

Case	Input data	Observed Output
1.	a b c	
2.	a b c	
3.	abc	
4.	5 a 12.5	
5.	abc 5 12	

16. Write 1 to 3 sentences describing your observations of the rules used by the input operator(>>) when it is used to read a character value.

17. Modify the program by changing the type of v from char to string. Note: you will also need to add another include directive: `#include <string>`

18. Recompile and run the program with each of the data sets in the following table (Enter all of the input in the data set the first time the program pauses for input). Record the output.

Case	Input data	Observed Output
1.	abc def ghi	
2.	abc def ghi	
3.	a b c d e f g h	
4.	5.13 -12 15	
5.	abc 5 12	

19. Write 1 to 3 sentences describing your observations of the rules used by the input operator(>>) when it is used to read a C++ string value.

20. Using a text editor, create a document called Lab1a.txt that describes the rules for reading integers, floats, characters, and C++ strings from an input stream. Each of your rules should describe exactly what will be read from the input stream and when the stream will fail. After you have completed your rules, use the following exercises to verify that they are correct.

21. In the boxes below, predict the output that will be generated by this program for each of the sets of keyboard input data. *Do not enter or run the program!*

```
#include <iostream>
#include <string>

using namespace std;
int main ()
{
    int i;
    float f;
    char c;
    string s;
    bool failflag;
    cout << "Enter line of data: ";
    cin >> i;
    failflag = cin.fail();
    // Set breakpoint 1 here
    while (i != 0)
    {
        cin >> f >> c >> s;
        failflag = cin.fail();
        // Set breakpoint 2 here
        cout << "The output is: ";
        cout << i << ' ' << f << ' ';
        cout << c << ' ' << s << endl;
        cout << "Enter line of data: ";
        cin >> i;
        failflag = cin.fail();
        // Set breakpoint 3 here
        cout << "At end of loop\n";
    }
    return 0;
}
```

Data Set 1 keyboard input

```
5 3.14 a bcd
7.13 4 abcd
0
```



predicted output

Data Set 2 keyboard input

```
5 3.14 a bcd
7.14 4 a bcd
3.14 4 a bcd
0
```



predicted output

22. Enter and run the program with each of the data sets. (Call the program Lab1b.cpp and the executable lab1b.)

Data Set 1 keyboard input

```
5 3.14 a bcd
7.14 4 abcd
0
```



actual program output

Data Set 2 keyboard input

```
5 3.14 a bcd
7.13 4 a bcd
3.14 4 a bcd
0
```



actual program output

23. Recompile your program so that it can run within the debugger:

```
g++ -g Lab1b.cpp -o lab1b
```

24. Using the debugger, look at the values in each variable after each failflag assignment statement. Record the values in the following table.

```
gdb lab1b
```

You will need to use the following gdb commands:

```
list
break <line number>
run
display <variable>
continue
quit
```

**Data Set 1**

Breakpoint	I	f	c	s	failflag
1					
2					
3					
1					
2					
3					
1					
2					
3					

## Data Set 2

Breakpoint	i	f	c	s	failflag
1					
2					
3					
1					
2					
3					
1					
2					
3					
1					
2					
3					

The debugging session for data set 1 is listed at the end of this lab exercise.

25. After verifying that your rules are correct, submit your Lab1a.txt file as yourlastnameL1.txt.

```
cp Lab1a.txt yourlastnameL1.txt
cp yourlastnameL1.txt ~tiawatts/cs215drop/.
```

## Part 2

1. Write a program called Lab1c.cpp based on the following specifications:
  - A. Your program should open, for input, a text file called “words.txt”. This file will contain several lines of text.
  - B. Your program should count the number of words in the file. A word is defined as a string of characters delimited by white space.
  - C. Your program should count the number of vowels in the file. Only A(a), E(e), I(I), O(o), and U(u) should be counted as vowels by your program.
  - D. Your program should count the number of consonants in the file.
  - E. Your program should count the number of digits in the file.
  - F. Your program should count the number of special characters in the file. Any non-white-space character that is not included in the counts described in statements C, D, or E should be counted as special characters by your program.
  - G. Your program should print out a summary of the counts.

## H. Sample Input and Output:

words.txt

```
This file contains 15 words.  
It has lots of letters and  
very few special  
characters!
```

console output

```
Words: 15  
Vowels: 22  
Consonants: 44  
Digits: 2  
Special characters: 2
```

2. Compile your program using the g++ compiler; create an executable call lab1c:
3. Using a text editor, create a file called words.txt for testing your program; you can use the sample data shown above.
4. Run the executable:  

```
./lab1c
```
5. If your program does not execute correctly, modify it, recompile it and execute it again.
6. When you are sure that your program is executing correctly, make a copy of the program to put in the drop box and move your copy to the drop box:  

```
cp Lab1c.cpp yourlastnameL1.cpp  
cp yourlastnameL1.cpp ~tiawatts/cs215drop/.
```
7. Wait about 2 minutes and then check the web page to verify that you file has reached the drop box:  
<http://www.cs.sonoma.edu/~tiawatts/cs215/lab1sub.txt>

## Debugging session for data set 1

```
$g++ -g Lab1b.cpp -o lab1b
gdb lab1b
GNU gdb Red Hat Linux (6.3.0.0-1.143.el4rh)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...Using host libthread_db
library "/lib/tls/libthread_db.so.1".
```

```
(gdb) list
1      #include <iostream>
2      #include <string>
3
4      using namespace std;
5      int main ()
6      {
7          int i;
8          float f;
9          char c;
10         string s;
(gdb) list
11         bool failflag;
12         cout << "Enter line of data: ";
13         cin >> i;
14         failflag = cin.fail();
15         // Set breakpoint 1 here
16         while (i != 0)
17         {
18             cin >> f >> c >> s;
19             failflag = cin.fail();
20             // Set breakpoint 2 here
(gdb) list
21             cout << "The output is: ";
22             cout << i << ' ' << f << ' ';
23             cout << c << ' ' << s << endl;
24             cout << "Enter line of data: ";
25             cin >> i;
26             failflag = cin.fail();
27             // Set breakpoint 3 here
28             cout << "At end of loop\n";
29         }
30         return 0;
(gdb) break 15
Breakpoint 1 at 0x8048c28: file Lab1b.cpp, line 15.
(gdb) break 20
Breakpoint 2 at 0x8048c79: file Lab1b.cpp, line 20.
(gdb) break 27
Breakpoint 3 at 0x8048d4c: file Lab1b.cpp, line 27.
(gdb) run
Starting program: /home/tiawatts/cs215f07/lab1b
Enter line of data: 5 3.14 a bcd

Breakpoint 1, main () at Lab1b.cpp:16
16         while (i != 0)
(gdb) display i
1: i = 5
```

```

(gdb) display f
2: f = 3.9911027e-34
(gdb) display c
3: c = 0 '\0'
(gdb) display s
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x27040c ""}}
(gdb) display failflag
5: failflag = false
(gdb) c
Continuing.

```

```

Breakpoint 2, main () at Lab1b.cpp:21
21          cout << "The output is: ";
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e04c "bcd"}}
3: c = 97 'a'
2: f = 3.1400001
1: i = 5
(gdb) c
Continuing.
The output is: 5 3.14 a bcd
Enter line of data: 7.13 4 abcd

```

```

Breakpoint 3, main () at Lab1b.cpp:28
28          cout << "At end of loop\n";
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e04c "bcd"}}
3: c = 97 'a'
2: f = 3.1400001
1: i = 7
(gdb) c
Continuing.
At end of loop

```

```

Breakpoint 1, main () at Lab1b.cpp:16
16          while (i != 0)
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e04c "bcd"}}
3: c = 97 'a'
2: f = 3.1400001
1: i = 7
(gdb) c
Continuing.

```

```

Breakpoint 2, main () at Lab1b.cpp:21
21          cout << "The output is: ";
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e064 "abcd"}}
3: c = 52 '4'
2: f = 0.129999995

```

```

1: i = 7
(gdb) c
Continuing.
The output is: 7 0.13 4 abcd
Enter line of data: 0

Breakpoint 3, main () at Lab1b.cpp:28
28             cout << "At end of loop\n";
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e064 "abcd"}}}
3: c = 52 '4'
2: f = 0.129999995
1: i = 0
(gdb) c
Continuing.
At end of loop

Breakpoint 1, main () at Lab1b.cpp:16
16             while (i != 0)
5: failflag = false
4: s = {static npos = 4294967295,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No
data fields>}, <No data fields>}, _M_p = 0x905e064 "abcd"}}}
3: c = 52 '4'
2: f = 0.129999995
1: i = 0
(gdb) c
Continuing.

Program exited normally.
(gdb) quit
$

```



## Laboratory Exercise 2 (Algorithm Development)

Topics: Code analysis  
Algorithm development

Goals: Upon successful completion of this lab you should be able to:

1. Read a C++ program containing loops and decision statements
2. Read a C++ program containing functions and function calls
3. Develop a flow chart for a program containing loops and decision statements
4. Use a debugger to follow the flow of a C++ program

Related text sections:  
Chapter 2  
Chapter 3

## Laboratory Exercise 2 Instructions

### Part 1.

1. Using your favorite text editor, create a file called *yourlastname.mail*. Enter the following information into the file (replaced the italicized words with the appropriate information). Make sure there is a carriage return at the end of the line.

```
mail -s "$1" you@favorite.isp < $2 # Firstname Lastname
```

for example:

```
mail -s "$1" cbrownc@sonoma.edu < $2 # Charlie Brown
```

2. Close the file and copy it to the CS 215 dropbox:

```
cp yourlastname.mail ~tiawatts/cs215drop/.
```

### Part 2.

On the next page, there is a short program and flowcharts for each of the functions in the program.

1. Predict the output of the program when the following keyboard input is provided:

```
3 6
12 8
3 4
5 0
-2 10
1 0
0 0
```

2. Enter and execute the program to determine if your predictions are correct.
3. If your predictions were not correct, use the debugger to trace the execution of the program; in particular, set breakpoints to watch the values passed into and out of the “find” function.
4. Using a text editor, open a file for the answers to the following questions:
  - a. What does the function find actually find?
  - b. What does it do to the input values?
  - c. Describe a problem where the function find would be an important component of a program designed to solve the problem.
5. Place your answers in the dropbox ~tiawatts/cs215drop as *yourlastnameL2.txt*.

### Part 3.

1. Using SFC-v2-3.exe (available at <http://www.cs.sonoma.edu/~tiawatts/SFC>), create a flowchart for the program you submitted for lab1, part 2.
2. Modify your flowchart so that it will print the number of vowels, consonants, digits and special characters in each word as well as the total number of vowels, consonants, digits and special characters in the complete file.

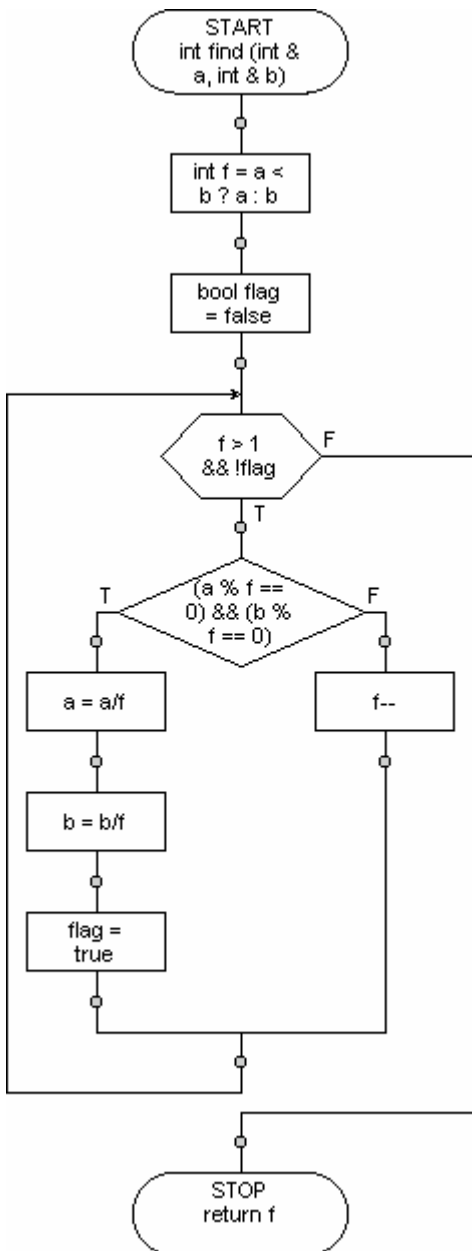
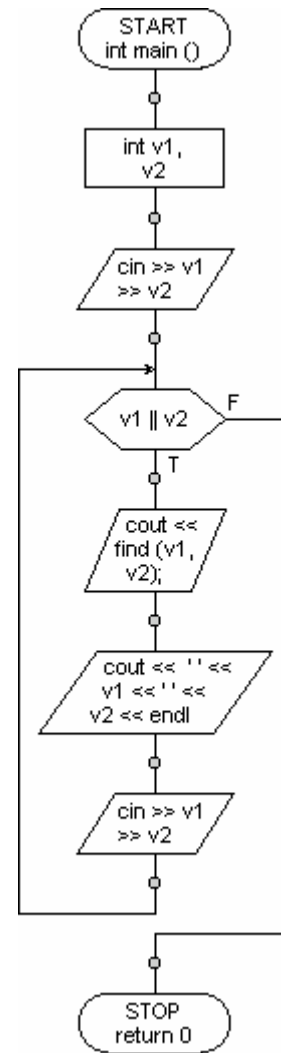
```

#include <iostream>
using namespace std;

int find (int & a, int & b);

int main ()
{
    int v1, v2;
    cin >> v1 >> v2;
    while (v1 || v2)
    {
        cout << find (v1, v2);
        cout << ' ' << v1 << ' ' << v2 << endl;
        cin >> v1 >> v2;
    }
    return 0;
}

```



```

int find (int & a, int & b)
{
    int f = a < b ? a : b;
    bool flag = false;
    while (f > 1 && !flag)
        if ((a % f == 0) && (b % f == 0))
        {
            a = a/f;
            b = b/f;
            flag = true;
        }
        else
            f--;
    return f;
}

```

#### Part 4.

1. Modify the program you submitted for lab 1 to reflect the changes you made in part 2. Sample input data and results are illustrated below.
2. Format the output of your program as illustrated below. All numeric values should be right justified in 8 spaces.
3. Place your completed program in the dropbox ~tiawatts/cs215drop as *yourlastnameL2.cpp*.
4. Place your completed flowchart in the dropbox ~tiawatts/cs215drop as *yourlastnameL2.sfc*.
5. Verify that all 3 of your files (.txt, .cpp, and .sfc) are in the dropbox.

Sample Input

```
This file contains
15 words. It has
lots of letters
and very few
special
characters!
```

Sample Output

Word	Vowels	Const.	Digits	Special
1	1	3	0	0
2	2	2	0	0
3	3	5	0	0
4	0	0	2	0
5	1	4	0	1
6	1	1	0	0
7	1	2	0	0
8	1	3	0	0
9	1	1	0	0
10	2	5	0	0
11	1	2	0	0
12	1	3	0	0
13	1	2	0	0
14	3	4	0	0
15	3	7	0	1
Totals	22	44	2	2

## Laboratory Exercise 3 (C++ structs)

Topics: C++ structs

C++ arrays

C strings

Sorting

Goals: Upon successful completion of this lab you should be able to:

1. Define and use a struct in a C++ program
2. Define and use a static array in a C++ program
3. Define and use C strings in a C++ programs
4. Sort the entries in an array

Related text sections:

Chapter 5

Chapter 6

Chapter 9

## Laboratory Exercise 3 Instructions

### Part 1.

1. Add the following structure definition to the program you submitted for lab 2.

```
struct worddata
{
    string word;
    int vowels;
    int consonants;
    int digits;
    int specialchars;
};
```
2. Modify your program so that it reads the words from the file “words.txt” into an array of worddata entries. There will be no more than 100 words in the file.
3. Modify your program so that it stores the number of vowels, consonants, digits, and special characters for each word in the worddata entry for the word.
4. Modify your program so that it stores the total number of vowels, consonants, digits, and special characters in a separate worddata variable.

### Part 2.

1. Create a flow chart for a function called wordsort that will sort an array of worddata entries into ASCII lexicographical order. The function should implement the *selection* sort (as discussed in lab.) The parameters for wordsort are an array of worddata entries and a count of the number of used entries in the array. The return type is void. The entries should be sorted into ascending word order.
2. Add your wordsort function to your program. You will also need to a worddata Swap function to your program.
3. Modify the output of your program so that it prints the word instead of the word number. Words should be left justified in 12 spaces. (Numeric values should remain right justified in 8 spaces.)
4. Test your program to make sure that it reads the words from the file, calculates the appropriate values, and prints the words and their corresponding values in sorted order. Sample data is illustrated on the next page.
6. Place your completed program in the dropbox ~tiawatts/cs215drop as *yourlastnameL3.cpp*.
7. Place your completed flowchart in the dropbox ~tiawatts/cs215drop as *yourlastnameL3.sfc*.
8. Verify that both of your files (.cpp and .sfc) are in the dropbox.

Sample Input

This file contains  
15 words. It has  
lots of letters  
and very few  
special  
characters!



Sample Output

Word	Vowels	Const.	Digits	Special
15	0	0	2	0
It	1	1	0	0
This	1	3	0	0
and	1	2	0	0
characters!	3	7	0	1
contains	3	5	0	0
few	1	2	0	0
file	2	2	0	0
has	1	2	0	0
letters	2	5	0	0
lots	1	3	0	0
of	1	1	0	0
special	3	4	0	0
very	1	3	0	0
words.	1	4	0	1
	---	---	---	---
Totals	22	44	2	2



## Laboratory Exercise 4 (C++ classes)

Topics: C++ classes

Class constructors

Class destructor

Class accessor functions

Class mutator functions

Goals: Upon successful completion of this lab you should be able to:

1. Write a description of a simple C++ class
2. Implement the member functions of a C++ class
3. Describe the purpose of a class constructor
4. Describe the purpose of a class destructor

Related text sections:

Chapter 6

## Laboratory Exercise 4 Instructions

1. Copy the file Lab4.cpp from the pickup folder:  
cp ~tiawatts/cs215pickup/Lab4.cpp .

```
// Title: Lab 4 Driver program
// Author: Dr. Watts
// Description: This program is designed to test the WordData class.

#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include "Lab4.h"

using namespace std;

const int MAX = 100;

int main ()
{
    // Open file for input
    ifstream input ("words.txt");
    string inword;
    WordData words[MAX];
    // Initialize word counter
    int count = 0;
    // Read until array is filled or end of file is found
    while (count < MAX && input >> inword)
    {
        // Insert word into array
        words[count].SetWord(inword);
        // Increment counter
        count ++;
    }
    // Close input file
    input.close();
    // Print report header
    cout << setw (12) << left << "Word";
    cout << setw (8) << right << "Vowels";
    cout << setw (8) << right << "Const.";
    cout << setw (8) << right << "Digits";
    cout << setw (8) << right << "Special";
    cout << endl;
    // Loop through all words in array
    for (int i = 0; i < count; i++)
    {
        // Print data for word
        words[i].WriteData(cout);
        cout << endl;
    }
    return 0;
}
```

- Using a text editor, enter the following preprocessor directives and class description into a file called Lab4.h:

```
#ifndef WORDDATA
#define WORDDATA

#include <cstring>
#include <string>
#include <iostream>
#include <iomanip>

using namespace std;

class WordData
{
public:
    WordData ();
    WordData (const WordData & OtherWord);
    ~WordData ();
    void SetWord (const string & InWord);
    string GetWord () const;
    void WriteData (ostream & outs) const;
private:
    char * word;           // C-string to hold the word
    int vowels;           // vowel counter
    int consonants;       // consonant counter
    int digits;           // digit counter
    int specialchars;     // special character counter
};

// Function implementations will be placed here

#endif
```

- Try to compile the file Lab4.cpp.  
`g++ Lab4.cpp -o lab4`  
The linker should return errors because only the function prototypes have been declared; the functions must still be implemented.
- Implement the default constructor: `WordData ()`;  
Specifications:
  - Will set the word to `new char [1]`; with `word[0] = '\0'`;
  - Will set the counters to 0.
- Implement the copy constructor:  
`WordData (WordData & OtherWord)`;  
Specifications:
  - Will copy the word from `OtherWord`
  - Will copy the counters from `OtherWord`

6. Implement the destructor: `~WordData ()`;

Specifications:

- a. Will release the memory used to store word

7. Implement the mutator function:

```
void SetWord (string InWord);
```

Specifications:

- a. Will allocate space to store the C string associated with InWord
- b. Will count the vowels in InWord (a, e, i, o, or u)
- c. Will count the consonants in InWord (letters that are not vowels)
- d. Will count the digits in InWord (0, 1, 2, 3, 4, 5, 6, 7, 8, or 9)
- e. Will count the “special characters” in InWord (characters that do not fall in any of the other categories)

8. Implement **and test** the accessor function: `string GetWord ()`;

Specifications:

- a. Will return the C string word as a C++ string

9. Implement the output function:

```
void WriteData (ostream & outs);
```

Specifications:

- a. Will write the value of word, left justified in 12 spaces.
- b. Will write the counter values, each right justified in 8 spaces.
- c. No new line at the end.

10. Compile your program:

```
g++ Lab4.cpp -o lab4
```

11. Execute your compiled program. The output should be:

Word	Vowels	Const.	Digits	Special
This	1	3	0	0
file	2	2	0	0
contains	3	5	0	0
15	0	0	2	0
words.	1	4	0	1
It	1	1	0	0
has	1	2	0	0
lots	1	3	0	0
of	1	1	0	0
letters	2	5	0	0
and	1	2	0	0
very	1	3	0	0
few	1	2	0	0
special	3	4	0	0
characters!	3	7	0	1

12. When your program is executing correctly, drop your completed header file in the `~tiawatts/cs215drop` folder as *yourlastnameL4.h*.

## Laboratory Exercise 5 (The STL)

Topics: Class accessor functions  
STL (Standard Template Library) vector containers

Goals: Upon successful completion of this lab you should be able to:

1. Declare and implement accessor functions
2. Use accessor functions in an application program
3. Use a vector container in an application program

Related text sections:

Chapter 6  
Chapter 7

## Laboratory Exercise 5 Instructions

For this lab you will be modifying the class you added to the character counting program in lab 4.

