Programming with Objects

(Solutions to Homework Problems)

Programming with Objects

(Solutions to Homework Problems)

A Comparative Presentation of
Object-Oriented Programming
with C++ and Java

Avinash C. Kak

Purdue University

A Wiley-Interscience Publication
JOHN WILEY & SONS, INC.

New York / Chichester / Weinheim / Brisbane / Singapore / Toronto

The errata and other information for this book are posted at

http://www.programming-with-objects.com

Preface

This book presents object-oriented programming with C++ and Java, which are today's two dominant languages for such programming. The presentation format is mostly comparative, all the way from the basic language constructs to application-level issues dealing with graphics programming, network programming, and database programming. This book is intended for a reader who is well-conversant with the important features of C: pointers, strings, arrays, and structures.

The author strongly believes in the notion that, in addition to the syntax, it is essential to also show a programming language through its applications to fully establish its beauty and power. Teaching a programming language divorced from its applications – not uncommon in many educational programs – would be like teaching English through just its grammar.

This book grew out of an attempt to meet a specific academic need for a comprehensive educational program in object-oriented programming. We wanted a program that would not be too indoctrinating with regard to any one style (or any one language, since language often dictates style) of object-oriented programming. While programming skill could have been taught by focusing on a single language, education in its larger sense demanded that we provide a broader menu of styles and concepts. The result was what the reader sees in this book: An integrated presentation of C++ and Java. There is educational value in comparing and contrasting the two languages, from basic language constructs to how the languages are used in application-level programming. Such comparisons may even inspire an enterprising student to think of new and more powerful object-oriented languages of the future. To further enhance

the educational value of this comparative approach, this book also includes treatment of simulated object-orientation in plain C, with GNOME/GTK+ presented as a major example of this approach.

This book is based on the philosophy that learning by comparison is very efficient and can be a lot of fun. Sometimes we find it easier to remember and learn things if we can anchor our memory and comprehension in interesting differences and similarities between supposedly similar objects, structures, and situations. Learning C++ and Java together can exploit this aspect of human cognition. Students find it interesting to compare C++ and Java programming constructs for doing the same thing.

Teaching and learning C++ and Java together have some unique advantages. First, because both C++ and Java were born out of C, they have much in common at the level of basic language structures. Teaching these structures together saves time. For example, once the concept of a vector in C++ is made clear and some of the more useful functions associated with C++ vectors are elucidated, the discussion of the Java ArrayList takes hardly any time. The Java discussion consists mostly of pointing out the Java functions that do the same thing as the previously discussed C++ functions.

Then there is also the unique process of learning by coding up a program in C++ that does the same thing as a given program in Java, or vice versa. My experience is that this approach enables the students to tackle more difficult projects in both C++ and Java than would otherwise be the case under the time constraints of a course.

Learning two large languages together does have its down side. One can get confused as to what feature belongs to which language. Fortunately, this difficulty is minimized by the modern programming practice of keeping one eye on the on-line documentation in one terminal window while programming in another terminal window. Both Java and C++ have become so large that it would be impossible for anyone to commit to memory all of the classes and all of the functions and attributes defined for the classes. So even if one were not learning two languages simultaneously, one would still need to refer to documentation while writing programs.

The book contains more material than can be accommodated in a typical one-semester course. In my experience, the book works well for a sequence of two back-to-back courses, the first focusing on the basic language constructs as presented in the first fifteen chapters, and the second focusing on application- and design-level issues. For the second course, I complement the material in the last five chapters with a book on design patterns.

It would be naive of me to assume that a manuscript as large as this would be free of errors. I'd be much grateful to the readers who would bring the errors to my attention at `kak@purdue.edu`. All corrections will be made available online at `www.programming-with-objects.com`, and the authors of the corrections will be duly acknowledged. The same applies to any slip-ups on my part in giving proper attributions to authors. Where my example programs were inspired directly by what I saw in other sources, I have acknowledged their authors in the "Credits and Suggestions for Further Reading" section at the end of each chapter.

The author will be glad to make available to the prospective instructors the solutions to the homework problems.

Finally, the book should also be useful to those who are transitioning from C++ to Java, or vice versa.

Purdue University Avinash Kak

Acknowledgments

Whatever merit this book has should go in large measure to the stalwarts of the object-oriented programming movement, to those who created C++ and Java, and to those who have been the chief expositors of these two languages over the last several years (see the references at the end of the book).

This book would not have been possible without the help of the following people:

- Guilherme DeSouza, a man with insights that are as deep as they are broad and for whom Linux is a religion to which he has converted many, this author included. Guilherme's insights in multiprocessing and multithreading played an important role in the revamping of Chapter 18.
- Elvia Suryadi, who can spot from a mile the slightest flaw in a logical argument, the minutest weakness in an explanation. Her constant feedback helped with the cleanup of many sections of the book, too numerous to be listed here individually. The homework problems in Chapters 13 and 20 are by Elvia.
- Malcolm Slaney, with an uncanny eye for rigor and precision, for providing critical feedback on the first five chapters.
- Carl Crawford, never a man to mince words, who insisted that my earlier version of the front matter did not do justice to the rest of the book.
- Susan Gottschlich, with deep insights in the software development cycles of industry, for her careful reading and feedback of the first 11 chapters.

- Robert Cromwell, who has always had his ear to the ground for the latest happenings in the world of software and hardware, for looking carefully at the C-related material in the book and suggesting improvements.
- Sarah Sellke, with many years of object-oriented software design and development experience for telecom and other applications, for her feedback on Chapter 19.
- Christina Pavlopoulou, as a source of great help in the early stages of my putting together this book.
- Prathima Venkatesan, who proofread and caught many errors in the draft copies of Chapters 4, 5, 6, and 7.
- Bob Mulvey, for helping me better understand the various shortcomings associated with the use of `setjmp`–`longjmp` macros for achieving multilevel return in C.
- Brett Maden, for creating the final versions of the figures in Chapters 15 and 17. He also contributed to the homework section of Chapter 17; the Qt and GNOME/GTK+ problems were supplied by him.
- Kheng Tan, for many of the final production figures for Chapters 14 and 16. Kheng also supplied the problems for the homework section of Chapter 14.

Thanks also go to the anonymous reviewers of the book; many of the comments I received through the review process helped in the revision of much material. Of the publisher reviews received nonanonymously, I wish to thank Simon Gray in particular for catching many errors in Chapter 2, 8, 10, 14, and 19 and suggesting improvements.

Many additional sources of help and information that proved important to the writing of this book are acknowledged near the end of each chapter in a section entitled "Credits and Suggestions for Further Reading," or sometimes just "Suggestions for Further Reading." Occasionally, I have also used a footnote for the same purpose.

I am also grateful to Subhash Kak, whose powers of exposition border on the lyrical, for his many wonderful suggestions for smoothing out the text at various places.

Finally, and most importantly, many thanks go to Carla for her loving friendship, support, and understanding, all things that give true meaning to life. Thanks also go to Carla for her direct contributions to the book by way of critical reading of its various sections.

A.C.K.

Contents

1

Why OO Programming – Some Parallels with Things at Large

no problems in this chapter

2

Baby Steps

PROBLEM 1:

The following code fragment is legal in C++, but not so in C. Why? What would it take to make the code legal in C? Also, what's the value of the variable y?

```
double x[10] = {5.1, 7.3};
x[0] = 3.12;
*x = 4.13;
double y;           //(A)
y = x[0];
```

Solution:

It is not legal in C because the declaration in line (A) comes after a couple of executable statements.

PROBLEM 2:

Write programs in C++ and Java for reading just one integer, just one string, and just one double from the console.

C++ Solution:

```
int x;
cin >> x;
cout << x << endl;

double y;
cin >> y;
cout << y << endl;

string buffer;
cin >> buffer;
cout << buffer << endl;
```

Java Solution:

```
//ReadConsole.java
import java.io.*;

class ReadConsole {

  public static void main( String[] args )
  {
    int x = readInt();
    System.out.println("The integer entered was: " + x );

    String str = readString();
    System.out.println("The string entered was: " + str );

    double d = readDouble();
    System.out.println("The double entered was: " + d );
  }

  public static int readInt( ) {
    int x = 0;
    try {
      while ( true ) {
        int ch = System.in.read();
        if ( ch == '\n' || ch == ' ' ) break;
        x = 10 * x + ch - 48;
      }
    } catch(IOException e){}
    return x;
  }
```

```
  public static String readString() {
    String word = "";
    try {
      while ( true ) {
        int ch = System.in.read();
        if ( ch == '\n' || ch == ' ' ) break;
        word += (char) ch;
      }
    } catch(IOException e){}
    return word;
  }

  public static double readDouble() {
    String s = readString();
    double x = Double.valueOf( s ).doubleValue();
    return x;
  }
}
```

PROBLEM 3:

Write programs in C++ and Java for finding the maximum of an array of `int`s. Your C++ program should use the object `cout` and the operator '`<<`' for displaying the result. Your Java program should use the `System.out.println` method for doing the same.

PROBLEM 4:

Write programs in C++ and Java that read a text file containing 100 or fewer words and store all the words into an array of strings.

Java Solution:

```
import java.io.*;
import java.util.*;


class ReadStringsFromFile {
```

```
    public static String[] readAllStrings( String fileName ) {
        String[] wordArr = new String[100];
        String allChars = "";
        int i = 0;
        try {
            int ch;
            Reader in = new FileReader( fileName );
            while ( -1 != ( ch = in.read() ) )
                allChars += (char) ch;
            System.out.println( allChars );
            StringTokenizer st = new StringTokenizer( allChars );
            while ( st.hasMoreTokens() )
                wordArr[i++] = st.nextToken();
        } catch( java.io.IOException e ) {}
        return wordArr;
    }


    public static void main(String[] args) {
        String[] strings = readAllStrings( "infile.txt" );
        for (int i=0; i<strings.length; i++)
            System.out.println( strings[i] );
    }

}
```

C++ Solution:

```
// SortTextFile.cc

#include <string>
#include <fstream>
use namespace std;

int compareStrings( const void* arg1, const void* arg2 );
int checkUpperCase( string buffer );

int main() {

  ifstream readFromFile( "in.txt" );
  ofstream writeToFile( "out.txt" );

  string buffer;
```

```
  string* word_list[100];

  int i = 0;
  while ( readFromFile >> buffer )
    if ( checkUpperCase( buffer ) )
      word_list[i++] = new string(buffer);

  qsort( word_list, i, sizeof(string*), compareStrings);

  int j = 0;
  while (j < i)
    writeToFile << *word_list[j++] << " ";
  writeToFile << endl;
}


int compareStrings( const void* arg1, const void* arg2 ) {
  return (**(string**) arg1).compare(**(string**) arg2);
}

int checkUpperCase( string word ) {
  return isupper( word[0] );
}
```

3

The Notion of a Class and Some Other Key Ideas

PROBLEM 1:

Will this C++ program compile? If the code shown in not legal, what's wrong with it and how will you fix it?

```
class X {
    int n;
public:
    void X( int i ) { n = i; }
};

int main() { X xobj( 100 ); }
```

Solution:

This program will not compile. A constructor can have no return type.

PROBLEM 2:

Will this Java program compile? If the code shown in not legal, what's wrong with it and how will you fix it?

```
class X {
    private int n;

    public void X( int i ) { n = i; }
}


class Test {
    public static void main( String[] args ) {
        X xobj = new X(100);
    }
}
```

Solution:

This program will not compile. A constructor can have no return type.

PROBLEM 3:

Do you see any parallels between the children's riddle: "Who is bigger? Mr. Bigger or Mr. Bigger's little baby?" and the question "Which is bigger? A class or a class's little baby, meaning a subclass?"

PROBLEM 4:

Provide C++ and Java definitions for an `Account` class that could be used for a bank account. The class would need at least three data members: `name`, `balance`, and `accountNumber`. Specify a constructor and a print function for the class.

Let the `Account` class be the parent class of the subclasses named `SavingsAccount` and `CheckingAccount`. The subclasses will have an additional data member named `interestRate`. The value of these rates would be different for the checking accounts and the savings accounts.

Your homework should show the class definitions. Your homework should also show some specific objects created from the classes.

PROBLEM 5:

The following Java class does something rather "peculiar"; in line (A) it sets its data member `y` equal to the value of the data member `x`. The class comes with two constructors, in lines (B) and (C). The constructor in line (B) supplies a value for each of the two data members of the class. On the other hand, the constructor in line (C) initializes only the data member `x`. What will be printed out by the statements in lines (D) and (E)? [**Note: You'll probably be surprised by the correct answer to the question, which you can find out by compiling and running the program. The discussion in Section 7.3 will help you understand the behavior of the program.**]

```
class X {
    private int x;
    private int y = x;                                          //(A)

    X( int xx, int yy ) { x = xx; y = yy; }                     //(B)
    X( int xx ) { x = xx; }                                     //(C)

    public String toString() { return "" + x + " " + y; };

    public static void main( String[] args ) {
        X xobj = new X( 100, 200 );
        System.out.println( xobj );                             //(D)

        xobj = new X( 300 );
        System.out.println( xobj );                             //(E)
    }
}
```

Solution:

The statement in line (D) prints out "100 200". And the statement in line (E) prints out "300 0".

PROBLEM 6:

The `print( X* )` function defined below is not able to do its job because `m` and `n` are in the private section of the class `X`? How can this situation be fixed with a single *additional* declaration in the definition of class `X`? (You are not allowed to change the access control property for any of the members of `X`.)

```
class X {
    int m;
    int n;
public:
    X( int mm, int nn ) { m = mm; n = nn; }
};

void print( X* ptr ) {
    cout << ptr->m << "  " << ptr->n << endl;
}
```

Solution:

Insert the following declaration anywhere in class `X`:

```
    friend void print( X* );
```

PROBLEM 7:

Here is an example of a nested inteface in Java. Would this code fragment compile?

```
interface X {
    interface Y {
        void doSomething_Y();
    }
    void doSomething_X();
}


class Z implements X {
    public void doSomething_X() {}
    public void doSomething_Y() {}
}
```

Solution:

No problem with this program.

PROBLEM 8:

The following Java program does not compile. The compiler reports a problem with the statement in line (B). How will you fix it?

```
interface X {
    interface Y {
        void doSomething_Y();
    }
    void doSomething_X();
}

class Z implements X {                          //(A)
    public void doSomething_X() {}
    public void doSomething_Y() {}
    public static void main( String[] args ) {
        X x = new Z();
        Y y = new Z();                          //(B)
    }
}
```

Solution:

Replace the syntax in line (A) by

```
class Z implements X, X.Y {
```

4

Strings

PROBLEM 1:

How many `String` objects will be created by the following Java statements:

```
String s1 = "hoity";
String s2 = s1;
s1 = s2 + "toity";
```

Solution:

The answer is 4. One for the string literal “hoity”; one for the String object from the literal that is referenced by s1; one for the string literal “toity”; and one for the String object returned by concatenating the string to which s2 points with “toity”.

PROBLEM 2:

How many `String` objects are created by the following Java statements:[1]

[1]Based on an example posted at the Java Virtual Machine Forum of the Java Developer Connection.

```
String s1 = "hoity";
String s2 = "hoity";
String s3 = "hoity";
String s4 = "hoity";
char[] charArr = [ 'h', 'o', 'i', 't', 'y' ];
String s5 = new String( charArr );
String s6 = new String( s1 );
```

Solution:

Only three String objects will be created. The first four statements will all result in the creation of a single String object. The same String object "hoity" will be referenced by the variables s1 through s4. The following sort of a program taken from the Forum at JDC can be used to test the answer to the question

```
class Test {

  public static void main(String args[])
  {
    String s1 = "abc";
    String s2 = "abc";
    char data[] = {'a', 'b', 'c'};
    String s3 = new String(data);
    String s4 = new String(s1);

    System.out.println("s1 (" + s1 + ") == s2 (" + s2 + ") : " + (s1 == s2));
    System.out.println("s1 (" + s1 + ") == s3 (" + s3 + ") : " + (s1 == s3));
    System.out.println("s1 (" + s1 + ") == s4 (" + s4 + ") : " + (s1 == s4));

    System.out.println("s1 (" + s1 + ").equals() s2 (" + s2 + ") : " + (s1.equals(s
    System.out.println("s1 (" + s1 + ").equals() s3 (" + s3 + ") : " + (s1.equals(s
    System.out.println("s1 (" + s1 + ").equals() s4 (" + s4 + ") : " + (s1.equals(s

  }
}
```

This program prints out

```
s1 (abc) == s2 (abc) : true
s1 (abc) == s3 (abc) : false
s1 (abc) == s4 (abc) : false
s1 (abc).equals() s2 (abc) : true
s1 (abc).equals() s3 (abc) : true
s1 (abc).equals() s4 (abc) : true
```

PROBLEM 3:

The goal of this homework is to write a Java program to determine whether a string is a palindrome. (A palindrome is a word, phrase, verse, or sentence that reads the same backward or forward.) The string may contain other characters that are not alphanumeric. These characters are to be removed first before determining whether a string is a palindrome. More specifically, your solution should be structured along the following lines:

1. Provide implementation for a function `removeAllMarks( String )` that takes the original string as its argument and returns a new string after the original string has been stripped of the punctuation marks.

   [**Suggestion:** A convenient way to do this is by scanning the argument string with the `charAt( int i )` method presented in Section 4.4.2 and checking whether the character at position `i` is alphanumeric or not. This test can be carried out by invoking `Character.isLetterOrDigit( char )` on the character. In order to create a clean version of the string, all characters that pass the test can be appended to a `StringBuffer` object with the `append( char )` method defined for the `StringBuffer` class. Finally, the `StringBuffer` object can be converted into a `String` object by using the `toString()` method of the `StringBuffer` class.]

2. Provide implementation for `reverseString(String str)` which returns a new string obtained by reversing the argument string.

   [**Suggestion:** You can convert the argument string into a `StringBuffer` object via the latter's constructor and then invoke the `reverse()` method on the `StringBuffer` object.]

3. Provide implementation for `isPalindrome( string )` that will return true or false as to whether the argument string is a palindrome. Compare the cleaned up version of the original string with the cleaned up and reversed version using the `compareToIgnoreCase( String )` method of the `String` class.

You may define and implement other functions to complete this homework. Print out the original string and whether or not it is a palindrome.

Solution:

```
import java.io.*;

public class Palindrome{

  protected static String removeAllMarks(String origString){
```

```
    int i = 0;
    char ch;

    StringBuffer sb1 = new StringBuffer();

    for(i = 0; i < origString.length(); i++){
      ch = origString.charAt(i);
      if((Character.isLetterOrDigit(ch)))
        sb1.append(ch);
    }

    return sb1.toString();
  }

  protected static String reverseString(String str){
    StringBuffer sb2 = new StringBuffer(str);

    sb2.reverse();

    return sb2.toString();
  }

  protected static boolean isPalindrome(String targetStr){
    String fixedStr = removeAllMarks(targetStr);
    String revStr = reverseString(fixedStr);

    if((fixedStr.compareToIgnoreCase(revStr)) == 0)
        return true;
    else
      return false;
  }

  public static void main(String[] args){
    String s1 = new String("Gate-man sees name, garage-man sees name-tag.");

    if(isPalindrome(s1))
      System.out.println("\"" + s1 + "\"" + " is a palindrome!");
    else
      System.out.println("\"" + s1 + "\"" + " is not a palindrome");
  }
}
```

PROBLEM 4:

Both the `StringFind.cc` program of Section 4.3.5 and the `StringFind.java` program of Section 4.4.5 suffer from one major shortcoming that is illustrated by the following example. If we ask the program to replace "ice" by "cream" in the following string,

```
ice and icecream are two different things
```

both programs would result in the following string

```
cream and creamcream are two different things
```

This happens because the programs blindly replace the substrings without making certain that the substrings are words. Modify those programs so that this does not happen.

Java Solution:

```
import java.io.*;

public class Replace{

  public static void main (String[] args) throws Exception {
    String newLine;
    System.out.println("1. Enter a line ");
    BufferedReader br1 = new BufferedReader(new InputStreamReader(System.in));

    System.out.println("2. Enter the word you would like to replace: ");
    BufferedReader br2 = new BufferedReader(new InputStreamReader(System.in));

    System.out.println("3. Enter the word you would like to replace it with: ");
    BufferedReader br3 = new BufferedReader(new InputStreamReader(System.in));
    while (true) {
      String line = br1.readLine();
      String iWord = br2.readLine();
      String rWord = br3.readLine();
      StringBuffer lineBuff = new StringBuffer(line);

      int i;

      int index = 0;
      while( ( index = line.indexOf( iWord, index ) ) != -1 ){
          if(line.charAt(index+iWord.length()) == ' '){
```

```
            lineBuff.delete(index, index+iWord.length());
            lineBuff.insert(index, rWord);
            line = lineBuff.toString();
            lineBuff = new StringBuffer(line);
          }
          index++;
      }

      newLine = lineBuff.toString();
      System.out.println(newLine);

      return;
    }
  }
}
```

C++ Solution:

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

int main(){
  string line;
  string iWord;
  string rWord;
  string::size_type pos = 0;

  cout << "Please enter a line: ";
  getline(cin,line);

  cout << "Your entered: " << "\"" << line << "\"" << endl;

  cout << "Enter the word you would like to replace in the line: ";
  cin >> iWord;

  cout << "Enter the word you would like to replace \"" << iWord
       << "\" with: ";

  cin >> rWord;

  assert (iWord != rWord);

  while((pos = line.find(iWord, pos)) != string::npos){
```

```
    line.replace(pos, iWord.size(), rWord);
    pos++;
  }

  cout << "The new sentence is: " << "\"" << line << "\"" << endl;

  return 0;
}
```

PROBLEM 5:

Extend the `StringFind.cc` program of Section 4.3.5 so that it prompts a user for the entry of a line of text, a word all of whose occurrences must be replaced, and a replacement word. The program should then display on the terminal the modified line of text.

Solution:

See the C++ solution to the previous problem.

PROBLEM 6:

Write a C++ program that reads a text file to accomplish the following

- As it reads each word into the program, it should drop any punctuations and other non-alphanumeric marks sticking to the words in the text file. In particular, it should detect and drop the following marks

  `, ; : # . * ! " ) ( > < ] [ \ _ - ?`

  if they are attached to either the beginning or the end of a word. Make sure that if there are multiple occurrences of a mark at either the beginning or the end of a words, your program deletes all of them. That is, the word "!!!great!!!" should get cleaned up to just "great".

- The program should store each cleaned up word in a vector of strings.

- Finally, the program should sort the cleaned up words by invoking the generic algorithm `sort` and write the words in sorted order into an output text file.

Use the code shown in Chapter 2 for reading from a text file and writing into a text file. Test your program on the following input

```
!!!Hello!!!
Where's you??????
You have not been seen in a >>>>long<<<< time.
Will you *****ever***** sur-
face again???????  I can't wait for e-v-e-r!
I have a question: Should "dirty-words" be
stored as two words: 'dirty' and 'words',
or as one word: dirtywords.
Yours
** ### !!!!
```

Solution:

```
#include <fstream>
#include <string>
#include <vector>
#include <algorithm>
use namespace std;

string cleanup( string& );

int main(int argc, char** argv)
{
    if ( 3 != argc ) {
        cerr << "wrong usage" << endl;
        exit(1);
    }

    ifstream in( *++argv );
    if (!in) {
        cerr << "cannot open input file" << endl;
        exit(1);
    }

    ofstream out( *++argv );
    if (!cout) {
        cerr << "cannot open output file" << endl;
        exit(1);
    }

    string buffer;
```

```
    string cleanBuffer;

    vector<string> wordVec;

    string clean_buffer;
    while ( in >> buffer ) {
        cleanBuffer = cleanup( buffer );
        if ( cleanBuffer.size() > 0 )
            wordVec.push_back( cleanBuffer );
    }

    sort( wordVec.begin(), wordVec.end() );

    int j = 0;
    while (j < wordVec.size() )
        out << wordVec[j++] << " ";

    return 0;
}

/*****************************************************************
 *  This function checks only the first and the last characters
 *  of a string to see if either is a punctuation. If a punctuation
 *  is found at either end, it is dropped.  Punctuation marks
 *  within the word are retained.
 *****************************************************************/

string cleanup( string& word ) {
    string filter(",;:#.*!\")(><][\\_-?");

    string::size_type pos = 0;

    //get rid of repeated punctuation marks at the beginning
    //as in    ">>>this is important<<<"
    while ( word.size() > 0 && !isalnum( word[0] ) ) {
        pos = filter.find( word[0] );
        if ( pos != string::npos )
            word = string( word.begin() + 1, word.end() );
    }

    //get rid of repeated punctuation marks at the end
    while ( word.size() > 0 && !isalnum( word[ word.size() - 1 ] ) ) {
        pos = filter.find( word[ word.size() - 1 ] );
        if ( pos != string::npos )
            word = string( word.begin(), word.end() - 1 );
```

```
    }
    return word;
}
```

PROBLEM 7:

Extend the above program by giving special consideration to hyphenations and apostrophes. For hyphenated words, if the hyphenation is at the end of a line, assume that the last word in the line got broken up into two parts. Join the two parts. If the hyphenation is in middle of a line, do not delete it. When you detect an apostrophe, also drop the letter that follows the apostrophe. *Hint: Use the* `peek()` *function to detect the end of a line.*

PROBLEM 8:

If you just feed an array of strings to the `java.util.Arrays.sort` method, it uses the natural order for strings for sorting. This order, which basically amounts to comparing ASCII encodings of the characters, places all the words that begin with uppercase letters before the words that begin with lowercase letters. If you are not happy with this sort result, it is possible to invoke `java.util.Arrays.sort` with a second argument that is a `Comparator` object which can tell the sort function to produce its output in the same way you see the words in an English dictionary. The `Comparator` object must invoke the locale-specific string comparison methods. The locale-specific string comparison methods are defined for the `Collator` class. The main goal of this homework is to see how the `Locale`, `Collator`, and `Comparator` classes work together to produce a sort that corresponds to English language dictionary order.

Write a Java program that sorts the words of English in a way that you'd find in an English language dictionary. For the purpose of testing your program, initilize an array of strings within the program with different words.

Solution:

```
// SortTextFile.java

import java.io.*;
import java.util.*;      // for classes Locale, Arrays
import java.text.*;      // for class Collator
```

```
public class SortTextFile {

    public static void main( String[] args ) {
        Locale loc = Locale.US;
        Collator coll = Collator.getInstance( loc );

        String[] wordArr = new String[100];
        String str = "";

        try {
            Reader in = new FileReader( "in.txt" );
            PrintWriter out = new PrintWriter( new
                         FileOutputStream( "out.txt" ) );
            int ch;
            while ( ( ch = in.read() ) != -1 )
                str += (char) ch;
            in.close();

            int i = 0;
            StringTokenizer st = new StringTokenizer( str );
            while ( st.hasMoreTokens() ) {
                if ( i == 100 ) {
                    System.err.println(
                        "Input file has more than 100 words.  Aborting." );
                    System.exit( 1 );
                }
                wordArr[ i++ ] = st.nextToken();
            }

            Arrays.sort( wordArr, 0, i, new MyComparator( coll ) );

            for (int j = 0; j < i; j++)
                out.print( wordArr[ j ] + " " );

            out.close();
        } catch( IOException e ) { e.printStackTrace(); }
    }
}


class MyComparator implements Comparator {

    Collator coll;

    MyComparator( Collator coll ) { this.coll = coll; }
```

```
    public int compare( Object str1, Object str2 ) {
        return coll.compare( (String) str1, (String) str2 );
    }
}
```

5

Using the Container Classes

PROBLEM 1:

With the C++ class `Pet` defined as

```
class Pet {
    string name;
    int age;
public:
    Pet( string n, int a ) : name( n ), age( a ) {}
};
```

three out of the nine `main`'s shown below do not compile. (The six others compile and run fine.) Identify the ones that do not compile. Give your reasons for why they do not compile.

1. `int main() { Pet pets[3]; }` //no-compile
2. `int main() { Pet* pets[3]; }` // no problem
3. `int main() { vector<Pet> pets; }` // no problem
4. `int main() { vector<Pet> pets(3); }` // no-compile
5. `int main() { vector<Pet*> pets; }` // no-problems
6. `int main() { vector<Pet*> pets(3); }` // no-problems

7. `int main() { list<Pet> pets(3); }` // no-compile
8. `int main() { list<Pet> pets; }` // no-problem
9. `int main() { list<Pet*> pets; }` // no-problem

Assume that all the needed library header files are included in the programs for the testing of the `main`'s shown.

Solution:

The solutions are shown in the commented out portions of the lines to the right of the questions. All no-compiles are a consequence of the missing no-arg constructor for the class `Pet`.

PROBLEM 2:

The class `X` in the following program is supplied with a destructor in line (A). Whenever this destructor is invoked, it lets us know by printing out the message shown in the body of the destructor. In `main`, we first construct an empty vector in line (B) for holding objects of type `X`. We then construct three objects of type `X` in lines (C) through (E) and push them into the vector in lines (F) through (H). When this program is compiled and run, it produces the following output

```
Destructor invoked for X object with p = 1
Destructor invoked for X object with p = 1
Destructor invoked for X object with p = 2
Destructor invoked for X object with p = 3
Destructor invoked for X object with p = 2
Destructor invoked for X object with p = 1
Destructor invoked for X object with p = 1
Destructor invoked for X object with p = 2
Destructor invoked for X object with p = 3
```

Explain this output. Explain especially why for object `x1` the destructor is invoked four times, for object `x2` three times, and for object `x3` only two times. Here is the program:

```
//VectorDest.cc

#include <iostream>
#include <vector>
```

```
using namespace std;

class X {
    int p;
public:
    //constructor:
    X( int q ) { p = q; }
    //destructor:
    ~X() {                                                    //(A)
        cout << "Destructor invoked for X object with p = "
             << p << endl; }
    };

int main()
{
    vector<X> vec;                                            //(B)

    X x1( 1 );                                                //(C)
    X x2( 2 );                                                //(D)
    X x3( 3 );                                                //(E)

    vec.push_back( x1 );                                      //(F)
    vec.push_back( x2 );                                      //(G)
    vec.push_back( x3 );                                      //(H)

    return 0;
}
```

Solution:

To obtain a clue to the answer, comment out the statement in line (H) and insert the following immediately after line (H):

```
while (1) {};
```

Compile the program and execute it. Now you will see only the following output:

```
Destructor invoked for X object with p = 1
```

This output means that the system had to make one call to `X`'s destructor when you pushed `x2` into the vector in line (G). Pushing `x2` meant that the system had to destroy the current vector, which called for destroying the copy of `x1` in the current vector. Subsequently, the system appropriated fresh memory for two elements and placed `x1` and `x2` in the freshly created new version of the vector. (You need the infinite 'while'

loop to see this in order to prevent the objects created in lines (C), (D), and (E) from being destroyed.)

If you extend the above reasoning, when x3 is pushed into the vector, that results in *one more* call to the destructor on behalf of x1 and one call to the destructor on behalf of x2. Now add to these counts the number of times the destructor would be called when the program execution terminates. There will be two more calls to the destructor on behalf of x1, one to destroy the object created in line (C) and the other to destory its copy in the vector. This makes a total of four for x1. By the same token, there will be two more calls on behalf of x2, making for a total of three. And there will be just two calls to the destructor on behalf of x3.

PROBLEM 3:

The following program is a slight variation on the program of the previous problem. The class X is now supplied with an additional function, changeState() in line (A). In main we do the same thing as before, except that we also change the state of each of the three objects created before the program is allowed to run to completion. This program produces the following output:

```
Destructor invoked for X object with p = 1
Destructor invoked for X object with p = 1
Destructor invoked for X object with p = 2
Destructor invoked for X object with p = 300
Destructor invoked for X object with p = 200
Destructor invoked for X object with p = 100
Destructor invoked for X object with p = 1
Destructor invoked for X object with p = 2
Destructor invoked for X object with p = 3
```

Explain this output. Explain especially the order in which the three objects created in main, x1, x2, and x3, are destroyed and the order in which the copies of the objects held by the vector are destroyed.

```
//VectorDestOrder.cc

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

class X {
    int p;
public:
```

```
    X( int q ) { p = q; }
    void changeState( int pp ) { p = pp; }                    //(A)
    ~X(){ cout << "Destructor invoked for X object with p = "
               << p << endl; }
};

int main()
{
    vector<X> vec;

    X x1( 1 );
    X x2( 2 );
    X x3( 3 );

    vec.push_back( x1 );
    vec.push_back( x2 );
    vec.push_back( x3 );

    x1.changeState(100);
    x2.changeState(200);
    x3.changeState(300);

    return 0;
}
```

Solution:

Use the same line of reasoning as presented in the solution to the previous problem. That line of reasoning will also make clear the sequencing of the output. Unless object destruction is called for by operations such pushing a new element into a vector that is already populated to the fullest, the objects are destroyed in the reverse order of their creation at program termination.

PROBLEM 4:

What's wrong with the following program:

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

int main()
```

```
{
    vector<string> wordVec;
    vector<string>::iterator p = wordVec.begin();
    wordVec.push_back("apples");
    wordVec.push_back("oranges");
    while ( p != wordVec.end() )
        cout << *p++ << endl;
    return 0;
}
```

Solution:

The declaration

```
vector<string> wordVec;
```

creates a vector object, but does not allocate any memory for its elements. So if this declaration is followed immediately by the statement

```
vector<string>::iterator p = wordVec.begin();
```

you will get a memory segmentation fault becuase the system will try to read a memory location that does not yet exist. The iterator can be initialized only after the first invocation of `push_back()`.

PROBLEM 5:

The goal of this Java homework is get you to appreciate the *fail-fast* property of the iterator defined for a `List`. Say you are scanning a list, item by item, and as you do so you wish to remove some of the items from the list. Removal of items will cause structural modifications to the list. In general, such structural modifications to a list as it is being iterated through can cause the iterator to exhibit non-deterministic program behavior. To protect a program against such problems (which can also happen when one thread is iterating through a list while another thread is modifying the list), Java makes the list iterator fail-fast. What that means is that if a method detects that a list is being modified structurally as it is being iterated through, the method will throw a `ConcurrentModificationException`. This also applies to the other containers that support iterators in the Java Collections Framework.

Write a Java class `PruneList` that does the following:

1. The class should require two command line arguments for its invocation. That is, the class would be invoked by a command line like

```
java PruneList filename  30
```

where the first argument, `filename`, is the name of the file containing an arbitrary number of integers in one or multiple lines. The second argument, in this case 30, a threshold value to be used for pruning the integer data.

Suggestion: You could construct a `BufferedReader` stream and invoke its `readline()` method to pull into your program each line of the data file at a time:

```
FileReader fr = new FileReader( args[0] );
BufferedReader br = new BufferedReader(fr);
```

2. Read the integers from the file named as described above into an `ArrayList`.

   Suggestion: You could then deploy the `StringTokenizer` class as shown below to extract the individual integers and stuff them into an `ArrayList` as shown below:

```
ArrayList list = new ArrayList();
String line;
while((line = br.readLine()) != null){
    StringTokenizer st = new StringTokenizer(line);
    while(st.hasMoreTokens()){
        list.add(new Integer(st.nextToken()));
    }
}
```

3. Sort the `ArrayList` container by invoking `Collections.sort`. (Sorting is not essential to the main goal of this homework. This step is included only because it makes it easier to see the result and to verify that the program is working correctly.)

4. Iterate through the `ArrayList` and remove from the list all items whose value exceeds the threshold specified by the command line in (a) above.

5. Print out the `ArrayList` before and after the removal of the items.

There are potentially two different ways of removing the items that meet the condition mentioned in (d) above. You could invoke the method `remove( int index )` defined directly for the `ArrayList`, or you could invoke the method `remove()` defined for the `ListIterator` class. However, the former will cause the `ConcurrentModificationException` to be thrown for reasons mentioned at the beginning of this homework problem. However, the latter approach will work just fine.

