LAB NO: 4 Date:

SOLVING PROBLEMS USING RAGGED ARRAYS, STRUCTURES AND POINTERS

Objectives:

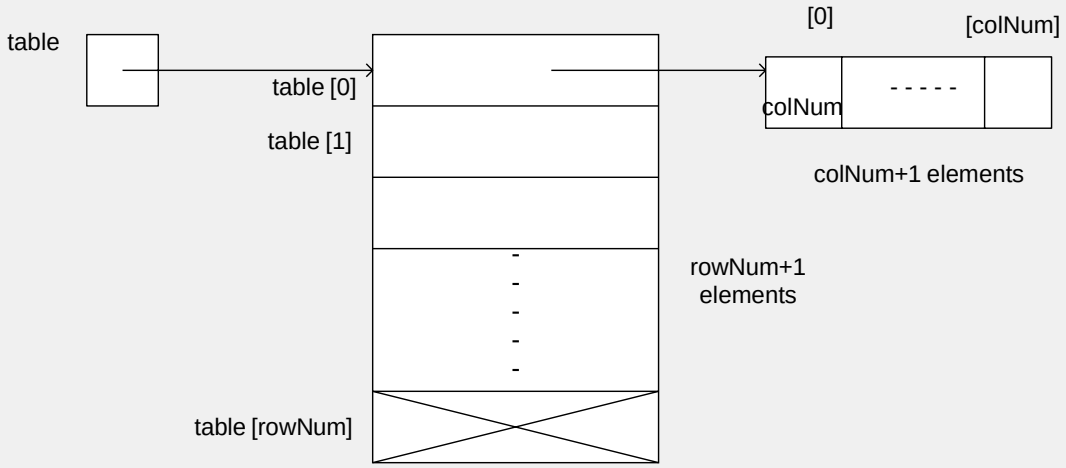
In this lab, student will be able to:

- Familiarize the usage of ragged arrays
- Write programs using structures and pointers
- Familiarize of dynamic memory allocation for structures

I. SOLVED EXERCISE:

1) Write a C program to implement a ragged array dynamically.

Description: In a ragged array the *table* pointer points to the first pointer in an array of pointers. Each array pointer points to a second array of integers, the first element of which is the number of elements in the list. A sample ragged array structure is shown below.



Algorithm: Construct a ragged array

Step 1: Declare a ragged array as a variable *table*.

Step 2: Ask the user for row size and set a variable – *rowNum*

Step 3: Allocate space for *(rowNum+1)* pointers as row pointers. The last row pointer will hold NULL

Step 4: Ask the user for column size and set a variable – *colNum*

Step 5: Allocate space for *(colNum+1)* data elements. The first element will hold value contained in colNum itself.

Step 6: Repeat step 3 for all rows

Step 7 : Display ragged array contents.

Step 8: Stop

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
int rowNum, colNum, i, j;
int **table;
printf("\n enter the number of rows \n");
scanf("%d", &rowNum);
table = (int **) calloc(rowNum+1, sizeof(int *));
for (i = 0; i < rowNum; i++)  /* this will tell which row we are in */
{
        printf("enter size of %d row", i+1);
```

```
        scanf("%d", &colNum);
        table[i] = (int *) calloc(colNum+1, sizeof(int));
           printf("\n enter %d row elements ", i+1);
                   for (j = 1; j <= colNum;  j++)
                     {
                       scanf("%d", &table[i][j]);
                      }
        table[i][0] = colNum;
        printf("size of row number [%d] = %d", i+1, table[i][0]);
}
table[i] = NULL;
for (i = 0; i < rowNum; i++) /* this will tell which row we are in */
{
        printf("displaying %d row elements\n", i+1);
              for (j = 0; j <= *table[i];  j++)
                          {
                        printf("%5d", table[i][j]);
                           }
 printf("\n");
 }
 //freeup the memory
 for (i = 0; i < rowNum; i++) {
        free(table[i]);
    }
 free(table);
 return 0;
}
```

Sample input and output:

enter the number of rows: 3

enter size of row 1: 4

enter row 1 elements: 10 11 12 13

enter size of row 2: 5

enter row 2 elements: 20 21 22 23 24

enter size of row 3

enter row 3 elements: 30 31 32

displaying

10 11 12 13

20 21 22 23 24

30 31 32

II. LAB EXERCISES :

Note: Use Pointers to structures and dynamic memory management functions in the following programs.

1) Implement Complex numbers using structures. Write functions to add, multiply, subtract two complex numbers.

```
#include <stdio.h>
typedef struct {
   float real;
   float imag;
} Complex;
```

```
    // Function to add two complex numbers
    Complex addComplex(Complex a, Complex b) {
        Complex result;
        result.real = a.real + b.real;
        result.imag = a.imag + b.imag;
        return result;
    }

    // Function to subtract two complex numbers
    Complex subtractComplex(Complex a, Complex b) {
        Complex result;
        result.real = a.real - b.real;
        result.imag = a.imag - b.imag;
        return result;
    }

    // Function to multiply two complex numbers
    Complex multiplyComplex(Complex a, Complex b) {
        Complex result;
        result.real = (a.real * b.real) - (a.imag * b.imag);
        result.imag = (a.real * b.imag) + (a.imag * b.real);
        return result;
    }
int main() {
        Complex num1, num2, result;
        printf("Enter the real and imaginary parts of the first complex number: ");
        scanf("%f %f", &num1.real, &num1.imag);
        printf("Enter the real and imaginary parts of the second complex number: ");
        scanf("%f %f", &num2.real, &num2.imag);
        result = addComplex(num1, num2);
        printf("Sum: %.2f + %.2fi\n", result.real, result.imag);
        result = subtractComplex(num1, num2);
        printf("Difference: %.2f + %.2fi\n", result.real, result.imag);
        result = multiplyComplex(num1, num2);
```

