VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

LAB ASSESSMENT-1

SLOT: L33-L34

CSE 2003: Data Structures and Algorithms

Submitted By:

Sashank Rijal

19BCE2484

Submitted to:

Annapurna Jonnalagadda

Q1

Algorithm:

1. If occurrence(a) is more than occurrence(b), then append "aab" to the end of the sentence.

2. If occurrence(b) is more than occurrence(a), then append "bba" to the end of the sentence.

3. If occurrence(a) equals occurrence(b), then append "ab" to the end of the sentence.

4. Moreover, because we lower by at most one in each iteration the difference between the occurrences of 'a' and 'b'

5. It is assured that "abb" and "aabb" are not followed by "abb" and "abba," respectively, in the next iteration.

Code:

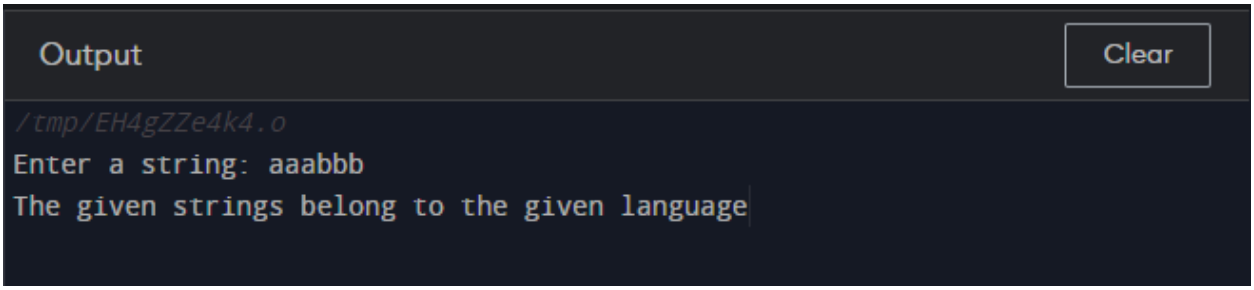
```
#include <stdio.h>

#include <string.h>


#define max 100

int top,stack[max];


void push(char x){


    // Push(Inserting Element in stack) operation

    if(top == max-1){

       printf("stack overflow");

    }  else {

       stack[++top]=x;

    }


}


char pop(){
```

```
   // Pop (Removing element from stack)
    return(stack[top--]);
}



main()
{
  char str[100];
  char str1[100];
  int flag=0;
  printf("Enter string");
  scanf("%s",&str);
  int len = strlen(str);
  int i;

  for(i=0;i<len;i++)
      push(str[i]);

  for(i=0;i<len;i++)
  {
  str1[i]=pop();
  if(i<len/2)
  {if (str1[i]!='b')
   flag++;

  }
  else
  {if (str1[i]!='a')
   flag++;
```

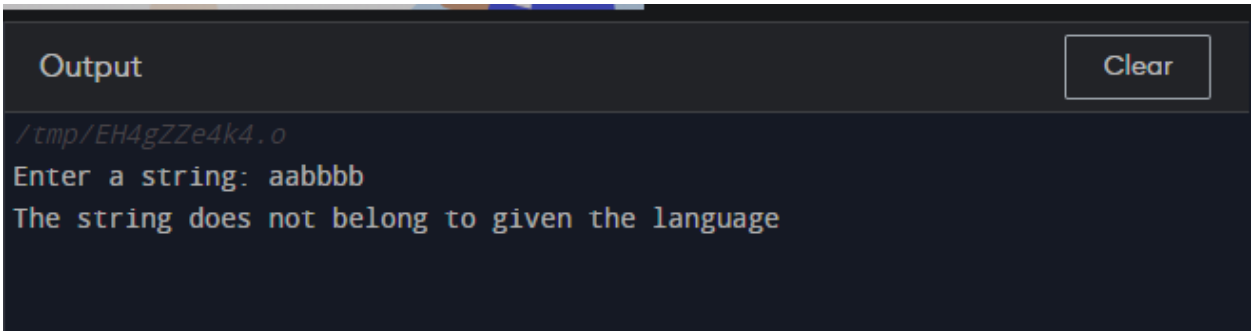
```
  }
  }
  if(flag==0)
  {
     printf("The given strings belong to the given language");
  }
  else
  printf("The string doesnt belong to given the language");
}
```