Solution:

```
//PruneList.java

import java.io.*;
import java.util.*;

class PruneList{

    public static void main(String[] args) throws IOException{
        FileReader fr = new FileReader( args[0] );
        BufferedReader br = new BufferedReader(fr);
        ArrayList list = new ArrayList();

        String line;
        while((line = br.readLine()) != null){
            StringTokenizer st = new StringTokenizer(line);

            while(st.hasMoreTokens()){
                list.add(new Integer(st.nextToken()));
            }
        }

        System.out.println( list );
        Collections.sort( list );
        System.out.println( list );

        int threshold = Integer.parseInt( args[1] );

        ListIterator iter = list.listIterator();
        while( iter.hasNext() ){
            if( ( ( (Integer) iter.next() ).intValue() ) >= threshold )
                iter.remove();
        }

        System.out.println( list );
    }
}
```

PROBLEM 6:

The goal of this homework is to get you to understand some of the subtleties involved in the use of the generic library `remove_if` function for removing elements from

a C++ sequence container. This function removes all elements for which a certain predicate is true, but not really. What it does is that all such elements are taken out of their existing locations, the remaining elements shuffled toward the beginning of the container, and then the "removed" elements placed at the end of the container, in the same order in which they existed originally in the container. So the overall size of the container remains unchanged. At the same time, the function returns the iterator pointing to the beginning of the section of the container where the "removed" elements were placed. If desired, this iterator value can be used to actually erase the "removed" elements, shrinking the size of the container.

Write a C++ program to do the following:

1. Write a C++ program, `PruneList.cc`, that requires two command-line arguments, one for the file containing formatted integer data and the other for an integer value to be used as a threshold in the manner described below. There can be an arbitrary number of integers in the named data file, in an arbitrary number of lines.

2. Establish an input file stream to read all the integers in the named file into a `vector<int>` container.

3. Sort the vector using the generic library `sort`. (This step is not essential to the main goals of this homework. It is included merely because it makes it easier to see the results and to visually verify that the program is working correctly.)

4. Display on your terminal the items in the vector by using the `copy` function from the generic library in the following manner:

   ```
   copy( vec.begin(), vec.end(), ostream_iterator<int>( cout, " " ) );
   ```

   where `vec` is the `vector<int>` object into which you read all of the integers from the data file.

5. Your program should use `remove_if` to eliminate each item from the vector `vec` whose value exceeds the threshold supplied as the second command-line argument in (a) above. This function takes three arguments, the first and the second being the beginning and the ending iterator values for that part of the container where the removal operations are to be conducted. In our case, these arguments can simply be `vec.begin()` and `vec.end()`. The third argument supplies the decision criterion for the removal of elements. The simplest possible call to `remove_if` will look like

   ```
   remove_if( vec.begin(), vec.end(), 30 );
   ```

   This will cause all elements that equal the value 30 to be moved to the end of the container. Obviously, this call will not work for us since we want all elements that are equal to or greater than a threshold to be removed. This can be done by

supplying a function object for the third argument. Here is a possible definition for such a function object:

```
template<class Arg> struct ThresholdCheck : public unary_function<Arg, bool>
{
    int threshold;
    ThresholdCheck( int thresh ) : threshold( thresh ) {};  //constructor
    bool operator() (const Arg& x) { return x >= threshold ? true : false; }
};
```

The class `ThresholdCheck`, defined as a templatized struct, is derived from the base class `unary_function` defined in the `functional` header file. The two template parameters supplied to the base class are for the argument type needed by the overloading of the '()' operator and for the result type returned by this operator. In our case, the overload definition for the '()' operator defines a predicate for comparing each vector element against the threshold.

6. With the function object defined as above, you can now invoke `remove_if` in the following manner:

```
vector<int>::iterator result =
   remove_if( vec.begin(), vec.end(), ThresholdCheck<int>( 50 ) );
```

   if you want 50 to be the decision threshold for the removal of the vector elements.

7. As mentioned before, `remove_if` does not actually remove the elements from the container; it just moves to the end of the container. To get rid of these elements altogether (and to thus shrink the size of the container) you must invoke the `erase()` function on the vector using the iterator returned by `remove_if`:

```
vec.erase( result, vec.end() );
```

8. Display the resulting vector.

Solution:

```
#include <fstream>
#include <vector>
#include <algorithm>
#include <functional>
using namespace std;

template<class Arg> struct ThresholdCheck : public unary_function<Arg, bool>
{
```

```
    int threshold;
    ThresholdCheck( int thresh ) : threshold( thresh ) {};  //constructor
    bool operator() (const Arg& x) { return x >= threshold ? true : false; }
};


int main( int argc, char* argv[] ){
    vector<int> vec;

    if ( argc != 3 )
        cerr << "usage: a.out filename thresholdInt" << endl;

    ifstream in( argv[ 1 ] );
    if (!in) cout << "Cannot open input file" << endl;

    int threshold = atoi( argv[ 2 ] );

    int num = 0;
    while (in >> num ){
        vec.push_back(num);
    }

    sort(vec.begin(), vec.end());

    copy( vec.begin(), vec.end(), ostream_iterator<int>( cout, " " ) );
    cout << endl;

    vector<int>::iterator result =
       remove_if( vec.begin(), vec.end(), ThresholdCheck<int>( threshold ) );

    vec.erase( result, vec.end() );

    copy( vec.begin(), vec.end(), ostream_iterator<int>( cout, " " ) );
    cout << endl;

    return 0;
}
```

PROBLEM 7:

Section 5.1.7 showed a C++ program that used a `map` container to efficiently construct a word histogram for a text file. Implied in that program was the fact that the container kept the `<key, value>` pairs of the histogram in a sorted ascending order by using

the $<$ operator defined for the string type. This comparison operator carries out a character-by-character comparison of two strings using their ASCII codes. This causes the histogram to create separate counts for the same two words if one has upper case letters in it.

The goal of this homework is to modify the earlier program to create a case-insensitive word histogram for a test file. This can be done by incorporating a user-defined comparison function in the `map` declaration:

```
map<string, int, Nocase> hist;
```

where `Nocase` is a class that defines a user-defined comparison function for the keys.

Solution:

```
#include <string>
#include <map>
#include <fstream>
using namespace std;

class Nocase {
public:
  bool operator() ( const string&, const string& ) const;
};

int main()
{
  map<string, int, Nocase> hist;
  ifstream cin_from_file( "inFile" );
  string word;
  while ( cin_from_file >> word )
    hist[ word ]++;

  typedef map<string, int>::const_iterator CI;

  for ( CI iter = hist.begin(); iter != hist.end(); ++iter )
      cout << iter->first << '\t' << iter->second << endl;
}

bool Nocase::operator()( const string& x, const string& y ) const {
  string::const_iterator p = x.begin();
  string::const_iterator q = y.begin();

  while (p != x.end() &&
         q != y.end() &&
```

```
        toupper(*p) == toupper(*q) ) {
    ++p;
    ++q;
  }
  if ( p == x.end() ) return q != y.end();
  return toupper( *p ) < toupper( *q );
}
```

PROBLEM 8:

Modify the Java class `WordHistogram` class of Section 5.2.3 so that it displays the words in the decreasing order of the frequencies associated with the words.

Solution:

```
import java.io.*;
import java.util.*;

class WordHistogram {

  public static void main (String args[]) throws IOException
  {
    String allChars = getAllChars( args[0] );              //(A)

    Map map = new TreeMap();                               //(B)
    StringTokenizer st = new StringTokenizer( allChars );  //(C)
    while ( st.hasMoreTokens() ) {                         //(D)
      String word = st.nextToken();                        //(E)
      Integer count = (Integer) map.get( word );           //(F)
      map.put( word, ( count==null ? new Integer(1)
                : new Integer( count.intValue() + 1 ) ) ); //(G)
    }
    System.out.println( "Total number of DISTINCT words: " + map.size() );
                                                           //(H)

    // The rest of this program makes it possible to display the
    // words in the decreasing order of their frequencies.

    Object[] keys = map.keySet().toArray();
    Object[] values = map.values().toArray();

    Pair[] pairs = new Pair[ keys.length ];
```

```
    for ( int i=0; i < keys.length; i++ )
      pairs[i] = new Pair( (String) keys[i], (Integer) values[i] );

    Arrays.sort( pairs, new PairComparator() );

    for ( int i=0; i < pairs.length; i++ )
      System.out.println( pairs[i].url + "       "  + pairs[i] .count );

  }


  static String getAllChars( String filename ) throws IOException
  {
    String str = "";
    int ch;
    Reader input = new FileReader( filename );
    while ( ( ch = input.read() ) != -1 )
      str += (char) ch;
    input.close();
    return str;
  }
}

class Pair {
  String url;
  Integer count;
  Pair( String u, Integer i ) { url = u; count = i; }
}

class PairComparator implements Comparator {

  public int compare( Object o1, Object o2 )
  {
    Integer int1 = ( (Pair) o1 ).count;
    Integer int2 = ( (Pair) o2 ).count;
    return int2.compareTo( int1 );
  }
}
```

PROBLEM 9:

This is a problem in the run-time resizing of C++ vectors. In what follows, we first provide the reader with an example from Stroustrup that nicely shows why you'd want to resize a vector at run time. The homework problem consists of writing code for this example.

Let's say we have a data source that is producing a stream of non-negative integers and our goal is to make a histogram of these integers and to keep this histogram continually updated as new integers arrive. To remind the reader about histogramming, suppose at some point in time the integers output by the stream was

```
3  0  2  1  1  0  1  0  1  2  3  1
```

our histogram would need to contain four bins. If we used an array to represent the histogram, we could declare the array as

```
int hist[4] = {0};
```

For the data stream shown above, the state of the histogram after that data was read would be

```
hist[0] = 3;
hist[1] = 5;
hist[2] = 2;
hist[3] = 2;
```

since we have three 0's, five 1's, two 2's, and two 3's in the stream. So if i is the next integer read from the data stream, all we'd need to do to update the histogram would be

```
hist[i]++;
```

This array based approach would work fine as long as we knew the largest integer that we expected the data stream to produce. If we don't know the largest integer, a vector based approach would be more suitable. For the vector based approach, we could declare our histogram initially as

```
vector<int> hist(4);
```

When an integer i is read from the data stream, we'd update the histogram just as before, but after we compare the integer with the size of the histogram:

```
if ( i < hist.size() ) hist[i]++;
```

But if it turns out that the histogram does not have a bin for the latest integer because the integer is too large, we could do the following

```
if ( i >= hist.size() ) {
  hist.resize( i + i );
  hist[i]++;
}
```

where the invocation of the function `resize` would increase the size of the vector to twice what's needed for including the integer i in the histogram. Of course, we could also have said just `hist.resize( i + 1 )` or, for that matter, `hist.resize( 10 * i )`. The choice made with `hist.resize( i + i )` may, depending on the properties of a specific data source, be a happy medium between the absolute minimum needed to accommodate the new integer and some upper wild guess for accommodating future integers. The choice made would hopefully reduce the overhead associated with vector resizing, in terms of memory allocation/deallocation and element copying.

Write a C++ program that implements the above.

Solution:

```
#include <vector>
#include <string>
#include <vector>
use namespace std;

void print( vector< int> );
void recordValueInHist( vector<int>&, int );

int main()
{
  vector<int> hist(4);                                  // (A)

  recordValueInHist( hist, 1 );                         // (B)
  recordValueInHist( hist, 1 );
  recordValueInHist( hist, 1 );

  print( hist );
  cout << "size of hist is " << hist.size()
       << endl;                                         // (C)

  recordValueInHist( hist, 8 );                         // (D)
  recordValueInHist( hist, 10 );

  print( hist );
  cout << "size of hist is " << hist.size()
       << endl;                                         // (E)
```

```
}

void recordValueInHist( vector<int>& v, int i ) {    // (F)
  if (i < 0) i = 0;
  if ( i >= v.size() ) v.resize( i + i );            // (G)
  v[i]++;                                            // (H)
}

void print( vector<int> vec ) {
  vector<int>::iterator p = vec.begin();
  while ( p < vec.end() )
    cout <<  *p++ << " ";
  cout << endl;
}
```

The statement at (A) declares `hist` to be a vector of 4 elements. In this example, that means that we can make bin counts for the integers 0, 1, 2, and 3 from the integer stream. The statements at (B) are supposed to capture the arrival of three integers, all 1's, in the integer stream. So the bin corresponding to integer 1 should hold the number 3. This is accomplished by the the `recordValueInHist()` function. The statements at (C) will print out

```
0 3 0 0
size of hist is 4
```

The statements at (D) record the arrival of two integers of value 5. When these integers are sent to the `recordValueInHist()` function, the function discovers there does not exist in the histogram a bin corresponding to the number 5. So `recordValueInHist()` resizes the histogram to double what it would take to accommodate the latest integer. The print statements at (E) will print out

```
0 3 0 0 0 0 0 0 1 0 1 0 0 0 0 0
size of hist is 16
```

Note that the first parameter of the function at (F) is a reference. This is an example of calling a function by reference, a concept that will be explained fully in Chapter 8.

If we were to delete the `&` symbol in the first parameter of the `recordValueInHist` function (and also in the prototype of this function), we will get what's referred to as *call by value*. In keeping with the discussion in Chapter 9, the program here will not exhibit the desired behavior if `recordValueInHist` is called by value because the resizing of the vector in the function will not be visible in `main()`. The only other option would be to use *call by pointer*, also discussed in Chapter 9. But, that is not a good idea for vectors for reasons explained earlier in Chapter 5. So, we have no choice but to use call by reference in this case.

To repeat a comment made at the beginning of Section **??**, if the additional memory needed for a vector to grow upon resizing is not contiguous to the memory block currently allocated, the resized vector may be moved to a new location, all of the previous elements copied over into the new block, and the previously used block deallocated. Therefore, it makes no sense to write functions that have pointers to vector elements. In addition to the explicit resizing of a vector by a call to `resize()`, the functions `push_back()`, `insert()`, and erase() implicitly resize a vector and may move the entire vector to a new location in the memory.

6

The Primitive Types and Their Input/Output

PROBLEM 1:

Is this code fragment legal in C++?

```
int a[10] = {0};
int i;
int* p;
p = &i;
a = &i;
```

Solution:

It is ilegal because of the last statement. You cannot assign to an array name (even though it can be treated like a pointer in other contexts).

PROBLEM 2:

Using the vector data structure in C++, write and test the following functions:

- readAllStrings()

- readAllInts()
- readAllDoubles()

Each function should read all of the data items on a line of the user's terminal and return the items in the form of a list. You must also allow the user to put any number of spaces between the entries. Also, there could be spaces before the first entry.

Solution:

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

int readOneInt();
vector<int> readAllInts();


bool eol = false;
bool intStarted;


int main()
{
  vector<int> dataVec = readAllInts();

  if ( dataVec.size() != 0 ) {
    cout << "Data entered: " << endl;
    vector<int>::iterator iter = dataVec.begin();
    while ( iter != dataVec.end() )
      cout << *iter++ << endl;
  }
  return 0;
}

vector<int> readAllInts() {
  vector<int> vec;
  int n = 0;
  while ( true ) {
    intStarted = false;
    if ( eol == true )
      break;
    n = readOneInt();
    if ( intStarted == true )
```

```
      vec.push_back( n );
  }
  return vec;
}

int readOneInt() {
  int ch;
  int x = 0;
  while ( true ) {
    ch = cin.get();
    if ( ch == '\n' ) {
      eol = true;
      break;
    }
    else if ( ch == ' ' ) break;
    else intStarted = true;
    x = 10 * x + ch - 48;
  }
  if ( intStarted == true )
    return x;
  return 0;
}
```

A More Compact Solution:

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

vector<int> readAllInts();

int main()
{
  vector<int> dataVec = readAllInts();

  if ( dataVec.size() != 0 ) {
    cout << "Data entered: " << endl;
    vector<int>::iterator iter = dataVec.begin();
    while ( iter != dataVec.end() )
      cout << *iter++ << endl;
  }
  return 0;
}
```

```
vector<int> readAllInts() {
  vector <int> vec;
  int x;
  while(true)
    {
      while( cin.peek()==' ')   // is next char in cin a ' ' ?
        cin.get();              // if so, get rid of it
      if ( cin.peek()=='\n')    // is next char a '\n' ?
        return(vec);            // if so, we are done.
      cin >> x;                 // otherwise, grab an int
      vec.push_back(x);         // and store it in vec
    }
}
```

PROBLEM 3:

Write a Java class `TerminalIO` with the following methods for terminal IO:

- readOneString()
- readOneInt()
- readOneDouble()
- readAllStrings()
- readAllInts()
- readAllDoubles()

Each method should read one or all, as the case may be, of the data items on a line of the user's terminal and, for multiple item entry, return the items in the form of an array of the appropriate kind.

Solution:

```
//TerminalIO.java

import java.io.*;
import java.util.*;       // for Vector


class TerminalIO {
```

```
public static String readOneString() {
    String word = "";
    try{
        while ( true ) {
            int ch = System.in.read();
            if ( ch == -1 || ch == '\n' || ch == ' ' ) break;
            word += (char) ch;
        }
    } catch( IOException e ) { e.printStackTrace(); }
    return word;
}

public static int readOneInt() {
    String str = readOneString();
    int data = Integer.parseInt( str );
    return data;
}

public static double readOneDouble() {
    String str = readOneString();
    double data = Double.valueOf( str ).doubleValue();
    return data;
}

public static int[] readAllInts( ) {
    boolean eol = false;
    boolean intStarted = false;
    Vector intVector = new Vector();
    int[] intList = null;
    int x = 0;
    try {
        while ( true ) {
            if ( eol == true ) break;
            while ( true ) {
                int ch = System.in.read();
                if ( ch == '\n' ) {
                eol = true;
                break;
                }
                else if ( ch == ' ' ) break;
                else intStarted = true;
                x = 10 * x + ch - 48;
            }
            if ( intStarted == true )
```

```
                intVector.addElement( new Integer( x ) );
            intStarted = false;
            x = 0;
        }
        Integer[] IntList = new Integer[ intVector.size() ];
        intVector.copyInto( IntList );

        intList = new int[ IntList.length ];
        for ( int i=0; i < IntList.length; i++ )
            intList[i] = IntList[i].intValue();
    } catch( java.io.IOException e ) {}
    return intList;
}

public static String[] readAllStrings( ) {
    boolean eol = false;
    boolean wordStarted = false;
    Vector wordVector = new Vector();
    String[] wordList = null;
    String word = "";
    try {
        while ( true ) {
            if ( eol == true ) break;
                while ( true ) {
                    int ch = System.in.read();
                    if ( ch == '\n' ) {
                        eol = true;
                        break;
                    }
                    else if ( ch == ' ' ) break;
                    else wordStarted = true;
                    word += (char) ch;
                }
                if ( wordStarted == true )
                wordVector.addElement( new String( word ) );
                wordStarted = false;
                word = "";
        }
        wordList = new String[ wordVector.size() ];
        wordVector.copyInto( wordList );
    } catch( java.io.IOException e ) {}
    return wordList;
}

public static double[] readAllDoubles() {
```

```
        String[] strList = readAllStrings();
        double[] doubleList = new double[ strList.length ];
        for (int i=0; i < doubleList.length; i++)
            doubleList[i] = Double.valueOf( strList[i] ).doubleValue();
        return doubleList;
    }

    public static void main( String[] args ) {

        int x = readOneInt();
        System.out.println( "int typed: " + x );
/*
        String str = readOneString();
        System.out.println( "String typed: " + str );

        double[] doubList = readAllDoubles();
        for( int i=0; i<doubList.length; i++)
            System.out.println( doubList[i] );

        String[] strList = readAllStrings();
        for( int i=0; i<strList.length; i++)
            System.out.println( strList[i] );

        int[] numList = readAllInts();
        int sum=0;
        for( int i=0; i<numList.length; i++)
            sum += numList[i];
        System.out.println( "The sum is " + sum );
*/
    }

}
```

PROBLEM 4:

Write a Java class, `FileReaderData`, with the following methods and their return types:

- String[] readAllStrings()
- int[] readAllInts()
- double[] readAllDoubles()

Solution:

```
// FileReaderData.java

import java.io.*;                  // for Reader and FileReader classes
import java.util.*;                // for Vector class

public class FileReaderData {
  static public String[] readAllStrings( String filename )
  {
    Vector Char_vector = new Vector();
    Vector Word_vector = new Vector();
    String[] Word_list = null;
    int ch = 0;
    boolean word_started = false;
    boolean done = false;

    Reader input = null;
    try {
      input = new FileReader( filename );
    }
    catch( IOException e )
    {
      System.out.println( "The data file does not exist. Program terminated." );
      System.exit( -1 );
    }

    while ( !done ) {

      try {
        ch = input.read();
      }
      catch (IOException e) {
        System.out.println( "Read error encontered" );
        done = true;
      }

      if ( ch != -1 && Character.isLetterOrDigit( (char) ch ) ) {
        Char_vector.addElement( new Character( (char) ch ) );
        word_started = true;
      }

      else if ( word_started == true ) {
        char[] char_array = new char[Char_vector.size()];
        for ( int i=0; i<Char_vector.size(); i++ ) {
```

```
        Character Ch = (Character) Char_vector.elementAt(i);
        char_array[i] = Ch.charValue();
      }
      Word_vector.addElement( new String( char_array ) );
      Char_vector = new Vector();
      word_started = false;
    }
    if ( ch == -1 ) done = true;

  }  // end of while loop

  Word_list = new String[ Word_vector.size() ];
  Word_vector.copyInto( Word_list );

  try {
    if ( input != null )
      input.close();
  }
  catch (IOException e) { }

  return Word_list;
}


public static int[] readAllInts( String filename )  {
  Integer[] Int_list = null;
  Vector Int_vector = new Vector();
  int ch = 0;
  boolean word_started = false;
  boolean done = false;
  int I = 0;

  Reader input = null;
  try {
    input = new FileReader( filename );
  }
  catch( IOException e ) {
    System.out.println( "The data file does not exist. Program terminated." );
    System.exit( -1 );
  }

  while ( !done ) {
    try {
      ch = input.read();
    }
```

```
      catch (IOException e) {
        System.out.println( "Error while reading file." );
        done = true;
      }

      if ( ch != -1 && Character.isDigit( (char) ch ) ) {
        I = I*10 + (int)ch - (int)'0';
        word_started = true;
      }
      else if ( word_started == true ) {
        Int_vector.addElement( new Integer( I ) );
        I = 0;
        word_started = false;
      }
      if ( ch == -1 ) done = true;

    } // end of while loop

    Int_list = new Integer[ Int_vector.size() ];
    Int_vector.copyInto( Int_list );

    try {
      if ( input != null )
        input.close();
    }
    catch (IOException e) { }

    int[] int_list = new int[ Int_list.length ];
    for (int i=0; i<Int_list.length; i++)
      int_list[i] = Int_list[i].intValue();
    return int_list;
  }

  // MAIN:
  public static void main( String[] args )
  {
    // For testing readAllStrings:
    String[] Str_list = null;
    Str_list = readAllStrings( "chap1/junk" ) ;

    System.out.println( "The number of words from Str_list.length is " + Str_list.l

    String temp = "";
    for ( int i=0; i<Str_list.length; i++ )
      temp += Str_list[i] + " ";
```

```
    System.out.println( "The words are:  " + temp );


    // For testing readAllInts:
    /*
    int[] num_list = null;
    num_list = readAllInts( "chap1/int_data" );
    System.out.println( "The number of ints in num_list is " + num_list.length );
    for (int i=0; i<num_list.length; i++)
      System.out.println( num_list[i] );
      */
  }
}
```

PROBLEM 5:

As you well know, one cannot always display the content of a file. For example, if a file contains object code, you cannot display it on a screen by reading it as a text file. Yet, it is sometimes useful to peer into such files for diagnostic purposes. This you can do by reading the file one byte at a time and printing out the hex representation of the file.

Write a Java program that creates a hex dump of a file regardless of whether it is an alphanumeric file, a binary file containing object code, or an image or a sound file of some kind. The program should read each byte from the file and print out its two-character hex representation into an output text file. Your program should take command line arguments for the input and the output file names. In other words, after compilation it should be possible to execute your Java program by using an invocation of the following type

```
java  class_name  in_file_name  out_file_name
```

The file `out_file_name` will be the hex dump of the contents of the file `in_file_name`. The hex output in the output file should be formatted so as to show the hex characters for 20 bytes at a time. It should look like:

```
7f 45 4c 46  1  2  1  0  0  0  0  0  0  0  0  0  2  0  2
 0  0  0  1  0  1 10 80  0  0  0 34  0  0 34 84  0  0  0  0
 0 34  0 20  0  5  0 28  0 19  0 17  0  0  0  6  0  0  0 34
       .....
       .....
```

Suggestions:

1. Since the input can be a binary file, use an input stream of type `FileInputStream` for reading the file one byte at a time.

2. The output file will be a regular alphanumeric file (since it will contain the hex characters representing each byte of the input source file). Therefore, you might wish to use a text output stream to feed a binary output stream by invoking

```
PrintWriter out =
     new PrintWriter( new FileOutputStream( ....  ) );
```

3. Recall that `main()` is invoked with the parameter `args` of type `String[]`. What that means is that `args[0]` will be the input file name and the args[1] the output file name. (Note the difference between the indexing of the command-line arguments in C++ and Java.)

4. Use the method `int read()` defined for `FileInputStream` to read one byte at a time. This method returns -1 for denoting the end of a file.

5. To create the hex representation of the value returned by `read()`, use the method `Integer.toHexString( int )` .

Solution:

```
import java.io.*;

class Test {

  public static void main( String[] args )
  {
    int ch = 0;
    FileInputStream fin = null;
    PrintWriter fout = null;

    try {
      fin = new FileInputStream( args[0] );
      fout = new PrintWriter( new FileOutputStream( args[1] ));

      int i = 1;
      while ( true ) {
        ch = fin.read();
        if (ch == -1) break;
        String str = Integer.toHexString( ch );
        if ( str.length() == 1 )  str = " " + str;
        fout.print(str);
        if (i++%20 == 0)
```

```
          fout.println();
        else fout.print( " " );
      }
      fout.print('\n');
      fout.close();
      fin.close();
    } catch (IOException e) { System.out.println( "IO error" ); }
  }
}
```

PROBLEM 6:

Write a C++ program that creates a hex dump of a file regardless of the nature of the file. Using the no-arg version of the `get()` function presented in Section 6.8.2,[1] the program should read each byte from the file, as in

```
ifstream in( "in_file", ios::binary );
....
int ch;
....
while ( ( ch = in.get() ) != EOF ) {
   ....
}
```

and print out its two-character hex representation into an output text file. Your program should take command-line arguments for the input and the output file names. In other words, you should be able to call the program in the following manner:

```
a.out  in_file  out_file
```

The file `out_file` will be the hex dump of the contents of the otherwise unprintable `in_file`. The hex output in the output file should be formatted in the same manner as in the previous Java problem, that is it should show the hex characters for 20 bytes at a time.

Solution:

```
#include <fstream>
```

[1] While it is true that Section 6.8.2 deals with text file I/O, but, as was mentioned in Section 6.8.3, the `get()` functions of Section 6.8.2 can also be used to read the bytes in a binary file provided the file is opened in the binary mode.

```
#include <cstdlib>
#include <iomanip>
using namespace std;

int main(int argc, char** argv)
{
    if (argc != 3) {
        cerr << "Usage is:  a.out infilename outfilename" << endl;
        exit(1);
    }

    ifstream from( argv[1], ios::binary);

    if (!from) {
        cerr << "cannot open input file" << endl;
        exit(1);
    }

    ofstream to( argv[2] );
    if (!to) {
        cerr << "cannot open output file" << endl;
        exit(1);
    }

    int ch = 0, i = 1;
    while ( ( ch = from.get() ) != EOF )
        to << hex << setw(2) << ch << (i++%20?' ':'\n');

    to << '\n';

    from.close();
    to.close();
    return 0;
}
```

PROBLEM 7:

Rework the previous problem by reading the input file with the `read()` function described in Section 6.8.3 for reading N bytes at a time, as in

```
const int N = 1000;
....
ifstream in( "in_file", ios::binary );
```

```
....
unsigned char buffer[N];                                    //(A)
while ( in.read( buffer, N ) ) {
   //output the hex for each byte
}
```

Be sure to use `gcount()` to get any remaining bytes in the input stream when `read()` hits the end-of-file condition.

PROBLEM 8:

With the help of bit patterns, explain precisely why the following replacement for the declaration in line (A) of the program fragment of the previous problem does not work

```
char buffer[N];
```

With this change, you will see a two-character hex representation for most of the bytes in the input file. But every once in a while it will generate an 8-character hex sequence for a single byte of the input file. This longish looking hex sequence would seem like it was generated by a "negative integer." Why?

PROBLEM 9:

A text file with the following integer entries, one per row, is provided to the program shown below. The program tries to read the integers, but instead reads garbage, as shown by the output displayed after the program. Why?

```
//file: input.txt
12
12
12
12
12
23
```

The program is

```
import java.io.*;

class Test {
```

```
    public static void main( String[] args ) {
        try {
            FileInputStream fin = new FileInputStream("input.txt");
            DataInputStream din = new DataInputStream(fin);
            for (int i = 1; i <= 6; i++) {
                System.out.println( din.readInt() );
            }
            fin.close();
        } catch(IOException e) {System.out.println(e);}
    }
}
```

The program produces the following output (the output will vary depending on the underlying architecture of the machine):

```
825362993
839528754
170996234
825362994
java.io.EOFException
```

Solution:

FileInputStream basically reads the raw bytes from the disk file. The readInt() method of the DataInputStream then takes FOUR bytes at a time and converts them into ints. Given this picture, the main source of your problem is that your file is a character file. This is how the bytes are stored in the disk for the file input.txt you sent me:

```
first_byte :  1

second_byte : 2

third_byte : <CR>

fourth_byte: <LF>        (depends on the operating system)

fifth_byte:   1

   .......
```

where the first_byte is the bit pattern for ascii representation of 1, the second_byte the bit pattern for the ascii representation of 2, and so on.

So the first value returned by readInt() will the integer corresponding to the bits patterns for

```
1   2  <CR>  <LF>
```

and that could be a very huge number.

What one needs to do is to read one char at a time by invoking

```
Reader input = new FileReader( your_filename );
char ch = input.read();
```

and then assemble the chars into ints by looking for white spaces.

PROBLEM 10:

As mentioned in Section 6.10, the `java.io` package contains stream classes for direct object level I/O. These, as shown in Figure 6.3, are `ObjectOutputStream` and `ObjectInputStream`. An object can be output directly to an external resource if the class to which the object belongs has implemented the `Serializable` interface. An object is converted into a stream of bytes so that it can be output via an object stream to, say, a disk file. This process is known as *object serialization*. The opposite process, namely reading a stream of bytes through an input object stream and converting those those bytes back into an object is known as *object serialization*.. The byte stream produced for an object through serialization includes a 64-bit long *serial version UID* that is a secure hash of the full class name, its super-interfaces, its members. This unique identifier is used during object input to discover any incompatibilities with the existing classes in the Java Virtual Machine.

Shown below is a simple class `User` that is serializable because it implmements the `Serializable` interface. Notice how the object streams are set up in lines (A) and (B) and how the `writeObject()` method of `ObjectOutputStream` is invoked to write the objects out to a disk file and how the `readObject()` method of `ObjectInputStream` is invoked to read the objects back from the disk file.

```
//ObjectIO.java

import java.io.*;


class User implements Serializable {
    private String name;
    private int age;


    public User( String nam, int yy ) { name = nam;  age = yy; }
    public String toString() { return "User: " + name + "  " + age ; }
```

```
    public static void main( String[] args ) throws Exception {
        User user1 = new User( "Melinda", 33 );
        User user2 = new User( "Belinda", 43 );
        User user3 = new User( "Tralinda", 53 );

        FileOutputStream os = new FileOutputStream( "object.dat" );
        ObjectOutputStream out = new ObjectOutputStream( os );

        out.writeObject( user1 );
        out.writeObject( user2 );
        out.writeObject( user3 );

        out.flush();
        os.close();

        FileInputStream is = new FileInputStream( "object.dat" );
        ObjectInputStream in = new ObjectInputStream( is );

        User user4 = (User) in.readObject();
        User user5 = (User) in.readObject();
        User user6 = (User) in.readObject();

        is.close();

        System.out.println( user4 );
        System.out.println( user5 );
        System.out.println( user6 );
    }
}
```

The straightforward approach shown above works only for simple classes. For a more complex class, such as when the class `User` is provided with an array data member, this approach would not work. Now you would need to provide the class with two methods of the following exact signatures:

```
    private void writeObject( ObjectOutputStream out )
                throws IOException

    private void readObject( ObjectInputStream in )
                throws IOException, ClassNotFoundException
```

When a class is endowed with these functions, the `ObjectOutputStream`'s `writeObject` will call the `writeObject` method defined as above for the serialization process. The same goes for the `ObjectInputStream`'s `readObject` method vis-a-vis the `readObject` defined as above for the class in question.

The goal of this homework is to create a serializable version of the following class by providing implementation code for the methods `writeObject` and `readObject` in accordance with the signatures shown above. Your implementation of these methods would need to write out each data member of the class and each element of the array separately by using the `writeInt` and `writeString` (or `writeUTF`) methods of the `ObjectOutputStream` class. To deserialize the byte streams, you'd need to use the `readInt` and `readString` (or `readUTF`) of the `ObjectInputStream` class for each data member of the class and also for each element of the array.[2]

```
class User implements Serializable {
    private String name;
    private int age;
    transient private String[] children;
    private int numChildren;

    // constructor(s)

    // implementation code for writeObject()

    // implementation code for readObject()
}
```

Solution:

```
//ObjectIO2.java

import java.io.*;
import java.util.*;

class User implements Serializable {
    private String name;
    private int age;
    transient private String[] children;
    private int numOfChildren;


    public User( String nam, int yy, String[] kids ) {
        name = nam;
```

[2]Note that we have marked the array data member `children` as *transient*. You declare a class data member *transient* if you do not want it to be serialized. So for the class shown we do not want the array reference to be serialized; however, you'd still want to serialize each element of the array. Your implementation code for the `readObject` method should then create a fresh array reference for `children` and then read off each element of the array from the byte stream into the array.

```
        age = yy;
        numOfChildren = kids.length;

        children = new String[ kids.length ];
        for ( int i = 0; i < kids.length; i++ )
            children[i] = kids[i];
    }


    private void writeObject(ObjectOutputStream s) throws IOException {

        s.defaultWriteObject();     //write out any hidden stuff

        s.writeUTF(name);
        s.writeInt(age);
        s.writeInt(numOfChildren);

        for(int i = 0; i < children.length; i++)
            s.writeUTF( children[i] );
    }


    private void readObject(ObjectInputStream s)
                    throws IOException, ClassNotFoundException {
        s.defaultReadObject();      //read in any hidden stuff

        name = s.readUTF();
        age = s.readInt();
        numOfChildren = s.readInt();

        children = new String[ numOfChildren ];
        for(int i = 0; i < children.length; i++)
            children[i] = s.readUTF();
    }


    public String toString() {
String str = "User: " + name + ",  " + age + ",  ";
for(int i = 0; i < children.length; i++)
  str+= children[i] + " ";
return str;
    }


    public static void main( String[] args ) throws Exception {
```

```
        String[] arr = new String[]{ "holly", "dolly", "polly", "tolly", "oily" };
        String[] arr2 = new String[]{ "cool", "tool", "stool" };

        User user1 = new User( "Melinda", 33, arr );
        User user2 = new User( "Belinda", 43, arr );
        User user3 = new User( "Tralinda", 53, arr );
        User user4 = new User( "Tralinda", 53, arr2 );

        FileOutputStream os = new FileOutputStream( "object.dat" );
        ObjectOutputStream out = new ObjectOutputStream( os );

        out.writeObject( user1 );
        out.writeObject( user2 );
        out.writeObject( user3 );
        out.writeObject( user4 );

        out.flush();
        os.close();

        FileInputStream is = new FileInputStream( "object.dat" );
        ObjectInputStream in = new ObjectInputStream( is );

        User user5 = (User) in.readObject();
        User user6 = (User) in.readObject();
        User user7 = (User) in.readObject();
        User user8 = (User) in.readObject();

        is.close();

        System.out.println( user5 );
        System.out.println( user6 );
        System.out.println( user7 );
        System.out.println( user8 );
    }
}
```

7

Declarations, Definitions, and Initializations

PROBLEM 1:

Why does this C++ program not compile?

```
class X {
    int i;
public:
    X( int ii ) { i = ii; }
};

int main() { X xarray[10]; }
```

Solution:

Because the class X does not possess a default no-arg constructor since it has been provided with a constructor of its own.

PROBLEM 2:

Is this C++ program legal?

```
class X {
    int i;
 };

 int main() { X xarray[10]; }
```

Solution:

Yes, because the class will be provided with a language-supplied no-arg constructor.

PROBLEM 3:

The class `X` does not possess a no-arg constructor, either user-defined or system-supplied. Nonetheless, `main` declares a vector as shown. Will this C++ program compile?

