WEEK -4

Lab Exercises:

Write a C program to:

1: Evaluate a given prefix expression using stack.

Code:

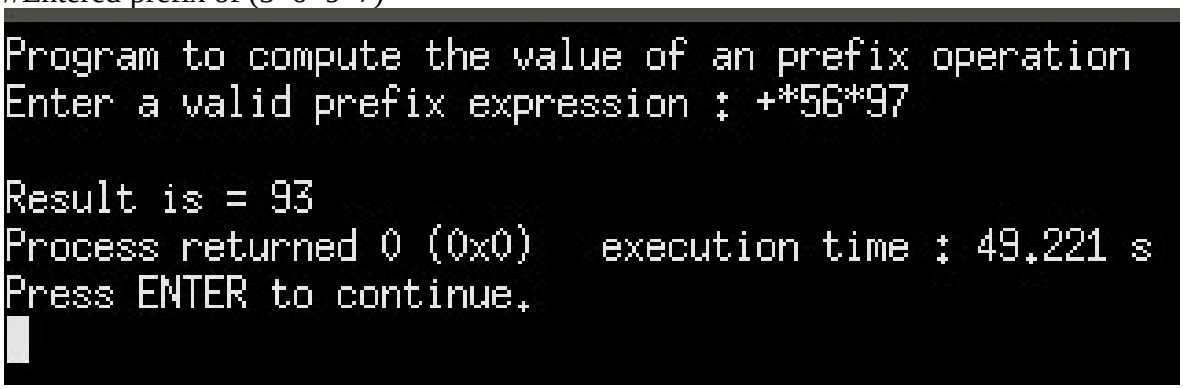
```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 200
#define EMPTY '\0'
typedef enum {
NO = 0,
YES = 1,
} BOOL;
BOOL isStackFull (int tos) {
if (tos == SIZE - 1)
return YES;
return NO;
}
BOOL isStackEmpty (int tos) {
if (tos == -1)
return YES;
return NO;
}
void push (int *stack, int item, int *tos) {
if (*tos == SIZE - 1)
return;
(*tos) += 1;
*(stack + (*tos)) = item;
}
int pop (int *stack, int *tos) {
if (*tos == -1)
return EMPTY;
return *(stack + ((*tos)--));
}
int output (char op, int a, int b) {
switch (op) {
case '+': return a+b;
case '-': return a-b;
case '*': return a*b;
```

```
case '/': return (int)(a/b);
case '$': return (int) a^b;
default : return 0;
}
}
int indexOf (char ch, char *str) {
char *ptr = strchr(str, ch);
if (ptr)
return (int)(ptr - str);
return -1;
}
BOOL operatorOn (char op) {
if (indexOf(op, "+-*/$") != -1)
return YES;return NO;
}
BOOL isNumber (char op) {
if (op >= '0' && op <= '9')
return YES;
return NO;
}
BOOL isAlphabet (char op) {
if ((op >= 'A' && op <= 'Z') || (op >= 'a' && op <= 'z'))
return YES;
return NO;
}
int numericValue (char ch) {
return (int)(ch - 48);
}
int prefix (char * exp) {
int tos = -1;
int *stack = (int *)calloc(SIZE, sizeof(int));
int l = (int)strlen(exp), i;
for (i = l - 1; i >= 0; --i) {
char z = *(exp + i);
if (isNumber(z))
push(stack, numericValue(z), &tos);
else if (isAlphabet(z)) {
int num;
printf("\n\t\t\t\tEnter the value of '%c': ", z);
scanf("%d", &num);
push(stack, num, &tos);
}
else if (operatorOn(z) && tos > 0) {
int a = pop(stack, &tos);
int b = pop(stack, &tos);
int res = output(z, a, b);
push(stack, res, &tos);
}
else
return EMPTY;
```

```
}
if (tos == 0)
return *stack;
return EMPTY;
}
int main(int argc, const char * argv[]) {
char *str = (char *)calloc(SIZE, sizeof(char));
printf("\nPogram to compute the value of an prefix operation");
printf("\nEnter a valid prefix expression : ");
scanf("%s", str);
int result = prefix(str);
if (result == EMPTY) {
printf("\nINVALID EXPRESSION");
exit(1);
}
printf("\nResult is = %d ", result);
return 0;
}
```

Test Case:

//Entered prefix of (5*6+9*7)



2: Convert an infix expression to prefix.

