LAB SUBMISSION – 2 – 19BCE1221 – DSA LAB

1. Write a program to implement stack using structure. Use push() and pop() functions to insert elements onto and delete from the stack.

struct stack

{

int items[10];

int top;

};

Example:

Input: 2 4 5 6 7 8

Output: 8 7 6 5 4 2

Push(S,10)

Output: 10 8 7 6 5 4 2

Pop()

Output: 8 7 6 5 4 2

Code:

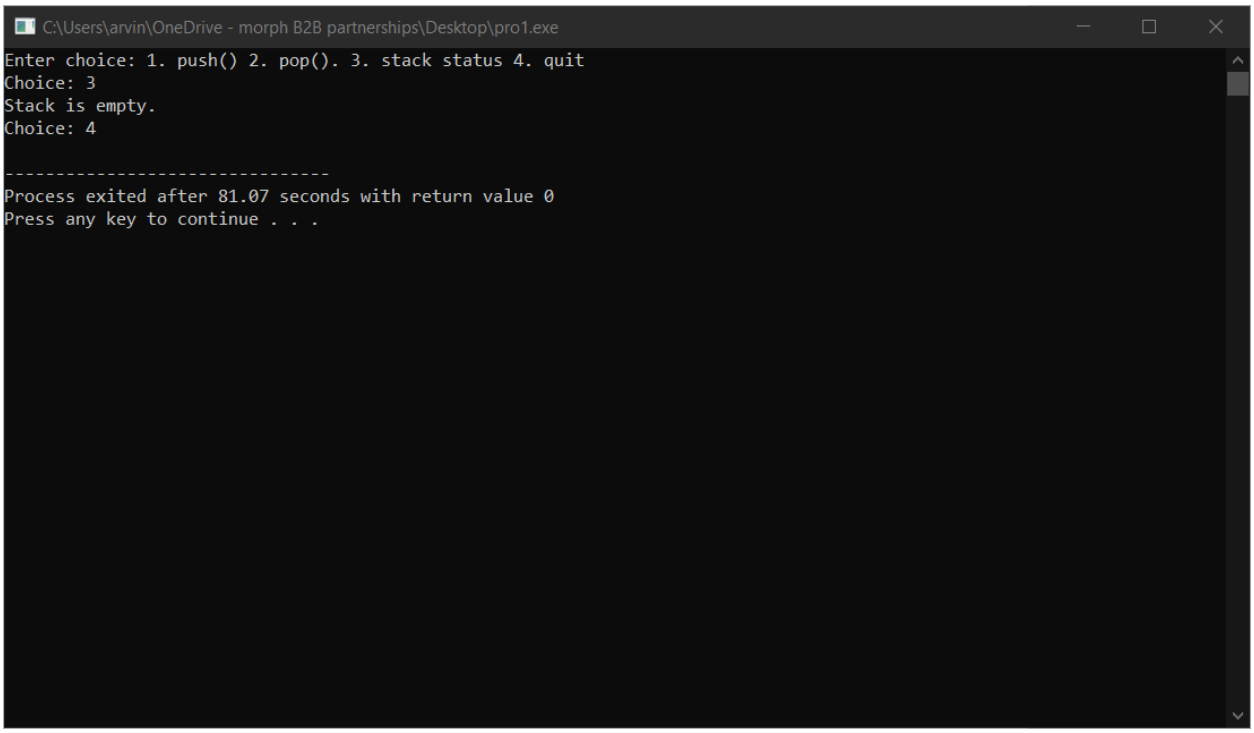
```
#include<stdio.h>
const int SIZE=5;
struct stack
{
	int items[SIZE];
	int top;
};
void initialise(struct stack *st)
```

```
{ st->top=-1;}
bool isfull(struct stack *st)
{ return(st->top==SIZE-1);}
bool isempty(struct stack *st)
{ return(st->top==-1); }
void push(struct stack *st, int value)
{
	if(!isfull(st))
	{
		st->top++;
		st->items[st->top]=value;
	}
	else
	{ printf("Stack Overflow\n"); }
}
void pop(struct stack *st)
{
	if(!isempty(st))
	{ st->top--;}
	else
	{ printf("Stack Underflow\n"); }
}
void stackStatus(struct stack *st)
{
	if(st->top==-1)
		printf("Stack is empty.\n");
```

```
	for(int i=st->top; i>=0; i--)
		printf("%d -> %d\n", i+1, st->items[i]);
}
int main()
{
	struct stack *s;
	initialise(s);
	printf("Enter choice: 1. push() 2. pop(). 3. stack status 4. quit\nChoice: ");
	int choice, value;
	scanf("%d", &choice);
	do
	{
		if(choice==1)
		{
			printf("Enter value to be pushed: ");
			scanf("%d", &value);
			push(s, value);
		}
		else if(choice==2)
		{ pop(s); }
		else if(choice==3)
		{stackStatus(s); }
		printf("Choice: ");
		scanf("%d", &choice);
	}while(choice!=4);
	return(0);
```