```
    printf("Product: %.2f + %.2fi\n", result.real, result.imag);
    return 0;
}
```

2) Write a C program to implement the following functions. Use pointers and dynamic memory management functions.

   i. To read one Student object where Student is a structure with name, roll number and CGPA as the data member
   ii. To display one Student object
   iii. To sort an array of Student structures according to the roll number.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct {
    char *name;
    int rollNumber;
    float CGPA;
} Student;

void readStudent(Student *s) {
    char buffer[100];
    printf("Enter name: ");
    scanf(" %[^\n]", buffer);
    s->name = (char *)malloc((strlen(buffer) + 1) * sizeof(char));
    strcpy(s->name, buffer);
    printf("Enter roll number: ");
    scanf("%d", &s->rollNumber);
    printf("Enter CGPA: ");
    scanf("%f", &s->CGPA);
}
void displayStudent(const Student *s) {
    printf("Name: %s\n", s->name);
    printf("Roll Number: %d\n", s->rollNumber);
    printf("CGPA: %.2f\n", s->CGPA);
}
void sortStudentsByRollNumber(Student *students, int n) {
    int i, j;
    Student temp;
```

```
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (students[j].rollNumber > students[j + 1].rollNumber) {
                // Swap the students
                temp = students[j];
                students[j] = students[j + 1];
                students[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n, i;
    Student *students;
    printf("Enter the number of students: ");
    scanf("%d", &n);

    // Allocate memory for the array of Student structures
    students = (Student *)malloc(n * sizeof(Student));
    for (i = 0; i < n; i++) {
        printf("\nEnter details for student %d:\n", i + 1);
        readStudent(&students[i]);
    }

    sortStudentsByRollNumber(students, n);
    printf("\nSorted student details by roll number:\n");
    for (i = 0; i < n; i++) {
        displayStudent(&students[i]);
        printf("\n");
    }
    for (i = 0; i < n; i++) {
        free(students[i].name);
    }
    free(students);
    return 0;
}
```

3) Samuel wants to store the data of his employees, which includes the following fields:

(i) Name of the employee (ii) Date of birth which is a collection of {day, month, year} (iii) Address which is a collection of {house number, zip code and state}. Write a 'C' program to read and display the data of N employees using pointers to array of structures.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct {
    int day;
    int month;
    int year;
} DateOfBirth;
typedef struct {
    int houseNumber;
    int zipCode;
    char state[50];
} Address;
typedef struct {
    char name[100];
    DateOfBirth dob;
    Address address;
```

```
    } Employee;
void readEmployee(Employee *emp) {
        printf("Enter the name of the employee: ");
        scanf(" %[^\n]", emp->name);
        printf("Enter the date of birth (dd mm yyyy): ");
        scanf("%d %d %d", &emp->dob.day, &emp->dob.month, &emp->dob.year);
        printf("Enter house number: ");
        scanf("%d", &emp->address.houseNumber);
        printf("Enter zip code: ");
        scanf("%d", &emp->address.zipCode);
        printf("Enter state: ");
        scanf(" %[^\n]", emp->address.state);
    }
void displayEmployee(const Employee *emp) {
        // Displaying employee name
        printf("Name: %s\n", emp->name);
        printf("Date of Birth: %02d/%02d/%04d\n", emp->dob.day, emp->dob.month,
    emp->dob.year);
        printf("Address:\n");
        printf("  House Number: %d\n", emp->address.houseNumber);
        printf("  Zip Code: %d\n", emp->address.zipCode);
```

```
    printf("  State: %s\n", emp->address.state);
}

int main() {
    int n, i;
    printf("Enter the number of employees: ");
    scanf("%d", &n);
    Employee *employees = (Employee *)malloc(n * sizeof(Employee));
    for (i = 0; i < n; i++) {
        printf("\nEnter details for employee %d:\n", i + 1);
        readEmployee(&employees[i]);
    }
printf("\nEmployee Details:\n");
    for (i = 0; i < n; i++) {
        printf("\nEmployee %d:\n", i + 1);
        displayEmployee(&employees[i]);
    }
    free(employees);
    return 0;
}
```

4) Create a structure STUDENT consisting of variables of structures:

a. DOB {day, month (use pointer), year},

b. STU_INFO {reg_no, name(use pointer), address},

c. COLLEGE {college_name (use pointer), university_name}

where structure types from i to iii are declared outside the STUDENT independently. Show how to read and display member variables of DOB type if pointer variable is created for DOB inside STUDENT and STUDENT variable is also a pointer variable. The program should read and display the values of all members of STUDENT structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct {
   int day;
   int *month;
   int year;
} DOB;

typedef struct {
   int reg_no;
   char *name;
   char address[100];
} STU_INFO;
typedef struct {
   char *college_name;
   char university_name[100];
} COLLEGE;

typedef struct {
   DOB *dob;
   STU_INFO stu_info;
   COLLEGE college;
```

```
} STUDENT;

void readStudent(STUDENT *student) {
    char buffer[100];

    student->dob = (DOB *)malloc(sizeof(DOB));
    student->dob->month = (int *)malloc(sizeof(int));
    printf("Enter day of birth: ");
    scanf("%d", &student->dob->day);

    printf("Enter month of birth (as a number): ");
    scanf("%d", student->dob->month);

    printf("Enter year of birth: ");
    scanf("%d", &student->dob->year);
    printf("Enter registration number: ");
    scanf("%d", &student->stu_info.reg_no);

    printf("Enter name: ");
    scanf(" %[^\n]", buffer);
    student->stu_info.name = (char *)malloc((strlen(buffer) + 1) * sizeof(char));
    strcpy(student->stu_info.name, buffer);

    printf("Enter address: ");
    scanf(" %[^\n]", student->stu_info.address);
    printf("Enter college name: ");
    scanf(" %[^\n]", buffer);
    student->college.college_name = (char *)malloc((strlen(buffer) + 1) *
sizeof(char));
    strcpy(student->college.college_name, buffer);

    printf("Enter university name: ");
    scanf(" %[^\n]", student->college.university_name);
}
```

```
void displayStudent(const STUDENT *student) {
    // Display Date of Birth
    printf("Date of Birth: %02d/%02d/%04d\n", student->dob->day, *(student-
>dob->month), student->dob->year);
    printf("Registration Number: %d\n", student->stu_info.reg_no);
    printf("Name: %s\n", student->stu_info.name);
    printf("Address: %s\n", student->stu_info.address);

    // Display College Information
    printf("College Name: %s\n", student->college.college_name);
    printf("University Name: %s\n", student->college.university_name);
}

int main() {
    STUDENT *student;
    student = (STUDENT *)malloc(sizeof(STUDENT));
    readStudent(student);
    printf("\nStudent Details:\n");
    displayStudent(student);
    free(student->dob->month);
    free(student->dob);
    free(student->stu_info.name);
    free(student->college.college_name);
    free(student);
    return 0;
}
```

III. ADDITIONAL EXERCISES:

1) Design an application using suitable data structure to automate the process of class representative election. The application should take the inputs – No of students(voters) present in the class and No of Candidates and their name(s). During the voting process, the faculty on duty of conduction of election, need to initiate the voting (say by pressing a particular key during which a beep sound is heard). Then, a student can come and cast his/her vote against the candidate by entering the candidate's serial number. After casting a vote, a beep sound is given to confirm that the voting is done. This process is repeated until all students cast their vote. At the end, the total votes obtained by each candidate should be displayed along with number of foul votes. the candidate who has won the maximum number of votes should be declared as class representative. (Hint: Use array of integers with array index as candidate number)

2) Design a suitable data structure for TEXT editor which includes operations such as inserting a line after the enter key is pressed and delete a line, search for a word in a line. When the key F2 is pressed, the content of the buffer should be saved in a file and displayed on the screen (Hint: Array of pointers to strings can be used as Data structure).