Code:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 200
#define EMPTY '\0'
typedef enum
{
NO = 0,
YES = 1,
} BOOL;
typedef struct Stack
{
```

```
char *arr;
int tos;
} STACK_t;
typedef STACK_t * STACK_p_t;
void initStack (STACK_p_t stack)
{
stack->arr = (char *)calloc(SIZE, sizeof(char));
stack->tos = -1;
}
BOOL isStackFull (STACK_t stack)
{
if (stack.tos == SIZE - 1)
return YES;
return NO;
}
BOOL isStackEmpty (STACK_t stack)
{
if (stack.tos == -1)
return YES;
return NO;
}
void push (STACK_p_t stack, char item)
{
if (stack->tos == SIZE - 1)
return;
stack->tos += 1;
*(stack->arr + stack->tos) = item;
}
char top (STACK_t stack)
{
if (stack.tos == -1)
return EMPTY;
return *(stack.arr + stack.tos);
}
char pop (STACK_p_t stack)
{
if (stack->tos == -1)
return EMPTY;
return *(stack->arr + (stack->tos)--);
}
void reverse (STACK_p_t stack)
{
int i;
for (i = 0; i <= stack->tos/2; ++i)
{
char ch = *(stack->arr + i);
*(stack->arr + i) = *(stack->arr + stack->tos - i);
*(stack->arr + stack->tos - i) = ch;
}
}
```

```
int indexOf (char character, char *string)
{
char *ptr = strchr(string, character);
if (ptr)
return (int)(ptr - string);return -1;
}
BOOL isOperator (char op)
{
if (indexOf(op, "+-*/%$") != -1)
return YES;
return NO;
}
BOOL isOperand (char op)
{
if ((op >= 65 && op <= 90) || (op >= 97 && op <= 122))
return YES;
if (op >= 48 && op <= 57)
return YES;
return NO;
}
int operatorPrecedence (char op)
{
if (indexOf(op, ")]}") != -1) return 0;
else if (indexOf(op, "+-") != -1) return 1;
else if (indexOf(op, "*/%") != -1) return 2;
else if (op == '$') return 3;
return -1;
}
char * toPrefix (char * exp)
{
STACK_p_t prefix = (STACK_p_t)malloc(sizeof(STACK_t));
STACK_p_t operator = (STACK_p_t)malloc(sizeof(STACK_t));
initStack(prefix);
initStack(operator);
int l = (int)strlen(exp);
int i;
for (i = l - 1; i >= 0; --i)
{
char z = *(exp + i);
if (isOperand(z))
push(prefix, z);
else if (operatorPrecedence(z) == 0)
push(operator, z);
else if (isOperator(z))
{
while (!isStackEmpty(*operator) && operatorPrecedence(z) <
operatorPrecedence(top(*operator)))
{
char op = pop(operator);
if (isOperator(op))
```

```
push(prefix, op);
}
push(operator, z);
}
else if (indexOf(z, "([{") != -1)
{
while (operatorPrecedence(top(*operator)) != 0)
push(prefix, pop(operator));
pop(operator);
}
else
continue;}
while(!isStackEmpty(*operator))
push(prefix, pop(operator));
reverse(prefix);
return prefix->arr;
}
int main(int argc, const char * argv[])
{
char *infix = (char *)calloc(SIZE, sizeof(char));
printf("INFIX TO PREFIX");
printf("\nEnter a valid Infix expression: ");
fgets(infix, SIZE, stdin);
char *prefix = toPrefix(infix);
printf("\nInfix: %s\nPrefix: %s", infix, prefix);
return 0;
}
```

Test Case:

3: Implement two stacks in an array.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
#define EMPTY '\0'
typedef enum {
NO = 0,
YES = 1,
}BOOL;
typedef struct Stack {
int tos1;
int tos2;
char *stack;
}STACK_t;
BOOL isStackFull (STACK_t stack) {
if (stack.tos1 == stack.tos2 - 1)
return YES;
return NO;
}
BOOL isStackEmpty1 (STACK_t stack) {
if (stack.tos1 == -1)
return YES;
return NO;
}
BOOL isStackEmpty2 (STACK_t stack) {
if (stack.tos2 == SIZE)
return YES;
return NO;
}
void push1 (STACK_t *stack, char item) {
if (isStackFull (*stack)) {
printf("\nSTACK 1 OVERFLOW");
return;
}
stack->tos1 += 1;
*(stack->stack + stack->tos1) = item;
}
void push2 (STACK_t *stack, char item) {
if (isStackFull(*stack)) {
printf("\nSTACK 2 OVERFLOW");
return;
}
stack->tos2 -= 1;
*(stack->stack + stack->tos2) = item;
}
char pop1 (STACK_t *stack) {
if (isStackEmpty1 (*stack)) {
printf("\nSTACK 1 UNDERFLOW");
```