1. Copy the header file Lab4.h to a file called Lab5.h.
2. Add member functions the WordData class in Lab5.h to access the counter values in the WordData class:

```
int GetNumVowels () const;
int GetNumConsonants () const;
int GetNumDigits () const;
int GetNumSpecialChars () const;
```
3. Implement the accessor function: `int GetNumVowels () const;`  
Specifications:
  - a. Will return the integer value vowels
4. Implement the accessor function: `int GetNumConsonants () const;`  
Specifications:
  - a. Will return the integer value consonants
5. Implement the accessor function: `int GetNumDigits () const;`  
Specifications:
  - a. Will return the integer value digits
6. Implement the accessor function: `int GetNumSpecialChars () const;`  
Specifications:
  - a. Will return the integer value specialchars
7. Copy the application program Lab4.cpp to a file called Lab5a.cpp. Modify the application program Lab5a.cpp to:
  - a. Include Lab5.h instead of Lab4.h
  - b. Print a line of dashes after the list of individual words.
  - c. Print a totals line with sums of the individual counts at the end of the list of individual words.

Word	Vowels	Const.	Digits	Special
This	1	3	0	0
file	2	2	0	0
contains	3	5	0	0
15	0	0	2	0
words.	1	4	0	1
It	1	1	0	0
has	1	2	0	0
lots	1	3	0	0
of	1	1	0	0
letters	2	5	0	0
and	1	2	0	0
very	1	3	0	0
few	1	2	0	0
special	3	4	0	0
characters!	3	7	0	1
-----				
Total	22	44	2	2

8. Drop your modified application program in the ~tiawatts/cs215drop folder as *yourlastnameL5.cpp*.
9. Drop your modified header file in the ~tiawatts/cs215drop folder as *yourlastnameL5.h*.
10. Enter and compile the following C++ program (call the source file Lab5b.cpp and the executable lab5b).

```

// Title: Lab 5 B Driver program
// Author: Dr. Watts
// Description: This program is designed to test the WordData class with a
// vector container

#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <vector>
#include "Lab5.h"

using namespace std;

int main ()
{
    // Open file for input
    ifstream input ("words.txt");
    string inword;
    // create a vector to hold a collection of WordData objects
    vector <WordData> words;
    // Initialize word counter
    // Read until array filled or end of file
    while (input >> inword)
    {
        // Insert word into a WordData object
        WordData oneword;
        oneword.SetWord(inword);
        // Push oneword onto the end of the words vector
        words.push_back (oneword);
    }
    // Close input file
    input.close();
    // Print report header
    cout << setw (12) << left << "Word";
    cout << setw (8) << right << "Vowels";
    cout << setw (8) << right << "Const.";
    cout << setw (8) << right << "Digits";
    cout << setw (8) << right << "Special";
    cout << endl;
    // Loop through all words in array
    for (int i = 0; i < words.size(); i++)
    {
        // Print data for word
        words[i].WriteData(cout);
        cout << endl;
    }
    return 0;
}

```

11. Execute lab5b; the output from lab5b should be the same as the output from lab 5a before you added the Totals line.
12. Add the totals line facility to Lab5b.cpp.

13. Modify Lab5a.cpp and Lab5b.cpp to read data from a file called “manywords.txt”.
14. Recompile each of the programs.
15. Copy the file manywords.txt from the cs215pickup folder to your folder.
16. Execute lab5a and lab5b. They should not produce identical outputs – how do they differ?  
Why do they differ?
17. Enter your answers to the previous questions into a file called Lab5.txt.
18. Drop your text file in the ~tiawatts/cs215drop folder as *yourlastname*L5.txt and print a copy to bring to class.

## Laboratory Exercise 6 (Operator Overloading)

Topics: overloaded operators in a class definition

Goals: Upon successful completion of this lab you should be able to:

- Declare and implement class operators

- Use class operators in an application program

- Sort an array of class objects

- Describe the relationship between the constructor, copy constructor, destructor, and assignment operator for a class

Related text sections:

- Chapter 7

- Chapter 8

## Laboratory Exercise 6 Instructions

1. Copy your header file for lab 5 to a file called Lab6.h.
2. Remove the prototype and implementation for “WriteData” from Lab6.h.
3. Add the following prototypes to the public section of your WordData class description:

```
WordData & operator = (const WordData & OtherWord);  
bool operator == (const WordData & OtherWord) const;  
WordData operator + (const WordData & OtherWord) const;  
friend istream & operator >> (istream & ins, WordData & w);  
friend ostream & operator << (ostream & outs, const WordData &  
w);
```
4. Implement the assignment operator for the WordData class:

```
WordData & operator = (const WordData & OtherWord);
```

Specifications:
  - a. Will check for self assignment.
  - b. Will release dynamically allocated space.
  - c. Will dynamically allocate new space.
  - d. Will copy values from OtherWord to this WordData object.
  - e. Will return this.
5. Implement the equality operator for the WordData class:

```
bool operator == (const WordData & OtherWord) const;
```

Specifications:
  - a. Will check for self testing.
  - b. Will return true if this WordData object is identical to OtherWord.
  - c. Will return false if this WordData object is not identical to OtherWord
6. Implement the addition operator for the WordData class:

```
WordData operator + (const WordData & OtherWord) const;
```

Specifications:
  - a. Will create a new WordData object.
  - b. Will concatenate OtherWord to the end of this WordData object.
  - c. Will add count values
  - d. Will return new WordData object.
7. Implement the input operator for the WordData class:

```
friend istream & operator >> (istream & ins, WordData & w);
```

Specifications:
  - a. Will read in a new word from the input stream (ins).
  - b. Will return if the input stream (ins) fails.
  - c. Will store the word read from ins into w.
  - d. Will calculate the counts.
  - e. Will return ins.
8. Implement the output operator for the WordData class:

```
friend ostream & operator << (ostream & outs, const WordData &  
w);
```

Specifications:

- a. Will write the value of word and the value of each of the counts on a single line. The word should be left justified in 12 spaces and the counts should each be right justified in 8 spaces. Should not write a new line.
  - b. Will return outs.
9. Copy your program file Lab5b.cpp to a file called Lab6a.cpp.
10. Remove all references to WriteData and SetWord from your Lab6a program.
11. Modify your Lab6a program to test all of the new methods you added to the class.
12. When you are satisfied that your class is working correctly, drop your modified header file in the `~tiawatts/cs215drop` folder as *yourlastnameL6.h*.
13. Ask me to test your class.



## Laboratory Exercise 7 (Makefiles and Templates)

Topics: dynamic arrays

makefiles

Template functions

Sorting

SLT vector class and iterators

Goals: Upon successful completion of this lab you should be able to:

1. Resize a dynamic array of objects
2. Create a simple makefile and use the make utility
3. Write a C++ program that uses an STL vector.

Related text sections:

Chapter 11

Chapter 19

## Laboratory Exercise 7 Instructions

### Part 1.

1. In your CS 215 directory, create a new subdirectory called Lab7.1.
2. Divide the header file you submitted for Lab 6 (Lab6.h) into 2 files: worddata.h and worddata.cpp, in directory Lab7.1.
  - a. your worddata.h file should contain your WordData class description and appropriate macroguards.
  - b. your worddata.cpp file should contain an “include” for your worddata.h file and should contain implementations for all of the friend and member functions you have created for the class.
3. Create a copy of your words.txt test file from lab 6 as Lab7.txt in directory Lab7.1.
4. Enter the following make commands into a new file called makefile in directory Lab7.1.

```
lab7 : application.o worddata.o
      g++ -o lab7 application.o worddata.o

application.o : application.cpp sorts.h worddata.h
      g++ -c application.cpp

worddata.o : worddata.cpp worddata.h
      g++ -c worddata.cpp

clean :
      rm -f core.* *.o lab7
```

5. The dependencies in this makefile indicate that 6 files are need to compile the executable lab7: application.o, worddata.o, application.cpp, sorts.h, worddata.h, amd worddata.cpp. The .o files will be created by the make utility. You have already created worddata.h and worddata.cpp. The missing files are application.cpp and sorts.h.
6. Enter the following template swap and sorts into a new file called sorts.h in directory Lab7.1.

```
#ifndef SORT_TMP
#define SORT_TMP

// Description : This file contains template implementations of 3
// simple sorts (exchange, insertion, and selection).

template <class T>
void Swap (T & a, T & b)
{
    T t = a;
    a = b;
    b = t;
}
```

```

template <class T>
void eSort (T * A, size_t N)
{ // implemetation of exchange (bubble) sort for array (A) of N entries
  // of type T
    bool swapped = true;
    for (int i = 0; swapped && i < N; i++)
    {
        swapped = false;
        for (int j = 1; j < N-i; j++)
            if (A[j-1] > A[j])
            {
                Swap (A[j], A[j-1]);
                swapped = true;
            }
    }
}

template <class T>
void iSort (T * A, size_t N)
{ // implemetation of insertion sort for array (A) of N entries
  // of type T
    for (int i = 1; i < N; i++)
        for (int j = i; j > 0 && A[j] > A[j-1]; j--)
            Swap (A[j], A[j-1]);
}

template <class T>
void sSort (T * A, size_t N)
{ // implemetation of selection sort for array (A) of N entries
  // of type T
    for (int i = 0; i < N; i++)
    {
        int p = i;
        for (int j = i+1; j < N; j++)
            if (A[p] > A[j])
                p = j;
        Swap (A[i], A[p]);
    }
}
#endif

```

7. Create an application program (called application.cpp) that
  - a. Uses a dynamic array of WordData entries. The initial size of the array should be 20 entries.
  - b. Reads words from the file Lab7.txt into the array of WordData entries using your overloaded input (>>) operator.
  - c. If the array size has been exceeded, increases the size of the array by 10 entries.
  - d. Sorts the entries in your array into ascending order using the eSort or sSort function found in sorts.h.
  - e. Prints a list of the sorted entries using your overloaded output (<<) function. The listing should be preceded by a heading line which identifies the columns and followed by a line of 2 blank lines.
  - f. Sorts the entries in your array into descending order using the iSort function found in sorts.h.

- g. Prints a list of the sorted entries using your overloaded output (<<) function. The listing should be preceded by a heading line which identifies the columns and followed by 2 blank lines.
  - h. Resorts the entries in your array into ascending order using the `ssort` function found in `sorts.h`.
  - i. Prints a list of the sorted entries using your overloaded output (<<) function. The listing should be preceded by a heading line which identifies the columns and followed by 2 blank lines.
8. Using the `make` utility, try to compile your program. You should simply enter “`make`” at the Linux prompt.
  9. Your program probably did not compile. If you carefully parse the error messages, you will see the statement “no match for operator `>`”; the class is missing a greater-than (`>`) operator. Add a greater-than operator prototype to the `WordData` class description in `worddata.h` and the implementation of the operator to your `worddata.cpp` file.
    - a. The prototype should be similar to the prototype for the `==` operator.
    - b. The implementation should use the `strcmp` function to compare the words in the implicitly and explicitly passed objects. If the word in `*this` strictly follows the other word, the operator should return `true`; otherwise it should return `false`.
  10. Use `make` to compile your files and create the executable “`lab7`”.
  11. Execute the program. You should see 2 lists of word data elements, the first sorted in ascending order, the second in descending order.
  12. Create a copy of your `manywords.txt` test file from lab 6 as `Lab7.txt` in directory `Lab7.1`.
  13. Execute the program. Again, you should see 2 lists of word data elements, the first sorted in ascending order, the second in descending order. These lists should be much longer than the lists from the previous execution.
  14. When it is executing correctly, drop a copy of your application program into the CS 215 drop box as **`yourlastnameL7.app`**.

## Part 2.

1. In your CS 215 directory, create a new subdirectory called `Lab7.2`.
2. Copy `worddata.h`, `worddata.cpp`, and `makefile` from `Lab7.1` to `Lab7.2`.
3. Remove all references to `sorts.h` from your new `makefile`:

```
lab7 : application.o worddata.o
      g++ -o lab7 application.o worddata.o

application.o : application.cpp worddata.h
      g++ -c application.cpp

worddata.o : worddata.cpp worddata.h
      g++ -c worddata.cpp

clean :
      rm -f core.* *.o lab7
```

4. Enter the following program as application.cpp:

```
// Lab 7 Part 2 application program
// Author: Dr. Watts
// Data: Spring 2009
// Description: This program is designed to test the WordData class when
//              it is used in a vector. The vector will also be sorted
//              using the sort in the algorithm library.

#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include "worddata.h"

using namespace std;

int main (int argc, char * argv [])
{
    // Test for existence of file name argument
    if (argc < 2)
    {
        cerr << "Usage: " << argv[0] << " <file-name>\n";
        exit (1);
    }

    // Test to see if file was opened
    ifstream ins (argv[1]);
    if (ins.fail ())
    {
        cerr << "No file " << argv[1] << " found\n";
        exit (2);
    }
    vector<WordData> words;
    WordData one;

    // Read words from file into WordData objects and push them onto the end
    // of the vector
    while (ins >> one)
        words.push_back (one);
    ins.close();

    // Sort the WordData object vector into ascending order
    sort (words.begin(), words.end());

    // Print the vector of Word Data objects using an index
    for (int i = 0; i < words.size(); i++)
        cout << words[i] << endl;
    cout << endl << endl;

    // Sort the WordData object vector into descending order
    sort (words.rbegin(), words.rend());

    // Print the vector of Word Data objects using an iterator
    for (vector<WordData>::iterator itr = words.begin();
         itr != words.end(); itr++)
        cout << *itr << endl;
    cout << endl << endl;

    return 0;
}
```

5. Using the make utility, try to compile your program. You should simply enter “make” at the Linux prompt.
6. Your program probably did not compile. If you carefully parse the error messages, you will see the statement “no match for operator <”; the class needs a less-than (<) operator for the algorithm sort function to compile. Add a less-than operator prototype to the WordData class description in worddata.h and the implementation of the operator to your worddata.cpp file.
  - a. The prototype should be similar to the prototype for the == operator.
  - b. The implementation should use the strcmp function to compare the words in the implicit and explicitly passed objects. If the word in \*this strictly follows the other word, the operator should return true; otherwise it should return false.
7. Try executing your program without designating an input file:  
lab7  
The program should produce an error message and exit.
8. Try executing your program with a non-existent input file:  
lab7 words.txt  
The program should produce an error message and exit.
9. Copy the words.txt and manywords.txt test files from lab 6 to Lab7.2.
10. Execute the program using words.txt as the input file:  
lab7 words.txt  
Again, you should see 2 lists of word data elements, the first sorted in ascending order, the second in descending order.
11. Execute the program using manywords.txt as the input file:  
lab7 manywords.txt  
The lists produced by this execution of lab7 should be much longer than the lists from the previous execution.
12. When the program is executing correctly, drop a copy of your worddata header and implementation files into the CS 215 drop box as **yourlastnameL7.h** and **yourlastnameL7.cpp** respectively.

## Laboratory Exercise 8 (Dynamic memory allocation)

Topics: Dynamic Array Lists  
Singly Linked Lists

Goals: Upon successful completion of this lab you should be able to:

1. Maintain a list class in a dynamic array
2. Insert items into the beginning or end of a list
3. Remove items from the beginning or end of a list
4. Resize a list
5. Write an application program that uses a dynamic array list class
6. Initialize a singly linked list class (of integers)
7. Traverse a singly linked list
8. Insert a node at the front of a singly linked list
9. Insert a node at the end of a singly linked list
10. Remove a node from the front of a singly linked list
11. Remove a node from the end of a singly linked list
12. Write an application program that uses a singly linked list class

Related text sections:

Chapter 10  
Chapter 17

## Laboratory Exercise 8 Instructions

### Part 1.

1. Logon to the linux system. Create a subdirectory called Lab8.1
2. Copy the 4 files from ~tiawatts/cs215pickup/Lab8.1 to your new subdirectory.
3. List the files you copied. You should see makefile, Alist.h, Alist.cpp, and application.cpp,.

#### makefile:

```
lab8 : application.o AList.o
      g++ -o lab8 application.o AList.o -g

application.o : application.cpp AList.h
      g++ -c application.cpp -g

AList.o : AList.cpp AList.h
      g++ -c AList.cpp -g

clean :
      rm -f core.* *.o lab8
```

#### AList.h

```
#ifndef ALIST_H
#define ALIST_H

#define MIN_SIZE 10
#define GROW_FACTOR 1.25
#define SHRINK_LIMIT 0.5
#define SHRINK_FACTOR 0.8

#include <iostream>
using namespace std;

class AList
{
public:
    AList ();
    AList (const AList & other);
    ~AList ();
    AList & operator = (const AList & other);
    bool AList::operator == (const AList & other);
    int Capacity () const;
    int Used () const;
    friend ostream & operator << (ostream & outs, const AList & L);
    bool Insert (const int & value);
    bool Delete (const int & value);
private:
    int * data;
    int capacity;
    int used;
};
#endif
```

## AList.cpp

```
#include "AList.h"

AList::AList ()
{
    data = new int [MIN_SIZE];
    capacity = MIN_SIZE;
    used = 0;
}

AList::AList (const AList & other)
{
    data = new int [other.capacity];
    for (int i = 0; i < other.used; i++)
        data[i] = other.data[i];
    capacity = other.capacity;
    used = other.used;
}

AList::~AList ()
{
    delete [] data;
}

AList & AList::operator = (const AList & other)
{
    if (this == &other)
        return * this;
    delete [] data;
    data = new int [other.capacity];
    for (int i = 0; i < other.used; i++)
        data[i] = other.data[i];
    capacity = other.capacity;
    used = other.used;
    return * this;
}

bool AList::operator == (const AList & other)
{
    if (used != other.used)
        return false;
    for (int i = 0; i < used; i++)
        if (data[i] != other.data[i])
            return false;
    return true;
}

int AList::Capacity () const
{
    return capacity;
}

int AList::Used () const
```

```

    {
        return used;
    }

ostream & operator << (ostream & outs, const AList & L)
{
    if (L.used)
        outs << L.data[0];
    for (int i = 1; i < L.used; i++)
        outs << ' ' << L.data[i];
    return outs;
}

bool AList::Insert (const int & value)
{ // This is a stubb; you will be completing this function.

    return false;
}

bool AList::Delete (const int & value)
{ // This is a stubb; you will be completing this function.
    return false;
}

```

4. You will need to complete the implementation of the Insert function. This function should:
  - a. Test to see if there is enough room in the data array to hold a new value.
  - b. If there is not enough room, the data array will need to be resized. The new capacity of the array should be the product of the current capacity and the GROWTH\_FACTOR.
  - c. If the data array is being resized, if the system cannot allocate enough space, false should be returned.
  - d. Locate the position where the new value should be stored to maintain ascending order.
  - e. Move the array entries following the position back one space each so that there is a space for the new value.
  - f. Insert the new value in the array in the desired position.
  - g. Increment the used count.
  - h. Return true if the insertion is successful.
  
5. You will need to complete the implementation of the Delete function. This function should:
  - a. Search for the input value in the data array.
  - b. If the value is not in the data array, return false.
  - c. If the value is in the list, remove it from the list by moving the data items following the value. If there is more than one occurrence of the value in the list, just remove the first one.
  - d. Decrement the used counter.
  - e. If less than SHRINK\_LIMIT of the capacity is currently in use, the data array will need to be resized. The new capacity of the array should be the maximum of MIN\_SIZE or the product of the current capacity and the SHRINK\_FACTOR.
  - f. If the data array is being resized, if the system cannot allocate enough space, false should be returned.
  - g. Return true if the deletion is successful.

## application.cpp

```
// Lab 8 Part 1 application program
// Author: Dr. Watts
// Data: Spring 2009
// Description: This program is designed to test the AList class.

#include <iostream>
#include <fstream>
#include "AList.h"

using namespace std;

int main (int argc, char * argv[])
{
    if (argc < 2)
    {
        cerr << "Usage: " << argv[0] << " <file-name>\n";
        exit (1);
    }
    ifstream ins (argv[1]);
    if (ins.fail ())
    {
        cerr << "No file " << argv[1] << " found\n";
        exit (2);
    }

    AList AL;
    char action;
    int value;

    while (ins >> action)
        switch (action)
        {
            case 'C' : cout << AL.Capacity() << endl;
                       break;
            case 'D' : ins >> value;
                       if (AL.Delete (value))
                           cout << value << " deleted from list\n";
                       else
                           cout << value << " not deleted from list\n";
                       break;
            case 'I' : ins >> value;
                       if (AL.Insert (value))
                           cout << value << " inserted in list\n";
                       else
                           cout << value << " not inserted in list\n";
                       break;
            case 'U' : cout << AL.Used() << endl;
                       break;
            case 'W' : cout << AL << endl;
                       break;
            default : cout << action << " is not a valid option\n";
        }
    return 0;
}
```

6. The application program provides a framework for testing this class. You will need to create an input file to drive the testing process. A simple input file might simply place a value in the list, check the list's capacity and its used count and print the list:
 

```
I 10
C
U
W
```
7. By adding new commands to the input file, test the accuracy of your Insert and Delete functions.
8. Have all of the functions in the class been tested? Add a cout statement at the beginning of each function to verify that it is actually being called. If any of the functions are not being called, add additional actions to the application program.
9. When you are convinced that your class is working correctly, remove the cout statements from your AList function implementations and copy your AList.cpp file to the dropbox as yourlastnameL8-1.cpp.