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

class X {
    int i;
public:
    X( int ii ) { i = ii; }
};

int main() { vector<X> xvec; }
```

Solution:

Yes, because xvec is a vector of zero size. So at this time no memory needs to be allocated.

PROBLEM 4:

This C++ program does not compile. Why?

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

```
class X {
    int i;
public:
    X( int ii ) { i = ii; }
};

int main() { vector<X> xvec(100); }
```

Solution:

No. Now it needs to allocate memory for for a vector that would hold 100 X objects. For that it needs to know how much memory to allocate for each X object, for which it needs to have access to the no-arg constructor of X.

PROBLEM 5:

Will this C++ program compile and execute?

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

class X { int i; };

int main() { vector<X> xvec; }
```

Solution:

Yes, because the class is provided with a language-supplied no-arg constructor. But the availability of this constructor is immaterial here anyway since xvec is only being declared.

PROBLEM 6:

Will this C++ program compile? If it does compile, will it do what the programmer mostly likely wanted it to do?

```
#include <string>
```

```
using namespace std;

class User {
    string name;
};

class UserGroup {
    User chief;
    User uList[10];
};

int main() { UserGroup ug(); }
```

Solution:

Yes, it will compile. But in all likelihood it will not serve its intended function. This program declares ug to be function taking void arguments and returning an object of type UserGrop. The statement inside main() will be taken to be a function prototype

PROBLEM 7:

What is the output of this C++ program?

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

class User {
public:
    string name;
};

class UserGroup {
public:
    User chief;
    User uList[10];
};

int main()
{
    UserGroup ug;
    cout << ug.chief.name;
    cout << ug.uList[0].name;
}
```

Solution:

This program will print out a null string (meaning nothing) for both the cout statements.

PROBLEM 8:

It was mentioned in Section 7.2 that when a programmer does not provide a class with any constructors, the class does not get a system-supplied default no-arg constructor if it has a `const` data member. In the following program, the class `Y` has not been provided with any constructors at all and it has a `const` data member. Class X has been provided with a no-arg constructor as shown. The program compiles and runs without errors. Why?

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

class Y {
public:
     const int yval;
};

class X {
public:
     Y y;
     int xval;
     X() {}
};

int main() {
    X x;
    cout<<x.y.yval<<endl;
    return 0;
}
```

Solution:

Because X has been provided with a no-arg constructor. So the compiler does not look for a no-arg constructor for the default initialization of the y data member of X. The constructor invocation

```
X(){}
```

does NOT call any constructors for the data members of `X`. It is a do-nothing invocation of the constructor, meaning that whatever happens to be in the memory allocated to the data members of `X` will do. So the no-arg constructor for `Y` will never be called.

8

Object Reference and Memory Allocation

PROBLEM 1:

How many times will the destructor of class `X` be invoked as the variables go out of scope when the fbw of execution hits the right brace of `main` in the following C++ program? Note that `main` consists of four groupings of statements: the 'A' grouping, the 'B' grouping, the 'C' grouping, and the 'D' grouping. In each grouping, the fi rst statement constructs a new object, with the other statements creating references to the previously created objects or references. So you need to ask yourself that as the variables `xobj1`, `xobj2`, and `xobj3` from the 'A' grouping go out of scope, how many times will `X`'s destructor be invoked. And so on for the other three groupings.

```
//Reference.cc

#include <iostream>
using namespace std;

class X {
    int n;
public:
    X( int nn ) : n(nn) {}
    ~X() {
        cout << "destructor invoked for X obj with n= " << n
             << endl;
    }
};
```

```
int main() {
    X xobj1( 100 );                                          //(A1)
    X& xobj2 = xobj1;                                        //(A2)
    X& xobj3 = xobj2;                                        //(A3)

    const X& xobj4 = X(200);                                 //(B1)
    const X& xobj5 = xobj4;                                  //(B2)
    const X& xobj6 = xobj5;                                  //(B3)

    X* p = new X(300);                                       //(C1)
    X*& q = p;                                               //(C2)
    X*& r = q;                                               //(C3)
    delete r;                                                //(C4)

    const X* s = new X(400);                                 //(D1)
    const X*& t = s;                                         //(D2)
    const X*& u = t;                                         //(D3)
    delete u;                                                //(D4)

    return 0;
}
```

Solution:

Four times, once for each grouping of lines in main().

PROBLEM 2:

Note carefully the difference between the `main` of the program in the previous problem and the `main` below. The statements that invoke the `delete` operator appear to be at odd locations — look at the statements in lines (C2) and (D2). The new `main` also includes in lines (E1) and (E2) additional statements involving a `const` pointer. Now answer the following questions:

1. Will this program compile?
2. Assuming this program compiles, are there any obvious memory leaks in the program? Is the destruction of the object constructed in line (E1) an issue from the standpoint of a memory leak?
3. Assuming that this program compiles and runs to completion, how many times will the destructor of the class `X` be invoked?

```
//Reference2.cc

#include <iostream>
```

```
using namespace std;

class X {
    int n;
public:
    X( int nn ) : n(nn) {}
    ~X() {
        cout << "destructor invoked for X obj with n= " << n
             << endl;
    }
};


int main() {
    X xobj1( 100 );                                          //(A1)
    X& xobj2 = xobj1;                                        //(A2)
    X& xobj3 = xobj2;                                        //(A3)

    const X& xobj5 = X(200);                                 //(B1)
    const X& xobj6 = xobj5;                                  //(B2)
    const X& xobj7 = xobj6;                                  //(B3)

    X* p = new X(300);                                       //(C1)
    delete p;                                                //(C2)
    X*& q = p;                                               //(C3)
    X*& r = q;                                               //(C4)

    const X* s = new X(400);                                 //(D1)
    delete s;                                                //(D2)
    const X*& t = s;                                         //(D3)
    const X*& u = t;                                         //(D4)

    X* const w = new X(500);                                 //(E1)
    *w = X(600);                                             //(E2)

    return 0;
}
```

Solution:

The program compiles fine. When you invoke delete on a pointer, the memory address held by the pointer does not change. So, as a variable, a pointer stays intact after the invocation of delete on it.

No, there are no memory leaks in the program. The memrory allocated to the object X(500) gets filled with the X(600) object. The destructor will be invoked for five out of six objects constructed in the program. There will be no destructor invocation for X(500). Note again that

there are no memory leaks even though the destructor is NOT invoked for the X(500) object. As mentioned, this object gets overwritten by the X(600) object.

PROBLEM 3:

Numerically intensive applications often require extensive memory allocation and deallocation. As an exercise that is somewhat numerically intensive, write a C++ program that extracts the numerical information contained in a JPEG image file. Note that this exercise is not about decoding a JPEG image, but about allocating (and, when so needed, deallocating) the memory needed for an array representation of the data generated by a publically available decoding function such as `djpeg`. With appropriate flags, a function such as `djpeg` can convert an encoded JPEG image into a PPM file in which each image pixel is represented by three consecutive bytes for the three RGB components of color at that pixel. The `djpeg` function also deposits at the top of the PPM file a header that contains information about the height of the image, its width, etc. More specifically, the format of a PPM file is

```
comment lines that start with the character #
the label P6 for color image and P5 for B&W
579 768     (these two numbers are the width and the height)
255         (the number of quantization levels for each color)
The rest of the data consists of 3 * width * height bytes.
Each pixel in the image is represented by three consecutive
bytes, the first for Red, the second for Green, and the third
for Blue.
```

Your program must read this header and determine whether the image is color or just black-and-white. Your program must then extract information about the height and the width of the image and the number of quantization levels for each color. Your program must also include a filter function that allows you do simple pixel manipulation (such as subtracting adjacent pixels to enhance edges). Finally, your program should invoke another publically available function, such as `cpeg` for converting the processed PPM file back into an output JPEG image.

Your program should be in the form of a class called `JpegToData` with the following data members:

```
int width

int height

int quantLevels

string colorLabel

int* pixelArray

int* resultArray
```

where `width` is the width of the image, `height` the height of the image, `quantLevels` the number of quantization levels for each of the colors, `pixelArray` the array of numbers corresponding to the pixels of the Jpeg image, and `resultArray` the array obtained by applying the filter to the contents of `pixelArray`. You need to provide the `JpegToData` class with at least the following methods:

```
void imageFilter()

void toJpeg()
```

The `imageFilter` method when applied to an object of type `JpegToData` numerically processes the `pixelArray` with some sort of a filter function and puts the result in `resultArray`. The `toJpeg` method creates a JPEG image out of the contents of `resultArray` by using the `cjpeg` function.

Help:

For executing the "djpeg" and "cjpeg" commands in C++, you'd need to invoke the `system` command as in

```
system( command_string_as_a_C_style_string );
```

In addition, your C++ program will most likely use the `get()`, `get(ch)`, and `peek()` functions defined for input stream objects in order to extract information from the ppm file, as in the following snippets of code:

```
ifstream in( ``temp.ppm'' );

if ( '#' == in.peek() )        // for getting rid of the comment block
  while( '\n' != in.get() )
    ;

in >> colorLabel;
in >> width;
....
....

int i = 0;
unsigned char pix;
while ( in.get( pix ) )
  pixelArray[ i++ ] = pix;

.....
```

and the `put( ch )` method defined for the file output stream for writing the pixels back into an output ppm file:

```
ofstream out( ``tempout.ppm'' );
//....
```

```
int* ptr = resultArray;
for ( int i = 0; i < 3 * wideth * height; i++ )
  out.put( (unsigned char) *ptr++ );
// ...
```

Solution:

```
// JpegToData.cc
/*
 *  This program takes a Jpeg image, converts it into a PPM file,
 *  applies a filter to the PPM file, and then reconverts the result
 *  into a Jpeg image that can be viewed with the help of a web browser.
 *
 *  This class captures the numerical attributes of a Jpeg image.
 *  To construct an object of this class, use the syntax
 *
 *           JpegToData  imageData( filename.jpeg );
 *
 *  where 'filename.jpeg' is a C-style string of type char* and where
 *  imageData is the name of the data object you want to create for
 *  the Jpeg image.
 *
 *  Once a data object, say imageData, is created, a filter can be
 *  can be applied to the data object by invoking
 *
 *           imageData.imageFilter();
 *
 *  The filtered image can be transformed back into a Jpeg image by
 *
 *           imageData.toJpeg();
 *
 *  Note the following members of this class:
 *
 *    int width                width of the image in pixels
 *
 *    int height               height of the image in pixels
 *
 *    int quantLevels          255 for most images
 *
 *    string colorLabel        P6 for color images and P5 for B&W
 *
 *    int* pixelArray          This is the array of pixels in the
 *                             jpeg image
 *    int* resultArray         If any filtering is done on the
 *                             image, the result is stored in
 *                             this array.
 *
```

```
 * The methods defined for this class are
 *
 *    JpegToData( char* filename )             constructor
 *
 *    JpegToData()                             default constructor
 *
 *    ~JpegToData()                            destructor
 *
 *    void setParameters( char* filename )     used by constructor
 *
 *    int toInteger( string str )              string to int conversion
 *
 *    void imageFilter();      takes the horizontal derivative of
 *                             the image;  the purpose of this filter
 *                             is to demonstrate how to set up the
 *                             iterations loops for pixel processing
 *
 *    void toJpeg();           reconverts a PPM file into a Jpeg image
 *
 *
 *  The numerical attributes of a jpeg image are extracted by
 *  first converting it into a PPM image by using the unix
 *  djpeg function.  The format of a PPM image is as follows
 *
 *      comment lines that start with the character #
 *      the label P6 for color image and P5 for B&W
 *      579 768     (these two numbers are the width and the height)
 *      255         (the number of quantization levels for each color)
 *      The rest of the data consists of 3 * width * height ints.
 *      Each pixel in the image is represented by three consecutive
 *      bytes, the first for Red, the second for Green, and the third
 *      for Blue.
*/

#include<fstream>
#include <string>
#include <cstdlib>            // for exit(), system()
#include <cctype>             // for isspace()
using namespace std;

class JpegToData {
public:
  int width;
  int height;
  int quantLevels;
  string colorLabel;
  int* pixelArray;
  int* resultArray;
```

```
//constructor:
JpegToData( string filename ) {
  setParameters( filename );
}

//default constructor:
JpegToData() {
  cout << "Default initializion of a 1000x1000x3 image" << endl;
  width = 1000;
  height = 1000;
  quantLevels = 255;
  colorLabel = "P6";

  int size = 3 * width * height;
  pixelArray = new int[ size ];
  resultArray = new int[ size ];

  for( int i=0; i < size; i++ ) {
    pixelArray[ i ] = 0;
    resultArray[ i ] = 0;
  }
}

//destructor:
~JpegToData() {
  delete [] pixelArray;
  delete [] resultArray;
}

//set class members from Jpeg data:
void setParameters( string imageName ) {
  string commandString = "djpeg -pnm < "
                    + imageName  + " > temp.ppm";
  system( commandString.c_str() );
  cout << "Finished conversion of JPEG image to PPM file" << endl;

  ifstream in( "temp.ppm" );
  if ( !in ) {
    cerr << "cannot open the ppm file" << endl;
    exit( 1 );
  }

  if ( '#' == in.peek() )        // get rid of comment block
    while ('\n' != in.get())
          ;

  in >> colorLabel;
  in >> width;
  in >> height;
```

```
  in >> quantLevels;

  if ( colorLabel != "P6" ) {
    cout << "Sorry. This program accepts only color images." << endl;
    exit(0);
  }

  pixelArray = new int[ 3 * width * height];

  while ( isspace( in.peek() ) )
    in.get();

  int i = 0;
  unsigned char ch_pix;
  while ( in.get( ch_pix ) )
      pixelArray[ i++ ] = ch_pix;

  in.close();
  system( "rm temp.ppm" );

  resultArray = new int[ 3 * width * height];
}

//an example image filter:
void imageFilter() {
  cout << "Applying image filter" << endl;

  int size = 3 * width * height;

  for( int i=0; i < size; i++ )
    resultArray[ i ] = pixelArray[ i ];

  int P = 0;
  int PE = 0;

  for (int colorOffset = 0; colorOffset < 3; colorOffset++) {
    for (int i=5; i < width - 5; i++ ) {
      for (int j=5; j < height - 5; j++ ) {
        P  = 3*j*width + 3*i + colorOffset;
        PE = P - 3;
        resultArray[ P ] = pixelArray[ P ]  -  pixelArray[ PE ];
      }
    }
  }
  cout << "Finished applying image filter" << endl;
}


//coverting processed data back into a Jpeg image:
```

```
  void toJpeg() {
    cout << "Converting PPM file to JPEG image" << endl;

    ofstream cout( "temp1.ppm" );
    if ( !cout ) {
      cerr << "Unable to open output ppm file";
      exit(0);
    }

    cout << colorLabel;
    cout << endl;
    cout << width << " " << height << endl;
    cout << quantLevels << endl;

    int* ptr = resultArray;
    for ( int i = 0; i < 3*width*height; i++ )
      cout.put( (unsigned char) *ptr++ );

    cout.close();

    system( "cjpeg < temp1.ppm > output.jpg" );
    system( "rm temp1.ppm" );

    cout << "Output is in the image file output.jpg" << endl;
  }
};


int main()
{

  JpegToData* imageData1 = new JpegToData( "input.jpg" );
  imageData1->imageFilter();
  imageData1->toJpeg();

  /*
  imageData1 = new JpegToData("gogh.jpg");
  imageData1->imageFilter();
  imageData1->toJpeg();

  JpegToData imageData2( "gogh.jpg" );
  imageData2.imageFilter();
  imageData2.toJpeg();
  */
  return 0;
}
```

PROBLEM 4:

Do the same as in the previous question but for Java. (The reader should note that Java comes with a class `java.awt.image.BufferedImage` class that gives you access to the numerical data for the pixels in an image.)

Solution:

```
// JpegToData.java

import java.io.*;

public class JpegToData {
  public int width;
  public int height;
  public int quantLevels;
  public String colorLabel;
  public int[] pixelArray;
  public int[] resultArray;

  //constructor:
  public JpegToData( String filename ) {
    setParameters( filename );
  }

  //default constructor:
  JpegToData() {
    System.err.println( "Default initializion of a 500x500x3 image" );

    width = 500;
    height = 500;
    quantLevels = 255;
    colorLabel = "P6";

    int size = 3 * width * height;
    pixelArray = new int[ size ];
    resultArray = new int[ size ];

    for( int i=0; i < size; i++ ) {
      pixelArray[ i ] = 0;
      resultArray[ i ] = 0;
    }
    System.err.println( "Default initializion finished" );
  }

  // extracts from the image a value for each member:
  public void setParameters( String imageName ) {
```

```
String commandString = "djpeg -pnm < " + imageName + " > temp.ppm" ;
String[] cmd = {"/bin/sh", "-c", commandString};
String[] propertyNameArray = {"", "", "", ""};
FileInputStream fin = null;
int ch = 0;

try {
  Process child = Runtime.getRuntime().exec(cmd);     // throws IOException
  child.waitFor();                                    // throws InterruptedException
  System.out.println( "Finished conversion of JPEG to PPM" );
} catch (IOException e) {
  System.err.println( "IO error: " + e );
} catch (InterruptedException e1) {
  System.err.println( "Exception: " + e1.getMessage() );
}

try {
  fin = new FileInputStream( "temp.ppm" );
} catch ( FileNotFoundException e ) {
  System.err.println( "Input file not found" );
  System.exit(0);
}

InputStreamReader rin = new InputStreamReader( fin );
BufferedReader bin = new BufferedReader( rin );

try {
  int i = 0;
  while ( i < 4 ) {
    while ( -1 != (ch = bin.read())) {              // throws IOException
      if ( '#' == ch ) {
        while ('\n' != bin.read())
          ;
        break;
      }
      if ( !Character.isLetterOrDigit( (char) ch ) )
        break;
      propertyNameArray[ i ] += new Character( (char) ch );
    }
    i++;
  }

  if ( !propertyNameArray[0].equals( "P6" ) ) {
    System.err.println( "Sorry.  This program accepts only color images. " );
    System.exit( 0 );
  }

  colorLabel =                propertyNameArray[ 0 ]  ;
  width =   Integer.parseInt( propertyNameArray[ 1 ] );
```

```
    height =  Integer.parseInt( propertyNameArray[ 2 ] );
    quantLevels = Integer.parseInt( propertyNameArray[ 3 ] );
  } catch (IOException e) {
    System.err.println( "IO error: " + e );
  }

  pixelArray = new int[ 3 * width * height];

  try {
    int i = 0;
    while ( -1 != (ch = bin.read()))               // throws IOException
      pixelArray[ i++ ] = ch;
    bin.close();
  } catch (IOException e) {
    System.err.println( "IO error in stuffing pixel array: " + e );
  }

  String cmd2 = "rm temp.ppm";

  try {
    Process child = Runtime.getRuntime().exec(cmd2);     // throws IOException
    child.waitFor();                                     // throws InterruptedException
  } catch (IOException e) {
    System.err.println( "IO error: " + e );
  } catch (InterruptedException e1) {
    System.err.println( "Exception: " + e1.getMessage() );
  }

  resultArray = new int[ 3 * width * height];

  System.out.println( "Image data object constructed" );
}


// an example image filter:
public void imageFilter() {
  System.out.println( "Applying image filter" );

  int size = 3 * width * height;

  for( int i=0; i < size; i++ )
    resultArray[ i ] = pixelArray[ i ];

  int temp = 0;
  int P = 0;
  int PE = 0;

  for (int colorOffset = 0; colorOffset < 3; colorOffset++) {
    for (int i=5; i < width - 5; i++ ) {
```

```
      for (int j=5; j < height - 5; j++ ) {
        P  = 3*j*width + 3*i + colorOffset;
        PE = P - 3;
        resultArray[ P ] = pixelArray[ P ]  -  pixelArray[ PE ];
      }
    }
  }
  System.out.println( "Finished applying image filter" );
}

// converting PPM file back into a JPEG image:
void toJpeg( String outputFilename ) {
  System.out.println( "Converting PPM file to JPEG image" );
  FileOutputStream fout =  null;

  try {
    fout = new FileOutputStream( "temp1.ppm" );
    BufferedOutputStream bout = new BufferedOutputStream( fout );
    DataOutputStream dout = new DataOutputStream( bout ); // for binary output

    dout.writeBytes( colorLabel + "\n" );
    dout.writeBytes( width + " " + height  + "\n" );
    dout.writeBytes( quantLevels  + "\n" );

    for ( int i = 0; i < 3*width*height; i++ )
      dout.write( resultArray[ i ] & 0xffff );
    dout.close();

  } catch (IOException e) { System.err.println( "IO error: " + e ); }

  String commandString1 = "cjpeg < temp1.ppm >" +  outputFilename ;
  String[] cmd1 = {"/bin/sh", "-c", commandString1};

  String commandString2 = "rm temp1.ppm";

  try {
    Process child1 = Runtime.getRuntime().exec(cmd1);     // throws IOException
    child1.waitFor();                                    // throws InterruptedException
    Process child2 = Runtime.getRuntime().exec(commandString2);    // throws IOException
    child2.waitFor();                                    // throws InterruptedException
    System.out.println( "Finished conversion of PPM to JPEG" );
  } catch (IOException e) {
    System.err.println( "IO error: " + e );
  } catch (InterruptedException e1) {
    System.err.println( "Exception: " + e1.getMessage() );
  }

  System.out.println( "Output image deposited in output.jpg" );
}
```

```
  public static void main( String[] args )  {

    JpegToData imageData1 = new JpegToData( "input.jpg" );
    imageData1.imageFilter();
    imageData1.toJpeg( "output1.jpg" );

    /*
    JpegToData imageData2 = new JpegToData("gogh.jpg");
    imageData2.imageFilter();
    imageData2.toJpeg( "output2.jpg" );
    */

    JpegToData composite = new JpegToData();
    System.out.println( composite.width + " " + composite.height );

  }

}
```

9

Functions and Methods

PROBLEM 1:

In the following program, `main` calls a function `g()` in line (B) and passes the argument to this function by value. The function `g()` is defi ned in line (A). How many times will the destructor of class `X` be invoked as the program runs to completion.

```
//CountDestructorInvoc1.cc

#include <iostream>
using namespace std;

class X {
    int n;
public:
    X( int nn ) : n(nn) {}
    ~X() {
        cout << "destructor invoked for X obj with n= " << n
             << endl;
    }
};

void g( X x ) {}                                          //(A)

int main() {
    X xobj( 100 );
```

```
    g( xobj );                                                  //(B)
    return 0;
}
```

Solution:

Two times on Solaris, one for the local object `xobj` in main and the other for the local object `x` inside the function `g()` since the arg is passed by value.

PROBLEM 2:

The program below differs from the program of the previous problem in only one small respect: the function call in line (B) passes the argument to the function in line (A) by reference. How many times will the destructor of class `X` be invoked in this case as the program runs to completion?

```
//CountDestructorInvoc2.cc

#include <iostream>
using namespace std;

class X {
    int n;
public:
    X( int nn ) : n(nn) {}
    ~X() {
        cout << "destructor invoked for X obj with n= " << n
             << endl;
    }
};

void g( X& x ) {}                                               //(A)

int main() {
    X xobj( 100 );
    g( xobj );                                                  //(B)
    return 0;
}
```

Solution:

Only one time on Solaris. Only one X object will be in main. Since this object is passed by reference to the function in line (A), no additional X objects will be constructed.

PROBLEM 3:

In a further variation on the programs shown in the previous two problems, the function `g()` in line (A) now takes an `X` argument by value and returns an object of type `X`. How many times will the destructor of `X` be invoked as the program runs to completion?

```
//CountDestructorInvoc3.cc

#include <iostream>
using namespace std;

class X {
    int n;
public:
    X( int nn ) : n(nn) {}
    ~X() {
        cout << "destructor invoked for X obj with n= " << n
             << endl;
    }
};

X g( X x ) { return x; }                                        //(A)

int main() {
    X xobj( 100 );
    g( xobj );                                                  //(B)
    return 0;
}
```

Solution:

Three on Solaris. main makes a local object of type X. When this object is passed by value to the function in (A), another local object of type X is constructed in g(). Finally, when g() executes its return statement, it makes one more object of type X.

PROBLEM 4:

In another variation on the programs shown in the previous homework problems, the function `g()` in line (A) below takes an argument of type `X` by reference and returns an object of type `X`. How many times will `X`'s destructor be invoked as this program runs to completion?

```
//CountDestructorInvoc4.cc
```

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

class X {
    int n;
public:
    X( int nn ) : n(nn) {}
    ~X() {
        cout << "destructor invoked for X obj with n= " << n
             << endl;
    }
};

X g( X& x ) { return x; }                                    //(A)

int main() {
    X xobj( 100 );
    g( xobj );                                               //(B)
    return 0;
}
```

Solution:

Two on Solaris. One for the local X object constructed inside main and the other for the X object made when the return statement in line (A) is executed. The object returned by g() will be a copy of the object that is supplied to g() by reference.

PROBLEM 5:

In yet another variation on the previous programs, the function `g()` in line (A) now takes its `X` argument by reference and also returns an `X` object by reference. How many times will `X`'s destructor be now invoked as the program runs to completion?

```
//CountDestructorInvoc5.cc

#include <iostream>
using namespace std;

class X {
    int n;
public:
    X( int nn ) : n(nn) {}
    ~X() {
        cout << "destructor invoked for X obj with n= " << n
```

```
            << endl;
    }
};

X& g( X& x ) { return x; }                                    //(A)

int main() {
    X xobj( 100 );
    g( xobj );                                                //(B)
    return 0;
}
```

Solution:

Only once. There is one and only X object constructed in this program and that is the xobj object in main(). This object is passed by reference to the function in line (A). That function returns the same object by reference.

PROBLEM 6:

In one last variation on the previous homework programs, the function `g()` in line (A) now takes its `X` argument by value. It then a returns a reference to the local variable `x`. Since, in general, returning a reference to a local variable is an error, will this program compile? If the program were to compile and it ran without problems, how many times would `X`'s destructor be invoked?

```
//CountDestructorInvoc6.cc

#include <iostream>
using namespace std;

class X {
    int n;
public:
    X( int nn ) : n(nn) {}
    ~X() {
        cout << "destructor invoked for X obj with n= " << n
             << endl;
    }
};

X& g( X x ) { return x; }                                     //(A)

int main() {
```

```
    X xobj( 100 );
    g( xobj );                                                    //(B)
    return 0;
}
```

Solution:

The program compiles and runs fine on both Linux and Solaris with g++ and CC. (One would think that this program should not compile since the function g is returning a reference to a local variable that is about to go out of scope. The destructor is invoked exactly twice, once for the local xobj in main and the other for the X object in g() created by passing the argument by value.

PROBLEM 7:

Provide implementations for the overloaded versions of the constructor, the member function `setLastName()`, and of the function `modifyLastName()` in the following code:

```
class User {
    string firstName;
    string lastName;
    int age;
public:
    User(string first, string last, int yy);                  //(A)
    User(const string& first, const string& last, int yy);    //(B)

    void setLastName( string newLastName );
    void setLastName( string* newLastName );

    string getFirst(){ return firstName; }
    string getLast(){ return lastName; }
};

User modifyLastName( User u, const string newName );          //(C)

User& modifyLastName( User& u, const string& newName );       //(D)

User* modifyLastName( User* p, string* const ptr );           //(E)
```

By inserting print statements in the implementations for the two constructors and the three version of the `modifyLastName()` function, find out what function calls will invoke which versions. In particular, state what you'd need to do to your program so that the version of the constructor in line (B) and the version of the function `modifyLastName` in line (D) will be invoked by the calls you see in the following `main`:

```
int main()
{
    User u1( "joe", "schmo", 88 );
    User u2 = modifyLastName( u1, "sixpack" );
    User& u3 = u1;
    User u4 = modifyLastName( u3, "smith" );
    User& u5 = modifyLastName( u3, "hardhat" );
    cout << "First name: " << u4.getFirst()
         << "  Last name: " << u4.getLast() << endl;
    return 0;
}
```

PROBLEM 8:

The Java program shown below is a slight modification of the program `Overload.java` of Section 9.12. In the program of Section 9.12, the first call to `foo` in `main` invokes the second `foo` and, if we uncomment the commented out statement, the second call to `foo` results in a compile-time error. Now examine the code in the program shown below and answer the following question:

Will any of the calls to `foo` in the lines (A), (B), and (C) elicit a compile-time error? Also, of the three overload definitions of `foo` provided, which `foo` will be selected for each call to the function by Java's overload resolution algorithm?

```
//Overload2.java

class Employee { String name; }
class Manager extends Employee { int level; }

class Test {
    // first foo:
    static void foo( Employee e1,
                     Employee e2, double salary ) {
        System.out.println( "first foo" );
    }

    // second foo:
    static void foo( Employee e,
                           Manager m, int salary ) {
        System.out.println( "second foo" );
    }

    // third foo:
    static void foo( Manager m, Employee e, int salary ) {
        System.out.println( "third foo" );
    }
```

```
    public static void main( String[] args )
    {
        Employee emp = new Employee();
        Manager man = new Manager();

        foo( emp, man, 100.25 );                                    //(A)
        foo( emp, emp, 100 );                                       //(B)
        foo( man, man, 100.25 );                                    //(C)
    }
}
```

10

Handling Exceptions

PROBLEM 1:

Many methods defi ned for the Java I/O stream classes throw exceptions of type `IOException`. Therefore, in a large Java program it is sometimes diffi cult to pinpoint the statement that might be responsible for generating this exception. The following homework exercise is about using a user-defi ned exception type that can be more helpful for localizing a problem.

Defi ne a new exception by

```
class FileIOException extends Exception {}
```

Now write a Java class, `FileIO`, with the static methods listed below. Each method should rethrow any exceptions of the form `IOException` as a `FileIOException`.

```
public static String readOneString( String filename )
                throws File IOException

public static int readOneInt( String filename )
                throws FileIOException

public static double readOneDouble( String filename )
                throws FileIOException

public static String[] readAllStrings( String filename )
                throws FileIOException
```

```
public static int[] readAllInts( String filename )
                throws FileIOException

public static double[]
            readAllDoubles( String filename )
                throws FileIOException

public static void
            writeOneString( String data, String filename )
                throws FileIOException

public static void
            writeOneInt( int data, String filename )
                throws FileIOException

public static void
            writeOneDouble( double data, String filename )
                throws FileIOException

public static void
            writeAllStrings( String[] strArray,
                             String filename )
                throws FileIOException

public static void
            writeAllInts( int[] intArray,
                               String filename )
                throws FileIOException

public static void writeAllDoubles(
             doubles[] doubleArray, String filename )
                throws FileIOException;
```

Note that the first three, of the form `readOne...()`, are for reading just one data item from a file. The next three methods, of the form `readAll....()` read all the data in a file and return it in the form of an array of the appropriate type. The next three methods, of the form `writeOne....()` are for writing just one item into a file. And the last three methods, of the form `writeAll....()`, for writing arrays of different types into output files. Each "read" method listed above should open the source file for reading data and then close it after the data are acquired. Similarly, each "write" method should open the destination file for writing and then close it after the writing is done.

Solution:

```
//Filename: FileIO.java
import java.io.*;
import java.util.*;
```

```
class FileIOException extends Exception {}

public class FileIO {

  public static String readOneString( String filename )
                    throws FileIOException
  {
    try {
      String word = "";
      Reader input = new FileReader( filename );
      while ( true ) {
        int ch = input.read();
        if ( ch == -1 || ch == '\n' || ch == ' ' ) break;
        word += (char) ch;
      }
      if ( input != null ) input.close();
      return word;
    } catch( IOException e ) { throw new FileIOException(); }
  }


  public static void writeOneString( String data, String filename )
                    throws FileIOException
  {
    try {
      Writer output = new PrintWriter( new FileOutputStream( filename ) );
      output.write( data );
      if ( output != null ) output.close();
    } catch( IOException e ) { throw new FileIOException(); }
  }


  public static int readOneInt( String filename )
                    throws FileIOException
  {
    String str = readOneString( filename );
    int data = Integer.parseInt( str );
    return data;
  }


  public static void writeOneInt( int data, String filename )
                    throws FileIOException
  {
    try {
      PrintWriter output = new PrintWriter( new FileOutputStream( filename ) );
      output.print( data );
      if ( output != null ) output.close();
```

```
  } catch( IOException e ) { throw new FileIOException(); }
}


public static double readOneDouble( String filename )
                  throws FileIOException
{
  String str = readOneString( filename );
  double data = Double.valueOf( str ).doubleValue();
  return data;
}


public static void writeOneDouble( double data, String filename )
                  throws FileIOException
{
  try {
    PrintWriter output = new PrintWriter( new FileOutputStream( filename ) );
    output.print( data );
    if ( output != null ) output.close();
  } catch( IOException e ) { throw new FileIOException(); }
}


static public String[] readAllStrings( String filename )
                  throws FileIOException
{
  try {
    Vector wordVector = new Vector();
    String[] wordArray = null;
    int ch = 0;
    String word = "";

    Reader input = new FileReader( filename );

    while ( true ) {
      if ( -1 == ( ch = input.read() ) ) break;
      while ( true ) {
        if ( ch == -1 || ch == '\n' || ch == ' ' ) break;
        word += (char) ch;
        ch = input.read();
      }
      if ( word.length() != 0 )
        wordVector.addElement( new String( word ) );
      if ( ch == -1 ) break;
      word = "";
    }

    wordArray = new String[ wordVector.size() ];
```

```
    wordVector.copyInto( wordArray );
    if ( input != null ) input.close();
    return wordArray;
  } catch( IOException e ) { throw new FileIOException(); }
}


public static void writeAllStrings( String[] strArray, String filename )
                  throws FileIOException
{
  try {
    PrintWriter output = new PrintWriter( new FileOutputStream( filename ) );
    for ( int i=0; i<strArray.length; i++ )
      output.print( strArray[i] + " " );
    if ( output != null ) output.close();
  } catch( IOException e ) { throw new FileIOException(); }
}


static public int[] readAllInts( String filename )
                   throws FileIOException
{
  try {
    Vector IntegerVector = new Vector();
    int ch = 0;
    int x = 0;
    boolean intStarted = false;
    boolean negativeNum = false;

    Reader input = new FileReader( filename );

    while ( true ) {
      if ( -1 == ( ch = input.read() ) ) break;
      while ( true ) {
        if ( ch == -1 || ch == '\n' || ch == ' ' ) break;
        if ( ch == '-' ) {
          negativeNum = true;
          intStarted = true;
          ch = input.read();
          continue;
        }
        x = 10 * x + ch - 48;
        intStarted = true;
        ch = input.read();
      }
      if ( intStarted == true )
        if ( negativeNum == true )
           IntegerVector.addElement( new Integer( -1 * x) );
        else IntegerVector.addElement( new Integer( x) );
```

```
        intStarted = false;
        negativeNum = false;
        if ( ch == -1 ) break;
        x = 0;
      }

      int[] intArray = new int[ IntegerVector.size() ];
      for ( int i = 0; i < IntegerVector.size(); i++ )
        intArray[i] = ( (Integer) IntegerVector.elementAt(i)).intValue();

      if ( input != null ) input.close();
      return intArray;
    } catch( IOException e ) { throw new FileIOException(); }
  }


  public static void writeAllInts( int[] intArray, String filename )
                    throws FileIOException
  {
    try {
      PrintWriter output = new PrintWriter( new FileOutputStream( filename ) );
      for ( int i=0; i<intArray.length; i++ )
        output.print( intArray[i] + " " );
      if ( output != null ) output.close();
    } catch( IOException e ) { throw new FileIOException(); }
  }


  public static double[] readAllDoubles( String filename )
                    throws FileIOException
  {
    String[] strArray = readAllStrings( filename );
    double[] doubleArray = new double[ strArray.length ];
      for ( int i = 0; i < strArray.length; i++ )
        doubleArray[i] = Double.valueOf( strArray[i] ).doubleValue();
    return doubleArray;
  }


 public static void writeAllDoubles( double[] doubleArray, String filename )
                     throws FileIOException
  {
    try {
      PrintWriter output = new PrintWriter( new FileOutputStream( filename ) );
      for ( int i=0; i<doubleArray.length; i++ )
        output.print( doubleArray[i] + " " );
      if ( output != null ) output.close();
    } catch( IOException e ) { throw new FileIOException(); }
  }
```

```
}


class Test {

  public static void main( String[] args )
  {
      /*
    //  Code for testing readOneString():
    String str = "";
    try {
      str = FileIO.readOneString( "infile" );
    } catch( FileIOException m ) {}
    System.out.println( "The word is:  " + str );
      */

    /*
    // Code for testing writeOneString():

