The Heritage Computer Challenge
2005
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 hcc2005. 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)
```

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
```

Cryptarithm

(20 points)

Save in folder named: **Crypt**

In a cryptarithm puzzle letters have been substituted for numbers in an arithmetic calculation. Each distinct letter is represented by a particular, but unique digit. Code a program to find all the solutions to the following cryptarithm.

```
    OIL
     NO
    OIL
+    ON
_______
   LINO
```

Note that the numbers of the solution(s) can not have leading zeros. The output should display the above cryptarithm followed by the solution(s) in the same format.

Roman Cash Register

(30 points)

Save in folder named: **Roman**

Back in the days of the Classical Roman Empire, merchants filled the market places daily, toting their wares, while customers scurried about, ready to spend their hard-earned wages. Customers were out to find a bargain. Merchants were out to make a dinari (the Roman unit of currency). Any merchant of those days would have given his (or her) ivory false teeth to have a cash register--especially one that could add Roman numerals and compute the heavy ten percent Roman sales tax.

Your job is to write a program that will simulate just such a cash register. Since Roman numerals didn't have decimal places, your ten percent tax should be computed to the nearest whole dinari. And just for the convenience of people today, please display an extra column showing each amount in equivalent dollars (US).

First have the user enter a maximum of five items, each followed by its price (in Roman numerals only, of course). Entering a blank ends the entry process early. Then display, each on a separate line but in neat columns, each item, its Roman numeral price, and its price in U.S. dollars. Display a subtotal, then the tax, and then the total.

Following is a sample run:

```
Roman Cash Register.

Enter up to 5 items and price below (blank to end early).
Item 1: SHIRT
Roman Price of SHIRT: XXI
Item 2: SHOES
Roman Price of SHOES: LIV
Item 3: TIE
Roman Price of TIE: XXIII
Item 4:
Item                Price         $ (US)
SHIRT                 XXI          21.00
SHOES                 LIV          54.00
TIE                  XXIII         23.00
Subtotal           XCVIII          98.00
10% Tax                 X          10.00
Total               CVIII         108.00
```

No single item should cost more than one hundred dinari.