}

Checking stack status when stack is empty



Pushing elements and attempt to push in full stack

```
C:\Users\arvin\OneDrive - morph B2B partnerships\Desktop\pro1.exe
Enter choice: 1. push() 2. pop(). 3. stack status 4. quit
Choice: 1
Enter value to be pushed: 10
Choice: 1
Enter value to be pushed: 20
Choice: 1
Enter value to be pushed: 30
Choice: 1
Enter value to be pushed: 40
Choice: 1
Enter value to be pushed: 50
Choice: 3
5 -> 50
4 -> 40
3 -> 30
2 -> 20
1 -> 10
Choice: 1
Enter value to be pushed: 60
Stack Overflow
Choice: 4

--------------------------------
Process exited after 25.39 seconds with return value 0
Press any key to continue . . .
```

Popping from stack and popping from empty stack:

```
C:\Users\arvin\OneDrive - morph B2B partnerships\Desktop\pro1.exe
Enter choice: 1. push() 2. pop(). 3. stack status 4. quit
Choice: 1
Enter value to be pushed: 10
Choice: 1
Enter value to be pushed: 20
Choice: 3
2 -> 20
1 -> 10
Choice: 2
Choice: 3
1 -> 10
Choice: 2
Choice: 3
Stack is empty.
Choice: 2
Stack Underflow
Choice: 4

--------------------------------
Process exited after 38.24 seconds with return value 0
Press any key to continue . . .
```

2. Write a program to verify balancing parentheses in an expression. Print ‘1’ if the expression is balanced and ‘-1’ if the expression is unbalanced.

Example:

Input: ((()))

Output: 1

Input: (((())))

Output: -1

Code:

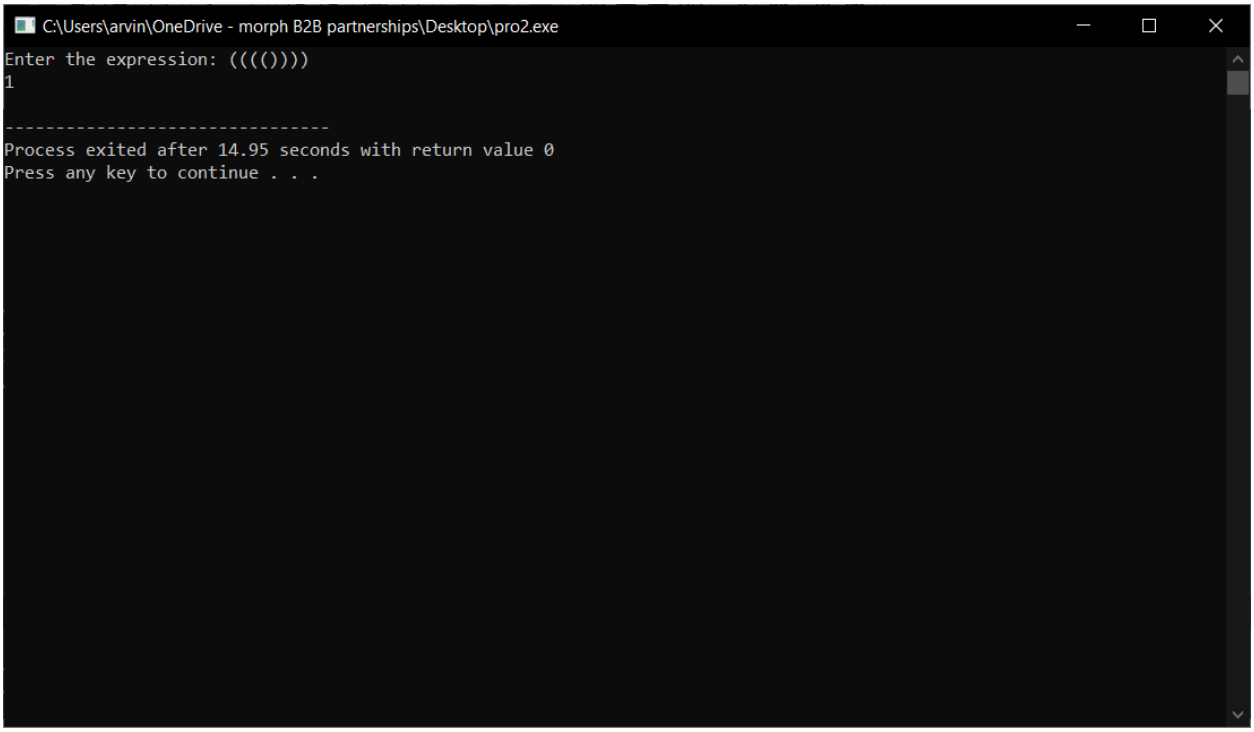
```
#include <stdio.h>
#include <stdlib.h>
#define bool int
struct s {
	char data;
	struct s* next;
};
void push(struct s** top_ref, int new_data)
{
	struct s* node = (struct s*)malloc(sizeof(struct s));
	if (node == NULL) {
		printf("Stack overflow\n");
		exit(0);
	}
	node->data = new_data;
	node->next = (*top_ref);
	(*top_ref) = node;
}
int pop(struct s** top_ref)
```

```
{
	char res;
	struct s* top;
	if (*top_ref == NULL) {
		printf("Stack overflow\n");
		exit(0);
	}
	else {
		top = *top_ref;
		res = top->data;
		*top_ref = top->next;
		free(top);
		return res;
	}
}
bool isMatchingPair(char character1, char character2)
{
	if (character1 == '(' && character2 == ')')
		return 1;
	else
		return 0;
}
bool areParenthesisBalanced(char exp[])
{
	int i = 0;
	struct s* stack = NULL;
```