Output:

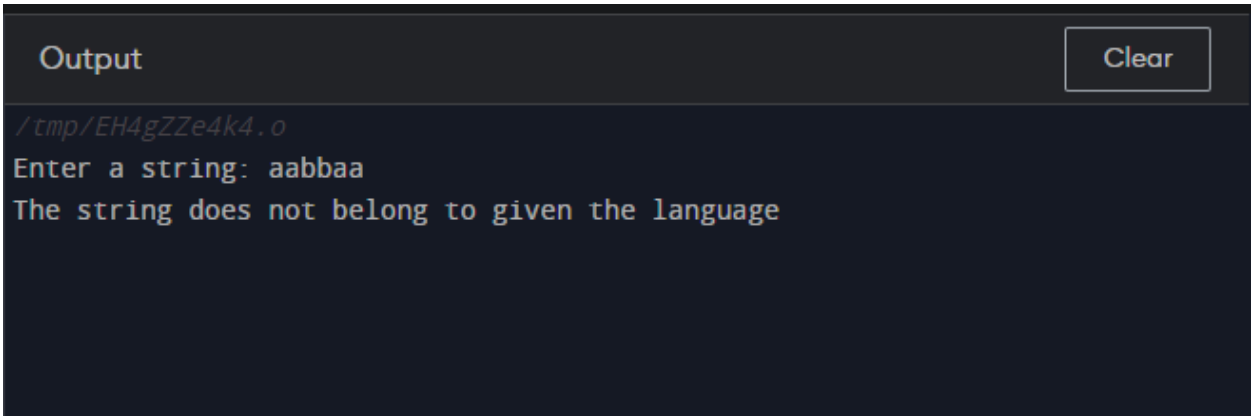
Test case 1:



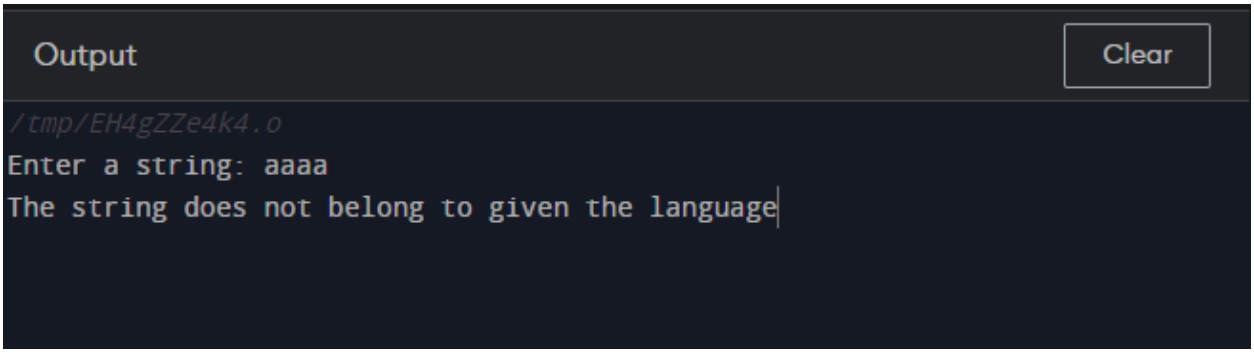
Test case 2:



Test case 3:



Test case 4:



Q2:

Algorithm:

1. Assign a variable n to accept any number of rows.

2. If the data is even, we throw an error while checking the modulo of the input.

3. Set variables that accept characters to their default values.

4. Iterate through variable i using for loop.

5. Print # after iterating variable j.

6. We also print @ after iterating over each loop.

Code:

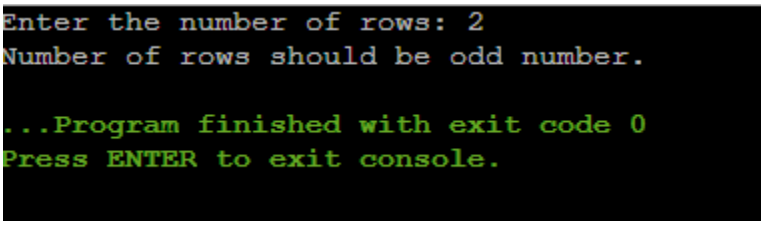
```
#include <stdio.h>
#include <stdlib.h>
int main()

{  int n;
   printf("Enter the number of rows: ");
   scanf("%d", &n);
   if(n%2 == 0)
   {
       printf("Number of rows should be odd number.");
       exit(0);
    }

   int j=1;
   char xx='#';
   char cc='@';
   char zz=xx;
   n=(n/2)+1;
     for(int i =1;i<=n;i++){
         if(i%2==0)
         {
            zz=xx;
         }
         else
         {
           zz=cc;
         }
```

```
    for(int j=i;j<=n;j++){
        printf("  ");
    }

    for(int j=1;j<i;j++){

        printf("%c",zz);
        printf(" ");


    }
    for(int j =1;j<=i;j++){

            printf("%c",zz);
            printf(" ");

    }
    printf("\n");
}

for(int i=2;i<n+1;i++){
     if(i%2==0)
       {
          zz=xx;
       }
       else
       {
         zz=cc;
       }
```

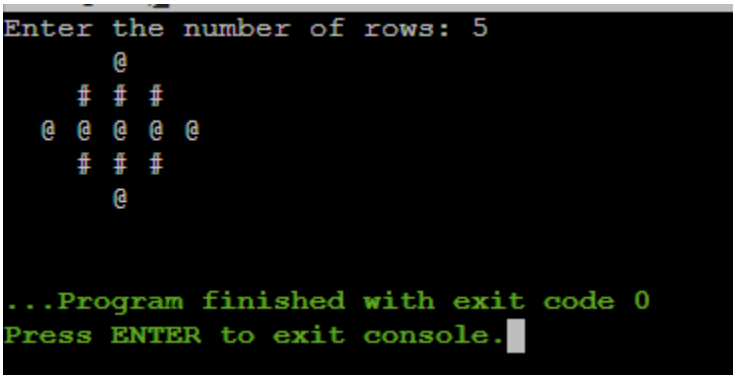
```
    for(int j=1;j<=i;j++){
       printf("  ");
    }
    for(int j=i;j<n;j++){

       printf("%c",zz);
       printf(" ");

    }
    for(int j=i;j<=n;j++){

          printf("%c",zz);
          printf(" ");


    }
    printf("\n");

  }

  return 0;
}
```

Output:

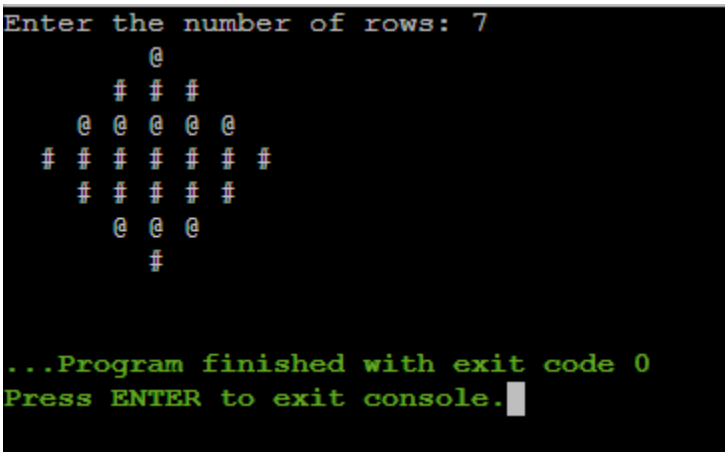
Test case 1:



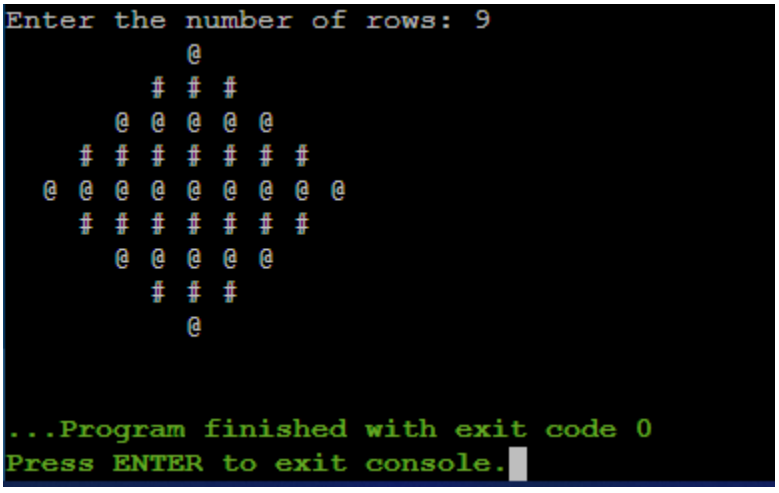
Test case 2:



Test case 3:



Test case 4:



Q3:

Algorithm:

Step 1: initialize function main.

Step 2: set variable str to accept string array. Set flag=0.

Step 3: Extract length of string and if number of a and b is not same then print string does not belong to given language.

Step 4: Loop over variable I. If (str[i]!='a') increase flag likewise for J.

Step 5: if flag=0 : print string belongs else: print string doesn't belong.

Code:

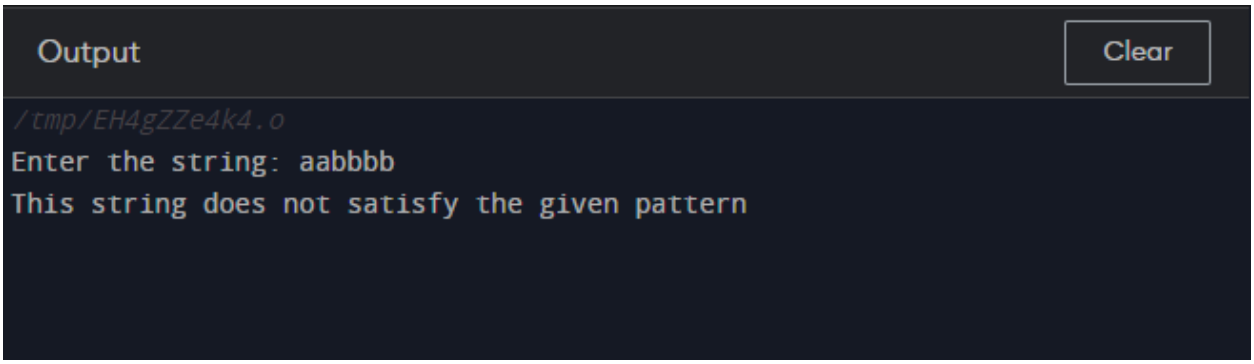
```
#include <stdio.h>

#include <string.h>


 void main()

{

  char str[1000];


  int flag=0;

  printf("Enter the string: ");

  scanf("%s",&str);

  int len = strlen(str);

  int i;


  if(len%2!=0)

  {

      printf("This string does not satisfy the given pattern");

      exit(0);
```

```
  }

  for(i=0;i<len;i++)
  {

  if(i<len/2)
  {if (str[i]!='a')
   flag++;

  }
  else
  {if (str[i]!='b')
   flag++;

  }
  }
  if(flag==0)
  {
     printf("This string satisfies the given pattern");
  }
  else{
  printf("This string does not satisfy the given pattern");
   }
}
```

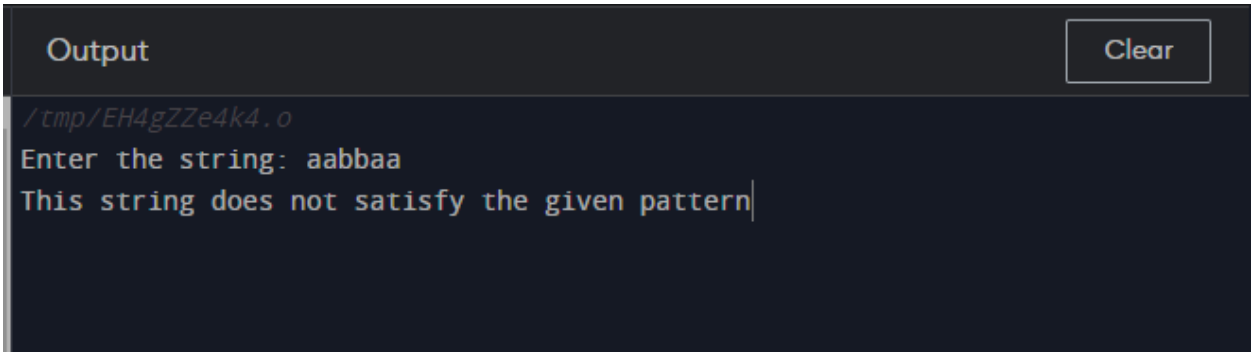
Output:

Test case 1:

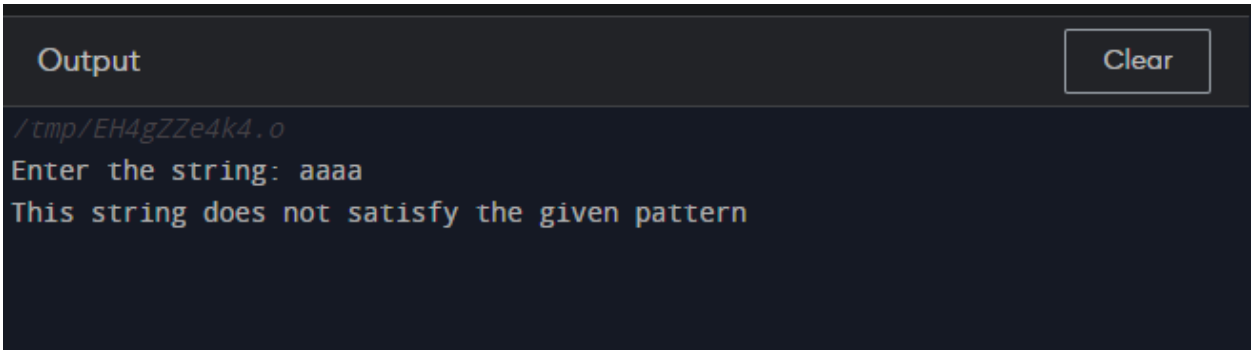
Test case 2:



Test case 3:



Test case 4:



Q4

Algorithm:

1. From left to right, scan the infix phrase.

2. Output the scanned character if it is an operand.

3. If the scanned operator's precedence is greater than the precedence of the stack's operator (or the stack is empty or the stack includes a '('), push it.

Otherwise, pop all the operators from the stack that are larger than or equal to the scanned operator in precedence. Push the scanned operator to the stack after that. (If parentheses appear during popping, halt and insert the scanned operator onto the stack.)

4. Push the scanned character to the stack if it is a '('.

5. If the scanned character is a ')', pop the stack and output it until another '(' appears, then discard both parentheses.

6. Continue with steps 2–6 until the infix expression has been scanned.

7. Print the results

8. Keep popping and outputting from the stack until it's full.

Code:

```
#include <stdio.h>

#include <string.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));

        return stack;
}
int isEmpty(struct Stack* stack)
{
        return stack->top == -1 ;
}
char peek(struct Stack* stack)
{
        return stack->array[stack->top];
}
```

