North Carolina A&T State University



Triad Programming Contest

Spring 2018

Contest

Problems

Contents

[1. Stack Script 2](#_Toc511136089)

[2. Fire Plane Dispatcher 3](#_Toc511136090)

[3. Square Farmland 4](#_Toc511136091)

[4. Set Algebra 5](#_Toc511136092)

[5. Star Search 6](#_Toc511136093)

[6. Comment Removal 7](#_Toc511136094)

[7. Sum to Nothing 8](#_Toc511136095)

[8. Asteroid Position 9](#_Toc511136096)

# Stack Script

Write a program that will interpret a simple stack oriented script language. The script language starts with data declarations followed by commands. Data declarations are one to a line in the format:

variablename number

where variablename is the name of a program variable. Program variables are case insensitive and may contain only letters. The number is an integer value. There may be up to 16 variables. The end of the variable list is denoted by a line containing only “program”. Following the program line are executable statements. There are seven possible statement types:

push *var* Push the value of the specified variable onto the stack.

pop *var* Remove the top of stack value and store it in the specified variable.

add Add the top two values on the stack. Remove the two values on top of the stack and store the addition result on the top of the stack.

sub Subtract the top of the stack value from the value immediately below the top of the stack. Remove the two values on top of the stack and store the subtraction result on the stack.

mult Multiply the top of the stack value with the value immediately below the top of the stack. Remove the two values on top of the stack and store the product result on the stack.

div Divide the top of the stack value by the value immediately below the top of the stack. Remove the two values on top of the stack and store the division result on the stack.

display Display “Top of stack:” followed by the value of the top of the stack. The value on the top of the stack is not changed or removed.

There may be up to 256 program statements. The end of the program statements is denoted by a line containing only “end”. All variable names and commands are case insensitive, so ADD, add and Add are all the same.

Your program is to execute a program written in this stack script language. The script programs are written by Computer Science faculty, so there are no errors in them. There will be no undeclared variables and no stack overflows or underflows. The script program should be read from **script.txt**

**Example input** from **script.txt**

dog 5

cat 3

bird 11

program

push bird

push cat

push dog

display

sub

add

display

end

**Example output**

Top of stack: 5

Top of stack: 9

# 

# Fire Plane Dispatcher

Airplanes to dump water and fire retardant on forest fires, stand ready at several airports. Write a program to be used by the fire service to dispatch the closest airplane to help put out a fire. Airports are located by their latitude and longitude which are each represented by three integers, the degrees, minutes and seconds. We will assume that all latitudes are north, and longitudes are west, the portion of the globe that encompasses North America. Airports are identified by A, B, C, D, etc. The first airport is identified by A, the second airport is B, the third airport is C, etc. There is a maximum of 10 airports. The fire location is identified by its latitude and longitude coordinates. Your program will calculate the closest airport to dispatch airplanes to fight the fire.

Input from **fire.txt** consists of:

the number of airports

latitude and longitude coordinates of each airport

latitude and longitude coordinates of the emergency location

Your program should print the airport that is closest to the fire. The closest airport is defined as the one with the shortest straight line distance. You may ignore the curvature of the earth. The airport must be identified by its letter.

**Sample input** from **fire.txt**

5

37 57 5 120 1 49

49 25 45 98 35 42

45 38 40 94 10 27

31 59 22 79 39 18

36 6 48 81 7 17

41 34 23 104 14 60

**Sample Output:**

The call should be serviced by airport B

# Square Farmland

You have been hired by a binary farmer to find the largest square plot of land for planting crops on her new farm. The plot to be farmed must be a square where the width and depth are the same. *(Her plants have square roots.)* She has created a text file that models her property. Every square hectare of land good for planting is denoted by a 1, and every hectare that is deemed not suitable is represented by a 0. The input for this program comes from **largestsquare.txt.** The first line of the input file will be the integer ***n***, the number of rows and columns of the matrix. The following ***n*** rows will contain ***n*** 1’s or 0’s separated by spaces. Your program should find the row and column position of the upper left corner (corner with the smallest index) of the largest square that is suitable for planting and output the number of hectares in that square. The first row and first column are numbered zero. If there is more than one square of the largest size, your program can output any largest square.