    String data = "Hello, Zaphod";
    try {
      FileIO.writeOneString( data, "outfile" );
    } catch( FileIOException m ) {
      System.out.println( " Caught FileIOException " );
      System.exit( -1 );
    }
    */

      /*
    // Code for testing readOneInt():

    int x = 0;
    try {
      x = FileIO.readOneInt( "infile" );
    } catch( FileIOException m ) {}

    System.out.println( "The int is:  " + x );
      */


    /*
    // Code for testing writeOneInt():

    int data = 123454321;
    try {
      FileIO.writeOneInt( data, "outfile" );
    } catch( FileIOException m ) {
      System.out.println( " Caught FileIOException " );
      System.exit( -1 );
```

```
}
*/

/*
// Code for testing writeAllDoubles():

try {
  double[] doubleArr = FileIO.readAllDoubles( "infile" );
  FileIO.writeAllDoubles( doubleArr, "outfile" );
} catch( FileIOException m ) {
  System.out.println( " Caught FileIOException " );
  System.exit( -1 );
}
*/

/*
// Code for testing writeAllInts():

try {
  int[] intArr = FileIO.readAllInts( "infile" );
  FileIO.writeAllInts( intArr, "outfile" );
} catch( FileIOException m ) {
  System.out.println( " Caught FileIOException " );
  System.exit( -1 );
}
*/

/*
// Code for testing writeAllStrings():

try {
  String[] strArr = FileIO.readAllStrings( "infile" );
  FileIO.writeAllStrings( strArr, "outfile" );
} catch( FileIOException m ) {
  System.out.println( " Caught FileIOException " );
  System.exit( -1 );
}
*/

/*
// Code for testing writeOneDouble():

double data = 12345.54321;
try {
  FileIO.writeOneDouble( data, "outfile" );
} catch( FileIOException m ) {
  System.out.println( " Caught FileIOException " );
  System.exit( -1 );
}
```

```
*/

/*
// Code for testing readAllStrings():

String[] wordList = null;
try {
  wordList = FileIO.readAllStrings( "infile" );
} catch( FileIOException m ) {
  System.out.println( " Caught FileIOException " );
  System.exit( -1 );
}
String temp = "";
for ( int i=0; i<wordList.length; i++ )
  temp += wordList[i] + " ";
System.out.println( "The words are:  " + temp );
*/


// Code for testing readAllInts():

int[] intArray = null;
try {
  intArray = FileIO.readAllInts( "infile" );
} catch( FileIOException m ) {
  System.out.println( " Caught FileIOException " );
  System.exit( -1 );
}
for ( int i=0; i<intArray.length; i++ )
  System.out.print( intArray[i] + "  " );
System.out.println();


/*
// Code for testing readAllDoubles():

double[] doubleArray = null;
try {
  doubleArray = FileIO.readAllDoubles( "infile" );
} catch( FileIOException m ) {
  System.out.println( " Caught FileIOException " );
  System.exit( -1 );
}
for ( int i=0; i<doubleArray.length; i++ )
  System.out.print( doubleArray[i] + "  " );
System.out.println();
*/

/*
```

```
    // Code for testing readOneDouble():

    double d = 0;
    try {
      d = FileIO.readOneDouble( "infile" );
    } catch( FileIOException m ) {}

    System.out.println( "The double is:  " + d );
    */
  }
}
```

PROBLEM 2:

The goal of this homework is to use C++ and Java exceptions to terminate search when you find a given word in a dictionary of words stored in the form of a binary tree. To keep matters simple, assume that all the words are stored in the leaf nodes of the tree and that the word stored at each internal node is the 'greater' of the words at the two child nodes according to ASCII-based lexicographic ordering of the words. You'd obviously need to first write C++ and Java programs that can construct binary-tree dictionaries from the words in a text file. And then write search programs in each language that search for a query word by descending down the dictionary tree. When there is a match between a query word and the word stored at a node, throw an exception to terminate the search.

Suggestions:

Although exceptions are used more frequently for handling situations that might otherwise result in runtime problems, they can also be used for intelligently terminating the flow of control in algorithms. As an example, let's say that we want to write a C++ program that would search for a word in a dictionary that is represented in the form of a binary tree for the purpose of search efficiency. With a binary tree, the search time for locating a word can be reduced from the worst-case linear to logarithmic.

Don't forget that there are ready-made highly-optimized container classes in the C++ Standard Library and in the Java Collections Framework that allow us to store objects in binary trees for efficient retrieval and other operations. See Chapter 5 for further details on the container classes. The discussion here is meant only to illustrate the utility of the exception throwing mechanism for terminating search.

For a simple program to demonstrate the key ideas, each node of this tree could be represented by an object of the following type

```
class Tnode {
public:
  string word;
  Tnode* left;
  Tnode* right;
  void print() { cout << word << " "; }
```

```
};
```

The tree would have to be constructed from the words that may initially be stored in an array. Let's say that we choose to use the following array for representing all the words

```
const int MAX = 1000;      // max of 1000 words allowed
string* words[MAX];
```

where the variable `MAX` specifies the maximum number of words that the program will be able to handle.

Given a list of words in the pointer array `words[]`, we wish to write a function of prototype

```
Tnode* makeTree(string* words[], int numNodes);
```

that would return a pointer to the root node of the binary tree. The second argument is the number of words in the array that is the first argument. In order to write this function, consider first the boundary case that occurs when `numWords` equals 1, then the tree will consist of only one word. We can instantiate this node in the following manner

```
if (1 == numNodes) {
  Tnode* leafNode = new Tnode();
  if (words[0] != 0) leafNode->word = *words[0];
  leafNode->left = static_cast<Tnode*>(0);
  leafNode->right = static_cast<Tnode*>(0);
  return leafNode;
}
```

This allows us to create what's called a leaf node of the tree. This node has no child nodes. That's the reason for why its left and right child nodes are set to null pointers. The `word` member of this node points to the word that this node stores at the bottom of the tree.

Now consider the case when we have two or more words. We now need to create an internal node by

```
Tnode* internalNode  =  new Tnode();

internalNode->left  = makeTree( &words[0],  numNodes/2 );

internalNode->right = makeTree( &words[numNodes/2],
                                              numNodes/2 );
```

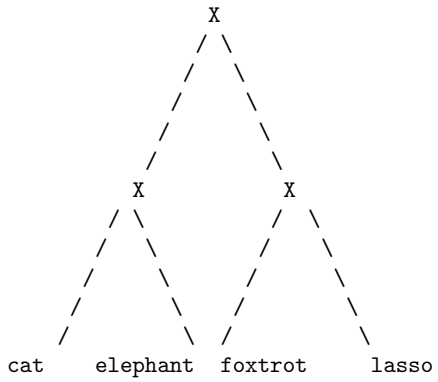
These two cases represent all there is to the `makeTree()` algorithm, except for one thing: we have yet to instantiate the `word` member at the interior nodes. As should be clear from the explanation above, all the words in the list `words[]` are actually stored in the leaf nodes.

To instantiate the `word` member at an interior node, we need to keep in mind how we may search for a word after the tree is fully built. Starting at the root, we could descend either into the left-subtree or the right subtree at the root. How to make that decision? The decision can be made on the basis of the word stored at the interior node. Suppose the original list of words in the list `words[]` is alphabetized. What that means is that all the leaf nodes created by the `makeTree` algorithm will exhibit a left-to-right order that will correspond to the order

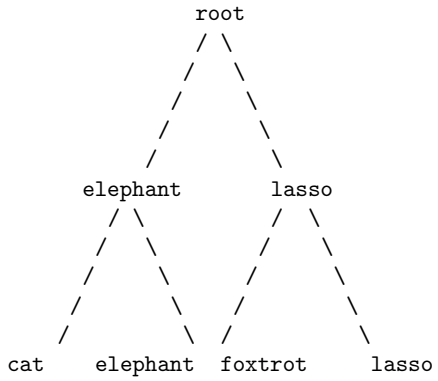
produced by the alphabetization of the list. To make this point clearer, consider a list of the following alphabetized list of four words

```
cat elephant foxtrot lasso
```

The makeTree algorithm will create the following tree for these four words:



Now let's say we need to figure out whether or not the word `dog` exists in this structure. To expedite this search, we could first, before using the tree for searches, examine the tree from bottom up and place at each interior node the word that is contained in the right descendent of that node. This would give us the following structure



Now suppose we want to figure out whether the word `dog` exists in this structure, we can compare `dog` to the words in the two descendents of the root. Since the highest word in the left descendent is `elephant` and since `dog` is ranked lower than `elephant`, we go down the left branch. At the `elephant` interior node, we now compare `dog` to the two descendents `cat` and `elephant`. These being leaf nodes, we now make direct comparisons, to reach the conclusion that `dog` does not exist in this structure.

That leaves us with just two tasks: to figure out the precise mechanics of how to place words at the interior nodes and then how exactly to write a recursive routine for conducting the search.

Placing words at the interior nodes can be accomplished by a program of the prototype

```
void fillInternals( Tnode* node );
```

that is invoked with a call to the root

```
fillInternals( rootNode );
```

where `rootNode` is initially the root node, and then recursively gets invoked on the left-descendent and the right-descendent:

```
fillInternals( node->left );
fillInternals( node->right );
```

followed by

```
if ( node->right->word.compare(node->left->word) >= 0 )
  node->word = node->right->word;      // (A)
else
  node->word = node->left->word;       // (B)
```

It is this step that actually places the word in an interior node. Ordinarily, when the number of words is a power of 2 and the word list is alphabetized, only the statement at (A) would be invoked, meaning that the interior node would acquire the word that is at its right descendent. The test in the `if` clause and the assignment at (B) take care of the situation when the number of words is not a power of 2. In this case, some of the nodes of the tree on its right will not be populated. For example, if the number of words in the text file is only 7, the last interior node will have only a left descendent, the right descendent will point to a null pointer. In such a situation, the interior node will have to be assigned the word at the left descendent. (When the number of words is not a power of 2, the tree generated by the `makeTree` program will be unbalanced, in the sense that that depth of the tree will not be same everywhere. It is not too difficult to write slightly more sophisticated computer programs that would generate a balanced tree regardless of the number of words.

That brings us to a discussion of how to set up the search for a query word and how to employ exception handling in this search. The search can be done by setting up two functions, one for fetching the query word from the user and the other for comparing the query word with the words in the tree. The routine for fetching the word could be set up with the following prototype

```
string queryWord;
cout << "Enter your query word: ";
cin >> queryWord;

try {
  search(queryWord, rootNode);
}
catch (int ret) {
  if (ret) {
    cout << endl << "Yes, the word is in the dictionary"
         << endl;
```

```
        return;
      }
    }
    cout << endl << "No, the word is not in the dictionary"
         << endl;
```

Notice how `search()` is called within a `try–catch` block. The function `search()` itself could be set up like

```
    void search(string queryWord, Tnode* node)
    {
      if ( queryWord.compare(node->word) == 0) throw(1);
      if ( leafNode(node)) return;
      if ( node->left->word.compare( queryWord ) >= 0 ) {
        search(queryWord, node->left);
        return;
      }
      if ( node->left->word.compare( queryWord ) < 0 )
        search(queryWord, node->right);
    }
```

The predicate `leafNode( node )` is true if both the left and the right descendents of the node are null pointers. Notice how `search()` throws 1 when there is a match between a `queryWord` and the word at the node. By throwing and catching we avoid having to percolate our way back up through the function calling stack established during the recursive calls.

C++ Solution:

```
// Dictionary.cc
/*
 *  This program first reads in a text file listed in the command line
 *  and then creates a simple binary dictionary from the words in
 *  the file.  Any punctuation marks sticking to the words are deleted
 *  before the dictionary is created.  The user is then given
 *  various options for playing with this dictionary.  For example,
 *  the entire dictionary can be displayed as a parenthesized
 *  binary tree.  And, of course, the user is allowed to search for
 *  a word in the dictionary.
 *
 *  Note that the binary tree used here is not a height-balanced tree
 *  like a 2-3 tree or an AVL tree.  The emphasis of this educational
 *  tool is not on creating optimum data structures, but on the
 *  use of the try-catch mechanism for terminating recursive search.
 *
 *  Compile this program by
 *
 *         CC dictionary.cc
 *
```

```
 *  and invoke it by
 *
 *         a.out filename
*/


#include <fstream>
#include <string>
#include <cstdlib>                    // for qsort
#include <iomanip>                    // for setw in print_menu
using namespace std;

class Tnode {
public:
  string word;
  Tnode* left;
  Tnode* right;
  void print() { cout << word << " "; }
};


const int MAX = 1000;                  // max of 1000 words allowed
string* words[MAX];

Tnode* rootNode;
int numWords;

Tnode* makeTree(string**, int);
void displayTree(Tnode*);
void fillInternals(Tnode*);
int nearest_pwr_2(int);
void display(void);
void enterSearchMode(void);
void printMenu(void);
int leafNode(Tnode*);
void search(string, Tnode*);
int compareStrings( const void* str1, const void* str2 );
string cleanup( string& );


/*********************************************************************
 * main:  First constructs a binary dictionary from the words in a
 *        given file.  Then prompts the user to enter an operation
 *        code, and, then, depending on the code entered, calls a
 *        function to perform the requested action.  Repeats until
 *        the user enters the command 'q'.  Prints an error message
 *        if the user enters an illegal code.
 *********************************************************************/
```

```
int main(int argc, char** argv)
{
  if (2 != argc) {
    cerr << "Usage: a.out filename" << endl;
    exit(0);
  }
  ifstream cin_from_file( *++argv );
  if (!cin_from_file) {
    cerr << "cannot open input file" << endl;
    exit(0);
  }

  int i = 0;
  string buffer;
  while ( cin_from_file >> buffer ) words[i++] = new string( cleanup(buffer) );
  numWords = i;
  qsort(words, numWords, sizeof(string*), compareStrings);
  int m = nearest_pwr_2(numWords);
  for (int j=numWords; j<m; j++) words[j] = 0;
  rootNode = makeTree(words, m);
  fillInternals(rootNode);

  char code;
  for(;;) {
    cout << "Enter operation code (enter '?' for menu): ";
    cin >> code;
    cout << endl;
    while (cin.get() != '\n')                  // skips to the end of line
      ;
    switch (code) {
      case 'd': display();                     //  display the tree
                break;
      case 's': enterSearchMode();
                break;
      case '?': printMenu();
                break;
      case 'q': return 0;

      default : cout << "Illegal code" << '\n';
    }
    cout << '\n';
  }
}


/*********************************************************
 * The comparison function for qsort
 *********************************************************/
int compareStrings( const void* str1, const void* str2 ) {
```

```
    return (**(string**) str1).compare(**(string**) str2 );
}

/*************************************************************
 *  This function scans a given string for any designated
 *  chars that are in 'filter'.  All such chars are removed
 *  and the cleaned up string sent back to main.
 *************************************************************/
string cleanup( string& word ) {
  char cleanChars[ word.size() ];
  string filter = string(",;.*!\"()[]\\_-?");
  int k = 0;

  for (int i=0; i < word.size(); i++) {
    bool bad_char_flag = false;
    for (int j=0; j < filter.size(); j++)
      if ( word[i] == filter[j] ) { bad_char_flag = true; break; }
    if ( bad_char_flag == false ) cleanChars[ k++ ] = word[i];
  }

  char goodChars[ k + 1 ];
  for (i=0; i<k; i++) goodChars[i] = cleanChars[i];
  goodChars[k] = '\0';
  string cleanWord = string( goodChars );
  return cleanWord;
}

/*************************************************************
 * ask for query word and invoke search function
 *************************************************************/
void enterSearchMode(void) {
  string queryWord;
  cout << "Enter your query word: ";
  cin >> queryWord;

  try {
    search(queryWord, rootNode);
  }
  catch (int ret) {
    if (ret) {
      cout << endl << "Yes, the word is in the dictionary" << endl;
      return;
    }
  }
  cout << endl << "No, the word is not in the dictionary" << endl;
  return;
}

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

```
In case a word match is found during a recursive depth-first
descent of the tree, it is this function that throws an exception
to the calling function.
*****************************************************************/
void search(string queryWord, Tnode* node) {
  if ( queryWord.compare(node->word) == 0) throw(1);
  if ( leafNode(node)) return;
  if ( node->left->word.compare( queryWord ) >= 0 ) {
    search(queryWord, node->left);
    return;
  }
  if ( node->left->word.compare( queryWord ) < 0 )
    search(queryWord, node->right);
}

/*****************************************************************
 * This function displays the binary dictionary created in the
 * form of a parenthesized tree.  It also shows separately the
 * leaves of this tree.
 *****************************************************************/
void display(void) {
  cout << '\n';
  cout <<  "binary dictionary displayed as a tree:" << '\n';
  cout << '\n';
  displayTree(rootNode);

  cout << '\n';
  cout << '\n';
  cout << "The words you entered in a sorted list: " << '\n';
  cout << '\n';
  int j = 0;
  while ( j < numWords ) cout << *words[j++] << " ";
  cout << endl;
}


/*****************************************************************
 * This is the workhorse of the binary dictionary data structure that
 * I create for storing the sorted list of words.   Its recursion
 * makes it self explanatory.
 *****************************************************************/
Tnode* makeTree(string* words[], int numNodes) {
  if (1 == numNodes) {
    Tnode* leafNode = new Tnode;
    if (words[0] != 0) leafNode->word = *words[0];
    leafNode->left = static_cast<Tnode*>(0);
    leafNode->right = static_cast<Tnode*>(0);
    return leafNode;
  }
```

```
  else {
    Tnode* internalNode = new Tnode;
    internalNode->left = makeTree(&words[0], numNodes / 2);
    internalNode->right = makeTree(&words[numNodes / 2], numNodes / 2);
    return internalNode;
  }
}


/*************************************************************************
 * After a tree is built by the makeTree program, its leaves will be
 * populated by all the words supplied by the user.  The program
 * fill_internals then traverses this tree and deposits at each internal
 * node the word in its right descendent.  If the right descendent does
 * not exist, then the internal node takes the value of its left
 * descendent.  These words at the internal nodes serve as decision
 * thresholds when we search for a keyword in the dictionary.
 *************************************************************************/
void fillInternals(Tnode* node) {
  if (leafNode(node))
    return;

  else {
    fillInternals(node->left);
    fillInternals(node->right);
    if ( node->right->word.compare(node->left->word) >= 0 )
      node->word = node->right->word;
    else
      node->word = node->left->word;
  }
}

/*****************************************************************
 * Displays the tree by printing out the contents of the member 'word'.
 * Starting at the root node, it descends the tree until it strikes the
 * leaf nodes and then it invokes the print function defined for the
 * Tnode struct.
 *****************************************************************/
void displayTree(Tnode* node) {
  if (leafNode(node)) {
    cout << '(';
    (*node).print();
    cout << ')';
    return;
  }
  else {
    cout << '(';
    (*node).print();
```

```
    displayTree(node->left);
    displayTree(node->right);
    cout << ')';
  }
}

/*********************************************************************
 * This function returns the nearest power of 2 larger than or equal to
 * the integer supplied as the argument.
 *********************************************************************/
inline int nearest_pwr_2(int i) {
  int power = 2;
  while (1) {
    if (power >= i)
      return power;
    power *= 2;
  }
}

/*********************************************************************
 * Outputs the menu options available at the top level.
 *********************************************************************/
void printMenu(void) {
  cout << setw(10) << 'd' << setw(30) << ":  for displaying as a tree the dictionary" << end
  cout << setw(10) << 's' << setw(30) << ":  for entering keyword search" << endl;
  cout << setw(10) << 'q' << setw(30) << ":  to quit this program" << endl;
  cout << setw(10) << '?' << setw(30) << ":  to print this menu" << endl;
  cout << '\n';
}

/*********************************************************************
 * leaf_node predicate
 *********************************************************************/
int leafNode(Tnode* node) {
  if ((node->left == static_cast<Tnode*>(0)) &&
      (node->right == static_cast<Tnode*>(0)))
    return 1;
  else
    return 0;
}
```

Java Solution:

```
// Dictionary.java

/*  To have the most fun with this Dictionary class, copy it into your
 *  own directory, compile it, and then invoke it on a text file by
```

```
*
*      java Dictionary text_file_name
*
*  It will prompt you for whether you want to see all the words in
*  the dictionary, or whether you wish to see the dictionary in a
*  parenthesized form, or for whether you wish to search for a word
*  in the dictionary.
*
*  If you copy this class into your own directory, be sure to
*  comment out the 'package morejava;' line at the beginning of
*  the source code.
*
*  You can also use this class in other java programs by using the
*  following constructor and methods:
*
*      Dictionary d1 = new Dictionary( text_file_name );
*
*      d1.showAllWords();      for printing out all words
*
*      d1.showDictionary();    for printing out a parenthesized form
*                              of the binary dictinary
*      d1.search( queryWord );  for searching for a query word in
*                                in the dictionary
*
*      The search function listed above can only be called inside a
*      a try-catch block.  If an exception of type Err() is caught
*      that means that queryWord was found in the dictionary.
*
*
*  For fetching words from a text file, the Dictionary class depends
*  on the FileReaderData class.  As you may recall, the
*  FileReaderData class cleans up any punctuations marks that might
*  be sticking to the words.
*
*  From an educational standpoint, a novel feature of the
*  class is that uses an inner class called Tree.  All the recursive
*  aspects of dealing with trees are inside this inner class.
*
*  Note that the binary tree used here is not a height-balanced
*  2-3 tree or the AVL tree.  The emphasis of this educational
*  tool is not on creating optimum data structures, but on the
*  use of the try-catch mechanism for terminating recursive search.
*
*  Compile this program by
*
*        javac Dictionary.java
*
*  and invoke it by
*
```

```
 *         java Dictionary filename
*/


public class Dictionary {

  String    textFile;
  String[]  allWords;
  Tree      rootOfTree;

  // constructor:
  public Dictionary( String filename ) throws Exception {
    textFile = new String( filename );
    allWords = FileIO.readAllStrings( textFile );
    System.out.println( "The number of words read is:  "  + allWords.length );
    sort();
    rootOfTree = new Tree( allWords );
    rootOfTree.fillInternals();
  }

  // print out the content of allWords:
  public void showAllWords() {
    for (int i=0; i<allWords.length; i++)
      System.out.print( allWords[i] + " " );
    System.out.println( "\n\n" );
  }

  // display the dictionary in a parenthesized form:
  public void showDictionary() {
    rootOfTree.displayTree();
    System.out.println( "\n\n" );
  }

  // search function:
  public void search( String queryWord ) throws Err {
    try {
      rootOfTree.treeSearch( queryWord );
    } catch ( Err e ) { throw new Err(); }
  }

  // sort the array allWords:
  void sort() {
    for (int i=0; i < allWords.length; i++ ) {
      for (int j=0; j < allWords.length; j++ ) {
        if ( allWords[i].compareTo( allWords[j] ) <= 0 ) {
          String temp = allWords[i];
          allWords[i] = allWords[j];
          allWords[j] = temp;
        }
```

```
    }
  }
}

// Err is an inner class of the Dictionary class:
class Err extends Exception {};


// Tree is an inner class of the Dictionary class:
class Tree {
  String word;
  Tree leftBranch;
  Tree rightBranch;

  void print() { System.out.print( word + " " ); }

  // constructor for the inner class Tree:

  Tree( String[] strings ) {
    if ( strings.length == 1 ) {
      if (strings[0] != null) word = new String( strings[0] );
      return;
    }
    int numNodes = nearestPowerOf2( strings.length );
    String[] firstHalf = new String[ numNodes / 2 ];
    String[] secondHalf = new String[ numNodes / 2 ];
    for (int i=0; i<numNodes/2; i++)
      if (strings[i] != null)  firstHalf[i] = new String( strings[i] );
    for (int i=numNodes/2; i<strings.length; i++)
      if (strings[i] != null) secondHalf[i-numNodes/2] = new String( strings[i] );
    leftBranch = new Tree( firstHalf );
    rightBranch = new Tree( secondHalf );
  }

  // Fills internal nodes of Tree:
  void fillInternals() {
    if (leftBranch == null && rightBranch == null) return;
    else {
      leftBranch.fillInternals();
      rightBranch.fillInternals();
      if (leftBranch.word == null && rightBranch.word == null) return;
      if (leftBranch.word != null && rightBranch.word == null) {
        word = new String( leftBranch.word );
        return;
      }
      if ( rightBranch.word.compareTo( this.leftBranch.word ) >= 0 )
        word = new String( rightBranch.word );
      else
        word = new String( leftBranch.word );
```

```
    }
  }

  // search function:
  void treeSearch( String queryWord ) throws Err {
    if ( word.compareTo( queryWord ) == 0 ) throw new Err();
    if ( leftBranch == null && rightBranch == null ) return;
    if ( leftBranch.word == null && rightBranch.word == null ) return;
    if ( leftBranch.word != null && leftBranch.word.compareTo( queryWord ) >= 0 ) {
      leftBranch.treeSearch( queryWord );
      return;
    }
    if ( rightBranch.word != null && leftBranch.word.compareTo( queryWord ) < 0 ) {
    rightBranch.treeSearch( queryWord );
    }
  }

  // used by the Tree constructor:
  int nearestPowerOf2( int j ) {
    int power = 2;
    while (true) {
      if (power >= j) return power;
    power *= 2;
    }
  }

  // displays the Tree in a parenthesized form:
  void displayTree() {
    if (leftBranch == null && rightBranch == null) {
      System.out.print( "(" );
      this.print();
      System.out.print( ")" );
      return;
    }
    else {
      System.out.print( "(" );
      this.print();
      leftBranch.displayTree();
      rightBranch.displayTree();
      System.out.print( ")" );
    }
  }
} // end of Tree inner class


// main:
public static void main( String[] args ) throws Exception
{
  if ( args.length != 1 ) {
```

```
      System.err.println( "Usage: java Dictionary text_file_name" );
      System.exit(0);
    }

    Dictionary d1 = new Dictionary( args[0] );

    while (true) {
        System.out.print( "\nEnter operation code  (Enter '?' for menu): " );
        String reply = TerminalIO.readOneString();
      if (reply.compareTo( "?" ) == 0) {
        System.out.println( "\n" );
        System.out.println( "p:  display all words in the text file" );
        System.out.println( "d:  display the dictionary in a parenthesized form" );
        System.out.println( "s:  enter search mode" );
        System.out.println( "q:  quit this session" );
        System.out.println( "\n" );
        continue;
      }
      if (reply.compareTo( "p" ) == 0) { d1.showAllWords(); continue; }
      if (reply.compareTo( "d" ) == 0) { d1.showDictionary(); continue; }
      if (reply.compareTo( "s" ) == 0) {
          System.out.print( "\nEnter query word: " );
          String queryWord = TerminalIO.readOneString();
        try {
          d1.search( queryWord );
        } catch( Err e ) {
          System.out.println( "\nYes, the word is in the dictionary.\n" );
          continue;
        }
        System.out.println( "\nNo, the word is not in the dictionary.\n" );
        continue;
      }
      if (reply.compareTo( "q" ) == 0) { System.exit(0);}
      System.out.println( "\nYou have entered illegal code. Try again.\n" );
    }
  }
}
```

PROBLEM 3:

Write a Java class `WordFinder` for locating a a word in a text fi le; the designated word could be a substring of a longer word. Your class should possess one method with the following prototype:

```
public static boolean searchFor( String file, String word )
```

The method should use the `readAllStrings` method of the `FileIO` class of Problem 1 to read all the words in the text file. Concatenate the words thus read into one long string and use the `substr` method of the `String` class to locate the desired word. Make sure your program has an appropriate handler for the `FileIOException` thrown by the methods of the `FileIO` class.

Solution:

```
import java.io.*;

abstract class WordFinder {

  public static boolean searchFor( String file, String word ) {
    FileReader input = null;
    String allChars = "";
    try {
      input = new FileReader( file );
      int ch = 0;
      while ( -1 != ( ch = input.read() ) )
        allChars += (char) ch;
      char startChar = word.charAt(0);
      char endChar = word.charAt( word.length() - 1 );
      int startPos = 0;

      while ( -1 != ( startPos = allChars.indexOf( startChar, startPos ) ) ) {
        int endPos = allChars.indexOf( endChar, startPos );
        if ( startPos > endPos ) continue;
        else if ( startPos == -1 ) break;
        else if ( endPos == -1 ) break;
        else {
          String substr = allChars.substring( startPos, endPos + 1 ).intern();
          if ( word == substr ) return true;
        }
        startPos++;
      }
      return false;

    } catch( FileNotFoundException e ) {
        System.out.println( "file not found" );
        e.printStackTrace();
    } catch( IOException e ) {
        System.out.println("problems with either read() or close()");
        e.printStackTrace();
    } finally {
        if ( input != null )
          try {
            input.close();
```

```
          } catch( IOException e ) {
              System.out.println( "problems with close in finally" );
              e.printStackTrace();
          }
    }
    return false;
  }
}


class Test {

  public static void main( String[] args )
  {
    String word = "jello";
    String filename = "infile";
    boolean foundIt = WordFinder.searchFor( filename, word );
    if ( foundIt )
      System.out.println( "The word " + word + " exists in file " + filename );
    else
      System.out.println( "The word " + word + " does not exist in file " + filename );
  }
}
```

PROBLEM 4:

Write a C++ function that would return an input stream to a file. But if something were to go wrong, such as if the file was missing, the function should throw the following user-specified exception:

```
    class IfstreamException {};
```

Solution:

```
#include <fstream>
#include <string>
using namespace std;

class IfstreamException {};

ifstream* ifstreamWithException( string filename )
    throw(IfstreamException)
{
    ifstream* cinFromFilePtr = new ifstream( filename.c_str() );
    if ( !*cinFromFilePtr ) throw ifstreamException();
```

```
    return cinFromFilePtr;
}


int main() {
    ifstream* infile;

    try {
        infile = ifstreamWithException( "nonExistantFile" );
    } catch( IfstreamException e) {
        cerr << "input stream error" << endl;
        exit(1);
    }

    string str;
    while ( *infile ) {
        *infile >> str;
        cout << str << " ";
    }
}
```

If a C++ function throws an exception not listed in its exception specification, the function

```
unexpected()
```

defined in the C++ standard library is invoked. The default behavior of `unexpected()` is to call the function `terminate()`. The C++ standard library also provides mechanism for overriding the actions performed by `unexpected()`.

PROBLEM 5:

Why is the following *not* a good example for demonstrating the utility of the `finally` clause in a `try–catch–finally` structure? The program tries to locate the character 'A' in a text file. If it finds the character, the program prints out a message to that effect and terminates, but only after closing the input stream properly in the `finally` clause. If it does not find the character, a message to that effect is printed out and the input stream is closed as before in the `finally` clause.

```
import java.io.*;

class Test {

  public static void main( String[] args )
  {
    FileReader input = null;
    try {
```

```
        input = new FileReader( "infile" );          // (A)
        int ch;
        while ( -1 != ( ch = input.read() ) ) {      // (B)
          if ( ch == 'A' ) {
            System.out.println( "found it" );
            System.exit( 0 );
          }
        }
        System.out.println( "did not find it" );
      } catch( FileNotFoundException e ) {
          System.out.println( "file not located" );
      } catch( IOException e ) {
          System.out.println( "problems with i/o" );
      } finally {
          if ( input != null ) {
            try {
              System.out.println( "about to close the input stream" );
              input.close();
            } catch( IOException e ) {
                System.out.println( "problems with close()" );
            }
          }
      }
    }
  }
```

Solution:

Because we could have closed the input stream outside the try-catch structure and just dropped the finally clause altogether. Since we are catching all the exceptions, the control must eventually shift to outside the try-catch structure and that is where we could test the input stream and close it if we find it non-null.

11

Classes, The Rest of the Story

PROBLEM 1:

Suppose we have three classes X, Y, and Z in a C++ program. Let's say each class depends on the other two. In what order will you declare/define each class in the source code?

Solution:

Partially define X and Y. Completely define Z, with the exception of anything that would have to call X or Y's constructors, data members, member functions, etc. Complete the definition of X, with the exception of anything that would have to call Y's constructors, data members, member functions, etc. Complete the definition of Y. Complete the definitions of the constructors/functions you have prototyped for X and Z.

PROBLEM 2:

Complete the following interleaved classes by adding constructors (at least a no-arg and one other constructor for each class), copy constructors, assignment operators, and destructors "in the appropriate places."

```
class Courses;
```

```
class Student {
public:
  string fn, ln;        // First name, Last name
  int s;                // Semester
  Student* rm;          // Array of roommate students
  int numRm;            // How many roommates
  Courses* c;           // Array of courses that this student is taking
  int numC;             // How many courses

.....

};

class Courses {
public:
  string t;             // Title of the course
  int n;                // Course number
  int maxS;             // Maximum number of students that can take the course
  Student* s;           // Array of students taking the course
  int numS;             // Number of students taking the course

.....

};
```

PROBLEM 3:

The following program will compile and run without any problems. But it has a deadly flaw embedded in it? What is it? Also, what is the output of the program?

```
class X {
public:
    X() {
        cout << "X's no-arg constructor invoked"
             << endl;
    }
 };


class Y  {
    X* x;
public:
    Y() {
        cout << "Y's no-arg constructor invoked" << endl;
        x = new X();
    }
```

```
};


int main()
{
    Y arr[5];
}
```

Solution:

The deadly flaw in the program is that it depends on the system supplied default destructor which is not going to be up to the job. When the array in `main()` goes out of scope, the system will seek to free up the memory occupied by each element of the array by invoking the destructor for the element. Finding none, it will use the default meaning of the destructor, which is to free up the memory occupied by each data member of of each `Y` object. In this example, that means that the memory occupied by the variable `x` will be freed. BUT THE MEMORY OCCUPIED BY THE X OBJECT TO WHICH POINTER x IS POINTING WILL NOT BE FREED. That is the deadly flaw. The output of the program is

```
Y's no-arg constructor invoked
X's no-arg constructor invoked
Y's no-arg constructor invoked
X's no-arg constructor invoked
Y's no-arg constructor invoked
X's no-arg constructor invoked
Y's no-arg constructor invoked
X's no-arg constructor invoked
Y's no-arg constructor invoked
X's no-arg constructor invoked
```

PROBLEM 4:

The following version of program of the previous problem also has a deadly flaw in it. What's it? Will it create a compile time problem or a run time problem?

```
#include <iostream>

 class X {};

 class Y  {
     X* x;
 public:
     Y() { cout << "Y's no-arg constructor invoked" << endl; }
     ~Y() {
```

```
        cout << "Y's destructor invoked" << endl;
        if ( x != 0 ) delete x;
    }
};



int main()
{
    Y arr[5];
}
```

Solution:

The output of the program is:

```
Y's no-arg constructor invoked
Y's no-arg constructor invoked
Y's no-arg constructor invoked
Y's no-arg constructor invoked
Y's no-arg constructor invoked
Y's destructor invoked
Segmentation fault (core dumped)
```

The deadly flaw in this program is that the no-arg constructor does not initialize the pointer data member x. So when the array in main() goes out scope and, consequenly, when the destructor for each element of the array is invoked, the `delete` operator in the destructor tries to free up the memory to which the random bits stored in x are pointing. That's what creates the memory segmentation fault.

PROBLEM 5:

Does the following program fix all problems with our earlier programs? If so, what is the output of the program?

```
#include <iostream>

class X {};

class Y  {
    X* x;
public:
    Y() {
        cout << "Y's no-arg constructor invoked" << endl;
        x = 0;
```

```
    }

    ~Y() {
        cout << "Y's destructor invoked" << endl;
        if ( x != 0 ) delete x;
    }
};


int main()
{
    Y arr[5];
}
```

Solution:

Yes, this is the correct version of the program. The output of the program is:

```
Y's no-arg constructor invoked
Y's no-arg constructor invoked
Y's no-arg constructor invoked
Y's no-arg constructor invoked
Y's no-arg constructor invoked
Y's destructor invoked
Y's destructor invoked
Y's destructor invoked
Y's destructor invoked
Y's destructor invoked
```

PROBLEM 6:

If the intent in the following C++ program is for ptr to be a pointer to an array of X objects, is there anything wrong with the program? (Of course, the compiler has no way of knowing directly if you intended for `ptr` to be a pointer to a single X object or a pointer to an array of X objects. But a human can, by looking at the implementation of the destructor. How is that possible?)