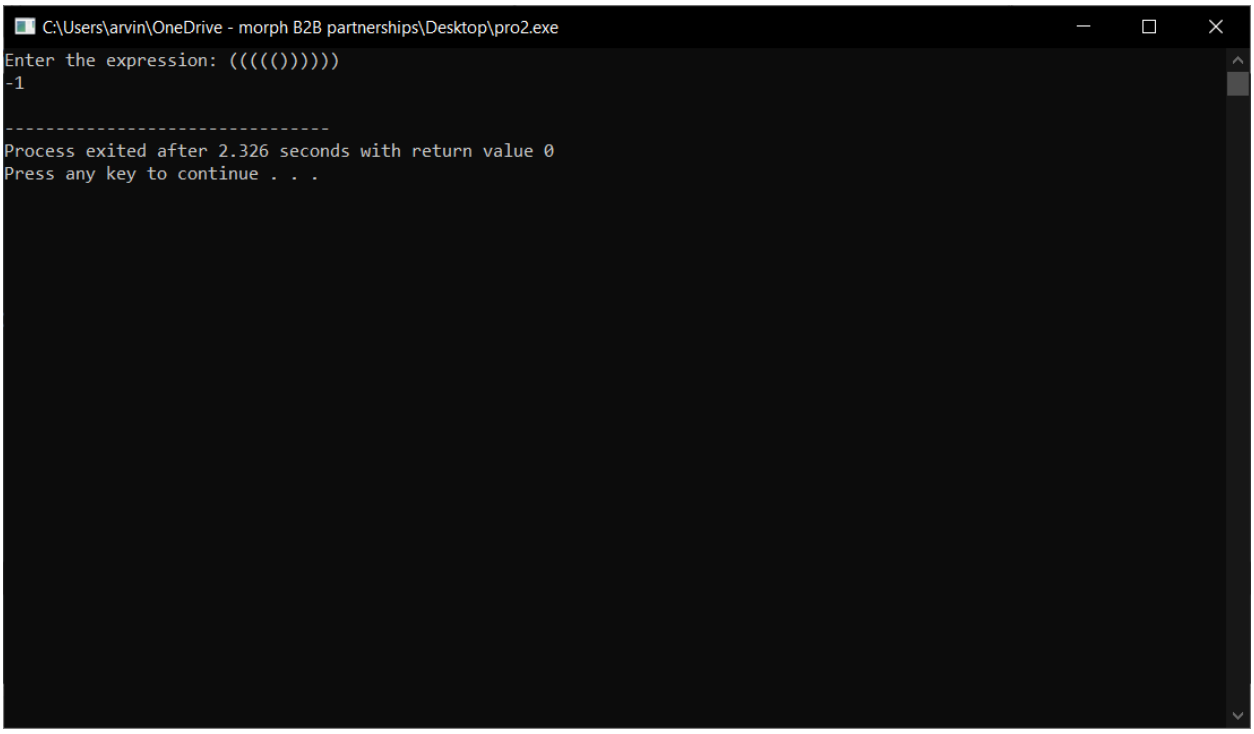
```
	while (exp[i]) {
		if (exp[i] == '{' || exp[i] == '(' || exp[i] == '[')
			push(&stack, exp[i]);
		if (exp[i] == '}' || exp[i] == ')' || exp[i] == ']') {
			if (stack == NULL)
				return 0;
			else if (!isMatchingPair(pop(&stack), exp[i]))
				return 0;
		}
		i++;
	}
	if (stack == NULL)
		return 1;
	else
		return 0;
}
int main()
{
	char exp[100];
	printf("Enter the expression: ");
	scanf("%s", &exp);
	if (areParenthesisBalanced(exp))
		printf("1 \n");
	else
		printf("-1 \n");
	return 0;
```

