

Programming Problems  
18<sup>th</sup> Annual Computer Science Programming Contest

Department of Mathematics and Computer Science  
Western Carolina University  
March 27<sup>th</sup>, 2007

## Criteria for Determining Scores

Each program should:

1. Execute without error, solve the appropriate problem efficiently and satisfy all the conditions specified in the problem statement.
2. Follow good programming style conventions such as avoiding confusing and unnecessary code.
3. Have good documentation and easily understandable variable names.
4. Start with an initial comment that includes:
  - a. The name of the team's school
  - b. The team's number
  - c. The names of the team's members

An example starting comment is:

*// Mytown High School, Team 3, Erik Johnson, Samantha Carter, and Derek Hart*

The score assigned each program will be based upon the extent that the program satisfies the properties listed above. The total accumulated score for all the programs represents the team score. All programs have the same weight in the scoring.

## Submitting Answers:

Send an email with your .java file included as an attachment to [contest@cs.wcu.edu](mailto:contest@cs.wcu.edu). IN the subject indicate the problem your are submitting (i.e., problem1) In the body of the message, include your school affiliation and the names of all team members.

## Administrative Notes

1. Each team may use two Java reference books and 3 pages of notes.
2. Each team may also look at the Java API on the web; however, no other site on the web may be used.
3. Each team may use up to **two** computers.
4. **Inappropriate code comments will result in a lower score and are grounds for disqualification for an award.**

## Programming Notes

### General

Where appropriate, you should do input verification. In other words if the user is supposed to input, for example, a positive number, an appropriate error message should be displayed when the user does not enter a positive number. The user should then get additional chances to enter input.

### Java

The version of Java installed in the electronic classrooms here is the Java SDK 1.5.

Output with Java 1.5 uses the standard form: `System.out.println` (or `System.out.print` as appropriate)

Input with Java 1.5 input can be handled by either of the following methods:

- `BufferedReader`
- `Scanner`

Use whichever method you are most experienced with.

### Input with `BufferedReader`:

- When using the `BufferedReader` make sure that the first statement you include in your Java program is the statement:

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

- Before doing any input from the keyboard you will need to declare a `BufferedReader` object that is tied to `System.in`. (In the following example the `BufferedReader` object is named "br")

```
BufferedReader br = new BufferedReader (new  
InputStreamReader (System.in));
```

- Then you may use this object to read Strings (one line at a time) from the keyboard:

```
System.out.println("Please enter a line of input> ");  
String line = br.readLine();  
System.out.println("Please enter another line> ");  
String line2 = br.readLine();
```

The above example prompts the user for input and reads two lines of input from the keyboard. `line1` references the first string, while `line2` references the second.

- Use `Integer.parseInt` or `Double.parseDouble` to convert any Strings to ints or doubles, respectively. Reading in a line of input and converting it to a single numeric type can be done in one statement. For example:

```
int i = Integer.parseInt( br.readLine() );
```

- If a line of input has `NUM_ELEMS` numeric values you can use a String tokenizer to divide the line into tokens and then convert each token into the appropriate numeric value; For example:

```
import java.io.*;
import java.util.*;
...
StringTokenizer line = new StringTokenizer(" ");
try{
    line = new StringTokenizer(br.readLine());
} catch (Exception e) {}
for (i = 0; i < NUM_ELEMS; i++)
    array[i] = Integer.parseInt(line.nextToken());
```

- Finally, any methods that contain a call to `readLine()` (as well as methods that call a method that calls `readLine()`) need a "throws Exception" clause in the method header, unless you want to deal with the Exception handling and place the `readLine()` call in a try...catch block. **This type of exception handling is NOT required for this contest.** So it is recommended that you just include the "throws Exception" clause.

```
public static void main(String[] args) throws Exception
{
    // throws exception code in here.
}
```

## Input with the Scanner class

- When using the Scanner class for input you will need to include the following import statement as the first line in all your Java files:

```
import java.util.Scanner;
```

- The next step is to declare a Scanner object that is tied to `System.in`

```
Scanner scanIn = new Scanner(System.in);
```