**Example input** from **largestsquare.txt**

13

0 0 0 0 0 0 0 0 0 0 1 1 1

1 1 1 1 0 0 0 0 0 0 1 1 1

0 0 0 0 1 1 0 0 1 1 1 1 1

1 1 0 0 1 1 0 1 0 1 0 1 0

1 1 0 0 1 1 1 1 0 0 0 0 0

1 1 0 0 1 1 0 0 1 1 1 0 1

1 1 1 **1** 1 1 1 1 1 1 0 1 1

0 0 0 1 1 1 1 0 0 0 0 1 0

0 1 1 1 1 1 1 1 1 1 0 1 1

0 0 1 1 1 1 1 1 0 0 1 1 1

0 1 1 0 1 0 1 0 1 1 1 1 0

1 1 1 0 0 1 1 0 1 0 0 1 1

1 0 1 0 1 0 1 0 1 0 1 0 1

**Example output**

Biggest plot is at 6, 3 of size 16 hectare

# Set Algebra

This program deals with sets of integers. The integers have the values from 0 to 255. A number can only appear in a set once although the same number can appear in multiple sets. There will be four sets identified by the uppercase letters A, B, C and D. The first four lines of input to the program will be an unordered list of numbers, each separated by a space. The first line represents set A, the second line is set B, the third line is set C and the fourth is set D. There can be anywhere from 1 to 255 numbers in a set. The last number on each line of input is -1 to indicate the end of the line. Note that -1 is not a member of the set.

After the first four lines of input defining the sets, there will be a statement defining operations to be done on the sets. There are two possible operations:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Operator** | **Definition** |
| union | & | The union of two sets is a set containing all the members of both sets with duplicates removed. |
| intersection | ^ | The intersection of two sets is a set containing only those members that appear in both sets. |

If set A is 1 5 3 22 and set B is 22 4 3 2, then A&B is 1 2 3 4 5 22 and A^B is 3 22.

The two operators can be combined for form expressions involving multiple sets. The operators for union and intersection have the same precedence and are always interpreted from left to right. If set A is 1 5 3 22, set B is 22 4 3 2 and set C is 1 9 3 then A^B&C is 1 3 9 22.

The program must interpret the set expression and display the results. Input is from the file **set.txt** If a number appears more than once in a set, any duplicates must be removed. The values in the resulting set may be displayed in any order. If the resulting set is empty, the program should display NULL. The equation may use each of the sets A, B, C or D, once, multiple times or not at all.

**Sample input** from **set.txt**

40 223 163 230 22 100 -1

223 22 255 6 73 -1

1 62 113 223 -1

62 113 160 22 -1

B&C&A^D

**Sample output**

22 62 113

# Star Search

ASCII art is the construction of recognizable graphical images by arranging symbols from the ASCII character set. For example, we might draw a simple star shape by arranging nine ASCII characters as follows:

\|/

-\*-

/|\

Your job is to write a program that finds occurrences of this star shape in a large grid of ASCII characters.

Input consists of several grids of ASCII characters from the file **stars.txt**. Input starts with an integer value, n, indicating how many grids your program is to process. Each grid description starts with a line containing two integers, **R** and **C**, reporting the number of rows and columns in the grid. The contents of the grid will be given on the next **R** lines of input. Each line will contain up to **C** characters. If the line contains less than **C** characters, your program should assume the trailing characters are spaces.

For each grid in the input, print out the locations of all the star shapes. Each star should be reported by printing the row and column of its center (the asterisk) on an output line. In your output, row and column numbers should be separated by one space, and the top row of a grid should be considered row 1. Likewise, the leftmost column should be considered column 1. For each grid, stars should be reported sorted by row number. If there are multiple stars on the same row, they should be sorted by column number. If a particular grid contains no stars, print the line “no stars” for that grid. Print a blank line after the output for each grid.

**Sample Input** from **stars.txt**

3

7 18

\|/ \|/

-\*--------------\*-

/|\ ASCII ART /|\

| IS |

\|/ EVERYWHERE \|/

-\*--------------\*-

/|\ /|\

2 2

\*-

|\

5 5

\|/

-\*-

/|\|/

-\*-

/|\

**Sample Output**

2 2

2 17

6 2

6 17

no stars

2 2

4 4

# Comment Removal

A sneaky student has gotten hold of the instructor’s solution to a programming assignment written in Conbol. The student wants to submit the program as his own. To make it look like a student program, he wants to remove all the comments. You are to write a program that reads a text file called **comment.txt** containing the Conbol program source code and display the program text without the comments.

