

Data Structures and Algorithms - Activity 2

Aamn Pathak¹, 22051662, CSE-54

October 5, 2023

¹School of Computer Engineering, KIIT-DU

Question 1 Postfix Expression Validation: Write a program using C programming language that validates a given postfix expression to ensure it is well-formed and follows the rules of postfix notation. Check for invalid characters and the correct number of operands and operators.

Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4
5  #define MAX_SIZE_1662 100
6
7  typedef struct {
8      int data_1662[MAX_SIZE_1662];
9      int top_1662;
10 } Stack_1662;
11
12 void initialize(Stack_1662* stack_1662) {
13     stack_1662->top_1662 = -1;
14 }
15
16 void push(Stack_1662* stack_1662, int value_1662) {
17     if (stack_1662->top_1662 < MAX_SIZE_1662 - 1) {
18         stack_1662->data_1662[++stack_1662->top_1662] = value_1662;
19     } else {
20         printf("stack_1662 overflow error\n");
21         exit(EXIT_FAILURE);
22     }
23 }
24
25 int pop(Stack_1662* stack_1662) {
26     if (stack_1662->top_1662 >= 0) {
27         return stack_1662->data_1662[stack_1662->top_1662--];
28     } else {
29         printf("stack_1662 underflow error\n");
30         exit(EXIT_FAILURE);
31     }
32 }
33
34 bool isOperand(char ch_1662) {
35     return (ch_1662 >= '0' && ch_1662 <= '9');
36 }
37
38 bool isOperator(char ch_1662) {
39     return (ch_1662 == '+' || ch_1662 == '-' || ch_1662 == '*' ||
40         ch_1662 == '/');
41 }
42
43 bool isValidExpression(char* expression) {
44     Stack_1662 stack_1662;
45     initialize(&stack_1662);
46
47     int i = 0;
48     while (expression[i] != '\0') {
49         if (isOperand(expression[i])) {
50             push(&stack_1662, expression[i] - '0');
```

```

50     } else if (isOperator(expression[i])) {
51         if (stack_1662.top_1662 < 1) {
52             return false;
53         }
54         int op2 = pop(&stack_1662);
55         int op1 = pop(&stack_1662);
56         push(&stack_1662, 0);
57     } else if (expression[i] != ' ') {
58         return false;
59     }
60     i++;
61 }
62
63 return (stack_1662.top_1662 == 0);
64 }
65
66 int main() {
67     char expression[MAX_SIZE_1662];
68     printf("enter a postfix expression: ");
69     gets(expression);
70
71     if (isValidExpression(expression)) {
72         printf("valid!\n");
73     } else {
74         printf("invalid!\n");
75     }
76
77     return 0;
78 }
79
80

```

Output

```

1
2 dsa\assignment\2 via C v6.3.0-gcc
3 $ gcc 1.c -o 1
4
5 dsa\assignment\2 via C v6.3.0-gcc
6 $ ./1.exe
7 enter a postfix expression: 532**64**
8 valid!
9
10 dsa\assignment\2 via C v6.3.0-gcc
11 $ ./1.exe
12 enter a postfix expression: 32*5+4*6**
13 invalid!
14
15 dsa\assignment\2 via C v6.3.0-gcc took 19s
16 $ ./1.exe
17 enter a postfix expression: 235**4*6**
18 invalid!
19
20 dsa\assignment\2 via C v6.3.0-gcc
21 $
22

```

Question 2 Solve the Tower of Hanoi problem using stacks instead of recursion. Implement a program using C programming language to move a tower of n disks from one peg to another while following the rules of the Tower of Hanoi puzzle.

Code

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct StackNode {
5      int data_1662;
6      struct StackNode* next_1662;
7  } StackNode;
8
9  typedef struct {
10     StackNode* top_1662;
11 } Stack_1662;
12
13 void initialize(Stack_1662* stack_1662) {
14     stack_1662->top_1662 = NULL;
15 }
16
17 int isEmpty(Stack_1662* stack_1662) {
18     return (stack_1662->top_1662 == NULL);
19 }
20
21 void push(Stack_1662* stack_1662, int value_1662) {
22     StackNode* newNode_1662 = (StackNode*)malloc(sizeof(StackNode));
23     if (newNode_1662 == NULL) {
24         printf("Memory allocation failed!\n");
25         exit(EXIT_FAILURE);
26     }
27     newNode_1662->data_1662 = value_1662;
28     newNode_1662->next_1662 = stack_1662->top_1662;
29     stack_1662->top_1662 = newNode_1662;
30 }
31
32 int pop(Stack_1662* stack_1662) {
33     if (isEmpty(stack_1662)) {
34         printf("Stack_1662 underflow!\n");
35         exit(EXIT_FAILURE);
36     }
37     StackNode* temp = stack_1662->top_1662;
38     int value_1662 = temp->data_1662;
39     stack_1662->top_1662 = temp->next_1662;
40     free(temp);
41     return value_1662;
42 }
43
44 typedef struct {
45     Stack_1662* pegs;
46     int num_pegs_1662;
47 } Tower;
48
49 void initializeTower(Tower* tower, int num_pegs_1662) {