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

class X {};
```

```
class Y  {
    X* ptr;
    int n;
public:
    Y() {
        cout << "no-arg constructor invoked" << endl;
        ptr = 0;
    }
    ~Y() {
        cout << "destructor invoked" << endl;
        if ( ptr != 0 ) delete ptr;
    }
};

int main()
{
    Y y;
}
```

Solution:

The program will compile and run without any problems and will produce the following output:

```
no-arg constructor invoked
destructor invoked
```

But if your intent was for `ptr` to point to an array of `X` objects, then the program has a serious memory leak. To fix this leak, you must replace the statement

```
delete ptr;
```

by the statement

```
delete[] ptr;
```

So whether you use `delete()` or `delete[]` in the destructor determines whether the pointer `ptr` was intended to point to a single object of type `X` or to an array of objects of type `X`. Remember the rule of thumb that in a well-written C++ program, there is a `delete` operator for every `new` operator, and there is a `delete[]` operator for every invocation of `new[]` operator.

PROBLEM 7:

What is the output of the following program? Note that it does not use pointers anywhere.

```
#include <iostream>

class X {
public:
    X() { cout << "X's no-arg constructor invoked" << endl; }
    ~X() { cout << "X's destructor invoked" << endl; }
};

class Y  {
    X x;
public:
    Y() {
        cout << "Y's no-arg constructor invoked" << endl;
    }

    ~Y() {
        cout << "Y's destructor invoked" << endl;
    }
};

int main()
{
    Y arr[5];
}
```

Solution:

The important point here is that X's no-arg constructor is automatically invoked even though we are not initializing the data member x inside Y's constructor.

The invocation of X's constructor is absolutely necessary because otherwise the system would not know how much memory to appropriate for each Y object.

Inside each Y object created resides the X object created by X's constructor. As each Y object goes out of scope, the contained X object also goes out of scope. Therefore, invocation of Y's destructor is followed by the invocation of X's destructor.

Here is the output:

```
X's no-arg constructor invoked
Y's no-arg constructor invoked
X's no-arg constructor invoked
Y's no-arg constructor invoked
X's no-arg constructor invoked
```

```
    Y's no-arg constructor invoked
    X's no-arg constructor invoked
    Y's no-arg constructor invoked
    X's no-arg constructor invoked
    Y's no-arg constructor invoked
    Y's destructor invoked
    X's destructor invoked
    Y's destructor invoked
    X's destructor invoked
    Y's destructor invoked
    X's destructor invoked
    Y's destructor invoked
    X's destructor invoked
    Y's destructor invoked
    X's destructor invoked
```

PROBLEM 8:

In the C++ code shown here, we have gratuitously supplied the class Dog with a copy constructor. If this program is run, how many times will the print statement in the copy constructor be executed?

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

class Dog {
    string name;
    int age;
public:
    Dog( string nam, int a ) : name( nam ), age( a ) {}
    Dog( const Dog& dog ) : name( dog.name ), age( dog.age ) {
        cout << "invoking Dog copy constructor" << endl;
    }
};

class Person {
    string name;
    Dog dog;
public:
    Person( string nam, Dog d ) : name( nam ), dog( d ) {}
};

int main()
{
```

```
    Dog dog( "Fido", 4 );
    Person p1( "Zaphod", dog );
    Person p2 = p1;
}
```

Solution:

Exactly three times:

1. When the dog argument is passed to the d parameter of the Person constructor. This is pass by value. So the copy constructor of Dog will be used to make a local copy of the dog in main.
2. When the statement

   ```
   dog( d )
   ```

   in the member initializer syntax of the `Person` constructor is executed.
3. During the initialization of `p2`

   ```
   Person p2 = p1;
   ```

   This will call the default copy constructor of `Person`, which will carry out a member-wise copy from `p1` to `p2`, using copy constructors for any class type members of `p1`.

PROBLEM 9:

In the code of the previous question, is there any way to reduce the number of times the copy constructor of Dog is invoked?

Solution:

Change the Person constructor to

```
Person( string nam, Dog& d ) : name( nam ), dog( d ) {}
```

where the second parameter is now a reference.

PROBLEM 10:

What would happen if we did not overload the output stream operator for the `Dog` class in the following example code?

```
class Dog {
    string name;
    int age;
public:
    Dog( string nam, int a ) : name( nam ), age( a ) {}
    friend ostream& operator<<( ostream& os, const Dog dog)
    {
        os << "Dog's Name: " << dog.name
           << "  Dog's Age: " << dog.age;
        return os;
    }
};


class Person {
    string name;
    Dog dog;
public:
    Person( string nam, Dog& d ) : name( nam ), dog( d ) {}
    friend ostream& operator<<( ostream& os, const Person per)
    {
        os << "Name: " << per.name << "   Dog:    " << per.dog;
        return os;
    }
};

int main()
{
    Dog dog( "Fido", 4 );
    Person p( "Zaphod", dog );
    cout << p << endl;
}
```

Solution:

The call

```
os << per.dog;
```

in the overload definition for $<<$ for the `Person` class would make no sense.

PROBLEM 11:

In the program of the previous question, is there any way for the output stream operator for `Person` to print out its `Dog`'s information without overloading the $<<$ operator for the `Dog` class? The `Dog`'s data members must remain private.

Solution:

Yes. You can declare the `Person`'s overload definition for $<<$ to be a `friend` in `Dog` class. See example below

```
//NoDogOS.cc
#include <string>

class Person;

class Dog {
    string name;
    int age;
public:
    Dog( string nam, int a ) : name( nam ), age( a ) {}
    friend ostream& operator<<( ostream& os, const Person per);
};


class Person {
    string name;
    Dog dog;
public:
    Person( string nam, Dog& d ) : name( nam ), dog( d ) {}
    friend ostream& operator<<( ostream& os, const Person per)
    {
        os << "Name: " << per.name << "   Dog's Name: "
           << per.dog.name  << "  Dog's age: "  << per.dog.age;
        return os;
    }
};


int main()
{
    Dog dog( "Fido", 4 );
    Person p( "Zaphod", dog );
    cout << p << endl;
}
```

PROBLEM 12:

Is Person p1's dog the same object as Person p2's dog in the code shown below?

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

class Dog {
    string name;
    int age;
public:
    Dog( string nam, int a ) : name( nam ), age( a ) {}
    friend ostream& operator<<( ostream& os, const Dog dog)
    {
        os << "Dog's Name: " << dog.name << "  Dog's Age: " << dog.age;
        return os;
    }
};

class Person {
    string name;
    Dog dog;
public:
    Person( string nam, Dog& d ) : name( nam ), dog( d ) {}
    void changeDog( Dog newDog ) { dog = newDog; }
    friend ostream& operator<<( ostream& os, const Person per)
    {
        os << "Name: " << per.name << "   Dog:    " << per.dog;
        return os;
    }
};


int main()
{
    Dog dog1( "Fido", 4 );
    Dog dog2( "Zoro", 8 );

    Person p1( "Zaphod", dog1 );
    cout << p1 << endl;

    Person p2 = p1;
    cout << p2 << endl;

    p1.changeDog( dog2 );

    cout << p1 << endl;
```

```
    cout << p2 << endl;

    return 0;
}
```

Solution:

No. Person's p1's dog is a different object than Person p2's dog. So if you change p1's dog, p2's dog will remain the same.

PROBLEM 13:

The class Person in the following C++ program has a vector data member, but is not provided with a copy constructor of its own. When we say

```
Person p2 = p1;
```

The system has no choice but to use the default copy constructor for Person. Is the vector data member of p2 the same as that of p1, or is it a copy?

```
#include <string>
#include <vector>
using namespace std;

class Dog {
    string name;
    int age;
public:
    Dog( string nam, int a ) : name( nam ), age( a ) {}
    friend ostream& operator<<( ostream& os, const Dog dog) {
        os << "Dog's Name: " << dog.name
           << "  Dog's Age: " << dog.age;
        return os;
    }
};

class Person {
    string name;
    vector<Dog> dogs;
public:
    Person( string nam, vector<Dog> dv )
                        : name( nam ), dogs( dv ) {}

    void getNewDog( Dog newDog ) {
        dogs.pop_back();
```

```
        dogs.push_back( newDog );
    }

    friend ostream& operator<<(ostream& os, const Person per){
        os << "Name: " << per.name << "   Dogs:    " << endl;
        printDogs( os, per );
        return os;
    }

    friend void printDogs( ostream& os, const Person& per  );
};

void printDogs( ostream& os, const Person& per  ) {
    vector<Dog>::const_iterator p = per.dogs.begin();
    while ( p != per.dogs.end() )
        os << *p++ << endl;    // needs overloaded '<<' for Dog
    os << endl << endl;
}

int main()
{
    Dog dog1( "Fido", 4 );
    Dog dog2( "Zoro", 8 );
    Dog dog3( "Ninja", 10 );

    vector<Dog> dogvec;
    dogvec.push_back( dog1 );
    dogvec.push_back( dog2 );

    Person p1( "Zaphod", dogvec );

    cout << p1 << endl;

    Person p2 = p1;        // uses system-supplied copy const.
    cout << p2 << endl;

    p1.getNewDog( dog3 );

    cout << p1 << endl;
    cout << p2 << endl;

    return 0;
}
```

Solution:

The fact that `Person` has not been provided with a copy constructor is not a problem. The system-supplied default copy constructor for `Person` invokes the system-supplied copy constructor for for `vector<Dog>` to make a copy of the vector data member of `p1` for `p2`. As a result, any changes made in `p1`'s vector data member are not reflected in `p2`'s vector data member.

12

Overloading Operators in C++

PROBLEM 1:

The following program elicits an "undefined symbol problem" message from the linker. What's the problem with this program? How can the problem be fixed?

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

class X {
    friend ostream& operator<<( ostream&, X& );
};

ostream& operator<<( ostream& os, const X& xobj ) {
    os << "message from an X object" << endl;
}

int main()
{
    X xob;
    cout << xob;
    return 0;
}
```

Solution:

Since accessing private members of X is not an issue, the compiler believes that it can use the global overload definition for the output operator. So as far as the compiler is concerned, the statement

```
cout << xob;
```

in `main()` is alright. But then when the linker tries to link togethre the various functions together, it notices that the compiler has asked for the $<<$ operator and that the class has a prototype for this operator. Not finding a definition for the specific prototype, the linker produces the error shown.

There are two ways to make the above program work:

- Delete the friend declaration altogether. Now the program would be able to use the overload outside the class. Since accessing private members of X is not an issue here, you do not need the friend declaration.
- Declare the second argument in the prototype to be of type 'const'

There is a third way also – delete the 'const' in the second argument of the operator overload definition – but it is not the right way because of the difficulty of initializing non-const variables by reference.

PROBLEM 2:

The following program has a flaw similar to the one you saw in the previous program, but it results in a different error message from the compiler. Why?

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

class X {
    int n;
public:
    X() : n(10) {}
    friend ostream& operator<<( ostream&, X& );
};

ostream& operator<<( ostream& os, const X& xobj ) {
    os << "n: " << xobj.n << endl;
}

int main()
{
    X xob;
    cout << xob;
```

```
        return 0;
    }
```

Solution:

The program tries to use the overload definition outside the class for the statement

```
        cout << xob
```

in `main()`. But then it discovers that it does not have access to the private data member n of X. Hence the error message. This error message comes about even before the compiler can figure out that it does not have a definition for the overload prototype inside the class.

PROBLEM 3:

Define a function object that would allow a vector of strings to be sorted in a case-insensitive manner.

Solution:

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


class Nocase {
public:
  bool operator() ( const string&, const string& ) const;
};

bool Nocase::operator() ( const string& x, const string& y ) const
{
  int p = 0;
  while ( ( p < x.size() ) &&
          ( p < y.size() ) &&
          ( toupper( x[p] ) == toupper( y[p] ) ) ) {
    ++p;
  }
  if ( p == x.size() ) return p != y.size();
  return toupper( x[p] ) < toupper( y[p] );
}
```

```
int main() {
  vector<string> countries;
  typedef vector<string>::iterator Iter;

  countries.push_back( "France" );
  countries.push_back( "Japan" );
  countries.push_back( "Greece" );
  countries.push_back( "India" );
  countries.push_back( "banana_republic" );

  vector<string> copy_countries = countries;

  cout << "Original list:      ";
  for ( Iter p = countries.begin(); p != countries.end(); p++ )    // (A)
    cout << *p << " ";
  cout << endl;

  sort(countries.begin(), countries.end() );                       // (B)

  cout << "Sorted list:        ";
  for ( p = countries.begin(); p != countries.end(); p++ )         // (C)
    cout << *p << " ";
  cout << endl;

  countries = copy_countries;
  sort( countries.begin(), countries.end(), Nocase() );            // (D)

  cout << "Sorted with Nocase: ";
  for ( p = countries.begin(); p != countries.end(); p++ )         // (E)
    cout << *p << " ";
  cout << endl;
}
```

This program produces the following output

```
Original list:      France Japan Greece India banana_republic
Sorted list:        France Greece India Japan banana_republic
Sorted with Nocase: banana_republic France Greece India Japan
```

PROBLEM 4:

Modify the program for the sorting example of Section 12.10 for the case when the object to be sorted has private data members.

Solution:

When the class whose objects are meant to be sorted has private data members, you'd have to make the following modifi cations to code of Section 12.10:

- Bring the `Cat_Comparator` class inside the `Cat` class. In OO, this is called nesting. So the `Cat_Comparator` class is a nested class inside the `Cat` class.
- Declare the overload defi nition of `operator()` for the `Cat_Comparator` class to be a friend of the `Cat` class to give it access to the private members of `Cat`.

```
#include <string>
#include <list>
using namespace std;

class Cat {
    string name;
    int age;
public:

    class Cat_Comparator {
    public:
        bool operator() ( const Cat& x1, const Cat& x2 ) const
        {
           return x1.age < x2.age;
        }
    };

    Cat( string nam, int yy) : name(nam), age( yy ) {}

    string getName() const { return name; }
    int getAge() const { return age; }

    friend bool Cat_Comparator::operator() ( const Cat& x1, const Cat& x2 ) const;
};

template<class T> void print( list<T> );

int main()
{
    Cat kitty1( "socks", 6 );
    Cat kitty2( "cuddles", 3 );
    Cat kitty3( "tabby", 8 );
    Cat kitty4( "afra", 12 );

    list<Cat> kittyList;

    kittyList.push_back( kitty1 );
    kittyList.push_back( kitty2 );
```

```
    kittyList.push_back( kitty3 );
    kittyList.push_back( kitty4 );

    print( kittyList );

    kittyList.push_front( kitty2 );
    cout << "After push_front:" << endl;
    print( kittyList );

    kittyList.pop_front( );
    cout << "After pop_front:" << endl;
    print( kittyList );

    kittyList.sort( Cat::Cat_Comparator() );
    cout << "After sort:" << endl;
    print( kittyList );
}

template<class T> void print( list<T> li ) {
  typedef list<T>::const_iterator CI;
  cout << "The number of items in the list: " <<  li.size() << endl;;
  for ( CI iter = li.begin(); iter != li.end(); iter++ )
      cout << iter->getName() << " " << iter->getAge() << endl;
  cout << endl << endl;
}
```

PROBLEM 5:

Modify the program of the previous problem to sort a vector of user-defi ned class type objects.

Solution:

```
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

class Cat {
    string name;
    int age;
public:

    class Cat_Comparator {
    public:
        bool operator() ( const Cat& x1, const Cat& x2 ) const
```

```
        {
           return x1.age < x2.age;
        }
    };

    Cat( string nam, int yy) : name(nam), age( yy ) {}

    string getName() const { return name; }
    int getAge() const { return age; }

    friend bool Cat_Comparator::operator() ( const Cat& x1, const Cat& x2 ) const;
};

template<class T> void print( vector<T> );

int main()
{
    Cat kitty1( "socks", 6 );
    Cat kitty2( "cuddles", 3 );
    Cat kitty3( "tabby", 8 );
    Cat kitty4( "afra", 12 );

    vector<Cat> kittyVec;

    kittyVec.push_back( kitty1 );
    kittyVec.push_back( kitty2 );
    kittyVec.push_back( kitty3 );
    kittyVec.push_back( kitty4 );

    print( kittyVec );

    sort( kittyVec.begin(), kittyVec.end(), Cat::Cat_Comparator() );
    cout << "After sort:" << endl;
    print( kittyVec );
}

template<class T> void print( vector<T> li ) {
  typedef vector<T>::const_iterator CI;
  cout << "The number of items in the list: " <<  li.size() << endl;;
  for ( CI iter = li.begin(); iter != li.end(); iter++ )
      cout << iter->getName() << " " << iter->getAge() << endl;
  cout << endl << endl;
}
```

PROBLEM 6:

The program of Section 12.11 shows how a list of class type objects can be sorted by merely supplying an overload definition for the operator $<$. Modify that code for the case of vector of class type objects.

Solution:

```
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

class Cat {
    string name;
    int age;
public:
    Cat( string nam, int yy) : name(nam), age( yy ) {}

    string getName() const { return name; }
    int getAge() const { return age; }

    friend bool operator<( const Cat& x1, const Cat& x2 );
};


bool operator<( const Cat& x1, const Cat& x2 ) {
    return x1.age < x2.age;
}

template<class T> void print( vector<T> );

int main()
{
    Cat kitty1( "socks", 6 );
    Cat kitty2( "cuddles", 3 );
    Cat kitty3( "tabby", 8 );
    Cat kitty4( "afra", 12 );

    vector<Cat> kittyVec;

    kittyVec.push_back( kitty1 );
    kittyVec.push_back( kitty2 );
    kittyVec.push_back( kitty3 );
    kittyVec.push_back( kitty4 );

    print( kittyVec );
```

```
    sort( kittyVec.begin(), kittyVec.end() );
    cout << "After sort:" << endl;
    print( kittyVec );
}

template<class T> void print( vector<T> li ) {
  typedef vector<T>::const_iterator CI;
  cout << "The number of items in the list: " <<  li.size() << endl;;
  for ( CI iter = li.begin(); iter != li.end(); iter++ )
      cout << iter->getName() << " " << iter->getAge() << endl;
  cout << endl << endl;
}
```

PROBLEM 7:

Pretending that a programmer would not need a copy constructor or a copy assignment operator and as an exercise in giving additional smarts to a smart pointer, write a smart pointer class for your own string type, for example for the `MyString` type discussed in Section 12.6. In addition to the usual smarts of automatic freeing up of the memory occupied by the string owned by a smart pointer, also give it the following smarts: Whenever a string goes out of scope, it should get dumped into a string archive.

Solution:

```
//SmartPtr5.cc

#include "MyString.h"
#include <fstream>
using namespace std;

class SmartPtr {
    MyString* str_ptr;

    // forbid copy construction
    SmartPtr( const SmartPtr& );

    // forbid assignment operation
    SmartPtr& operator=( const SmartPtr& );

public:
    SmartPtr( MyString* s ) : str_ptr( s ) {};

    MyString& operator*() const { return *str_ptr; }
```

```
    MyString* operator->() const { return str_ptr; }

    ~SmartPtr() {
        ofstream out( "string_archive.txt", ios::app );
        out << *this;
        delete str_ptr;
        out.close();
    }

    friend ostream& operator<< ( ostream&, const SmartPtr& );
};

ostream& operator<< ( ostream& os, const SmartPtr& smartptr ) {
    os << *smartptr << endl;
    return os;
}

int main()
{
    MyString* s1 = new MyString( "here" );
    SmartPtr p( s1 );
    cout << p->getSize() << "    " << p->getCharArray() << endl;

    MyString* s2 = new MyString( "there" );
    SmartPtr q( s2 );
    cout << q->getSize() << "    " << q->getCharArray() << endl;

    MyString* s3 = new MyString( "everywhere" );
    SmartPtr r( s3 );
    cout << r->getSize() << "    " << r->getCharArray() << endl;

//    SmartPtr s = p;                   // not allowed
//    p = q;                            // not allowed
}
```

If you examine the contents of the file "string_archive.txt", the strings will show up in that file with `s3` first, then `s2`, and `s1` last. That is because as variables go out of scope, the destructors are called in an order that is opposite to the one in which the constructors were invoked (see Chapter 11).

PROBLEM 8:

Generalize the smart pointer of the previous question so that it can be used to point to either a single object or to an array of objects. Overload the increment and the decrement operators so that an array can be stepped through in the same manner as with a regular pointer. Make sure

that you range check the pointer as it is incremented or decremented so that there is not range violation.

Solution:

```
//SmartPtrArray.cc

#include "MyString.h"
#include <fstream>
using namespace std;


class SmartPtr {
    MyString* str_ptr;
    MyString* str_ptr_original;
    int strArraySize;
    int offset;

    // copy constructor needed by postfix ++ and -- ops
    // but not appropriate for general use
    SmartPtr( const SmartPtr& other )
        : strArraySize( other.strArraySize ),
          offset( other.offset )
    {
        str_ptr_original = new MyString[ strArraySize ];

        for (int i=0; i < strArraySize; i++)
            *(str_ptr_original + i) =
               MyString( *(other.str_ptr_original + i ));

        str_ptr = str_ptr_original + offset;
    }


    // forbid assignment operations
    SmartPtr& operator=( const SmartPtr& );

public:
    SmartPtr( MyString* s = 0, int arraySize = 0 )
          : str_ptr( s ),
            str_ptr_original( s ),
            strArraySize( arraySize ),
            offset( 0 ) {}

    MyString& operator*() const { return *str_ptr; }

    MyString* operator->() const { return str_ptr; }
```

```
~SmartPtr()
{
    if ( str_ptr_original != 0 )
        if ( strArraySize == 0 ) delete str_ptr_original;
        else delete[] str_ptr_original;
}

SmartPtr& operator++() {
    if ( strArraySize == 0) {
        cerr << "\nCannot increment pointer to a single object\n";
        return *this;
    }
    if ( offset >= strArraySize - 1 ) {
        cerr << "\nAlready at the end of the array\n";
        return *this;
    }
    ++offset;
    ++str_ptr;
    return *this;
}

SmartPtr& operator--() {
    if (strArraySize == 0) {
      cerr << "\nCannot decrement pointer to a single object\n";
      return *this;
    }
    if ( offset <= 0 ) {
        cerr << "\nAlready at the beginning of the array\n";
        return *this;
    }
    --offset;
    --str_ptr;
    return *this;
}

const SmartPtr operator++(int) {
    SmartPtr oldValue = *this;

    if ( strArraySize == 0) {
        cerr << "\nCannot increment pointer to a single object\n";
        return *this;
    }
    if ( offset == strArraySize ) {
        cerr << "\nAlready one past the end of the array\n";
        return *this;
    }
    ++(*this);
    return oldValue;
```

```
    }

    const SmartPtr  operator--(int) {
        SmartPtr oldValue = *this;

        if ( strArraySize == 0) {
            cerr << "\nCannot decrement pointer to a single object\n";
            return *this;
        }
        if ( offset == -1 ) {
            cerr << "\nAlready one before the beginning of the array\n";
            return *this;
        }

        --(*this);
        return oldValue;
    }

    int getOffset() const { return offset; }

    friend ostream& operator<<( ostream&, const SmartPtr& );

    friend istream& operator>>( istream&, SmartPtr& );
};


ostream& operator<<( ostream& os, const SmartPtr& smartptr ) {
    os << smartptr->getCharArray();
    return os;
}

istream& operator>>( istream& is, SmartPtr& smartptr ) {
    char buffer[100];
    is >> buffer;
    *(smartptr.str_ptr) = MyString( buffer );
    return is;
}

int main()
{
    MyString* s1 = new MyString("hello");
    SmartPtr p( s1 );
    cout << p->getSize() << endl;                    // 5
    cout << p->getCharArray() << endl;               // hello

    MyString* s2 = new MyString("mellow");
    SmartPtr q( s2 );
    cout << q->getSize() << endl;                    // 6
    cout << q->getCharArray() << endl;               // mellow
```

```
    int arrSize = 10;
    MyString* strArr = new MyString[ arrSize ];
    for (int i=0; i<arrSize; i++)
        *(strArr + i) = MyString("");
    SmartPtr pArr( strArr, arrSize );

    ifstream in( "infile.txt" );

    for (int i=0; i < arrSize; i++) {
        in >> pArr;
        cout << pArr << " ";
        ++pArr;
    }                                   // the hungry brown fox tried to jump
                                        //               over a cat
    cout << endl << endl;

    for (int i=0; i < 8; i++) {
        cout << pArr << " ";
        --pArr;
    }                                   // cat a over jump to tried fox brown

    in.close();

    cout << "\n\nTesting post decrement op: " << endl;
    cout << pArr   << endl;                              // hungry
    cout << pArr-- << endl;                              // hungry
    cout << pArr-- << endl;                              // the ( and warn msg )
    cout << pArr-- << endl;                              // the ( and warn msg )

    cout << "\n\nTesting post increment op: " << endl;
    cout << pArr   << endl;                               // the
    cout << pArr++ << endl;                               // the
    cout << pArr++ << endl;                               // hungry
    cout << pArr++ << endl;                               // brown
    cout << pArr++ << endl;                               // fox

    *pArr = MyString("fought");
    cout << pArr << endl;                                 // fought
    cout << pArr++ << endl;                               // fought
    cout << pArr++ << endl;                               // to

//    SmartPtr pArr2 = pArr;           // forbidden
                                       // copy constructor in private section
}
```

13

Generics and Templates

PROBLEM 1:

Provide implementations for a copy constructor and a copy assignment operator for the `LinkedList` class of Section 13.1.1.

Solution:

See the solution to the next problem. The solution here is just an easier version of that.

PROBLEM 2:

Provide implementations for a copy constructor and a copy assignment operator for the parameterized `LinkedList` class of Section 13.1.2.

Solution:

```
//LinkedListGenericWithCopyConstructor.cc

#include <iostream>
```

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

template <class T> class LinkedList {

    struct Node {
        Node* previous;
        Node* next;
        const T& item;
        Node( Node* p, Node* n, const T& t)
                 : previous(p), next(n), item(t) {}
    };

    Node* head;
public:
    LinkedList() : head() {}
    LinkedList( const T& t )  : head( new Node(0, 0, t)) {}

    // COPY CONSTRUCTOR WITH AN IN-CLASS DEFINITION:
    LinkedList( LinkedList& other ) {
        if ( other.head == 0 ) {
            head = 0;
            return;
        }
        Node* p = other.head;
        head = new Node( 0, 0, p->item );
        Node* q = head;
        while ( p->next ) {
            p = p->next;
            Node* r = new Node( 0, 0, p->item );
            q->next = r;
            r->previous = q;
            q = r;
        }
    }

    ~LinkedList();
    void addToList( const T& );
    void removeFromList( const T& );
    void printAll();
};


template<class T> LinkedList<T>::~LinkedList() {
    Node* p = head;
    while ( p != 0 ) {
        Node* temp = p;
        p = p->next;
        delete temp;
```

```
    }
}


template <class T> void LinkedList<T>::addToList( const T& item ) {
    Node* p = head;
    //check if the list was created previously.  If not
    //start the list:
    if ( p == 0 ) {
        head = new Node( 0, 0, item );
        return;
    }
    //find the end of the list:
    while (p->next)
        p = p->next;
    //now add a new node at the end:
    p->next = new Node(0, 0, item);
    p->next->previous = p;
}

template<class T> void LinkedList<T>::removeFromList( const T& item ) {
    Node* p = head;
    for (; p->item != item; p = p->next)                          //(A)
        if (p->next == 0) return;   // item not in list
    if (p == head) {                // item in the first node
        head = head->next;
        head->previous = 0;
        delete( p );
        return;
    }
    p->previous->next = p->next;
    if (p->next != 0)         // item to be deleted is at the end of list
      p->next->previous = p->previous;
    delete( p );
}

template<class T> void LinkedList<T>::printAll() {
    for (Node* p = head; p; p = p->next )
        cout << p->item << ' ';
}

int main() {

    //a lined list of ints:
    int i = 1;
    LinkedList<int>* numlist = new LinkedList<int>(i);
    numlist->addToList( 5 );
    numlist->addToList( 6 );
    numlist->printAll();                       // 1 5 6
```

```
    cout << endl;
    numlist->removeFromList( 6 );
    numlist->printAll();                          // 1 5
    cout << endl;


    // TEST COPY CONSTRUCTOR:

    LinkedList<int> list2 = *numlist;

    list2.printAll();

    cout << endl;

    list2.addToList( 100 );
    list2.printAll();                              // 1 5 100

    cout << endl;

    numlist->printAll();                           // 1 5

    delete numlist;
}
```

PROBLEM 3:

Create a parameterized Java class `Buffer<T>` that could be used as a buffer for different types of data. At the least, your class should implement the following interface:

```
interface BufferInterface<T> {
    public void addElement(T data);
    public void addElementAt(T data, int index);
    public T removeElementAt(int index);
    public int getSize();
}
```

Solution:

```
//Buffer.java

//compile this program with the gjc compiler

public class Buffer<T> implements BufferInterface<T>{

  private T[]  buff_;
```

```
 private int size_;

 public Buffer(int size) {
   buff_ = new T[size];
   size_ = 0;
 }

/**
 * store the element at the next empty location
 **/
 public void addElement(T data) {
   if (size_ == buff_.length) {
     System.out.println("buffer is full");
   }
   else {
     int i;
     for (i=0;i<buff_.length;i++) {
       if (buff_[i] == null) break;
     }
     buff_[i] = data;
     size_++;
   }
 }

/**
 *  override the element at index specified
 **/
 public void addElementAt(T data, int index) {
   if (index >= buff_.length) {
     System.out.println("index out of bound");
   }
   else {
     if (buff_[index] == null) {
       size_++;
     }
     buff_[index] = data;
   }
 }

 public T removeElementAt(int index) {
   if (index >= buff_.length) {
     System.out.println("index out of bound");
     return null;
   }
   else {
     T val = buff_[index];
     if (val != null) {
       size_--;
       buff_[index] = null;
```

```
      }
      return val;
    }
  }

  public int getSize() {
    return size_;
  }

  public void printBuffer() {
    for (int i=0;i<buff_.length;i++) {
      System.out.println("buffer[" + i + "] = " + buff_[i]);
    }
  }

}//class Buffer

class Test {

  public static void main (String[] args) {
    System.out.println("\n----Integer buffer example");
    Buffer<Integer> intBuffer = new Buffer<Integer>(5);
    intBuffer.addElement(new Integer(8));
    intBuffer.addElement(new Integer(9));
    //intBuffer.addElement("error");  //error
    intBuffer.removeElementAt(2);
    System.out.println("Buffer size = "+ intBuffer.getSize());
    intBuffer.printBuffer();

    System.out.println("\n----String buffer example");
    Buffer<String> strBuffer = new Buffer<String>(5);
    strBuffer.addElementAt("hello",3);
    strBuffer.addElement("first hi");
    strBuffer.removeElementAt(0);
    strBuffer.addElement("first hi again");
    System.out.println("Buffer size = "+ strBuffer.getSize());
    strBuffer.printBuffer();

  }
}
```

PROBLEM 4:

Write a parameterized lookup table class, `Lookup<T1, T2>`, in Java for storing `<key, value>` pairs where `T1` is the type parameter for `key` and `T2` the type parameter for `value`. The class `Lookup` should support at least the following methods:

boolean addEntry(T1 key, T2 value) – to add an entry into the table. This method should return true if the operation is successful, and false otherwise. Only one `value` entry is allowed for a `key`. If `<key, value>` is inserted a second time for the same `key`, the new `value` should replace the old.

boolean removeEntry(T1 key) – to remove an entry from the table for a given `key`. It should return true if operation is successful, and false otherwise.

T1 retrieveKey(T2 value) – to retrieve the key associated with a given value.

T2 retrieveValue(T1 key) – to return the value associated with the specified key.

boolean hasValue(T2 value) – to test whether a particular is stored in the table.

boolean hasKey(T1 key) – to test whether the specified key is stored in the table.

boolean expandTable(int size) – to increase the capacity of the table so that it can hold the specified number of `<key, value>` pairs. Must return false if the table cannot be expanded.

boolean isFull() – returns true if the table is full, false otherwise.

void printTable() – to display the table in a two-column format, with the first column for the keys and the second for the corresponding values.

The user needs to specify the size of the table when it is first constructed.

Solution:

```
//Lookup.java

//compile with the gjc compiler

public class Lookup<T1,T2> {
  T1[] key_;
  T2[] value_;
  int total_elements_;

  public Lookup(int size) {
    key_ = new T1[size];
    value_ = new T2[size];
    total_elements_ = 0;
  }

  public boolean addEntry(T1 key, T2 value) {
    // key can't be null
    if (key == null) return false;

    int index = -1;
    for (int i=0;i<key_.length;i++) {
      if (key_[i] != null && key_[i].equals(key)) {
        index = i;
        break;
```

```
      }

      if (index == -1 && key_[i] == null) {
        index = i;
      }
    }

    if (index >= 0) {
      key_[index] = key;
      value_[index] = value;
      total_elements_++;
      return true;
    }
    return false;

  }

  public boolean removeEntry(T1 key) {
    if (key == null) return false;
    for (int i=0;i<key_.length;i++) {
      if (key_[i] != null && key_[i].equals(key)) {
        key_[i] = null;
        value_[i] = null;
        total_elements_--;
        return true;
      }
    }
    return false;
  }

  public T2 retrieveValue( T1 key ) {
    T2 value = null;

    for (int i=0;i<key_.length;i++) {
      if (key_[i] != null && key_[i].equals(key)) {
        value = value_[i];
        break;
      }
    }

    return value;
  }

  public T1 retrieveKey(T2 value) {
    T1 key = null;

    for (int i=0;i<value_.length;i++) {
      if (value_[i] != null && value_[i].equals( value )) {
        key = key_[i];
```

```
      break;
    }
  }

  return key;
}

public boolean hasValue( T2 value ) {
  for (int i=0;i<value_.length;i++) {
    if ( value_[i] != null && value_[i].equals( value ) ) return true;
  }
  return false;
}

public boolean hasKey(T1 key) {
  for (int i=0;i<key_.length;i++) {
    if (key_[i] != null && key_[i].equals(key)) return true;
  }
  return false;
}

public boolean isFull() {
  if (total_elements_ == key_.length) {
    return true;
  }
  return false;
}

public boolean expandTable( int size ) {
  if (size <= key_.length) return false;
  T1[] newKey = new T1[size];
  T2[] newValue = new T2[size];
  for (int i=0;i<key_.length;i++) {
    newKey[i] = key_[i];
    newValue[i] = value_[i];
  }

  key_ = newKey;
  value_ = newValue;
  return true;
}

public void printTable() {
  for(int i=0;i<key_.length;i++) {
    if (key_[i] != null) {
      System.out.println(key_[i] + ", " + value_[i]);
    }
  }
  System.out.println("\n");
```

```
  }
}

class Test {

  public static void main(String[] args) {
    Lookup<String,Integer> phonedir = new Lookup<String,Integer>(100);
    phonedir.addEntry("Tom",new Integer(4567));
    phonedir.addEntry("Ann",new Integer(3333));
    phonedir.addEntry("Jack",new Integer(5432));
    phonedir.addEntry("Tom",new Integer(5672));
    phonedir.printTable();

    System.out.println(phonedir.retrieveValue("Tom")); //5672
    System.out.println(phonedir.hasKey("Tim")); //false
    System.out.println(phonedir.hasValue(new Integer(3333))); //True
    System.out.println(phonedir.retrieveKey(new Integer(3333))); //Ann

    phonedir.removeEntry("Tom");
    phonedir.printTable();

  }
}
```

PROBLEM 5:

Write a parameterized C++ version of the Java class of the previous homework problem. As for the Java homework, the C++ class will have two template parameters, `T1` for `key` type and `T2` for `value` type. Assume that the types used for `T1` and `T2` have pre-existing overload definitions for the operators == and ! =. No need to write separate template specializations for the pointer types for `T1` and `T2`.

Solution:

```
//Lookup.cc

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

template<class T1, class T2> class Lookup {
  T1** key_;
  T2** item_;
  int size_;
  int total_elements_;
```

```
public:
  Lookup(int size);
  bool addEntry (const T1& key, const T2& item);
  bool removeEntry(const T1& key);
  T2& retrieveItem(const T1& key);
  T1& retrieveKey(const T2& item);
  bool hasItem(const T2& item);
  bool hasKey(const T1& key);
  bool isFull();
  bool expandTable(int size);
  void printTable();

