Data Structures and Algorithms - Activity 2

Aamn Pathak[1], 22051662, CSE-54

October 5, 2023

[1]School 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

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_SIZE_1662 100

typedef struct {
  int data_1662[MAX_SIZE_1662];
  int top_1662;
} Stack_1662;

void initialize(Stack_1662* stack_1662) {
  stack_1662->top_1662 = -1;
}

void push(Stack_1662* stack_1662, int value_1662) {
  if (stack_1662->top_1662 < MAX_SIZE_1662 - 1) {
    stack_1662->data_1662[++stack_1662->top_1662] = value_1662;
  } else {
    printf("stack_1662 overflow error\n");
    exit(EXIT_FAILURE);
  }
}

int pop(Stack_1662* stack_1662) {
  if (stack_1662->top_1662 >= 0) {
    return stack_1662->data_1662[stack_1662->top_1662--];
  } else {
    printf("stack_1662 underflow error\n");
    exit(EXIT_FAILURE);
  }
}

bool isOperand(char ch_1662) {
  return (ch_1662 >= '0' && ch_1662 <= '9');
}

bool isOperator(char ch_1662) {
  return (ch_1662 == '+' || ch_1662 == '-' || ch_1662 == '*' || ch_1662 == '/');
}

bool isValidExpression(char* expression) {
  Stack_1662 stack_1662;
  initialize(&stack_1662);

  int i = 0;
  while (expression[i] != '\0') {
    if (isOperand(expression[i])) {
      push(&stack_1662, expression[i] - '0');
```

```
      } else if (isOperator(expression[i])) {
        if (stack_1662.top_1662 < 1) {
          return false;
        }
        int op2 = pop(&stack_1662);
        int op1 = pop(&stack_1662);
        push(&stack_1662, 0);
      } else if (expression[i] != ' ') {
        return false;
      }
      i++;
    }

    return (stack_1662.top_1662 == 0);
}

int main() {
    char expression[MAX_SIZE_1662];
    printf("enter a postfix expression: ");
    gets(expression);

    if (isValidExpression(expression)) {
      printf("valid!\n");
    } else {
      printf("invalid!\n");
    }

    return 0;
}


```

Output

```

dsa\assignment\2 via C v6.3.0-gcc
$ gcc 1.c -o 1

dsa\assignment\2 via C v6.3.0-gcc
$ ./1.exe
enter a postfix expression: 532*+64*+
valid!

dsa\assignment\2 via C v6.3.0-gcc
$ ./1.exe
enter a postfix expression: 32*5+4*6*+
invalid!

dsa\assignment\2 via C v6.3.0-gcc took 19s
$ ./1.exe
enter a postfix expression: 235*+4*6*+
invalid!

