

Programming Problems  
11th Annual CS Programming Contest  
(Draft)

WCU Department of Mathematics and Computer Science

April 25, 2000

**Please read all instructions before you begin.**

## Instructions

If you have competed in this contest in the past, you may notice that this year's problems are more difficult than the problems used in previous years. This is intentional.

Because these problems are more difficult, *you should not expect to finish them all*. The judges will not expect you to submit solutions to all of the problems. Although you should try to get as many problems completed as you can, your team will probably achieve a higher score if you concentrate on completely solving a subset of the problems assigned rather than submitting incomplete solutions to all the problems.

We suggest that you read all the problems before you begin, then select the ones you feel you are most able to implement. After you finish those, then work on the problems that you are less familiar with.

## Scoring

The programs your team submits will be scored using the following rubric:

### Individual Program Scoring Criteria

<i>Item</i>	<i>Points</i>
Correctness of Program	60
Comments and Style	30
Efficiency of Program	10
<hr/> Total	<hr/> 100

The total score for your team will be determined by adding the scores for each of the programs which your team submits. These total scores will be used to determine the first, second, and third place teams.

## Problem 1

This problem is to write a simple cash register program.

Your program will prompt the user for a list of purchases. Each purchase has two identifiers: a non-negative number representing the price of the purchase, and a letter representing whether the purchase is taxable or not. Taxable purchases use the letter 't', while non-taxable purchases use the letter 'n'. The user will never enter a dollar sign, and the user will never enter white space between the price and the type for each purchase.

Your program must allow the user to purchase as many items as the user wants to. The user ends the input by entering a purchase (taxable or non-taxable) with a price of 0 dollars. At this point, you program computes the total due by charging a 6 percent sales tax on the taxable sub-total and adding the non-taxable sub-total to this.

Your program should output the total of both the taxable and non-taxable parts, the sales tax due, and the total due. The following shows how a typical program session looks:

```
program outputs: Enter a purchase followed by its type:
user inputs:    10.40t
program outputs: Enter a purchase followed by its type:
user inputs:    2.00n
program outputs: Enter a purchase followed by its type:
user inputs:    1.15t
program outputs: Enter a purchase followed by its type:
user inputs:    4.50t
program outputs: Enter a purchase followed by its type.
user inputs:    0t
program outputs: Taxable total:  $ 16.05
program outputs: Sales tax:    $ 0.96
program outputs: Non-taxable total:  $ 2.00
program outputs: Grand Total:    $ 19.01
```

## Problem 2

*Derived from Problem B, ACM International Collegiate Programming Contest , Asia Regional Contest, Kyoto, 1999-12-04*

People in Silverland like squares. They have square houses, square cars, and square coins. Not only are the coins square in shape, the coins have the values of perfect squares.

Coins are minted with values of all square numbers up to 81 ( $= 9^2$ ), i.e., 1-credit coins, 4-credit coins, 9-credit coins, 16-credit-coins, 25-credit coins, 36-credit coins, 49-credit coins, 64-credit coins, and 81-credit coins. Because of the values of these coins, there is often more than one way to pay a certain number of credits with coins. Note that a certain set of coins should only be counted once, independent of the sequence the coins are used in.

For example, there are five distinct combinations of coins to pay 12 credits:

- twelve 1-credit coins,
- one 4-credit coin and eight 1-credit coins,
- two 4-credit coins and four 1-credit coins,
- one 9-credit coin and three 1-credit coins,
- three 4-credit coins

The following would not be another distinct sequence:

- a 1-credit coin, two 4-credit coins and then three 1-credit coins,

Write a program to count the number of ways to pay a certain amount with Silverland coins.

Your program should input from the user a credit amount, which must be positive and less than 300. You should verify that the input does match these requirements. Your program should output the number of ways to pay the amount input in Silverland coins.

### Problem 3

For this program a "date" is represented by three integers, the month, day, and year of the date. Thus the date **February 14, 1999** is represented by **2 14 1999** .

Write a program which inputs two such dates from the keyboard and outputs the number of days between the two dates to the user. The user will always enter the dates so that the first date entered occurs before or on the second date entered.

The number of days "between" the dates includes the second date but not the first date; the following examples of program input should illustrate this:

#### **Program run 1**

```
program outputs: Input the First Date:
user inputs :    1 1 2000
program outputs: Input the Second Date:
user inputs:     1 1 2000
program outputs: There are no days between those dates.
```

#### **Program run 2**

```
program outputs: Input the First Date:
user inputs:     1 5 1999
program outputs: Input the Second Date:
user inputs:     1 7 1999
program outputs: There are 2 days between those dates.
```

#### **Program run 3**

```
program outputs: Input the First Date:
user inputs:     2 28 2000
program outputs: Input the Second Date:
user inputs:     3 1 2000
program outputs: There are 2 days between those dates.
```

#### **Program run 4**

```
program outputs: Input the First Date:
user inputs:     2 28 1900
program outputs: Input the Second Date:
user inputs:     3 1 1900
program outputs: There is 1 day between those dates.
```

*(Continued on next page)*

*(Problem 1, continued)*

Your program must correctly handle leap years. A year is a leap year if either of the following two conditions is true:

1. The year is divisible by 4 but not by 100; i.e. the remainder upon dividing the year by 4 is 0, but the remainder upon dividing the year by 100 is not 0
2. The year is divisible by 400, i.e. the remainder upon dividing the year by 400 is 0.

The following examples illustrate the rules for leap years:

1. 1999 is not a leap year, because it is not divisible by 4 or 400.
2. 1996 is a leap year because it is divisible by 4 but not by 100.
3. 1900 is not a leap year because it is divisible by 4 and by 100.
4. 2000 is a leap year because it is divisible by 400.

## Problem 4

The price of a single stock can vary greatly from day to day. Your program should input the price of a stock for several consecutive days, then output the best day to buy and sell the stock in order to make the most profit.

Note that the program cannot merely find the day with the highest price and the day with the lowest price; the sale of the stock must come *after* the purchase of the stock, not before. Your program should output the day of the buy, the buying price, the day of the sale, the selling price, and the profit from the sale.

Your program must allow the user to enter as many days worth of data as the user wants to.

Here is an example of the program input and output:

```
program outputs: Please enter the price for each day.
program outputs: End your input with a 0.
user inputs:    10
user inputs:    2
user inputs:    5
user inputs:    3
user inputs:    7
user inputs:    4
user inputs;    0
program outputs: You should buy on day 2 at a price of 2
program outputs: You should sell on day 5 at a price of 7
program outputs: You will earn 5 dollars for this sale
```