#### Part 2.

1. Create a subdirectory called Lab8.2
2. Copy the 2 files from ~tiawatts/cs215pickup/Lab8.2 to your new subdirectory.
3. List the files you copied. You should see makefile and application.cpp.

makefile

```
lab8 : application.o LList.o
      g++ -o lab8 application.o LList.o -g

application.o : application.cpp LList.h
      g++ -c application.cpp -g

LList.o : LList.cpp LList.h
      g++ -c LList.cpp -g

clean :
      rm -f core.* *.o lab8
```

4. You will be using a text editor to create the files LList.h and LList.cpp.

5. Enter the following class description into a file called LList.h:

```
#ifndef LLIST_H
#define LLIST_H

#include <iostream>
using namespace std;

class LList
{
    private:
        class LNode
        {
            public:
                LNode ();
                int data;
                LNode * next;
        };

    public:
        LList ();
        LList (const LList & other);
        ~LList ();
        LList & operator = (const LList & other);
        bool operator == (const LList & other);
        int Size () const;
        friend ostream & operator << (ostream & outs, const LList & L);
        bool InsertFirst (const int & value);
        bool InsertLast (const int & value);
        bool DeleteFirst ();
        bool DeleteLast ();
    private:
        LNode * first;
        int size;
};

#endif
```

6. Enter the following class description into a file called LList.cpp:

```
#include "LList.h"

LList::LNode::LNode ()
{
    data = 0;
    next = NULL;
}

LList::LList ()
{
    first = NULL;
    size = 0;
}
```

```

LList::LList (const LList & other)
{
    first = NULL;
    size = 0;
    for (LNode * n = other.first; n != NULL; n = n->next)
        InsertLast (n->data);
}

LList::~LList ()
{
    while (first)
        DeleteFirst();
}

LList & LList::operator = (const LList & other)
{
    if (this == &other)
        return * this;
    while (first)
        DeleteFirst();
    for (LNode * n = other.first; n != NULL; n = n->next)
        InsertLast (n->data);
    return * this;
}

bool LList::operator == (const LList & other)
{
    if (size != other.size)
        return false;
    LNode * n = first;
    LNode * m = other.first;
    while (n != NULL)
    {
        if (n->data != m->data)
            return false;
        n = n->next;
        m = m->next;
    }
    return true;
}

int LList::Size () const
{
    return size;
}

ostream & operator << (ostream & outs, const LList & L)
{
    if (L.first == NULL)
        return outs;
    outs << L.first->data;
    for (LList::LNode * n = L.first->next; n != NULL; n = n->next)
        outs << ' ' << n->data;
    return outs;
}

```

```

bool LList::InsertFirst (const int & value)
{ // This is a stubb; you will be completing this function.
  return false;
}

bool LList::InsertLast (const int & value)
{ // This is a stubb; you will be completing this function.
  return false;
}

bool LList::DeleteFirst ()
{ // This is a stubb; you will be completing this function.
  return false;
}

bool LList::DeleteLast ()
{ // This is a stubb; you will be completing this function.
  return false;
}

```

6. Enter the following data into a file called test.in:

```

I 5
J 10
I 30
J 40
I 8
W
S
J -3
I 17
J 24
I 35
W
S
D
E
D
E
D
E
D
W
S

```

## 7. After carefully reading the application program:

```
// Lab 8 Part 2 application program
// Author: Dr. Watts
// Data: Spring 2009
// Description: This program is designed to test the LList class.

#include <iostream>
#include <fstream>
#include "LList.h"

using namespace std;

int main (int argc, char * argv[])
{
    if (argc < 2)
    {
        cerr << "Usage: " << argv[0] << " <file-name>\n";
        exit (1);
    }
    ifstream ins (argv[1]);
    if (ins.fail ())
    {
        cerr << "No file " << argv[1] << " found\n";
        exit (2);
    }
    LList LL;
    char action;
    int value;

    while (ins >> action)
        switch (action)
        {
            case 'D' : if (LL.DeleteFirst ())
                        cout << "First value deleted from list\n";
                    else
                        cout << "First value not deleted from list\n";
                    break;
            case 'E' : if (LL.DeleteLast ())
                        cout << "Last value deleted from list\n";
                    else
                        cout << "Last value not deleted from list\n";
                    break;
            case 'I' : ins >> value;
                    if (LL.InsertFirst (value))
                        cout << value << " inserted at front of list\n";
                    else
                        cout << value << " not inserted at front of list\n";
                    break;
            case 'J' : ins >> value;
                    if (LL.InsertLast (value))
                        cout << value << " inserted at end of list\n";
                    else
                        cout << value << " not inserted at end of list\n";
                    break;
            case 'S' : cout << LL.Size() << endl;
                    break;
            case 'W' : cout << LL << endl;
                    break;
            default : cout << action << " is not a valid option\n";
        }
    return 0;
}
```

7a. Predict the output of the program for test.in after the class has been completed:

7b. Predict the output of the program for test.in with the class in its current state:

8. Compile (make) and execute the the program using test.in as the input file:

```
lab8 test.in
```

were your predictions for 7b correct?

9. Implement the InsertFirst function. This function should:
  - a. Allocate space for a new node.
  - b. Return false if space cannot be allocated for the node.
  - c. Set the data of the new node to the input value
  - d. Set the next of the new node to the current first.
  - e. Set the first of the list to the new node.
  - f. Increment the size of the list.
  - g. Return true.
10. Compile and test the program to verify that your InsertFirst function is working correctly.
11. Implement the DeleteFirst function: This function should:
  - a. Return false if the list is empty (first == NULL OR size == 0).
  - b. Save the first pointer in a temporary variable.
  - c. Change first to the current first's next.
  - d. Delete the node that was first (its address is saved in your temporary variable.)
  - e. Decrement the size of the list.
  - f. Return true.
12. Compile and test the program to verify that your DeleteFirst function is working correctly.
13. Implement the InsertLast function. This function should:
  - a. If the list is empty, call InsertFirst and return its result.
  - b. Allocate space for a new node.
  - c. Return false if space cannot be allocated for the node.
  - d. Set the data of the new node to the input value
  - e. Set the next of the new node to NULL.
  - f. Traverse to the last node in the current list; save the pointer to the last node in a temporary variable.
  - g. Set the next of the last node to the new node.
  - h. Increment the size of the list.
  - i. Return true.
14. Compile and test the program to verify that your InsertLast function is working correctly.
15. Implement the DeleteLast function: This function should:
  - a. Return false if the list is empty (first == NULL OR size == 0).
  - b. If there is only one node in the list (size == 1), call DeleteFirst and return its result
  - c. Traverse to the 2<sup>nd</sup> to last node in the list..
  - d. Save the pointer to the last node in a temporary variable.
  - e. Change the next of the 2<sup>nd</sup> to last item to NULL..
  - f. Delete the node that was last (its address is saved in your temporary variable.)
  - g. Decrement the size of the list.
  - h. Return true.
16. Compile and test the program to verify that your DeleteLast function is working correctly.
17. When you are convinced that your LList class is working correctly, copy your LList.cpp file to the dropbox as yourlastnameL8-2.cpp.

## Laboratory Exercise 9 (Template classes)

Topics: Template classes

Goals: Upon successful completion of this lab you should be able to:

1. Define a template class
2. Test a template class using standard C++ types
3. Test a template class using a user defined type

Related text sections:

Chapter 16, 17

## Laboratory Exercise 9 Instructions

1. Create a new directory (folder) called Lab9.
2. Concatenate your LList.h and LList.cpp files from Lab 8 Part 2 into a new file called LList.tmp in your new Lab9 directory. Modify the class description as indicated below. Modify the function implementations as described in lecture (the modified default constructors are illustrated below).

```
#ifndef LLIST_TMP
#define LLIST_TMP

#include <iostream>
using namespace std;

template <class LT> class LList;

template <class LT> ostream & operator << (ostream & outs, const LList<LT> & L);

template <class LT>
class LList
{
private:
    class LNode
    {
public:
        LNode ();
        LT data;
        LNode * next;
    };

public:
    LList ();
    LList (const LList & other);
    ~LList ();
    LList & operator = (const LList & other);
    bool operator == (const LList & other);
    int Size () const;
    friend ostream & operator << <> (ostream & outs, const LList<LT> & L);
    bool InsertFirst (const LT & value);
    bool InsertLast (const LT & value);
    bool DeleteFirst ();
    bool DeleteLast ();
private:
    LNode * first;
    int size;
};

template <class LT>
LList<LT>::LNode::LNode ()
{
    next = NULL;
}

template <class LT>
LList<LT>::LList ()
{
    first = NULL;
    size = 0;
}

    . . . More modified functions here . . .

#endif
```

3. Create an application program to test your linked list template. This program should instantiate LList objects using two of the standard C++ types: int, float, double, char, or bool. For example:

```
LList <int> L1;
LList <char> L2;
```

4. Your application program should specifically test each of the functions in the template class. For example:

```
L1.InsertFirst (10);
L2.InsertLast ('c');
cout << L1 << endl;
cout << L2.Size() << endl;
```

5. Compile and execute your program. Since there is only one .cpp file, you can simply use a g++ command to compile the program.
6. Once you are convinced that your linked list template is working correctly for standard C++ types, make a copy of your application program to test user/compiler defined types.
7. Copy the files CoordPt.h and CoordPt.cpp from CS215pickup/Lab9. (Listings for these files start on the next page.)

8. Add include statements to your new application program to include the string library and CoordPt.h.

9. Modify the LList instantiations to use string and CoordinatePoint:

```
LList <string> L1;
LList <CoordinatePoint> L2;
```

10. Modify your application program so that the functions test the new LList instantiations:

```
L1.InsertFirst ("Hello");
L2.InsertLast (CoordinatePoint (1,2));
cout << L1 << endl;
cout << L2.Size() << endl;
```

11. Create a makefile to compile and link your application program with the CoordPt implementation.

```
lab9 : application.o CoordPt.o
      g++ -o lab9 application.o CoordPt.o -g

application.o : application.cpp LList.tmp CoordPt.h
      g++ -c application.cpp -g

CoordPt.o : CoordPt.cpp CoordPt.h
      g++ -c CoordPt.cpp -g

clean :
      rm -f core.* *.o lab
```

12. Try to compile the program. What is wrong? Modify the CoordPt class so that the program will compile. Compile and test your new application program.

13. When you are convinced that your LList template class is working correctly, copy your LList.tmp file to the dropbox as yourlastnameL9.tmp.

```

#ifndef COORDPT_H
#define COORDPT_H

#include <iostream>
using namespace std;

class CoordinatePoint
{
public:
    CoordinatePoint();
        // Precondition: default constructor - no parameters required.
        // Postcondition: a new object with x set to 0 and y set to 0 will be
        // created.

    CoordinatePoint(int x_coord, int y_coord);
        // Precondition: x_coord and y_coord must be integer values.
        // Postcondition: a new object with x set to x_coord and y set to
        // y_coord will be created.

    CoordinatePoint(const CoordinatePoint & Other);
        // Precondition: copy constructor - Other must be a valid object.
        // Postcondition: a new object with x set to Other.x and y set to Other.y
        // will be created.

    ~CoordinatePoint();
        // Precondition: destructor - no parameters required.
        // Postcondition: no action since there is no dynamic memory.

    CoordinatePoint & operator = (const CoordinatePoint & Other);
        // Precondition: assignment operator - Other must be a valid object.
        // Postcondition: for the existing object, x will be set to Other.x
        // and y will be set to Other.y.

    float Magnitude () const;
        // Precondition: x and y are integer values.
        // Postcondition: the distance from the origin (0,0) will be
        // calculated (using the Pythagorean Theorem) and returned.

    float Distance (const CoordinatePoint & Other) const;
        // Precondition: *this and Other must be a valid objects.
        // Postcondition: the distance from the *this to Other will be
        // calculated (using the Pythagorean Theorem) and returned.

    CoordinatePoint operator + (const CoordinatePoint & Other);
        // Precondition: *this and Other must be a valid objects.
        // Postcondition: the sum of *this and Other will be
        // calculated and returned.

    friend istream & operator >> (istream & input, CoordinatePoint & p);
        // Precondition: p is an object of type CoordinatePoint.
        // Postcondition: the contents of p will be read from the input stream -
        // input will be in the format (x,y) - where x and y are integer values.
        // The parenthesis and comma will be required but will not be stored -
        // the fail flag will be set if the input format is incorrect.

    friend ostream & operator << (ostream & output, const CoordinatePoint & p);
        // Precondition: p is an object of type CoordinatePoint.
        // Postcondition: the contents of p will be sent to the output stream
        // using the format (x,y) where x and y are the x and y coordinates of p.
private:
    int x, y;
};
#endif

```

```

#include "CoordPt.h"
#include <cmath>
using namespace std;

CoordinatePoint::CoordinatePoint()
// Precondition: default constructor - no parameters required.
// Postcondition: a new object with x set to 0 and y set to
// 0 will be created.
{
    x = 0;
    y = 0;
}

CoordinatePoint::CoordinatePoint(int x_coord, int y_coord)
// Precondition: x_coord and y_coord must be integer values.
// Postcondition: a new object with x set to x_coord and y set to
// y_coord will be created.
{
    x = x_coord;
    y = y_coord;
}

CoordinatePoint::CoordinatePoint(const CoordinatePoint & Other)
// Precondition: copy constructor - Other must be a valid object.
// Postcondition: a new object with x set to Other.x and
// y set to Other.y will be created.
{
    x = Other.x;
    y = Other.y;
}

CoordinatePoint::~CoordinatePoint()
// Precondition: destructor - no parameters required.
// Postcondition: no action since there is no dynamic memory.
{
}

CoordinatePoint & CoordinatePoint::operator = (const CoordinatePoint & Other)
// Precondition: assignment operator - Other must be a valid object.
// Postcondition: for the existing object, x will be set to Other.x
// and y will be set to Other.y.
{
    x = Other.x;
    y = Other.y;
    return *this;
}

float CoordinatePoint::Magnitude () const
// Precondition: x and y are integer values.
// Postcondition: the distance from the origin (0,0) will be
// calculated (using the Pythagorean Theorem) and returned.
{
    return sqrt (float (x * x + y * y));
}

float CoordinatePoint::Distance (const CoordinatePoint & Other) const
// Precondition: *this and Other must be a valid objects.
// Postcondition: the distance from the *this to Other will be
// calculated (using the Pythagorean Theorem) and returned.
{
    int deltax = x - Other.x;
    int deltay = y - Other.y;
    return sqrt (float (deltax * deltax + deltay * deltay));
}

```

```

}

CoordinatePoint CoordinatePoint::operator + (const CoordinatePoint & Other)
// Precondition: *this and Other must be a valid objects.
// Postcondition: the sum of *this and Other will be
// calculated and returned.
{
    int xvalue = x + Other.x;
    int yvalue = y + Other.y;
    return CoordinatePoint (xvalue, yvalue);
}

istream & operator >> (istream & input, CoordinatePoint & p)
// Precondition: p is an object of type CoordinatePoint.
// Postcondition: the contents of p will be read from the input stream -
// input will be in the format (x,y) - where x and y are integer values.
// The parenthesis and comma will be required but will not be stored -
// the fail flag will be set if the input format is incorrect.
{
    char format;
    input >> format >> p.x >> format >> p.y >> format;
    return input;
}

ostream & operator << (ostream & output, const CoordinatePoint & p)
// Precondition: p is an object of type CoordinatePoint.
// Postcondition: the contents of p will be sent to the output stream
// using the format (x,y) where x and y are the x and y coordinates
// of p.
{
    output << '(' << p.x << ',' << p.y << ')';
    return output;
}

```

## Laboratory Exercise 10 (Doubly Linked Lists)

Topics: Doubly Linked Lists

Goals: Upon successful completion of this lab you should be able to:

1. Define a doubly linked list class
2. Define a corresponding node class
3. Insert items into the beginning or end of a doubly linked list
4. Remove items from the beginning or end of a doubly linked list
5. Write an application program that uses a doubly linked list class
6. Write traversal functions using passed functions for a doubly linked list.

Related text sections:

Chapter 16, 17

## Laboratory Exercise 10 Instructions

1. Create a new directory (folder) called Lab10.
2. Copy your LList.tmp file from Lab 9 to a new file called LList2.tmp in your new Lab10 directory. Modify the class description as indicated below. Modify the function implementations as described in lecture (the modified default constructors and the traversal functions are illustrated below).

```
#ifndef LLIST2_TMP
#define LLIST2_TMP

#include <iostream>
using namespace std;

template <class LT> class LList2;

template <class LT> ostream & operator << (ostream & outs, const LList2<LT> & L);

template <class LT>
class LList2
{
private:
    class LNode
    {
public:
        LNode ();
        LT data;
        LNode * next;
        LNode * prev;
    };

public:
    LList2 ();
    LList2 (const LList2 & other);
    ~LList2 ();
    LList2 & operator = (const LList2 & other);
    bool operator == (const LList2 & other);
    int Size () const;
    friend ostream & operator << <> (ostream & outs, const LList2<LT> & L);
    bool InsertFirst (const LT & value);
    bool InsertLast (const LT & value);
    bool DeleteFirst ();
    bool DeleteLast ();
    void Forward (void function (const LT & param));
    void Backward (void function (const LT & param));
private:
    LNode * first;
    LNode * last;
    int size;
};

template <class LT>
LList2<LT>::LNode::LNode ()
{
    next = NULL;
    prev = NULL;
}

template <class LT>
LList2<LT>::LList2 ()
{
```

```

    first = NULL;
    last = NULL;
    size = 0;
}
        . . . More modified functions here . . .

template <class LT>
void LList2<LT>::Forward (void function (const LT & param))
{
    for (LNode * n = first; n; n = n->next)
        function (n->data);
}

template <class LT>
void LList2<LT>::Backward (void function (const LT & param))
{
    for (LNode * n = last; n; n = n->prev)
        function (n->data);
}

#endif

```

3. Modify the InsertFirst and InsertLast functions to update the previous links and the pointer to the last node as needed.
4. Modify the DeleteFirst and DeleteLast functions to update the previous links and the pointer to the last node as needed.
5. Create an application program to test the functions of your doubly linked list template class using a standard C++ type for the instantiation.
6. Two new traversal functions have been added to this class: Forward and Backward. These functions are designed to traverse the list and apply a function to the data in each node of the list. To use these functions you will need to create a function in your application program to pass to the traversal. This function will need to have one argument of the same type as your list instantiation. For example, if you are using an integer list:

```
LList2 <int> L;
```

then, the following function could be used when traversing the list:

```

void PrintValue (const int & value)
{
    cout << "The value in the list is " << value << endl;
}

```

To call the Forward traversal, include the command

```
L.Forward (PrintValue);
```

in your application program. As the Forward method progresses through the nodes of L (following the next links, it will call the function PrintValue for each node. The value in data will be passed to the function.

7. Compile and test your new application program.

8. Copy the files Fraction.h, Fraction.cpp, Lab10app.cpp, makefile, and Lab10.in from ~tiawatts/cs215pickup/Lab10. (Lab10app.cpp is listed below; listings of Lab10.in, makefile, Fraction.h, and Fraction.cpp start on the next page.)

#### Lab10app.cpp

```
#include <iostream>
#include <fstream>
#include "LList2.tmp"
#include "Fraction.h"

using namespace std;

// Add a global variable for holding the sum of the fractions here

// Add function prototype here

int main ()
{
    ifstream input ("Lab10.in");
    fraction one;
    LList2 <fraction> FL;

    while (input >> one)
        FL.InsertLast (one);
    cout << "The fractions are: ";
    cout << FL << endl;

// Add code to find the sum of the fractions in the list FL here

// Add code to print the sum here

    return 0;
}

// Add function implementation here
```

9. Add code (where indicated) to find and print the sum of the fractions read from the input file into the list FL.
- You will need to declare a global variable of type fraction to hold the sum of the values.
  - Your function header will need one parameter: a constant reference to an object of type fraction. Your function's return type should be void.
  - Your function should add the fraction passed to it to the global sum.
  - To find the sum of the fractions, you will need to use one of the traversal functions (Forward or Backward) from LList2. You will pass the name of your function to the traversal function.
  - Add code (where indicated) to print the calculated sum stored in your global variable.
10. Compile and execute your program. The output should be:

```
The fractions are: 3 1/2 4 2/3 0 1/2 1 3/4
The total is 10 5/12
```

11. When you are convinced that your LList2 template class is working correctly, copy your LList2.tmp file to the dropbox as yourlastnameL10.tmp.
12. When you are convinced that your program is working correctly, copy your well documented Lab10app.cpp file to the dropbox as yourlastnameL10.cpp.

