Week 4

1)stack_operations.h

```
# define MAX 10
# define true 1
# define false 0

typedef struct{
	int a[MAX];
	int top;
}Stack;

void push(Stack *s, int n);
int pop(Stack *s);
int isEmpty(Stack *s);
int isFull(Stack *s);
void display(Stack *s);

void push(Stack *s, int n){
	if (!isFull(s)){
		// s->top++;
		s->a[s->top++] = n;
	}
}

int pop(Stack *s){
	if(!isEmpty(s)){
		s->top--;
		return s->a[s->top];
		// return(s->arr[s->top--]);
	}
}

int isEmpty(Stack *s){
	if (s->top==-1)
		return(true);
	else
		return(false);
}

int isFull(Stack *s){
	if(s->top == MAX-1)
		return true;
	return false;
}

void display(Stack *s){
	printf("Stack: \n");
	for(int i = 0; i < s->top; i++){
		printf("%c ", s->a[i]);
	}
```

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

l4q1.c

```
// Evaluate a given prefix expression using stack.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "stack_operations.h"

int eval(int a, int b, char x){
	if(x=='+')
		return(a+b);
	if(x=='-')
		return(a-b);
	if(x=='*')
		return(a*b);
	if(x=='/')
		return(a/b);
	if(x=='%')
		return(a%b);
}
int main(){
	char *str = (char *)calloc(10, sizeof(char));
	printf("Enter Expression\n");
	scanf("%s", str);
	Stack s; s.top = 0;
	// printf("%d\n", strlen(str));
	for(int i = strlen(str)-1; i >= 0; i--){
		// printf("%d\n", i);
		if(str[i]>='0' && str[i]<='9'){
			int d = str[i]-'0';
			// printf("%d\n", d);
			push(&s, d);
			// display(&s);
		}

		else{
			int a = pop(&s);
			int b = pop(&s);
			int value = eval(a, b, str[i]);
			push(&s, value);
		}
		// display(&s);

	}
	int ans = pop(&s);
	printf("Answer = %d\n", ans);

	return 0;
```

}

```
student@lplab-Lenovo-Product:~/Parth_Shukla_dsa/lab4$ ./l4q1
Enter Expression
+34
Answer = 7
student@lplab-Lenovo-Product:~/Parth_Shukla_dsa/lab4$ ./l4q1
Enter Expression
+3*45
Answer = 23
student@lplab-Lenovo-Product:~/Parth_Shukla_dsa/lab4$
```

2)
stack_operations_char.h

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

#define MAX 20
#define TRUE 1
#define FALSE 0

typedef struct stack
{
char item[MAX];
int top;
}stack;

int isEmpty(stack*);
int isFull(stack*);
int push(stack*, char);
char pop(stack*);
void display(stack*);
stack* init_stack();

int isEmpty(stack *s)
{
if(s->top == -1) return TRUE;
return FALSE;
}

int isFull(stack *s)
{
if(s->top == MAX - 1) return TRUE;
return FALSE;
}

int push(stack *s, char x)
{
if(isFull(s))
return FALSE;
s->item[++s->top] = x; return TRUE;
}
```

```
char pop(stack *s)
{
if(isEmpty(s)) return FALSE;
return(s->item[s->top--]);
}

void display(stack *s)
{
if(isEmpty(s)) return;
int i;
for(i = 0; i <= s->top; i++)
printf("%c ", s->item[i]);
printf("\n");
}

stack* init_stack()
{
stack* s = (stack *)malloc(sizeof(stack)); s->top = -1;
return s;
}
```

l4q2.c

```
// Convert an infix expression to prefix.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "stack_operations_char.h"

void Inf2Pref(char* , char*);
void Inf2Post(char*, char*);
int isOperand(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;
```

```
}

void Inf2Pref(char* src, char* dest)
{
	char* inf_rev = (char *)calloc(MAX, sizeof(char));
	char* post_rev = (char *)calloc(MAX, sizeof(char));
	int i;
	int j = 0;
	for(i = strlen(src) - 1; i > -1; i--)
	{
		if(src[i] == '(')
			inf_rev[j] = ')';
		else if(src[i] == ')')
			inf_rev[j] = '(';
		else
			inf_rev[j] = src[i];
		j++;
	}
	inf_rev[j] = '\0';
	char* post = (char *)calloc(MAX, sizeof(char));
	Inf2Post(inf_rev, post);
	j = 0;
	for(i = strlen(post) - 1; i > -1; i--)
	{
		post_rev[j] = post[i];
		j++;
	}
	post_rev[j] = '\0';
	strcpy(dest, post_rev);
}

void Inf2Post(char* src, char* dest)
{
	char* exp = (char *)calloc(MAX, sizeof(char));
	stack* s = init_stack();
	int i;
	int j = 0;
	for(i = 0; i < strlen(src); i++)
	{
		if(isOperand(src[i]))
		{
			exp[j] = src[i];
			j++;
		}
		else if(src[i] == '(')
		{
			push(s, src[i]);
		}
		else if(src[i] == ')')
		{
			while(isEmpty(s) == FALSE && s->item[s->top] != '(')
			{
```

