190905514
Mohammad Tofik

WEEK 4 LAB 4

1) Evaluate a given prefix expression using stack.

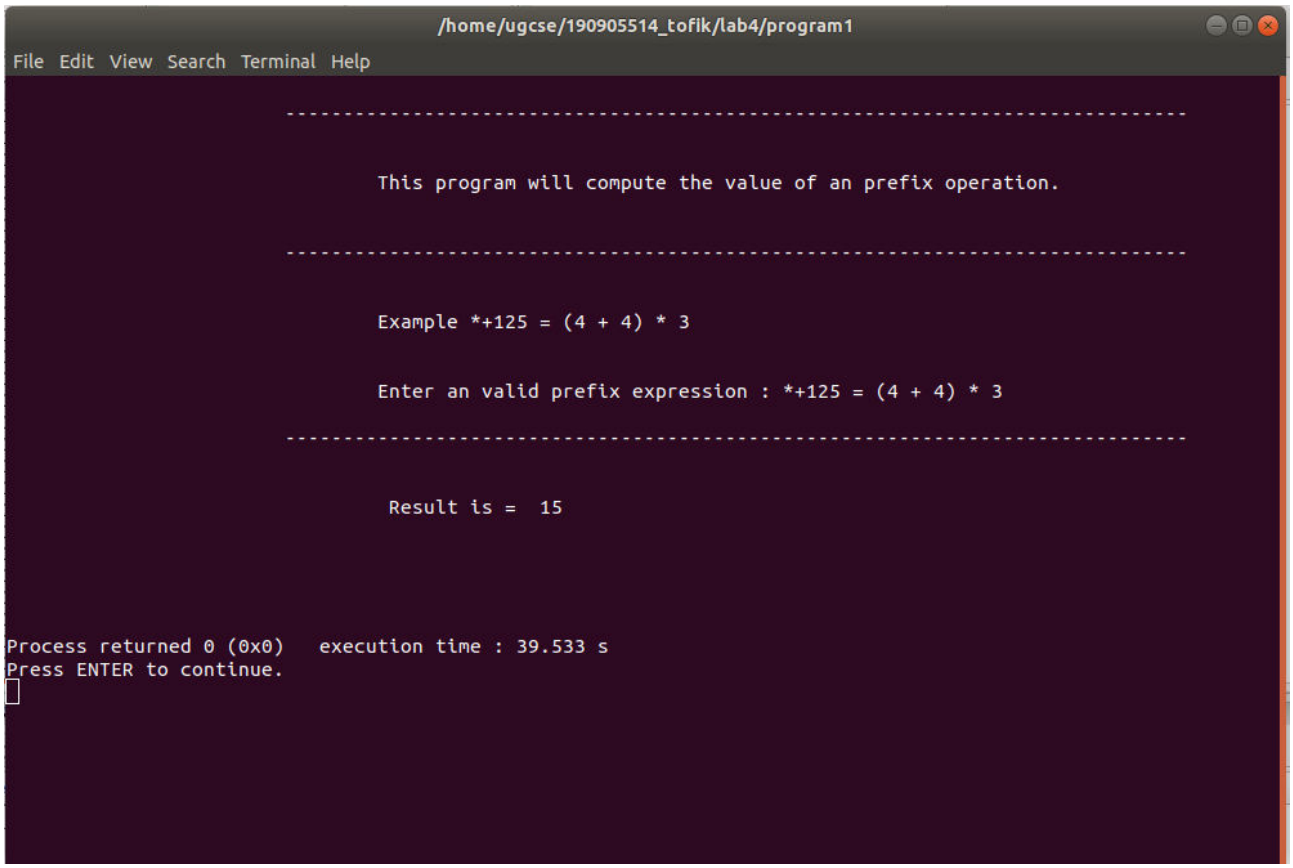
Program1.c

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 999
#define EMPTY -32768
typedef enum {
   ALLISWELL = 0,
   PRACTICE = 1,
} BOOLEAN;
BOOLEAN isStackFull (int tos) {
   if (tos == SIZE - 1)
      return PRACTICE;
   return ALLISWELL;
}
BOOLEAN isStackEmpty (int tos) {
   if (tos == -1)
      return PRACTICE;
   return ALLISWELL;
}
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;
}

BOOLEAN operatorOn (char op) {
   if (indexOf(op, "+-*/$") != -1)
      return PRACTICE;
```

```
    return ALLISWELL;
}
BOOLEAN isNumber (char op) {
    if (op >= '0' && op <= '9')
        return PRACTICE;
    return ALLISWELL;
}



BOOLEAN isAlphabet (char op) {
    if ((op >= 'A' && op <= 'Z') || (op >= 'a' && op <= 'z'))
        return PRACTICE;
    return ALLISWELL;
}