### Lab10.in

```
3 1/2
4 2/3
0 1/2
1 3/4
```

### makefile

```
lab10 : Lab10app.o Fraction.o
        g++ -o lab10 Lab10app.o Fraction.o -g

Lab10app.o : Lab10app.cpp LList2.tmp Fraction.h
        g++ -c Lab10app.cpp -g

Fraction.o : Fraction.cpp Fraction.h
        g++ -c Fraction.cpp -g

clean :
        rm -f core.* *.o lab10
```

### Fraction.h

```
#ifndef FRACTION_H
#define FRACTION_H

#include <iostream>
using namespace std;

class fraction
{
public:
    fraction ();
    fraction (int w);
    fraction (int n, unsigned d);
    fraction (int w, unsigned n, unsigned d);
    ~fraction ();

    fraction operator = (const fraction & other);
    fraction operator + (const fraction & other) const;

    friend istream & operator >> (istream & ins, fraction & f);
    friend ostream & operator << (ostream & outs, const fraction & f);

private:
    void reduce ();

    int whole;
    int numerator;
    unsigned denominator;
};

#endif
```

## Fraction.cpp

```
#include "Fraction.h"

fraction::fraction ()
{
    whole = 0;
    numerator = 0;
    denominator = 1;
}

fraction::fraction (int w)
{
    whole = w;
    numerator = 0;
    denominator = 1;
}

fraction::fraction (int n, unsigned d)
{
    whole = 0;
    numerator = n;
    denominator = d;
    reduce();
}

fraction::fraction (int w, unsigned n, unsigned d)
{
    whole = w;
    numerator = n;
    denominator = d;
    this->reduce();
}

fraction::~fraction ()
{
}

fraction fraction::operator = (const fraction & other)
{
    this->whole = other.whole;
    this->numerator = other.numerator;
    this->denominator = other.denominator;
    this->reduce();

    return *this;
}

fraction fraction::operator + (const fraction & other) const
{
    fraction sum;

    sum.whole = this->whole + other.whole;
    sum.numerator = this->numerator * other.denominator
        + this->denominator * other.numerator;
    sum.denominator = this->denominator * other.denominator;
    sum.reduce();

    return sum;
}
```

```

istream & operator >> (istream & ins, fraction & f)
{
    // Note: trivial version of >>

    char temp;

    ins >> f.whole >> f.numerator >> temp >> f.denominator;

    return ins;
}

ostream & operator << (ostream & outs, const fraction & f)
{
    // Note: trivial version of <<

    outs << f.whole << ' ' << f.numerator << '/' << f.denominator;

    return outs;
}

void fraction::reduce ()
{
    int f = 2;

    denominator = denominator == 0 ? 1 : denominator;
    whole += numerator / denominator;
    numerator %= denominator;
    while (f <= numerator)
        if ((numerator % f == 0) && (denominator % f == 0))
        {
            numerator /= f;
            denominator /= f;
        }
        else
            f++;
}

```



## Laboratory Exercise 11 (Templated Lists)

Topics: Template Linked Lists

Iterators

Overloaded ++, --, and \* operators

Goals: Upon successful completion of this lab you should be able to:

1. Define an iterator for a template container class
2. Overload the increment (++) and decrement (--) operators for an iterator.
3. Overload the de-referencing operator (\*) for an iterator.
4. Use the Iterator for a Linked List class in an application program

Related text sections:

Chapter 16

Chapter 17

## Laboratory Exercise 11 Instructions

1. Copy your LList2.tmp file from Lab 10 to a new Lab 11 directory.
2. Modify the node type to be a struct.. You will no longer need the “public:” key word since all of the elements of a struct are public by default.
3. Add an iterator class and begin/end and rbegin/rend functions to your LList2 class description:

```
template <class LT> class LList2;

template <class LT> ostream & operator << (ostream & outs, const LList2<LT> & L);

template <class LT>
class LList2
{
    private:
        typedef struct LNode
        {
            LNode ();
            LT data;
            LNode * next;
            LNode * prev;
        };

    public:
        class Iterator
        {
            public:
                Iterator ();
                Iterator (LNode * NP);
                const LT operator * () const;
                Iterator operator ++ ();
                Iterator operator ++ (int);
                Iterator operator -- ();
                Iterator operator -- (int);
                bool operator == (const Iterator & other) const;
                bool operator != (const Iterator & other) const;
            private:
                LNode * current;
        };

        LList2 ();
        LList2 (const LList2 & other);
        ~LList2 ();
        LList2 & operator = (const LList2 & other);
        bool operator == (const LList2 & other);
        int Size () const;
        friend ostream & operator << >> (ostream & outs, const LList2<LT> & L);
        bool InsertFirst (const LT & value);
        bool InsertLast (const LT & value);
        bool DeleteFirst ();
        bool DeleteLast ();
        void Forward (void function (const LT & param));
        void Backward (void function (const LT & param));
        Iterator begin () const;
        Iterator rbegin () const;
        Iterator end () const;
        Iterator rend () const;
};
```

```

private:
    LNode * first;
    LNode * last;
    int size;
};

```

4. Add the following Iterator function implementations to your LList2.tmp file:

```

template <class LT>
LList2<LT>::Iterator::Iterator ()
{
    current = NULL;
}

template <class LT>
LList2<LT>::Iterator::Iterator (LNode * NP)
{
    current = NP;
}

template <class LT>
const LT LList2<LT>::Iterator::operator * () const
{
    return current->data;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::Iterator::operator ++ ()
{
    current = current->next;
    return *this;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::Iterator::operator ++ (int)
{
    Iterator temp = *this;
    current = current->next;
    return temp;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::Iterator::operator -- ()
{
    current = current->prev;
    return *this;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::Iterator::operator -- (int)
{
    Iterator temp = *this;
    current = current->prev;
    return temp;
}

template <class LT>
bool LList2<LT>::Iterator::operator == (const Iterator & other) const
{
    return current == other.current;
}

```

```

template <class LT>
bool LList2<LT>::Iterator::operator != (const Iterator & other) const
{
    return current != other.current;
}

```

5. Add the following LList2 function implementations to your LList2.tmp file:

```

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::begin () const
{
    Iterator temp (first);
    return temp;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::rbegin () const
{
    Iterator temp (last);
    return temp;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::end () const
{
    Iterator temp;
    return temp;
}

template <class LT>
typename LList2<LT>::Iterator LList2<LT>::rend () const
{
    Iterator temp;
    return temp;
}

```

6. Enter the following application program in a new file called application.cpp. **DO NOT COMPILE THIS PROGRAM!**

```

#include <iostream>
#include "LList2.tmp"

using namespace std;

#define MAX 8

int main ()
{
    LList2 <int> L1;
    for (int i = 0; i < MAX; i++)
        if (i % 2)
            L1.InsertFirst(i);
        else
            L1.InsertLast(i);
    cout << "L1 in order: ";
    cout << L1 << endl;
    cout << "Testing begin, end, post++ and *\n";
}

```

```

    for (LList2<int>::Iterator itr = L1.begin (); itr != L1.end(); itr++)
        cout << "Value: " << *itr << endl;
    return 0;
}

```

7. **Before** compiling and executing the program, predict the output of the program:

8. Compile the program using the following command:

```
g++ -w application.cpp -o lab11a
```

Note: the `-w` option will suppress all warnings.

9. Execute the program. How accurate was your prediction?

10. Add code to the program to test the `rbegin`, `rend`, and `post--` functions:

```

    cout << "Testing rbegin, rend, post-- and *\n";
    for (LList2<int>::Iterator itr = L1.rbegin(); itr != L1.rend(); itr--)
        cout << "Value: " << *itr << endl;

```

**Before** compiling and executing the program, predict the output of this segment of the program:

Compile and execute the program; how accurate was your prediction?

11. Add code to the program to test the `pre++` function:

```

    cout << "Testing pre++\n";
    cout << "Should write second item in list L1\n";
    List2<int>::Iterator itr = L1.begin();
    cout << *++itr << endl;

```

**Before** compiling and executing the program, predict the output of this segment of the program:

Compile and execute the program; how accurate was your prediction?

12. Add code to the program to test the `pre--` function:

```

    cout << "Testing pre--\n";
    cout << "Should write second to last item in list L1\n";
    itr = L1.rbegin();
    cout << *--itr << endl;

```

**Before** compiling and executing the program, predict the output of this segment of the program:

Compile and execute the program; how accurate was your prediction?

13. Copy the files `card.h` and `card.cpp` from the `Lab11` subdirectory in the `cs215pickup` directory. These files contain a class description and implementation for a set of playing cards. Carefully review the class description and its implementation. (Listings of `card.h` and `card.cpp` start on the next page.)
14. Copy the file `cards.in` from `cs215pickup`. This file contains a list of cards in the `card` class format.

```
AC AD AH AS
2C 2D 2H 2S
3C 3D 3H 3S
4C 4D 4H 4S
```

15. Create a new application that includes `card.h` and your `LList2.tmp`, reads the cards in the input file, stores them in a linked list, and writes out the list of cards. A sample makefile for this program is illustrated below:

```
lab11 : Lab11app.o card.o
      g++ -o lab11 Lab11app.o card.o -g

Lab11app.o : Lab11app.cpp LList2.tmp card.h
      g++ -c Lab11app.cpp -g -w

card.o : card.cpp card.h
      g++ -c card.cpp -g

clean :
      rm -f core.* *.o lab11
```

16. Add an additional segment of code to your program that creates two new linked lists, each holding half of the deck, and writes out the two new lists of cards.
17. Add an additional segment of code to your program that shuffles the two half lists, starting with the first card in the second list, into a new linked list and writes out the shuffled list.
18. For the input file you copied from the pickup folder, the output should be:

```
The cards: AC AD AH AS 2C 2D 2H 2S 3C 3D 3H 3S 4C 4D 4H 4S
```

```
First half: AC AD AH AS 2C 2D 2H 2S
```

```
Second half: 3C 3D 3H 3S 4C 4D 4H 4S
```

```
Shuffled cards: 3C AC 3D AD 3H AH 3S AS 4C 2C 4D 2D 4H 2H 4S 2S
```

18. When you are convinced that your card shuffling program is working correctly, copy your well documented `.cpp` file to the dropbox as `yourlastnameL11.cpp`.

## card.h

```
#ifndef CARD_H
#define CARD_H
#include <iostream>
using namespace std;

class card
{
public:
    card ();
    card (const card & other);
    ~card ();
    card & operator = (const card & other);
    bool operator == (const card & other) const;
    bool operator != (const card & other) const;
    bool operator < (const card & other) const;
    bool operator > (const card & other) const;
    bool operator <= (const card & other) const;
    bool operator >= (const card & other) const;
    friend istream & operator >> (istream & ins, card & C);
    friend ostream & operator << (ostream & outs, const card & C);
    char rank;
    char suit;
private:
    void evaluate ();
    int sortval;
};
```

## card.cpp

```
#include "card.h"

card::card ()
{
    rank = suit = '?';
    sortval = 0;
}

card::card (const card & other)
{
    rank = other.rank;
    suit = other.suit;
    sortval = other.sortval;
}

card::~~card ()
{
}

card & card::operator = (const card & other)
{
    if (this == &other)
        return * this;
    rank = other.rank;
    suit = other.suit;
    sortval = other.sortval;
    return * this;
}

bool card::operator == (const card & other) const
{
    return sortval == other.sortval;
}
```

```

bool card::operator != (const card & other) const
{
    return sortval != other.sortval;
}

bool card::operator < (const card & other) const
{
    return sortval < other.sortval;
}

bool card::operator > (const card & other) const
{
    return sortval > other.sortval;
}

bool card::operator <= (const card & other) const
{
    return sortval <= other.sortval;
}

bool card::operator >= (const card & other) const
{
    return sortval >= other.sortval;
}

istream & operator >> (istream & ins, card & C)
{
    card T;
    ins >> T.rank;
    ins >> T.suit;
    T.evaluate();
    if (T.sortval)
        C = T;
    else
        ins.setstate(ios::failbit);
    return ins;
}

ostream & operator << (ostream & outs, const card & C)
{
    outs << C.rank << C.suit;
    return outs;
}

void card::evaluate ()
{
    rank = toupper(rank);
    suit = toupper(suit);
    sortval = 0;
    char r[] = {'A','2','3','4','5','6','7','8','9','T','J','Q','K'};
    char s[] = {'C','D','H','S'};
    int i;
    for (i = 0; i < 13 && rank != r[i]; i++);
        sortval = (i+1) * 100;
    for (i = 0; i < 4 && suit != s[i]; i++);
        sortval += i+1;
    if (sortval / 100 < 1 || sortval / 100 > 13 ||
        sortval % 100 < 1 || sortval % 100 > 4)
        sortval = 0;
}

```

## Laboratory Exercise 12 (Exception Handling)

Topics: Overloaded [] operator  
Exception Handling

Goals: Upon successful completion of this lab you should be able to:

1. Overload the indexing ([]) operator for a template container class.
2. Use the C++ try...throw...catch syntax
3. Incorporate exception handling in a template container class
4. Write a program that incorporates exception handling

Related text sections:  
Chapter 18

## Laboratory Exercise 12 Instructions

1. Copy your LList2.tmp file from Lab 11 to a new Lab 12 directory.
2. Add the following prototype and draft implementation to your LList2 template class:

```
LT & operator [] (const int & index) const;

template <class LT>
LT & LList2<LT>::operator [] (const int & index) const
{
    LList2<LT>::LNode * n = first;
    return n->data;
}
```

*Note that the function operator[] returns a reference type. This causes it to be an l-value, allowing you to use subscripted expressions on either side of assignment operators.*

The complete description of the LList2 template is at the end of this Laboratory Exercise.

3. Enter the following program in a file called application.cpp.

```
#include <iostream>
#include "LList2.tmp"

using namespace std;

#define MAX 10

int main ()
{
    LList2 <int> L1, L2;
    for (int i = 0; i < MAX; i++)
        if (i % 2)
            L1.InsertFirst(i);
        else
            L1.InsertLast(i);
    cout << "L1 in order: ";
    cout << L1 << endl;
    cout << "Testing the [] operator\n";
    cout << "First to last:\n";
    for (int i = 0; i < L1.Size(); i++)
        cout << "L1[" << i << "] is " << L1[i] << endl;
    L1[3] = 7;
    cout << "Last to first:\n";
    for (int i = L1.Size()-1; i >= 0; i--)
        cout << "L1[" << i << "] is " << L1[i] << endl;
    return 0;
}
```

4. Compile and execute the program. Are the results what you expected from the code in the application program?
5. Modify the [] operator function to traverse through the list until it reaches the entry corresponding to the position in the index parameter. The operator should then return the data item in the selected entry.
6. Test your program again. The correct output should be:

```

First to last:
L1[0] is 9
L1[1] is 7
L1[2] is 5
L1[3] is 3
L1[4] is 1
L1[5] is 0
L1[6] is 2
L1[7] is 4
L1[8] is 6
L1[9] is 8
Last to first:
L1[9] is 8
L1[8] is 6
L1[7] is 4
L1[6] is 2
L1[5] is 0
L1[4] is 1
L1[3] is 7
L1[2] is 5
L1[1] is 7
L1[0] is 9

```

7. If you did not get the correct output, review the code in your [] operator and modify it.
8. Currently the dereferencing operator in your template can only be used on the Right Hand Side (rhs) of an assignment statement. Edit the dereferencing operator (\*) for the iterator in LList2.tmp to allow it to be used on the Left Hand Side (lhs) of an assignment statement:

The prototype should be:

```
LT & operator * () const;
```

The implementation should be:

```

template <class LT>
LT & LList2<LT>::Iterator::operator * () const
{
    return current->data;
}

```

9. Add statements to application.cpp to test the dereferencing operator as both a lhs and as a rhs operator.
10. Copy the C++ files in ~tiawatts/cs215pickup/Lab12 to your Lab 12 directory.

11. Compile the file Lab12a.cpp.
12. **Before running the program** carefully read the source code and predict the output if the value calculated for “num” is 10 and the value calculated for “val” is 7.
  
13. Execute the program. Executing this program should result in a Segmentation Fault. Using gdb, determine where the program crashed and modify the .cpp file to eliminate the crash. Describe the problem:
  
14. Are you now getting the expected output? If not, continue to debug the .cpp file until the program executes correctly.
15. Compile the file Lab12b.cpp.
16. **Before running the program** carefully read the source code and predict the output if the value calculated for “num” is 8 and the value calculated for “val” is 3.
  
17. Execute the program. Executing this program should result in a Segmentation Fault. Using gdb, determine where the program crashed and modify the .cpp file to eliminate the crash. Describe the problem:

18. Are you now getting the expected output? If not, continue to debug the .cpp file until the program executes correctly.
19. Compile the file Lab12c.cpp.
20. **Before running the program** carefully read the source code and predict the output if the value calculated for “num” is 11 and the value calculated for “val” is 5.

21. Execute the program. Executing this program should result in a Segmentation Fault. Using gdb, determine where the program crashed and modify the .cpp file to eliminate the crash. Describe the problem:

22. Are you now getting the expected output? If not, continue to debug the .cpp file until the program executes correctly.

23. Modify the \*, ++, --, and [] operators for the Iterator class to catch exceptions using C++ try-throw-catch structures. Each of these operators should pass an appropriate error message to the catch and should then exit from the program. Each exit command should have a unique numeric identifier argument. For example:

```
template <class LT>
LT & LList2<LT>::Iterator::operator * () const
{
    try
    {
        if (current == NULL)
            throw ("Cannot dereference a NULL pointer");
        return current->data;
    }
    catch (const char * message)
    {
        cerr << message << endl;
        exit (1);
    }
}
```

24. When you are convinced that your Linked List template is working correctly, copy your well documented .tmp file to the dropbox as yourlastnameL12.tmp.

```

template <class LT> class LList2;

template <class LT> ostream & operator << (ostream & outs, const LList2<LT> & L);

template <class LT>
class LList2
{
private:
    typedef class LNode
    {
    public:
        LNode ();
        LT data;
        LNode * next;
        LNode * prev;
    };

public:
    typedef class Iterator
    {
    public:
        Iterator ();
        Iterator (LNode * NP);
        LT & operator * () const;
        Iterator operator ++ ();
        Iterator operator ++ (int);
        Iterator operator -- ();
        Iterator operator -- (int);
        bool operator == (const Iterator & other) const;
        bool operator != (const Iterator & other) const;
    private:
        LNode * current;
    };
    LList2 ();
    LList2 (const LList2 & other);
    ~LList2 ();
    LList2 & operator = (const LList2 & other);
    bool operator == (const LList2 & other);
    int Size () const;
    friend ostream & operator << <> (ostream & outs, const LList2<LT> & L);
    bool InsertFirst (const LT & value);
    bool InsertLast (const LT & value);
    bool DeleteFirst ();
    bool DeleteLast ();
    void Forward (void function (const LT & param));
    void Backward (void function (const LT & param));
    Iterator begin () const;
    Iterator rbegin () const;
    Iterator end () const;
    Iterator rend () const;
    LT & operator [] (const int & index) const;
private:
    LNode * first;
    LNode * last;
    int size;
};

```

## Laboratory Exercise 13 (Intro to MFC)

Topics: Microsoft Foundation Classes

Goals: Upon successful completion of this lab you should be able to:

1. Write a simple windows program using the MFC Frame Window Class

Related text sections:

Chapters 1 - 5 (Introduction to MFC)

## Laboratory Exercise 13 Instructions

1. Create a new Visual C++ application project called Lab13.
2. Add a new header file called CLab13Win.h to your project. Enter the following class description:

```
#include <afxwin.h>

class CLab13Win : public CFrameWnd
{
public:
    CLab13Win ();
    afx_msg void OnPaint ();
private:
    int m_nMessageX;
    int m_nMessageY;
    DECLARE_MESSAGE_MAP ()
};
```

3. Add a new header file called CLab13App.h to your project. Enter the following class description:

```
#include <afxwin.h>
#include "CLab13Win.h"

class CLab13App : public CWinApp
{
public:
    BOOL InitInstance ();
};
```

4. Add a new implementation file called CLab13Win.cpp to your project. Enter the following class method implementation and message map:

```
#include <afxwin.h>
#include "CLab13Win.h"

CLab13Win::CLab13Win ()
{
    Create (NULL, "Lab13");
    m_nMessageX = 100;
    m_nMessageY = 100;
}

afx_msg void CLab13Win::OnPaint ()
{
    CPaintDC dc (this);
    dc.DrawText("Hello World", CRect (m_nMessageX, m_nMessageY,
                                     m_nMessageX+80, m_nMessageY+16), DT_LEFT);
}

BEGIN_MESSAGE_MAP (CLab13Win, CFrameWnd)
    ON_WM_PAINT ()
END_MESSAGE_MAP ()
```

5. Add a new implementation file called CLab13App.cpp to your project. Enter the following class method implementation and application variable declaration

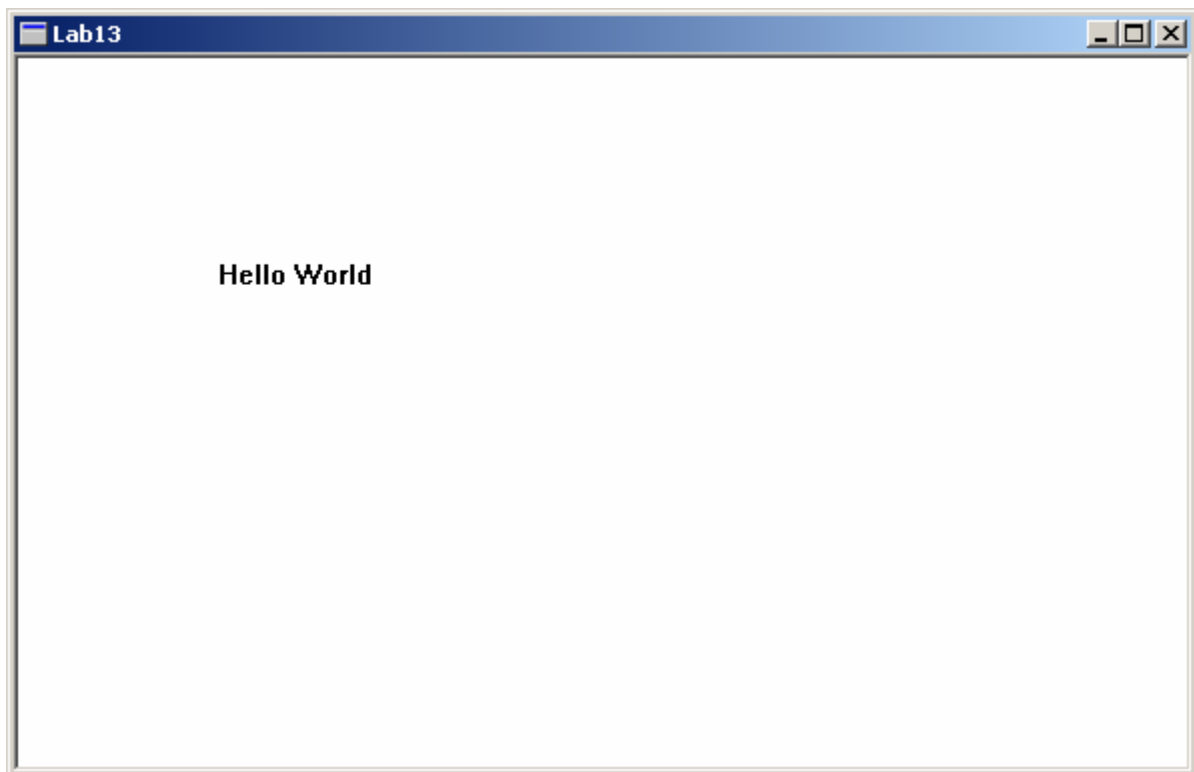
```
#include <afxwin.h>
#include "CLab13App.h"

BOOL CLab13App::InitInstance ()
{
    m_pMainWnd = new CLab13Win();
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();

    return TRUE;
}

CLab13App Lab13App;
```

6. Compile and execute the program – you should see:



You have now written an MFC “Hello World” program!