};


template<class T1, class T2> Lookup<T1,T2>::Lookup(int size) {
  key_ = new T1*[size];
  item_ = new T2*[size];
  size_ = size;
  //initialize table
  for (int i=0;i<size_;i++) {
    key_[i] = 0;
    item_[i] = 0;
  }
}

template<class T1, class T2>
bool Lookup<T1,T2>::addEntry(const T1& key, const T2& item) {

  int index = -1;
  for (int i=0;i<size_;i++) {
    if (key_[i] != 0 && *key_[i] == key) {
      index = i;
      break;
    }
    if (index == -1 && key_[i] == 0) {
      index = i;
      break;
    }
  }

  if (index >= 0) {
    key_[index] = new T1(key);
    item_[index] = new T2(item);
    total_elements_++;
    return true;
  }
```

```
  return false;
}

template<class T1, class T2>
bool Lookup<T1,T2>::removeEntry(const T1& key) {
  for (int i=0;i<size_;i++) {
    if (key_[i] != 0 && *key_[i] == key) {
      key_[i] = 0;
      item_[i] = 0;
      total_elements_--;
      return true;
    }
  }
  return false;
}

template<class T1, class T2>
T2& Lookup<T1,T2>::retrieveItem(const T1& key) {
  T2* item = 0;

  for (int i=0;i<size_;i++) {
    if (key_[i] != 0 && *key_[i] == key) {
      item = item_[i];
      break;
    }
  }
  return *item;
}

template<class T1, class T2>
T1& Lookup<T1,T2>::retrieveKey(const T2& item) {
  T1* key = 0;

  for (int i=0;i<size_;i++) {
    if (item_[i] != 0 &&  *item_[i] == item) {
      key = key_[i];
      break;
    }
  }
  return *key;
}

template<class T1, class T2>
bool Lookup<T1,T2>::hasItem(const T2& item) {
  for (int i=0;i<size_;i++) {
    if (item_[i] != 0 && *item_[i] == item) {
      return true;
    }
  }
```

```
  return false;
}

template<class T1, class T2>
bool Lookup<T1,T2>::hasKey(const T1& key) {
  for (int i=0;i<size_;i++) {
    if (key_[i] != 0 && *key_[i] == key) {
      return true;
    }
  }
  return false;
}

template<class T1, class T2> bool Lookup<T1,T2>::isFull() {
  if (total_elements_ == size_) {
    return true;
  }
  return false;
}

template<class T1, class T2>
bool Lookup<T1,T2>::expandTable(int size) {
  if (size <= size_) return false;
  T1* newkey[] = new T1*[size];
  T2* newitem[] = new T2*[size];

  for (int i=0;i<size_;i++) {
    newkey[i] = key_[i];
    newitem[i] = item_[i];
  }

  for (int i=size_;i<size;i++) {
    newkey[i] = 0;
    newitem[i] = 0;
  }

  key_ = newkey;
  item_ = newitem;
  return true;
}

template<class T1, class T2>
void Lookup<T1,T2>::printTable() {
  for (int i=0;i<size_;i++) {
    if (key_[i] != 0) {
      cout<<*key_[i] <<", " << *item_[i]<<endl;
    }
  }
  cout<<endl;
```

```
}

int main() {
  Lookup<string,int> phonedir = Lookup<string,int>(100);
  phonedir.addEntry("Tom",4567);
  phonedir.addEntry("Ann",3333);
  phonedir.addEntry("Jack",5432);
  phonedir.addEntry("Tom",5672);
  phonedir.printTable();

  cout<< phonedir.retrieveItem("Tom")<<endl; //5672
  cout<< phonedir.hasKey("Tim")<<endl; //false
  cout<< phonedir.hasItem(3333)<<endl; //True
  cout<< phonedir.retrieveKey(3333)<<endl; //Ann

  phonedir.removeEntry("Tom");
  phonedir.printTable();
  return 0;
}
```

PROBLEM 6:

Write a C++ template class that can serve as a dynamic FIFO queue for an arbitrary data type with support for at least the following methods. `T` is the template parameter for the element type in the queue.

void enqueue(const T& item) – for adding a new element at the end of the queue.

T& dequeue() – for removing the element at the front of the queue. The method must throw an exception if the element does not exist.

int size() – for returning the number of elements currently in the queue.

T elementAt(int i) – for ascertaining the element (without removing it) at the specifi ed location.

void printQueue() – for displaying the contents of the queue.

Solution:

```
//MyQueue.cc

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

class NoSuchElementException{};
```

```
template <class T> class MyQueue {

  struct Node {
    T item;
    Node* next;
    Node (T t) : next(0), item(t) {}
  };

  Node* head_;
  Node* tail_;

public:
  MyQueue ();
  void enqueue (const T& item);
  T& dequeue ();
  int size ();
  T elementAt (int i);
  void printQueue ();
  ~MyQueue() { delete head_,tail_;}
};

 /**
  * Constructor for MyQueue
  **/
  template <class T> MyQueue<T>::MyQueue() : head_(0), tail_(0) {}

 /**
  * store the new element at the tail of the queue
  **/
  template <class T> void MyQueue<T>::enqueue (const T& item) {
    Node* new_item = new Node(item);

    if (head_ == 0) {
      head_ = tail_ = new_item;
    }
    else {
      (*tail_).next = new_item;
      tail_ = new_item;
    }
  }

 /**
  * remove the element from the head of the queue
  **/
  template <class T> T& MyQueue<T>::dequeue() {
    Node* temp = head_;
    if (head_ == 0) {
      throw NoSuchElementException();
    }
```

```
    else {
      head_ = (*head_).next;
      (*temp).next = 0;
      return (*temp).item;
    }
  }

  template <class T> void MyQueue<T>::printQueue() {
    Node* current;
    for (current = head_;current != 0; current = (*current).next) {
      cout << (*current).item << " ";
    }
    cout << endl;
  }

int main() {
  MyQueue<int> intQueue;
  intQueue.enqueue(5);
  intQueue.enqueue(6);
  intQueue.enqueue(7);
  intQueue.dequeue();
  intQueue.printQueue();

  MyQueue<string> strQueue;
  strQueue.enqueue("one");
  strQueue.enqueue("two");
  strQueue.enqueue("three");
  strQueue.dequeue();
  strQueue.printQueue();

  return 0;
}
```

PROBLEM 7:

Defi ne a parameterized Java class `MyQueue<T>` to serve as a dynamic FIFO queue for elements of arbitrary data type. The class `MyQueue<T>` should implement the following interface

```
interface QueueInterface<T> {
    public void enqueue (T item);
    public T dequeue ();
    public Iterator<T> getIterator();

    class NoSuchElementException extends RuntimeException {}
}
```

where `Iterator<T>` is a parameterized interface defi ned as

```
interface Iterator<T> {
    public boolean hasNext();
    public T next();
    public boolean getNext();
}
```

with all methods possessing the usual semantics for the queue container and for Java iterators.

Solution:

```
//MyQueue1.java

//compile with the gjc compiler


public class MyQueue<T> implements QueueInterface<T> {

  protected class Node {
     T item;
     Node next = null;
     Node (T item) { this.item = item; }
  }

  protected class QueueIterator implements Iterator<T> {
    private Node ptr = head_;

    public boolean hasNext() { return ptr != null; }

    public T next() {
      if (ptr != null) {
        T item = ptr.item;
        ptr = ptr.next;
        return item;
      } else throw new QueueInterface.NoSuchElementException();
    }

    public boolean getNext() {
      if (ptr == null) return false;
      else {
        ptr = ptr.next;
        return true;
      }
    }
  }

  protected Node head_ = null;
  protected Node tail_ = null;
```

```
 /**
  * constructor for MyQueue
  **/
  public MyQueue() {}

  public Iterator<T> getIterator() {
    return new QueueIterator();
  }

/**
 * store the new element at the tail of the queue
 **/
  public void enqueue (T item) {
    Node new_item = new Node(item);

    if (head_ == null) {
      head_ = tail_ = new_item;
    }
    else {
      tail_.next = new_item;
      tail_ = new_item;
    }


  }

/**
 * remove the element from the head of the queue
 **/
  public T dequeue () {
    Node temp = head_;
    if (head_ == null) {
      throw new QueueInterface.NoSuchElementException();
    }
    else {
      head_ = head_.next;
      temp.next = null;
      return temp.item;
    }
  }

  public void printQueue() {
    Node current;
    for (current = head_;current != null;current = current.next) {
      System.out.println(current.item +" ");
    }
  }
```

```
}

class Test {
    public static void main (String[] args) {

    //Queue that holds Integer only
    MyQueue<Integer> intQueue = new MyQueue<Integer>();
    intQueue.enqueue(new Integer(5));
    intQueue.enqueue(new Integer(6));
    intQueue.enqueue(new Integer(7));
    //intQueue.enqueue(new String("one")); //error
    intQueue.dequeue();
    intQueue.printQueue();

    //Queue that holds String only
    MyQueue<String> strQueue = new MyQueue<String>();
    strQueue.enqueue("one");
    strQueue.enqueue("two");
    strQueue.enqueue("three");
    //strQueue.enqueue(new Integer(1)); //error
    strQueue.dequeue();
    strQueue.printQueue();
    }
}
```

PROBLEM 8:

Augment your solution for the previous problem by providing the implementation code for the following methods:

1. A parameterized Java method that returns an `int` for the number of elements in the queue.
2. A parameterized Java method that returns the i^{th} element in the queue.

Both these methods should work on any implementation of the `QueueInterface` defined in part (a) of the problem.

Solution:

```
//MyQeue2.java

//compile with the gjc compiler


class QueueMethods {
```

```
  public static <T> int QueueSize(QueueInterface<T> concrete_queue) {
    if (concrete_queue == null) return 0;

    Iterator<T> iter = concrete_queue.getIterator();
    int size = 0;
    for (iter=null;iter.hasNext();iter.getNext()) {
      size++;
    }
    return size;
  }

  public static <T> T ElementAt(QueueInterface<T> concrete_queue, int index) {
    if (concrete_queue == null) return null;

    T element = null;
    Iterator<T> iter = concrete_queue.getIterator();
    for(int i=0;i<index-1;i++) {
      if (!iter.getNext()) {
        return null;
      }
    }

    if (iter.hasNext()) {
      element = iter.next();
    }

    return element;
  }
}
```

14

Modeling Diagrams for OO Programs

PROBLEM 1:

Construct a use-case diagram that portrays the following actors and use-cases:

1. an actor named `Student`
2. an actor named `College`
3. the following use cases specific to the actor `Student`: "List All Courses", "Pay Fees", "Registration", and "Search for a Course". The use case "Search for a Course", meant to help a student decide which courses to sign up for, should extend the use case "List All Courses."
4. the following uses specific to the actor `College`: "Maintain a Database of Students", "Registration", "Administer Examinations", "Check Student Qualifications". The use case "Registration" is the same as for the actor `Student`.

Solution:

The solution is shown in Figure 14.1.

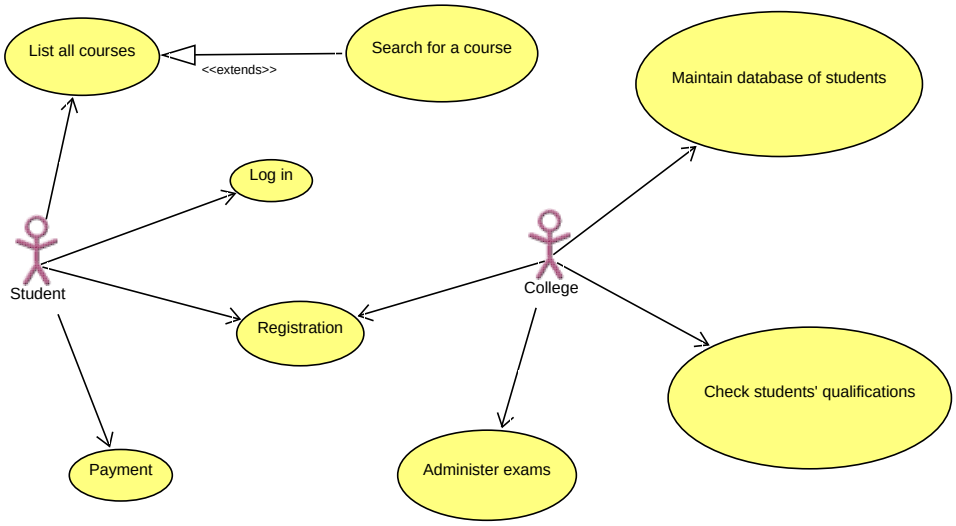


Fig. 14.1

PROBLEM 2:

Draw a sequence diagram showing the time progression of the interaction between the various classes needed for the following closely related set of use cases for an on-line course registration system for a college:

> The student first logs into the on-line course registration system. The student browses through the courses being offered next semester and selects what he/she needs. The on-line registration system makes sure that the student has the pre-requisites for the courses selected and there is space available in those courses. The course selection is automatically forwarded to a faculty advisor. Upon approval from the advisor, a fee statement is sent to the student. If the faculty advisor disapproves, the student is notified. The student is considered registered when the fee is received. At that point, the student is sent his/her class schedule.

You may first want to list the individual use cases needed for the interaction described above; then identify the classes the required for the use cases; and, finally, construct an interaction diagram. Also show a collaboration diagram.

Solution:

The solution is shown in Figure 14.2.

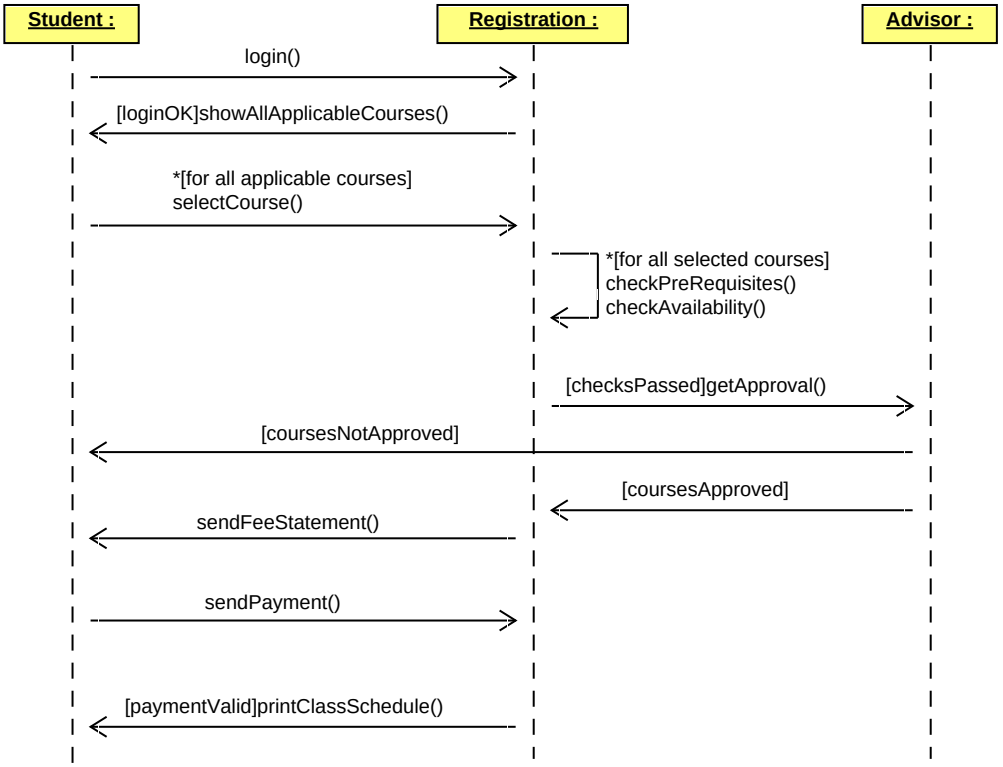

Student :
Registration :
Advisor :
login()
[loginOK]showAllApplicableCourses()
*[for all applicable courses]
selectCourse()
*[for all selected courses]
checkPreRequisites()
checkAvailability()
[checksPassed]getApproval()
[coursesNotApproved]
[coursesApproved]
sendFeeStatement()
sendPayment()
[paymentValid]printClassSchedule()

Fig. 14.2

PROBLEM 3:

Draw a state diagram that shows the different states of a `Student` object in the uses cases of the previous problem.

Solution:

The solution is shown in Figure **??**.

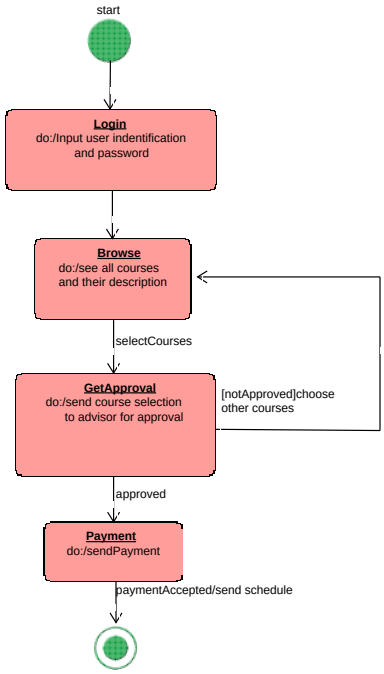


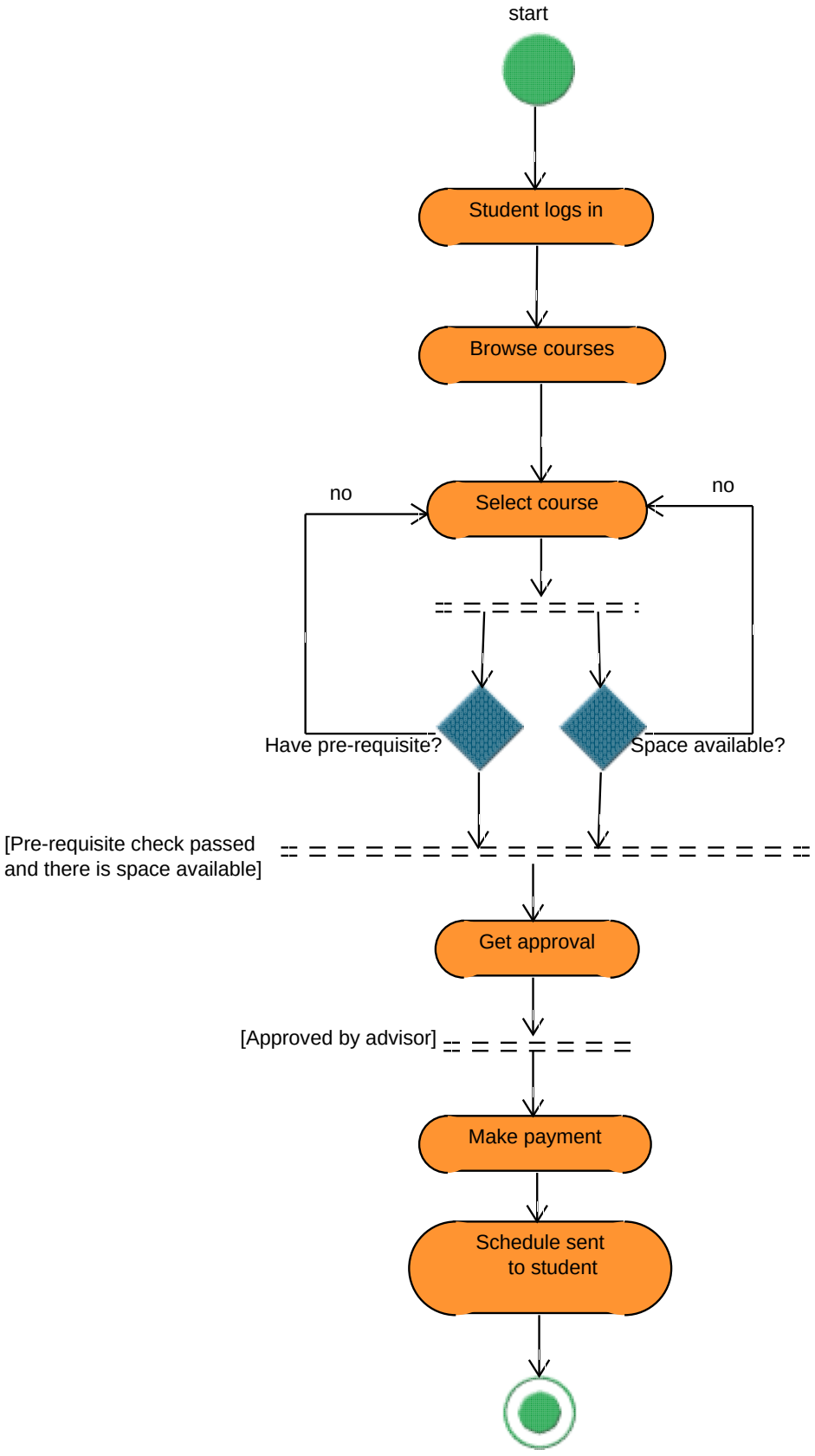
Fig. 14.3

PROBLEM 4:

Constuct an activity diagram for the interactions of Problem 4.

Solution:

The solution is shown in Figure 14.4.


start
Student logs in
Browse courses
Select course
no
no
Have pre-requisite?
Space available?
[Pre-requisite check passed
and there is space available]
Get approval
[Approved by advisor]
Make payment
Schedule sent
to student

15

Extending Classes

PROBLEM 1:

Identify and correct the error in the derived-class constructor syntax in this Java code.

```
class Employee{
  private String fName, lName;
  public Employee(String f, String l) {
    fName = f;
    lName = l;
  }
}

class Manager extends Employee{
  private short level;
  public Manager(String f, String l, short lvl) {
    Employee(f, l);
    level = lvl;
  }
  public Manager(String f, String l) {
    this(f, l, 10);
  }
}
```

State what the members of the following object will be set to after the following line is executed:

```
Employee e = new Manager ( ‘‘John’’,  ‘‘Doe’’);
```

Solution:

`Employee(f, l)` should be `super(f, l)`.

`e.fName = John, e.lName = Doe, e.level = 10`

PROBLEM 2:

Is the following C++ syntax legal? Note that the base class `X` and the derived class `Y` use the same identifiers `m` and `foo` for the class members. If the code is legal, what's its output?

```
class X {
public:
    int m;
    X( int mm ) : m( mm ) {}
    void foo() { cout << "X's foo invoked" << endl; }
};

class Y : public X {
public:
    int m;
    Y( int a, int b ) : X(a), m( b ) {}
    void foo() { cout << "Y's foo invoked" << endl; }
};

int main() {
    X xobj( 10 );
    xobj.foo();

    Y yobj( 10, 20 );
    yobj.foo();

    yobj.X::foo();
}
```

Solution:

The code is legal. Its output is

```
X's foo invoked
Y's foo invoked
X's foo invoked
```

PROBLEM 3:

If the goal was to write correctly the copy constructor for a derived-class, is the following example correct?

```
class X {
   int m;
public:
   X( int mm ) : m( mm ) {}
   X( const X& other ) : m( other.m ) {}
};


class Y : public X {
   int n;
public:
   Y( int mm, int nn ) : X( mm ), n( nn ) {}
   Y( const Y& other ) : n( other.n ) {}
};
```

Solution:

The copy constructor for Y is wrong. It is missing `X( other )` in the initializer part.

PROBLEM 4:

The following C++ program compiles without any problems. When run, it even prints out the "hello" called for in line (B) of `main`. But subsequently the program aborts with a memory segmentation fault. Why? (Hint: If you comment out the assignment statement in line (A), the program runs flawlessly.)

```
//MysteryBug.cc

#include <iostream>
using namespace std;

class X {
    int* p;
    int size;
public:
    X() { p = 0; size = 0; }
    X( int* ptr, int sz ) : size( sz ) {
        p = new int[ size ];
```

```
        for ( int i=0; i<size; i++ ) p[i] = ptr[i];
    }
    ~X() { delete[] p; }
};


class Y : public X {
    int n;
public:
    Y() {};
    Y( int* ptr, int sz, int nn ) : X( ptr, sz ), n( nn ) {}
    Y( const Y& other ) : X( other ), n( other.n ) {}
    Y& operator=( const Y& other ) {
        if ( this == &other ) return *this;
        X::operator=( other );
        n = other.n;
        return *this;
    }
};

int main() {
    int data[ 3 ] = {3, 2, 1};
    Y y1( data, 3, 10 );
    Y y2;
    y2 = y1;                            //(A)
    cout << "hello" << endl;            //(B)
}
```

Solution:

The segmentation fault is caused by the last statement in main:

```
    y2 = y1;
```

If you comment out this statement, the program runs fine. Why does this statement create a problem? It is because the base class X has not been provided with a proper assignment operator. So the compiler uses the default copy assignment for executing the `y2 = y1` statement. As a result, the `X` sub-objects sitting inside the two `Y` objects `y1` and `y2` will have their pointer data member p pointing to exactly the same block of memory for the int array.

So when both `y1` and `y2` go out of scope, the system will invoke the default destructor for both. That will cause `X`'s destructor to be invoked for destroying the `X` sub-objects inside the `Y` objects. But destruction of the `X` sub-objects will cause the 'delete' operator to be applied to the same block of memory on the heap, which causes the segmentation fault.

PROBLEM 5:

While the following C++ program compiles fine

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

class X {
public:
    void foo();
    X() { }
};

class Y : public X {
public:
    void foo(){ cout << "Y's foo invoked" << endl; }
    Y() {}
};

int main() {}
```

the following program does not compile and reports a problem from the linker. Why?

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

class X {
public:
    virtual void foo();
    X() { }
};

class Y : public X {
public:
    void foo(){ cout << "Y's foo invoked" << endl; }
    Y() {}
};

int main() {}
```

Solution:

Now it is the linker that will squawk. Remember, if a function is declared to be virtual, the compiler wants to see its definition right there. Since no definition has been provided for the virtual foo() in X, the linker will notice the lack of the definition and complain.

If in the above example, we did not extend X to Y, the linker will stop complaining. By extending X to Y, the linker has to construct the virtual table that associates the types with the functions defined at different levels along an inheritance path. The linker would want to construct such a virtual table, but it would not be able to locate the function definition for the declaration in X.

PROBLEM 6:

Will the following C++ program compile?

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

class X {
public:
    virtual void f();
};

class Y : public X { };

int main()
{
    Y yobj;
}
```

Solution:

No, because the non-pure virtual function in X has no definition. A virtual function must be defined as and when it is declared virtual.

PROBLEM 7:

Will the following version of the previous program compile?

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

```
class X {
public:
    virtual void f() = 0;
};

class Y : public X { };

int main()
{
    Y yobj;
}
```

Solution:

No, because Y remains abstract. So an object of type Y cannot be constructed in main()

PROBLEM 8:

Will the following version of the previous program compile and run?

```
#include <iostream>

class X {
public:
    virtual void f() = 0;
};

class Y : public X { void f() {}; };

int main()
{
    Y yobj;
}
```

Solution:

Yes. There are no problems with this program.

PROBLEM 9:

With regard to exception specifi cations for overriding functions in Java, is the following code legal?

```
class MyException extends Exception {}

class X {
    public void foo() throws MyException  {
        throw new MyException();
    }
}


class Y extends X {
    public void foo()  {
        System.out.println( "Y's foo invoked" );
    }
}
```

Solution:

This code is legal. The exception specifi cation of the overriding `foo()` in Y is a null set, which is a subset of the exception specifi cation of `foo()` in X.

PROBLEM 10:

Do you see any problems with the following Java code? Note that both the interfaces `X` and `Y` possess a function of the same signature.

```
interface X { public int foo( int m ); }

interface Y { public int foo( int n ); }

class W implements X, Y {
      public int foo( int x ) { return x; }
}
```

Solution:

No problems with this program. Different interfaces are allowed to have functions of the same signature.

PROBLEM 11:

Vis-a-vis the code in the previous problem, the Java code shown here shows two interfaces possessing function `foo` of the same name but differente signatures. Do you see any problems with this code? Will it compile?

```
interface X { public int foo( int m ); }

interface Y { public int foo( int n, int p ); }

class W implements X, Y {
      public int foo( int x ) { return x; }
}
```

Solution:

You will get a compiler error. The compiler will insist that you declare W to be abstract since it does not defi ne `foo(int, int)` inherited from the Y interface.

PROBLEM 12:

Will the following Java program compile?

```
interface X { public int foo( int m ); }

class X { public int foo( int q ) { return q; } }

class W implements X {
      public int foo( int x ) { return x; }
}
```

Solution:

No. The compiler will complain that it has two classes of the same name but with different defi nitions. Remember, inherently, an interface is also a class.

16

Multiple Inheritance in C++

PROBLEM 1:

In the following program, `X` is a virtual base for `Y`, which in turn is a regular base for `Z`. The constructors for both `Y` and `Z` invoke `X`'s constructor. Notice that `Z`'s constructor in line (B) invokes its two superclass constructors with two different arguments, 22 and 11. While the argument of 11 gets passed on to `X` constructor directly in line (B), the argument 22 gets passed on to `X` constructor in line (A). What will be printed out in line (C) for the data member `x` inherited by `Z`? Will it be 11 or 22?

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

class X {
public:
    int x;
    X( int xx ) : x( xx ) {}
};

class Y : virtual public X {
public:
    Y( int xx ) : X( xx ) {}         //(A)
};

class Z : public Y {
public:
```

```
    Z() : Y( 22 ), X( 11 ) {}        //(B)
};

int main()
{
    Z zobj;
    cout  << zobj.x << endl;         //(C)
    return 0;
}
```

Solution:

11

PROBLEM 2:

In the following program, even though the derived class Z does not make a direct invocation of the constructor of the virtual base X, the program compiles fine. Why?

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

class X {
    int x;
public:
    X() {}
    X( int xx ) : x( xx ) {}
};

class Y : virtual public X {
public:
    Y( int xx ) : X( xx ) {}
};

class Z : public Y {
public:
    Z( int xx ) : Y( xx ) {}            // (A)
};

int main() {}
```

Solution:

Not finding an explicit invocation of the virtual base X in line (A), the system uses the user-defined no-arg constructor for X by default.

The lesson here is that if you have virtual base and you do not invoke it directly in a downstream constructor, the system will try to invoke the no-arg constructor of the virtual base.

PROBLEM 3:

In the following program, `X` is a virtual base for `Y`, which in turn is a regular base for `Z`. The compiler outputs the following error message for this program:

```
No matching function for call to X::X()
```

Why?

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

class X {
    int x;
public:
    X( int xx ) : x( xx ) {}
};

class Y : virtual public X {
public:
    Y( int xx ) : X( xx ) {}
};

class Z : public Y {
public:
    Z( int xx ) : Y( xx ) {}            // (A)
};

int main() {}
```

Solution:

Not finding an explicit invocation of the virtual base constructor in line (A), the compiler tries to invoke the no-arg constructor for the virtual base. But since X has a regular constructor defined for it, it does not have a no-arg constructor.

PROBLEM 4:

In the following program, X is a virtual base for Y, which in turn is a regular base for Z. The output statement in line (A) causes the number -4261484 (or some other equally unexpected and bizarre looking number) to be printed out on the terminal. Why?

```
#include <iostream>

class X {
    int x;
public:
    X() {}
    X( int xx ) : x( xx ) {}
    int getx() const { return x; }
};


class Y : virtual public X {
public:
    Y( int xx ) : X( xx ) {}
};


class Z : public Y {
public:
    Z( int xx ) : Y( xx ) {}
};


int main()
{
    Z zobj( 100 );
    cout << zobj.getx() << endl;           //(A)
}
```

Solution:

Not finding a direct invocation of the virtual base constructor in the constructor for class Z, the system tries to use the the no-arg constructor for X. But the no-arg constructor for X is a do-nothing constructor. It deposits random bits in the memory assigned to the data member x. So that is what is output by the program. (Lesson: In a large program, this could be a very difficult to locate bug since the code compiles and runs without complaining.)

PROBLEM 5:

The following program has a diamond hierarchy of classes. `X` is a public base of `Y` and `T`. And, the classes `Y` and `T` are both superclasses of `U`. Note that `X` at the apex of the hierarchy is not a virtual base. This program does not compile. Why?

```
#include <iostream>

class X {
public:
    int x;
    X( int xx ) : x( xx ) {}
};


class Y : public X {
public:
    Y( int xx ) : X( xx ) {}
};


class T : public X {
public:
    T( int xx ) : X( xx ) {}
};


class U : public Y, public T {
public:
    U( int xx ) : Y( xx ), T( xx ) {}
};


int main()
{
    U uobj( 100 );
    cout  << uobj.x << endl;          // (A)
}
```

Solution:

The compiler says "The request for x is ambiguous."

The reason is that without a virtual base, there are two versions of the data member x visible in U. Since X is no longer a virtual base, U contains inside it two sub-objects of type X, each with its own x. The ambiguity referred to by the compiler is about these two x's being simultaneously visible in line (A).

PROBLEM 6:

Write a Java version of the mixin classes example of Section 16.9. Use Java interfaces for serving the same purpose as the mixin classes of that section.

PROBLEM 7:

Write a Java version of the role playing classes of Section 16.10. Use Java interfaces for the abstract role types `SalesType` and `ManagerType` of that section.

PROBLEM 8:

C++ also makes available a function `typeid()` that can be used to ascertain the class identity of an object. Using this function, the implementation of the `checkReadyForPromotion()` function in Section 16.10 would become

```
bool checkReadyForPromotion( Employee* e )
{
  Role* r = e->getActiveRole();

  if ( typeid( *r ) == typeid( Manager ) ) {
    if ( r->getRoleExperience() >= MinYearsForPromotion ) {
      cout << "yes, ready for promotion in the active role" << endl;
      return true;
    }
    else {
      cout << "No, not yet ready for promotion in the active role" << endl;
      return false;
    }
  }
  else {
    cout << "Unable to determine if ready for promotion" << endl;
    return false;
  }
}
```

Incorporate this version of `checkReadyForPromotion()` in the source code in Section 16.10.

17

OO for Graphical User Interfaces, A Tour of Three Toolkits

PROBLEM 1:

As an excercise in using a panel to organize components in a window, write a Java program that allows you to enter text in a `JTextArea` component and then to fetch the text entered for display in a terminal window. Your top-level frame should consist of two `JPanel` objects arranged vertically. The top `JPanel` object should contain a label at the very top announcing the purpose of the window, and a text-entry area below that. The text-entry area can be an object of type `JTextArea`. The bottom panel should consisst of three buttons as follows:

1. A button, with label "Start", that allows text to be entered in the text-area part of the window.
2. A button, labeled "Stop", that when pressed disallows any further keyboard entry of text.
3. A button, labeled "Get Text", that extracts the entire text entered by the user in the text window and then displays the text in the user's terminal.

Solution:

```
//Chap17Prob1.java

import javax.swing.*;                       // for JTextArea, JPanel, etc.
import javax.swing.event.*;                 // for DocumentListener
import javax.swing.text.*;                  // for Document interface
import java.awt.*;           // for Container, BorderLayout, etc
```

```
import java.awt.event.*;      // for WindowAdapter


public class Chap17Prob1 {

    public static void main( String[] args ) {
        JFrame f = new JFrame( "Chap17Prob1" );

        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        Container contentPane = f.getContentPane();

        JPanel textPanel = new JPanel();
        textPanel.setBorder(
            BorderFactory.createEmptyBorder( 10, 10, 10, 10 ) );
        textPanel.setLayout(
                    new BoxLayout( textPanel, BoxLayout.Y_AXIS ) );

        JLabel label = new JLabel( "A window for editing text" );
        textPanel.add( label );
        textPanel.add(
                Box.createRigidArea( new Dimension( 0, 10 ) ) );

        final JTextArea textArea = new JTextArea( 30, 20 );
        textArea.setEditable( false );
        JScrollPane textPane = new JScrollPane( textArea );
        textPanel.add( textPane );

        contentPane.add( textPanel, BorderLayout.CENTER );

        JButton startButton = new JButton( "Start" );
        startButton.addActionListener( new ActionListener() {
                public void actionPerformed( ActionEvent e ) {
                    textArea.setEditable( true );
                }
            });
        JButton stopButton = new JButton( "Stop" );
        stopButton.addActionListener( new ActionListener() {
                public void actionPerformed( ActionEvent e ) {
                    textArea.setEditable( false );
                }
            });
        JButton gettextButton = new JButton( "Get Text" );
        gettextButton.addActionListener( new ActionListener() {
                public void actionPerformed( ActionEvent e ) {
```

```
                System.out.println( textArea.getText() );
            }
        });

    JPanel buttonPanel = new JPanel();
    buttonPanel.setLayout(
             new BoxLayout( buttonPanel, BoxLayout.X_AXIS ) );
    buttonPanel.setBorder(
             BorderFactory.createEmptyBorder(0,10,10,10 ) );
    buttonPanel.add( Box.createHorizontalGlue() );
    buttonPanel.add( startButton );
    buttonPanel.add(
            Box.createRigidArea( new Dimension( 10, 0 ) ) );
    buttonPanel.add( stopButton );
    buttonPanel.add(
            Box.createRigidArea( new Dimension( 10, 0 ) ) );
    buttonPanel.add( gettextButton );

    contentPane.add( buttonPanel, BorderLayout.SOUTH );

    f.pack();
    f.setLocation( 200, 300 );
    f.setVisible( true );
  }
}
```

PROBLEM 2:

Design a `ColorMixer` applet for your web page. As depicted in Figure 17.44, the upper half of the applet should consist of a `Canvas` object that shows the result of mixing R, G, and B color components. The lower third of the applet should be a panel consisting of three sub-panels, as depicted in greater detail in Figure 17.45. Each of these subpanels should like what is shown in Figure 17.46 — that is, each subpanel should consist of a `Canvas` object at the top and a `Scrollbar` and a `TextField` object at the bottom. The scrollbar is for altering the color value of a color component. The text fi eld is for displaying a numerical readout of the current value of that component.

Solution:

A notable feature of the `addPaneledScrollbar()` method in the solution below is the manner in which it makes a color component panel a listener for the `AdjustmentEvents` generated by a user's interaction with the scrollbar for the color component. When a user clicks on either of the arrows at the two ends of a scrollbar, the scrollbar object generates a `UNIT_INCREMENT` or a `UNIT_DECREMENT` event, depending on which arrow was clicked on. When he/she clicks on either side of the slider, the scrollbar object generates a `BLOCK_INCREMENT` or a

BLOCK_DECREMENT event, depending on what side of the slider the user clicked on. And when a user drags the slider, the scrollbar generates the TRACK event. Regardless of which of the three modes of interaction was chosen by a user, the method getValue() returns the current numerical value associated with the scrollbar object.

The following code shows the entire code for the applet. Also shown, inside a commented out block at the beginning, is the HTML code that must go into a separate file, in this example a file called ColorMixer.html.

