Andrew Taylor

atayl136

Homework 2: Stacks

1a) Use operations to set i to the bottom element of the stack, leaving unchanged:

Use empty to confirm elements exist

Create an alternate stack and confirm it is empty

Peek a variable from the stack called OldTop

While Empty == False

Pop the top integer a variable x

Push x onto the alternate stack

Set x as i

While Empty(alternate) == False

Pop the top integer a variable x

Push x onto the original stack

If x == OldTop

Print Done.

Else

Error

(now you have i and the original stack is unchanged and it is apparent from the top of the stack.)

1b) Use operations to set i to the 3rd element from the bottom of the stack, *may* be changed

Use empty to confirm elements exist

Create an alternate stack and confirm it is empty

Set n = 3

Set variable called counter = 0

While Empty == False

Pop the top integer a variable x

Push x onto the alternate stack

While Empty(alternate) == False

Pop the top integer a variable x

Counter +1

If counter = n

i = x

Push x onto the original stack

(the instructions were permissive “may” but I preferred to leave the stack unchanged)

2a) Simulate action of algorithm checking delimiters of each strings by using a stack and contents

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Action |  |  |  |  |  |  |
| Push | Push | Pop(match) | Push | Push | Pop(match) | Pop(match) |
| Stack |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  | ( |  |  |
|  | [ |  | [ | [ | [ |  |
| { | { | { | { | { | { | { |

The stack is not left empty there is an error with **{[A+B]-[(C-D)].**

2b) Simulate action of algorithm checking delimiters of each strings by using a stack and contents

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Action |  |  |  |  |  |  |  |  |  |
| Push | Push | Pop,match | Push | Push | Push | Pop,match | Pop,match | Pop,match | Pop,match |
| Stack |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | [ |  |  |  |  |
|  |  |  |  | ( | ( | ( |  |  |  |
|  | ( |  | { | { | { | { | { |  |  |
| ( | ( | ( | ( | ( | ( | ( | ( | ( |  |

The stack is left empty this is a valid string **((H) \* {([J+K])}).**

3) Write an algorithm to determine whether an input string is of the form xCy.

def xCyFunction(string):

string = input

Create empty stack

yFlag = false

For char in string

If char == A or B

If yFlag == false

Push char to stack

Else

Pop stack assign to var

If char == var

Continue

Else

break

Elif char == C

yFlag == True

Else

Print This is not the form xCy

If stack Empty == True

Print Done. confirmed input is of the form xCy as defined

else

Print This is not the form xCy

4) Write an algorithm as above except aDbDcD…Dz with any number of such strings in-between:

def aDbDcDFunction(string):

string = input

Create empty stack to compare A and B

yFlag = false

For char in string

If char == A or B

If yFlag == false

Push char to stack

Else

Pop stack assign to var

If char == var

If Empty Stack

yFlag == False

Continue

Else

Continue

Else

Break (or maybe Exception)

Elif char == C

yFlag == True

Continue

Elif char == D

yFlag == False

Continue

Else

Print User input error not A B C or D

If stack Empty == True

Print Done. confirmed input is of the form aDbDcD as defined

else

Print This is not the form aDbDcD

The key to this function is y separates both C and D from the As and Bs so a flag is actually a decent idea.

5) 1D array without arrays using two stacks

The top of stack1 and stack2 is the index, the bottom of stack1 is the begging of the array and the bottom of stack2 is the end of the array.

insert an element at a position in an array:

def Insert(stack1, stack2, integer value for position):

# cycle to top

while(!stack1.empty())

stack2.push(stack1.pop())

# index to position and push to insert

for int to position-1

stack1.push(stack2.pop())

stack1.push(value)

Read an element at a position in an array:

def Read(stack1, stack2, integer position):

# cycle to Top

while(!stack1.empty())

stack2.push(stack1.pop())

# index to position and peek to read

for int to position

stack1.push(stack2.pop())

stack2.peek

6) Design a method for keeping two stacks in one fixed space linear array so neither stack overflows until all memory is used. The stack is s[SPACESIZE].

Let’s say the total SPACESIZE in items = x, and n = number of items or elements.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S[0] → |  |  |  | S[n] | S[x+k] |  |  |  | ← S[x] |

The stacks start at opposite ends of the array and grow towards each other so, Stack1 starts at s[0], grows to s[n] length and Top, Stack2 starts at s[x], with a length of abs k (k is kept as a negative number), so grows to s[x+k] Top. Where k or n is a line that can drift across the midline. If one stack grows beyond the midline the values above will change. (assuming spacesize can be given in number of items n and abs k). In practice the items will not be of uniform spacesize but this approach would still work because it is looking at variables which ultimately aim to sum the total spacesize of the elements against the limitation of the total SPACESIZE. Two different variables for the actual spacesize should be used, but here the number of items is how I’m keeping track, for simplicity. A better treatment would introduce a formula between item totals and space totals, but it would be arbitrary.

The point is there won’t be an overflow until the stacks meet in the array where that occurs.

Functions used:

Push1(Data) – pushes Data item onto Stack1, incrementing n, Temp and thus Top at s[n] positively.

Push2(Data) – pushes Data item onto Stack2, incrementing, k Temp and thus Top at s[x+k] negatively.

Pop1 – pops an element from Stack1 using those pointers and returns it. Uses Stack1 Top s[n].

Pop2 – pops an element from Stack2 using those pointers and returns it. Uses Stack 2 Top s[x-k].

MaxLength is == (s[x+k]-s[n]=0). When this length is reached, no matter where either stack is, there will be an overflow on the next push. Another way in terms of spacesize is simply: x-k-n = 0 is the overflow.

7a) Transform to Prefix: - \* + A B - + $ C - D E F G, Transform to Postfix: - A B + C D E - $ F + G - \*

7b) Transform to Prefix: + A $ / + \* - B C - D E F G - H J, Transform toPostfix: A B C - D E - \* F + G / H J - $ +

8) Transform to Infix:

a) ( A + ( ( ( B $ C ) \* D ) - ( ( E + F ) / ( G \* H ) ) ) ) + I

b) ( A - ( B $ C ) ) + ( D \* ( ( E \* F ) \* G ) )

c) ( ( A - B ) + C ) $ ( D + ( E - F ) )

d) ( A \* ( B $ ( C + ( D - E ) ) ) ) - ( E \* F )

9) Evaluate postfix with A = 1, B = 2, C = 3

a) A B + C - B A + C $ -

Infix: ( ( A + B ) - C ) - ( ( B + A ) $ C) = ( ( 1 + 2 ) - 3 ) - ( ( 2 + 1 ) ^ 3 ) = 0 - ( 3 ^ 3 ) = - 27

b) A B C + \* C B A - + \*

Infix: ( A \* ( B + C ) ) \* ( C + ( B - A ) ) = ( 1 \* ( 2 + 3 ) ) \* ( 3 + ( 2 - 1 ) ) = ( 5 \* 4 ) = 20

10) Write an infix to prefix function

Def prefixfunction(string)

Create stack

Create dictionary of operators, keys operators, values precedence

For character in string[-1]

If not operator

Print to output

Elif character in operators

If stack has precedence then Pop

Push

Elif char is right )

Push

Elif char is left (

Pop until match

After the loop we Pop until Empty, printing

check if empty, Done

else

error