

Assignment 6

DSA LAB

2029196

Adarsh Kumar

Q1. WAP Write a menu driven program to perform the following operations of a stack using array by using suitable user defined functions for each case.

a) Check if the stack is empty b) Display the contents of stack c) Push d) Pop.

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

int data[20], top = -1;

void push (int val)
{
    if (top == 20 - 1)
        printf("\n Overflow\n");
    else
    {
        top++;
        data[top] = val;
    }
}

void pop ()
{
    if (top == -1)
        printf("\n Underflow\n");
    else
        top--;
}

void display ()
{
    if (top == -1)
        printf("stack is empty");
    else
    {
        for (int i = top; i >= 0; i--)
            printf("%d ", data[i]);
        printf("\n");
    }
}

void check ()
{
    if (top == -1)
        printf("The stack is empty\n");
    else
        printf("The stack is not empty\n");
}
```

```

int main()
{
    int opt, n;
    while (1)
    {
        printf("\n 1 check if the stack is empty");
        printf("\n 2 Display the contents of stack");
        printf("\n 3 Push");
        printf("\n 4 pop");
        printf("\n 5 Exit");
        printf("\n\n Enter your choice: ");
        scanf("%d", &opt);
        switch (opt)
        {
            case 1: check();
                    break;
            case 2: display();
                    break;
            case 3: printf("\n Enter element to push: ");
                    scanf("%d", &n);
                    push(n);
                    break;
            case 4: pop();
                    break;
            case 5: exit(0);
                    break;
        }
    }
    return 0;
}

```

OUTPUT:-

```

1 Check if the stack is empty
2 Display the contents of stack
3 Push
4 Pop
5 Exit

```

```

Enter your choice: 2
stack is empty

```

```

1 Check if the stack is empty
2 Display the contents of stack
3 Push
4 Pop
5 Exit

```

```

Enter your choice: 3

```

```

Enter Element to push: 23

```

```

1 Check if the stack is empty
2 Display the contents of stack
3 Push
4 Pop
5 Exit

```

```

Enter your choice: 2
23

```


Q2. WAP Write a menu driven program to perform the following operations of a stack using linked list by using suitable user defined functions for each case..

a) Check if the stack is empty b) Display the contents of stack c) Push d) Pop.

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

struct node
{
    int data;
    struct node *next;
};

typedef struct
{
    struct node *top;
} STACK;

void init (STACK *s)
{
    s->top = NULL;
}

void push (STACK *s, int val)
{
    struct node *current = malloc (sizeof (struct node));
    if (current == NULL)
        printf ("Overflow");
    current->data = val;
    current->next = s->top;
    s->top = current;
}

void pop (STACK *s, int *val)
{
    if (s->top == NULL)
        printf ("Underflow");

    struct node *ptr;
    *val = s->top->data;
    ptr = s->top;
    s->top = ptr->next;
    free (ptr);
}
```



```

void display (STACK s)
{
    if (s.top == NULL)
        printf ("The stack is empty \n");
    else
    {
        while (s.top != NULL)
        {
            printf ("%d", s.top->data);
            s.top = s.top->next;
        }
        printf ("\n");
    }
}

void check (STACK s)
{
    if (s->top == NULL)
        printf ("The List is empty \n");
    else
    {
        printf ("The list is not empty: ");
        display (s);
    }
}

int main ()
{
    STACK s1;
    int (*s1);
    int opt, n, p;
    while (1) {
        printf ("\n 1 Check if the stack is empty");
        printf ("\n 2 Display the contents of stack");
        printf ("\n 3 push");
        printf ("\n 4 pop");
        printf ("\n 5 Exit");
        printf ("\n Enter your choice:");
        scanf ("%d", &opt);
        switch (opt) {
            case 1: check (&s1);
                    break;
            case 2: display (s1);
                    break;
            case 3: printf ("\n Enter the element:");
                    scanf ("%d", &n);
                    push (&s1, n);
                    break;
            case 4: pop (&s1, &p);
                    printf ("The value %d is deleted \n", p);
                    break;
            case 5: exit (0);
                    break;
        }
    }
    return 0;
}

```

OUTPUT:-

```
1 Check if the stack is empty
2 Display the contents of stack
3 Push
4 Pop
5 Exit
```

Enter your choice: 3

Enter Element to push: 45

```
1 Check if the stack is empty
2 Display the contents of stack
3 Push
4 Pop
5 Exit
```

Enter your choice: 4

The value 45 is deleted

```
1 Check if the stack is empty
2 Display the contents of stack
3 Push
4 Pop
5 Exit
```