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("\n\t\t\t----------------------------------------------------------------------------\n\n");
    printf("\n\t\t\t\tThis program will compute the value of an prefix operation.\n\n");
    printf("\n\t\t\t----------------------------------------------------------------------------\n\n");
    printf("\n\t\t\t\tExample *+125 = (4 + 4) * 3 \n\n");
    printf("\n\t\t\t\tEnter an valid prefix expression : ");
    scanf("%s", str);
    printf("\n\t\t\t----------------------------------------------------------------------------\n\n");
    int result = prefix(str);

    if (result == EMPTY) {
        printf("\n\t\t\t\tINVALID EXPRESSION.\n\n");
```

```
        exit(6);
    }

    printf("\n\t\t\t\t Result is =  %d ", result);
    printf("\n\n\n\n\n");

    return 0;
}
```

OUTPUT :



2)Convert an infix expression to prefix.

Program2.c

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 1000
#define EMPTY '\0'
typedef enum
{
   ALLISWELL = 0,
   PRACTICE = 1,
} BOOLEAN;
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;
}
BOOLEAN isStackFull (STACK_t stack)
{
   if (stack.tos == SIZE - 1)
      return PRACTICE;
   return ALLISWELL;
}
BOOLEAN isStackEmpty (STACK_t stack)
{
   if (stack.tos == -1)
      return PRACTICE;
   return ALLISWELL;
}
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;
}
BOOLEAN isOperator (char op)
{
    if (indexOf(op, "+-*/%$") != -1)
        return PRACTICE;
    return ALLISWELL;
}

BOOLEAN isOperand (char op)
{
    if ((op >= 65 && op <= 90) || (op >= 97 && op <= 122))
        return PRACTICE;
    if (op >= 48 && op <= 57)
        return PRACTICE;
    return ALLISWELL;
}
    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("\n\t\t\t----------------------------------------------------------------\n\n");
    printf("\n\t\t\t\tCONVERTING AN INFIX EXPRESSION INTO PREFIX \n\n");
    printf("\n\t\t\t----------------------------------------------------------------\n\n");
    printf("\n\t\t\t\tPrefix e.g. \"b/c * f - h + k/i/(d+e)");
    printf("\n\t\t\t\tEnter an valid Infix expression : ");
    fgets(infix, SIZE, stdin);

    char *prefix = toPrefix(infix);

    printf("\n\t\t\t\tInfix: %s\n\tPrefix: %s\n\n", infix, prefix);

    return 0;

}
```

```
/home/ugcse/190905514_tofik/lab4/program2
File  Edit  View  Search  Terminal  Help

                        ----------------------------------------------------------------


                                CONVERTING AN INFIX EXPRESSION INTO PREFIX


                        ----------------------------------------------------------------


                                Prefix e.g. "b/c * f - h + k/i/(d+e)
                                Enter an valid Infix expression : b/c * f - h + k/i/(d + e)

                                Infix: b/c * f - h + k/i/(d + e)

        Prefix: +-*/bcfh//ki+de


Process returned 0 (0x0)   execution time : 51.827 s
Press ENTER to continue.
```

3) Implement two stacks in an array.