There are two types of comments in Conbol. Comments can begin with two exclamation characters, **!!** and end with two exclamation characters, **!!** This form of comment can extend across multiple lines or only a part of a line. All characters in the between !! and !! should be removed, including the !! and !! and any end of line characters inside the comment. The other form of a comment starts with exclamation dash, **!-** and continues to the end of the line. When you remove a comment of this form, eliminate the **!-** and all following characters. You should leave the end of line character.

Conbol programs have string constants that are in “double quotes”. If the start characters of a comment (either !- or !!) are inside a string constant, then they are part of the string constant and do not represent the start of a comment. String constants never span more than one line. It is possible that double quotes may appear inside a comment. If they do, they should be removed just like any other character. You may assume that the input program is syntactically correct.

**Example input** from **comment.txt**

public class { !! My commented program !!

public static void main(String[] unused) { !- main method

String thing = "fake !- comment ";

thing = !! short comment !! "real stuff";

!! long

comment

!!

thing = "!! " + "!! "; !- weird

int num = 5 / 6;

}

}

**Example output**

public class {

public static void main(String[] unused) {

String thing = "fake !- comment ";

thing = "real stuff";

thing = "!! " + "!! ";

int num = 5 / 6;

}

}

# Sum to Nothing

Given a set of integers, is there a non-empty subset of the numbers whose sum is zero? For example, given the set { −7, −3, −2, 5, 8 }, the answer is yes because the subset { −3, −2, 5 } sums to zero.

The input to this program comes from the text file **zero.txt**. The first line of the file is the number of sets to be checked. Each set contains an integer number **N** (between 2 and 10) giving the number of values to follow. The next line for each set has **N** integers. If there is a subset of the **N** integers (size 1 to **N**) that sums to zero, the program should display the integers in the subset, each separated by a space. If more than one subset exists, only one has to be shown. If no subset exists for that set, the program should print “No zero sum”.

**Example input** from **zero.txt**

3

5

−7 −3 −2 5 8

3

-1 -2 4

6

2 4 6 -1 -3 8

**Example output**

−3, −2, 5

No zero sum

4 -1 -3

# Asteroid Position

There are lots of objects moving around in space. Write a program that will predict the position of an asteroid at a given time in the future based on its current position and speed.

In 1687, Isaac Newton formulated the principles governing the motion of two particles under the influence of their mutual gravitational attraction. As a close approximation, we will assume everything orbits the sun on a two-dimensional plane. Let Δ*x* represent the difference in the position of two object, A & B, along the X coordinate and let Δ*y* represent the difference along the Y coordinate. Calculate r, the distance between the two objects in meters, using

Asteroids are primarily influenced by the gravitational attractions of the sun since it is six orders of magnitude more massive than any of the planets. We can represent the mass of the sun,   
1.9890 × 1030 Kg, as MSUN and the mass of an asteroid as MA.

The force in Newtons pulling an asteroid to the sun can be calculated using

where G is the gravitational constant, 6.67 × 10−11 in the units N·m2·kg−2. The force can be divided into X and Y components by:

The force will accelerate the asteroid following Newton's second law of motion, F = ma. Over a small change in time, Δt, the velocity of an asteroid is changed by

After calculating the change in velocity of an asteroid, the position of the asteroid can be updated by

By repeating these calculations every Δt seconds, the asteroid will move under the gravitational force of the sun. A Δt of 21,600 seconds (6 hours) should be used for an accurate location.

The program must first read in data about the asteroid. The input file, **asteroid.txt**, contains the position, velocity and mass of the asteroid. A single line contains:

X position of the asteroid in meters from the sun

Y position of the asteroid in meters from the sun

X direction velocity of the asteroid in meters/second

Y direction velocity of the asteroid in meters/second

Mass of the asteroid in Kg

We will assume the sun is motionless at position 0, 0.

Display the X and Y coordinates of the asteroid in one week, 604,800 seconds, from the start.

**Sample input** from **asteroid.txt**

7.7578E+09 5.7378E+10 -4.7468E+04 6.4179E+03 8.7E+19

**Sample output**

After a week, the asteroid is at -2.1021142446132698E10, 6.818917214943033E10

Note that accuracy within four decimal digits is sufficient.