```
/*
\\HTML filename: ColorMixer.html

<HTML>
<TITLE>
An Applet for Testing Color Mixtures
</TITLE>
<BODY BGCOLOR="#000000">
<APPLET CODE="ColorMixer.java"  WIDTH=480 HEIGHT=590>
</APPLET>

</BODY>
</HTML>
*/

\\Java filename: ColorMixer.java

import java.awt.*;
import java.awt.event.*;
import java.applet.*;


public class ColorMixer extends Applet {

  private Canvas mainCanvas = new Canvas();

  private int redValue = 128;
  private int greenValue = 128;
  private int blueValue = 128;

  private Panel redPanel = new Panel();
  private Panel greenPanel = new Panel();
  private Panel bluePanel = new Panel();

  private BreakPanel breakPanel = new BreakPanel();

  private TextField redTF = new TextField( 3 );
  private TextField greenTF = new TextField( 3 );
  private TextField blueTF = new TextField( 3 );
```

```
private Canvas redCanvas = new Canvas();
private Canvas greenCanvas = new Canvas();
private Canvas blueCanvas = new Canvas();

private Scrollbar redbar = new Scrollbar( Scrollbar.HORIZONTAL, 128, 0, 0, 255);
private Scrollbar greenbar = new Scrollbar( Scrollbar.HORIZONTAL, 128, 0, 0, 255);
private Scrollbar bluebar = new Scrollbar( Scrollbar.HORIZONTAL, 128, 0, 0, 255);

public void init()
{
  setLayout( new BorderLayout() );

  mainCanvas.setSize( 480, 410 );
  add( mainCanvas, "North" );

  breakPanel.setSize(480,10);
  add( breakPanel );

  ScrollbarPanel sbp = new ScrollbarPanel();
  add( sbp, "South" );

  redCanvas.setBackground( new Color( redValue, 0, 0 ) );
  greenCanvas.setBackground( new Color( 0, greenValue, 0 ) );
  blueCanvas.setBackground( new Color( 0, 0, blueValue ) );

  mainCanvas.setBackground( new Color( redValue, greenValue, blueValue ) );
}

// Inner class: ScrollBarPanel
class ScrollbarPanel extends Panel {

  public ScrollbarPanel()
  {
    setLayout( new GridLayout( 1, 3 ) );

    addPaneledScrollbar( redPanel, redbar, redTF, redCanvas );
    addPaneledScrollbar( greenPanel, greenbar, greenTF, greenCanvas );
    addPaneledScrollbar( bluePanel, bluebar, blueTF, blueCanvas );
  }

  void addPaneledScrollbar( Panel pa, Scrollbar sbar, TextField tf, Canvas cv )
  {
    pa.setLayout( new BorderLayout() );
    cv.setSize( 100, 100 );
    pa.add( cv, "North" );
    Panel r = new Panel();
    tf.setText( "128" );
    tf.setBackground( Color.white );
    r.add( tf );
```

```
      sbar.setUnitIncrement( 16 );
      sbar.setBlockIncrement( 16 );
      sbar.addAdjustmentListener(
          new AdjustmentListener() {
            public void adjustmentValueChanged( AdjustmentEvent evt )
            {
              redValue = redbar.getValue();
              greenValue = greenbar.getValue();
              blueValue = bluebar.getValue();

              redCanvas.setBackground( new Color( redValue, 0, 0 ) );
              greenCanvas.setBackground( new Color( 0, greenValue, 0 ) );
              blueCanvas.setBackground( new Color( 0, 0, blueValue ) );

              redTF.setText( Integer.toString(redValue) );
              greenTF.setText( Integer.toString(greenValue) );
              blueTF.setText( Integer.toString(blueValue) );

              mainCanvas.setBackground( new Color( redValue, greenValue, blueValue ) );
             }
          } );
      r.add( sbar );
      pa.add( r, "South" );
      add( pa );
    }
  }
  // end of inner class ScrollBarPanel
}

class BreakPanel extends Panel {

  public void paint( Graphics g )
  {
    Shape sh = g.getClip();
    g.setColor( Color.white );
    g.fillRect(sh.getBounds().x, sh.getBounds().y, sh.getBounds().width, sh.getBounds().heig
  }
}
```

PROBLEM 3:

Create a simple two function (add/subtract) calculator using Qt. It should also include an all clear (AC) button. Use `QGridLayout` to manage the positioning of the buttons and the display. A `QLineEdit` object can be used for the display, and `QPushButton` objects should be used for the buttons.

Solution:

```
////////////////  file:  Calc.h   ////////////////

#ifndef CALC_H
#define CALC_H

#include <qdialog.h>

#include <qlayout.h>
#include <qpushbutton.h>

#include "MyPushButton.h"
#include "MyLineEdit.h"

class Calc : public QDialog
{

public:
  Calc ();

};

#endif

////////////////  file: Calc.cc   /////////////////

#include "Calc.h"

Calc::Calc ()
{
  MyPushButton* buttons[10];

  buttons[0] = new MyPushButton ("0", this);
  buttons[1] = new MyPushButton ("1", this);
  buttons[2] = new MyPushButton ("2", this);
  buttons[3] = new MyPushButton ("3", this);
  buttons[4] = new MyPushButton ("4", this);
  buttons[5] = new MyPushButton ("5", this);
  buttons[6] = new MyPushButton ("6", this);
  buttons[7] = new MyPushButton ("7", this);
  buttons[8] = new MyPushButton ("8", this);
  buttons[9] = new MyPushButton ("9", this);

  QPushButton* button_plus = new QPushButton ("+", this);
  QPushButton* button_minus = new QPushButton ("-", this);
  QPushButton* button_equal = new QPushButton ("=", this);
```

```
QPushButton* button_clear = new QPushButton ("C", this);

MyLineEdit* text = new MyLineEdit (this);
text->setReadOnly (TRUE);

QGridLayout* layout = new QGridLayout (this, 5, 4);
layout->addMultiCellWidget (buttons[0], 4, 4, 0, 2);
layout->addWidget (buttons[1], 3, 0);
layout->addWidget (buttons[2], 3, 1);
layout->addWidget (buttons[3], 3, 2);
layout->addWidget (buttons[4], 2, 0);
layout->addWidget (buttons[5], 2, 1);
layout->addWidget (buttons[6], 2, 2);
layout->addWidget (buttons[7], 1, 0);
layout->addWidget (buttons[8], 1, 1);
layout->addWidget (buttons[9], 1, 2);

layout->addWidget (button_plus, 3, 3);
layout->addWidget (button_minus, 2, 3);
layout->addWidget (button_equal, 4, 3);

layout->addMultiCellWidget (text, 0, 0, 0, 3);
layout->addWidget (button_clear, 1, 3);

layout->activate ();

QObject::connect (button_clear,
                  SIGNAL (pressed ()),
                  text,
                  SLOT (needsCleared ()));

// operator buttons
QObject::connect (button_plus,
                  SIGNAL (pressed ()),
                  text,
                  SLOT (opAdd ()));

QObject::connect (button_minus,
                  SIGNAL (pressed ()),
                  text,
                  SLOT (opSubtract ()));

QObject::connect (button_equal,
                  SIGNAL (pressed ()),
                  text,
                  SLOT (opEqual ()));


for (int i = 0; i < 10; i++)
```

```
  {
    QObject::connect (buttons[i],
                      SIGNAL (sendChar (const QString&)),
                      text,
                      SLOT (addText (const QString&)));
  }
}


////////////////////   file:  MyLineEdit.h   ///////////////////

// this is for the window at the top of the calculator where you
// show the numbers entered by the user and the results of calculations.

#ifndef MYTEXTEDIT_H
#define MYTEXTEDIT_H

#include <qlineedit.h>


typedef enum
{
  ADD,
  SUBTRACT,
  NONE
} operation;



class MyLineEdit : public QLineEdit
{
  Q_OBJECT

  QString number;
  int total;
  int saved_number;
  bool clear_flag;
  bool set_clear_flag;
  operation current_op;

  void compute ();

public:
  MyLineEdit (QWidget* wid);

signals:
  void clearMe ();
  void newText (const QString&);
```

```
public slots:
  void needsCleared ();
  void addText (const QString&);

  void opAdd ();
  void opSubtract ();
  void opEqual ();

};

#endif


////////////////////  file:  MyLineEdit.cc   ////////////////////

#include "MyLineEdit.h"
#include <iostream>

MyLineEdit::MyLineEdit (QWidget* wid) : QLineEdit (wid) {
  number = "0";
  total = 0;
  saved_number = 0;
  clear_flag = FALSE;
  current_op = NONE;

  setAlignment (Qt::AlignRight);

  QObject::connect (this,
                    SIGNAL (newText (const QString&)),
                    this,
                    SLOT (setText (const QString&)));

  emit newText (number);
}

void MyLineEdit::needsCleared () {
  number = "0";
  total = 0;
  saved_number = 0;

  emit newText (number);
}

void MyLineEdit::addText (const QString& text) {
  if ((number == "0" && text != "0") || clear_flag)
  {
    clear_flag = FALSE;
    number = text;
  }
```

```
  else
  {
    number += text;
  }

  emit newText (number);
}

void MyLineEdit::opAdd () {
  saved_number = text ().toInt ();

  compute ();

  clear_flag = TRUE;
  current_op = ADD;
}

void MyLineEdit::opSubtract () {
  saved_number = text ().toInt ();

  compute ();

  clear_flag = TRUE;
  current_op = SUBTRACT;
}

void MyLineEdit::opEqual () {
  saved_number = text ().toInt ();

  compute ();

  clear_flag = TRUE;
  current_op = NONE;
}

void MyLineEdit::compute () {
  if (current_op == ADD)
  {
    total += saved_number;
  }
  else if (current_op == SUBTRACT)
  {
    total -= saved_number;
  }
  else
  {
    total = saved_number;
  }
```

```
  number.setNum (total);

  emit newText (number);
}


///////////////// file:  MyPushButton.h   ////////////////

#ifndef MYPUSHBUTTON_H
#define MYPUSHBUTTON_H

#include <qpushbutton.h>
#include <qstring.h>
#include <qwidget.h>


class MyPushButton : public QPushButton
{
  Q_OBJECT

  QString num;

public:
  MyPushButton (const QString&, QWidget*);

signals:
  void sendChar (const QString&);

public slots:
  void buttonPressed ();
};

#endif


/////////////////  file: MyPushButton.cc  ////////////////

#include "MyPushButton.h"

MyPushButton::MyPushButton (const QString& title, QWidget* wid) :
    QPushButton (title, wid)
{
  num = title;

  QWidget::connect (this, SIGNAL (pressed ()),
                    this, SLOT (buttonPressed ()));
}

void MyPushButton::buttonPressed () {
```

```
  emit sendChar (num);
}


//////////////////   file: main.cc   ////////////////////

#include "Calc.h"
#include <qapplication.h>

int main (int argc, char** argv)
{
  QApplication a (argc, argv);

  Calc* win = new Calc ();

  win->show ();
  a.setMainWidget (win);

  return a.exec ();
}


//////////////////   file: Makefile   ///////////////////////

CC      = g++
MOC     = moc

LIBDIR  = -L$(QTDIR)/lib
LIBS    = -lqt
CFLAGS  = -I$(QTDIR)/include

OBJS    = moc_MyLineEdit.o MyLineEdit.o moc_MyPushButton.o MyPushButton.o \
          Calc.o main.o

calc: $(OBJS)
        $(CC) -o $@ $(OBJS) $(LIBDIR) $(LIBS)

main.o: main.cc
        $(CC) -c -o $@ $(CFLAGS) $<

Calc.o: Calc.cc Calc.h
        $(CC) -c -o $@ $(CFLAGS) $<

moc_MyLineEdit.cc: MyLineEdit.h
        $(MOC) -o $@ $<

moc_MyLineEdit.o: moc_MyLineEdit.cc
        $(CC) -c -o $@ $(CFLAGS) $<
```

```
MyLineEdit.o: MyLineEdit.cc MyLineEdit.h
        $(CC) -c -o $@ $(CFLAGS) MyLineEdit.cc

moc_MyPushButton.cc: MyPushButton.h
        $(MOC) -o $@ $<

moc_MyPushButton.o: moc_MyPushButton.cc
        $(CC) -c -o $@ $(CFLAGS) $<

MyPushButton.o: MyPushButton.cc MyPushButton.h
        $(CC) -c -o $@ $(CFLAGS) MyPushButton.cc

clean:
        rm -f calc
        rm -f *.o
        rm -f moc_*

//////////////////////////    END of the Calc program   //////////////
```

PROBLEM 4:

Design a countdown timer using Qt. A slider should control the amount of time the timer is set for, an LCD display to show how much time is left, and buttons to start, stop, and reset the timer. When the timer is started, the start and reset buttons should be disabled and the stop button enabled. When the timer expires or is stopped, the stop button should be disabled and the start and reset buttons should be enabled.

Solution:

```
////////////////////  file:  MyTimer.h  ////////////////////

#include <qtimer.h>

class MyTimer : public QTimer
{
  Q_OBJECT

public:
  MyTimer (QObject*);

signals:
  void setStartButton (bool);
  void setStopButton (bool);
  void setResetButton (bool);
```

```
public slots:
  void reachedZero ();
  void oneSecStart ();
  void oneSecStop ();
};


//////////////////    file: MyTimer.cc   ////////////////////

#include "MyTimer.h"

MyTimer::MyTimer (QObject* wid) : QTimer (wid) {}

void MyTimer::reachedZero () {
  oneSecStop ();
}

void MyTimer::oneSecStart () {
  // disable start and reset buttons, enable stop
  emit setStartButton (FALSE);
  emit setStopButton (TRUE);
  emit setResetButton (FALSE);

  start (1000);
}

void MyTimer::oneSecStop () {
  // enable start and reset, disable stop
  emit setStartButton (TRUE);
  emit setStopButton (FALSE);
  emit setResetButton (TRUE);

  stop ();
}


//////////////////   file: TimerGUI.h  ///////////////////

#ifndef TIMERGUI_H
#define TIMERGUI_H

#include <qdialog.h>
#include <qslider.h>
#include <qlayout.h>
#include <qpushbutton.h>
#include "MyTimer.h"
#include "MyLCDNumber.h"
```

```
class TimerGUI : public QDialog {

public:
  TimerGUI();

  QSlider* timeSlider;
  QPushButton* startButton;
  QPushButton* stopButton;
  QPushButton* resetButton;
  MyLCDNumber* LCDNumber;

  MyTimer* timer;

protected:
  QGridLayout* TimerLayout;
};

#endif // TIMERGUI_H


//////////////////  file: TimerGUI.cc  ////////////////////////

#include "TimerGUI.h"

TimerGUI::TimerGUI() : QDialog() {
  setName( "Timer" );
  setEnabled( TRUE );
  resize( 280, 245 );
  TimerLayout = new QGridLayout( this, 1, 1, 11, 6, "TimerLayout");

  timeSlider = new QSlider( this, "timeSlider" );
  timeSlider->setFocusPolicy( QSlider::ClickFocus );
  timeSlider->setMinValue( 1 );
  timeSlider->setMaxValue( 300 );
  timeSlider->setLineStep( 30 );
  timeSlider->setPageStep( 60 );
  timeSlider->setValue( 1 );
  timeSlider->setOrientation( QSlider::Horizontal );
  timeSlider->setTickmarks( QSlider::Both );

  TimerLayout->addMultiCellWidget( timeSlider, 0, 0, 0, 2 );

  startButton = new QPushButton( this, "startButton" );
  startButton->setText( "Start" );

  TimerLayout->addWidget( startButton, 2, 0 );

  stopButton = new QPushButton( this, "stopButton" );
```

```
stopButton->setText( "Stop" );
stopButton->setEnabled( FALSE );

TimerLayout->addWidget( stopButton, 2, 1 );

resetButton = new QPushButton( this, "resetButton" );
resetButton->setText( "Reset" );

TimerLayout->addWidget( resetButton, 2, 2 );

LCDNumber = new MyLCDNumber( this, "LCDNumber" );
LCDNumber->setFrameShape( QLCDNumber::NoFrame );
LCDNumber->setFrameShadow( QLCDNumber::Plain );
LCDNumber->setNumDigits( 3 );
LCDNumber->setMode( QLCDNumber::DEC );

TimerLayout->addMultiCellWidget( LCDNumber, 1, 1, 0, 2 );

// tab order
setTabOrder( timeSlider, startButton );
setTabOrder( startButton, stopButton );
setTabOrder( stopButton, resetButton );

QObject::connect (timeSlider,
                  SIGNAL (valueChanged (int)),
                  LCDNumber,
                  SLOT (display (int)));

QObject::connect (timeSlider,
                  SIGNAL (valueChanged (int)),
                  LCDNumber,
                  SLOT (setCurrVal (int)));

timer = new MyTimer (this);

QObject::connect (startButton,
                  SIGNAL (pressed ()),
                  timer,
                  SLOT (oneSecStart ()));

QObject::connect (stopButton,
                  SIGNAL (pressed ()),
                  timer,
                  SLOT (oneSecStop ()));

QObject::connect (resetButton,
                  SIGNAL (pressed ()),
                  LCDNumber,
                  SLOT (performReset ()));
```

```
  QObject::connect (timer,
                    SIGNAL (timeout ()),
                    LCDNumber,
                    SLOT (decrementCounter ()));

  QObject::connect (timer,
                    SIGNAL (setStartButton (bool)),
                    startButton,
                    SLOT (setEnabled (bool)));

  QObject::connect (timer,
                    SIGNAL (setStopButton (bool)),
                    stopButton,
                    SLOT (setEnabled (bool)));

  QObject::connect (timer,
                    SIGNAL (setResetButton (bool)),
                    resetButton,
                    SLOT (setEnabled (bool)));

  QObject::connect (LCDNumber,
                    SIGNAL (reachedZero ()),
                    timer,
                    SLOT (reachedZero ()));
}


////////////////////  file:  MyLCDNumber.h  ////////////////////

#ifndef MYLCDNUMBER_H
#define MYLCDNUMBER_H

#include <qlcdnumber.h>

class MyLCDNumber : public QLCDNumber {
  Q_OBJECT

  int currVal;

public:
  MyLCDNumber (QWidget*, char*);

signals:
  void reachedZero ();

public slots:
  void setCurrVal (int);
  void performReset ();
```

```
  void decrementCounter ();
};

#endif


////////////////////  file: MyLCDNumber.cc  ////////////////////

#include "MyLCDNumber.h"

MyLCDNumber::MyLCDNumber (QWidget* wid, char* title) :
  QLCDNumber (wid, title), currVal (0) {}

void MyLCDNumber::setCurrVal (int val) {
  currVal = val;
}

void MyLCDNumber::performReset () {
  display (currVal);
}

void MyLCDNumber::decrementCounter () {
  int currVal = intValue ();

  if (currVal > 0) {
    display (--currVal);
  }
  else {
    emit reachedZero ();
  }
}


////////////////  file: main.cc   ////////////////////////////

#include <qapplication.h>
#include "TimerGUI.h"

int main (int argc, char** argv)
{
  QApplication a (argc, argv);

  TimerGUI* timer = new TimerGUI();
  timer->show ();
  a.setMainWidget (timer);

  return a.exec ();
}
```

```
/////////////////   file: Makefile   ///////////////////////

CC      = g++
UIC     = uic
MOC     = moc

LIBDIR  = -L$(QTDIR)/lib
LIBS    = -lqt
CFLAGS  = -I$(QTDIR)/include

OBJS    = moc_MyTimer.o MyTimer.o moc_MyLCDNumber.o MyLCDNumber.o \
          TimerGUI.o main.o

timer: $(OBJS)
        $(CC) -o $@ $(CFLAGS) $(OBJS) $(LIBDIR) $(LIBS)

moc_MyLCDNumber.cc: MyLCDNumber.h
        $(MOC) -o $@ $<

moc_MyLCDNumber.o: moc_MyLCDNumber.cc
        $(CC) -c -o $@ $(CFLAGS) $<

MyLCDNumber.o: MyLCDNumber.cc MyLCDNumber.h
        $(CC) -c -o $@ $(CFLAGS) MyLCDNumber.cc

moc_MyTimer.cc: MyTimer.h
        $(MOC) -o $@ $<

moc_MyTimer.o: moc_MyTimer.cc
        $(CC) -c -o $@ $(CFLAGS) $<

MyTimer.o: MyTimer.cc MyTimer.h
        $(CC) -c -o $@ $(CFLAGS) MyTimer.cc

main.o: main.cc
        $(CC) -c -o $@ $(CFLAGS) $<

TimerGUI.o: TimerGUI.cc TimerGUI.h
        $(CC) -c -o $@ $(CFLAGS) TimerGUI.cc

#Timer.h: Timer.ui
#       $(UIC) $< -o $@

#Timer.cc: Timer.ui Timer.h
#       $(UIC) -impl Timer.h Timer.ui -o $@

clean:
        rm -f *.o moc_* timer
```

```
//////////////////// END of Timer homework ////////////////
```

PROBLEM 5:

Write a program using GNOME/GTK+ with a user interface divided into two main parts. The top is an empty area where the user can click. The bottom is a label where the coordinates of any mouse click in the upper area are displayed. The coordinates should be relative to the top left corner of the window.

Solution:

```
//////////////////  file:  clicker.c   ////////////////////

#include <gnome.h>
#include <stdio.h>

void close_window (GtkWidget* widget, GdkEvent* event, gpointer data)
{
  gtk_main_quit ();
}


void canvas_clicked (GtkWidget* widget, GdkEvent* event, gpointer data)
{
  GdkEventButton event_info = *(GdkEventButton*)event;
  char coord_str[50];

  sprintf (coord_str, "Mouse click at: %3.0lf, %3.0lf",
           event_info.x, event_info.y);

  gtk_label_set_text (GTK_LABEL (*(GtkLabel**)data), coord_str);
}


int main (int argc, char** argv)
{
  GtkWidget* top_window;

  GtkWidget* table;

  GtkWidget* canvas;    /* area to click in */
  GtkWidget* frame;
  GtkWidget* label;     /* display of click coordinates */

  gnome_init ("gnomewin", "1.0", argc, argv);
```

```
  top_window = gnome_app_new ("clicker", "Clicker");
  gtk_widget_set_usize (top_window, 500, 250);

  gtk_signal_connect (GTK_OBJECT (top_window), "delete-event",
                      GTK_SIGNAL_FUNC (close_window), NULL);

  frame = gtk_frame_new ("Click mouse in here");
  gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_ETCHED_OUT);

  /* area where the user will click */
  canvas = gnome_canvas_new ();
  gtk_container_add (GTK_CONTAINER (frame), GTK_WIDGET (canvas));

  gtk_signal_connect (GTK_OBJECT (canvas), "button-press-event",
                      GTK_SIGNAL_FUNC (canvas_clicked), &label);


  /* label, will display mouse coordinates, for now just instructions */
  label = gtk_label_new ("click on the canvas above");

  table = gtk_table_new (10, 1, TRUE);
  /* frame and canvas go on top */
  gtk_table_attach_defaults (GTK_TABLE (table), frame, 0, 1, 0, 9);
  /* label goes on bottom */
  gtk_table_attach_defaults (GTK_TABLE (table), label, 0, 1, 9, 10);

  gnome_app_set_contents (GNOME_APP (top_window), GTK_WIDGET (table));
  gtk_widget_show_all (top_window);

  gtk_main ();
  exit (0);
}

//////////////////////  file:  Makefile  ////////////////////

FILE    = clicker

CC      = gcc
LDLIBS  = `gnome-config --libs gnomeui`
CFLAGS  = `gnome-config --cflags gnomeui`

gnomewin: $(FILE).o
        $(CC) $(LDLIBS) $(FILE).o -o $(FILE)

gnomewin.o: $(FILE).c
        $(CC) $(CFLAGS) -c $(FILE).c

clean:
```

```
        rm -f $(FILE)
        rm -f $(FILE).o

////////////////// END of clicker.c   ////////////////////
```

PROBLEM 6:

Write a program using GNOME/GTK+ that can display images. The program should include a menu with open (open a picture), close (close a picture), exit (exit the program), and about (short message about the author) items. The image should be displayed inside of a scrolled window.

Solution:

```
////////////////////  file:  image_disp.c   ////////////////////

#include <gnome.h>
#include <stdio.h>

GtkWidget* scrolly;             /* scroller that holds image */
GtkWidget* image = NULL;        /* currently displayed image */


void close_window (GtkWidget* widget, GdkEvent* event, gpointer data)
{
  gtk_main_quit ();
} /* close_window */


/* begin menu setup */

open_callback(GtkWidget *button, gpointer data) {
  GtkWidget* file_dialog;
  GtkWidget* file_entry;
  GtkWidget* invalid_filename;
  gchar* filename;
  int ans;


  file_dialog = gnome_dialog_new ("Open .jpg file",
                                  GNOME_STOCK_BUTTON_OK,
                                  GNOME_STOCK_BUTTON_CANCEL,
                                  NULL);
```

```
  gnome_dialog_set_close (GNOME_DIALOG (file_dialog), TRUE);
  gnome_dialog_close_hides (GNOME_DIALOG (file_dialog), TRUE);

  file_entry = gnome_file_entry_new (NULL, "Open .jpg file ENTRY");

  gnome_dialog_editable_enters (GNOME_DIALOG (file_dialog),
    GTK_EDITABLE (gnome_file_entry_gtk_entry (GNOME_FILE_ENTRY (file_entry))));
  gnome_dialog_set_default (GNOME_DIALOG (file_dialog), GNOME_OK);

  gtk_box_pack_start (GTK_BOX (GNOME_DIALOG (file_dialog)->vbox),
                      file_entry, TRUE, TRUE, GNOME_PAD);

  gtk_widget_show_all (file_dialog);

  ans = gnome_dialog_run (GNOME_DIALOG (file_dialog));

  if (ans == GNOME_OK) {
    filename = gnome_file_entry_get_full_path (GNOME_FILE_ENTRY (file_entry),
                                               TRUE);
    if (filename != NULL) {
      if (GTK_IS_WIDGET (image)) {
        gtk_widget_destroy (GTK_WIDGET (image));
      }

      image = gnome_pixmap_new_from_file (filename);

      gtk_scrolled_window_add_with_viewport (GTK_SCROLLED_WINDOW (scrolly),
                                             GTK_WIDGET (image));

      gtk_widget_show (image);
    }
    else {
      invalid_filename = gnome_error_dialog ("File does not exist!");
      ans = gnome_dialog_run (GNOME_DIALOG (invalid_filename));
    }
  }

} /* open_callback */


close_callback(GtkWidget *button, gpointer data) {

  if (GTK_IS_WIDGET (image)) {
    gtk_widget_destroy (GTK_WIDGET (image));
  }

} /* close_callback */
```

```
about_callback(GtkWidget *button, gpointer data) {
  GtkWidget* dialog;
  int ans;

  dialog = gnome_ok_dialog ("created by Brett");

  /* keep user from doing other things */
  ans = gnome_dialog_run (GNOME_DIALOG (dialog));

} /* about_callback */

GnomeUIInfo file_menu[] = {
  GNOMEUIINFO_MENU_OPEN_ITEM (open_callback, NULL),
  GNOMEUIINFO_MENU_CLOSE_ITEM (close_callback, NULL),
  GNOMEUIINFO_SEPARATOR,
  GNOMEUIINFO_MENU_EXIT_ITEM (gtk_main_quit, NULL),
  GNOMEUIINFO_END
};

GnomeUIInfo help_menu[] = {
  GNOMEUIINFO_ITEM_NONE("About","About the application",
                        about_callback),
  GNOMEUIINFO_END
};

GnomeUIInfo menubar[] = {
  GNOMEUIINFO_MENU_FILE_TREE(file_menu),
  GNOMEUIINFO_SUBTREE("Help",help_menu),
  GNOMEUIINFO_END
};

/* end of menu */

int main (int argc, char** argv)
{
  GtkWidget* top_window;

  GtkWidget* menu;
  GtkWidget* menu_open;
  GtkWidget* menu_close;

//  GtkWidget* image;


  gnome_init ("gnomewin", "1.0", argc, argv);

  top_window = gnome_app_new ("image_disp", "Image Displayer");
  gtk_widget_set_usize (top_window, 500, 250);
```

```
  gtk_signal_connect (GTK_OBJECT (top_window), "delete-event",
                      GTK_SIGNAL_FUNC (close_window), NULL);


  gnome_app_create_menus (GNOME_APP (top_window), menubar);


  /* add scroll pane for image */
  scrolly = gtk_scrolled_window_new (NULL, NULL);


  gnome_app_set_contents (GNOME_APP (top_window), GTK_WIDGET (scrolly));
  gtk_widget_show_all (top_window);

  gtk_main ();
  exit (0);
}


////////////////////  file: Makefile  ////////////////////

FILE    = image_disp

CC      = gcc
LDLIBS  = `gnome-config --libs gnomeui`
CFLAGS  = `gnome-config --cflags gnomeui`

gnomewin: $(FILE).o
        $(CC) $(LDLIBS) $(FILE).o -o $(FILE)

gnomewin.o: $(FILE).c
        $(CC) $(CFLAGS) -c $(FILE).c

clean:
        rm -f $(FILE)
        rm -f $(FILE).o

////////////////////// END of image_disp  ////////////////////////
```

PROBLEM 7:

Write a Java applet program that allows the viewer at a client machine to see a "picture postcard" that will display an image and show a brief message, both as specifi ed in the HTML fi le. More specifi cally, the applet should do the following

1. The applet will display an image retrieved from a URL that is specified in the HTML file.

2. The applet will draw a rectangular box whose width, height, and location will be specified by appropriate parameters in the HTML file.

3. The applet will display a greetings message inside the box. The message will also be supplied as a string in the HTML file.

Your applet should work with an HTML file similar to the following

```
<HTML>
<TITLE>
A Flexible Applet
</TITLE>
<BODY BGCOLOR="#000000">
<APPLET CODE="FlexiApplet.class"  WIDTH=650 HEIGHT=400>
<PARAM NAME=BOX_WIDTH VALUE="170">
<PARAM NAME=BOX_HEIGHT VALUE=50>
<PARAM NAME=BOX_X VALUE=80>
<PARAM NAME=BOX_Y VALUE=220>
<PARAM NAME=MESSAGE VALUE="Greetings from Purdue">
<PARAM NAME=IMAGE_URL VALUE="http://rvl2.ecn.purdue.edu/~ack/PurduePortrait.jpg">
</APPLET>
</BODY>
</HTML>
```

Solution:

In the following solution, note how the image is acquired from the URL specified in the HTML file.

```
    AppletContext context = getAppletContext();              // (A)

    imageURL = getParameter( "IMAGE_URL" );                  // (B)
    if ( imageURL == null ) {                                // (C)
      noImageComment = "Image not specified in html file";   // (D)
      context.showStatus( noImageComment );                  // (E)
    }
    else {
      try {
        URL url = new URL( imageURL );                       // (F)
        image = context.getImage( url );                     // (G)
        imageWidth = image.getWidth( this );                 // (H)
        imageHeight = image.getHeight( this );               // (I)
      }
      catch( MalformedURLException mal ) {}
```

`AppletContext` is an interface in the `java.applet` package whose methods can be used by an applet to interact with its browser environment at run time. We need to acquire the `AppletContext` primarily because of its method `getImage()` for retrieving the image from the specified URL. This we do in line (G). But before getting to that line, we also check whether or not the user specified a URL for the image in line (C). If no URL was specified, we invoke the `showStatus()` method of the `AppletContext` to report that fact in the status bar at the bottom the browser.[1] In line (F), we then form the URL object from the URL string supplied in the HTML file. The resulting network connection if failed will result in the applet not starting. The `getImage()` method in line (G) returns immediately whether or not the image was actually there. As the image data is made available by this method, it is drawn incrementally on the screen by the `drawImage()` method call in the code for `paint()` shown in the program below. The syntax of the lines (H) and (I) should be clear from our earlier use of those methods. Recall that the argument to `getWidth()` and `getHeight()` is an `ImageObserver` object and an `Applet` object is an `ImageObserver` object because the `ImageObserver` interface is implemented by `Container`, an indirect parent of `Applet`. Here is the source code for the program:

```
// filename: FlexiApplet.html
/*
<HTML>
<TITLE>
A Flexible Applet
</TITLE>
<BODY BGCOLOR="#000000">
<APPLET CODE="FlexiApplet.class"  WIDTH=650 HEIGHT=400>
<PARAM NAME=BOX_WIDTH VALUE="170">
<PARAM NAME=BOX_HEIGHT VALUE=50>
<PARAM NAME=BOX_X VALUE=80>
<PARAM NAME=BOX_Y VALUE=220>
<PARAM NAME=MESSAGE VALUE="Greetings from Purdue">
<PARAM NAME=IMAGE_URL VALUE="http://rvl2.ecn.purdue.edu/~ack/PurduePortrait.jpg">
</APPLET>
</BODY>
</HTML>
*/

// filename: FlexiApplet.java

import java.applet.*;
import java.awt.*;
import java.net.*;

public class FlexiApplet extends Applet {
```

[1] A message thus shown usually gets quickly overwritten by other routine messages, such as *Applet Running*, the user may or may not get to see it.

```
int boxWidth;
int boxHeight;
int box_X;
int box_Y;
String messageString;
int imageWidth;
int imageHeight;
String imageURL;
String noImageComment;
Image image;

public void init() {
  String boxWidthString = getParameter( "BOX_WIDTH" );
  if ( boxWidthString != null )
    boxWidth = Integer.parseInt( boxWidthString );
  else boxWidth = 200;

  String boxHeightString = getParameter( "BOX_HEIGHT" );
  if ( boxHeightString != null )
    boxHeight = Integer.parseInt( boxHeightString );
  else boxHeight = 100;

  String boxX_coordString = getParameter( "BOX_X" );
  if ( boxX_coordString != null )
    box_X = Integer.parseInt( boxX_coordString );
  else box_X = 100;

  String boxY_coordString = getParameter( "BOX_Y" );
  if ( boxY_coordString != null )
    box_Y = Integer.parseInt( boxY_coordString );
  else box_Y = 300;

  messageString = getParameter( "MESSAGE" );
  if ( messageString == null )
    messageString = "Greetings";

  AppletContext context = getAppletContext();

  imageURL = getParameter( "IMAGE_URL" );
  if ( imageURL == null ) {
    noImageComment = "Image not specified in html file";
    context.showStatus( noImageComment );
  }
  else {
    try {
      URL url = new URL( imageURL );
      image = context.getImage( url );
      imageWidth = image.getWidth( this );
      imageHeight = image.getHeight( this );
```

```
      }
      catch( MalformedURLException mal ) {}
    }
  }

  public void paint( Graphics g ) {
    g.drawRect( box_X, box_Y, boxWidth, boxHeight );
    Font f = new Font( "SansSerif", Font.ITALIC, 14 );
    g.setFont( f );
    g.drawString( messageString, box_X + 10, box_Y + 30 );
    g.drawImage( image, 50, 50, imageWidth, imageHeight, this );
  }
}
```

PROBLEM 8:

Write an applet that allows a remote user to draw a free-form sketch by mouse clicking on different points. It should also be possible to run the program as an application with command-line invocation.

Solution:

```
/** SketchApplet.java **/
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;


public class SketchApplet extends JApplet {

  Polygon polygon = new Polygon();
  int clickCount;
  Point point;

  public void init() {
    addMouseListener( new MouseAdapter() {
          public void mouseClicked( MouseEvent e )
          {
            clickCount = e.getClickCount();
            point = e.getPoint();
            if ( clickCount == 1 ) {
              polygon.addPoint( point.x, point.y );
              clickCount = 0;
            }
```

```
            if ( clickCount == 2 ) {
              repaint();
              clickCount = 0;
            }
          }
    });
  }

  public void paint( Graphics g ) {
    g.drawPolyline( polygon.xpoints, polygon.ypoints, polygon.npoints );
  }

  // Needed to run the applet as a stand-alone application:
  public SketchApplet() {
    init();
  }

  // Needed to run the applet as a stand-alone application:
  public static void main(String[] args) {
    SketchFrame skf = new SketchFrame();
    skf.setSize( 400, 400 );
    skf.setLocation( 200, 200 );
    skf.show();
  }
}

// Needed to run the applet as a stand-alone application:
class SketchFrame extends JFrame {
  SketchFrame()
  {
    addWindowListener( new WindowAdapter() {
          public void windowClosing( WindowEvent e )
          { System.exit( 0 ) ; }
    });

    getContentPane().add( "Center", new SketchApplet() );
  }
}
```

Note how the `main()` for the `SketchApplet` applet constructs an object of type `SketchFrame` that is defined separately in the same file. The `SketchFrame` simply extends the `JFrame` class and its constructor merely adds the `Panel` object that the applet `SketchApplet` is at the center of the frame. The new class may now be executed by any of the following three methods:

- remotely via a web browser
- locally through the appletviewer
- by invoking the command line interpreter: *java SketchApplet*

18

Multithreaded Object-Oriented Programming

PROBLEM 1:

Write a clock program that displays an analog clock in which the second hand is updated every one second and the minute and the hour hands accordingly. Implement the timer part of the program as a thread that wakes up after every one second and gets the new time from the system. [This thread can be a *daemon thread*. A daemon thread runs in the background and the termination of a program is not predicated upon the daemon thread running to completion. The garbage collection thread in Java is a daemon thread.] For this exercise, you could declare your timer thread to be a daemon thread by

```
Clock c = new Clock();
Thread t = new Thread( c );
t.setDaemon( true );
```

assuming that the `Clock` object `c` is of type `Runnable`.

Solution:

```
//filename: Clock.java

import java.awt.*;
import java.util.*;

public class Clock extends Frame implements Runnable {
```

```
  private int seconds;

  public Clock() {
    setSize(200, 200);
    setTitle("JavaClock");
  }

  public void paint( Graphics g ) {
    g.drawOval( 30, 40, 144, 144 );

    double hourAngle = 2*Math.PI*( seconds - 3*60*60 ) / (12*60*60 );
    double minuteAngle = 2*Math.PI*( seconds - 15*60 )/(60*60);
    double secondAngle = 2*Math.PI*( seconds - 15 ) / 60;

    g.drawLine( 102, 112, 102 + (int)(40*Math.cos( hourAngle ) ),
                          112 + (int)(40*Math.sin( hourAngle ) ) );
    g.drawLine( 102, 112, 102 + (int)(55*Math.cos( minuteAngle ) ),
                          112 + (int)(55*Math.sin( minuteAngle ) ) );
    g.drawLine( 102, 112, 102 + (int)(70*Math.cos( secondAngle ) ),
                          112 + (int)(70*Math.sin( secondAngle ) ) );
  }

  public void run() {
    while ( true ) {
      try {
        Thread.sleep( 1000 );
      } catch( InterruptedException e ) {}

      GregorianCalendar d = new GregorianCalendar();
      seconds =     d.get(Calendar.HOUR) * 60 * 60
                  + d.get(Calendar.MINUTE) * 60
                  + d.get(Calendar.SECOND);
      repaint();
    }
  }

  public static void main( String[] args ) {
    Clock c = new Clock();
    Thread t = new Thread( c );
    t.setDaemon( true );
    c.show();
    t.start();
  }
}
```

PROBLEM 2:

Write a multithreaded C++ program that solves the famous Dining Philosopher's Problem. Your program should demonstrate how both thread deadlock and starvation can be avoided by using mutex locks in conjunction with *condition variables*. The statement of the dining philosopher's problem is:

> Five philosophers are sitting around a round table, with one chopstick between each pair of philosophers. Each philosopher alternates between two states, THINKING and EATING, both for random periods of time. Suppose we represent each philosopher with one thread, the goal in solving this problem is to come with a scheduling strategy so that all philosophers get to eat for roughly the same time.

Obviousely, only two philosophers can eat at one time. Moreover, a philosopher can eat only if both of his/her two neighbors are not eating. Additionally, if it should happen that each philosopher has picked up one chopstick and is waiting for the other chopstick to be put down by the neighbor, we will have a deadlock.

To see how we could get into a starvation situation for one of the philosophers, while only two philosophers can eat at any given time, it cannot be two adjacnt philosphers. Let's say the philosophers are indexed 0, 1, 2, 3, and 4. Let's also say that 0 and 2 are eating. Let's assume that after 0 and 2 are done, 1 and 3 switch into eating state. Now if it should happend that after 1 and 3 are done eating, 0 and 2 should resume eating, and so on back to 1 and 3, the philosopher 4 would end up starving.

Solution:

Shown here is a C solution (using POSIX threads) to the problem. Have yet to create a C++ solution.

Shown here is a solution to the dinining philosopher's problem that will not deadlock and will not let any philosopher starve. The heart of the solution lies in the following scheduler that uses a condition variable for each philosopher:

```
typedef struct {
  pthread_mutex_t *mon;
  pthread_cond_t *cv[MAXTHREADS];
  int state[MAXTHREADS];
} Scheduler;
```

where `MAXTHREADS` is equal to the number of philosophers, in our example 5, and where `state[n]` is the state of the philosopher indexed n. As mentioned before, a philosopher can either be in `EATING` state or `THINKING` state. So the array `state[MAXTHREADS]` keep track of the state of all the philosophers.

The solution shown below, by Planck and Wolski, solves the problem by using the following strategy:

1. Each philospher whose THINKING time has come to an end who now wants to start EATING, but who finds unable to do so because at least one of his neighbors is eating, will be placed in a Queue. Recall, a queue works on a first-in-first-out basis.
2. If a philosopher ready to switch into the EATING state finds that he is able to do so will nonetheless be placed in the Queue if the Queue is not empty. In other words, a philosopher can starting EATING immediately after the end of the THINKING state only if he is able to (because neither neighbor is eating) and if the Queue is empty.
3. Every time a philospher stops EATING, we check the philosopher at the head of the Queue to see if neither of his neighbors are eating. If neither neighbor is eating, the condition variable on which this philospher (the one at the head of the Queue) is waiting is signaled.

Using the `Queue` data structure ensures that no philosopher will starve because the philosopher who has been waiting the longest in the `Queue` is guaranteed to get out first. Note that as long as the `Queue` is non-empty, even if a philosopher is able to eat because neither of his neigbors is eating, he is still placed at the tail of the Queue. In this solution, a deadlock is avoided by testing under a mutex lock whether or not a philosopher can transition to EATING by checking the states of the two neighbors. In contrast, consider a solution in which each chopstick is represented separately and the different philosopher threads can cut in on one another during the computations needed for acquiring both chopsticks. With that solution, there would be a possibility that each philosopher would take possession of, say, the left chopstick and wait indefinitely for the right chopstick to materialize, creating instant deadlock.

In the solution below, the overall behavior of a philosopher is orchestrated by the function `philospher()`. This is the thread function, meaning the function that is supplied to each philospher thread. This function cycles indefinitely through the two permissible states, THINKING and EATING, putting each thread to sleep for a random interval in each state. When the time allocated for THINKING is over, the function `pickup()` is called. This function represents conceptually the process of acquiring both chopsticks. After the random time allocated for EATING is over, the `philosopher()` function calls `putdown()`. This function represents conceptually the act of putting down both chopsticks. Of course, at the completion of the `putdown()` function, the state of the philosopher reverts back to THINKING.

The job of `main()` is to initialize the five philosophers, to launch the five threads corresponding to the philosophers, and to then print out every 10 seconds a string that shows total blocked time for all five threads and the blocked time for each thread. The *blocked time* is defined as the time interval between the moment a philosopher requests `pickup()` and the moment the philosopher is actually allowed to change its state from THINKING to EATING.

```
// avi_2.c
// Based entirely on dphil_6.c by Planck and Wolski.

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>


#define MAXTHREADS 5
```

```
#define THINKING 0
#define EATING 1
#define MAX_THINK_EAT_TIME 10       // to be used unless user specified
                                    // by command line input

typedef struct {
    int id;             /* The philosopher's id: 0 to 4 */
    long t0;            /* The time when the program started */
    long t;             /* The current time  */
    int* blocktime;     /* Array of ints for blocked times for each philso */
    pthread_mutex_t* blockmon;    /* The blocking monitor */
} Phil_struct;


// node needed for the global Queue

typedef struct node {
    int value;
    struct node* next;
} node;


// When a philosopher is done thinking, he wants to eat provided he can
// get hold of the two chopsticks, which is the same thing as saying
// that provided his two neighbors are not in EATING state.  If at the
// end of THINKING time, a philosopher cannot transition into EATING
// state becaue of the neighbors is in EATING state, the philosopher is
// put into the Queue.

// After the Queue has at least one entry, whenver a philosopher is done
// thinking, he is automatically put at the back end of the Queue even
// if his two neighbors are both not eating.

node* Queue = NULL;


// The Scheduler's job is to keep track of the state of each philosopher
// and to be the keeper of the condition variable for each philospher.
// During the pickup() operation, a philosopher uses the condition
// variable to temporarily suspend its lock for the pick thread of
// execution if it finds that it cannot transition into the EATING
// state because either the Queue is NOT empty or because one of his
// neighbors is eating.  The lock is later restored when the condition
// variable is signalled by the philosopher being at the head of the
// queue and being able to transition to the EATING state.

typedef struct {
  pthread_mutex_t *mon;
```

```
  pthread_cond_t *cv[MAXTHREADS];
  int state[MAXTHREADS];
} Scheduler;

// The scheduler is initialized by initialize_schedule.  Initialization
// consists of starting all philosophers in THINKING state and
// the memory allocation and initialization for the monitor and
// for each of the condition variables.
void* initialize_scheduler();

void pickup( Phil_struct* );
void putdown( Phil_struct* );

// The test_queue() function is called at the end of the pickup()
// operation after the lock is restored by condition variable signalling.
// This also causes the philosopher to be taken off the Queue and the
// state of the philosopher to be change to EATING.
// This function is also called at the end of putdown() operation
// when the state of the philosopher changes to THINKING.
void test_queue();


// This is the thread execution function and its signature must be
// as shown.  This function invokes orchestrates the behavior of
// each philosopher, in the sense that a philosopher is first made
// to think for a random number of seconds.  Then the pickup() operation
// is invoked to get the state changed from THINKING to EATING.
// This state change can get blocked if one or both neighbors of the
// philosopher are eating.
void* philosopher( void* v );


// Functions for the global queue:
int isQueueEmpty();
void addToQueue( int );
void deleteFromQueue();
void printQueue();


Scheduler* scheduler;

main( int argc, char** argv)
{
  int i;
  pthread_t threads[MAXTHREADS];
  Phil_struct ps[MAXTHREADS];
  long t0;
  pthread_mutex_t* blockmon;    // blocking monitor
  int* blocktime;               // array of blocked times for philosophers
```

```
  char s[500];                  // For composing an output message
  char* curr;
  int total;

  srandom(time(0));
  initialize_scheduler();
  t0 = time(0);

  blocktime = (int*) malloc( sizeof(int) * MAXTHREADS );

  blockmon = (pthread_mutex_t*) malloc(sizeof(pthread_mutex_t));

  pthread_mutex_init( blockmon, NULL );   // use default attributes

  for (i = 0; i < MAXTHREADS; i++) blocktime[i] = 0;

  for (i = 0; i < MAXTHREADS; i++) {
    ps[i].id = i;
    ps[i].t0 = t0;
    ps[i].blocktime = blocktime;  //pointer to the array of blocked times
    ps[i].blockmon = blockmon;    //pointer to the blocking monitor
    pthread_create(threads+i, NULL, philosopher, (void *) (ps+i));
  }

  while(1) {
    pthread_mutex_lock(blockmon);
    curr = s;                     // curr will point initially to a block
                                  // of allocated  memory for 500 chars
    total=0;
    for(i=0; i < MAXTHREADS; i++)
            total += blocktime[i];
    sprintf(curr,"%3d Total blocktime: %5d : ",
                   time(0)-t0,
                   total);
    curr = s + strlen(s);         // In the 500 char allocated array,
                                  // curr will now point to just after
                                  // where the above string ended
    for(i=0; i < MAXTHREADS; i++) {
        sprintf(curr, "%5d ", blocktime[i]);
        curr = s + strlen(s);
    }
    pthread_mutex_unlock(blockmon);
    printf("%s\n", s);            // output the message composed in s
    fflush(stdout);
    sleep(10);
  }
}
```

```
void* philosopher( void* v ) {  // must be the signature of a thread function
  Phil_struct* ps;
  long st;
  long t;

  ps = (Phil_struct*) v;
  while(1) {                    // run the loop indefinitely

    /* First the philosopher thinks for a random number of seconds */
    st = ( random()%MAX_THINK_EAT_TIME ) + 1;
    printf("%3d Philosopher %d thinking for %d second%s\n",
                time(0)-ps->t0, ps->id, st, (st == 1) ? "" : "s");
    fflush(stdout);
    sleep(st);

    /* Now, the philosopher wakes up and wants to eat.  He calls pickup
       to pick up the chopsticks */

    printf("%3d Philosopher %d no longer thinking -- calling pickup()\n",
            time(0)-ps->t0, ps->id);
    fflush(stdout);
    t = time(0);
    pickup(ps);
    pthread_mutex_lock( ps->blockmon );
    ps->blocktime[ps->id] += (time(0) - t);
    pthread_mutex_unlock(ps->blockmon);

    /* When pickup returns, the philosopher can eat for a random number of
       seconds */

    st = ( random()%MAX_THINK_EAT_TIME ) + 1;
    printf("%3d Philosopher %d eating for %d second%s\n",
                time(0)-ps->t0, ps->id, st, (st == 1) ? "" : "s");
    fflush(stdout);
    sleep(st);

    /* Finally, the philosopher is done eating, and calls putdown to
       put down the chopsticks */

    printf("%3d Philosopher %d no longer eating -- calling putdown()\n",
            time(0)-ps->t0, ps->id);
    fflush(stdout);
    putdown(ps);
  }
}


//void test_queue(Scheduler *pp)
```

```
void test_queue()
{
  int id;

  if ( !isQueueEmpty( ) ) {
    id = Queue->value;
    if ( scheduler->state[(id+1)%MAXTHREADS] != EATING &&
            scheduler->state[(id+(MAXTHREADS-1))%MAXTHREADS] != EATING) {
      pthread_cond_signal( scheduler->cv[id] );
    }
  }
}

/* If the queue is empty and your neighbors are not eating, then
   go ahead and eat.  Otherwise, put yourself on the queue and
   wait */

void pickup(Phil_struct *ps) {
  pthread_mutex_lock( scheduler->mon );
  if ( isQueueEmpty( ) &&
//                pp->state[(ps->id+(phil_count-1))%phil_count] != EATING &&
          scheduler->state[(ps->id+1)%MAXTHREADS] != EATING &&
             scheduler->state[(ps->id+(MAXTHREADS-1))%MAXTHREADS] != EATING) {
    scheduler->state[ps->id] = EATING;
  } else {
    addToQueue( ps->id);
    pthread_cond_wait( scheduler->cv[ps->id], scheduler->mon );
    deleteFromQueue( );
    scheduler->state[ps->id] = EATING;

    test_queue();
  }
  pthread_mutex_unlock(scheduler->mon);
}

/* Set state to thinking and then signal the first person on the
   queue if they can eat */

void putdown(Phil_struct *ps) {
  pthread_mutex_lock( scheduler->mon );
  scheduler->state[ps->id] = THINKING;
  test_queue();
  pthread_mutex_unlock(scheduler->mon);
}

void* initialize_scheduler() {
  int i;
  scheduler = malloc( sizeof(Scheduler) );
  scheduler->mon =  malloc(sizeof(pthread_mutex_t));
```

```
  pthread_mutex_init( scheduler->mon, NULL);
  for (i = 0; i < MAXTHREADS; i++) {
    scheduler->cv[i] = malloc(sizeof(pthread_cond_t));
    pthread_cond_init( scheduler->cv[i], NULL );
    scheduler->state[i] = THINKING;
  }
}

int isQueueEmpty() {
    return Queue == NULL ? 1 : 0;
}

void addToQueue( int newValue ) {
  node* local = Queue;
  node* newNode;
  printf( "<><>Appending to queue %d<><>\n", newValue );
  newNode = malloc( sizeof( node ) );
  newNode->next = NULL;
  newNode->value = newValue;

  if ( Queue == NULL ) {
    Queue = newNode;
    printQueue();                    //<<<
    return;
  }
  while ( local->next != NULL )
      local = local->next;
  local->next = newNode;
  printQueue( );                      // <<<
}

void deleteFromQueue() {
  node* first = Queue;
  printf( "<**> deleting %d <**>\n", first->value );
  Queue = Queue->next;
  free( first );
}

void printQueue( ) {
    node* local = Queue;
    printf( "PRINTING QUEUE: " );
    while( local != NULL ) {
        printf( "%d  ", local->value );
        local = (node*) local->next;
    }
    printf( "\n" );
}
```

19

Network Programming

PROBLEM 1:

Write an implementation of the `ChatServer` class of Section 19.2 that uses a *nonstatic* version of the data member `clientVec`. You may do this by making the following changes to Java program of Section 19.2:

- In the `main` of the `ChatServer` class, pass the server object to the `ClientHandler` constructor by

```
ClientHandler clh = new ClientHandler(socket, this);
```

- Add the following additional data member to the `ClientHandler` class

```
ChatServer chat_server;
```

- Change the `ClientHandler` constructor so that it accepts the two arguments shown below:

```
public ClientHandler( Socket s, ChatServer c ) {
    try {
        sock = s;
        chat_server = c;
        // .....
```

- With all of the above changes, the `for` loop in the `run()` method of `ClientHandler` can be changed to:

```
for (int i = 0; i < chat_server.clientVec.size(); i++) {
  ClientHandler cl =
```

```
            (ClientHandler) chat_server.clientVec.elementAt(i);
        // ......
```

Solution:

The solution is contained in the implementation suggestions listed above.

PROBLEM 2:

Write an applet program that elicits a piece of information from a client viewing the applet and sends it back to a server program running at the host that is the home of the applet. This homework is also an exercise in launchig a `JFrame` object from an applet. Your frame should consist of a textfi eld and a button. When a client presses the button, the information typed into the textfi eld should get transmitted back to the server.

Solution:

```
<!-- filename: FetchInfo.html -->
<HTML>
<TITLE>FetchInfoApplet</TITLE>
<BODY>
<APPLET CODE="FetchInfoApplet.class" WIDTH=520 HEIGHT=420>
</APPLET>
</BODY>
</HTML>


//filename: FetchInfoApplet.java

//////////////   class FetchInfoApplet   /////////////

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.applet.*;

//NOTE:
//
//It is instructive to compare the commented out implementation with the
//working implementation.  The commented out implementation works only
//with the appletviewer tool.
//
```

```
//A basic problem with the commented out implementation is that I have a
//function call, getInfo(), inside the init() method.  While legal, this
//is not the best way to set up a graphical program in Java.  The role of
//init() in an applet program should be the same as that of a constructor
//in a graphical application, that is, it is to be used for bringing
//together all the AWT components that will create the visual look needed
//for the applet.
//
//The second major problem with the commented out implementation is the
//while loop in the init() method.  This is wrong on two scores: First, it
//is bad style because of what was said earlier about init().  But even
//more importantly, it does not pay sufficient respect to the fact that
//Java AWT is event driven.  So when an event takes place, you know about
//it and you can take appropriate action.  It should not also be necessary
//to use the sort of while loop that is used in the commented out init().
//
//My original reason for using the while loop was to introduce an indeterminate
//amout of wait until the user clicked the button.  The working implementation
//has a much neater solution that addresses this problem.  This solution
//puts the server-side readUTF() invocation inside a while loop.
//This obviously makes the applet code simpler.
//
//Other highlights of the implementation shown:
//
//a)  Note how we send the applet object down to the InfoWindow
//    object constructor.
//
//b)  Also note how we use the applet CODEBASE in the socket
//    constructor.
//

public class FetchInfoApplet extends Applet {

  public void init( ) {
    InfoWindow iw = new InfoWindow(this);
    iw.setSize( 300, 300 );
    iw.setVisible( true );
  }
}

// InfoWindow class

class InfoWindow extends Frame
    implements ActionListener {

  final private TextField tf;
  private Socket socket = null;
  private DataOutputStream output = null;
```

```
  public InfoWindow(Applet appl) {

    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent we) {
        dispose();
        try {
          socket.close();
        }
        catch (Exception e) {e.printStackTrace();}
      }});

    tf = new TextField();
    add( tf, "North" );
    Button b = new Button( "enter name" );

    b.addActionListener(this);
    add( b,"South");

    try {
      System.out.println("Connecting to: " + appl.getCodeBase().getHost());
      Socket socket = new Socket(appl.getCodeBase().getHost(), 1789);
      output = new DataOutputStream(socket.getOutputStream());
    } catch(Exception e) { e.printStackTrace(); }
  }

  public void actionPerformed(ActionEvent evt) {
    String info = tf.getText();
    try {
      output.writeUTF(info);
    }
    catch (IOException e) {
        e.printStackTrace();
    }
  }

}


//  public class FetchInfoApplet extends Applet {
//
//    private String infoString;
//
//    public void init( )
//    {
//      InfoWindow iw = new InfoWindow();
//      iw.setSize( 300, 300 );
//      iw.setVisible( true );
//      infoString = iw.getInfo();
//      while ( infoString == null ) {
```

```
//        try {
//          Thread.sleep( 5 );
//        } catch( Exception e ) {}
//      }
//    }
//
//    public void start()
//    {
//      try {
//        Socket socket = new Socket( "localhost", 1000 );
//        DataOutputStream output = new DataOutputStream( socket.getOutputStream() );
//        output.writeUTF( infoString );
//      } catch(IOException e) {}
//    }
//  }
//
//  class InfoWindow extends Frame {
//    private String info;
//
//    public InfoWindow()
//    {
//      final TextField tf = new TextField();
//      add( tf, "North" );
//      Button b = new Button( "enter name" );
//      b.addActionListener(new ActionListener() {
//        public void actionPerformed(ActionEvent evt) {
//          info = tf.getText();
//        }
//      });
//      add( b,"South");
//    }
//
//    public String getInfo() { return info; }
//  }
//
//


//File: FetchInfoServer.java

///////////////  class FetchInfoServer.java  ////////////

import java.awt.*;
import java.net.*;
import java.io.*;

public class FetchInfoServer {

  public static void main( String[] args )
```

```
  {
    try {
      ServerSocket server = new ServerSocket( 1789 );
      System.out.println("Server awaiting connections\n");
      Socket socket = server.accept();
      DataInputStream input = new DataInputStream( socket.getInputStream() );
      System.out.println( "connection made ........" );
      for (;;) {
        String fetchedString = input.readUTF();
        System.out.println( fetchedString );
      }
    } catch( IOException e ) { e.printStackTrace(); }
  }
}

//  public class FetchInfoServer {
//
//    public static void main( String[] args )
//    {
//      try {
//        ServerSocket server = new ServerSocket( 1000 );
//        System.out.println("Server awaiting connections\n");
//        for (;;) {
//          Socket socket = server.accept();
//          System.out.println( "connection made ........" );
//          DataInputStream input = new DataInputStream( socket.getInputStream() );
//          String fetchedString = input.readUTF();
//          System.out.println( fetchedString );
//        }
//      } catch( IOException e ) {}
//    }
//  }
//
```

PROBLEM 3:

Write an applet version of the chat program shown in Section 19.2. By pointing the browser to the HTML file referring to the applet, the applet should first elicit the chat name of the client and then show two separate windows, one for showing all of the chat so far and the other for typing in a new submission.

Solution:

```
<!-- filename: EE462Chat.html -->
<HTML>
```

```
<HEAD>
<TITLE>EE462 Chat Room</TITLE>
</HEAD>
<BODY>
<APPLET CODE="EE462ChatClient" WIDTH=450 HEIGHT=500></APPLET>
</BODY>
</HTML>

//file: EE462ChatClient.java

//////////////  class EE462ChatClient.java  ////////////

import java.awt.*;
import java.io.*;
import java.awt.event.*;
import java.util.*;
import java.applet.*;
import java.net.*;

public class EE462ChatClient extends Applet implements Runnable{

  private TextArea chatArea;
  private Button chatButton;
  private Frame submitFrame;
  private Button submit;
  private TextArea messageArea;
  private Label label;

  private Socket sock;
  private PrintWriter out;
  private BufferedReader in;
  private Thread textServer;

  public void init() {
   chatArea = new TextArea("", 30, 10, TextArea.SCROLLBARS_VERTICAL_ONLY);
   chatArea.setEditable(false);
   chatButton = new Button("Chat!");

   setLayout(new BorderLayout());

   add(chatArea, "North");
   add(chatButton, "South");

   submitFrame = new Frame("Welcome to EE462's Chatroom!");
   submitFrame.setLayout(new BorderLayout());

   label = new Label("Type your name and press Submit", Label.LEFT);
   messageArea = new TextArea("", 5, 10, TextArea.SCROLLBARS_VERTICAL_ONLY);
   messageArea.setEditable(true);
```

```
   submit = new Button("Submit");

   submitFrame.add(label, "North");
   submitFrame.add(messageArea, "Center");
   submitFrame.add(submit, "South");

   chatButton.addActionListener(new ActionListener()
   {
    public void actionPerformed(ActionEvent evt)
    {
     submitFrame.setSize(400, 300);
     submitFrame.setVisible(true);
    }
   });

   submit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
     label.setText("Type \"logoff\" to end session, press Submit to send"
                   + " message");
     String str = messageArea.getText();
     messageArea.setText("");
     out.println(str);
     out.flush();
    }
   });

   submitFrame.addWindowListener(new WindowAdapter() {
                                  public void windowClosing(WindowEvent e)
                                  {
                                   submitFrame.setVisible(false);
                                  }
                                 }
                                );
  }

  public void start() {
   try{
//     sock = new Socket("horizon.ecn.purdue.edu", 3408);
     sock = new Socket("localhost", 3408);
     out = new PrintWriter(sock.getOutputStream());
     in = new BufferedReader(new InputStreamReader(sock.getInputStream()));
    }catch(Exception e) {}
   textServer = new Thread(this);
   textServer.start();
  }

  public void run() {
   while (true) {
    try{
```

```
      String input = in.readLine();
      chatArea.append(input + "\n");
     }catch(Exception e) {}
   }
  }
}


//file: EE462ChatServer.java

//////////////   class EE462ChatServer  //////////////

import java.io.*;
import java.util.*;
import java.net.*;

public class EE462ChatServer{

  public Vector clients;
  public static EE462ChatServer chatServer;

  public EE462ChatServer() {
   clients = new Vector();
  }

  public void run() {
   try{
     ServerSocket server = new ServerSocket(3408);
     System.out.println("ChatServer running...\n");

     while (true) {
      Socket s = server.accept();
      ClientThread ct = new ClientThread(s);
      clients.addElement(ct);
      ct.start();
     }
    }catch(Exception e) {}
  }

  public static void main(String[] args) {
   EE462ChatServer.chatServer = new EE462ChatServer();
   EE462ChatServer.chatServer.run();
  }
}

class ClientThread extends Thread{
  private Socket sock;
  private String userName;
  private static Vector chatStore = new Vector();
```

```
private static Vector names = new Vector();
private BufferedReader in = null;
private PrintWriter out = null;

public ClientThread(Socket s) {
 sock = s;

 try{
   out = new PrintWriter(sock.getOutputStream());

   in = new BufferedReader(new InputStreamReader(sock.getInputStream()));

   userName = in.readLine();
   System.out.println(userName + " logged in");
   ClientThread.names.addElement(userName);

  }catch(Exception e) {}
}

public void run() {
 try{
   boolean done = false;
   String strWithName = "";
   for (int i=0; i<EE462ChatServer.chatServer.clients.size(); i++) {
    ClientThread ct = (ClientThread)EE462ChatServer.chatServer.clients.elementAt(i);
    ct.out.println(userName + " signed in");
    ct.out.flush();
   }
   if (ClientThread.names.size() != 0) {
     out.println("\nCurrent Chatters:\n");
     for (int i=0; i<ClientThread.names.size(); i++)
       out.println((String)ClientThread.names.elementAt(i));
     out.println("\n");
     out.flush();
   }

   if (ClientThread.chatStore.size() != 0) {
     out.println("\nChat History:\n");
     for (int i=0; i<ClientThread.chatStore.size(); i++)
       out.println((String)ClientThread.chatStore.elementAt(i));
     out.println("\n");
     out.flush();
   }
   while (!done) {
     String str = in.readLine();
     if (str.equals("logoff")) {
       strWithName = userName + " signed off";
       done = true;
     } else
```

```
        strWithName = userName + ": " + str;
       ClientThread.chatStore.addElement(strWithName);

       for (int i=0; i<EE462ChatServer.chatServer.clients.size(); i++) {
         ClientThread ct = (ClientThread)EE462ChatServer.chatServer.clients.elementAt(i);
         ct.out.println(strWithName);
         ct.out.flush();
       }
     }
     System.out.println(userName + " logged off");

     in.close();
     out.close();
     sock.close();
    }catch(Exception e) {}
  }
}
```

20

Database Programming

PROBLEM 1:

Write a JDBC program that can serve as a front-end server to a MySQL database. Your JDBC server should perform the following tasks:

1. Establishe a password-protected connection between a client and the database. The client must specify his/her username, password, and the database name.
2. Report back any error messages to the client if there is any error while connecting with the database and allows the client to try to reconnect repeatedly until the client enters "exit" as the username.
3. Execute each SQL command received from the client and display the retrieved results to the client as necessary.
4. Display on the client's terminal an error message if the SQL command is invalid.
5. Continue receiving SQL commands until "exit" is entered by the user.

[HINT: The `ChatServer` program of Chapter 19 would be a good starting point for the program needed here.]

Solution:

```
//////////////////   file: Connect.java  ////////////////////
```

```
import java.sql.*;
import java.util.*;

public class Connect {

  private Connection con;
  private ResourceBundle rbConnect;
  private Statement stmt;

  private Hashtable errors;

  private String sDriver,
         sDriverKey="MySqlDriver",
         srbName="Connect",
         sURL,
   sURLKey="MySqlURL",
         sUserID,
         sPassword,
         sDatabaseName;


  public Connect( String username,String passwd,String dbName, Hashtable err ){
    sUserID = username;
    sPassword = passwd;
    sDatabaseName = dbName;
    errors = err;
  }

  public boolean startConnection() {
    try {
      rbConnect = ResourceBundle.getBundle( srbName );
      sDriver = rbConnect.getString( sDriverKey );
      sURL = rbConnect.getString( sURLKey );
    } catch ( MissingResourceException mre ) {
      errors.put( "connection","ResourceBundle problem for " +
         srbName + ", program ends.\n"  + "Specific error: " +
        mre.getMessage() );
      return false;
    }

    //load driver
    try {
      Class.forName( sDriver ).newInstance();
    } catch ( Exception e ) { //error
      errors.put( "connection", "Cannot load driver." );
      return false;
    }

    //connect to database
```

```
    try {
      con = DriverManager.getConnection( sURL+sDatabaseName,sUserID,sPassword );
      DatabaseMetaData dbmd = con.getMetaData();
      stmt = con.createStatement();
    } catch ( SQLException SQLe ) {
        close();
        errors.put("connection", "problems connecting to " +sURL + ":" + "/n" +
          SQLe.getMessage() );

        return false;
      }

    return true;
  }

  public void close() {
    if ( con!=null ) {
      try { con.close(); }
      catch ( Exception e ) {}
    }
  }

  public ResultSet executeQuery( String s ) {
    ResultSet rs = null;
    try {
      rs= stmt.executeQuery( s );
    } catch (Exception e) {}
    return rs;
  }
}


////////////////////// file: Connect.properties  ////////////////////

#PropertiesResourceBundle for database connection
MySqlDriver=org.gjt.mm.mysql.Driver
MySqlURL=jdbc:mysql:///


////////////////////// file: JDBCServer.java  ///////////////////////

import java.io.*;
import java.net.*;
import java.util.*;
import java.sql.*;

public class JDBCServer {

  public static void main( String[] args ) {
```

```
    try {
      ServerSocket server = new ServerSocket( 7800 );

      for(;;) {
        Socket socket= server.accept();
        System.out.println( "A new client checked in:  " );
        ClientHandler clh = new ClientHandler( socket );
        clh.startConnection();
        clh.start();
      }
    } catch ( Exception e ) { System.out.println( e ); }
  }
}

class ClientHandler extends Thread {

  private String username,password,dbname;
  private Socket sock;
  private BufferedReader buff_reader = null;
  private PrintWriter out = null;
  private Connect con;
  private boolean done = false;

  private Hashtable err = new Hashtable();

  public ClientHandler ( Socket s ) {
    try {
      sock = s;
      out = new PrintWriter( sock.getOutputStream() );
      InputStream in_stream = sock.getInputStream();
      InputStreamReader in_reader = new InputStreamReader( in_stream );
      buff_reader = new BufferedReader( in_reader );
    } catch ( Exception e ) {}
  }

  public void startConnection() {
    boolean success = false;
    out.println( "\n Welcome to JDBC Server" );

    try {
      while ( !success  && !done ) {

        //ask for username
        out.print( "\nEnter username: " );
        out.flush();
        username = buff_reader.readLine();

        if ( username.equals( "exit" ) ) {
          done = true;
```

```
          continue;
        }
        out.print( "\nEnter password: " );
        out.flush();
        password = buff_reader.readLine();

        out.print( "\nEnter database name: " );
        out.flush();
        dbname = buff_reader.readLine();

        out.print( "\n\n" );
        out.flush();

        System.out.println( "\n"+username + " tries to log in" );

        con = new Connect( username,password,dbname,err );
        success=con.startConnection();

        if ( success == false ) {
          out.print( (String)err.get("connection") );
        }


      }
    }catch ( Exception e ) {}
      System.out.println( username + " logging in" );
}

  public void run() {
    try {
      while ( !done ){
        out.print( "\nEnter query : " );
        out.flush();

        String str = buff_reader.readLine();

        if ( str.equals("exit") ) {
          done = true;
        }

        ResultSet rs = con.executeQuery( str );
        if ( rs==null ) { //query fails
          out.println( "Error in executing query" );
        }
        else {
          ResultSetMetaData rsmd = rs.getMetaData();
          int numcols = rsmd.getColumnCount();
          if ( numcols > 0 ) {
```

```
            //print col title
            for( int i=1;i<=numcols;i++ ) {
              if( i>1 ) out.print( "\t| " );
              out.print( rsmd.getColumnLabel( i ) );
            }
            out.println( "" );

            while ( rs.next() ) {
              for( int i=1;i<=numcols;i++ ) {
                if( i>1 ) out.print( "\t| " );
                out.print( rs.getString( i ) );
              }
              out.println( "" );
            }
          }
        }
      }
      System.out.println( "\n"+username +" logged off" );
      buff_reader.close();
      con.close();
      out.close();
      sock.close();
    }catch( Exception e ) {}
  }

}
```

PROBLEM 3:

Write a C++ implementation for the previous problem using Mysql++ classes. [Suggestion: As with the previous problem, the C++ implementation of the chat server program of Chapter 19 would be a good starting point for the solution needed here.]