```

```

50     tower->pegs = (Stack_1662*)malloc(num_pegs_1662 * sizeof(
Stack_1662));
51     if (tower->pegs == NULL) {
52         printf("Memory allocation failed!\n");
53         exit(EXIT_FAILURE);
54     }
55     tower->num_pegs_1662 = num_pegs_1662;
56
57     for (int i = 0; i < num_pegs_1662; i++) {
58         initialize(&tower->pegs[i]);
59     }
60 }
61
62 void moveRing(Tower* tower, int source, int destination) {
63     int ring = pop(&tower->pegs[source]);
64     push(&tower->pegs[destination], ring);
65 }
66
67 void solveTower(int num_rings_1662, int num_pegs_1662) {
68     Tower tower;
69     initializeTower(&tower, num_pegs_1662);
70
71     for (int i = num_rings_1662; i >= 1; i--) {
72         push(&tower.pegs[0], i);
73     }
74
75     for (int tries = 0; tries < 1 + num_rings_1662 % 2; tries++)
76     {
77         int move_peg_one_right = 1;
78
79         for (int moves = 0; moves < (1 << num_rings_1662) - 1;
moves++) {
80             if (move_peg_one_right) {
81                 for (int peg = 0; peg < num_pegs_1662; peg++) {
82                     if (!isEmpty(&tower.pegs[peg])) {
83                         if (tower.pegs[peg].top_1662->data_1662 == 1) {
84                             int next_peg = (peg + 1) % num_pegs_1662;
85                             moveRing(&tower, peg, next_peg);
86                             printf("Moving value_1662 1 from peg %d to peg %d
\n\n", peg + 1, next_peg + 1);
87                             break;
88                         }
89                     }
90                 }
91             } else {
92                 int moved_a_ring = 0;
93                 for (int peg = 0; peg < num_pegs_1662; peg++) {
94                     if (!isEmpty(&tower.pegs[peg])) {
95                         int value_1662 = tower.pegs[peg].top_1662->
data_1662;
96                         if (value_1662 != 1) {
97                             for (int n = 0; n < num_pegs_1662; n++) {
98                                 int next_peg = (peg + n) % num_pegs_1662;
99                                 if (next_peg == peg) {
100                                     continue;
101                                 }
102                                 if (isEmpty(&tower.pegs[next_peg]) ||

```

```

102         value_1662 < tower.pegs[next_peg].top_1662->
data_1662) {
103             moveRing(&tower, peg, next_peg);
104             moved_a_ring = 1;
105             printf("Moving value_1662 %d from peg %d to
peg %d\n\n", value_1662, peg + 1, next_peg + 1);
106             break;
107         }
108     }
109 }
110 }
111
112     if (moved_a_ring) {
113         break;
114     }
115 }
116
117     if (!moved_a_ring) {
118         printf("Error, failed to move\n");
119         exit(EXIT_FAILURE);
120     }
121 }
122
123
124     move_peg_one_right = !move_peg_one_right;
125 }
126     printf("Finished pass\n\n");
127 }
128
129     free(tower.pegs);
130 }
131
132 int main() {
133     int num_rings_1662;
134     int num_pegs_1662 = 3;
135
136     printf("Enter the number of rings: \n");
137     scanf("%d", &num_rings_1662);
138
139     solveTower(num_rings_1662, num_pegs_1662);
140     return 0;
141 }
142

```

```

1
2 dsa\assignment\2 via C v6.3.0-gcc
3 $ gcc 2.c -o 2
4
5 dsa\assignment\2 via C v6.3.0-gcc
6 $ ./2
7 Enter the number of rings:
8 2
9 Moving value_1662 1 from peg 1 to peg 2
10
11 Moving value_1662 2 from peg 1 to peg 3
12
13 Moving value_1662 1 from peg 2 to peg 3
14

```

```

15 Finished pass
16
17
18 dsa\assignment\2 via C v6.3.0-gcc
19 $ ./2
20 Enter the number of rings:
21 4
22 Moving value_1662 1 from peg 1 to peg 2
23
24 Moving value_1662 2 from peg 1 to peg 3
25
26 Moving value_1662 1 from peg 2 to peg 3
27
28 Moving value_1662 3 from peg 1 to peg 2
29
30 Moving value_1662 1 from peg 3 to peg 1
31
32 Moving value_1662 2 from peg 3 to peg 2
33
34 Moving value_1662 1 from peg 1 to peg 2
35
36 Moving value_1662 4 from peg 1 to peg 3
37
38 Moving value_1662 1 from peg 2 to peg 3
39
40 Moving value_1662 2 from peg 2 to peg 1
41
42 Moving value_1662 1 from peg 3 to peg 1
43
44 Moving value_1662 3 from peg 2 to peg 3
45
46 Moving value_1662 1 from peg 1 to peg 2
47
48 Moving value_1662 2 from peg 1 to peg 3
49
50 Moving value_1662 1 from peg 2 to peg 3
51
52 Finished pass
53
54
55 dsa\assignment\2 via C v6.3.0-gcc
56 $
57

```

Disclaimer All the programs were compiled on gcc.exe (MinGW.org GCC-6.3.0-1) 6.3.0. This document was generated using L^AT_EX.