Enter your choice: 4

Underflow

Q3 WAP to convert an infix expression into its equivalent postfix notation.

```
#include <stdio.h>

char stack[100];
int top = -1;

void push(char x)
{
    stack[++top] = x;
}

char pop()
{
    if (top == -1)
        return -1;
    else
        return stack[top--];
}

int precedence(char x)
{
    if (x == '(')
        return 0;
    if (x == '+' || x == '-')
        return 1;
```

```

    if (x == '*' || x == '/')
        return 2;
    return 0;
}

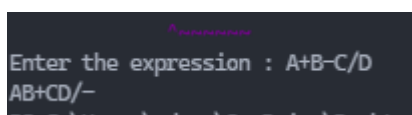
int main()
{
    char exp[100];
    char *e, x;
    printf("Enter the expression : ");
    scanf("%s", exp);
    e = exp;

    while (*e != '\0')
    {
        if (isalnum(*e))
            printf("%c ", *e);
        else if (*e == '(')
            push(*e);
        else if (*e == ')')
        {
            while ((x = pop()) != '(')
                printf("%c ", x);
        }
        else
        {
            while (precedence(stack[top]) >= precedence(*e))
                printf("%c ", pop());
            push(*e);
        }
        e++;
    }

    while (top != -1)
    {
        printf("%c", pop());
    }
    return 0;
}

```

OUTPUT:-



```

Enter the expression : A+B-C/D
AB+CD/-

```

Q4. WAP to convert an infix expression into its equivalent prefix notation.

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

struct Stack
{
    int top;
    int maxSize;
    int *array;
};

struct Stack *create(int max)
{
    struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
    stack->maxSize = max;
    stack->top = -1;
    stack->array = (int *)malloc(stack->maxSize * sizeof(int));
    return stack;
}

int isFull(struct Stack *stack)
{
    if (stack->top == stack->maxSize - 1)
    {
        printf("Will not be able to push maxSize reached\n");
    }
    return stack->top == stack->maxSize - 1;
}

int isEmpty(struct Stack *stack)
{
    return stack->top == -1;
}

void push(struct Stack *stack, int item)
{
    if (isFull(stack))
        return;
    stack->array[++stack->top] = item;
}

int pop(struct Stack *stack)
```

```

{
    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[stack->top--];
}

int peek(struct Stack *stack)
{
    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[stack->top];
}

int checkIfOperand(char ch)
{
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}

int precedence(char ch)
{
    switch (ch)
    {
        case '+':
        case '-':
            return 1;

        case '*':
        case '/':
            return 2;

        case '^':
            return 3;
    }
    return -1;
}

int getPostfix(char *expression)
{
    int i, j;
    struct Stack *stack = create(strlen(expression));
    if (!stack)
        return -1;

    for (i = 0, j = -1; expression[i]; ++i)

```



```

{
    if (checkIfOperand(expression[i]))
        expression[++j] = expression[i];
    else if (expression[i] == '(')
        push(stack, expression[i]);
    else if (expression[i] == ')')
    {
        while (!isEmpty(stack) && peek(stack) != '(')
            expression[++j] = pop(stack);
        if (!isEmpty(stack) && peek(stack) != '(')
            return -1;
        else
            pop(stack);
    }
    else
    {
        while (!isEmpty(stack) && precedence(expression[i]) <= p
precedence(peek(stack)))
            expression[++j] = pop(stack);
        push(stack, expression[i]);
    }
}
while (!isEmpty(stack))
    expression[++j] = pop(stack);
expression[++j] = '\0';
}

void reverse(char *exp)
{
    int size = strlen(exp);
    int j = size, i = 0;
    char temp[size];

    temp[j--] = '\0';
    while (exp[i] != '\0')
    {
        temp[j] = exp[i];
        j--;
        i++;
    }
    strcpy(exp, temp);
}

void brackets(char *exp)

```

```

{
    int i = 0;
    while (exp[i] != '\0')
    {
        if (exp[i] == '(')
            exp[i] = ')';
        else if (exp[i] == ')')
            exp[i] = '(';
        i++;
    }
}

void InfixtoPrefix(char *exp)
{
    int size = strlen(exp);
    reverse(exp);
    brackets(exp);
    getPostfix(exp);
    reverse(exp);
}

int main()
{
    printf("The infix is: ");

    char expression[] = "A+B-C/D";
    printf("%s\n", expression);
    InfixtoPrefix(expression);

    printf("The prefix is: ");
    printf("%s\n", expression);

    return 0;
}

```

OUTPUT:-

```

Drive\Desktop\3 rd Sem\DA Lab\DA
\q4 }
The infix is: A+B-C/D
The prefix is: +A-B/CD
PS C:\Users\adars\OneDrive\Desktop\

```

Q5. Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e.,),], or }) of the exact same type. There are three types of matched pairs of brackets: [], {}, and (). A matching pair of brackets is not balanced if the set of brackets it encloses are not matched. WAP to determine whether the input sequence of brackets is balanced or not. If a string is balanced, it print YES on a new line; otherwise, print NO on a new line.

Example: Input: {[()]} and Output: YES

Input: {[()]} and Output: NO.

