North Carolina A&T State University



Triad Programming Contest

Spring 2017

Contest

Problems

Contents

[Drunken Drivers 2](#_Toc510466483)

[Grocery Store Simulation 3](#_Toc510466484)

[Butterfly Sequence 4](#_Toc510466485)

[Marathon Winners 5](#_Toc510466486)

[Packing Boxes 6](#_Toc510466487)

[Password Validator 7](#_Toc510466488)

[Evaluating Post-fix Notation Expressions 8](#_Toc510466489)

[Rock, Paper, Scissors, Candle, Tree Tournament 9](#_Toc510466490)

[Travel Planning 10](#_Toc510466491)

# Drunken Drivers

The National Highway Traffic Safety Administration has a database containing information on all of the fatal accidents in the United States, the Fatality Analysis Reporting System (FARS). You are to write a program that reads a file to determine which hour of the week is most likely to have fatal accidents involving drunken drivers and which hour is the safest.

Input is in the file, accident.txt, where each line contains the following data for the year 2015 in which there were over 30,000 accidents. The data in each line of the file is:

day Day of the month

month month (1 = January, 2 = February, etc.)

DayOfWeek The day of the week of the accident where 1 = Sunday, 2 = Monday, …, 7 = Saturday

hour The hour 0 – 23, the accident occurred. A value of 99 is used when the hour is unknown.

minute The minute of the hour 0 – 59, the accident occurred. A value of 99 is used when the time is unknown.

drunkDrv non-zero if this accident involved a drunken driver and 0 otherwise

Ignore any line of data where the hour or minute is greater than 23.

Your program must display the day of the week and hour of the day that has the largest percentage of drunk driver related accidents and the day of the week and hour of the day with the lowest percentage of accidents involving drunks. Percentages can be displayed with any accuracy with one or more digits to the right of the decimal point.

**Sample input**

1 1 5 2 40 1

1 1 5 22 13 0

1 1 5 1 25 1

4 1 1 0 57 1

7 1 4 7 9 0

8 1 5 9 59 0

8 1 5 18 33 0

3 1 7 21 30 0

13 1 3 8 0 0

5 1 2 18 45 0

remaining 32,156 more lines can be found in the sample data input file.

**Sample output**

Most drunk accidents 68.0% on Saturday at 2:00

Least drunk accidents 2.8% on Tuesday at 9:00

# Grocery Store Simulation

We want to analyze a supermarket's waiting lines to determine the optimum number of checkout counters to install. Input is from the file grocery.txt There are three parameters that influence the number check checkout counters. These numbers will appear on one line of input. The values are:

**s:** The salary of the checkout clerks given as dollars per minute. If you have two checkout counters, you will have to pay twice this salary. If there are three checkout counters, you will have to pay three times this amount and so forth.

**p:** The probability that for a given minute, a customer will want to check out. This is given as a number between 0.0 and 1.0. No more than one customer will want to checkout in a minute. The customer will join the shortest waiting line or go directly to the checkout clerk if on has nobody waiting.

**w:** The estimated lost future revenue for each minute that a customer waits. Customers don't like to wait and will go elsewhere in the future if they have to wait. For each minute each customer has to wait before they start checking out, it cost the company this much.

Assumptions:

1. Each minute that a customer waits in line costs the store loses **w** dollars in lost future purchases.
2. Each clerk receives **s** dollars per minute salary.
3. There is a **p** probability that a customer arrives in line in any one minute. There is a 1-**p** probability that no customer arrives in line in any one minute.
4. At most, one customer can arrive to be checked out in any one minute.
5. The time required to check out one customer is a uniformly distributed random integer from 3 to 7 minutes.
6. When a customer arrives in line, she remains in line until she is served.
7. Initially all checkout lines are empty.
8. A customer will join the shortest line and will not change lines.
9. No more than 10 checkout counters can be installed.
10. As a simplification, we assume everything happens at the beginning of each minute.

The output for the program is just the optimal number of checkout counters that minimizes the sum of salary and lost future purchases.

You can write a program that simulates from 1 to 10 checkout lines measuring the cost of each one. You can run the simulation as long as you feel necessary, but 100,000 minutes gives a good estimate. The program should display the number of lines that has the lowest cost.

**Sample Input**

0.10 0.3 2.00

**Sample Output**

4

# Butterfly Sequence

Consider the following algorithm to generate a sequence of numbers. Start with an integer *n*. If *n* is even, divide the number by 2 to get the next number in the sequence. If *n* is odd, multiply by 3 and add 1 to create the next number in the sequence. For example, the following sequence of numbers will be generated for *n* = 11:

11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

The numbers wander up and down until they get to 1.