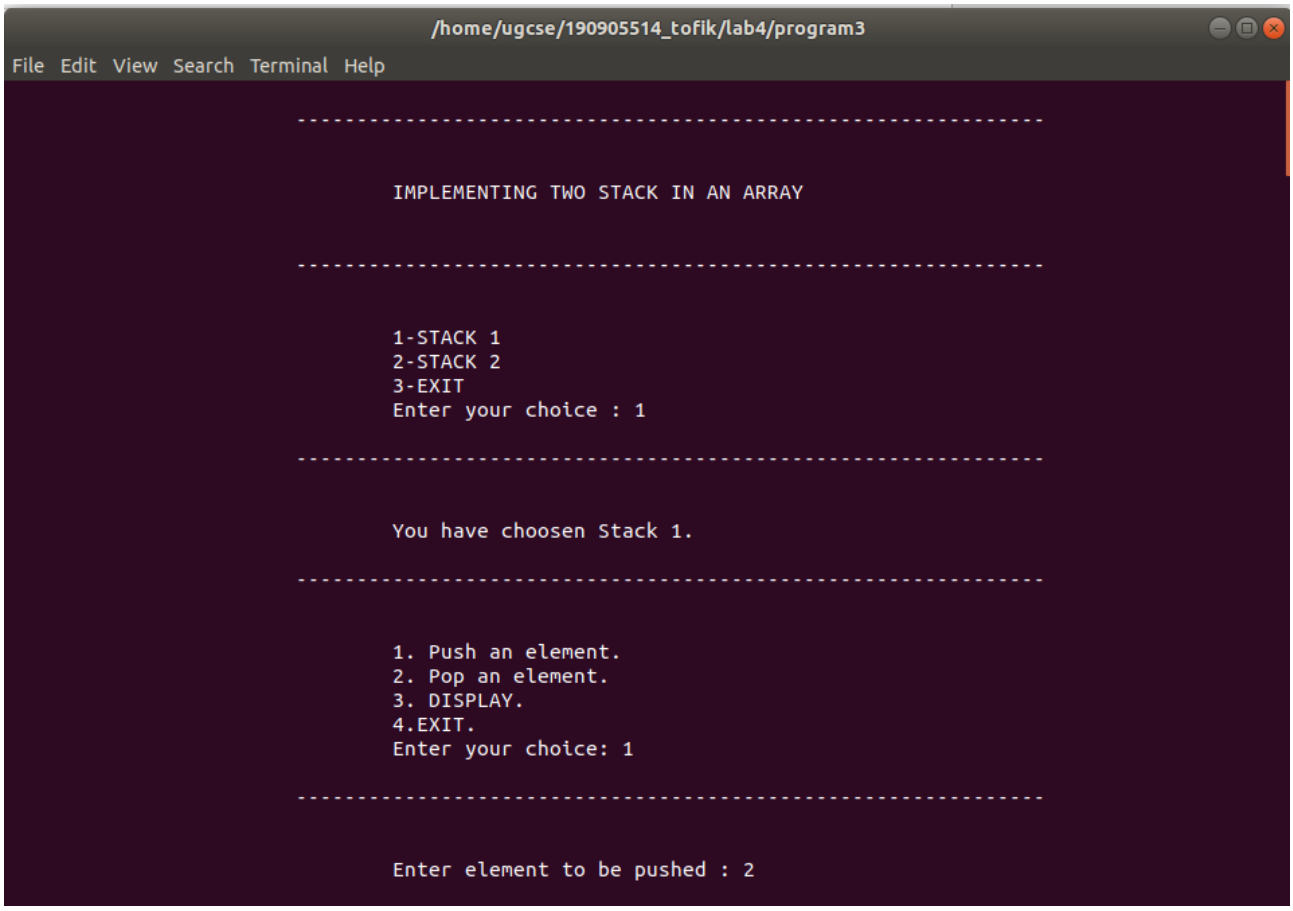
program3.c

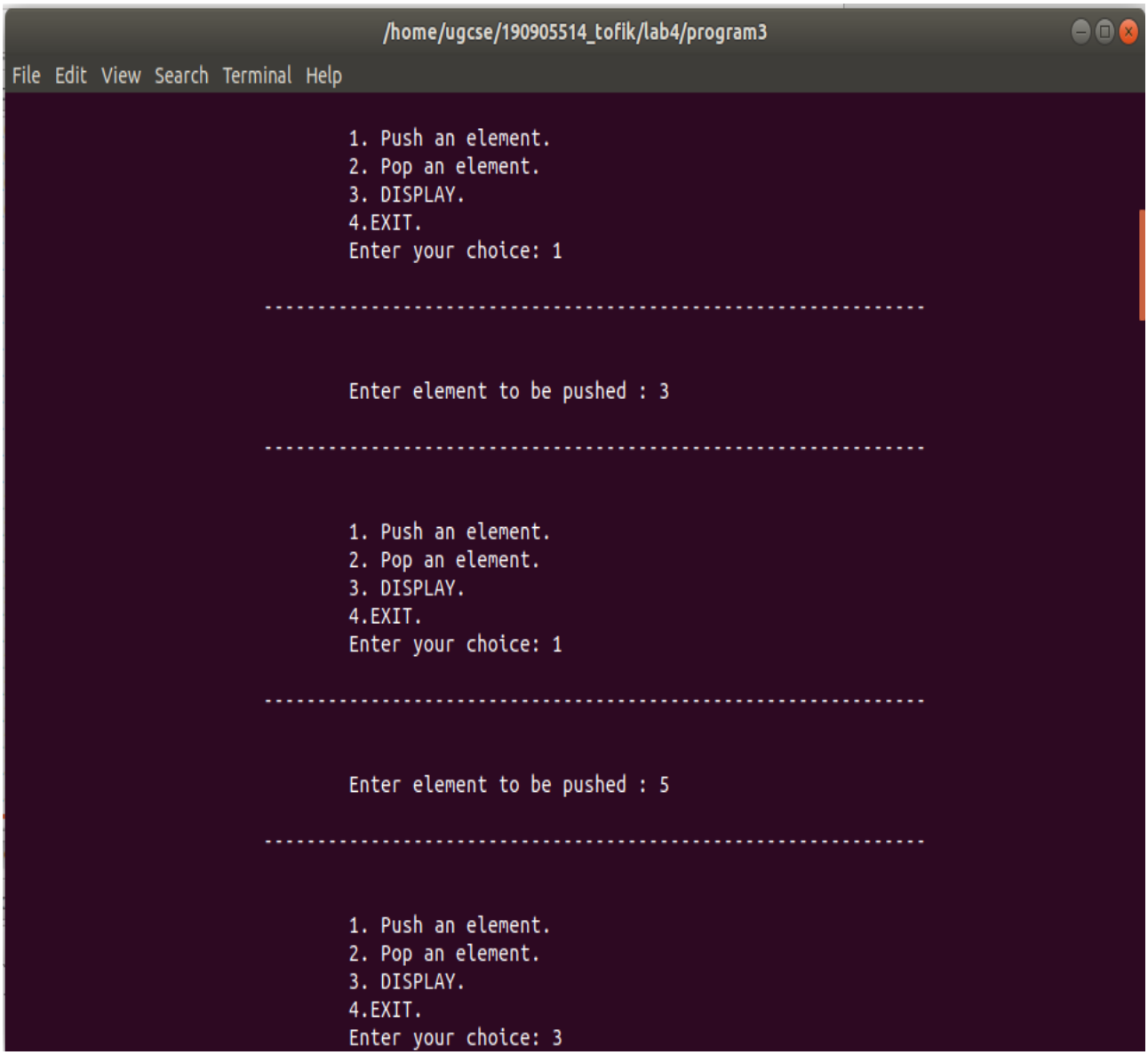
```
#include <stdio.h>
#include <stdlib.h>


#define SIZE 10
```