- The scanner object can now be used to read Strings, doubles and ints directly from the keyboard. The methods `nextInt()` and `nextDouble()` allow you to read integers or doubles from the keyboard; for Example:

```
System.out.println("Enter quantity of toys> ");
int number = scanIn.nextInt();
System.out.println("Enter price for each item> ");
double price = scanIn.nextDouble();
```

- The `nextLine()` method returns the next line of input from the keyboard, `nextLine()` uses the newline character to determine the end of input. The `next()` method returns the next word from the input, `next()` uses whitespace (tabs, spaces or newline characters) to determine the end of input.

```
System.out.println("Enter name> ");
String name = scanIn.nextLine();
System.out.println("Enter name> ");
String anotherName = scanIn.next();
```

- `nextLine()` would return a String consisting of multiple words such as "John Smith", while `next()` would return a single word, "John".
- The Scanner class has methods that test to see if it is possible to read a specific type of input as the next item.

```
boolean hasNext()
boolean hasNextDouble()
boolean hasNextInt()
boolean hasNextLine()
```

These functions return true if it is possible to read any non-empty String, double, int or line of input, respectively. The following example reads an undetermined number of integers from the keyboard. All these functions may BLOCK if there is NO input available. In particular, pressing the enter key will have no apparent effect as the method `hasNextInt()` is expecting some valid input besides whitespace.

```
System.out.println("Enter numbers to add (q to quit)> ");
while (scanIn.hasNextInt())
{
    num = num + scanIn.nextInt();
}
System.out.println("num is " + num);
```

This program will stop when input other than an integer is entered.

- Finally, any methods that contain a call to `next()`, `nextLine()`, `nextInt()`, or `nextDouble()` (as well as methods that call one of the above methods) need a "throws Exception" clause in the method header, unless you want to deal with the Exception handling and place the methods call in a try...catch block. **This type of exception handling is NOT required for this contest.** So it is recommended that you just include the "throws Exception" clause.

```
public static void main(String[] args) throws Exception
{
    // Scanner methods in here.
}
```

## Other Modifications with Java 1.5

### Generics

- With Java 1.5 collections are generic classes with type parameters. For example, `ArrayList<E>` collects elements of type `E`; `Map<K, V>` maps keys of type `K` to values of type `V`. When a generic type is used, the type parameters are replaced with the actual types; for example, `ArrayList<Dog>` or `Map<Location, Dog>`. In versions of Java before 1.5 collections store elements of type `Object`.

#### Java 1.5

```
private ArrayList<Dog> dogs;
private Map<Location, Dog> environment;
```

#### Pre Java 1.5

```
private ArrayList dogs; // contains Dog objects
private Map environment; // Maps Location objects to Dog objects
```

### Autoboxing

- Autoboxing is the automatic conversion between primitive types and corresponding wrapper classes.

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
// 25 is automatically converted to new Integer(25)
numbers.add(25);
int num = numbers.get(0); // intValue is automatically called
```

## **Problem 1: No Consecutive Entries**

Two robots communicate to each other wirelessly. To ensure the proper reception of a message, the messages are repeated until acknowledgment is received. Duplicate input should be ignored.

Your task is to write a program that removes consecutive repeated characters from user input. Typing 'quit' should terminate the program.

### **Example of input/output:**