}

Screenshot of balanced string:



Screenshot of not balanced string:



3. Write a program to convert an infix notation into a postfix notation.

Example:

Input: 6*(5+(2+3)*8+3)

Output: 6 5 2 3 + 8 * + 3 + *

Code:

```
#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

const int SIZE=100;

char stack[SIZE];

int top = -1;
```

```
void push(char item)
{
	if(top >= SIZE-1)
	{
		printf("\nStack Overflow.");
	}
	else
	{
		top = top+1;
		stack[top] = item;
	}
}
char pop()
{
	char item ;
	if(top <0)
	{
		printf("stack under flow: invalid infix expression");
		getchar();
		exit(1);
	}
	else
	{
		item = stack[top];
		top = top-1;
		return(item);
```

```
        }
}
int is_operator(char symbol)
{
        if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-')
        {
                return 1;
        }
        else
        {
        return 0;
        }
}
int precedence(char symbol)
{
        if(symbol == '^')
        {
                return(3);
        }
        else if(symbol == '*' || symbol == '/')
        {
                return(2);
        }
        else if(symbol == '+' || symbol == '-')
        {
                return(1);
```

```
        }
        else
        {
                return(0);
        }
}
void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
        int i, j;
        char item;
        char x;
        push('(');
        strcat(infix_exp,")");
        i=0;
        j=0;
        item=infix_exp[i];
        while(item != '\0')
        {
                if(item == '(')
                {
                        push(item);
                }
                else if( isdigit(item) || isalpha(item))
                {
                        postfix_exp[j] = item;
                        j++;
```