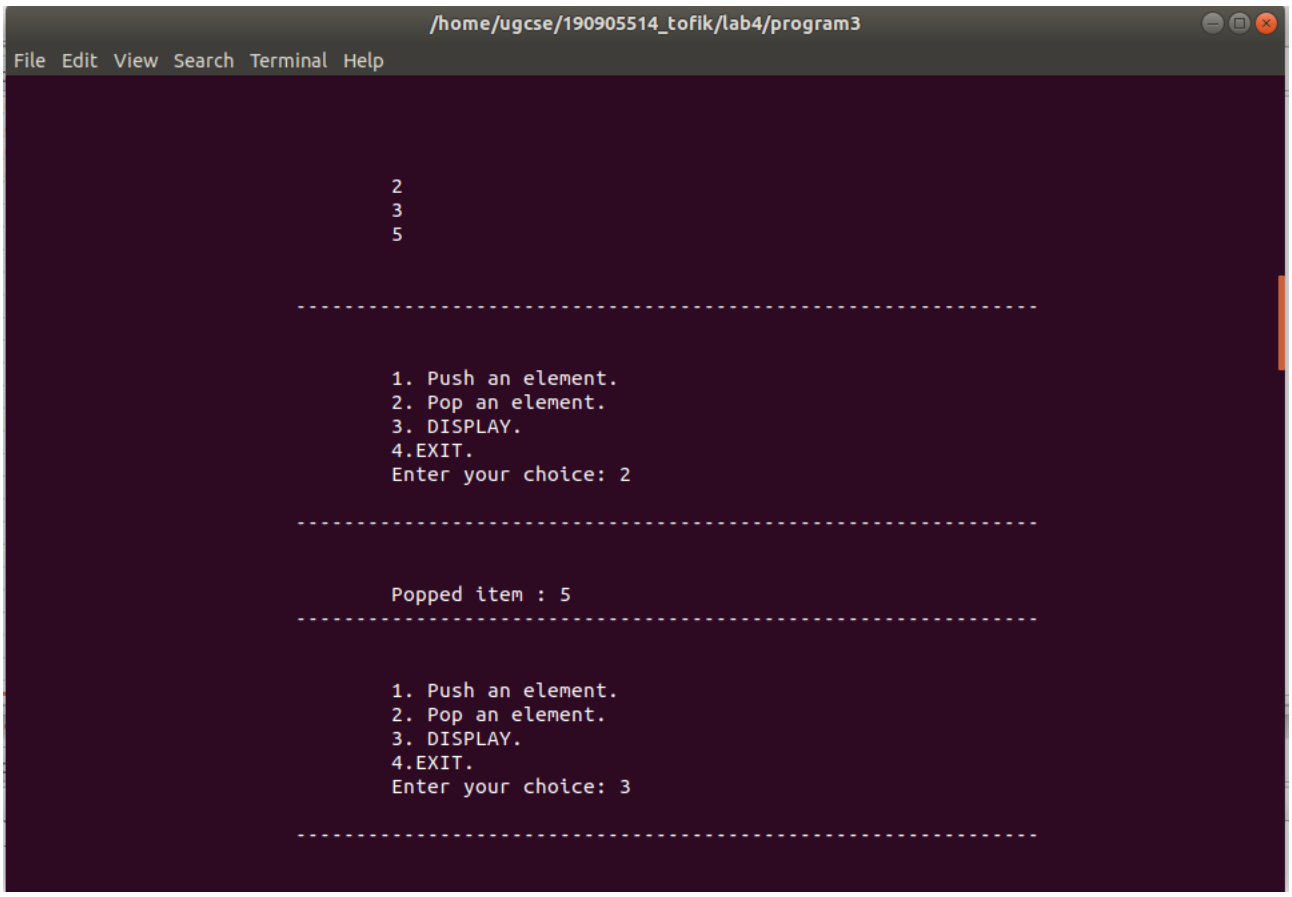
```
#define EMPTY '\0'
typedef enum {
    ALLISWELL = 0,
    PRACTICE = 1,
}BOOLEAN;
typedef struct Stack {
    int tos1;
    int tos2;
    char *stack;
}STACK_t;
BOOLEAN isStackFull (STACK_t stack) {
    if (stack.tos1 == stack.tos2 - 1)
        return PRACTICE;
    return ALLISWELL;
}
BOOLEAN isStackEmpty1 (STACK_t stack) {
    if (stack.tos1 == -1)
        return PRACTICE;
    return ALLISWELL;
}
BOOLEAN isStackEmpty2 (STACK_t stack) {
    if (stack.tos2 == SIZE)
        return PRACTICE;
    return ALLISWELL;
}
void push1 (STACK_t *stack, char item) {
   if (isStackFull (*stack)) {
        printf("\n\t\t\t\tSTACK 1 OVERFLOW\n");
        return;
    }
   stack->tos1 += 1;
    *(stack->stack + stack->tos1) = item;
}
void push2 (STACK_t *stack, char item) {
    if (isStackFull(*stack)) {
  printf("\n\t\t\t\tSTACK 2 OVERFLOW\n");
        return;
    }
  stack->tos2 -= 1;
    *(stack->stack + stack->tos2) = item;
}
char pop1 (STACK_t *stack) {
    if (isStackEmpty1 (*stack)) {
        printf("\n\t\t\t\tSTACK 1 UNDERFLOW\n");
        return EMPTY;
    }
    return *(stack->stack + (stack->tos1)--);
}
char pop2 (STACK_t *stack) {
    if (isStackEmpty2 (*stack)) {
    printf("\n\t\t\t\tSTACK 2 UNDERFLOW\n");
        return EMPTY;
    }
 return *(stack->stack + (stack->tos2)++);
}
void display (STACK_t stack, char stackChoice) {
    printf("\n\n");
    char *pi;
    if (stackChoice == '1')
        for (pi = stack.stack; pi <= stack.stack + stack.tos1; ++pi)
            printf("\n\t\t\t\t%c", *pi);
    if (stackChoice == '2')
        for (pi = stack.stack + SIZE - 1; pi >= stack.stack + stack.tos2; --pi)
            printf("\n\t\t\t\t%c", *pi);
    printf("\n\n");
}

int main(int argc, const char * argv[]) {
```

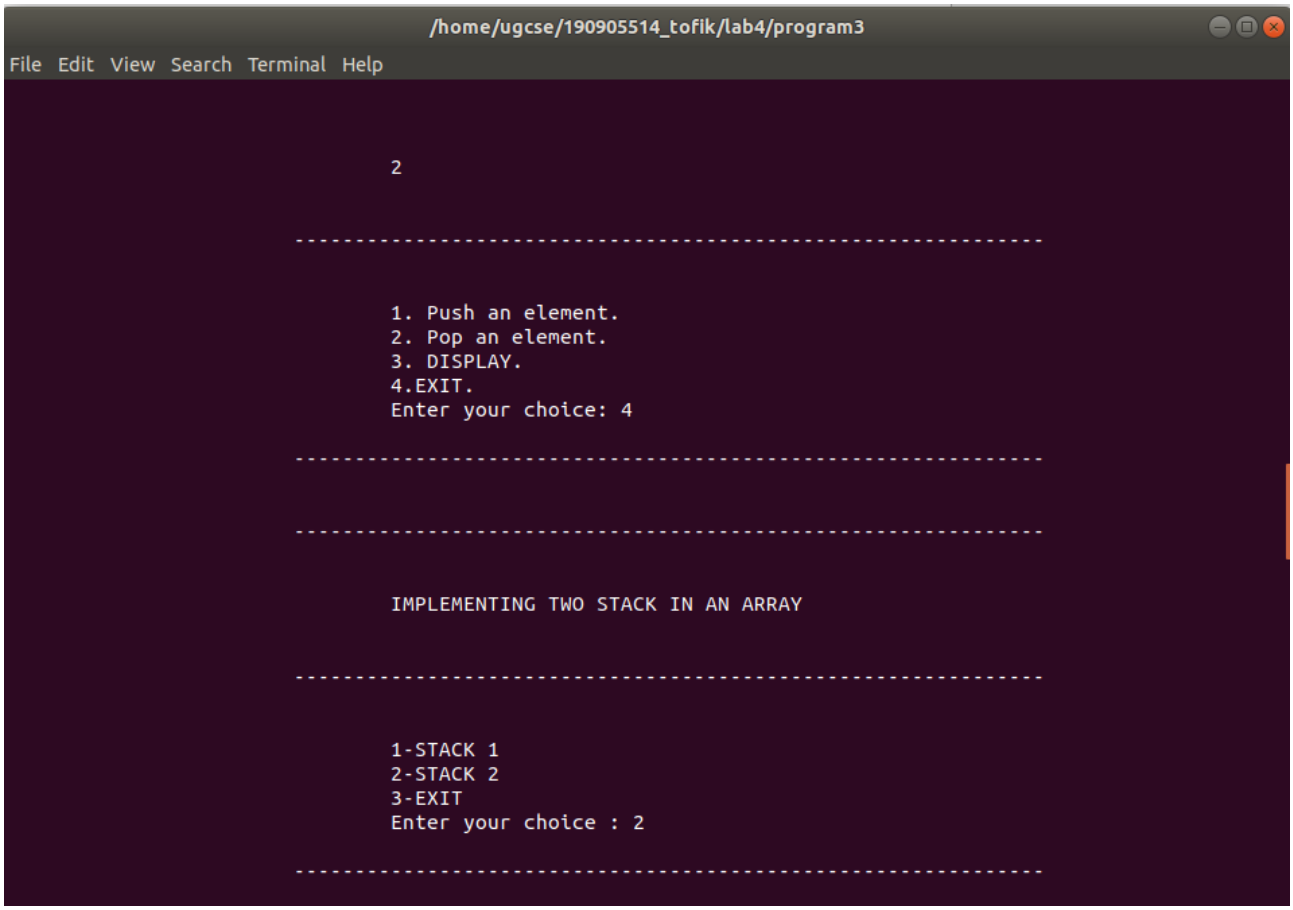
```
    STACK_t stack;
    stack.stack = (char *)calloc(SIZE, sizeof(char));
    stack.tos1 = -1;
    stack.tos2 = SIZE;

    char stackChoice;
    do {

     printf("\n\t\t\t-----------------------------------------------------------------\n\n");
            printf("\n\t\t\t\tIMPLEMENTING TWO STACK IN AN ARRAY \n\n");
            printf("\n\t\t\t-----------------------------------------------------------------\n\n");
            printf("\n\t\t\t\t1-STACK 1");
            printf("\n\t\t\t\t2-STACK 2");
            printf("\n\t\t\t\t3-EXIT");
            printf("\n\t\t\t\tEnter your choice : ");
            scanf(" %c", &stackChoice);
            printf("\n\t\t\t-----------------------------------------------------------------\n\n");

        if (!(stackChoice == '1' || stackChoice == '2'))
            exit(6);
        printf("\n\t\t\t\tYou have choosen Stack %c.\n", stackChoice);

        char choice;
        do {
  printf("\n\t\t\t-----------------------------------------------------------------\n\n");
  printf("\n\t\t\t\t1. Push an element.");
  printf("\n\t\t\t\t2. Pop an element.");
  printf("\n\t\t\t\t3. DISPLAY.");
  printf("\n\t\t\t\t4.EXIT.");
  printf("\n\t\t\t\tEnter your choice: ");
  scanf(" %c", &choice);
  printf("\n\t\t\t-----------------------------------------------------------------\n\n");
            if (choice == '1') {
    char item;
                printf("\n\t\t\t\tEnter 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("\n\t\t\t\tPopped 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;
}
```