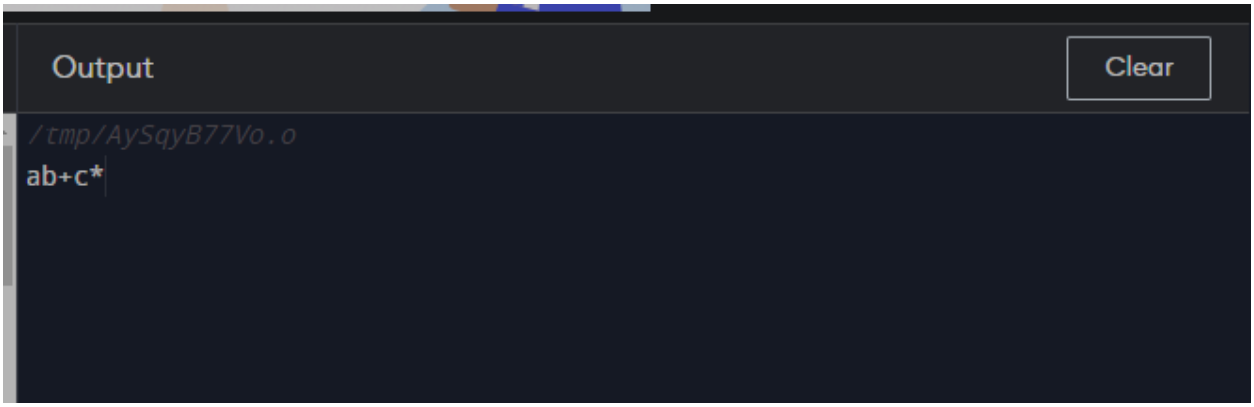
```
char pop(struct Stack* stack)
{
	if (!isEmpty(stack))
		return stack->array[stack->top--] ;
	return '$';
}
void push(struct Stack* stack, char op)
{
	stack->array[++stack->top] = op;
}

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

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

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

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

int infixToPostfix(char* exp)
{
        int i, k;
        struct Stack* stack = createStack(strlen(exp));
        if(!stack)
                return -1 ;

        for (i = 0, k = -1; exp[i]; ++i)
        {
                if (isOperand(exp[i]))
                        exp[++k] = exp[i];

                else if (exp[i] == '(')
                        push(stack, exp[i]);


                else if (exp[i] == ')')
                {
                        while (!isEmpty(stack) && peek(stack) != '('
                                exp[++k] = pop(stack);
                        if (!isEmpty(stack) && peek(stack) != '('
                                return -1;
                        else
```

```
                pop(stack);
        }
        else
        {
            while (!isEmpty(stack) &&
                Prec(exp[i]) <= Prec(peek(stack)))
                exp[++k] = pop(stack);
            push(stack, exp[i]);
        }

    }

    while (!isEmpty(stack))
        exp[++k] = pop(stack );

    exp[++k] = '\0';
    printf( "%s", exp );
}

int main()
{
    char exp[] = "(a+b)*c";
    infixToPostfix(exp);
    return 0;
}
```

Output:

Test Case:



Q5

Algorithm:

1. Create a stack and a queue first.

2. Push the element onto the stack if the stack is empty.

3. Push the next element in the queue to the top of the stack if it is greater than the top of the stack.

4. If not, pop the components out of the stack until you find a smaller or equal element at the top.

5. Push the popped items from step 4 into the auxiliary queue, then push the current element and empty the auxiliary queue on the stack once a smaller/equal element is discovered.

6. Repeat until the stack's size is equal to the input queue's size and the input queue is empty.

7. Now move all of the items from the stack to the queue. The elements in the resultant queue will be in descending order.

Code:

```
#include <stdio.h>

#define SIZE 10


void enQueue(int);

void deQueue();

void display();


int items[SIZE], front = -1, rear = -1;
```

```
void main() {
  deQueue();
  enQueue('A');
  enQueue('N');
  enQueue('F');

  display();
  deQueue();
  display();

  return 0;
}

void enQueue(int value) {
  if (rear == SIZE - 1)
    printf("\nenque -> %s");
  else {
    if (front == -1)
      front = 0;
    rear++;
    items[rear] = value;
    printf("\nEnque -> %s", value);
  }
}

void deQueue() {
  if (front == -1)
    printf("\n%s -> (Enqueue)");
  else {
```

```
    printf("\nDeque : %d", items[front]);
    front++;
    if (front > rear)
      front = rear = -1;
  }
}

void display() {
  if (rear == -1)
  else {
    int i;
    for (i = front; i <= rear; i++)
      printf("%d  ", items[i]);
  }
  printf("\n");
}
```