Documentation_Lab-5

11.c **Program to implement a modified Stack ADT using arrays**

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

//Code to parse integer from a string
int stoi(char *str)                    // converts character to integer
{
        int x;
        sscanf(str, "%d", &x);
        return x;
}

int push(int *A, int *top, int key, int n)          // pushes the element into the stack
{
                                                    // n stores the maximum no of elements in stack
                                                    // key is the value to be pushed
                                                    // top stores the address of the top of array
        if((*top)==n-1)                 // if top is n-1 then stack is full
        {
                return -1 ;
        }

        (*top)++;
        A[*top]=key;                                // pushing the value and increasing the top
        return 1 ;


}

int pop(int *A, int *top, int no)                   // takes out the element at the top
{                                                   // top stores the address of the top of array
   int value = A[*top];
   if((*top)==-1)                                   // checks if stack is empty
                                                    // no maintains that we need to make changes in
                                                          positive array or in the negative array
   {
        if(no==-1)
                return 0;
        else
                return -1;
```

```
    }

    (*top)--;              //  decreasing the top value and returning the value at the top
    return value;
}

void displayElement(int *A, int *top)          // printing the elements of the array
                                               // top stores the address of the top of array
{
    if((*top)==-1)                             // is Empty
    {
            printf("\n");
            return ;
    }
    for(int i=(*top); i>=0; i--)               // else if not Empty printf the values from
top
            printf("%d\n", A[i]);
    return;
}

int main (int argc, char **argv)
{
    int n1, n2;
    int *arrp, *arrn, top1, top2;
    top1 = -1;                         // initializes both top to be -1
    top2 = -1;



        char line[128];
        char v1[15];
        char v2[15];
        char v3[15];

        int ret;
        int lineNo = 0;

        while (fgets(line, sizeof line, stdin) != NULL )
        // taking input from standard input in the line array
        {
        sscanf(line, "%s %s %s", v1, v2, v3);
                                   // storing the strings in line in v1 , v2 & v3
        lineNo++;

        if(lineNo == 1)            // line 1 contains the size of the positive array
        {
```

```
n1 = stoi(v1);
arrp = (int*)malloc(n1*sizeof(int));              // allocating size for the positive array
continue;
}
if(lineNo == 2)                    // line 2 contains the size of the negative array
{
        n2 = stoi(v1);
        arrn = (int*)malloc(n2*sizeof(int));     // allocating size for the negative
array
        continue;
}
if(strcmp(v1,"PSH") == 0)
{
if(stoi(v2)>=0)                    // positive value means push in the positive array
{
                ret = push(arrp, &top1, stoi(v2), n1);
                if(ret<0)                          // was not pushed
                        printf("-1\n");
}
else if(stoi(v2)<0)                // negative value means push in the negative array
{
                ret =push(arrn, &top2, stoi(v2), n2);
                if(ret<0)          // was not pushed
                        printf("-1\n");
}
}
else if(strcmp(v1,"POPN") == 0)
{
ret = pop(arrn, &top2, -1);        // -1 for popping in the negative array
printf("%d\n", ret);
}
else if(strcmp(v1,"POPP") == 0)
{
ret = pop(arrp, &top1, 1);         // 1 for popping in the negative array
printf("%d\n", ret);
}
else if(strcmp(v1,"PRTP") == 0)        // printing the positive elements
{
displayElement(arrp, &top1);
}
else if(strcmp(v1,"PRTN") == 0)        // printing the negative elements
{
displayElement(arrn, &top2);
}
else
{
```