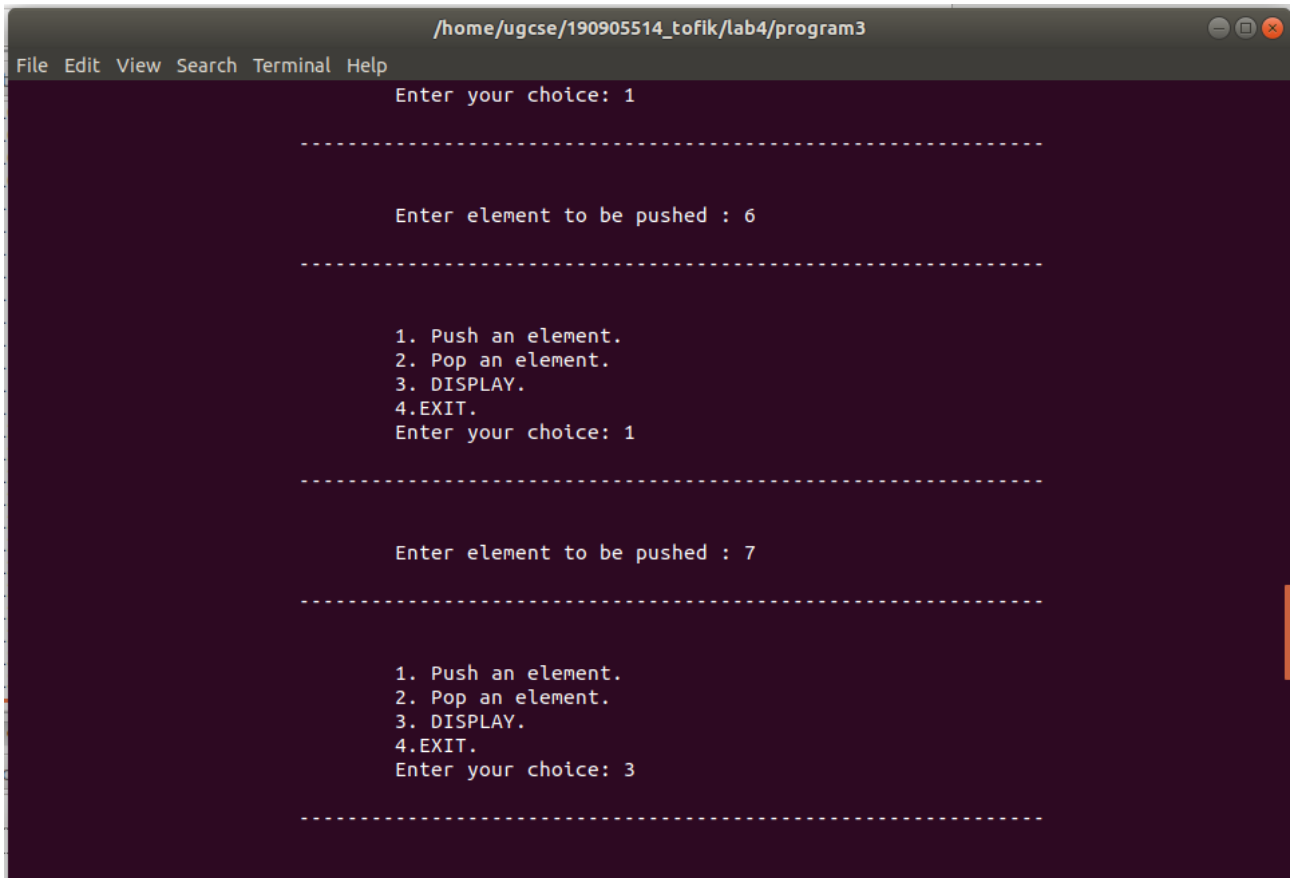
OUTPUT :

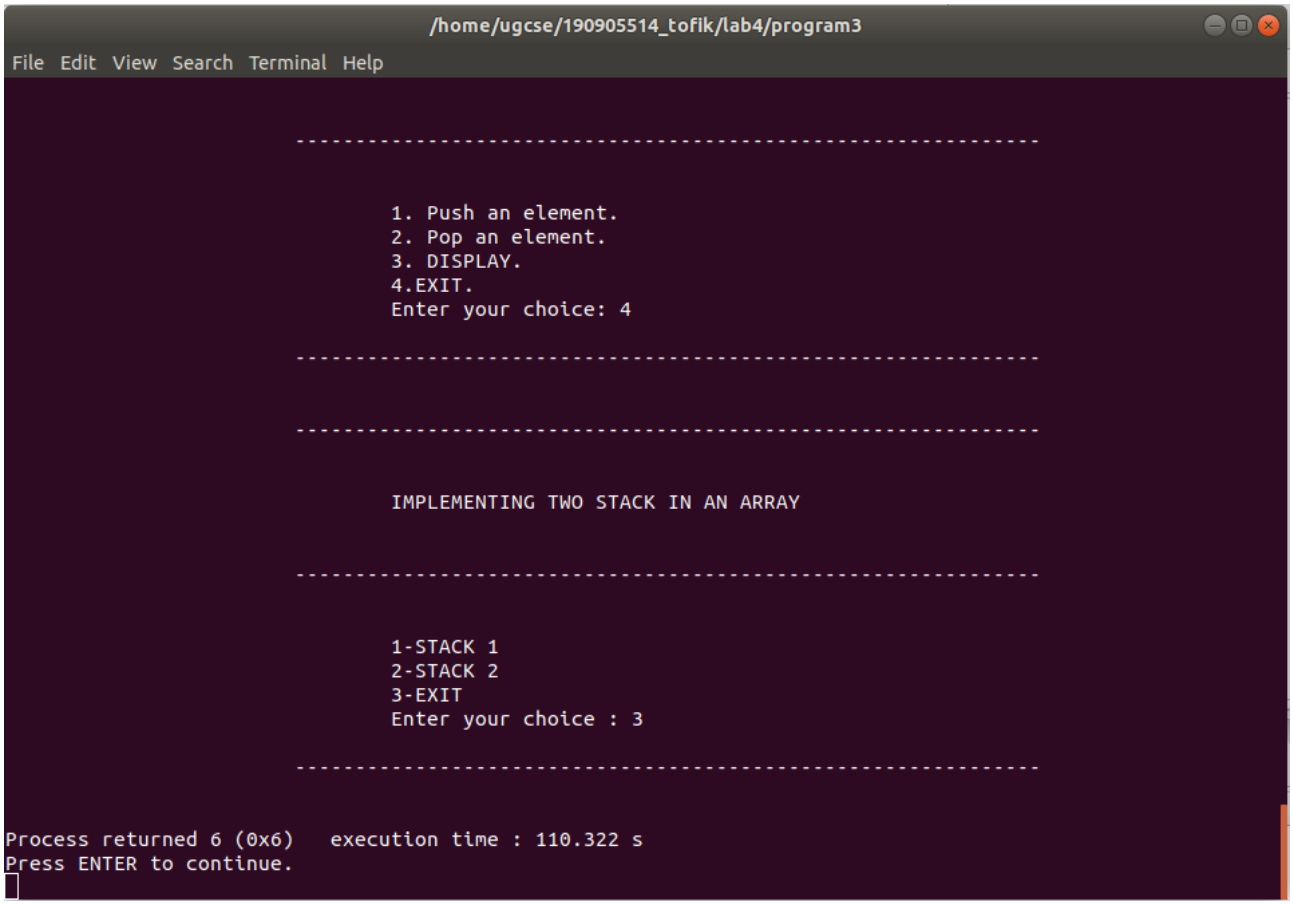

/home/ugcse/190905514_tofik/lab4/program3
File Edit View Search Terminal Help
IMPLEMENTING TWO STACK IN AN ARRAY
1-STACK 1
2-STACK 2
3-EXIT
Enter your choice : 1
You have choosen Stack 1.
1. Push an element.
2. Pop an element.
3. DISPLAY.
4.EXIT.
Enter your choice: 1
Enter element to be pushed : 2


/home/ugcse/190905514_tofik/lab4/program3
File Edit View Search Terminal Help
1. Push an element.
2. Pop an element.
3. DISPLAY.
4.EXIT.
Enter your choice: 1
Enter element to be pushed : 3
1. Push an element.
2. Pop an element.
3. DISPLAY.
4.EXIT.
Enter your choice: 1
Enter element to be pushed : 5
1. Push an element.
2. Pop an element.
3. DISPLAY.
4.EXIT.
Enter your choice: 3


/home/ugcse/190905514_tofik/lab4/program3
File Edit View Search Terminal Help
2
3
5
1. Push an element.
2. Pop an element.
3. DISPLAY.
4.EXIT.
Enter your choice: 2
Popped item : 5
1. Push an element.
2. Pop an element.
3. DISPLAY.
4.EXIT.
Enter your choice: 3


/home/ugcse/190905514_tofik/lab4/program3
File Edit View Search Terminal Help
2
1. Push an element.
2. Pop an element.
3. DISPLAY.
4.EXIT.
Enter your choice: 4
IMPLEMENTING TWO STACK IN AN ARRAY
1-STACK 1
2-STACK 2
3-EXIT
Enter your choice : 2


/home/ugcse/190905514_tofik/lab4/program3
File Edit View Search Terminal Help
Enter your choice: 1
Enter element to be pushed : 6
1. Push an element.
2. Pop an element.
3. DISPLAY.
4.EXIT.
Enter your choice: 1
Enter element to be pushed : 7
1. Push an element.
2. Pop an element.
3. DISPLAY.
4.EXIT.
Enter your choice: 3


/home/ugcse/190905514_tofik/lab4/program3
File Edit View Search Terminal Help
1. Push an element.
2. Pop an element.
3. DISPLAY.
4.EXIT.
Enter your choice: 4
IMPLEMENTING TWO STACK IN AN ARRAY
1-STACK 1
2-STACK 2
3-EXIT
Enter your choice : 3
Process returned 6 (0x6) execution time : 110.322 s
Press ENTER to continue.