dsa\assignment\2 via C v6.3.0-gcc
$

```

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

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct StackNode {
  int data_1662;
  struct StackNode* next_1662;
} StackNode;

typedef struct {
  StackNode* top_1662;
} Stack_1662;

void initialize(Stack_1662* stack_1662) {
  stack_1662->top_1662 = NULL;
}

int isEmpty(Stack_1662* stack_1662) {
  return (stack_1662->top_1662 == NULL);
}

void push(Stack_1662* stack_1662, int value_1662) {
  StackNode* newNode_1662 = (StackNode*)malloc(sizeof(StackNode
));
  if (newNode_1662 == NULL) {
    printf("Memory allocation failed!\n");
    exit(EXIT_FAILURE);
  }
  newNode_1662->data_1662 = value_1662;
  newNode_1662->next_1662 = stack_1662->top_1662;
  stack_1662->top_1662 = newNode_1662;
}

int pop(Stack_1662* stack_1662) {
  if (isEmpty(stack_1662)) {
    printf("Stack_1662 underflow!\n");
    exit(EXIT_FAILURE);
  }
  StackNode* temp = stack_1662->top_1662;
  int value_1662 = temp->data_1662;
  stack_1662->top_1662 = temp->next_1662;
  free(temp);
  return value_1662;
}

typedef struct {
  Stack_1662* pegs;
  int num_pegs_1662;
} Tower;

void initializeTower(Tower* tower, int num_pegs_1662) {
```

```
    tower->pegs = (Stack_1662*)malloc(num_pegs_1662 * sizeof(Stack_1662));
    if (tower->pegs == NULL) {
      printf("Memory allocation failed!\n");
      exit(EXIT_FAILURE);
    }
    tower->num_pegs_1662 = num_pegs_1662;

    for (int i = 0; i < num_pegs_1662; i++) {
      initialize(&tower->pegs[i]);
    }
  }

  void moveRing(Tower* tower, int source, int destination) {
    int ring = pop(&tower->pegs[source]);
    push(&tower->pegs[destination], ring);
  }

  void solveTower(int num_rings_1662, int num_pegs_1662) {
    Tower tower;
    initializeTower(&tower, num_pegs_1662);

    for (int i = num_rings_1662; i >= 1; i--) {
      push(&tower.pegs[0], i);
    }

    for (int tries = 0; tries < 1 + num_rings_1662 % 2; tries++) {
      int move_peg_one_right = 1;

      for (int moves = 0; moves < (1 << num_rings_1662) - 1; moves++) {
        if (move_peg_one_right) {
          for (int peg = 0; peg < num_pegs_1662; peg++) {
            if (!isEmpty(&tower.pegs[peg])) {
              if (tower.pegs[peg].top_1662->data_1662 == 1) {
                int next_peg = (peg + 1) % num_pegs_1662;
                moveRing(&tower, peg, next_peg);
                printf("Moving value_1662 1 from peg %d to peg %d\n\n", peg + 1, next_peg + 1);
                break;
              }
            }
          }
        } else {
          int moved_a_ring = 0;
          for (int peg = 0; peg < num_pegs_1662; peg++) {
            if (!isEmpty(&tower.pegs[peg])) {
              int value_1662 = tower.pegs[peg].top_1662->data_1662;
              if (value_1662 != 1) {
                for (int n = 0; n < num_pegs_1662; n++) {
                  int next_peg = (peg + n) % num_pegs_1662;
                  if (next_peg == peg) {
                    continue;
                  }
                  if (isEmpty(&tower.pegs[next_peg]) ||
```

```
                    value_1662 < tower.pegs[next_peg].top_1662->
data_1662) {
                      moveRing(&tower, peg, next_peg);
                      moved_a_ring = 1;
                      printf("Moving value_1662 %d from peg %d to
peg %d\n\n", value_1662, peg + 1, next_peg + 1);
                      break;
                    }
                  }
                }
              }

              if (moved_a_ring) {
                break;
              }
            }

            if (!moved_a_ring) {
              printf("Error, failed to move\n");
              exit(EXIT_FAILURE);
            }
          }


          move_peg_one_right = !move_peg_one_right;
        }
        printf("Finished pass\n\n");
      }

      free(tower.pegs);
    }

    int main() {
      int num_rings_1662;
      int num_pegs_1662 = 3;

      printf("Enter the number of rings: \n");
      scanf("%d", &num_rings_1662);

      solveTower(num_rings_1662, num_pegs_1662);
      return 0;
    }
```

```
dsa\assignment\2 via C v6.3.0-gcc
$ gcc 2.c -o 2

dsa\assignment\2 via C v6.3.0-gcc
$ ./2
Enter the number of rings:
2
Moving value_1662 1 from peg 1 to peg 2

Moving value_1662 2 from peg 1 to peg 3

Moving value_1662 1 from peg 2 to peg 3

```

```
    Finished pass


    dsa\assignment\2 via C v6.3.0-gcc
    $ ./2
    Enter the number of rings:
    4
    Moving value_1662 1 from peg 1 to peg 2

    Moving value_1662 2 from peg 1 to peg 3

    Moving value_1662 1 from peg 2 to peg 3

    Moving value_1662 3 from peg 1 to peg 2

    Moving value_1662 1 from peg 3 to peg 1

    Moving value_1662 2 from peg 3 to peg 2

    Moving value_1662 1 from peg 1 to peg 2

    Moving value_1662 4 from peg 1 to peg 3

    Moving value_1662 1 from peg 2 to peg 3

    Moving value_1662 2 from peg 2 to peg 1

    Moving value_1662 1 from peg 3 to peg 1

    Moving value_1662 3 from peg 2 to peg 3

    Moving value_1662 1 from peg 1 to peg 2

    Moving value_1662 2 from peg 1 to peg 3

    Moving value_1662 1 from peg 2 to peg 3

    Finished pass


    dsa\assignment\2 via C v6.3.0-gcc
    $

```

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