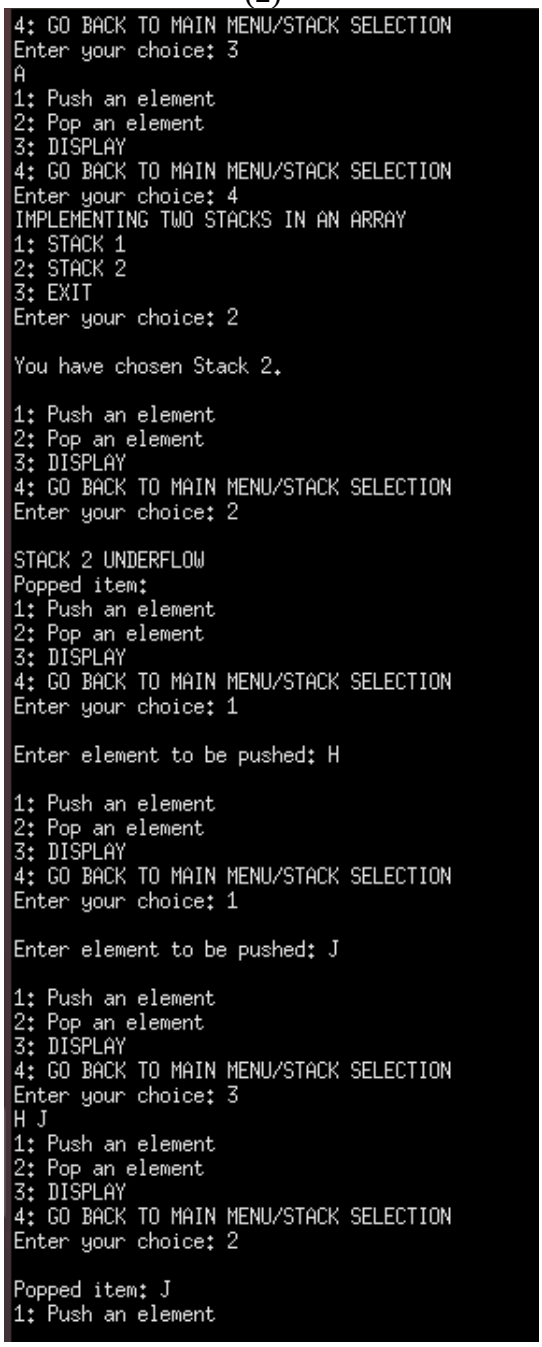
```
return EMPTY;
}
return *(stack->stack + (stack->tos1)--);
}
char pop2 (STACK_t *stack) {
if (isStackEmpty2 (*stack)) {
printf("\nSTACK 2 UNDERFLOW");
return EMPTY;
}
return *(stack->stack + (stack->tos2)++);
}
void display (STACK_t stack, char stackChoice) {
char *pi;
if (stackChoice == '1')
for (pi = stack.stack; pi <= stack.stack + stack.tos1; ++pi)
printf("%c ", *pi);
if (stackChoice == '2')
for (pi = stack.stack + SIZE - 1; pi >= stack.stack + stack.tos2; --pi)
printf("%c ", *pi);
}
int main(int argc, const char * argv[]) {STACK_t stack;
stack.stack = (char *)calloc(SIZE, sizeof(char));
stack.tos1 = -1;
stack.tos2 = SIZE;
char stackChoice;
do {
printf("IMPLEMENTING TWO STACKS IN AN ARRAY");
printf("\n1: STACK 1");
printf("\n2: STACK 2");
printf("\n3: EXIT");
printf("\nEnter your choice: ");
scanf(" %c", &stackChoice);
if (!(stackChoice == '1' || stackChoice == '2'))
exit(1);
printf("\nYou have chosen Stack %c.\n", stackChoice);
char choice;
do {
printf("\n1: Push an element");
printf("\n2: Pop an element");
printf("\n3: DISPLAY");
printf("\n4: GO BACK TO MAIN MENU/STACK SELECTION");
printf("\nEnter your choice: ");
scanf(" %c", &choice);
if (choice == '1') {
char item;
printf("\nEnter element to be pushed: ");
scanf(" %c", &item);
if (stackChoice == '1') push1(&stack, item);
if (stackChoice == '2') push2(&stack, item);
}
```

```
else if (choice == '2') {
char item = '\0';
if (stackChoice == '1') item = pop1(&stack);
if (stackChoice == '2') item = pop2(&stack);
printf("\nPopped item: %c ", item);
}
else if (choice == '3') {
display(stack, stackChoice);
}
else
break;
}while (choice == '1' || choice == '2' || choice == '3');
}while (stackChoice == '1' || stackChoice == '2');
return 0;
}
```