For an input *n*, the *cycle-length* of *n* is the number of values generated up to and *including* the 1. In the example above, the cycle length of 11 is 15. Given any number, you are to determine the cycle length of that number.

The file butterfly.txt has a list of numbers whose cycle length is to be determined. The list is preceded by an integer indicating the number of values to follow. For each number in butterfly.txt, display the number, a space and its cycle length.

**Example input**

4

11

13

9

2

**Example output**

11 15

13 10

9 20

2 2

# Marathon Winners

The results from the big marathon are in! You are to display the names of the top three winners of each age group in order (gold, silver, bronze) along with their medal.

All input will be read from a file named marathon.txt. Data collected on each runner, in order, is first name, last name, finish time (saved as a decimal number), and age (integer). All data is separated by a space.

The age groups are:

Youngsters (less than 26)

Midsters (26 to 59)

Geezers (60 and up)

The output should give the age group followed by the names of the three fasters runners in that age group displayed in ascending order of their times.

**Sample Input**

Woodhouse Coe 2.59 70

Winston Churchhill 3.01 75

Steven Wozniac 3.39 60

Sterling Archer 2.30 35

Roy O'Dowd 3.30 37

about 100 more records can be seen in the sample input file.

Sample Output

Youngsters Age Group Winners:

Gold: Kat McMillan

Silver: Katie Smith

Bronze: Eddie Lane

Midsters Age Group Winners:

Gold: Patrick Star

Silver: Ronaldo McNair

Bronze: Barry Dillon

Geezers Age Group Winners:

Gold: Johnathan Kennedy

Silver: Plato Grekus

Bronze: Ada Lovelace

# Packing Boxes

Optimal packing has become a necessary requirement for many package delivery services. The idea of getting as much as you can into a small compact space for delivery is key to a successful business. You are to write a program that will find the optimal combination of items, which will utilize the given space. Each item has a value and a weight. You want to select the items to put into a box so that you maximize the sum of the value of all items in the box while not exceeding the weight limit for the box. To make things simple, all weights and values are integer numbers.

The first input will be the maximum weight a box can hold. The sum of the weight of all items put in the box cannot exceed this amount.

The next input, ***n***, is the number of items available for shipping. This is followed by ***n*** lines where each line contains a one character identifier (i.e. ‘A’ or ‘X’), the value of the item and the weight of the item, each separated by a space.

The output should be the total value, total weight (formatted as shown below) followed on the next line by the one character identifiers of the items to be put in the box (in any order).

total value: *V* total weight: *W*

**Sample input**

8

4

A 1 2

B 2 2

C 2 2

Z 3 4

**Sample output**

total value: 7 total weight: 8

B C Z

# Password Validator

Write a program that will validate a password.

Password must have,

1. 8 to 32 characters
2. at least one numeric digit
3. at least one uppercase letter
4. at least one lowercase letter
5. must not have space or slash (/)
6. starting character must not be a number

Sample passwords will be read in from a file named password.txt with the first line containing the number of passwords that will follow. Each password is on a separate line of the file.

You should display the password followed by YES if the password if valid, or NO if the password is not valid.

**Sample Input: Sample Output:**

4

Pass5 Pass5 NO

passwordpasswordpasswordpassword passwordpasswordpasswordpassword NO

p4223076 p4223076 NO

Password1 Password1 YES

PASSWORD47 PASSWORD47 NO

# Evaluating Post-fix Notation Expressions

We usually write a mathematical expression in infix notation where the operator is between the two operands, such as 2+3. In post-fix notation, the operator comes after the two operands, such as 2 3 +. The advantage of post-fix notation is that any equation can be written without the need for parenthesis to change the order of evaluation. Expressions are evaluated left to right with an operator acting on the two previous numbers or results.

Write a program to evaluate post-fix notation expressions. The program should read post-fix expressions from the text file “postfix.txt”. The first line of the file will contain an integer specifying the number of lines to follow. The program should evaluate each expression and output the numerical result and the expression. All numbers in the expressions are integers. The standard four math operators +, -, \* and / may appear in the expressions. Subtraction is evaluated as the first value minus the second value: “5 4 -” in post-fix is the same as “5 – 4” in infix. Division in post-fix “25 8 /” is the same as “25 / 8” giving 3 with the remainder using integer division being dropped. It can be assumed that all expressions are valid post-fix expressions.

All numbers and operators are separated with a space. Each post-fix expression is terminated by a period.