7. This program will be more interesting if the “Hello World” text string is movable. One way to move items is to drag them with the mouse. Modify CLab13Win.h and CLab13Win.cpp as indicated:

CLab13Win.h:

```
#include <afxwin.h>

class CLab13Win : public CFrameWnd
{
public:
    CLab13Win ();
    afx_msg void OnPaint ();
    afx_msg void OnMouseMove( UINT nFlags, CPoint point );
private:
    int m_nMessageX;
    int m_nMessageY;
    DECLARE_MESSAGE_MAP ()
};
```

CLab13Win.cpp:

```
#include <afxwin.h>
#include "CLab13Win.h"

CLab13Win::CLab13Win ()
{
    Create (NULL, "Lab13");
    m_nMessageX = 100;
    m_nMessageY = 100;
}

afx_msg void CLab13Win::OnPaint ()
{
    CPaintDC dc (this);
    dc.DrawText("Hello World", CRect (m_nMessageX, m_nMessageY,
        m_nMessageX+80, m_nMessageY+16), DT_LEFT);
}

afx_msg void CLab13Win::OnMouseMove( UINT nFlags, CPoint point )
{
    if (nFlags == MK_LBUTTON)
    {
        m_nMessageX = point.x;
        m_nMessageY = point.y;
        Invalidate (TRUE);
    }
}

BEGIN_MESSAGE_MAP (CLab13Win, CFrameWnd)
    ON_WM_PAINT ()
    ON_WM_MOUSEMOVE( )
END_MESSAGE_MAP ()
```

8. Compile and execute the modified program.
9. The next set of modifications will create a patterned background for the window. Modify CLab13Win.h and CLab13Win.cpp as indicated:

CLab13Win.h:

```
#include <afxwin.h>

class CLab13Win : public CFrameWnd
{
public:
    CLab13Win ();
    afx_msg void OnPaint ();
    afx_msg void OnMouseMove( UINT nFlags, CPoint point );
private:
    int m_nMessageX;
    int m_nMessageY;
    int m_nNumRects;
    int m_nNumAreas;
    int m_nRedMin;
    int m_nRedMax;
    int m_nGreenMin;
    int m_nGreenMax;
    int m_nBlueMin;
    int m_nBlueMax;
    DECLARE_MESSAGE_MAP ()
};
```

Modify the constructor and OnPaint methods of CLab13Win.cpp:

```
CLab13Win::CLab13Win ()
{
    Create (NULL, "Lab13");
    m_nMessageX = 100;
    m_nMessageY = 100;
    m_nNumRects = 5;
    m_nNumAreas = 3;
    m_nRedMin = 0;
    m_nRedMax = 255;
    m_nGreenMin = 0;
    m_nGreenMax = 0;
    m_nBlueMin = 255;
    m_nBlueMax = 0;
}

afx_msg void CLab13Win::OnPaint ()
{
    CPaintDC dc (this);
    dc.SetBkMode(TRANSPARENT);
    dc.SetTextColor(RGB (255,255,255));
    CRect rect;
    GetClientRect (&rect);
    int height = rect.Height () / m_nNumAreas;
    int width = rect.Width () / m_nNumAreas;
    int deltaX = width / 2 / m_nNumRects;
    int deltaY = height / 2 / m_nNumRects;
    int deltaR = (m_nRedMax - m_nRedMin) / (m_nNumRects - 1);
    int deltaG = (m_nGreenMax - m_nGreenMin) / (m_nNumRects - 1);
    int deltaB = (m_nBlueMax - m_nBlueMin) / (m_nNumRects - 1);
```

```

for (int aX = 0; aX < m_nNumAreas; aX++)
    for (int aY = 0; aY < m_nNumAreas; aY++)
    {
        int ulX = aX * width;
        int ulY = aY * height;
        int lrX = (aX+1) * width;
        int lrY = (aY+1) * height;
        for (int r = 0; r < m_nNumRects; r++)
        {
            int red = m_nRedMin + r * deltaR;
            int green = m_nGreenMin + r * deltaG;
            int blue = m_nBlueMin + r * deltaB;
            CBrush paintBrush;
            paintBrush.CreateSolidBrush (RGB(red, green,
                blue));
            CBrush * pBrushSv = dc.SelectObject
                (&paintBrush);
            dc.Rectangle (ulX + r*deltaX, ulY + r*deltaY,
                lrX - r*deltaX, lrY - r*deltaY);
            dc.SelectObject (pBrushSv);
        }
    }
dc.DrawText("Hello World", CRect (m_nMessageX, m_nMessageY,
    m_nMessageX+80, m_nMessageY+16), DT_LEFT);
}

```

10. Compile and execute the program – you should see:



11. Recompile and execute the program after changing the values of `m_nNumRects`, and `m_nNumAreas`. Try several values for each of these variables. Describe the purpose of each:
  
12. Recompile and execute the program after changing the values `m_nRedMin`, `m_nRedMax`, `m_nGreenMin`, `m_nGreenMax`, `m_nBlueMin`, and `m_nBlueMax = 0`. Try several values for each of these variables. Describe the purpose of each:
  
13. The next set of modifications will allow the user to employ the arrow keys to modify the patterned background for the window. Modify `CLab13Win.h` and `CLab13Win.cpp` as indicated:

Add the function `OnKeyDown` to `CLab13Win.cpp`:

```
afx_msg void CLab13Win::OnKeyDown( UINT nChar, UINT nRepCnt, UINT nFlags )
{
    switch (nChar)
    {
        case 37: // Left arrow key
            m_nNumRects--;
            Invalidate (TRUE);
            break;
        case 38: // Up arrow key
            m_nNumAreas++;
            Invalidate (TRUE);
            break;
        case 39: // Right arrow key
            m_nNumRects++;
            Invalidate (TRUE);
            break;
        case 40: // Down arrow key
            m_nNumAreas--;
            Invalidate (TRUE);
            break;
        default:
            MessageBox ("Key not recognized");
    }
}
```

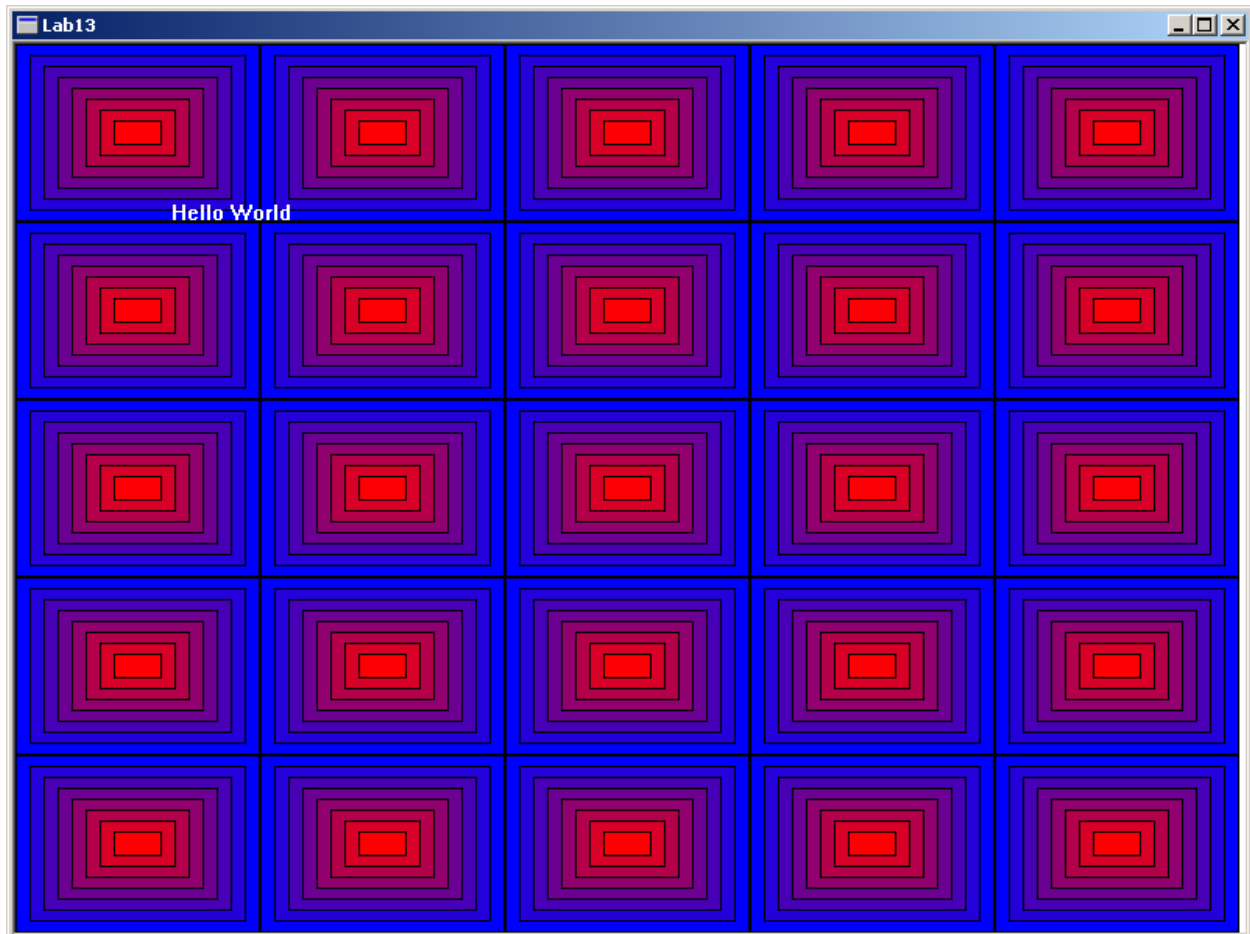
Add the Key down windows message to the message map:

```
BEGIN_MESSAGE_MAP (CLab13Win, CFrameWnd)
    ON_WM_PAINT ( )
    ON_WM_KEYDOWN( )
    ON_WM_MOUSEMOVE( )
END_MESSAGE_MAP ( )
```

Add the prototype for `OnKeyDown` to `CLab13Win.h`:

```
afx_msg void OnKeyDown( UINT nChar, UINT nRepCnt, UINT nFlags );
```

14. Compile and execute the program. After pressing the up arrow (↑) twice and the right arrow (→) three times you should see:



15. Experiment with the up, down, left, and right arrow keys until the program crashes. Modify the program to prevent crashes.
16. When you are convinced that your program is working correctly, tar and zip your folder into a file called as *yourlastnameL13.gz* and drop it into the course dropbox..

# Laboratory Exercise 14 (Inheritance and Polymorphism)

Topics: Hierarchical Class Structures and Polymorphic Containers

Goals: Upon successful completion of this lab you should be able to:

1. Define a hierarchical class structure
2. Define Virtual Functions
3. Write application programs that use polymorphism
4. Create a doubly linked list of pointers to polymorphic objects.
5. Write a program that uses a doubly linked list of pointers to polymorphic objects

Related text sections:

Chapter 14

Chapter 15

Chapter 17

## Laboratory Exercise 14 Instructions (part 1)

1. Create a new folder for Lab 14 and copy the files in `~tiawatts/cs215pickup/Lab14` to your new folder. The files are listed at the end of this lab.

2. Use the makefile to compile the program. Execute the program using the following input:

```
option: a
side: 3
option: b
one leg: 3
the other leg: 4
option: q
```

Your interactive session should look like:

```
Select one of the following options:
  a. Create an Equilateral Triangle
  b. Create a Right Triangle
  q. Exit
a
Enter the length of the side of the equilateral triangle: 3
The area of your 3 sided polygon is 0; its perimeter is 0.
Select one of the following options:
  a. Create an Equilateral Triangle
  b. Create a Right Triangle
  q. Exit
b
Enter the length of one leg of the right triangle: 3
Enter the length of the other leg of the right triangle: 4
The area of your 3 sided polygon is 0; its perimeter is 0.
Select one of the following options:
  a. Create an Equilateral Triangle
  b. Create a Right Triangle
  q. Exit
q
Good bye
```

3. The zero values in the output indicate that the functions to calculate the area and perimeters of the equilateral and right triangles have not been completed. Using the formulas you found for the homework assignment, complete the Area and Perimeter functions for the Equilateral and Right classes.
4. Recompile and execute the program. The correct output should be:  
The area of your 3 sided polygon is 3.89711; its perimeter is 9.  
The area of your 3 sided polygon is 6; its perimeter is 12.
5. Using the equilateral and right triangle classes as a model, complete the implementation of the scalene triangle class. Include a menu option (c) for a scalene triangle. Include a case in the create function for a scalene triangle.
6. Compile and test your modifications.

7. Add two new classes to the polygon heirarchy: Square and Rectangle. These new classes should both be children of the Quadrilateral class. Add options (d and e) to the menu and cases to the create function to incorporate these new classes in the application program.
8. Complete the implementation of the Regular polygon class. Include the Regular polygon class in the implementation program (option f). The interactive session should ask the user first for the number of sides, then for the length of each side.
9. When you are convinced that your polygon class heirarchy is working correctly, copy your well documented .h and .cpp files to the dropbox as *yourlastnameL14.h* and *yourlastnameL14.cpp* respectively.

### makefile

```
lab14 : Lab14.o polygon.o
        g++ -o lab14 Lab14.o polygon.o -g

Lab14.o : Lab14.cpp polygon.h
        g++ -c Lab14.cpp -g

polygon.o : polygon.cpp polygon.h
        g++ -c polygon.cpp -g

clean :
        rm -f core.* *.o lab14
```

### Lab14.cpp

```
#include <iostream>
#include "polygon.h"

char menu ();

int main ()
{
    char option;
    do
    {
        option = menu ();
        Polygon * pp = create (option);
        if (pp)
        {
            std::cout << *pp;
            delete pp;
        }
    } while (option != 'Q');
    return 0;
}

char menu ()
{
    char choice;
    std::cout << "Select one of the following options:\n";
    std::cout << "\ta. Create an Equilateral Triangle\n";
    std::cout << "\tb. Create a Right Triangle\n";
    std::cout << "\tq. Exit\n";
    std::cin >> choice;
    return toupper(choice);
}
```

```

Polygon * create (char choice)
{
    Polygon * pp = NULL;
    switch (choice)
    {
        case 'A':
        {
            int side;
            std::cout << "Enter the length of the side of the "
                << "equilateral triangle: ";
            std::cin >> side;
            pp = new Equilateral (side);
            break;
        }
        case 'B':
        {
            int side1, side2;
            std::cout << "Enter the length of one leg of the "
                << "right triangle: ";
            std::cin >> side1;
            std::cout << "Enter the length of the other leg of the "
                << "right triangle: ";
            std::cin >> side2;
            pp = new Right (side1, side2);
            break;
        }
        case 'Q':
        {
            std::cout << "Good bye\n";
            break;
        }
        default:
            std::cout << "Invalid option - please try again\n";
    }
    return pp;
}

```

## **polygon.h**

```

#ifndef POLYGON_H
#define POLYGON_H

#include <iostream>
using namespace std;

class Polygon
{
public:
    Polygon ();
    Polygon (const Polygon & p);
    ~Polygon ();
    Polygon & operator = (const Polygon & p);
    virtual int Sides () const { return 0; }
    virtual double Area () const { return 0; }
    virtual double Perimeter () const { return 0; }
    friend ostream & operator << (ostream & outs, const Polygon & P);
protected:
    double side;
private:
};

```

```

class Triangle : public Polygon
{
    public:
        Triangle ();
        Triangle (const Triangle & p);
        ~Triangle ();
        Triangle & operator = (const Triangle & p);
        int Sides () const;
    private:
};

class Equilateral : public Triangle
{
    public:
        Equilateral ();
        Equilateral (double S);
        ~Equilateral ();
        Equilateral (const Equilateral & p);
        Equilateral & operator = (const Equilateral & p);
        double Area () const;
        double Perimeter () const;
    private:
};

class Right : public Triangle
{
    public:
        Right ();
        Right (double S1, double S2);
        ~Right ();
        Right (const Right & p);
        Right & operator = (const Right & p);
        double Area () const;
        double Perimeter () const;
    private:
        double side2;
};

class Scalene : public Triangle
{
    public:
        Scalene ();
        Scalene (double S1, double S2, double S3);
        ~Scalene ();
        Scalene (const Scalene & p);
        Scalene & operator = (const Scalene & p);
        double Area () const;
        double Perimeter () const;
    private:
        double side2, side3;
};

class Quadrilateral : public Polygon
{
    public:
        Quadrilateral ();
        Quadrilateral (const Quadrilateral & p);
        ~Quadrilateral ();
        Quadrilateral & operator = (const Quadrilateral & p);
        int Sides () const;
    private:
};

```

```

};

// Add Square and Rectangle classes here

class Regular : public Polygon
{
public:
    Regular ();
    Regular (int N, double S);
    Regular (const Regular & p);
    ~Regular ();
    Regular & operator = (const Regular & p);
    int Sides () const;
    double Area () const;
    double Perimeter () const;
private:
    int num_sides;
};

#endif

```

### **polygon.cpp**

```

include <iostream>
#include <cmath>
#include "polygon.h"
using namespace std;

Polygon::Polygon ()
{
    side = 0;
}

Polygon::Polygon (const Polygon & p)
{
    side = p.side;
}

Polygon::~Polygon ()
{
}

Polygon & Polygon::operator = (const Polygon & p)
{
    side = p.side;
    return * this;
}

ostream & operator << (ostream & outs, const Polygon & P)
{
    outs << "The area of your " << P.Sides()
        << " sided polygon is " << P.Area ()
        << "; its perimeter is " << P.Perimeter() << ".\n";
    return outs;
}

Triangle::Triangle ()
{
    side = 0;
}

Triangle::Triangle (const Triangle & p)
{

```

```

        side = p.side;
    }

Triangle::~Triangle ()
{
}

Triangle & Triangle::operator = (const Triangle & p)
{
    side = p.side;
    return * this;
}

int Triangle::Sides () const
{
    return 3;
}

Equilateral::Equilateral ()
{
    side = 0;
}

Equilateral::Equilateral (double S)
{
    side = S;
}

Equilateral::~Equilateral ()
{
}

Equilateral::Equilateral (const Equilateral & p)
{
    side = p.side;
}

Equilateral & Equilateral::operator = (const Equilateral & p)
{
    side = p.side;
    return * this;
}

double Equilateral::Area () const
{
    // Calculate and return area here
    return 0;
}

double Equilateral::Perimeter () const
{
    // Calculate and return perimeter here
    return 0;
}

Right::Right ()
{
    side = side2 = 0;
}

Right::Right (double S1, double S2)
{
    side = S1;
}

```

```

        side2 = S2;
    }

Right::~~Right ()
{
}

Right::Right (const Right & p)
{
    side = p.side;
    side2 = p.side2;
}

Right & Right::operator = (const Right & p)
{
    side = p.side;
    side2 = p.side2;
    return * this;
}

double Right::Area () const
{
    // Calculate and return area here
    return 0;
}

double Right::Perimeter () const
{
    // Calculate and return perimeter here
    return 0;
}

Scalene::Scalene ()
{
    side = side2 = side3 = 0;
}

Scalene::Scalene (double S1, double S2, double S3)
{
    side = S1;
    side2 = S2;
    side3 = S3;
}

Scalene::~~Scalene ()
{
}

Scalene::Scalene (const Scalene & p)
{
    side = p.side;
    side2 = p.side2;
    side3 = p.side2;
}

Scalene & Scalene::operator = (const Scalene & p)
{
    side = p.side;
    side2 = p.side2;
    side3 = p.side2;
    return * this;
}

```

```

double Scalene::Area () const
{
    // Calculate and return area here
    return 0;
}

double Scalene::Perimeter () const
{
    // Calculate and return perimeter here
    return 0;
}

Quadrilateral::Quadrilateral ()
{
}

Quadrilateral::Quadrilateral (const Quadrilateral & p)
{
}

Quadrilateral & Quadrilateral::operator = (const Quadrilateral & p)
{
    return * this;
}

int Quadrilateral::Sides () const
{
    return 4;
}

Regular::Regular ()
{
}

Regular::Regular (const Regular & p)
{
}

Regular & Regular::operator = (const Regular & p)
{
    return * this;
}

int Regular::Sides () const
{
    return num_sides;
}

double Regular::Area () const
{
    // Calculate and return area here
    return 0;
}

double Regular::Perimeter () const
{
    // Calculate and return perimeter here
    return 0;
}

```

## Laboratory Exercise 14 Instructions (part 2)

1. Copy the file LList2.tmp from your Lab 12 folder to your new folder. Modify the application program (Lab14.cpp) to include your LList2.tmp file. Modify the makefile to include LList2.tmp as a dependency for creating Lab14.o.
2. Instantiate a linked list of polygon pointers and modify the do while loop in the main function of the application:

```
LList2 <Polygon *> PLP;
do
{
    option = menu ();
    Polygon * pp = create (option);
    if (pp)
        PLP.InsertLast (pp);
} while (option != 'Q');
```

3. Add statements after the loop to print the contents of the linked list from beginning to end.
4. Use the makefile to compile the program. Execute the program using the following input:

```
option: a
side: 3
option: b
one leg: 3
the other leg: 4
option: q
```

The output should be:

```
The area of your 3 sided polygon is 3.89711; its perimeter is 9.
The area of your 3 sided polygon is 6; its perimeter is 12.
```

If your output looks more like:

```
0x804c290 0x804c2b0
```

than the correct output, you are printing the addresses of the objects instead of the data in the objects. Fix your program to print the data.

Test the other classes in your polygon class heirarchy.

5. Add a segment of code to print the data in reverse order.
6. Add a segment of code to find and print the total of the areas of the polygons in the list.
7. Add a segment of code to find and print the total of the perimeters of the polygons in the list.
8. When you are convinced that your application program is working correctly, copy your well documented .cpp file to the dropbox as *yourlastnameL14.app*.