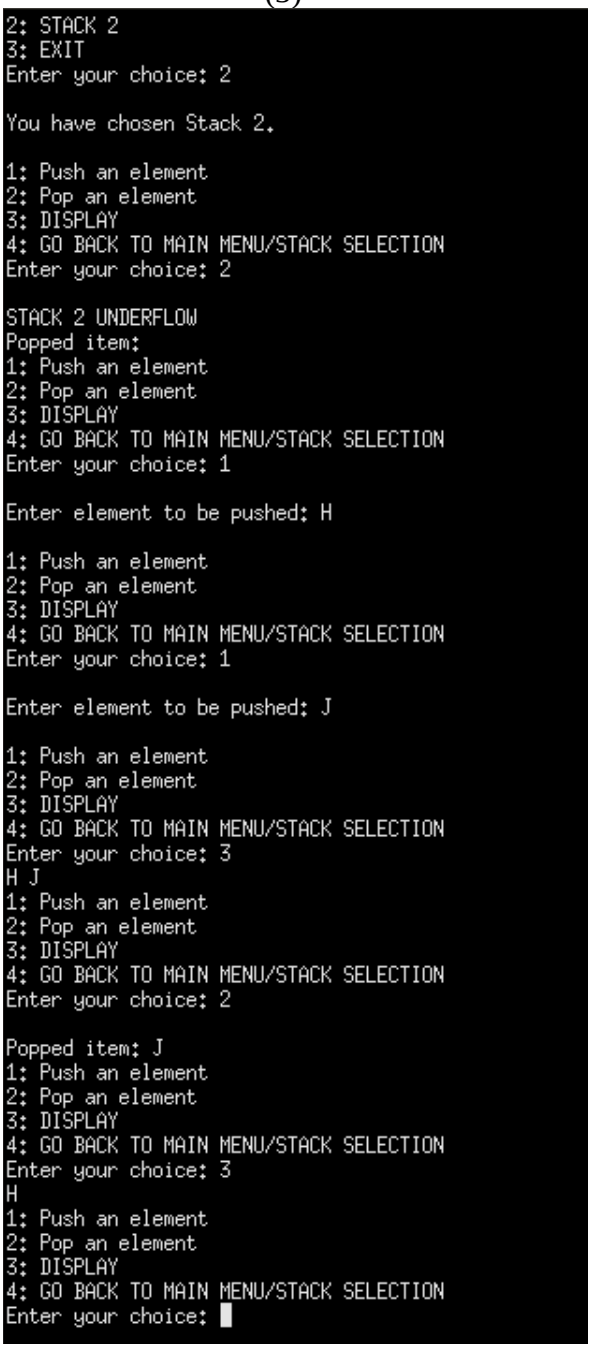
Test Case:

```
IMPLEMENTING TWO STACKS IN AN ARRAY
1: STACK 1
2: STACK 2
3: EXIT
Enter your choice: 1

You have chosen Stack 1.

1: Push an element
2: Pop an element
3: DISPLAY
4: GO BACK TO MAIN MENU/STACK SELECTION
Enter your choice: 2

STACK 1 UNDERFLOW
Popped item:
1: Push an element
2: Pop an element
3: DISPLAY
4: GO BACK TO MAIN MENU/STACK SELECTION
Enter your choice: 1

Enter element to be pushed: A

1: Push an element
2: Pop an element
3: DISPLAY
4: GO BACK TO MAIN MENU/STACK SELECTION
Enter your choice: 1

Enter element to be pushed: B

1: Push an element
2: Pop an element
3: DISPLAY
4: GO BACK TO MAIN MENU/STACK SELECTION
Enter your choice: 3
A B
1: Push an element
2: Pop an element
3: DISPLAY
4: GO BACK TO MAIN MENU/STACK SELECTION
Enter your choice: 2

Popped item: B
1: Push an element
2: Pop an element
3: DISPLAY
4: GO BACK TO MAIN MENU/STACK SELECTION
Enter your choice: 3
A
1: Push an element
2: Pop an element
3: DISPLAY
```

(2)



(3)



//1, 2 and 3 is the sequential numbering of the test case.