```
int same(char a, char b)
{
    if (a == '[' && b == ']')
        return 1;
    if (a == '{' && b == '}')
        return 1;
    if (a == '(' && b == ')')
        return 1;
    return 0;
}

int check(char *a)
{
    char stack[1001], top = -1;
    for (int j = 0; j < strlen(a); j++)
    {
        if (a[j] == '[' || a[j] == '{' || a[j] == '(')
            stack[++top] = a[j];
        if (a[j] == ']' || a[j] == '}' || a[j] == ')')
        {
            if (top == -1)
            {
                return 0;
            }
            else
            {
                if (!same(stack[top--], a[j]))
                {
                    return 0;
                }
            }
        }
    }
    if (top != -1)
    {
        return 0;
    }
    return 1;
}

int main()
{
    char a[1001];
    scanf("%s", a);
```

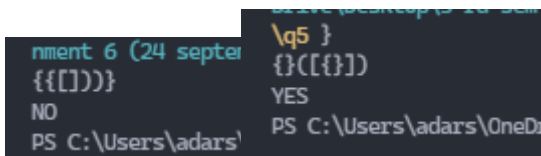


```

int valid = check(a);
if (valid == 1)
    printf("YES\n");
else
    printf("NO\n");
return 0;
}

```

OUTPUT:-



```

nment 6 (24 septer
{{[()]})
NO
PS C:\Users\adars\
\q5 }
{}([{}])
YES
PS C:\Users\adars\OneD

```

Q6. WAP to reverse a stack with using extra stack.

```

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

#define MAX 100

int top = -1;
int item;
char stack_string[MAX];

void pushChar(char item)
{
    if (isFull())
    {
        printf("\nStack is FULL !!!\n");
        return;
    }
    top = top + 1;
    stack_string[top] = item;
}

char popChar()
{
    if (isEmpty())
    {
        printf("\nStack is EMPTY!!!\n");
        return 0;
    }
    item = stack_string[top];
    top = top - 1;
    return item;
}

```

```

}

int isEmpty()
{
    if (top == -1)
        return 1;
    else
        return 0;
}

int isFull()
{
    if (top == MAX - 1)
        return 1;
    else
        return 0;
}

int main()
{
    char str[MAX];
    int i;

    printf("Input a string: ");
    scanf("%[^\n]s", str);

    for (i = 0; i < strlen(str); i++)
        pushChar(str[i]);

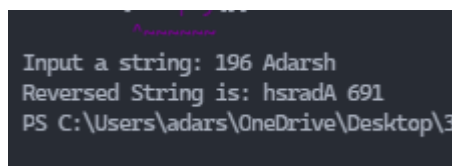
    for (i = 0; i < strlen(str); i++)
        str[i] = popChar();

    printf("Reversed String is: %s\n", str);

    return 0;
}

```

OUTPUT:-



```

Input a string: 196 Adarsh
Reversed String is: hsradA 691
PS C:\Users\adars\OneDrive\Desktop\3

```

Q7. WAP to sort the elements inside a stack using only push and pop operation. Any number of additional stacks may be used.

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

struct node
{
    int data;
    struct node *next;
};

typedef struct
{
    struct node *top;
} STACK;

void init(STACK *s)
{
    s->top = NULL;
}

void push(STACK *s, int val)
{
    struct node *current;
    current = malloc(sizeof(struct node));
    if (current == NULL)
        printf("Overflow");
    current->data = val;
    current->next = s->top;
    s->top = current;
}

void pop(STACK *s, int *val)
{
    if (s->top == NULL)
        printf("Underflow");

    struct node *ptr;
    *val = s->top->data;
    ptr = s->top;
    s->top = ptr->next;
    free(ptr);
}

void insort(STACK *s, int val)
{
}
```



```

    if (isempty(*s))
        push(s, val);
    else
    {
        int d;
        pop(s, &d);
        if (d < val)
        {
            insert(s, val);
            push(s, d);
        }
        else
        {
            push(s, d);
            push(s, val);
        }
    }
}

int isempty(STACK s)
{
    if (s.top == NULL)
        return 1;
    else
        return 0;
}

void sort(STACK *s)
{
    if (isempty(*s) == 0)
    {
        int d;
        pop(s, &d);
        sort(s);
        insert(s, d);
    }
}

void display(STACK s)
{
    if (s.top == NULL)
        printf("The list is empty\n");
    else
    {

```

```

        while (s.top != NULL)
        {
            printf("%d ", s.top->data);
            s.top = s.top->next;
        }
        printf("\n");
    }
}

int main()
{
    STACK s1;
    init(&s1);
    push(&s1, 5);
    push(&s1, 21);
    push(&s1, 9);
    push(&s1, 1);

    display(s1);

    sort(&s1);

    display(s1);

    return 0;
}

```

OUTPUT:-

```

1 9 21 5
1 5 9 21

```