## Laboratory Exercise 15 (Ordered Linked Lists)

Topics: Ordered Linked Lists

Goals: Upon successful completion of this lab you should be able to:

1. Insert nodes into the “middle” of a doubly linked list
2. Remove nodes from the “middle” of a doubly linked list
3. Determine if a value is contained in a doubly linked list
4. Use a doubly linked list to store an ordered set
5. Create a new ordered set containing the union of 2 ordered sets
6. Create a new ordered set containing the intersection of 2 ordered sets

Related text sections:

Chapter 17

## Laboratory Exercise 16 Instructions

1. Copy your LList2.tmp file from Lab12 to a new Lab 15 directory.
2. Rename your LList2.tmp file OrderedSet.tmp.
3. Rename the class OrderedSet. Change the template type to ST.
4. Move the methods

```
bool InsertFirst (const ST & value);
bool InsertLast (const ST & value);
bool DeleteFirst ();
bool DeleteLast ();
```

from the public section of the class to the private section of the class.
5. Add the following methods to the public section of the OrderedSet class:

```
void Clear ();
bool IsEmpty () const;
bool IsIn (const ST & value) const;
bool Insert (const ST & value);
bool Delete (const ST & value);
OrderedSet operator + (const OrderedSet & other);
OrderedSet operator * (const OrderedSet & other);
```
6. Implement the Clear method to delete all items in the set.
7. Implement the IsEmpty method to return true if there are no nodes in the set and false if there are nodes in the set.
8. Implement the IsIn method to return true if the specified item is in the set and false if it is not in the set.
9. Implement the Insert method to maintain the nodes of the set in ascending order based on the node data. Duplicate data items are not permitted; if the value is already in the list, the method should simply return false. The Insert method should use the InsertFirst and InsertLast methods from the private section of the class when it is appropriate.
10. Implement the Delete method to remove the specified node from the set. If the value is not in the list, the method should simply return false. The Delete method should use the DeleteFirst and DeleteLast methods from the private section of the class when it is appropriate.
11. Implement the output operator (<<) to list the items in the set in set braces ({ and }), separated by commas. For example: { 2, 4, 5, 7, 10 } .
12. Implement the + operator to return an ordered set containing the union of its two ordered set operands. { 1, 3, 5, 7, 9 } + { 1, 2, 3, 5, 7 } = { 1, 2, 3, 5, 7, 9 }
13. Implement the \* operator to return an ordered set containing the intersection of its two ordered set operands. { 1, 3, 5, 7, 9 } \* { 1, 2, 3, 5, 7 } = { 1, 3, 5, 7 }
14. The full class definition is on the next page:

```

template <class ST>
class OrderedSet
{
    private:
        typedef class Node
        {
            public:
                Node ();
                ST data;
                Node * next;
                Node * prev;
        };

    public:
        typedef class Iterator
        {
            public:
                Iterator ();
                Iterator (Node * NP);
                const ST operator * () const;
                Iterator operator ++ ();
                Iterator operator ++ (int);
                Iterator operator -- ();
                Iterator operator -- (int);
                bool operator == (const Iterator & other) const;
                bool operator != (const Iterator & other) const;
            private:
                Node * current;
        };

        OrderedSet ();
        OrderedSet (const OrderedSet & other);
        ~OrderedSet ();
        OrderedSet & operator = (const OrderedSet & other);
        bool operator == (const OrderedSet & other);
        int Size () const;
        void Clear ();
        bool IsEmpty () const;
        bool IsIn (const ST & value) const;
        bool Insert (const ST & value);
        bool Delete (const ST & value);
        OrderedSet operator + (const OrderedSet & other);
        OrderedSet operator * (const OrderedSet & other);
        void Forward (void function (const ST & param));
        void Backward (void function (const ST & param));
        Iterator begin () const;
        Iterator rbegin () const;
        Iterator end () const;
        Iterator rend () const;
        ST operator [] (const int & index) const;
        friend ostream & operator << >> (ostream & outs, const OrderedSet<ST> & S);

    private:
        Node * first;
        Node * last;
        int size;
        bool InsertFirst (const ST & value);
        bool InsertLast (const ST & value);
        bool DeleteFirst ();
        bool DeleteLast ();
};

```

15. Use the following program to test the Insert method and the output (<<) operator of the OrderedSet class. Before running the program, predict its output:

```
// OrderedSet test program 1
#include <iostream>
#include <cstdlib>
#include "OrderedSet.tmp"

using namespace std;

int main ()
{
    OrderedSet<int> S1;
    cout << "Insert values in set:\n";
    S1.Insert (1);
    S1.Insert (5);
    S1.Insert (3);
    S1.Insert (7);
    S1.Insert (-1);
    cout << "Elements in S1: " << S1 << endl;
    return 0;
}
```

16. Use the following program to test the Size and IsEmpty methods of the OrderedSet class. Before running the program, predict its output:

```
// OrderedSet test program 2
#include <iostream>
#include <cstdlib>
#include "OrderedSet.tmp"

using namespace std;

int main ()
{
    OrderedSet<char> S1;
    cout << "Test if new set is empty:\n";
    if (S1.IsEmpty ())
        cout << "S1 is empty\n";
    else
        cout << "S1 is NOT empty\n";
    cout << "Insert values in set:\n";
    S1.Insert ('c');
    S1.Insert ('x');
    S1.Insert ('r');
    S1.Insert ('z');
    S1.Insert ('a');
    cout << "Test if modified set is empty:\n";
    if (S1.IsEmpty ())
        cout << "S1 is empty\n";
    else
        cout << "S1 is NOT empty\n";
    cout << "Elements in S1: " << S1 << endl;
    cout << "The size of S1 is: " << S1.Size() << endl;
    return 0;
}
```

17. Use the following program to test the Delete, Clear and IsIn methods of the OrderedSet class. Before running the program, predict its output:

```
// OrderedSet test program 3
#include <iostream>
#include <cstdlib>
#include "OrderedSet.tmp"

using namespace std;

int main ()
{
    OrderedSet<int> S1;
    cout << "Insert values 7,12,-3,-4,15,12, and 10 in set:\n";
    S1.Insert (7);
    S1.Insert (12);
    S1.Insert (-3);
    S1.Insert (-4);
    S1.Insert (15);
    S1.Insert (12);
    S1.Insert (10);
    cout << "Elements in S1: " << S1 << endl;
    cout << "The size of S1 is: " << S1.Size() << endl;
    cout << "Test IsIn and delete items found in set:\n";
    if (S1.IsIn (-4))
        cout << "-4 is in S1\n";
    else
        cout << "-4 is not in S1\n";
    S1.Delete(-4);
    if (S1.IsIn (5))
        cout << "5 is in S1\n";
    else
        cout << "5 is not in S1\n";
    S1.Delete(5);
    if (S1.IsIn (7))
        cout << "7 is in S1\n";
    else
        cout << "7 is not in S1\n";
    S1.Delete(7);
    if (S1.IsIn (15))
        cout << "15 is in S1\n";
    else
        cout << "15 is not in S1\n";
    S1.Delete(15);
    cout << "Elements in S1: " << S1 << endl;
    cout << "Test clear:\n";
    S1.Clear();
    cout << "Test attempt to delete item from empty set:\n";
    if (S1.IsIn (7))
        cout << "7 is in S1\n";
    else
        cout << "7 is not in S1\n";
    S1.Delete(7);
    return 0;
}
```

18. Use the following program to test some of the iterator methods of the OrderedSet class. Before running the program, predict its output:

```
// OrderedSet test program 4
#include <iostream>
#include <cstdlib>
#include <string>
#include "OrderedSet.tmp"

using namespace std;

int main ()
{
    OrderedSet<string> S1;
    cout << "Insert values in set:\n";
    S1.Insert ("Hello");
    S1.Insert ("CS 215");
    S1.Insert ("students");
    cout << "Elements in S1: " << S1 << endl;
    cout << "Testing begin, rbegin, and *:\n";
    cout << "The contents of begin() is: " << *S1.begin() << endl;
    cout << "The contents of rbegin() is: " << *S1.rbegin() << endl;
    return 0;
}
```

19. Use the following program to test more of the iterator methods and some of the exception handling of the OrderedSet class. Before running the program, predict its output:

```
// OrderedSet test program 5
#include <iostream>
#include <cstdlib>
#include "OrderedSet.tmp"

using namespace std;

int main ()
{
    OrderedSet<int> S1;
    S1.Insert (7);
    S1.Insert (12);
    S1.Insert (-3);
    S1.Insert (-4);
    cout << "Elements in S1: " << S1 << endl;
    OrderedSet<int>::Iterator i = S1.begin();
    cout << "Testing iterator = begin(); it contains: " << *i << endl;
    cout << "Testing ++ operators:\n";
    cout << *i++ << ' ';
    cout << *++i << ' ';
    cout << *i << endl;
    for (i = S1.begin(); i != S1.end(); i++)
        cout << "*i = " << *i << ' ';
    cout << endl;
    cout << "Testing ++ exception handling:\n";
    i = S1.end();
    i++;
    return 0;
}
```

20. Use the following program to test more of the iterator methods and some of the exception handling of the OrderedSet class. Before running the program, predict its output:

```
// OrderedSet test program 6
#include <iostream>
#include <cstdlib>
#include "OrderedSet.tmp"

using namespace std;

int main ()
{
    OrderedSet<int> S1;
    S1.Insert (7);
    S1.Insert (12);
    S1.Insert (-3);
    S1.Insert (-4);
    cout << "Elements in S1: " << S1 << endl;
    OrderedSet<int>::Iterator i = S1.rbegin();
    cout << "Testing iterator = rbegin(); it contains: " << *i << endl;
    cout << "Testing -- operators:\n";
    cout << *--i << ' ';
    cout << *i-- << ' ';
    cout << *i << endl;
    for (i = S1.rbegin(); i != S1.rend(); i--)
        cout << "*i = " << *i << ' ';
    cout << endl;
    cout << "Testing -- exception handling:\n";
    i = S1.end();
    i--;
    return 0;
}
```

21. Use the following program to test the bracket ( [ ] ) operator and some of the exception handling of the OrderedSet class. Before running the program, predict its output:

```
// OrderedSet test program 7
#include <iostream>
#include <cstdlib>
#include "OrderedSet.tmp"

using namespace std;

int main ()
{
    OrderedSet<int> S1;
    S1.Insert (6);
    S1.Insert (-5);
    S1.Insert (4);
    cout << "Elements in S1: " << S1 << endl;
    cout << "Testing [] operator:\n";
    int sum = 0;
    for (int i = 0; i < S1.Size(); i++)
        sum = sum + S1[i];
    cout << "The sum of the integers is: " << sum << endl;
    cout << "The contents of S1[-2] is: ";
    cout << S1[-2] << endl;
    return 0;
}
```

22. Use the following program to test the union (+) and intersection (\*) operators of the OrderedSet class. Before running the program, predict its output:

```
// OrderedSet test program 8
#include <iostream>
#include <cstdlib>
#include "OrderedSet.tmp"

using namespace std;

int main ()
{
    OrderedSet<int> S1;
    cout << "Insert values 7,12,-3,-4,15,12, and 10 in S1:\n";
    S1.Insert (7);
    S1.Insert (12);
    S1.Insert (-3);
    S1.Insert (-4);
    S1.Insert (15);
    S1.Insert (12);
    S1.Insert (10);
    OrderedSet<int> S2;
    cout << "Insert values -1,2,3,-14,5,12 and 0 in S2:\n";
    S2.Insert (-1);
    S2.Insert (2);
    S2.Insert (3);
    S2.Insert (-14);
    S2.Insert (5);
    S2.Insert (12);
    S2.Insert (0);
    cout << "Elements in S1: " << S1 << endl;
    cout << "Elements in S2: " << S2 << endl;
    OrderedSet<int> S3;
    cout << "Testing S3 = S1 + S2\n";
    S3 = S1 + S2;
    cout << "Elements in S3: " << S3 << endl;
    cout << "Testing S3 = S2 + S1\n";
    S3 = S2 + S1;
    cout << "Elements in S3: " << S3 << endl;
    cout << "Testing S3 = S1 * S2\n";
    S3 = S1 * S2;
    cout << "Elements in S3: " << S3 << endl;
    cout << "Testing S3 = S2 * S1\n";
    S3 = S2 * S1;
    cout << "Elements in S3: " << S3 << endl;
    return 0;
}
```

23. When you are convinced that your OrderedSet template is working correctly, copy your well documented .tmp file to the dropbox as yourlastnameL15.tmp.

## Laboratory Exercise 16 (More MFC)

Topics: MFC and Polymorphic Containers

Goals: Upon successful completion of this lab you should be able to:

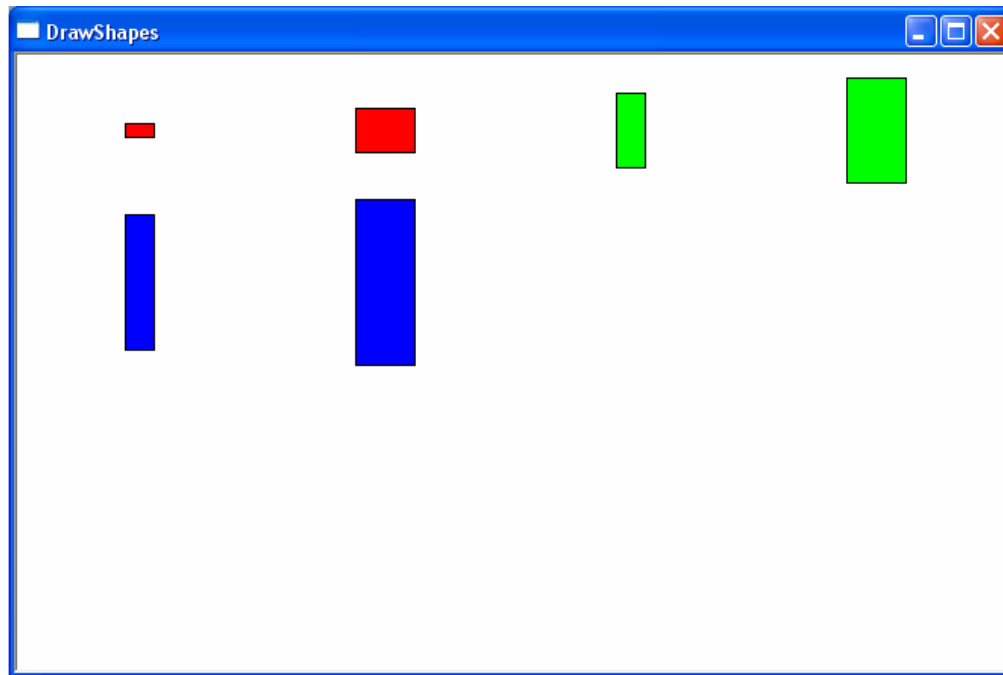
7. Write a simple windows program using the MFC Frame Window and Dialog classes
8. Include a doubly linked list of pointers to polymorphic objects in an MFC program

Related text sections:

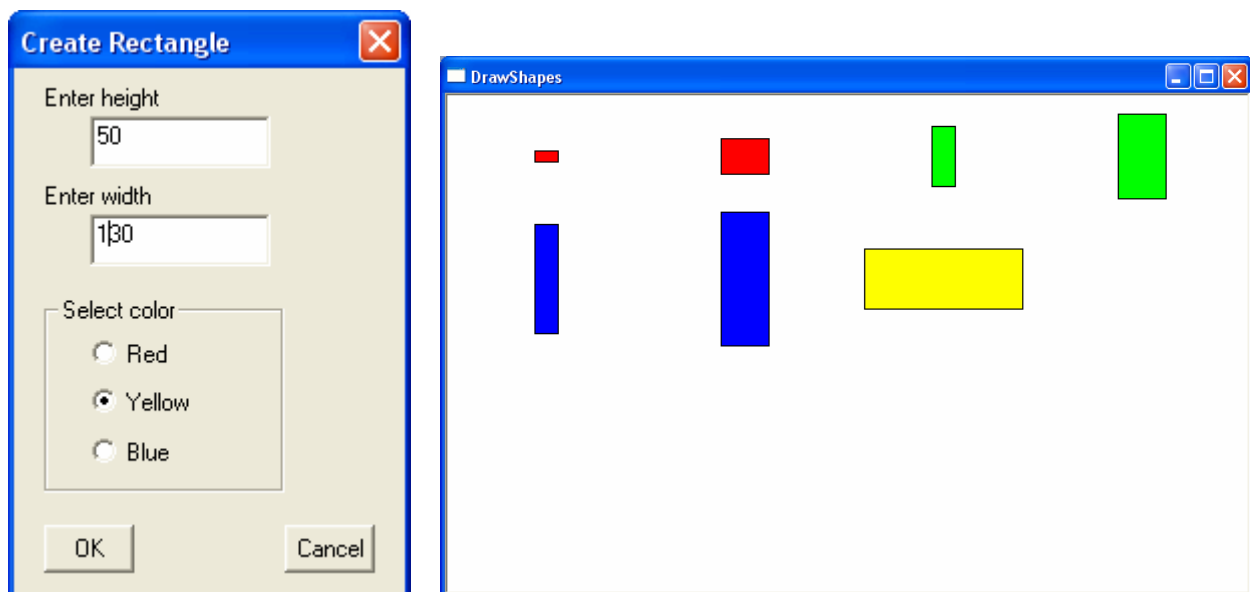
- Chapter 5 (Introduction to MFC)
- Chapter 14
- Chapter 15
- Chapter 17

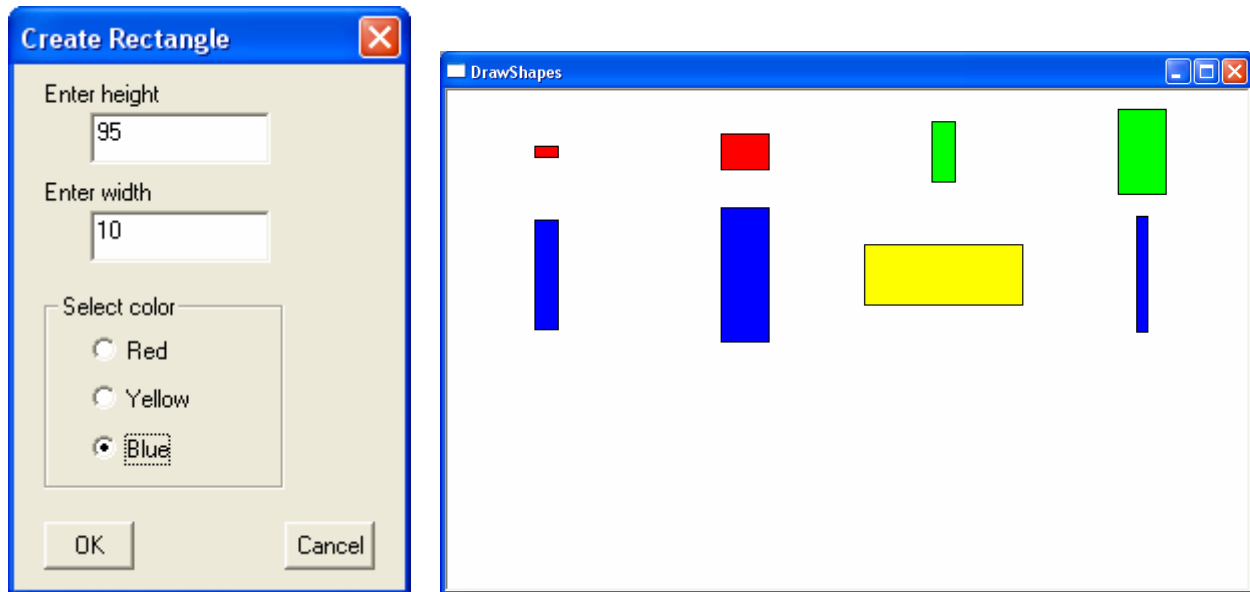
## Laboratory Exercise 16 Instructions

23. Create a new Visual C++ application project called Lab16.
24. Copy the header (.h) and implementation (.cpp) files from ~tiawatts/cs215pickup/Lab16 to the folder created for your Lab16 project.
25. Compile and execute the program. You should see:



26. Press any arrow key to popup the "Create Rectangle" dialog box to add more rectangles to the display:





27. Copy your LList2.tmp template file to a new header file called LList2.h in your Lab16 project. Modify the output (<<) prototype as indicated:
 

```
// friend ostream & operator << >> (ostream & outs, const LList2<LT> & L);
   friend ostream & operator << (ostream & outs, const LList2<LT> & L);
```
28. Replace the array m\_rectangles in the class CShapesDocument with a linked list:
 

```
CRectangle * m_rectangles [16]; →
   LList2 <CRectangle> m_rectangles;
```
29. Modify CShapesDocument.cpp as needed to accommodate the array to linked list replacement in the previous step.
  - a. Each call to the Add function in the class CShapeDocument should use InsertLast to add the new shape to the end of the m\_rectangles linked list.
  - b. The four loop in the Paint function should use an iterator to traverse the m\_rectangles linked list.
  - c. Currently the CShapeDocument Paint function paints the rectangles in a 4x4 grid in the device context window. Only 16 rectangles can be accommodated. Modify the Add function to accept more than 16 rectangles (theoretically it should accept an infinitely many rectangles) and to calculate new values for m\_nCols and m\_nRows such that if there are N rectangles in the list, m\_nCols and m\_nRows are each less than or equal to the ceiling of the square root of N, and m\_nRows is less than or equal to m\_nCols.
30. Compile and execute the modified program. Add more rectangles to the display to test your modified Add and Paint functions.
31. Create a copy of your Polygon class files from Labs 13 and 14. Modify the class names to follow the standards recommended for MFC:
  - a. Polygon → CPolygon
  - b. Triangle → CTriangle
  - c. Equilateral → CEquilateral
  - d. etc.

10. Add a “Paint” function to your CPolygon class. The Paint function for CRectangle is listed below; simplified CPolygon class currently used in Lab 16 is listed at the end of this Lab exercise. Use it as a model.

```
void CRectangle::Paint (CPaintDC & dc, int midX, int midY)
{
    CBrush colorBrush;
    colorBrush.CreateSolidBrush (color);
    CBrush * pBrushSv = dc.SelectObject (&colorBrush);
    dc.Rectangle (midX - side2 / 2, midY - side / 2,
                 midX + side2 / 2, midY + side / 2);
    dc.SelectObject (pBrushSv);
}
```

32. Add 3 new Create Dialogs to your project:

- a. Equilateral Triangle
- b. Right Triangle
- c. Regular Polygon

33. Modify the switch statement in the function OnKeyDown in the class CShapeWin so that each arrow key adds a different shape to the document.

34. Add Paint functions to the classes CEquilateral, CRight, and CRegular.

35. Replace the linked list m\_rectangles in CShapeDocument with a linked list of polygons:

```
LList2 <CPolygon> m_polygons;
```

36. Test your modified program. You should be able to add and display a variety of shapes using the arrow keys and the new dialogs.