```
Welcome to the no-repeat protocol concept program.
```

```
Type 'quit' to exit the program.
```

```
Input to clear:
```

```
HHHello howw wwwee dooo?
```

```
Helo how we do?
```

```
Input to clear:
```

```
quuit
```

```
quit
```

```
Input to clear:
```

```
quit
```

## **Problem 2: Word Count, Letter Frequency**

The English department of Western Carolina University is conducting a statistical analysis over several books and authors. Among the various metrics being used for this research is the word count for each book as well as the frequency of each letter (the number of times each letter is repeated).

As a computer science student your role is to help the research by creating a program that would do this automatically for a given *line* of input.

The program should terminate after displaying the word count and the letter frequency (only display alphabetic letters that occurred at least once, in alphabetical order).

### **Recommendations:**

- Make sure your program handles empty input.
- What is a word? Is a composite word like *apple-pie* one or two words? Make sure you document what choices you made in your source code comments.
- Letter frequency is not case-sensitive.
- You should count numbers as a word, but do not need to count number frequency.

### **Example of input/output:**

Input to analyze:

**Hello, my name is Pierre. I am from France.**

Input is 9 words long.

Letter Frequency:

A : 3

C : 1

E : 5

F : 2

H : 1

I : 3

L : 2

M : 4

N : 2

O : 2

P : 1

R : 4

S : 1

Y : 1

Input to analyze:

**There are 10 cats in the pound.**

Input is 7 words long.

Letter Frequency:

A : 2

C : 1

D : 1

E : 4

H : 2

I : 1

N : 2

O : 1

P : 1

R : 2

S : 1

T : 3

U : 1

**Problem 3: A recursive function to calculate the first 10 Catalan numbers.**

Catalan numbers are a sequence of natural numbers that occur in many problems in mathematics.

The function beneath computes the the Catalan number for the integer *num* passed in as an argument. The function below uses a typical loop structure (or iteration) to compute the Catalan numbers. Your task is to rewrite it using **recursion**.

```
int catalan (int num)
{
    if (num == 0)
    {
        return 1;
    }
    else
    {
        int value = 1;
        for (int i = 1; i <= num; i++)
        {
            value = value * (4 * i - 2) / (i + 1);
        }
        return value;
    }
}
```

Once you have rewritten the function using recursion, you should use it to display the Catalan numbers from 0 to 10.

**Example of input/output:**

```
0 --> 1
1 --> 1
2 --> 2
3 --> 5
4 --> 14
5 --> 42
6 --> 132
7 --> 429
8 --> 1430
9 --> 4862
10 --> 16796
```

Hint:

A recursive method is one that calls itself. Here is a typical example of a recursive function to find the factorial of a number.

```
int factorial (int num)
{
    if (num == 0)
        return 1 ;
    else
    {
        return num * factorial(num - 1) ;
    }
}
```

That function is said to be recursive because it calls itself. Notice that to avoid an infinite loop there is a stopping condition (when num equals 0).

#### Problem 4: String Parser

*The explanation for this problem is long, but read it carefully, make sure you understand and proceed step-by-step, and you may find out the problem to be easier to solve than initially expected.*

In this problem you are asked to write a program that will use two input strings. The first string is referred to as *pattern*; the second string is referred to as *target*.

The pattern string is cyclic, meaning that when you reach the last character you keep reading the string from its beginning.

Your program will scan concurrently both the pattern and the target strings, character by character.

As you read characters from the pattern, you will apply the following rules:

- If the character in the pattern is anything other than a ( - ) or a ( ? ) then that character is printed to the screen. You should advance to the next character in the pattern.
- If a minus ( - ) character is encountered in the pattern, the current character from the target string is printed to the screen. You should advance to the next character in the pattern and target strings.
- If a question mark ( ? ) is encountered, compare the next character from the pattern with the current character from the target. If both characters equal, output the next character of the pattern, if not display the third character after the question mark of the pattern.
- Schematically, we can state that “?ABC” stands for “if A equals the current character of the target, then output B, otherwise output C.”
- Assume the following:
  1. neither of A, B, or C can be question marks.
  2. only B and C can be minuses (in which case they output the current target character – not the next! This is the character that you used for the comparison with A).

The program should terminate if:

1. The pattern is empty,
2. All the characters of the target have been read.

Example #1 of input/output:

Pattern String:

--?aA-B-

Target String:

How is Anna?

Result of the parse:

HowB is BAnnAB?

Here is a small walkthrough of the first several characters:

Current pattern char	Current target char	Output	Reason
-	H	H	Print current character in target string
-	o	o	print current char in target string
?	w	w	if (a == w) print 'A' else print '-'  However '-' means print out current character in target string
B	"	B	print out pattern character
-	"	"	print current char in target string
- (start back at the beginning of pattern string)	i	i	print current char in target string
-	s	s	print current char in target string
...	...	...	...

Example #2 of input/output:

Pattern String:

**ABC-D-E**

Target String:

**ab,de f**

Result of the parse:

ABCaDbEABC,DdEABCeD EABCf

Example #3 of input/output:

Pattern String:

**--?aA-B-**

Target String:

**Dragons**

Result of the parse:

DrABgons