The output must display the result, a colon (":") and the original post-fix expression.

**Sample Input: Sample Output:**

4

3 6 \* . 18: 3 6 \*

2 2 + 4 \* 8 / . 2: 2 2 + 4 \* 8 /

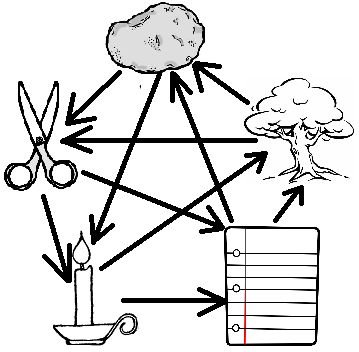
1 1 + 2 6 \* 12 2 / + + . 20: 1 1 + 2 6 \* 12 2 / + +

2 5 3 + \* . 16: 2 5 3 + \*

# Rock, Paper, Scissors, Candle, Tree Tournament

A tournament is being held for champion players of the game Rock, Paper, Scissors, Candle Tree. For players A, B, and C determine who wins each game.

**Scissors** cuts **paper**, **paper** covers **rock**, **rock** crushes **candle**, **candle** burns down **tree**, **tree** blunts **scissors**, **scissors** cuts the wick off **candle**, **candle** burns **paper**, **paper** shades out **tree**, **tree** grows on **rock**, and **rock** crushes **scissors**.

A player wins the game if another player does not defeat them. If two or more players are undefeated, it is a draw.

Input: Input is from the file tournament.txt The first line in the file is an integer that represents the number of games that follow. Each following line represents a game and has three uppercase characters separated by a space. The first character represents the choice of player A, the second character is the choice of player B and the third is from player C. R represents rock, P represents paper, S represents scissors, C represents candle, and T represents tree. The only letters in the input will be upper case R, P, S, C and T.

Output: For each game, display the original input, three spaces and which player won each game. Display DRAW if the round results in a draw.

**Sample Output**

T S R A wins

T P R B wins

R P S DRAW

P P S C wins

S S C DRAW

**Sample Input:**

5

T S R

T P R

R P S

P P S

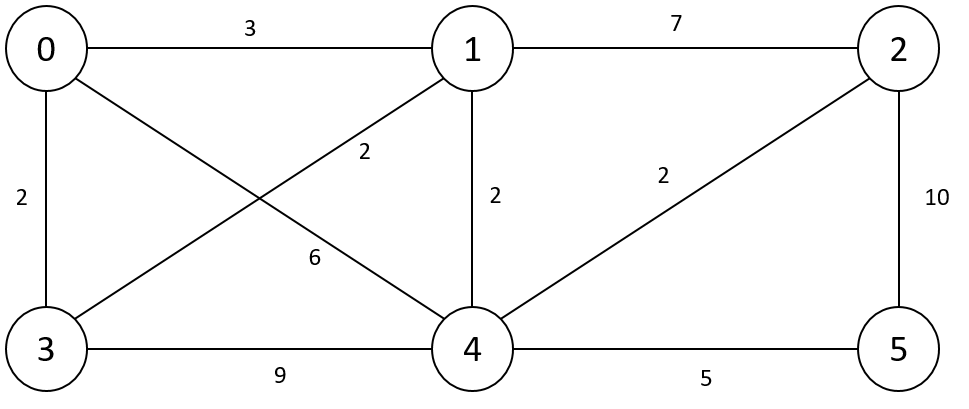
S S C

R P T

# Travel Planning

The location of the next Olympics has been chosen and you have been asked to write a program to calculate how far each guest must travel from their city. Each city is represented by a vertex numbered from 0-N, and the edges between each vertex represent the highways connecting each city and their length in miles. The input to the program starts with an integer specifying the number of cities on our map (which will always be less than 20), which is followed by the total number of highways connecting each city. Following this are a number of edges giving the distance between cities over existing highways. Each line will contain the number of the first city (numbered from zero to one less than the maximum), the number of the second city and the distance in miles between the cities. All highways operate in both directions. The last line contains the number of the city that will be hosting the Olympics, the destination of the travelers. The output from the program should be a list of city numbers and their distance from the destination city. You should find the **shortest path** from each city to the destination, as the athlete’s sponsors are cheap and don’t want to pay too much for gas.

Consider the map



**Sample Input: Sample Output:**

6 city distance

10 0 0

0 4 6 1 3

0 1 3 2 7

0 3 2 3 2

1 2 7 4 5

1 4 2 5 10

2 5 10

3 1 2

4 3 9

4 2 2

5 4 5

0