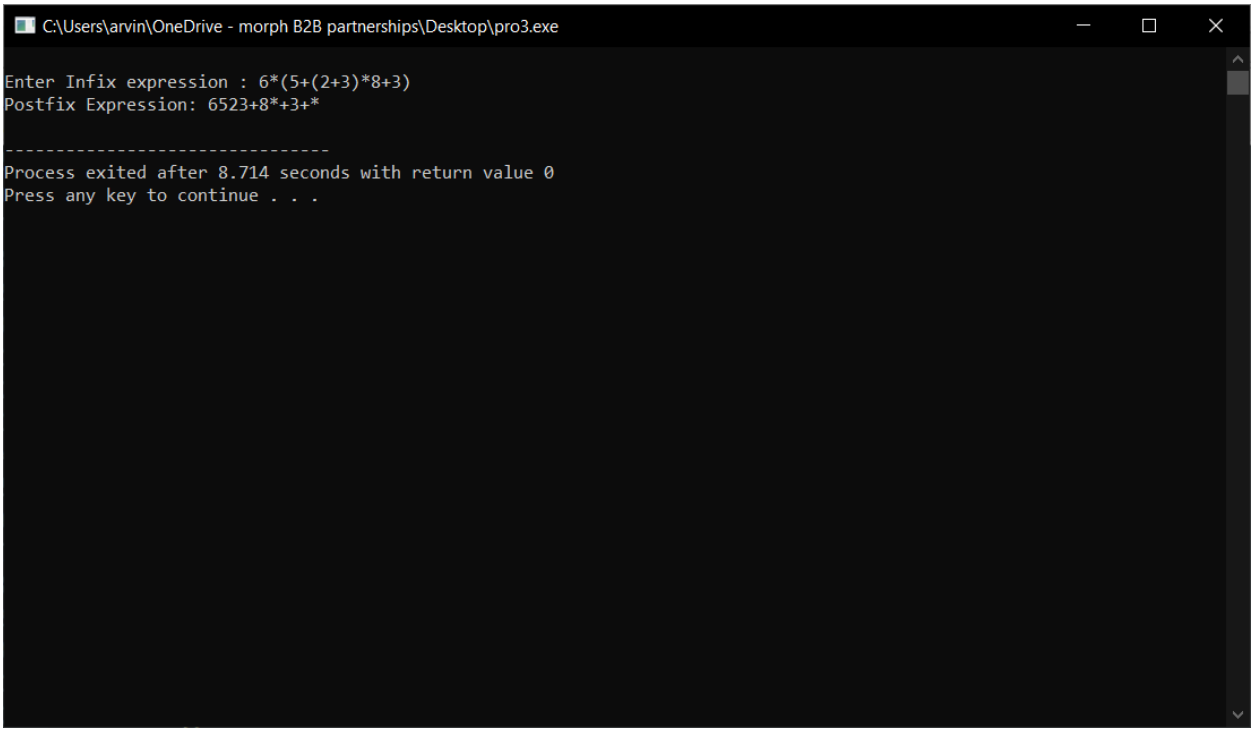
```
}
else if(is_operator(item) == 1)
{
        x=pop();
        while(is_operator(x) == 1 && precedence(x)>= precedence(item))
        {
                postfix_exp[j] = x;
                j++;
                x = pop();
        }
        push(x);
        push(item);
}
else if(item == ')')
{
        x = pop();
        while(x != '(')
        {
                postfix_exp[j] = x;
                j++;
                x = pop();
        }
}
else
{
        printf("\nInvalid infix Expression.\n");
```

```
                getchar();
                exit(1);
            }
            i++;
            item = infix_exp[i];
        }
        if(top>0)
        {
            printf("\nInvalid infix Expression.\n");
            getchar();
            exit(1);
        }
        if(top>0)
        {
            printf("\nInvalid infix Expression.\n");
            getchar();
            exit(1);
        }
        postfix_exp[j] = '\0';
}
int main()
{
        char infix[SIZE], postfix[SIZE];
        printf("\nEnter Infix expression : ");
        gets(infix);
        InfixToPostfix(infix,postfix);
```

```
        printf("Postfix Expression: ");
        puts(postfix);

        return 0;
}
```

Output Screenshot:



4. Write a program to evaluate a postfix notation.

Example:

Input: 6 5 2 3 + 8 * + 3 + *

Output: 288

Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
struct Stack
{
    int top;
    unsigned capacity;
    int* array;
};
struct Stack* createStack( unsigned capacity )
{
    struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack));

    if (!stack) return NULL;

    stack->top = -1;
    stack->capacity = capacity;
    stack->array = (int*) malloc(stack->capacity * sizeof(int));
```

```
    if (!stack->array) return NULL;


    return stack;
}
int isEmpty(struct Stack* stack)
{
    return stack->top == -1 ;
}
char peek(struct Stack* stack)
{
    return stack->array[stack->top];
}
char pop(struct Stack* stack)
{
    if (!isEmpty(stack))
        return stack->array[stack->top--] ;
    return '$';
}
void push(struct Stack* stack, char op)
{
    stack->array[++stack->top] = op;
}
int evaluatePostfix(char* exp)
{
    struct Stack* stack = createStack(strlen(exp));
    int i;
```

```
    if (!stack) return -1;
    for (i = 0; exp[i]; ++i)
    {
        if (isdigit(exp[i]))
            push(stack, exp[i] - '0');
        else
        {
            int val1 = pop(stack);
            int val2 = pop(stack);
            switch (exp[i])
            {
            case '+': push(stack, val2 + val1); break;
            case '-': push(stack, val2 - val1); break;
            case '*': push(stack, val2 * val1); break;
            case '/': push(stack, val2/val1); break;
            }
        }
    }
    return pop(stack);
}
int main()
{
    char exp[100];
    printf("Enter the postfix expression: ");
          scanf("%s", &exp);
    printf ("Postfix evaluation: %d", evaluatePostfix(exp));
```

```
    return 0;
}
```

Output Screenshot: