The Heritage Computer Challenge
2004
Heritage High School
Newport News, Virginia

C++

Instructions

The problems for this contest appear on the following pages, listed in order of difficulty. The maximum number of points you can earn is indicated under the title to each problem.

Problems are designed in the format used by The Great Computer Challenge, held annually each Spring at Old Dominion University. Some of these problems were actually used at the Great Computer Challenge in previous years.

How to save your work:

1. Create a folder on your personal drive K named hcc2004. All other project folders will created **inside this folder.**
2. Each solution should be saved as a project in a folder whose name is IDENTICAL to the project file name (minus the extension). For each project, place your source code in the default source file named main.cpp.

********** Problems **********

Where is My Robot?

(10 points)

Save in folder named: **Robot**

The problem is one of finding the coordinates of the missing Robot. The Robot is known to have strted at the Cartesian coordinates of (0,0) and to have made a series of moves as dictated by his input sequence. Each step in the input sequence is one of the words north, south, east, and west. For an input of east, the Robot moves on unit in the positive x direction and of course no go west the Robot moves on unit in the negative x direction. Similarly a movement to the north casues the y value to increase by one while south causes the y value to decrease by one.

Given an input sequence, your program must tell us the final coordinates of the Robot.

Input to your program will be a series of directions (north, south, east, or west) each separated by one space.

The output to your program will be the final locatin of the Robot in Cartesian coordinates.

Sample

Input

```
west south east east east north north
```

Output

```
(2,1)
```

Palindromic Primes
(10 points)

Save in folder named: **PalPrimes**

A "palindromic prime" is a prime that is also a prime when its digits are reversed. 11, 13, and 17 are such primes.

Find and display the next ten "palindromic primes" that are not reversals of a smaller palindromic prime. Thus, 31 and 71 would not be displayed because they are the reversals of 13 and 17, which are already listed.

When you run out of two-digit numbers, switch to three, etc.

Perfect Numbers
(20 points)

Save in folder named: **Perfect**

The Greeks began an examination of numerology by classifying all positive integers as either **perfect**, **abundant**, or **deficient**. This classification scheme is based on the factors (even divisors) of the number. If the sum of all the factors of a number (excluding the number itself) equals the number then it is said to be **perfect**. For example, the factors of 6 are 1, 2, 3, and 6. Therefore, the number 6 is a **perfect** number. The total of the factors of 6 (excluding 6 itself) is 1 + 2 + 3 = 6.

An **abundant** number is one in which this sum of its factors (excluding the number itself) is greater than the number. An example of an **abundant** number is 12 because 1 + 2 + 3 + 4 + 6 = 16 > 12. All numbers that are neither **perfect** nor **abundant** are **deficient**.

This program accepts a sequence of integers, calssifies each as abundant, perfect, or deficient and displays the factors of the number. It terminates when given a input of zero.

For input, the user is prompted by "Specify a positive integer (0 to quit): ".

For output, display a line like "**This number is** *type* **(factors givein below).**" where *type* is either **perfect**, **abundant**, or **deficient**. On the following line, the factors of the input integer are shown.

Sample:

```
Specify a positive integer (0 to quit): 6
This number is perfect (factors given below).
1 2 3
Specify a positive integer (0 to quit): 12
This number is abundant (factors given below).
1 2 3 4 6
Specify a positive integer (0 to quit): 333
This number is deficient (factors given below).
1 3 9 37 111
Specify a positive integer (0 to quit): 0
```

The Krypton Factor

(20 points)

You have been employed by the organisers of a Super Krypton Factor Contest in which contestants have very high mental and physical abilities. In one section of the contest the contestants are tested on their ability to recall a sequence of characters which has been read to them by the Quiz Master. Many of the contestants are very good at recognising patterns. Therefore, in order to add some difficulty to this test, the organisers have decided that sequences containing certain types of repeated subsequences should not be used. However, they do not wish to remove all subsequences that are repeated, since in that case no single character could be repeated. This in itself would make the problem too easy for the contestants. Instead it is decided to eliminate all sequences containing an occurrence of two adjoining identical subsequences. Sequences containing such an occurrence will be called ``easy''. Other sequences will be called ``hard''.

For example, the sequence ABACBCBAD is easy, since it contains an adjoining repetition of the subsequence CB. Other examples of easy sequences are:

- BB
- ABCDACABCAB
- ABCDABCD

Some examples of hard sequences are:

- D
- DC
- ABDAB
- CBABCBA

In order to provide the Quiz Master with a potentially unlimited source of questions you are asked to write a program that will read input lines that contain integers n and L (in that order), where $n > 0$ and L is in the range $1 <= L <= 26$, and for each input line prints out the nth hard sequence (composed of letters drawn from the first L letters in the alphabet), in increasing alphabetical order (alphabetical ordering here corresponds to the normal ordering encountered in a dictionary), followed (on the next line) by the length of that sequence. The first sequence in this ordering is A. You may assume that for given n and L there do exist at least n hard sequences.

For example, with $L = 3$, the first 7 hard sequences are:

A
AB
ABA
ABAC
ABACA
ABACAB
ABACABA

As each sequence is potentially very long, split it into groups of four (4) characters separated by a space. If there are more than 16 such groups, please start a new line for the 17th group.

Therefore, if the integers 7 and 3 appear on an input line, the output lines produced should be

```
ABAC ABA
7
```

Input is terminated by a line containing two zeroes. Your program may assume a maximum sequence length of 80.

Sample Input

```
30 3
0 0
```

Sample Output

```
ABAC ABCA CBAB CABA CABC ACBA CABA
28
```

Text Formatting

(30 points)

Save in folder named: **Formatting**

The problem is to reformat a paragraph to a specified width. This means that no line in the paragraph should be longer than the width specified and each line should be as long as possible without going over the specified with. In the paragraph, words are separated by spaces and line breaks. No word should be split between two lines in the output. Leading or trailing punctuation should be treated as part of the word.

Input

Any number of lines of text, followed by a line which begins with an asterisk (*), followed by a space, followed by an integer. The integer specifies the width of the paragraph. The asterisk is not considered to be part of the paragraph.

Output

The input lines of text reformatted to the width specified.

Sample

Input

```
This function treats a string
(the value of EXPR) as a vector of unsigned integers, and
returns the value of the element specified by OFFSET and
BITS.
The function may also be assigned to, which causes
the element to be modified.
* 40
```

Output

```
This function treats a string (the value
of EXPR) as a vector of unsigned
integers, and returns the value of the
element specified by OFFSET and BITS.
The function may also be assigned to,
which causes the element to be modified.
```