```
                        exp[j] = pop(s);
                        j++;
                }
            pop(s);
            }
            else
            {
                while(isEmpty(s) == FALSE && precedence(src[i]) <= precedence(s-
>item[s->top]))
                {
                        exp[j] = pop(s);
                        j++;
                }
                push(s, src[i]);
            }
        }
        while(isEmpty(s) == FALSE)
        {
            exp[j] = pop(s); j++;
        }
        exp[j] == '\0';
        strcpy(dest, exp);
}

int main(int argc, char const *argv[])
{
        char* exp = (char *)calloc(MAX, sizeof(char));
        char* pref = (char *)calloc(MAX, sizeof(char));
        printf("Enter expression: ");
        scanf("%s", exp);
        Inf2Pref(exp, pref);
        printf("Prefix expression: %s\n", pref);
        return 0;
}
```

```
student@lplab-Lenovo-Product:~/Parth_Shukla_dsa/lab4$ ./l4q2
Enter expression: ab+-gf
The prefix expression is: +ab-gf
student@lplab-Lenovo-Product:~/Parth_Shukla_dsa/lab4$ ./l4q2
Enter expression: hf+-sgj*/
The prefix expression is: +hf-*sgj/
```

3)

```
// Implement two stacks in an array.

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

typedef struct TwoStacks
{
        int* arr;
```

```
	int size;
	int top1,
	top2;
}TwoStacks;

TwoStacks* init_Stack(int size)
{
	TwoStacks* s = (TwoStacks *)malloc(sizeof(TwoStacks));
	s->arr = (int *)calloc(size, sizeof(int));
	s->top1 = -1;
	s->top2 = size;
	return s;
}

void push1(TwoStacks* s, int n)
{
	if(s->top1 < s->top2 -1)
	{
		s->top1++;
		s->arr[s->top1] = n;
	}
	else
	{
		printf("Overflow\n");
		exit(0);
	}
}

void push2(TwoStacks* s, int n)
{
	if(s->top1 < s->top2 -1)
	{
		s->top2--;
		s->arr[s->top2] = n;
	}
	else
	{
		printf("Overflow\n");
		exit(1);
	}
}

int pop1(TwoStacks* s, int sz)
{
	if(s->top1 >= 0)
	{
		int x = s->arr[s->top1];
		s->top1--;
		return x;
	}

	else
```

```
	{
		printf("Underflow\n");
		exit(1);
	}
}

int pop2(TwoStacks* s, int size)
{
	if(s->top2 < size)
	{
		int x = s->arr[s->top2];
		s->top2++;
		return x;
	}
	else
	{
		printf("Underflow\n");
		exit(1);
	}
}

int main(int argc, char const *argv[])
{
	TwoStacks* s = init_Stack(5);
	printf("Pushing 1 to stack 1\n");
	push1(s, 1);
	printf("Pushing 2 to stack 2\n");
	push2(s, 2);
	printf("Pushing 3 to stack 2\n");
	push2(s, 3);
	printf("Pushing 4 to stack 1\n");
	push1(s, 4);
	printf("Pushing 5 to stack 2\n");
	push2(s, 5);
	printf("Popped Element from stack 1 is: %d\n", pop1(s, 5));
	printf("Popped Element from stack 1 is: %d\n", pop1(s, 5));
	printf("Pushing 6 to stack 2\n");
	push2(s, 6);
	printf("Popped Element from stack 2 is %d\n", pop2(s, 5));
	return 0;
}
```

```
student@lplab-Lenovo-Product:~/Parth_Shukla_dsa/lab4$ cc -o l4q3 l4q3.c
student@lplab-Lenovo-Product:~/Parth_Shukla_dsa/lab4$ ./l4q3
Pushing 1 to stack 1
Pushing 2 to stack 2
Pushing 3 to stack 2
Pushing 4 to stack 1
Pushing 5 to stack 2
Popped Element from stack 1 is: 4
Popped Element from stack 1 is: 1
Pushing 6 to stack 2
Popped Element from stack 2 is 6
```