LAB NO:5 Date:

STACK CONCEPTS

In this lab, student will be able to:

- Understand stack as a data structure
- Design programs using stack concepts

I. SOLVED EXERCISE:

1) Write a c program to check if the given parenthesized expression has properly matching open and closing parenthesis.

Description: We use a stack to check the expression for matching opening and closing parenthesis. Here, the expression is scanned from start to end if an opening brace is encountered then it is pushed on to the stack. When a closing parenthesis is encountered a pop operation is performed. Ideally, if the number of opening andclosing braces matches, then the stack will be empty after checking the entire expression.

Algorithm:

Step1: Set balanced to true

Step2: Set symbol to the first character in current expression

Step3: while (there are still more characters AND expression is still balanced)

```
if (symbol is an opening symbol)
    Push symbol onto the stack
else if (symbol is a closing symbol)
    if the stack is empty
```

```
                    Set balanced to false
                Else
                    Set openSymbol to the character at the top of the stack
                    Pop the stack
                    Set balanced to (symbol matches openSymbol)
            Set symbol to next character in current expression
Step4:  if (balanced)
                Write "Expression is well formed."
            else
                Write "Expression is not well formed."
   Step5: stop
```

Program:

File name: stack_operations.h

```
  # define MAX 10
  # define true 1
  # define false 0
  /* Structure definition */
  typedef struct
  {
        char item[MAX];
        int top;
  }stack;
  void push(stack *ps,char x);
  char pop(stack *ps);
  int empty(stack *ps);
  /* Push operation */
```

```
void push(stack *ps,char x)
{
        if (ps->top!=MAX-1)
        {
                ps->top++;
                ps->item[ps->top]=x;
        }
}

 /* Pop operation */
char pop(stack *ps)
{
        if(!empty(ps))
          return(ps->item[ps->top--]);
}

/* Stack empty operation */
int empty(stack *ps)
{
        if (ps->top==-1)
                return(true);
        else
                return(false);
}
```

File name: check_expr.c

```
#include <stdio.h>
#include <stdlib.h>
#include "stack_operations.h"
void main()
{
	char expn[25],c,d;
	int i=0;
	stack s;
	s.top=-1;
	printf("\n Enter the expression: ");
	gets(expn);
	while((c=expn[i++])!='\0')
	{
		if(c=='(')
			push(&s,c);
		else
			if(c==')')
			{
				d=pop(&s);
				if(d!='(')
				{
				printf("\n Invalid Expression");
				break;
				}
			}
	}
```

```
        if(empty(&s))
                printf("\n Balanced Expression");
        else
                printf("\n Not a Balanced  Expression");
 }
```

Sample Input and Output

Run 1:

Enter the expression: (a+b)+(c*d*(a-b)

Not a Balanced Expression

Run 2:

Enter the expression: (a+b)+(c*d*(a-b))

Balanced Expression

Lab 5 Programs few with solution – few do by yourself

1) Implement a menu driven program to define a stack of characters. Include push, pop and display functions. Also include functions for checking error conditions such as underflow and overflow (ref. figure 1) by defining isEmpty and isFull functions. Use these function in push, pop and display functions appropriately. Use type defined structure to define a STACK containing a character array and an integer top. Do not use global variables.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
typedef struct {
    char items[MAX];
    int top;
} STACK;
void initialize(STACK *s) {
    s->top = -1;
}

int isFull(STACK *s) {
    return s->top == MAX - 1;
}
int isEmpty(STACK *s) {
    return s->top == -1;
}
void push(STACK *s, char value) {
    if (isFull(s)) {
        printf("Stack Overflow! Cannot push '%c'.\n", value);
    } else {
        s->items[++(s->top)] = value;
        printf("Pushed '%c' onto the stack.\n", value);
    }
}
char pop(STACK *s) {
    if (isEmpty(s)) {
        printf("Stack Underflow! Cannot pop.\n");
        return '\0'; // Return a null character to indicate failure
    } else {
        char popped = s->items[(s->top)--];
        printf("Popped '%c' from the stack.\n", popped);
        return popped;
    }
}

void display(STACK *s) {
```

```
    if (isEmpty(s)) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements:\n");
        for (int i = s->top; i >= 0; i--) {
            printf("%c\n", s->items[i]);
        }
    }
}

int main() {
    STACK stack;
    char value;
    int choice;

    initialize(&stack);
    do {
        printf("\nMenu:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter a character to push: ");
                scanf(" %c", &value);
                push(&stack, value);
                break;

            case 2:
                pop(&stack);
                break;

            case 3:
                display(&stack);
                break;

            case 4:
                printf("Exiting...\n");
                break;

            default:
```

```
            printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

2) Convert a given decimal number to binary using stack.

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

#define MAX 32
typedef struct {
    int items[MAX];
    int top;
} STACK;


void initialize(STACK *s) {
    s->top = -1;
}

int isFull(STACK *s) {
    return s->top == MAX - 1;
}

int isEmpty(STACK *s) {
    return s->top == -1;
}

void push(STACK *s, int value) {
    if (isFull(s)) {
        printf("Stack Overflow! Cannot push '%d'.\n", value);
    } else {
        s->items[++(s->top)] = value;
    }
}

int pop(STACK *s) {
    if (isEmpty(s)) {
        printf("Stack Underflow! Cannot pop.\n");
        return -1; // Return -1 to indicate failure
    } else {
        return s->items[(s->top)--];
```

```
    }
}

void decimalToBinary(int decimal) {
    STACK stack;
    initialize(&stack);

    if (decimal == 0) {
        printf("Binary: 0\n");
        return;
    }

    while (decimal > 0) {
        push(&stack, decimal % 2);
        decimal /= 2;
    }

    printf("Binary: ");
    while (!isEmpty(&stack)) {
        printf("%d", pop(&stack));
    }
    printf("\n");
}

int main() {
    int decimal;