```
        printf("INVALID\n");
        }
        }

        if(arrp)          // freeing up the allocated memory
        free(arrp);
        if(arrn)
         free(arrn);
   return 0;
}
```

Time Complexity:

1. **Push and Pop take constant time to execute O(1).**
2. **Printing takes O(n1+n2) time.**

Space Complexity:

Only creating of array takes O(n1+n2) space rest every function takes O(1) space complexity.

21.c **Program to implement the Stack ADT using singly linked list**

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

int stoi(char *str)                // converts character to integer
{
        int x;
        sscanf(str, "%d", &x);
        return x;
}

typedef struct node
                        // defining a node, it has an character value and a node pointer
{
   char value;
   struct node *next_link;
}node;

typedef struct stack
// defining a stack, as it is dynamic so it doesn't have a capacity, only node *top
{
   node* top;
```

```
}stack;

stack* createStackLinkedList()
// creating a Stack and allocation of the memory to stack pointer, top points to NULL initially
{
    stack *st;
    st=(stack*)malloc(sizeof(stack));
    st->top=NULL;
    return st;
}

void push(stack *st, char* key)                 // pushing the character key into the stack
{
    node *add;
    add=(node*)malloc(sizeof(node));            // creating a node and storing the key to it
    add->value=*key;
    if(st->top==NULL)                           // if stack is empty
    {
            add->next_link=NULL;
                    // NULL , as there is no node below the first node to point
            st->top=add;
            return;
    }
 add->next_link = st->top;
// else if non Empty , then stores the address of top (i.e. address of the node below the formed node) into the link of the newly formed node.
    st->top=add;          // now top points to the last element in the stack
    return;
}

char pop(stack *st)               // takes out the element at the top
{
    node* deleter;                // node pointer which we will free up
    char ex_val ;
    if(st->top==NULL)             // if empty then nothing to pop
            return '-';

    deleter = st->top;            // else deleter points to the top node
    ex_val = (st->top)->value;    // extracting the top value in ex_val
    st->top=(st->top)->next_link;
          // breaking up the link of top , storing the address of the below node
    free(deleter);
                    // freeing up the memory and returning the extracted value
    return     ex_val;
```

```
}

char peek(stack *st)            // gives the element at the top of stack
{
   if(st->top==NULL)            // if Empty
         return '-';
   return (st->top)->value;     // else return top value
}

void display(stack *st)         // printing up the elements
{
   node* temp;
   temp =st->top;               // storing the top node pointer in the temp
   while(temp!=NULL)            // until reach the first element in the node print
   {
         printf("%c\n", temp->value);
         temp=temp->next_link;        // going to the node below  in the stack
   }
   return;
}

int size(stack *st)             // printing the size of the stack
{
   node* temp;
   temp=st->top;
   int size = 0;          // initializing the size to be zero
   while(temp!=NULL)
// until reach the first node as first node has NULL in it's link value
   {
         temp = temp->next_link;      // increase the size and go to the below node
         size++;
   }
   return size;
}

int isEmpty(stack *st)          // checking is empty
{
   if(st->top==NULL)            // top has NULL if empty
         return 1;              // returning 1 if empty
   else return -1;              // else return -1, i.e. not empty
}

int main (int argc, char **argv)
{
        char line[128];
```

```
char v1[15];
char v2[15];
char v3[15];

stack *st;
st = createStackLinkedList();          // allocating memory
char ret;

while (fgets(line, sizeof line, stdin) != NULL )
// taking input from standard input in the line array
{
sscanf(line, "%s %s %s", v1, v2, v3); // storing the strings in line in v1 , v2 & v3

if(strcmp(v1,"PSH") == 0)
{
        push(st, v2);
}
else if(strcmp(v1,"POP") == 0)
{
ret = pop(st);
if(ret=='-')                // means empty, therefore return -1
        printf("-1\n");
else
       printf("%c\n", ret);
}
else if(strcmp(v1,"TOP") == 0)
{
ret = peek(st);
if(ret=='-')                // means empty, therefore return -1
        printf("-1\n");
else
        printf("%c\n", ret);
}
else if(strcmp(v1,"PRT") == 0)
{
display(st);
}
else if(strcmp(v1,"SZE") == 0)
{
        ret = size(st);
        printf("%d\n", ret);
}
else if(strcmp(v1, "EMP")==0)
{
        ret = isEmpty(st);
        printf("%d\n", ret);
```

```
        }
        else
        {
        printf("INVALID\n");
        }
        }

        if(st)              // freeing up the stack memory
        free(st);

        return 0;
}
```

Time Complexity:

Display elements and size take O(n) time, rest take constant i.e. O(1) time.

Space Complexity:

Every function takes O(1) space complexity.

22.c **Program decide whether the symbols are balanced**

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

int stoi(char *str)              // converts character to integer
{
        int x;
        sscanf(str, "%d", &x);
        return x;
}

typedef struct node
{                        // defining a node, it has an character value and a node pointer
   char value;
   struct node *next_link;
}node;

typedef struct stack
// defining a stack, as it is dynamic so it doesn't have a capacity, only node *top
{
```

```
    node* top;
}stack;

stack* createStackLinkedList()
// creating a Stack and allocation of the memory to stack pointer, top points to NULL initially
{
    stack *st;
    st=(stack*)malloc(sizeof(stack));
    st->top=NULL;
    return st;
}

void push(stack *st, char key)       // same pushing the character key into the stack
{
    node *add;
    add=(node*)malloc(sizeof(node));
   add->value=key;
    if(st->top==NULL)
    {
            add->next_link=NULL;
            st->top=add;
            return;
    }

    add->next_link = st->top;
    st->top=add;
    return;
}

char pop(stack *st, char key)        // checks if the element in the top matches the opposite of element in the input key , if yes then go on pop the, else it is not a balanced symbol
{
    node* deleter;
    if(st->top==NULL)
            // if stack is empty and still you want to pop , not balanced symbols, return 0
            return '0';
    else if(st->top->value=='{' && key=='}' || st->top->value=='(' && key==')' || st->top->value=='[' && key==']')
            // if it matches with the opposite of top with the key, go on
    {
```

```
        deleter = st->top;
        st->top=(st->top)->next_link;
        free(deleter);
        return     '1';
    }
    Else        // if doesn't matches return 0
        return '0';

}


int main (int argc, char **argv)
{
        char line[128];
        char v1;
        char v2;
        char v3;
    int len=0 ;

        stack *st;
        st = createStackLinkedList();       // allocating memory
        char ret;

        fgets(line, sizeof line, stdin) ;       // takes one line of input as string
        len = strlen(line);                     // stores the length of the input line
        for(int i=0; i<len; i++)
        {
                v1 = line[i];                           // v1 stores the character of the line

                if(v1=='{' || v1=='(' || v1=='[')
        // if input v1 is any one of the opening parenthesis push it on the stack
                {
                        push(st, v1);
                }
                else if(v1=='}' || v1==')' || v1==']')
        // if it's is one of the closing parenthesis pop it , if it matches with the go on
                {
                        ret = pop(st, v1);
                        if(ret=='0')            // if it's doesn't matches return 0
                        {
                         printf("0\n");
                                return 0;
                         }
```

```
                }
                else if(v1!='\n')                // if input is nothing like an opening or
        closing parenthesis print invalid input. \n means input has ended.
                        printf("INVALID\n");
        }
    if(st->top!=NULL)          // after coming out if stack is not empty, (example only ‘
{ ‘ left) return 0
    printf("0\n");
    Else                  // else stack is empty return 1
    printf("1\n");
        if(st)
        free(st);                  free up the memory allocated to the stack

        return 0;
}
```

Time Complexity:

Take constant i.e. O(1) time.

Space Complexity:

Every function takes O(1) space complexity.