37. When you are convinced that your program is working correctly, tar and zip your folder into a file called as *yourlastnameL16.tgz* and drop it into the course dropbox..

```

#include <iostream>
using namespace std;

template <class LT> class LList2;

template <class LT> ostream & operator << (ostream & outs, const LList2<LT> & L);

template <class LT>
class LList2
{
private:
    typedef class LNode
    {
        public:
            LNode ();
            LT data;
            LNode * next;
            LNode * prev;
    };

public:
    typedef class Iterator
    {
        public:
            Iterator ();
            Iterator (LNode * NP);
            LT & operator * () const;
            Iterator operator ++ ();
            Iterator operator ++ (int);
            Iterator operator -- ();
            Iterator operator -- (int);
            bool operator == (const Iterator & other) const;
            bool operator != (const Iterator & other) const;
        private:
            LNode * current;
    };
    LList2 ();
    LList2 (const LList2 & other);
    ~LList2 ();
    LList2 & operator = (const LList2 & other);
    bool operator == (const LList2 & other);
    int Size () const;
// friend ostream & operator << >> (ostream & outs, const LList2<LT> & L);
friend ostream & operator << (ostream & outs, const LList2<LT> & L);
    bool InsertFirst (const LT & value);
    bool InsertLast (const LT & value);
    bool DeleteFirst ();
    bool DeleteLast ();
    void Forward (void function (const LT & param));
    void Backward (void function (const LT & param));
    Iterator begin () const;
    Iterator rbegin () const;
    Iterator end () const;
    Iterator rend () const;
    LT & operator [] (const int & index) const;
private:
    LNode * first;
    LNode * last;
    int size;
};

```

```

// File: CShapesApp.h

#include <afxwin.h>
#include "CShapesWin.h"

class CShapesApp : public CWinApp
{
    public:
        BOOL InitInstance ();
};

// File: CShapesApp.cpp

#include <afxwin.h>
#include "CShapesApp.h"

BOOL CShapesApp::InitInstance ()
{
    m_pMainWnd = new CShapesWin();
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();

    return TRUE;
}

CShapesApp shapesApp;

// File: CShapesWin.h

#include <afxwin.h>
#include "CShapesDocument.h"

class CShapesWin : public CFrameWnd
{
    public:
        CShapesWin ();
        afx_msg void OnPaint ();
        afx_msg void OnKeyDown( UINT nChar, UINT nRepCnt, UINT nFlags );
    private:
        CShapesDocument m_doc;
        DECLARE_MESSAGE_MAP ()
};

// File: CShapesWin.cpp

#include <afxwin.h>
#include "CShapesWin.h"
#include "CRectDialog.h"
#include "shapeIds.h"

CShapesWin::CShapesWin ()
{
    Create (NULL, "DrawShapes");
}

```

```

afx_msg void CShapesWin::OnPaint ()
{
    CPaintDC dc (this);
    CRect rect;
    GetClientRect (&rect);
    m_doc.Paint (dc, rect);
}

afx_msg void CShapesWin::OnKeyDown( UINT nChar, UINT nRepCnt, UINT nFlags )
{
    switch (nChar)
    {
        case 37: // Left arrow key
        case 38: // Up arrow key
        case 39: // Right arrow key
        case 40: // Down arrow key
            {
                CRectDialog rectDialog;
                if (rectDialog.DoModal() == IDOK)
                    if (m_doc.Add (new CRectangle (rectDialog.m_nHeight,
                                                    rectDialog.m_nWidth, rectDialog.m_Color)) == TRUE)
                        Invalidate (TRUE);
                break;
            }
        default:
            MessageBox ("Key not recognized");
    }
}

BEGIN_MESSAGE_MAP (CShapesWin, CFrameWnd)
    ON_WM_PAINT ()
    ON_WM_KEYDOWN( )
END_MESSAGE_MAP ()

// File:  CShapesDocument.h

#include <afxwin.h>
#include "CPolygon.h"
#include "LList2.h"

const int c_nMax = 16;

class CShapesDocument : public CDocument
{
public:
    CShapesDocument ();
    void Paint (CPaintDC & dc, CRect & winArea);
    BOOL Add (CRectangle * rect);
    CRectangle * m_rectangles [16];
    int m_nRCount;
    int m_nRows;
    int m_nCols;
};

```

```

// File: CShapesDocument.cpp

#include "CShapesDocument.h"

CShapesDocument::CShapesDocument ()
{
    m_nRows = 4;
    m_nCols = 4;

    m_nRCount = 0;
    for (int i = 0; i < c_nMax; i++)
    {
        m_rectangles[i] = NULL;
    }

    // For testing
    m_rectangles [0] = new CRectangle (10, 20, RGB(255,0,0));
    m_rectangles [1] = new CRectangle (30, 40, RGB(255,0,0));
    m_rectangles [2] = new CRectangle (50, 20, RGB(0,255,0));
    m_rectangles [3] = new CRectangle (70, 40, RGB(0,255,0));
    m_rectangles [4] = new CRectangle (90, 20, RGB(0,0,255));
    m_rectangles [5] = new CRectangle (110, 40, RGB(0,0,255));
    m_nRCount = 6;
}

BOOL CShapesDocument::Add (CRectangle * rect)
{
    if (m_nRCount >= c_nMax)
        return FALSE;
    m_rectangles [m_nRCount] = rect;
    m_nRCount++;
    return TRUE;
}

void CShapesDocument::Paint (CPaintDC & dc, CRect & winArea)
{
    int halfRow = winArea.Height() / (2 * m_nRows);
    int halfCol = winArea.Width() / (2 * m_nCols);
    int p = 0;
    for (int r = 0; p < m_nRCount && r < m_nRows; r++)
        for (int c = 0; p < m_nRCount && c < m_nCols; c++)
            m_rectangles[p++]->Paint (dc, (2 * c + 1) * halfCol, (2 * r
+ 1) * halfRow);
}

// File: shapeIds.h

#define IDC_OK          2000
#define IDC_Cancel     2001
#define IDC_Height     2002
#define IDC_Width      2003
#define IDC_Red        2004
#define IDC_Yellow     2005
#define IDC_Blue       2006

```

```

// File: Rectangle.rc

#include <afxres.h>
#include "shapeIds.h"

Rectangle DIALOG 50,50,130,130
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU

CAPTION "Create Rectangle"
{
    LTEXT        "Enter height", IDC_STATIC, 20, 15, 50, 8
    EDITTEXT     IDC_Height, 20, 25, 60, 16
    LTEXT        "Enter width", IDC_STATIC, 20, 45, 50, 8
    EDITTEXT     IDC_Width, 20, 55, 60, 16
    GROUPBOX     "Select color", IDC_STATIC, 20, 75, 50, 50
    AUTORADIOBUTTON "Red", IDC_Red, 30, 60, 50, 16, WS_GROUP
    AUTORADIOBUTTON "Yellow", IDC_Yellow, 30, 70, 50, 16
    AUTORADIOBUTTON "Blue", IDC_Blue, 30, 80, 50, 16
    PUSHBUTTON   "OK", IDC_OK, 20,100,30,15, NOT WS_TABSTOP
    PUSHBUTTON   "Cancel", IDC_Cancel, 80,100,30,15, NOT WS_TABSTOP
}

// File: CRectDialog.h

#include <afxwin.h>

class CRectDialog : public CDialog
{
public:
    CRectDialog ();
    afx_msg void OnOK ();
    afx_msg void OnCancel ();
    int m_nHeight;
    int m_nWidth;
    COLORREF m_Color;
private:
    DECLARE_MESSAGE_MAP ()
};

// File: CRectDialog.cpp

#include "CRectDialog.h"
#include "shapeIds.h"

const int TEXT_MAX = 20;

CRectDialog::CRectDialog () : CDialog ("Rectangle")
{
    m_nHeight = 0;
    m_nWidth = 0;
}

```

```

afx_msg void CRectDialog::OnOK ()
{
    char editText [TEXT_MAX + 1];
    CEdit * heightEdit = (CEdit *) (GetDlgItem (IDC_Height));
    heightEdit->GetWindowText (editText, TEXT_MAX);
    m_nHeight = atoi (editText);
    if (m_nHeight <= 0)
    {
        EndDialog (!IDOK);
        return;
    }
    CEdit * widthEdit = (CEdit *) (GetDlgItem (IDC_Width));
    widthEdit->GetWindowText (editText, TEXT_MAX);
    m_nWidth = atoi (editText);
    if (m_nWidth <= 0)
    {
        EndDialog (!IDOK);
        return;
    }
    int color = GetCheckedRadioButton (IDC_Red, IDC_Blue);
    switch (color)
    {
    case IDC_Red:
        m_Color = RGB (255, 0, 0);
        break;
    case IDC_Yellow:
        m_Color = RGB (255, 255, 0);
        break;
    case IDC_Blue:
        m_Color = RGB (0, 0, 255);
        break;
    default:
        m_Color = RGB (255, 255, 255);
    }
    EndDialog (IDOK);
}

afx_msg void CRectDialog::OnCancel ()
{
    m_nHeight = 0;
    m_nWidth = 0;
    EndDialog (!IDOK);
}

BEGIN_MESSAGE_MAP (CRectDialog, CDialog)
    ON_COMMAND (IDC_OK, OnOK)
    ON_COMMAND (IDC_Cancel, OnCancel)
END_MESSAGE_MAP ()

```

```

// File: CPolygon.h

#ifndef CPolygon_H
#define CPolygon_H

#include <iostream>
#include <afxwin.h>

using namespace std;

class CPolygon
{
public:
    CPolygon ();
    CPolygon (const CPolygon & p);
    ~CPolygon ();
    CPolygon & operator = (const CPolygon & p);
    virtual int Sides () const { return 0; }
    virtual double Area () const { return 0; }
    virtual double Perimeter () const { return 0; }
    friend ostream & operator << (ostream & outs, const CPolygon & P);
    virtual void Paint (CPaintDC & dc, int midX, int midY) {}
protected:
    double side;
    COLORREF color;
private:
};

class CQuadrilateral : public CPolygon
{
public:
    CQuadrilateral ();
    CQuadrilateral (const CQuadrilateral & p);
    ~CQuadrilateral ();
    CQuadrilateral & operator = (const CQuadrilateral & p);
    int Sides () const;
private:
};

class CRectangle : public CQuadrilateral
{
public:
    CRectangle ();
    CRectangle (double L, double W);
    CRectangle (double L, double W, COLORREF C);
    ~CRectangle ();
    CRectangle (const CRectangle & p);
    CRectangle & operator = (const CRectangle & p);
    double Area () const;
    double Perimeter () const;
    void Paint (CPaintDC & dc, int midX, int midY);
private:
    double side2;
};

#endif

```

```

// File: CPolygon.cpp

#include <iostream>
#include <cmath>
#include "CPolygon.h"
using namespace std;

#define M_PI 3.14159

CPolygon::CPolygon ()
{
    side = 0;
    color = RGB (255, 0, 0);
}

CPolygon::CPolygon (const CPolygon & p)
{
    side = p.side;
    color = p.color;
}

CPolygon::~CPolygon ()
{
}

CPolygon & CPolygon::operator = (const CPolygon & p)
{
    side = p.side;
    return * this;
}

ostream & operator << (ostream & outs, const CPolygon & P)
{
    outs << "The area of your " << P.Sides()
        << " sided CPolygon is " << P.Area ()
        << "; its perimeter is " << P.Perimeter() << ".\n";
    return outs;
}

CQuadrilateral::CQuadrilateral ()
{
}

CQuadrilateral::CQuadrilateral (const CQuadrilateral & p)
{
}

CQuadrilateral::~CQuadrilateral ()
{
}

CQuadrilateral & CQuadrilateral::operator = (const CQuadrilateral & p)
{
    return * this;
}

```

```

int CQuadrilateral::Sides () const
{
    return 4;
}

CRectangle::CRectangle ()
{
    side = 0;
}

CRectangle::CRectangle (double L, double W)
{
    side = L;
    side2 = W;
}

CRectangle::CRectangle (double L, double W, COLORREF C)
{
    side = L;
    side2 = W;
    color = C;
}

CRectangle::~~CRectangle ()
{
}

CRectangle::CRectangle (const CRectangle & p)
{
    side = p.side;
    side2 = p.side2;
    color = p.color;
}

CRectangle & CRectangle::operator = (const CRectangle & p)
{
    side = p.side;
    side2 = p.side2;
    color = p.color;
    return * this;
}

double CRectangle::Area () const
{
    // Calculate and return area here
    return side * side2;
}

double CRectangle::Perimeter () const
{
    // Calculate and return perimeter here
    return 2 * side + 2 * side2;
}

```

```
void CRectangle::Paint (CPaintDC & dc, int midX, int midY)
{
    CBrush colorBrush;
    colorBrush.CreateSolidBrush (color);
    CBrush * pBrushSv = dc.SelectObject (&colorBrush);
    dc.Rectangle (midX - side2 / 2, midY - side / 2,
                 midX + side2 / 2, midY + side / 2);
    dc.SelectObject (pBrushSv);
}
```

# Exam 1 Review Questions

1. Define each of the following terms:
  - a. program
  - b. function
  - c. case sensitive
  - d. declaration
  - e. variable
  - f. initialization
  - g. literal
  - h. scope
  - i. type casting
  - j. order of operator evaluation
  - k. order of operand evaluation
  - l. namespace
  - m. preprocessor
  - n. conditional operator expression
  - o. constant pass by reference
  - p. overloading
  - q. multi-dimensional arrays
  - r. out of range error
  - s. structs
  - t. classes
  
2. Describe each of the following relationships:
  - a. key vs. reserved words
  - b. C-strings vs C++ strings
  - c. global vs. local definitions
  - d. cout vs. cerr
  - e. break vs. continue
  - f. exit vs. return
  - g. pass by value vs. pass by reference
  - h. preconditions vs. post conditions
  - i. static vs. dynamic arrays
  - j. default vs. copy constructors
  
3. What is structured programming? Describe the three types of control structures used in structured programming.
  
4. What is short circuit evaluation? Why is short circuit evaluation important?
  
5. Describe the purpose of each of the following:
  - a. constructor
  - b. destructor
  - c. copy constructor
  - d. assignment operator
  - e. friend function

## Exam 2 Review Questions

1. What is a "memory leak"? How does a memory leak occur? Why are memory leaks dangerous?
2. What is a "dangling pointer"? How does a dangling pointer occur? Why are dangling pointers dangerous?
3. What is a constructor? When is a constructor called?
4. What is a destructor? When is a destructor called?
5. How do the copy constructor and the assignment operator for a class differ? Why? Give an example of a call to a copy constructor and a call to an assignment operator.
6. What is the purpose of operator overloading?
7. What is a friend function or operator? How does a friend function differ from member function?
8. Why are the input (>>) and output (<<) operators overloaded as "friends"?
9. Is it possible to overload the assignment operator as a friend?
10. Is it possible to overload an addition/concatenation operator as a friend?
11. What is a template function?
12. What is a template class?
13. What is the purpose of a template function or class?
14. Given the following declaration: `float * A;` write the code to dynamically allocate an array with E entries for A.
15. Write the code to release the memory allocated for A.
16. Given the following declaration: `int ** M;` write the code to dynamically allocate a two dimensional array with E rows of F entries for M.
17. Write the code to release the memory allocated for M.
18. What is a linked list?
19. What are the benefits of using a linked list instead of a single dimension array?

20. What are the benefits of using a single dimension array instead of a linked list?

21. Use the follow description of a singly linked list class to answer parts a - ?.

```
class LList
{
    private:
        class Lnode
        {
            public:
                Lnode ();
                int data;
                Lnode * next;
        };
    public:
        LList ();
        LList (const LList & other);
        ~LList ();
        LList & operator = (const LList & other);
        bool InsertFirst (const int & value);
        bool DeleteFirst (int & value);
    private:
        Lnode * first;
};
```

- a. Why is the Lnode class a private attribute of the LList class?
- b. Why are the functions and attributes of Lnode all public?
- c. Why is there no copy constructor, destructor, or assignment operator associated with Lnode?
- d. Why is there a copy constructor, a destructor, and an assignment operator associated with LList?
- e. Write the implementation of Lnode();
- f. Write the implementation of LList();
- g. Write the implementation of LList(const LList & other);
- h. Write the implementation of ~LList();
- i. Write the implementation of LList & operator = (const LList & other);
- j. What is wrong with the following code segment?  

```
for (Lnode * n = first; n->next != NULL; n = n->next)
    cout << n->data <<
        " is not the last value in the list.\n";
```
- k. How would you fix the code in part j to print all but the last value in the list?

22. Modify the class for part 21 to be a template class.

23. Modify the class for part 22 to be a doubly linked list class.

## Final Review Questions

1. What is a makefile? Why do we use makefiles?
2. What are the benefits of separate compilation? What are its drawbacks?
3. Why is a template class not divided into a header (.h) and an implementation (.cpp) file for separate compilation?
4. What factors might prevent a template instantiation from compiling?
5. In the following friend prototype belonging to class Ctype, what is the purpose of the <X>?  
`friend ostream operator << >> (ostream& outs, const Ctype<X> & obj);`
6. What are the benefits of linked lists? The drawbacks?
7. Why is a doubly linked list more powerful than a singly linked list?
8. Under what conditions is a dynamic array a better data structure than a linked list?
9. What is an exception? Describe the function of each part of the try/throw/catch exception handling methodology.
10. What is an iterator?
11. How does the pre ++ operator differ from the post ++ operator defined for an iterator class?
12. Under what conditions would it be appropriate to have an iterator method throw an exception?
13. In a class that contains an iterator, what do the methods begin() and end() do?
14. What is the purpose of a dereferencing (\*) operator in an iterator class?
15. What does the acronym STL stand for? What is the STL? What is the purpose of the STL?
16. Given the following prototype in template class Ctype:  
`void Apply (void What (const T & data) const);`  
and the following declaration and call in main,  
`Ctype<int> O;`  
`O.Apply (funky);`  
write the prototype of funky.

17. Use the following (incomplete) node and list class descriptions to answer questions a, b, c, and d.

```
class list
{
    private:
        class node
        {
            public:
                node * next;
                node * prev;
                int data;
        }
        node * first;
        node * last;
    public:
        // Will insert "v" as the second value in the list
        void InsertSecond (const int & v);
        // Will delete the second value from the list
        void DeleteSecond ();
};
```

- a. Code the implementation of InsertSecond.
  - b. In what situations might InsertSecond throw an exception?
  - c. Code the implementation of DeleteSecond.
  - d. In what situations might DeleteSecond throw an exception?
18. What is a union? How does a union differ from a struct?
19. What is inheritance? What are its benefits? What are its drawbacks?
20. What is a derived class?
21. What is a virtual function?
22. What is polymorphism?
23. Describe the public, private, and protected sections of a class.
24. A stack is a “Last In First Out” data structure. New items are “pushed” on to the top of the stack, and the only item that can be “popped” from the stack is the item most recently “pushed” onto the stack. What changes would you make to your LList2 template class to use it to implement a stack?
25. A queue is a “First In First Out” data structure. New items are “inserted” on to the end of the queue, and the only item that can be “removed” from the queue is the item at the front of the queue. What changes would you make to your LList2 template class to use it to implement a queue?

26. Use the following (incomplete) node and list class descriptions to answer questions a – f.

```
class list
{
    private:
        class node
        {
            public:
                node * next;
                node * prev;
                int size;
                // data will hold a dynamic array of
                // "size" integers.
                int * data;
                node ();
                node (int Size);
                ~node ();
        }
        node * first;
        node * last;
    public:
        list ();
        ~list ();
        friend ostream & operator << (ostream & outs,
                                        const list & L);
};
```

- Draw an illustration of a list maintained in this data structure.
- Write the implementation of node ().
- Write the implementation of node (int Siize); where Size is the desired size of the dynamic array.
- Write the implementation of ~node();
- Write the implementation of list ();
- Write the implementation of ~list ();
- Write the implementation of the output operator for the list. The output operator should write the data for each node on a separate line starting with the data in the first node of the list.

27. What is the MFC? What is its purpose?

28. How does “event” programming differ from “procedural” programming?

29. What function is called to update the contents of an MFC Frame Window?

30. What is a “modal” dialog?

31. Why is there no “main” function in an MFC program?

32. What is the purpose of and MFC message map?

Homework 1  
 Code Analysis Homework Assignment

This homework assignment is designed to give you practice at understanding and following C++ code.

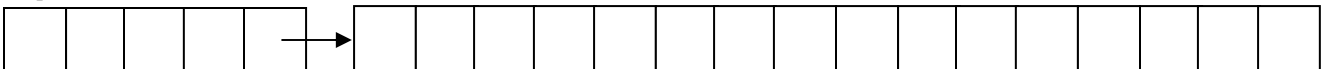
Step 1: TURN OFF YOUR COMPUTER!

Step 2: Use the code distributed with this assignment to determine the affect of each line of this program on the List object IL. Following each line of code is a block diagram of IL; fill in each of the appropriate values in IL to illustrate the modifications made to IL by the method called on the line of code. Use a question mark (?) to indicate an unknown (garbage) value.

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

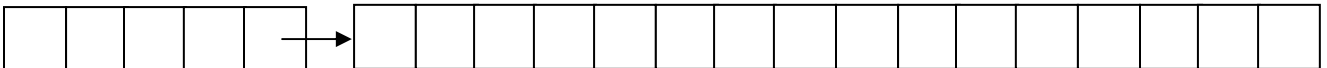
```
void main ()
{
    int i;
    List <int> IL (6);
```

Cap. Used First Last Data



```
cin >> IL; // Keyboard input: 1 2 3 4 5 6 7
```

Cap. Used First Last Data



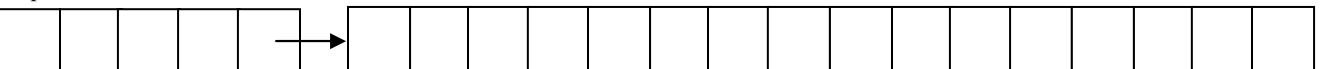
```
IL.Method4(i);
```

Cap. Used First Last Data



```
IL.Method1(7);
```

Cap. Used First Last Data

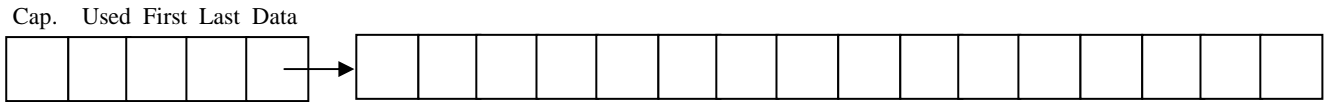


```
IL.Method4(i);
```

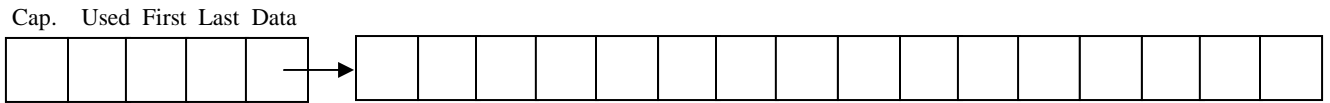
Cap. Used First Last Data



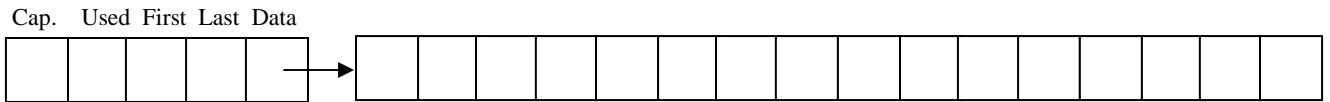
```
IL.Method1(8);
```



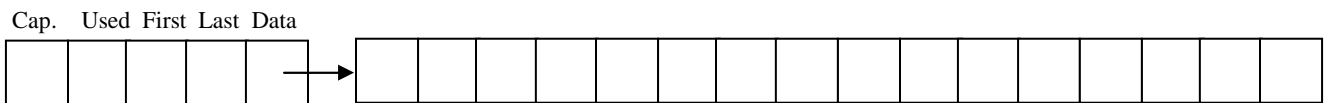
IL.Method2(9);



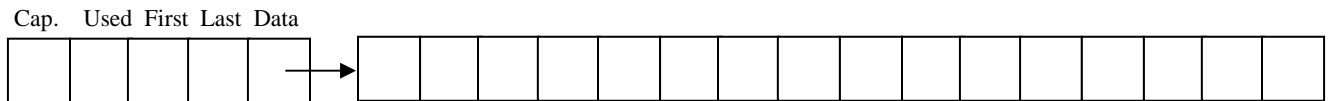
IL.Method2(10);



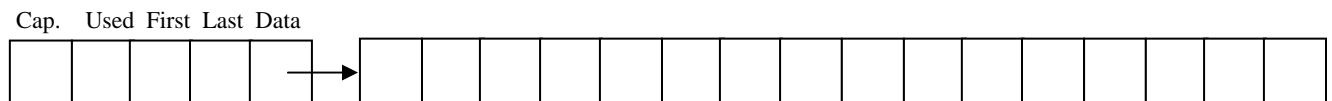
IL.Method2(11);



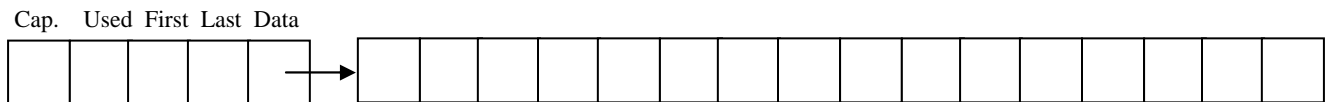
IL.Method2(12);



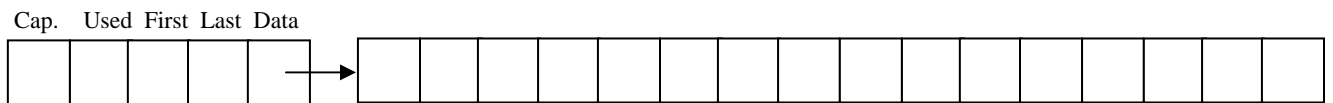
IL.Method2(13);



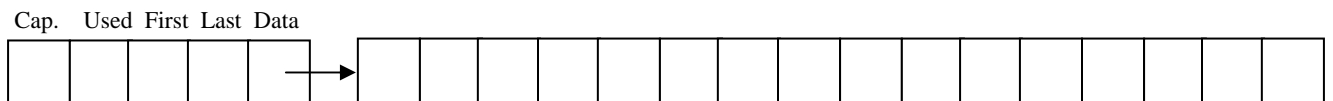
IL.Method4(i);



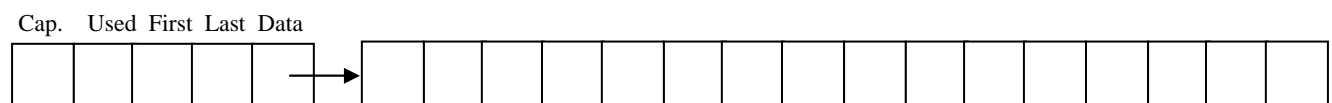
IL.Method3(i);



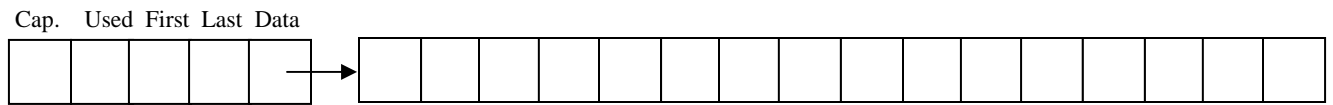
IL.Method4(i);



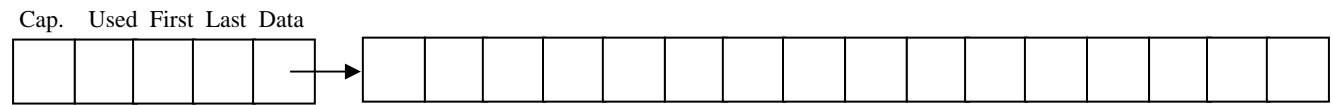
IL.Method3(i);



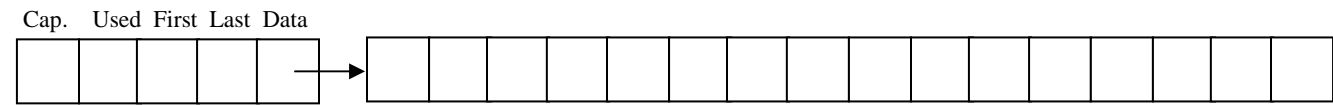
IL.Method3(i);



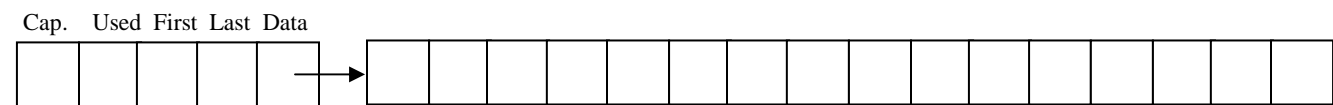
`IL.Method4(i);`



`IL.Method3(i);`



`IL.Method4(i);`



`cout << IL << endl;`

}

Step 3: Write a description of the data structure implemented in this assignment and the actions performed by Methods 1 – 5.

```

// List.h for CS 215 Code Analysis Homework

#ifndef LIST_H
#define LIST_H

#include <iostream>
using namespace std;

template <class T>
class List
{
public:
    List ();
    List (const size_t & in_size);
    List (const List<T> & other);
    ~List ();
    List<T> & operator = (const List<T> & other);
    size_t Length () const;
    bool Method1 (const T & value);
    bool Method2 (const T & value);
    bool Method3 (T & value);
    bool Method4 (T & value);
    void Show ();
    friend istream & operator >> <T> (istream & ins, List<T> & L);
    friend ostream & operator << <T> (ostream & outs, const List<T> & L);

private:
#define CHUNK 4
    bool Method5 (const size_t & new_capacity);

    size_t capacity;
    T * data;
    size_t used;
    int first;
    int last;
};

template <class T>
List<T>::List ()
{
    capacity = 0;
    data = NULL;
    used = 0;
    first = last = -1;
}

template <class T>
List<T>::List (const size_t & size)
{
    capacity = size;
    data = new T [capacity];
    used = 0;
    first = last = -1;
}

```

```

template <class T>
List<T>::List (const List<T> & other)
{
    capacity = other.capacity;
    data = new T [capacity];
    used = other.used;
    first = other.first;
    last = other.last;
    int i, j = first;
    for (i = 0; i < used; i++)
    {
        data[j] = other.data[j];
        j = (j + 1) % capacity;
    }
}

template <class T>
List<T>::~~List ()
{
    delete [] data;
}

template <class T>
List<T> & List<T>::operator = (const List<T> & other)
{
    if (this == & other)
        return * this;
    if (capacity != other.capacity)
    {
        delete [] data;
        capacity = other.capacity;
        data = new T [capacity];
    }
    used = other.used;
    first = other.first;
    last = other.last;
    int i, j = first;
    for (i = 0; i < used; i++)
    {
        data[j] = other.data[j];
        j = (j + 1) % capacity;
    }
    return * this;
}

template <class T>
size_t List<T>::Length () const
{
    return used;
}

```

```

template <class T>
bool List<T>::Method1 (const T & value)
{
    if (used == capacity)
        if (Method5 (capacity + CHUNK) == false)
            return false;
    if (first == -1)
        first = last = 0;
    else if (first == 0)
        first = capacity - 1;
    else
        first--;
    data[first] = value;
    used++;
    return true;
}

```

```

template <class T>
bool List<T>::Method2 (const T & value)
{
    if (used == capacity)
        if (Method5 (capacity + CHUNK) == false)
            return false;
    if (last == -1)
        first = last = 0;
    else
        last = (last + 1) % capacity;
    data[last] = value;
    used++;
    return true;
}

```

```

template <class T>
bool List<T>::Method3 (T & value)
{
    if (used == 0)
        return false;
    value = data[first];
    if (capacity - used >= 1.5 * CHUNK)
        if (Method5 (capacity - CHUNK) == false)
            return false;
    used--;
    if (used == 0)
        first = last = -1;
    else
        first = (first + 1) % capacity;
    return true;
}

```

```

template <class T>
bool List<T>::Method4 (T & value)
{
    if (used == 0)
        return false;
    value = data[last];
    if (capacity - used >= 1.5 * CHUNK)
        if (Method5 (capacity - CHUNK) == false)
            return false;
    used--;
    if (used == 0)
        first = last = -1;
    else if (last == 0)
        last = capacity - 1;
    else
        last--;
    return true;
}

template <class T>
bool List<T>::Method5 (const size_t & new_capacity)
{
    T * temp = data;
    data = new T [new_capacity];
    if (data == NULL)
    {
        data = temp;
        return false;
    }
    if (used > 0)
    {
        int j = first;
        int k = (new_capacity - used) / 2;
        first = k;
        last = first + used - 1;
        for (int i = 0; i < used; i++)
        {
            data[k] = temp[j];
            j = (j+1) % capacity;
            k++;
        }
    }
    capacity = new_capacity;
    delete [] temp;
    return true;
}

template <class T>
void List<T>::Show ()
{
    cout << "c:" << capacity << ' ';
    cout << "u:" << used << ' ';
    cout << "f:" << first << ' ';
    cout << "l:" << last << ' ';
    for (int i = 0; i < capacity; i++)
        cout << "d[" << i << "]:" << data[i] << ' ';
    cout << endl;
}

```

```

}

template <class T>
istream & operator >> (istream & ins, List<T> & L)
{
    int i;
    for (i = 0; (i < L.capacity) && !ins.fail(); i++)
        ins >> L.data[i];
    L.used = i;
    L.first = 0;
    L.last = i - 1;
    return ins;
}

template <class T>
ostream & operator << (ostream & outs, const List<T> & L)
{
    int i;
    int j = L.first;
    for (i = 0; i < L.used; i++)
    {
        outs << L.data[j] << ' ';
        j = (j+1) % L.capacity;
    }
    return outs;
}

#endif

```

Homework 2  
 Due at the beginning of Lab 13

Use textbooks, the internet, or other resources to complete the following table:

Polygon Type	Area Calculation	Perimeter Calculation	Area Formula	Perimeter Formula
Right Triangle	leg = 3" leg = 4"	leg = 3" leg = 4"		
Equilateral Triangle	side = 4"	side = 4"		
Scalene Triangle	side = 4" side = 5" side = 6"	side = 4" side = 5" side = 6"		
Square	side = 4"	side = 4"		
Rectangle	length = 4" width = 7"	length = 4" width = 7"		
Regular N-sided Polygon	10 sides side = 4"	10 sides side = 4"		

# Summary of Essential Linux Commands

**man command** - get the manual pages for a given 'command'; for example, run 'man' on any of the shell commands given below to get information on usage and extra options.

**info command** - get info for the given 'command' (run 'info' in a shell).

**^D** - quits programs, exits shells.

**^L** - clears a console/terminal.

**!*pattern*** - re-runs the last command executed beginning with *pattern* (in a shell).

**^z** - suspends the execution of a program (in a shell).

**ls** - get a file listing; use '-al' to get a long file listing that includes hidden files and attributes.

**cd directory** - change directory to 'directory'.

**cp path1 path2** - copies from 'path1' to 'path2'.

**mv path1 path2** - moves or renames 'path1' to 'path2'.

**rm filename** - removes 'filename' (asks for confirmation).

**rm -rf path** - removes 'path' and everything that it contains (recursively) without asking for confirmation (CAREFUL!).

**rmdir directory** - removes the directory 'directory' (asks for confirmation).

**touch filename** - creates the empty file 'filename'.

**mkdir dirname** - creates the directory 'dirname'.

**chmod flags filename** - change that access rules for the given file.

**mount /floppy** - mounts the floppy drive one /floppy. Similarly, mount /cdrom mounts the cdrom.

**mount -t udf /dev/cdrom /mnt/cdrom** - Adaptec DirectCD uses a different filesystem (UDF) and so cdroms cut with it need to be mounted differently. If this is not done, the dreaded "DriveReady SeekComplete Error" happens.

**umount /floppy** - unmounts the floppy drive. Similarly, umount /cdrom mounts the cdrom. Always do this before ejecting the disk media.

**ssh** - secure shell, a secure form of telnet; NEVER use telnet, because it sends passwords out in the free and clear.

**sftp** - secure ftp, a secure form of ftp; NEVER use regular ftp.

**tar -xvf archive.tar** - extracts files from the tar archive 'archive.tar'; use -zxvf if the archive ends in .tar.gz or .tgz to invoke the gzip filter. To create a tar archive use -cvf, and to list its contents use tvf; again, insert the z option if the gzip filter is to be used.

**unzip archive.zip** - extracts files from the zip archive 'archive.zip'; use the corresponding 'zip' program to create archives.

**find . -name filename** -finds a file starting from current directory; use '-iname' for case insensitivity

**find . -path pathname** -finds a path starting from current directory; use '-iname' for case insensitivity

**grep pattern file** - look for a 'pattern' in a given 'file' or files. Use '-i' for case insensitivity, '-r' for directory recursion, '-n' to print out the line numbers, and '-s' to suppress the error messages. For example, use 'grep -irns pattern \*' to look for the pattern in any files in the current directory and subdirectories.

**more filename** - views 'filename' in a terminal that allows you to page down by pressing space.

**less filename** - views 'filename' in a terminal that allows you to scroll up and down with the arrows; type "/" to search. Less is more!

**emacs -nw filename** - edits 'filename' in the terminal (type ^x^s to save, and ^x^c to exit).

**sed** - stream editor

**diff file1 file2** - prints differences between 'file1' and 'file2'

**kill processid** - kills process with id 'processid'.

**jobs** - lists the jobs running in the shell.

**bg** - moves a suspended program into the background; type this immediately after the keystroke ^z.

**fg #** - moves job '#' into the foreground.

**top** - lists processes in terms of cpu usage. Note: Top will report that almost all available memory is 'used'. The memory is actually being employed as a cache by the linux kernel and is released to user programs as needed.

**uptime** - gives the system uptime and load averages.

**cat file1 file2 ...** - concatenates files to stdout; use 'cat > out.txt' to write what is typed to stdin into the file 'out.txt' (finish with a ^D at the beginning of a line).

**wc** - print the number of bytes, words, and lines in a text file.

**emacs filename** - edits 'filename' (type ^x^s to save, and ^x^c to exit).

**nedit** - text editor with a nice gui. Uses mac keystrokes.

**pine** - a nice email reading and sending program

**gcc** - the GNU C Compiler. gcc file.c will create an executable called a.out

**g++** - the GNU C++ Compiler. g++ file.cpp will create an executable called a.out

**make** - makes a C program into an executable given the existence of a 'Makefile'.

## NAME

rand, srand - random number generator.

## SYNOPSIS

```
#include <stdlib.h>
```

```
int rand(void);
```

```
void srand(unsigned int seed);
```

## DESCRIPTION

The `rand()` function returns a pseudo-random integer between 0 and `RAND_MAX`.

The `srand()` function sets its argument as the seed for a new sequence of pseudo-random integers to be returned by `rand()`. These sequences are repeatable by calling `srand()` with the same seed value.

If no seed value is provided, the `rand()` function is automatically seeded with a value of 1.

## RETURN VALUE

The `rand()` function returns a value between 0 and `RAND_MAX`. The `srand()` returns no value.

## NOTES

The versions of `rand()` and `srand()` in the Linux C Library use the same random number generator as `random()` and `srandom()`, so the lower-order bits should be as random as the higher-order bits. However, on older `rand()` implementations, the lower-order bits are much less random than the higher-order bits. In *Numerical Recipes in C: The Art of Scientific Computing* (William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling; New York: Cambridge University Press, 1992 (2nd ed., p. 277)), the following comments are made:

```
"If you want to generate a random integer between 1 and 10, you
should always do it by using high-order bits, as in
    j=1+(int) (10.0*rand()/(RAND_MAX+1.0));
and never by anything resembling
    j=1+(rand() % 10);
(which uses lower-order bits)."
```

Random-number generation is a complex topic. The *Numerical Recipes in C* book (see reference above) provides an excellent discussion of practical random-number generation issues in Chapter 7 (Random Numbers). For a more theoretical discussion which also covers many practical issues in depth, please see Chapter 3 (Random Numbers) in Donald E. Knuth's *The Art of Computer Programming*, volume 2 (Seminumerical Algorithms), 2nd ed.; Reading, Massachusetts: Addison-Wesley Publishing Company, 1981.

CONFORMING TO SVID 3, BSD 4.3, ISO 9899

## SEE ALSO

`random(3)`, `srandom(3)`, `initstate(3)`, `setstate(3)`

# Summary of Essential emacs Commands

Use "emacs foo" to create a file named foo, or to edit an existing file named "foo". You can do some work just by typing in the window once emacs comes to life. The backspace or delete key should do what you think it does.

In the list below, C- means "control key", M- means "meta key" (escape). For meta commands, press the meta key, then the other key. Thus M-f stands for the keyboard sequence "press meta key", " press f".

C-x C-s Save file  
C-x C-f Find (open) file. Emacs asks for file name at bottom of screen.  
C-x C-z Leave emacs temporarily (return with "fg" ("foreground"))  
C-x C-c Quit emacs

C-g Abort command (get out of trouble)

C-f Move cursor forward one character  
C-b Move cursor backwards one character

C-n Move cursor to next line  
C-p Move cursor to previous line  
C-a Move cursor to beginning of line  
C-e Move cursor to end of line

C-v Go forward one screen  
M-v Go backward one screen

C-k Kill text from cursor to end of line. (Cut text out)  
C-y Yank back killed text. (Paste text in)

C-l Redisplay the screen if it is garbled.

M-f Move cursor forward one word  
M-b Move cursor backwards one word  
M-> Move to end of file  
M-< Move to beginning of file

C-u 7 C-n Move forward 7 lines  
C-u 10 C-p Move back 10 lines

To go to a specific line: Type M-x goto-line <return>, then type the line number followed by <return>. There are shortcuts. Type the command "help -q emacs goto " in a local window to find out about it. (Local shortcut is M-x #).

Emacs has an excellent on-line help tutorial. To use it, start up emacs. When you are "in emacs", type C-h, then type the letter "t". Follow directions. When you wish to leave the tutorial, type C-x C-c.

Many keyboards have "arrow" keys. These move the cursor around in the expected way. The DEC machines have a "compose" key which works like the meta key, except: hold the compose key down while pressing the other key.

## Summary of Essential vi Commands

### Cursor control and position

h	Left
j	Down
k	Up
l (or spacebar)	Right
w	Forward one word
b	Back one word
e	End of word
(	Beginning of current sentence
)	Beginning of next sentence
{	Beginning of current paragraph
}	Beginning of next paragraph
[[	Beginning of current section
]]	Beginning of next section
0	Start of current line
\$	End of current line
^	First non-white character of current line
+ or RETURN	First character of next line
-	First character of previous line
n	character <i>n</i> of current line
H	Top line of current screen
M	Middle line of current screen
L	Last line of current screen
nH	<i>n</i> lines after top line of current screen
nL	<i>n</i> lines before last line of current screen
Ctrl-F	Forward one screen
Ctrl-B	Back one screen
Ctrl-D	Down half a screen
Ctrl-U	Up half a screen
Ctrl-E	Display another line at bottom of screen
Ctrl-Y	Display another line at top of screen
z RETURN	Redraw screen with cursor at top
z .	Redraw screen with cursor in middle
z -	Redraw screen with cursor at bottom
Ctrl-L	Redraw screen without re-positioning
Ctrl-R	Redraw screen without re-positioning
/text	Search for <i>text</i> (forwards)
/	Repeat forward search
?text	Search for <i>text</i> (backwards)
?	Repeat previous search backwards
n	Repeat previous search
N	Repeat previous search, but it opposite direction
/text/+n	Go to line <i>n</i> after <i>text</i>
?text?-n	Go to line <i>n</i> before <i>text</i>
%	Find match of current parenthesis, brace, or bracket.

Ctrl-G	Display line number of cursor
<i>n</i> G	Move cursor to line number <i>n</i>
: <i>n</i>	Move cursor to line number <i>n</i>
G	Move to last line in file

## Editing

A	Append to end of current line
i	Insert before cursor
I	Insert at beginning of line
o	Open line above cursor
O	Open line below cursor
ESC	End of insert mode
Ctrl-I	Insert a tab
Ctrl-T	Move to next tab position
Backspace	Move back one character
Ctrl-U	Delete current line
Ctrl-V	Quote next character
Ctrl-W	Move back one word
cw	Change word
cc	Change line
C	Change from current position to end of line
dd	Delete current line
<i>n</i> dd	Delete <i>n</i> lines
D	Delete remainder of line
dw	Delete word
d}	Delete rest of paragraph
d^	Delete back to start of line
<i>c/pat</i>	Delete up to first occurrence of pattern
dn	Delete up to next occurrence of pattern
<i>dfa</i>	Delete up to and including <i>a</i> on current line
<i>dta</i>	Delete up to, but not including, <i>a</i> on current line
dL	Delete up to last line on screen
dG	Delete to end of file
J	Join two lines
p	Insert buffer after cursor
P	Insert buffer before cursor
rx	Replace character with <i>x</i>
<i>Rtext</i>	Replace <i>text</i> beginning at cursor
s	Substitute character
<i>ns</i>	Substitute <i>n</i> characters
S	Substitute entire line
u	Undo last change
U	Restore current line

## File Handling

:w	Write file
:w!	Write file (ignoring warnings)
:w! <i>file</i>	Overwrite <i>file</i> (ignoring warnings)
:wq	Write file and quit
:q	Quit
:q!	Quit (even if changes not saved)
:w <i>file</i>	Write file as <i>file</i> , leaving original untouched
ZZ	Quit, only writing file if changed
:x	Quit, only writing file if changed
: <i>n1,n2</i> w <i>file</i>	Write lines <i>n1</i> to <i>n2</i> to <i>file</i>
: <i>n1,n2</i> w >> <i>file</i>	Append lines <i>n1</i> to <i>n2</i> to <i>file</i>
:e <i>file2</i>	Edit <i>file2</i> (current file becomes alternate file)
:e!	Reload file from disk (revert to previous saved version)
:e#	Edit alternate file
%	Display current filename
#	Display alternate filename
:n	Edit next file
:n!	Edit next file (ignoring warnings)
:n <i>files</i>	Specify new list of <i>files</i>
:r <i>file</i>	Insert <i>file</i> after cursor
:r ! <i>command</i>	Run <i>command</i> , and insert output after current line

# GDB QUICK REFERENCE GDB Version 4