    // Read a decimal number from the user
    printf("Enter a decimal number: ");
    scanf("%d", &decimal);
    decimalToBinary(decimal);

    return 0;
}
```

3) Determine whether a given string is palindrome or not using stack.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX 100
typedef struct {
```

```
    char items[MAX];
    int top;
} STACK;
void initialize(STACK *s) {
    s->top = -1;
}

int isFull(STACK *s) {
    return s->top == MAX - 1;
}

int isEmpty(STACK *s) {
    return s->top == -1;
}

void push(STACK *s, char value) {
    if (isFull(s)) {
        printf("Stack Overflow! Cannot push '%c'.\n", value);
    } else {
        s->items[++(s->top)] = value;
    }
}

char pop(STACK *s) {
    if (isEmpty(s)) {
        printf("Stack Underflow! Cannot pop.\n");
        return '\0';
    } else {
        return s->items[(s->top)--];
    }
}

int isPalindrome(char str[]) {
    int length = strlen(str);
    STACK stack;
    initialize(&stack);

    for (int i = 0; i < length; i++) {
        push(&stack, str[i]);
    }

    for (int i = 0; i < length; i++) {
        if (tolower(str[i]) != tolower(pop(&stack))) {
            return 0;
        }
```

```
    }

    return 1;
}

int main() {
    char str[MAX];

    // Read a string from the user
    printf("Enter a string: ");
    scanf("%s", str);
    if (isPalindrome(str)) {
        printf("The string '%s' is a palindrome.\n", str);
    } else {
        printf("The string '%s' is not a palindrome.\n", str);
    }

    return 0;
}
```

4) Given an array *arr* with n elements and a number k, k<n. The task is to delete k elements which are smaller than next element (i.e., we delete arr[i] if arr[i] < arr[i+1]) or become smaller than next because next element is deleted. Example:
Input: arr[] = {20, 10, 25, 30, 40}, k = 2
Output: 25 30 40 - **Solve by yourself**

Additional questions

1) Write a program to implement multiple stacks (say n stacks) in a single array. Implement ADD(i. X) and DELETE(i) to add X and delete an element from stack i, 1<= i <= n.

2) Given an array, print the Next Greater Element (NGE) for every element using stack. The Next Greater Element for an element x is the first greater element on the right side of x in array. Elements for which no greater element exist, consider next greater element as -1. For the input array [13, 7, 6, 12}, the next greater elementsfor each element are as follows.

Element	NGE

13 → -1
7 → 12
6 → 12
12 → -1

LAB NO: 6 Date:

STACK APPLICATIONS

Objectives:

In this lab, student will be able to:

- Identify the need for Stack data structure in a given problem.
- Develop c programs applying stack concepts

I. SOLVED EXERCISE:

1) Program for evaluation of postfix expression in C

Algorithm for Evaluation of Postfix Expression

Create an empty stack and start scanning the postfix expression from left to right.

- If the element is an operand, push it into the stack.
- If the element is an operator **O**, pop twice and get A and B respectively. Calculate B**O**A and push it back to the stack.
- When the expression is ended, the value in the stack is the final answer.

Evaluation of a postfix expression using a stack is explained in below example (fig. 2):

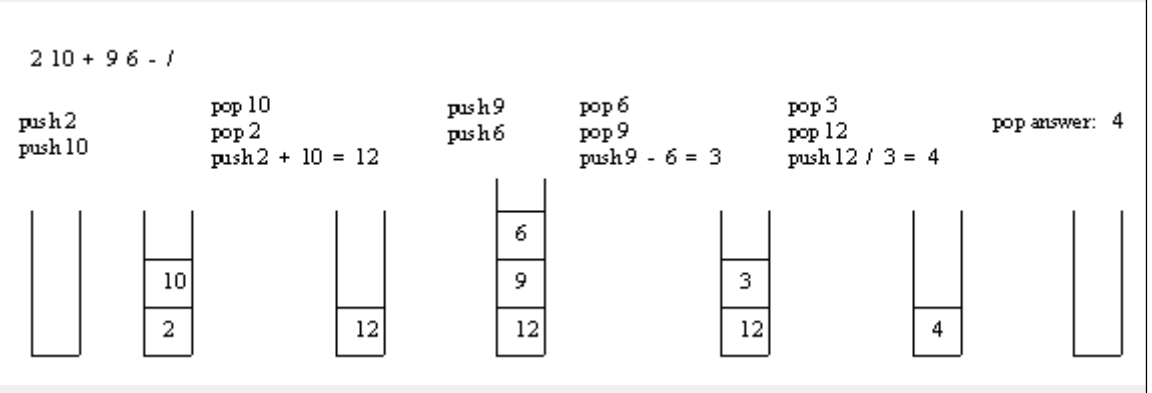


Figure 2: Illustrates the evaluation of a postfix expression using a stack

File name: eval_postfix_fun.h

```
#define MAX 20

typedef struct stack
{
   int data[MAX];
   int top;
}stack;

void init(stack *);
int empty(stack *);
int full(stack *);
int pop(stack *);
void push(stack *,int);
int evaluate(char x,int op1,int op2);

int evaluate(char x,int op1,int op2)
{
   if(x=='+')
      return(op1+op2);
   if(x=='-')
      return(op1-op2);
   if(x=='*')
      return(op1*op2);
   if(x=='/')
      return(op1/op2);
   if(x=='%')
      return(op1%op2);
}

void init(stack *s)
{
   s->top=-1;
}

int empty(stack *s)
{
   if(s->top==-1)
      return(1);
```

```
    return(0);
}

int full(stack *s)
{
    if(s->top==MAX-1)
        return(1);

    return(0);
}

void push(stack *s,int x)
{
    s->top=s->top+1;
    s->data[s->top]=x;
}

int pop(stack *s)
{
    int x;
    x=s->data[s->top];
    s->top=s->top-1;
    return(x);
}
```

File name:eval_postfix_expr.c

```
#include<stdio.h>
#include "eval_postfix_fun.h"
int main()
{
    stack s;
    char x;
    int op1,op2,val;
    init(&s);
    printf("Enter the expression(eg: 59+3*)\nsingle digit operand and operators
            only:");

    while((x=getchar())!='\n')
    {
```

```
            if(isdigit(x))
               push(&s,x-'0');       /*x-'0' for removing the effect of ascii  */
            else
            {
               op2=pop(&s);
               op1=pop(&s);
               val=evaluate(x,op1,op2);
               push(&s,val);
            }
         }
         val=pop(&s);
         printf("\nvalue of expression=%d",val);
         return 0;
      }
```

Sample Input and Output:
Enter the expression(eg: 59+3*)
single digit operand and operators only: 12+3*
value of expression= 9

II. LAB EXERCISES:

Write a C program to:

1) Evaluate a given prefix expression using stack.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#define MAX 100
typedef struct {
   int top;
   int items[MAX];
} Stack;

void initStack(Stack *s) {
   s->top = -1;
}

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

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

```
void push(Stack *s, int value) {
    if (isFull(s)) {
        printf("Stack Overflow\n");
        return;
    }
    s->items[++(s->top)] = value;
}

int pop(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack Underflow\n");
        return -1; // Return -1 for underflow, error condition
    }
    return s->items[(s->top)--];
}

// Function to evaluate a prefix expression
int evaluatePrefix(char prefix[]) {
    Stack s;
    initStack(&s);

    int i;
    // Start from the end of the prefix expression
    for (i = strlen(prefix) - 1; i >= 0; i--) {
        char ch = prefix[i];

        // If the character is an operand (number), push it to the stack
        if (isdigit(ch)) {
            push(&s, ch - '0'); // Convert character to integer
        }
        // If the character is an operator, pop two operands from the stack, apply the operator,
and push the result back
        else if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^') {
            int op1 = pop(&s);
            int op2 = pop(&s);

            switch (ch) {
                case '+': push(&s, op1 + op2); break;
                case '-': push(&s, op1 - op2); break;
                case '*': push(&s, op1 * op2); break;
                case '/': push(&s, op1 / op2); break;
                case '^': push(&s, pow(op1, op2)); break;
            }
        }
    }

    // The result of the expression is now at the top of the stack
    return pop(&s);
}
```

```
int main() {
    char prefix[MAX];

    printf("Enter a prefix expression: ");
    gets(prefix); // Read the prefix expression from the user

    int result = evaluatePrefix(prefix); // Evaluate the prefix expression

    printf("Result of the prefix expression is: %d\n", result); // Output the result

    return 0;
}
```

2) Convert an infix expression to prefix.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>  // For isdigit() and isalpha()

#define MAX 100  // Maximum size of stack

// Stack structure
typedef struct {
    int top;
    char items[MAX];
} Stack;

// Function to initialize the stack
void initStack(Stack *s) {
    s->top = -1;
}

// Function to check if the stack is empty
int isEmpty(Stack *s) {
    return s->top == -1;
```

```
}

// Function to check if the stack is full
int isFull(Stack *s) {
    return s->top == MAX - 1;
}

// Function to push an element onto the stack
void push(Stack *s, char value) {
    if (isFull(s)) {
        printf("Stack Overflow\n");
    } else {
        s->items[++(s->top)] = value;
    }
}

// Function to pop an element from the stack
char pop(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack Underflow\n");
        return '\0';
    } else {
        return s->items[(s->top)--];
    }
}

// Function to peek the top element of the stack
char peek(Stack *s) {
    if (isEmpty(s)) {
        return '\0';
    } else {
        return s->items[s->top];
    }
```

```
}

// Function to check the precedence of operators
int precedence(char ch) {
    switch (ch) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return 0;
    }
}

// Function to check if a character is an operator
int isOperator(char ch) {
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^');
}

// Function to reverse a string
void reverseString(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - i - 1];
        str[len - i - 1] = temp;
    }
}
```

```
// Function to convert infix to postfix expression
void infixToPostfix(char infix[], char postfix[]) {
    Stack s;
    initStack(&s);
    int i, k = 0;

    for (i = 0; i < strlen(infix); i++) {
        char ch = infix[i];

        if (isdigit(ch) || isalpha(ch)) {
            // If the character is an operand, add it to postfix expression
            postfix[k++] = ch;
        } else if (ch == '(') {
            // If the character is '(', push it to stack
            push(&s, ch);
        } else if (ch == ')') {
            // If the character is ')', pop and output from the stack until '(' is found
            while (!isEmpty(&s) && peek(&s) != '(') {
                postfix[k++] = pop(&s);
            }
            pop(&s);  // Pop '(' from the stack
        } else if (isOperator(ch)) {
            // If the character is an operator
            while (!isEmpty(&s) && precedence(peek(&s)) >= precedence(ch)) {
                postfix[k++] = pop(&s);
            }
            push(&s, ch);
        }
    }

    // Pop all remaining operators from the stack
    while (!isEmpty(&s)) {
        postfix[k++] = pop(&s);
```

```
    }

    postfix[k] = '\0';  // End the postfix expression with a null character
}

// Function to convert infix to prefix expression
void infixToPrefix(char infix[], char prefix[]) {
    // Step 1: Reverse the infix expression
    reverseString(infix);

    // Step 2: Replace '(' with ')' and vice versa
    for (int i = 0; i < strlen(infix); i++) {
        if (infix[i] == '(') {
            infix[i] = ')';
        } else if (infix[i] == ')') {
            infix[i] = '(';
        }
    }

    // Step 3: Convert the modified infix to postfix
    char postfix[MAX];
    infixToPostfix(infix, postfix);

    // Step 4: Reverse the postfix expression to get the prefix expression
    reverseString(postfix);
    strcpy(prefix, postfix);
}

// Main function
int main() {
    char infix[MAX], prefix[MAX];

