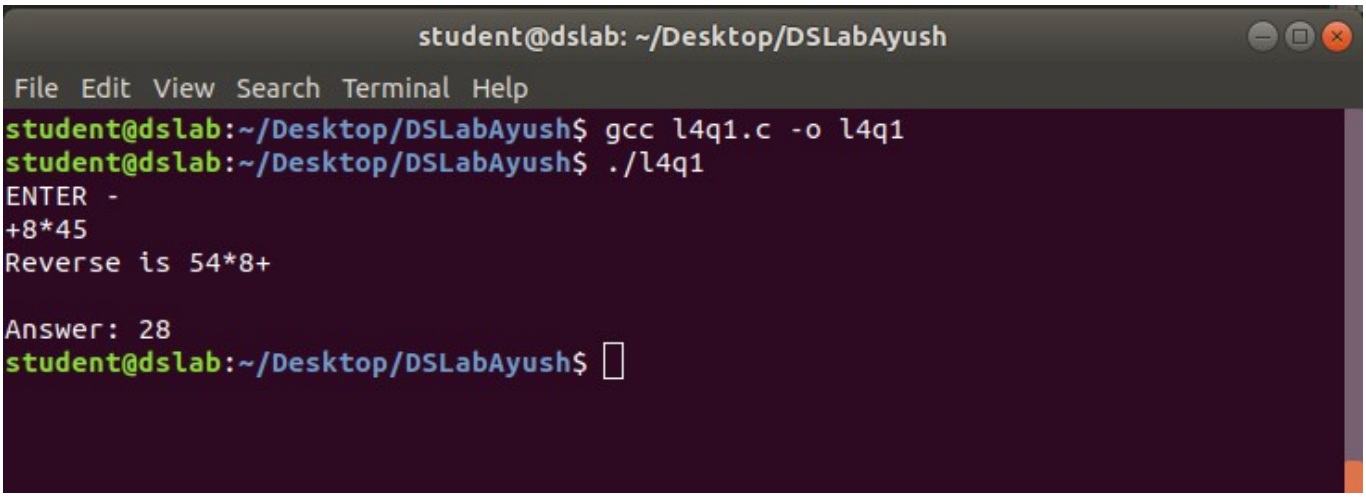
Ayush Goyal
190905522

DSA Lab 4 (Session 2)

Q1) Evaluate a given prefix expression using stack.

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

#define STACK_SIZE 50
#define EXPR_SIZE 50

typedef enum{lparan,rparan,plus,minus,times,divide,mod,eos,operand}PRECEDENCE;

int stack[STACK_SIZE];
char expr[EXPR_SIZE];

void push(int *top,int item){
	stack[++(*top)]=item;
}

int pop(int *top){
	return stack[(*top)--];
}

PRECEDENCE get_token(char *symbol,int *n){
	*symbol = expr[(*n)++];
	switch(*symbol){
		case '+':return plus;
		case '-':return minus;
		case '*':return times;
		case '/':return divide;
		case '%':return mod;
		case '(':return lparan;
		case ')':return rparan;
		case '\0':return eos;
		default :return operand;
	}
}

int eval(){
	PRECEDENCE token;
	char symbol;
	int n=0,c;
	int op1,op2;
	int top=-1;
	token = get_token(&symbol,&n);
	while(token!=eos){
		if(token==operand){
			c = symbol -'0';
			push(&top,c);
```

```
		}
		else{
			op2 = pop(&top);
			op1 = pop(&top);
			if(token==plus){
				push(&top,op2+op1);
			}
			else if(token==minus){
				push(&top,op2-op1);
			}
			else if(token==times){
				push(&top,op2*op1);
			}
			else if(token==divide){
				push(&top,op2/op1);
			}
			else if(token==mod){
				push(&top,op2%op1);
			}
		}
		token = get_token(&symbol,&n);
		//printf("%d\n",stack[top]);
	}
	return pop(&top);
}

int main(){
	char ex[50];
	int i,j=0;
	printf("ENTER -\n");
	scanf("%s",ex);
	for(i=strlen(ex)-1;i>=0;i--){
		expr[j]=ex[i];
		j++;
	}
	expr[j]='\0';
	printf("Reverse is %s\n",expr);
	printf("\nAnswer: %d\n",eval());
	return 0;
}
```



Q2) Convert an infix expression to prefix.

```
#define size 50
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
	char s[size];
	int top;
}STACK;


void push(STACK *s1,char elem){
	s1->top++;
	s1->s[s1->top]=elem;
}

char pop(STACK *s1){
	return s1->s[s1->top--];
}

int pre(char elem){
	switch(elem){
		case '#':return 0;
		case '(':return 1;
		case '+':
		case '-':return 2;
		case '*':
		case '/':return 3;
		case '$':return 4;
	}
}


int main(){
	STACK s1;
	s1.top=-1;
	char infix[50], prefix[50], revinf[50], revpref[50], elem, temp, ch;
	int i=0,k=0;
	printf("Enter infix expression\n");
	scanf("%s",infix);
	int len = strlen(infix);
	push(&s1,'#');
	for(i=len-1;i>=0;i--){
		revinf[k]=infix[i];
		k++;
	}
	revinf[k]='\0';
```

```
	i=0,k=0;
	while((ch=revinf[i++])!='\0'){
		switch(ch){
			case ')':push(&s1,ch);
				printf("PUSH %c\n",ch);
				break;
			case '(':while(s1.s[s1.top]!=')'){
					temp=pop(&s1);
					prefix[k++]=temp;
				}
				elem=pop(&s1);
				break;
			case '+':
			case '-':
			case '*':
			case '/': while(pre(s1.s[s1.top])>pre(ch)){
					temp=pop(&s1);
					prefix[k++]=temp;
					printf("POP and APPEND %c\n",temp);
				 }
				 push(&s1,ch);
				 printf("PUSH %c\n", ch);
				 break;
			case '$':if(s1.s[s1.top]=='$'){
					temp=pop(&s1);
					prefix[k++]=temp;
					printf("POP and APPEND %c\n",temp);
				}
				push(&s1,'$');
				break;
			default: prefix[k++]=ch;
		}
	}
	int m=0;
	while(s1.s[s1.top]!='#'){
		temp=pop(&s1);
		prefix[k++]=temp;
	}
	prefix[k]='\0';
	for(i=len-1;i>=0;i--){
		revpref[m]=prefix[i];
		m++;
	}
	revpref[m]='\0';
	printf("\n\nGiven Infix Expn: %s  Prefix Expn: %s\n",infix,revpref);
	return 0;
}
```

```
student@dslab: ~/Desktop/DSLabAyush
File  Edit  View  Search  Terminal  Help
student@dslab:~/Desktop/DSLabAyush$ gcc l4q2.c -o l4q2
student@dslab:~/Desktop/DSLabAyush$ ./l4q2
Enter infix expression
a*b+c
PUSH +
PUSH *


Given Infix Expn: a*b+c  Prefix Expn: +*abc
student@dslab:~/Desktop/DSLabAyush$ ./l4q2
Enter infix expression
a+b*c
PUSH *
POP and APPEND *
PUSH +


Given Infix Expn: a+b*c  Prefix Expn: +a*bc
student@dslab:~/Desktop/DSLabAyush$
```

Q3) Implement two stacks in an array.

```
 #include <stdio.h>
 #define SIZE 10
int ar[SIZE];
int top1 = -1;
int top2 = SIZE;

void push_stack1 (int data){
    if(top1 < top2 - 1){
                ar[++top1] = data;
    }
    else{
        printf ("Stack Full! Cannot Push\n");
    }
}

void push_stack2 (int data){
        if(top1 < top2 - 1){
                ar[--top2] = data;
        }
        else{
                printf("Stack Full! Cannot Push\n");
        }
}

void pop_stack1 (){
        if (top1 >= 0){
                int popped_value = ar[top1--];
```

```
		printf("%d is being popped from Stack 1\n", popped_value);
	}
	else{
		printf ("Stack Empty! Cannot Pop\n");
	}
}

void pop_stack2 (){
	if (top2 < SIZE) {
		int popped_value = ar[top2++];
		printf ("%d is being popped from Stack 2\n", popped_value);
	}
	else{
		printf ("Stack Empty! Cannot Pop\n");
	}
}

void print_stack1 (){
	int i;
	for (i = top1; i >= 0; --i) {
		printf ("%d ", ar[i]);
	}
	printf ("\n");
}

void print_stack2 (){
	int i;
	for (i = top2; i < SIZE; ++i){
		printf ("%d ", ar[i]);
	}
	printf ("\n");
}

int main(){
	int ar[SIZE];
	int i;
	int num_of_ele;
	printf ("We can push a total of 10 values\n");
	for (i = 0; i <= 6; ++i){
		push_stack1 (i);
		printf ("Value Pushed in Stack 1 is %d\n", i);
	}
	for (i = 1; i <= 4; ++i){
		push_stack2 (i);
		printf ("Value Pushed in Stack 2 is %d\n", i);
	}
	//Print Both Stacks
	print_stack1 ();
	print_stack2 ();
	//Pushing on Stack Full
	printf ("Pushing Value in Stack 1 is %d\n", 11);
	push_stack1 (11);
```

```
        //Popping All Elements From Stack 1
        num_of_ele = top1 + 1;
        while (num_of_ele){
                pop_stack1 ();
                --num_of_ele;
        }
        //Trying to Pop From Empty Stack
        pop_stack1 ();
        return 0;
}
```