    printf("Enter infix expression: ");
```

```
    gets(infix);  // Input the infix expression

    infixToPrefix(infix, prefix);  // Convert infix to prefix

    printf("Prefix expression: %s\n", prefix);  // Output the prefix expression

    return 0;
}
```

3) Implement two stacks in an array.

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

#define MAX 100
// Structure to represent two stacks
typedef struct {
    int arr[MAX];  // Array to hold the stack elements
    int top1;      // Top for Stack 1
    int top2;      // Top for Stack 2
} TwoStacks;

// Function to initialize two stacks
void initTwoStacks(TwoStacks *stacks) {
    stacks->top1 = -1;          // Initialize top1 for Stack 1
    stacks->top2 = MAX;         // Initialize top2 for Stack 2
}

// Function to push an element to Stack 1
void push1(TwoStacks *stacks, int value) {
    // Check for overflow
    if (stacks->top1 < stacks->top2 - 1) {
        stacks->arr[++(stacks->top1)] = value;  // Increment top1 and add the value
    } else {
        printf("Stack Overflow in Stack 1\n");
```

```
    }
}

// Function to push an element to Stack 2
void push2(TwoStacks *stacks, int value) {
    // Check for overflow
    if (stacks->top1 < stacks->top2 - 1) {
        stacks->arr[--(stacks->top2)] = value;  // Decrement top2 and add the value
    } else {
        printf("Stack Overflow in Stack 2\n");
    }
}

// Function to pop an element from Stack 1
int pop1(TwoStacks *stacks) {
    // Check for underflow
    if (stacks->top1 >= 0) {
        return stacks->arr[(stacks->top1)--];  // Return the top element and decrement top1
    } else {
        printf("Stack Underflow in Stack 1\n");
        return -1;  // Return -1 for underflow
    }
}

// Function to pop an element from Stack 2
int pop2(TwoStacks *stacks) {
    // Check for underflow
    if (stacks->top2 < MAX) {
        return stacks->arr[(stacks->top2)++];  // Return the top element and increment top2
    } else {
        printf("Stack Underflow in Stack 2\n");
        return -1;  // Return -1 for underflow
    }
}
```

```
// Function to display elements of Stack 1
void displayStack1(TwoStacks *stacks) {
    printf("Stack 1: ");
    for (int i = 0; i <= stacks->top1; i++) {
        printf("%d ", stacks->arr[i]);
    }
    printf("\n");
}

// Function to display elements of Stack 2
void displayStack2(TwoStacks *stacks) {
    printf("Stack 2: ");
    for (int i = MAX - 1; i >= stacks->top2; i--) {
        printf("%d ", stacks->arr[i]);
    }
    printf("\n");
}

// Main function
int main() {
    TwoStacks stacks;
    initTwoStacks(&stacks);  // Initialize two stacks

    // Perform some operations
    push1(&stacks, 10);
    push2(&stacks, 20);
    push1(&stacks, 30);
    push2(&stacks, 40);
    push1(&stacks, 50);

    displayStack1(&stacks);  // Display Stack 1
    displayStack2(&stacks);  // Display Stack 2
```

```
    printf("Popped from Stack 1: %d\n", pop1(&stacks));  // Pop from Stack 1
    printf("Popped from Stack 2: %d\n", pop2(&stacks));  // Pop from Stack 2
    displayStack1(&stacks);  // Display Stack 1 after popping
    displayStack2(&stacks);  // Display Stack 2 after popping
    return 0;
}
```

4) To convert a prefix expression to postfix using stack.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>  // For isdigit() and isalpha()

#define MAX 100
typedef struct {
    int top;
    char items[MAX][MAX];  // Stack array to hold strings
} Stack;

// Function to initialize the stack
void initStack(Stack *s) {
    s->top = -1;
}

// Function to check if the stack is empty
int isEmpty(Stack *s) {
    return s->top == -1;
}

// Function to push an element onto the stack
void push(Stack *s, char *value) {
    if (s->top >= MAX - 1) {
        printf("Stack Overflow\n");
```

```
    } else {
        strcpy(s->items[++(s->top)], value);
    }
}

// Function to pop an element from the stack
void pop(Stack *s, char *result) {
    if (isEmpty(s)) {
        printf("Stack Underflow\n");
    } else {
        strcpy(result, s->items[(s->top)--]);
    }
}

// Function to reverse a string
void reverseString(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - i - 1];
        str[len - i - 1] = temp;
    }
}

// Function to check if a character is an operator
int isOperator(char ch) {
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^');
}

// Function to convert prefix to postfix expression
void prefixToPostfix(char prefix[], char postfix[]) {
    Stack s;
    initStack(&s);
```

```
    int length = strlen(prefix);

    // Reverse the prefix expression to process from left to right
    reverseString(prefix);

    for (int i = 0; i < length; i++) {
        char ch = prefix[i];

        // If the character is an operand (number or letter), push it to the stack
        if (isalnum(ch)) {
            char operand[2] = {ch, '\0'};  // Convert character to string
            push(&s, operand);
        }
        // If the character is an operator, pop two operands from the stack
        else if (isOperator(ch)) {
            char op1[MAX], op2[MAX];

            pop(&s, op1);  // First operand
            pop(&s, op2);  // Second operand

            // Form a new postfix expression: op1 op2 operator
            char temp[MAX];
            sprintf(temp, "%s%s%c", op1, op2, ch);  // Combine into postfix form

            // Push the resulting expression back to the stack
            push(&s, temp);
        }
    }

    // The result of the expression is now at the top of the stack
    pop(&s, postfix);
}
int main() {
```

```
    char prefix[MAX], postfix[MAX];
    printf("Enter a prefix expression: ");
    gets(prefix);  // Input the prefix expression
    prefixToPostfix(prefix, postfix);  // Convert prefix to postfix
    printf("Postfix expression: %s\n", postfix);  // Output the postfix expression
    return 0;
}
```

III. ADDITIONAL EXERCISES:

1) Convert an infix expression to postfix.
2) Reverse a stack using recursion [Note: Only the IsEmpty, Push and Pop operationsare allowed on stack]

Understand queue as a data structure, Design programs using queue concepts

LAB NO: 7 **Date:**

QUEUE CONCEPTS

I. SOLVED EXERCISE:

1) Implement a queue of integers. Include functions insertq, deleteq and displayq.

Description:

Whenever we perform an insertion the rear pointer is incremented by 1 and front remains the same. Whenever a deletion is performed the front is incremented by one. But in real life problem the front pointer will be in the same position the object which is in the queue shifts to the front. But when we implement a queue in computer the front moves, this is because if we shift the elements to the front the time complexity increases. To implement a linear queue we consider two pointers or markers front and rear. Initial value of front and rear is assumed to be -1

Algorithm: Queue Insert

Step 1. Check for overflow

If R== N-1

Then write (‘OVERFLOW’) Return

Step2. else Increment rear pointer

R= R +1

Step3: Insert element

Q[R] = val Step4: If F = -1

then F = 0

Return.

Algorithm: Queue Delete Step1: Check for underflow

If F = = -1

Then Write (‘UNDERFLOW’) Return

Step2: Delete an element

val = Q[F] Step3: Queue empty?

if F > R

then F=R =-1 else F= F +1

Step4. Return an element

Return(val)

Step5: Stop

Program:

File name: queue_fun.h

```
#define MAX 20 typedef struct {
int x[MAX]; int front;
int rear;
} queue;
void insertq(queue *, int); void displayq(queue);
int deleteq(queue *);
void insertq(queue * q,int x)
{
if(q->rear==MAX)
{
printf("\nOverflow\n");
```

```
}
else
{
q->x[++q->rear]=x; if(q->front==-1)
{
q->front=0;
}
}

}
int deleteq(queue * q)
{
int x;
if(q->front==-1)
{
printf("\nUnderflow!!!\n");
}
else if(q->front==q->rear)
{
x=q->x[q->front];
q->front=q->rear=-1; return x;
}
else
{
return q->x[q->front++];
}
}
void displayq(queue q)
{
int i;
if(q.front==-1&&q.rear==-1)
{
printf("\nQueue is Empty!!!");

}
else
{
printf("\nQueue is:\n"); for(i=q.front;i<=q.rear;i++)
{
printf("%d\n",q.x[i]);
}
}
}
```
File name: queue.c #include <stdio.h> #include "queue_fun.h"
```
int main()
{
queue q; q.front=-1; q.rear=-1;
int ch,x,flag=1; while(flag)
{
printf("\n\n1. Insert Queue\n2. Delete Queue\n3. Display Queue\n4. Exit\n\n"); printf("Enter
```

```
your choice: ");

scanf("%d",&ch); switch(ch)
{
case 1:
printf("\nEnter the Element:"); scanf("%d",&x); insertq(&q,x);
break; case 2:
x=deleteq(&q);
printf("\nRemoved %d from the Queue\n",x); break;
case 3:
displayq(q); break;
case 4:
flag=0; break;
default:
printf("\nWrong choice!!! Try Again.\n");
}
}
return 0;
}
```

II. LAB EXERCISES:

1) Implement a circular queue of Strings using structures. Include functions insertcq, deletecq and displaycq.

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

#define MAX 5  // Maximum size of the circular queue

// Structure to represent a circular queue
typedef struct {
    char items[MAX][50];  // Array to store strings
    int front, rear;      // Indices for front and rear
} CircularQueue;

// Function to initialize the circular queue
void initializeQueue(CircularQueue *q) {
    q->front = -1;
    q->rear = -1;
}

// Function to check if the queue is full
int isFull(CircularQueue *q) {
    return ((q->rear + 1) % MAX == q->front);
}

// Function to check if the queue is empty
int isEmpty(CircularQueue *q) {
    return (q->front == -1);
```

```
}

// Function to insert a string into the circular queue
void insertcq(CircularQueue *q, char *str) {
    if (isFull(q)) {
        printf("Queue is Full! Cannot insert '%s'.\n", str);
        return;
    }

    // Calculate the new rear position
    q->rear = (q->rear + 1) % MAX;
    strcpy(q->items[q->rear], str);  // Copy string to the new position

    // Update front if it was initially -1 (indicating the queue was empty)
    if (q->front == -1) {
        q->front = 0;
    }

    printf("Inserted: %s\n", str);
}

// Function to delete a string from the circular queue
void deletecq(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is Empty! Cannot delete.\n");
        return;
    }

    printf("Deleted: %s\n", q->items[q->front]);

    // Check if the queue will be empty after this deletion
    if (q->front == q->rear) {
        q->front = -1;  // Queue becomes empty
        q->rear = -1;
    } else {
        // Move the front forward
        q->front = (q->front + 1) % MAX;
    }
}

// Function to display all elements in the circular queue
void displaycq(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is Empty!\n");
        return;
    }

    printf("Queue elements are:\n");
    int i = q->front;
    while (1) {
```

```
        printf("%s ", q->items[i]);
        if (i == q->rear) {
            break;  // Stop when the rear is reached
        }
        i = (i + 1) % MAX;  // Move circularly in the queue
    }
    printf("\n");
}

    int main() {
    CircularQueue q;
    initializeQueue(&q);

    insertcq(&q, "Abc");
    insertcq(&q, "Bcd");
    insertcq(&q, "Cde");
    insertcq(&q, "Def");
    insertcq(&q, "Efg");

    displaycq(&q);

    deletecq(&q);
    deletecq(&q);

    displaycq(&q);

    insertcq(&q, "Fgh");
    insertcq(&q, "Ghk");

    displaycq(&q);

    return 0;
}
```

2) Implement two circular queues of integers in a single array where first queue will run from 0 to N/2 and second queue will run from N/2+1 to N-1 where N is the size of the array.

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

#define N 10  // Total size of the array
#define MID N/2  // Middle index for splitting the array

// Structure to represent two circular queues
typedef struct {
    int arr[N];          // Array to hold both queues
    int front1, rear1;   // Front and rear for the first queue
    int front2, rear2;   // Front and rear for the second queue
} TwoCircularQueues;
```

```
// Function to initialize both circular queues
void initializeQueues(TwoCircularQueues* q) {
    q->front1 = -1;             // Initialize front and rear for Queue 1
    q->rear1 = -1;
    q->front2 = MID + 1;        // Initialize front and rear for Queue 2
    q->rear2 = MID + 1;
}

// Function to check if the first queue is full
int isFullQueue1(TwoCircularQueues* q) {
    return ((q->rear1 + 1) % (MID + 1) == q->front1);
}

// Function to check if the second queue is full
int isFullQueue2(TwoCircularQueues* q) {
    return ((q->rear2 + 1) % N == q->front2);
}

// Function to check if the first queue is empty
int isEmptyQueue1(TwoCircularQueues* q) {
    return (q->front1 == -1);
}

// Function to check if the second queue is empty
int isEmptyQueue2(TwoCircularQueues* q) {
    return (q->front2 == MID + 1);
}

// Function to insert an element into the first circular queue
void insertcq1(TwoCircularQueues* q, int value) {
    if (isFullQueue1(q)) {
        printf("Queue 1 is Full! Cannot insert %d.\n", value);
        return;
    }
    if (isEmptyQueue1(q)) {  // If queue is initially empty
        q->front1 = 0;
    }
    q->rear1 = (q->rear1 + 1) % (MID + 1);
    q->arr[q->rear1] = value;
    printf("Inserted %d into Queue 1.\n", value);
}

// Function to insert an element into the second circular queue
void insertcq2(TwoCircularQueues* q, int value) {
    if (isFullQueue2(q)) {
        printf("Queue 2 is Full! Cannot insert %d.\n", value);
        return;
    }
    if (isEmptyQueue2(q)) {  // If queue is initially empty
```

```
        q->front2 = MID + 1;
    }
    q->rear2 = (q->rear2 + 1) % N;
    q->arr[q->rear2] = value;
    printf("Inserted %d into Queue 2.\n", value);
}

// Function to delete an element from the first circular queue
void deletecq1(TwoCircularQueues* q) {
    if (isEmptyQueue1(q)) {
        printf("Queue 1 is Empty! Cannot delete.\n");
        return;
    }
    printf("Deleted %d from Queue 1.\n", q->arr[q->front1]);
    if (q->front1 == q->rear1) {  // Queue has only one element
        q->front1 = -1;
        q->rear1 = -1;
    } else {
        q->front1 = (q->front1 + 1) % (MID + 1);
    }
}

// Function to delete an element from the second circular queue
void deletecq2(TwoCircularQueues* q) {
    if (isEmptyQueue2(q)) {
        printf("Queue 2 is Empty! Cannot delete.\n");
        return;
    }
    printf("Deleted %d from Queue 2.\n", q->arr[q->front2]);
    if (q->front2 == q->rear2) {  // Queue has only one element
        q->front2 = MID + 1;
        q->rear2 = MID + 1;
    } else {
        q->front2 = (q->front2 + 1) % N;
    }
}

// Function to display elements of the first circular queue
void displaycq1(TwoCircularQueues* q) {
    if (isEmptyQueue1(q)) {
        printf("Queue 1 is Empty!\n");
        return;
    }
    printf("Queue 1 elements: ");
    int i = q->front1;
    while (1) {
        printf("%d ", q->arr[i]);
        if (i == q->rear1) break;
        i = (i + 1) % (MID + 1);
    }
```

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

// Function to display elements of the second circular queue
void displaycq2(TwoCircularQueues* q) {
    if (isEmptyQueue2(q)) {
        printf("Queue 2 is Empty!\n");
        return;
    }
    printf("Queue 2 elements: ");
    int i = q->front2;
    while (1) {
        printf("%d ", q->arr[i]);
        if (i == q->rear2) break;
        i = (i + 1) % N;
    }
    printf("\n");
}

// Main function to demonstrate the functionality
int main() {
    TwoCircularQueues q;
    initializeQueues(&q);

    // Inserting elements into both queues
    insertcq1(&q, 10);
    insertcq1(&q, 20);
    insertcq1(&q, 30);
    insertcq2(&q, 40);
    insertcq2(&q, 50);
    insertcq2(&q, 60);

    // Displaying elements of both queues
    displaycq1(&q);
    displaycq2(&q);

    // Deleting elements from both queues
    deletecq1(&q);
    deletecq2(&q);

    // Displaying elements after deletion
    displaycq1(&q);
    displaycq2(&q);

    return 0;
}
```

3) Vignesh and his wife Lata are facing a cash crisis. They go to the nearby ATM to get some cash. There are 3 ATMs inside the same room. People are standing in queue outside, and go inside the room in groups of 3 to the ATMs, fetch their money and come out. Lata has an irrational fear in getting money from ATM that her ATM pin will somehow be stolen and her money will be lost. So she will always like to go into room with Vignesh. Vignesh is standing at position K in line, immediately followed by lata (at position K+1). Can you tell whether Vignesh and Lata both will be able to get money in such a way that Lata does not feel insecure? Using queue, find whether they can get money for the given set of N and K. Input: the first line contains an integer T denoting the number of test cases. T test cases follow. The only line of each test case contains two spaces separated integers N and K , Where N denotes number of people in the queue. And K denotes the position of Vignesh . Constraints

1<=T<=100 3<=N<=100 1<=K<=N-1, N is divisible by 3.
Sample Input
2
3 1
6 3
Output
Yes
No

```
#include <stdio.h>

void checkGroups(int N, int K) {
    // Check if Vignesh (at position K) and Lata (at position K+1) are in the same group of 3
    if ((K - 1) / 3 == K / 3) {
        printf("Yes\n");
    } else {
        printf("No\n");
    }
}

int main() {
    int T; // Number of test cases
    scanf("%d", &T);

    while (T--) {
        int N, K;
        scanf("%d %d", &N, &K);

        checkGroups(N, K);
    }

    return 0;
}
```

4) Implement a queue with two stacks without transferring the elements of the second stack back to stack one. (use stack1 as an input stack and stack2 as an output stack)

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

#define MAX 100  // Maximum size of the stack

// Stack structure
typedef struct {
    int top;
    int items[MAX];
} Stack;

// Function to initialize the stack
void initStack(Stack *s) {
    s->top = -1;
}

// Function to check if the stack is empty
int isEmpty(Stack *s) {
    return s->top == -1;
}

// Function to check if the stack is full
int isFull(Stack *s) {
    return s->top == MAX - 1;
}

// Function to push an element onto the stack
void push(Stack *s, int value) {
    if (isFull(s)) {
        printf("Stack Overflow\n");
    } else {
        s->items[++(s->top)] = value;
    }
}

// Function to pop an element from the stack
int pop(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack Underflow\n");
        return -1;  // Indicate an error
    } else {
        return s->items[(s->top)--];
    }
}

// Queue structure using two stacks
typedef struct {
    Stack stack1;  // Input stack
```

```
    Stack stack2;  // Output stack
} Queue;

// Function to initialize the queue
void initQueue(Queue *q) {
    initStack(&q->stack1);
    initStack(&q->stack2);
}

// Function to add an element to the queue
void enqueue(Queue *q, int value) {
    push(&q->stack1, value);  // Always push to stack1
    printf("Enqueued: %d\n", value);
}

// Function to remove an element from the queue
int dequeue(Queue *q) {
    if (isEmpty(&q->stack2)) {  // If stack2 is empty, transfer all elements from stack1
to stack2
        while (!isEmpty(&q->stack1)) {
            int temp = pop(&q->stack1);
            push(&q->stack2, temp);
        }
    }

    if (isEmpty(&q->stack2)) {  // If stack2 is still empty, queue is empty
        printf("Queue is Empty! Cannot dequeue.\n");
        return -1;
    }

    int dequeuedValue = pop(&q->stack2);
    printf("Dequeued: %d\n", dequeuedValue);
    return dequeuedValue;
}

// Function to display the current elements in the queue
void displayQueue(Queue *q) {
    // To display the queue correctly from front to rear, we need to display elements in
stack2 first
    // then display elements in stack1 in reverse order.
    printf("Queue elements: ");

    // Display elements in stack2 (front part of the queue)
    for (int i = q->stack2.top; i >= 0; i--) {
        printf("%d ", q->stack2.items[i]);
    }

    // Display elements in stack1 (rear part of the queue) in reverse order
    for (int i = 0; i <= q->stack1.top; i++) {
        printf("%d ", q->stack1.items[i]);
```

```
    }

    printf("\n");
}

// Main function to demonstrate queue operations using two stacks
int main() {
    Queue q;
    initQueue(&q);

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
    displayQueue(&q);

    dequeue(&q);
    displayQueue(&q);

    enqueue(&q, 40);
    displayQueue(&q);

    dequeue(&q);
    displayQueue(&q);

    dequeue(&q);
    displayQueue(&q);

    return 0;
}
```

III. ADDITIONAL EXERCISES:

1) Design a data representation sequentially mapping n queues into a single array A(1:m). Represent each queue as a circular queue within A. Write algorithms ADDQ, DELETEQ and QUEUE-FULL for this representation. Illustrate the working with a menu driven program.

2) Design a data representation, sequentially mapping n data objects into an array A(1:m). n1 of these data objects are stacks and the remaining n2 = n - n1 are queues. Write algorithms to add and delete elements from these data structures. Use the same SPACE_FULL algorithm for both types of data structures. This algorithm should provide space for the i-th data object if there is some space not currently being used. Note that a circular queue with space for r elements can hold only r – 1 element. Illustrate the working by writing a c program.