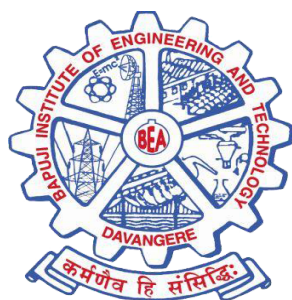


# DATA STRUCTURES LABORATORY (18CSL38)



[As per Choice Based Credit System (CBCS) scheme]

(Effective from the academic year 2020 -2021)



Bapuji Educational Association(R.)  
Bapuji Institute of Engineering and Technology (BIET)  
Davangere, Karnataka - 577004

Department of Computer Science and Engineering

Visit: <https://nvnkr.blogspot.com/p/data-structures-and-applications.html>

"BIET" <https://www.bietdvg.edu/department/UG/CS/home>

---

## INDEX

ARRAY OPERATIONS USING A SIMPLE ARRAY.	2
ARRAY OPERATIONS USING DYNAMIC MEMORY ALLOCATION.	6
STRING OPERATIONS USING NAIVE IMPLEMENTATION.	16
ARRAY IMPLEMENTATION OF STACK	18
CONVERSION FROM INFIX TO POSTFIX EXPRESSION.	21
APPLICATION OF STACK: EVALUATION OF SUFFIX EXPRESSION.	24
ARRAY IMPLEMENTATION QUEUE.	27
OPERATIONS ON SINGLY LINKED LIST (SLL).	32
OPERATIONS ON DOUBLY LINKED LIST (DLL).	39
OPERATIONS ON SINGLY CIRCULAR LINKED LIST (SCLL) WITH HEADER NODES: REPRESENT AND EVALUATE A POLYNOMIAL.	45
OPERATIONS ON SINGLY CIRCULAR LINKED LIST (SCLL) WITH HEADER NODES: SUM OF TWO POLYNOMIALS.	49
OPERATIONS ON BINARY SEARCH TREE (BST) OF INTEGERS	54
IMPLEMENTATION OF HASH TABLE.	60
OPERATION ON GRAPHS.	66

---

## ARRAY OPERATIONS USING A SIMPLE ARRAY.

### 1. Design, Develop and Implement a menu driven Program in C for the following Array operations

- a. Creating an Array of N Integer Elements
- b. Display of Array Elements with Suitable Headings
- c. Inserting an Element (ELEM) at a given valid Position (POS)
- d. Deleting an Element at a given valid Position(POS)
- e. Exit.

Support the program with functions for each of the above operations.

```
/* SIMPLE ARRAY IMPLEMENTATION */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int linearArray[100];
int noOfElements = 0;

void createArray ( void )
{
    int i;

    printf ( "\n\tEnter the size of the array: ?\b" );
    scanf ( "%d", &noOfElements );

    printf ( "\n\tEnter the array elements one by one\n\t" );
    for ( i = 1; i <= noOfElements; ++ i )
    {
        scanf ( "%d", &linearArray[i] );
        printf ( "\n\t" );
    }
}

void displayElements ( void )
{
    int i;

    if ( noOfElements == 0 )
    {
        printf ( "\n\n\tArray List is Empty\n" );
        return;
    }

    printf ( "\n\tThe array elements are..\n\n" );
    printf ( "\t(index : value)\n\n" );

    for ( i = 1; i <= noOfElements; ++ i )
    {
        printf ( "\t [%d]\t:%5d\n\n", i, linearArray[i] );
    }
}
```

```

    }
    printf ( "\n" );
}

void insertAnElement ( void )
{
    int element, position;
    int counter;

    if ( noOfElements == 0 )
    {
        printf ( "\n\n\tArray List is Empty\n" );
        return;
    }

    displayElements ( );

    printf ( "\n\tEnter the valid position: ?\b" );
    scanf ( "%d", &position );

    if ( (position <= 0) || (position > noOfElements) )
    {
        printf ( "\n\tInvalid Position\n" );
        return;
    }

    printf ( "\n\tEnter an Element to insert: ?\b" );
    scanf ( "%d", &element );

    /*Initialize counter*/
    counter = noOfElements;
    while ( counter >= position )
    {
        /*Move elements downwards*/
        linearArray[counter + 1] = linearArray[counter];
        counter = counter -1;
    }
    linearArray[position] = element;

    /*Reset the number of elements in Array*/
    noOfElements = noOfElements + 1;

    printf ( "\n\tItem Inserted successfully\n" );
}

void deleteAnElement ( void )
{

```

```

int element, position;
int i;

if ( noOfElements == 0 )
{
    printf ( "\n\n\tArray List is Empty\n" );
    return;
}

displayElements ( );

printf ( "\n\tEnter the valid position: ?\b" );
scanf ( "%d", &position );

if ( (position <= 0) || (position > noOfElements) )
{
    printf ( "\n\tInvalid Position\n" );
    return;
}

printf ( "\n\tDeleted Element: %d\n\n", linearArray[position] );

for ( i = position; i < noOfElements; ++ i )
{
    /*Move elements upward*/
    linearArray[i] = linearArray[i + 1];
}

/*Reset the number of elements in Array*/
noOfElements = noOfElements - 1;
}

int main ( void )
{
    int ch;

    do{
        //clrscr();
        system("cls");
        printf ( "\n\t\t\t ARRAY DEMONSTRATION\n\n" );
        printf ( "\n\t\t1. Create Array\t\t2. Display Array\n" );
        printf ( "\n\t\t3. Insert Element\t4. Delete Element\n" );
        printf ( "\n\t\t\t\t\t5. Exit\n\n" );

        printf ( "\n\tEnter your choice: ?\b" );
        scanf ( "%d", &ch );
    }
}

```

---

```
switch ( ch )
{
    case 1: createArray ( );
            break;
    case 2: displayElements ( );
            break;
    case 3: insertAnElement ( );
            break;
    case 4: deleteAnElement ( );
            break;
    case 5: return 0;
    default: printf ( "\n\tInvalid Option\n\n" );
}
fflush(stdin);
getchar();
//getch();
}while ( 1 );
}
```

---

## ARRAY OPERATIONS USING DYNAMIC MEMORY ALLOCATION.

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

int *arrayPtr;
int nElements;

int * allocateMemory ( int n )
{
    int *temp;

    /*Allocate enough memory*/
    temp = ( int * ) malloc ( n * sizeof ( int ) );

    /*Check for unsuccessful allocation*/
    if ( temp == NULL )
    {
        printf ( "\n\tERROR: Malloc failed to allocate memory\n\n" );
        getchar();
        exit( EXIT_FAILURE );
    }

    printf ( "\n\n\tAllocation Successful\n" );

    return temp;
}

int * reAllocateMemory ( int *ptr, int n )
{
    int *temp;

    /*Modify previously allocated memory*/
    temp = ( int * ) realloc ( ptr, n * sizeof ( int ) );

    /*Check for unsuccessful allocation*/
    if ( temp == NULL )
    {
        printf ( "\n\tERROR: Realloc failed to allocate memory\n\n" );
        getchar();
        exit( EXIT_FAILURE );
    }

    printf ( "\n\n\tReallocation Successful\n" );

    return temp;
}

void readArrayElements ( void )
{
```

```

int i;

    arrayPtr = allocateMemory ( nElements );

printf ( "\n\n\tEnter the array elements one by one\n\n\t" );
for ( i = 1; i <= nElements; ++ i )
{
    scanf ( "%d", ( arrayPtr + i ) );
    printf ( "\n\t" );
}

printf ( "\n\n\tArray elements read successfully\n\n" );
}

void displayElements ( void )
{
    int i;

    if ( nElements == 0 )
    {
        printf ( "\n\n\tArray List is Empty\n" );
        return;
    }

    printf ( "\n\tThe array elements are..\n\n" );
    printf ( "\t(index : value)\n\n" );

    for ( i = 1; i <= nElements; ++ i )
    {
        printf ( "\t [%d]\t:%5d\n\n", i, arrayPtr[i] );
    }
    printf ( "\n" );
}

void insertAnElement ( int element, int position )
{
    int counter;

    /*modify the previously allocated memory blocks*/
    arrayPtr = reAllocateMemory ( arrayPtr, ( nElements + 1 ) );

    /*Initialize counter*/
    counter = nElements;
    while ( counter >= position )
    {
        /*Move elements downwards*/
        arrayPtr[counter + 1] = arrayPtr[counter];
        counter = counter - 1;
    }
}

```



```

        arrayPtr[position] = element;

        /*Reset the number of elements in Array*/
        nElements = nElements + 1;

        printf ( "\n\tItem Inserted successfully\n" );
    }

void deleteAnElement ( int position )
{
    int i;

    printf ( "\n\tDeleted Element: %d\n\n", arrayPtr[position] );

    /* if only one element, reset array size and free memory */
    if ( nElements == 1 )
    {
        nElements = 0;
        free ( arrayPtr );
        return;
    }

    for ( i = position; i < nElements; ++ i )
    {
        /*Move elements upward*/
        arrayPtr[i] = arrayPtr[i + 1];
    }

    /*modify the previously allocated memory blocks*/
    arrayPtr = reAllocateMemory ( arrayPtr, ( nElements - 1 ) );

    /*Reset the number of elements in Array*/
    nElements = nElements - 1;
}

int main ( void )
{
    int element, position;
    int ch;

    do{
        //clrscr();
        //system("clear");
        printf ("\n\t\t\t\t\t ARRAY DEMONSTRATION\n\n" );
        printf ("\n\t\t\t1. Create Array\t\t2. Display Array\n" );
        printf ("\n\t\t\t3. Insert Element\t4. Delete Element\n" );
        printf ("\n\t\t\t\t\t5. Exit\n\n" );

        printf ( "\n\tEnter your choice: ?\b" );
    }
}

```

```

scanf ( "%d", &ch );

switch ( ch )
{
    case 1: if ( nElements == 0 )
            {
                printf ( "\n\n\tEnter the array size: ?\b" );
                scanf ( "%d", &nElements );

                free(arrayPtr);

                readArrayElements ( );
            }
        else
            {
                printf ( "\n\n\tArray List already exist\n\n" );
            }
        break;
    case 2: displayElements ( );
            break;
    case 3: if ( nElements == 0 )
            {
                printf ( "\n\n\tInsufficient Memory\n" );
                printf ( "\n\tTry option 1 to Allocate Memory\n" );
            }
        else
            {
                displayElements ( );

                printf ( "\n\t Enter the valid position: ?\b" );
                scanf ( "%d", &position );

                if ( (position < 1) || (position > nElements) )
                {
                    printf ( "\n\t Invalid Position\n" );
                }
                else
                {
                    printf ( "\n\tEnter an Element to insert: ?\b"

);

                    scanf ( "%d", &element );
                    insertAnElement ( element, position );
                }
            }
        break;
    case 4: if ( nElements == 0 )
            {
                printf ( "\n\n\tInsufficient Memory\n" );

```

---

```
        printf ( "\n\tTry option 1 to Allocate Memory\n" );
    }
    else
    {
        displayElements ( );

        printf ( "\n\tEnter the valid position: ?\b" );
        scanf ( "%d", &position );

        if ( (position < 1) || (position > nElements) )
        {
            printf ( "\n\tInvalid Position\n" );
        }
        else
        {
            deleteAnElement ( position );
        }
    }
    break;
case 5: return 0;
default: printf ( "\n\tInvalid Option\n\n" );
}
fflush(stdin);
getchar();
//getch();
}while ( 1 );
}
```

**OUTPUT: RUN - 1**

ARRAY DEMONSTRATION

- 1. Create Array                      2. Display Array
- 3. Insert Element 4. Delete Element
- 5. Exit

Enter your choice: 1

Enter the array size: 5

Allocation Successful

Enter the array elements one by one

10

20

30

40

50

Array elements read successfully

ARRAY DEMONSTRATION

- 1. Create Array                      2. Display Array
- 3. Insert Element 4. Delete Element
- 5. Exit

Enter your choice: 2

The array elements are..

(index : value)

[1] : 10

[2] : 20

[3] : 30

[4] : 40

[5] : 50

ARRAY DEMONSTRATION

- 1. Create Array                      2. Display Array
- 3. Insert Element 4. Delete Element
- 5. Exit

Enter your choice: 3

The array elements are..

(index : value)

[1] : 10

[2] : 20

[3] : 30

[4] : 40

[5] : 50

Enter the valid position: 3

Enter an Element to insert: 300

Reallocation Successful

Item Inserted successfully

ARRAY DEMONSTRATION

- 1. Create Array                      2. Display Array
- 3. Insert Element 4. Delete Element
- 5. Exit

Enter your choice: 2

The array elements are..

(index : value)

[1] : 10

[2] : 20

[3] : 300

[4] : 30

[5] : 40

[6] : 50

ARRAY DEMONSTRATION

1. Create Array                      2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 4

The array elements are..

(index : value)

[1] : 10

[2] : 20

[3] : 300

[4] : 30

[5] : 40

[6] : 50

Enter the valid position: 6

Deleted Element: 50

Reallocation Successful

ARRAY DEMONSTRATION

1. Create Array                      2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 2

The array elements are..

(index : value)

[1] : 10

[2] : 20

[3] : 300

[4] : 30

[5] : 40

ARRAY DEMONSTRATION

1. Create Array                      2. Display Array

3. Insert Element                      4. Delete Element

5. Exit

**OUTPUT: RUN - 2**

ARRAY DEMONSTRATION

1. Create Array                      2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 1

Enter the array size: 1

Allocation Successful

Enter the array elements one by one

10

Array elements read successfully

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 2

The array elements are..

(index : value)

[1] : 10

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 3

The array elements are..

(index : value)

[1] : 10

Enter the valid position: 2

Invalid Position

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 3

The array elements are..

(index : value)

[1] : 10

Enter the valid position: 0

Invalid Position

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 3

The array elements are..

(index : value)

[1] : 10

Enter the valid position: 1

Enter an Element to insert: 100

Reallocation Successful

Item Inserted successfully

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 2

The array elements are..

(index : value)

[1] : 100

[2] : 10

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 4

The array elements are..

(index : value)

[1] : 100

[2] : 10

Enter the valid position: 3

Invalid Position

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 4

The array elements are..

(index : value)

[1] : 100

[2] : 10

Enter the valid position: 2

Deleted Element: 10

Reallocation Successful

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 2

The array elements are..

(index : value)

[1] : 100

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 4

The array elements are..

(index : value)

[1] : 100

Enter the valid position: 1

Deleted Element: 100

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

---

5. Exit

Enter your choice: 2

Array List is Empty

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 3

Insufficient Memory

Try option 1 to Allocate Memory

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 4

Insufficient Memory

Try option 1 to Allocate Memory

ARRAY DEMONSTRATION

1. Create Array                    2. Display Array

3. Insert Element 4. Delete Element

5. Exit

Enter your choice: 5



---

## STRING OPERATIONS USING NAIVE IMPLEMENTATION.

2. Design, Develop and Implement a Program in C for the following operations on Strings
    - a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)
    - b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR
- Support the program with functions for each of the above operations. Don't use Built-in functions.

```
#include <stdio.h>
int flag;
int string_length( char *s )
{
    char *p = s;
    while (*p != '\0')
        p++;
    return p - s;
}

void string_copy( char *s, char *t )
{
    while ( *s++ = *t++ );
}

void string_n_copy( char *s, char *t, int n )
{
    int i;

    for ( i = 1; i <= n; ++ i )
        *s++ = *t++;
    *s = '\0';
}

void string_cat( char *s, char *t )
{
    s += string_length(s);

    string_copy ( s, t);
}

int brute_force_string_match ( char t[], char p[] )
{
    int i, j;
    int n = string_length ( t );
    int m = string_length ( p );

    for ( i = 0; i <= ( n - m ); ++ i )
    {
        j = 0;
        while ( ( j < m ) && ( p[j] == t[i + j] ) )
```

```

        {
            j = j + 1;
        }
        if ( j == m ) return i;
    }
    return -1;
}

int main()
{
    char text[50], pat[50], rep[50];
    char buf[50];
    int i;

    printf("\n\nEnter text:\n");
    gets(text);

    printf("\n\nEnter Pattern:\n");
    gets(pat);

    printf("\n\nEnter Replacement String:\n");
    gets(rep);

    while ( 1 )
    {
        i = brute_force_string_match ( text, pat );
        if ( i == -1 ) break;
        string_n_copy ( buf, text, i );
        string_cat ( buf, rep );
        string_cat ( buf, text + i + string_length ( pat ) );
        string_copy ( text, buf );
        printf ( "\n\nAfter String Replacement:" );
        printf ( "\n\n%s\n\n", text );
        flag = 1;
    }

    if ( flag == 0 )
        printf ( "\nPattern does not exist\n" );
    return 0;
}

```

---

## ARRAY IMPLEMENTATION OF STACK

3. Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

- a. Push an Element on to Stack
- b. Pop an Element from Stack
- c. Demonstrate how Stack can be used to check Palindrome
- d. Demonstrate Overflow and Underflow situations on Stack
- e. Display the status of Stack
- f. Exit

Support the program with appropriate functions for each of the above operations

```
#include <stdio.h>
int stk[100];
int stack_size;
int top = -1;

int is_full( void )
{
    return ( top == stack_size - 1 );
}

int is_empty( void )
{
    return ( top == -1 );
}

void push( int item )
{
    stk[ ++top ] = item;
}

int pop( void )
{
    return stk[top--];
}

void display( void )
{
    int i;
    printf ( "\n\nThe STACK elements are...\n\n" );
    for ( i = top; i >= 0; --i )
        printf ( "%d\n\n", stk[i] );
}

int is_palindrome( void )
{
    int i, j;
    for ( i = 0, j = top; i < j; ++ i, -- j )
        if ( stk[i] != stk[j] )
```

```

        return 0;
    return 1;
}

int main( void )
{
    int item, ch;

    printf ( "\nEnter the size of STACK: ?\b" );
    scanf ( "%d", &stack_size );

    do
    {
        printf ( "\n\t1. Push \t 2. Pop \t 3. Display\n" );
        printf ( "\n\t4. Palindrome Check \t 5. Exit \n" );
        printf ( "\nEnter your choice: ?\b" );
        scanf ( "%d", &ch );

        switch ( ch )
        {
            case 1: if ( is_full() )
                    {
                        printf ( "\nSTACK is Overflow\n\n" );
                    }
                    else
                    {
                        printf ( "\nEnter an Item to insert: ?\b" );
                        scanf ( "%d", &item );
                        push ( item );
                    }
                    Break;

            case 2: if ( is_empty() )
                    {
                        printf ( "\nSTACK is Underflow\n" );
                    }
                    else
                    {
                        printf ( "\nPopped: %d\n", pop() );
                    }
                    break;

            case 3: if ( is_empty() )
                    {
                        printf ( "\nSTACK is Empty\n" );
                    }
                    else
                    {
                        display();
                    }
        }
    }
}

```

---

```
                break;

        case 4: if ( is_empty() )
                {
                        printf ( "\nSTACK is Empty\n" );
                }
                else
                {
                        if ( is_palindrome() )
                                printf ( "\nIt is Palindrome\n" );
                        else
                                printf ( "\nIt is not a Palindrome\n"
);
                }
                break;

        case 5: return 0;
        default: printf ( "\nInvalid Option\n" );
    }
    getch();
}while(1);

return 0;
}
```

---

## CONVERSION FROM INFIX TO POSTFIX EXPRESSION.

4. Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, \*, /, %(Remainder), ^(Power) and alphanumeric operands.

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

typedef enum {lparen, rparen, plus, minus, times, divide, mod, pow, eos, operand}
PRECEDENCE;
PRECEDENCE stack[100];
int top = 0;
char infix[100];

/* isp and icp arrays -- index is value of precedence
   lparen, rparen, plus, minus, times, divide, mod, pow, eos */
int isp[] = {0, 19, 12, 12, 13, 13, 13, 14, 0};
int icp[] = {20, 19, 12, 12, 13, 13, 13, 14, 0};

void getInfix ( void )
{
    printf ( "\n\nEnter the valid Infix expression\n\n" );
    gets ( infix );
}

void push ( PRECEDENCE symbol )
{
    /* add an item to the global stack */
    stack[++ top] = symbol;
}

PRECEDENCE pop ( void )
{
    /* delete and return the top element from the stack */
    return stack[top --];
}

PRECEDENCE getToken ( char *symbol, int *n )
{
    *symbol = infix[(*n)++];
    switch ( *symbol )
    {
        case '(' : return lparen;
        case ')' : return rparen;
        case '+' : return plus;
        case '-' : return minus;
        case '/' : return divide;
        case '*' : return times;
    }
}
```

```

        case '%' : return mod;
        case '^' : return pow;
        case '\0' : return eos;
        case ' ' : return eos;
    /* no error checking, default is operand */
        default : return operand;
    }
}

void printToken ( PRECEDENCE symbol )
{
    switch ( symbol )
    {
        case plus :      putchar ( '+' ); break;
        case minus :    putchar ( '-' ); break;
        case divide :   putchar ( '/' ); break;
        case times :    putchar ( '*' ); break;
        case mod :      putchar ( '%' ); break;
        case pow :      putchar ( '^' );
    }
}

void postfix ( void )
{
    char symbol;
    PRECEDENCE token;
    int n = 0;
    stack[0] = eos;
    printf ( "\n\n Equivalent Postfix expression\n\n" );
    token = getToken ( &symbol, &n );
    while ( token != eos )
    {
        if ( token == operand )
        {
            putchar ( symbol );
        }
        else if ( token == rparen )
        {
            /* unstack tokens until left parenthesis */
            while ( stack[top] != lparen )
            {
                printToken ( pop ( ) );
            }
            pop ( ); /* discard the left parenthesis */
        }
        else
        {
            /* remove and print symbols whose isp is greater
            than or equal to the current token's icp */

```

```

                while ( isp[stack[top]] >= icp[token] )
                {
                    printToken ( pop ( ) );
                }
                push ( token );
            }
            token = getToken ( &symbol, &n );
        } //while ends
        while ( ( token = pop ( ) ) != eos )
        {
            printToken ( token );
        }
        putchar ( '\n' );
    }

int main ( void )
{
    printf ( "\n\nINFIX TO POSTFIX Demonstration\n\n" );
    getInfix ( );
    postfix ( );
    fflush ( stdin );
    getchar();
    return 0;
}

/*
OUTPUT:-
    INFIX TO POSTFIX Demonstration

    Enter the valid Infix expression
    ((a*b)+(c/d))
    Equivalent Postfix expression
    ab*cd/+

    INFIX TO POSTFIX Demonstration

    Enter the valid Infix expression
    (A+B)*D+E/(F+A*D)+C
    Equivalent Postfix expression
    AB+D*EFAD*+/+C+
*/

```



---

## APPLICATION OF STACK: EVALUATION OF SUFFIX EXPRESSION.

### 5. Design, Develop and Implement a Program in C for the following Stack Applications

#### a. Evaluation of Suffix expression with single digit operands and operators:

+, -, \*, /, %, ^

```
/* Preprocessor directives */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*--Global variables--*/
int stack[50];
int top = 0;
char postfix[20];

typedef enum {plus, minus, times, divide, mod, power, eos, operand} PRECEDENCE;

/*-- Function definition--*/
void getPostfix ( void )
{
    printf ( "\nEnter the valid POSTFIX expression: ?\b" );
    gets ( postfix );
}

void push ( int symbol )
{
    /* add an item to the global stack */
    stack[++ top] = symbol;
}

int pop ( void )
{
    /* delete and return the top element from the stack */
    return stack[top --];
}

PRECEDENCE getToken ( char *symbol, int *n )
{
    *symbol = postfix[(*n)++];
    switch ( *symbol )
    {
        case '(' : return lparen;
        case ')' : return rparen;
        case '+' : return plus;
        case '-' : return minus;
        case '/' : return divide;
        case '*' : return times;
        case '%' : return mod;
        case '^' : return power;
        case '\0' : return eos;
        case ' ' : return eos;
    }
}
```

```

        default      : return operand; /* no error checking, default is
operand */
    }
}

int eval ( void )
{
    PRECEDENCE token;
    char symbol;
    int op1, op2;
    int n = 0;
    token = getToken ( &symbol, &n );
    while ( token != eos )
    {
        if ( token == operand )
        {
            push ( symbol - '0' );
        }
        else
        {
            op2 = pop ( );
            op1 = pop ( );
            switch ( token )
            {
                case plus      :   push ( op1 + op2 );      break;
                case minus     :   push ( op1 - op2 );      break;
                case times     :   push ( op1 * op2 );      break;
                case divide    :   push ( op1 / op2 );      break;
                case mod       :   push ( op1 % op2 );      break;
                case power     :   push ( (int)pow( (double)op1,
(double)op2 ) );
            }

            token = getToken ( &symbol, &n );
        }
        return pop ( );
    }
}

void printResult ( void )
{
    printf ( "\n\nEvaluated POSTFIX expression: %d\n\n", eval ( ) );
}

int main ( void )
{
    system ( "cls" );

    getPostfix ( );

    printResult ( );
}

```

---

```
    fflush ( stdin );  
    getchar ( );  
    return 0;  
}
```

```
/*
```

**OUTPUT:-**

Enter the valid POSTFIX expression: 6 2 / 3 - 4 2 \* +

Evaluated POSTFIX expression: 8

```
*/
```

---

### ARRAY IMPLEMENTATION QUEUE.

6. Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)

- a. Insert an Element on to Circular QUEUE
- b. Delete an Element from Circular QUEUE
- c. Demonstrate Overflow and Underflow situations on Circular QUEUE
- d. Display the status of Circular QUEUE
- e. Exit

Support the program with appropriate functions for each of the above operations

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

int queue[100];
int queueSize;
int front = 0, rear = 0;

void queueFull ( void )
{
    printf ( "\n\n Insertion Failed: Queue is Full\n\n" );

    if ( front ) rear = front - 1;
    else rear = queueSize - 1;
}

int queueEmpty ( void )
{
    printf ( "\n\nQueue is Empty\n\n" );

    front = rear = 0;
}

void addq ( int item )
{
    rear = ( rear + 1 ) % queueSize;

    if ( front == rear )
    {
        queueFull ( );
        return;
    }
    queue[rear] = item;

    printf ( "\n\n Item is inserted Successfully\n\n" );
}

void deleteq ( void )
{
    int item;
    if ( front == rear )
```

```

        {
            queueEmpty ( );
            return;
        }
        front = ( front + 1 ) % queueSize;
        printf ( "\n\n Deleted Item: %d\n\n", queue[front] );
    }
}

void displayQueue ( void )
{
    int i;

    if ( front == rear )
    {
        queueEmpty ( );
        return;
    }

    printf ( "\n\n Circular Queue Elements are..\n\n" );
    i = ( front + 1 ) % queueSize;
    while ( i != rear )
    {
        printf ( "\t%d\n\n", queue[i] );
        i = ( i + 1 ) % queueSize;
    }
    printf ( "\t%d\n\n", queue[rear] );

    printf ( "\n\n" );
}

int main ( void )
{
    int item;
    int ch;

    printf ( "\n\nEnter the size of Circular Queue Memory: ?\b" );
    scanf ( "%d", &queueSize );
    queueSize = queueSize + 1;
    do{
        system("cls");
        printf ( "\n\n\t\t CIRCULAR QUEUE DEMONSTRATION\n\n" );
        printf ( "\n\n\t\t 1. Insert\t\t2. Delete\n\n" );
        printf ( "\n\n\t\t 3. Display\t\t4. Exit\n\n" );

        printf ( "\n\n Enter your choice: ?\b" );
        scanf ( "%d", &ch );

        switch ( ch )
        {
            case 1: printf ( "\n\nEnter an Element to insert: ?\b" );
                    scanf ( "%d", &item );
                    addq ( item );

```

```

                break;
            case 2: deleteq ( );
                break;
            case 3: displayQueue ( );
                break;
            case 4: return 0;
            default: printf ( "\n\nInvalid Option\n\n" );
        }
        fflush(stdin);
        getchar();
    }while ( 1 );
}

```

/\*

OUTPUT:-

Enter the size of Circular Queue Memory: 5

CIRCULAR QUEUE DEMONSTRATION

- |            |           |
|------------|-----------|
| 1. Insert  | 2. Delete |
| 3. Display | 4. Exit   |

Enter your choice: 2

Queue is Empty

CIRCULAR QUEUE DEMONSTRATION

- |            |           |
|------------|-----------|
| 1. Insert  | 2. Delete |
| 3. Display | 4. Exit   |

Enter your choice: 3

Queue is Empty

CIRCULAR QUEUE DEMONSTRATION

- |            |           |
|------------|-----------|
| 1. Insert  | 2. Delete |
| 3. Display | 4. Exit   |

Enter your choice: 1

Enter an Element to insert: 10

Item is inserted Successfully

Enter your choice: 1

---

Enter an Element to insert: 20  
Item is inserted Successfully

Enter your choice: 1

Enter an Element to insert: 30  
Item is inserted Successfully

Enter your choice: 1

Enter an Element to insert: 40  
Item is inserted Successfully

Enter your choice: 1

Enter an Element to insert: 50  
Item is inserted Successfully

Enter your choice: 1

Enter an Element to insert: 60  
Insertion Failed: Queue is Full

CIRCULAR QUEUE DEMONSTRATION

- |            |           |
|------------|-----------|
| 1. Insert  | 2. Delete |
| 3. Display | 4. Exit   |

Enter your choice: 3

Circular Queue Elements are..

10

20

30

40

50

CIRCULAR QUEUE DEMONSTRATION

- |            |           |
|------------|-----------|
| 1. Insert  | 2. Delete |
| 3. Display | 4. Exit   |

Enter your choice: 2  
Deleted Item: 10

---

Enter your choice: 2  
Deleted Item: 20

CIRCULAR QUEUE DEMONSTRATION

- |            |           |
|------------|-----------|
| 1. Insert  | 2. Delete |
| 3. Display | 4. Exit   |

Enter your choice: 3

Circular Queue Elements are..

30

40

50

CIRCULAR QUEUE DEMONSTRATION

- |            |           |
|------------|-----------|
| 1. Insert  | 2. Delete |
| 3. Display | 4. Exit   |

Enter your choice: 2  
Deleted Item: 30

Enter your choice: 2  
Deleted Item: 40

Enter your choice: 2  
Deleted Item: 50

Enter your choice: 2  
Queue is Empty

CIRCULAR QUEUE DEMONSTRATION

- |            |           |
|------------|-----------|
| 1. Insert  | 2. Delete |
| 3. Display | 4. Exit   |

Enter your choice: 3  
Queue is Empty



---

## OPERATIONS ON SINGLY LINKED LIST (SLL).

7. Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Branch, Sem, PhNo

- a. Create a SLL of N Students Data by using front insertion.
- b. Display the status of SLL and count the number of nodes in it
- c. Perform Insertion / Deletion at End of SLL
- d. Perform Insertion / Deletion at Front of SLL (Demonstration of stack)
- e. Exit

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

typedef struct StudentRecord
{
    char USN[20];
    char name[20];
    char dept[20];
    char phone[20];
}Record;

typedef struct Node *ListPointer;
typedef struct Node
{
    Record data;
    ListPointer link;
}Node;

ListPointer head;
int tRecords;

Record getNextRecord ( void )
{
    Record data;

    scanf ( "%s", data.USN );
    scanf ( "%s", data.name );
    scanf ( "%s", data.dept );
    scanf ( "%s", data.phone );

    return data;
}

ListPointer getNode ( Record data )
{
    ListPointer temp;

    temp = ( ListPointer ) malloc ( sizeof ( *temp ) );

    if ( temp == NULL )
    {
        printf ( "\nAllocation Failed\n" );
    }
}
```

```

        getchar (); //getch();
        exit(EXIT_FAILURE);
    }else
    {
        temp->data = data;
        temp->link = NULL;
    }

    return temp;
}

void insert(ListPointer targetNode, ListPointer newNode)
{
    newNode->link = targetNode->link;
    targetNode->link = newNode;

    tRecords = tRecords + 1;
}

void dDelete (ListPointer head, ListPointer trail, ListPointer x)
{
    if ( trail )
    {
        trail->link = x->link;
    }else
    {
        head->link = x->link;
    }

    free(x);

    tRecords = tRecords - 1;
}

void getLastAndPrevNode(ListPointer head, ListPointer *prev, ListPointer *last)
{
    ListPointer cur, trail;

    cur = trail = head;

    if ( head->link != NULL )
    {
        cur = head->link;
        trail = NULL;
        while ( cur->link != NULL )
        {
            trail = cur;
            cur = cur->link;
        }
    }

    *prev = trail;
}

```

```

        *last = cur;
    }

void addFront ( ListPointer head, Record data )
{
    ListPointer temp;
    temp = getNode ( data );

    insert(head, temp);
}

void addRear ( ListPointer head, Record data )
{
    ListPointer prev, last, temp;
    temp = getNode ( data );

    getLastAndPrevNode(head, &prev, &last);

    insert(last, temp);
}

void display ( ListPointer node, int tRecords )
{
    int i;

    printf("\nUSN\tNAME\tDEPT.\tPHONE\n");

    for ( i = 0; i < tRecords; ++ i )
    {
        printf ( "\n%s", node->data.USN );
        printf ( "\t%s", node->data.name );
        printf ( "\t%s", node->data.dept );
        printf ( "\t%s\n", node->data.phone );

        node = node->link;
    }
}

void deleteFront ( ListPointer head )
{
    ListPointer node;
    node = head->link;

    printf ( "\nDeleted Record: %s\n", node->data.USN );
    display ( node, 1 );

    dDelete(head, NULL, node);
}

void deleteRear ( ListPointer head )
{
    ListPointer prev, last;

```

```

        getLastAndPrevNode(head, &prev, &last);

        printf ( "\nDeleted Record: %s\n", last->data.USN );
        display ( last, 1 );

        dDelete(head, prev, last);
    }

int main ( void )
{
    Record data;
    int nRecords;
    int choice;
    int i;

    printf("\n.. SINGLY LINKED LIST DEMONSTRATION ..\n");
    printf ( "\n\n1. N front Insertion\n\n2. Display\n");
    printf ( "\n3. Front Insertion\n\n4. Delete Front\n");
    printf ( "\n5. Rear Insertion\n\n6. Delete Rear\n");
    printf ( "\n7. Exit\n");

    head = getNode(data);

    while ( 1 )
    {
        printf ( "\nChoice: " );
        scanf ( "%d", &choice );

        switch ( choice )
        {
            case 1: printf("\n.. * FRONT INSERTION ..\n");
                    printf ( "\nHow many STD's: " );
                    scanf ( "%d", &nRecords );

                    printf("\nGive %d record details one by one\n",
nRecords);

                    printf("\nUSN\tNAME\tDEPT.\tPHONE\n");
                    for ( i = 0; i < nRecords; ++ i )
                    {
                        data = getNextRecord();
                        addFront ( head, data );
                    }
                    break;
            case 2: if ( nRecords == 0 )
                    {
                        printf ( "\nEmpty list\n" );
                    }
                    else
                    {
                        printf("\n Student's details...\n");
                    }
        }
    }
}

```

```

        display ( head->link, tRecords );
        printf ( "\nNo. of STD's: %d\n", tRecords );
    }
    break;
case 3: printf("\n.. SINGLE FRONT INSERTION ..\n");
        printf("\nGive record details..\n");
        printf("\nUSN\tNAME\tDEPT.\tPHONE\n");
        data = getNextRecord();
        addFront ( head, data );
        break;
case 4: if ( tRecords == 0 )
        {
            printf ( "\nEmpty list\n" );
        }
        else
        {
            printf("\n.. SINGLE FRONT DELETION ..\n");
            deleteFront ( head );
        }
        break;
case 5: printf("\n.. SINGLE BACK INSERTION ..\n");
        printf("\nGive record details..\n");
        printf("\nUSN\tNAME\tDEPT.\tPHONE\n");
        data = getNextRecord();
        addRear ( head, data );
        break;
case 6: if ( tRecords == 0 )
        {
            printf ( "\nEmpty list\n" );
        }
        else
        {
            printf("\n.. SINGLE BACK DELETION ..\n");
            deleteRear ( head );
        }
        break;
case 7: return 0;

default: printf ( "\nWrong Choice\n" );
    }
}
}

```

/\*

---

**OUTPUT:**

.. SINGLY LINKED LIST DEMONSTRATION ..

1. N front Insertion
2. Display
3. Front Insertion
4. Delete Front
5. Rear Insertion
6. Delete Rear
7. Exit

Choice: 1

.. \* FRONT INSERTION ..

How many STD's: 3

Give 3 record details one by one

USN	NAME	DEPT.	PHONE
111	John	CSE	11111
222	Jack	CSE	22222
333	Kate	ISE	33333

Choice: 2

Student's details...

USN	NAME	DEPT.	PHONE
-----	------	-------	-------

333	Kate	ISE	33333
-----	------	-----	-------

222	Jack	CSE	22222
-----	------	-----	-------

111	John	CSE	11111
-----	------	-----	-------

No. of STD's: 3

Choice: 3

.. SINGLE FRONT INSERTION ..

Give record details..

USN	NAME	DEPT.	PHONE
-----	------	-------	-------

444	Daniel	ECE	44444
-----	--------	-----	-------

Choice: 2

Student's details...

USN	NAME	DEPT.	PHONE
-----	------	-------	-------

444	Daniel	ECE	44444
-----	--------	-----	-------

333	Kate	ISE	33333
-----	------	-----	-------

222	Jack	CSE	22222
-----	------	-----	-------

111	John	CSE	11111
-----	------	-----	-------

No. of STD's: 4

Choice: 4

.. SINGLE FRONT DELETION ..

Deleted Record: 444

USN	NAME	DEPT.	PHONE
-----	------	-------	-------

444	Daniel	ECE	44444
-----	--------	-----	-------

Choice: 2

Student's details...

USN	NAME	DEPT.	PHONE
-----	------	-------	-------

333	Kate	ISE	33333
-----	------	-----	-------

222	Jack	CSE	22222
-----	------	-----	-------

111	John	CSE	11111
-----	------	-----	-------

No. of STD's: 3

Choice: 5

.. SINGLE BACK INSERTION ..

Give record details..

USN	NAME	DEPT.	PHONE
-----	------	-------	-------

555	Kathy	ECE	55555
-----	-------	-----	-------

Choice: 2

---

Student's details...

USN NAME DEPT. PHONE

333 Kate ISE 33333

222 Jack CSE 22222

111 John CSE 11111

555 Kathy ECE 55555

No. of STD's: 4

Choice: 6

.. SINGLE BACK DELETION ..

Deleted Record: 555

USN NAME DEPT. PHONE

555 Kathy ECE 55555

Choice: 2

Student's details...

USN NAME DEPT. PHONE

333 Kate ISE 33333

222 Jack CSE 22222

111 John CSE 11111

No. of STD's: 3

Choice: 4

.. SINGLE FRONT DELETION ..

Deleted Record: 333

USN NAME DEPT. PHONE

333 Kate ISE 33333

Choice: 6

.. SINGLE BACK DELETION ..

Deleted Record: 111

USN NAME DEPT. PHONE

111 John CSE 11111

Choice: 2

Student's details...

USN NAME DEPT. PHONE

222 Jack CSE 22222

No. of STD's: 1

Choice: 4

.. SINGLE FRONT DELETION ..

Deleted Record: 222

USN NAME DEPT. PHONE

222 Jack CSE 22222

Choice: 2

Empty list

Choice: 6

Empty list

Choice: 7 Exit

\*/

---

## OPERATIONS ON DOUBLY LINKED LIST (DLL).

8. Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo

- a. Create a DLL of N Employees Data by using end insertion.
- b. Display the status of DLL and count the number of nodes in it
- c. Perform Insertion and Deletion at End of DLL
- d. Perform Insertion and Deletion at Front of DLL
- e. Demonstrate how this DLL can be used as Double Ended Queue
- f. Exit

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

typedef struct EmployeeRecord
{
    char SSN[20];
    char name[20];
    char dept[20];
    char salary[20];
}Record;

typedef struct Node *ListPointer;
typedef struct Node
{
    ListPointer lLink;
    Record data;
    ListPointer rLink;
}Node;

ListPointer head;
int tRecords;

Record getNextRecord ( void )
{
    Record data;

    scanf ( "%s", data.SSN );
    scanf ( "%s", data.name );
    scanf ( "%s", data.dept );
    scanf ( "%s", data.salary );

    return data;
}

ListPointer getNode ( Record data )
{
    ListPointer temp;

    temp = ( ListPointer ) malloc ( sizeof ( *temp ) );

    if ( temp == NULL )
    {
        printf ( "\n\nAllocation Failed\n\n" );
    }
}
```



```

        getchar (); //getch();
        exit ( EXIT_FAILURE );
    }
    else
    {
        temp->data = data;
        temp->lLink = temp->rLink = temp;
    }

    return temp;
}

/** newNode will be added to right of the targetNode */
void insert(ListPointer targetNode, ListPointer newNode)
{
    newNode->lLink = targetNode;
    newNode->rLink = targetNode->rLink;
    targetNode->rLink->lLink = newNode;
    targetNode->rLink = newNode;

    tRecords = tRecords + 1;
}

void dDelete ( ListPointer deleted )
{
    if ( deleted == deleted->rLink )
    {
        printf ( "\n\nNo EMPLOYEE records exist\n\n" );
        return;
    }

    deleted->lLink->rLink = deleted->rLink;
    deleted->rLink->lLink = deleted->lLink;

    free ( deleted );

    tRecords = tRecords - 1;
}

void addFront ( ListPointer head, Record data )
{
    ListPointer temp;
    temp = getNode ( data );

    insert(head, temp);
}

void addRear ( ListPointer head, Record data )
{
    ListPointer temp;
    temp = getNode ( data );

    insert(head->lLink, temp);
}

void display ( ListPointer node, int tRecords )
{
    int i;

```

```

printf("\nSSN\tNAME\tDEPT.\tSALARY\n");

for ( i = 0; i < tRecords; ++ i )
{
    printf ( "\n%s", node->data.SSN );
    printf ( "\t%s", node->data.name );
    printf ( "\t%s", node->data.dept );
    printf ( "\t%s\n", node->data.salary );

    node = node->rLink;
}
}

void deleteFront ( ListPointer head )
{
    ListPointer node;
    node = head->rLink;

    printf ( "\nDeleted Record: %s\n", node->data.SSN );
    display ( node, 1 );

    dDelete(node);
}

void deleteRear ( ListPointer head )
{
    ListPointer node;
    node = head->lLink;

    printf ( "\nDeleted Record: %s\n", node->data.SSN );
    display ( node, 1 );

    dDelete(node);
}

int main ( void )
{
    Record data;
    int nRecords;
    int choice;
    int i;

    printf("\n.. DOUBLY LINKED LIST DEMONSTRATION ..\n");
    printf ( "\n\n1. N Back Insertion\n\n2. Display\n");
    printf ( "\n\n3. Front Insertion\n\n4. Delete Front\n");
    printf ( "\n\n5. Rear Insertion\n\n6. Delete Rear\n");
    printf ( "\n\n7. Exit\n");

    head = getNode(data);

    while ( 1 )
    {
        printf ( "\nChoice: " );
        scanf ( "%d", &choice );

        switch ( choice )
        {
            case 1: printf("\n.. * BACK INSERTION ..\n");
                    printf ( "\nHow many EMP's: " );

```

```

scanf ( "%d", &nRecords );

printf("\nGive %d record details one by one\n",
nRecords);

printf("\nSSN\tNAME\tDEPT.\tSALARY\n");
for ( i = 0; i < nRecords; ++ i )
{
    data = getNextRecord();
    addRear ( head, data );
}
break;
case 2: if ( tRecords == 0 )
{
    printf ( "\nEmpty list\n" );
}
else
{
    printf("\n Employee's details...\n");
    display ( head->rLink, tRecords );
    printf ("\nNo. of EMP's: %d\n", tRecords );
}
break;
case 3: printf("\n.. SINGLE FRONT INSERTION ..\n");
printf("\nGive record details..\n");
printf("\nSSN\tNAME\tDEPT.\tSALARY\n");
data = getNextRecord();
addFront ( head, data );
break;
case 4: if ( tRecords == 0 )
{
    printf ( "\nEmpty list\n" );
}
else
{
    printf("\n.. SINGLE FRONT DELETION ..\n");
    deleteFront ( head );
}
break;
case 5: printf("\n.. SINGLE BACK INSERTION ..\n");
printf("\nGive record details..\n");
printf("\nSSN\tNAME\tDEPT.\tSALARY\n");
data = getNextRecord();
addRear ( head, data );
break;
case 6: if ( tRecords == 0 )
{
    printf ( "\nEmpty list\n" );
}
else
{
    printf("\n.. SINGLE BACK DELETION ..\n");
    deleteRear ( head );
}
break;
case 7: return 0;

default: printf ( "\nWrong Choice\n" );
}
}

```

}

/\*

.. DOUBLY LINKED LIST DEMONSTRATION ..

1. N back Insertion
2. Display
3. Front Insertion
4. Delete Front
5. Rear Insertion
6. Delete Rear
7. Exit

Choice: 1

.. \* BACK INSERTION ..

How many EMP's: 3

Give 3 record details one by one

SSN	NAME	DEPT.	SALARY
111	John	CSE	11111
222	Jack	CSE	22222
333	Kate	ISE	33333

Choice: 2

Employee's details...

SSN	NAME	DEPT.	SALARY
111	John	CSE	11111
222	Jack	CSE	22222
333	Kate	ISE	33333

No. of EMP's: 3

Choice: 3

.. SINGLE FRONT INSERTION ..

Give record details..

SSN	NAME	DEPT.	SALARY
444	Daniel	ECE	44444

Choice: 2

Employee's details...

SSN	NAME	DEPT.	SALARY
444	Daniel	ECE	44444
111	John	CSE	11111
222	Jack	CSE	22222
333	Kate	ISE	33333

No. of EMP's: 4

Choice: 4

.. SINGLE FRONT DELETION ..

Deleted Record: 444

SSN	NAME	DEPT.	SALARY
444	Daniel	ECE	44444

Choice: 2

Employee's details...

SSN	NAME	DEPT.	SALARY
111	John	CSE	11111
222	Jack	CSE	22222
333	Kate	ISE	33333

No. of EMP's: 3

Choice: 5

.. SINGLE BACK INSERTION ..

Give record details..

SSN	NAME	DEPT.	SALARY
555	Kathy	ECE	55555

Choice: 2

Employee's details...

SSN	NAME	DEPT.	SALARY
111	John	CSE	11111
222	Jack	CSE	22222

---

333 Kate ISE 33333

555 Kathy ECE 55555

No. of EMP's: 4

Choice: 6

.. SINGLE BACK DELETION ..

Deleted Record: 555

SSN NAME DEPT. SALARY

555 Kathy ECE 55555

Choice: 2

Employee's details...

SSN NAME DEPT. SALARY

111 John CSE 11111

222 Jack CSE 22222

333 Kate ISE 33333

No. of EMP's: 3

Choice: 4

.. SINGLE FRONT DELETION ..

Deleted Record: 111

SSN NAME DEPT. SALARY

111 John CSE 11111

Choice: 6

.. SINGLE BACK DELETION ..

Deleted Record: 333

SSN NAME DEPT. SALARY

333 Kate ISE 33333

Choice: 2

Employee's details...

SSN NAME DEPT. SALARY

222 Jack CSE 22222

No. of EMP's: 1

Choice: 4

.. SINGLE FRONT DELETION ..

Deleted Record: 222

SSN NAME DEPT. SALARY

222 Jack CSE 22222

Choice: 2

Empty list

Choice: 6

Empty list

Choice: 7 Exit

\*/

---

## OPERATIONS ON SINGLY CIRCULAR LINKED LIST (SCLL) WITH HEADER NODES: REPRESENT AND EVALUATE A POLYNOMIAL.

9. Design, Develop and Implement a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes.

a. Represent and Evaluate a Polynomial.

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

typedef struct Polynomial
{
    int coef;
    int expon;
}Record;

typedef struct Node *ListPointer;
typedef struct Node
{
    Record data;
    ListPointer link;
}Node;

ListPointer head1;

Record getNextRecord ( void )
{
    Record data;

    scanf ( "%d", &data.coef );
    scanf ( "%d", &data.expon );

    return data;
}

ListPointer getNode ( Record data )
{
    ListPointer temp;

    temp = ( ListPointer ) malloc ( sizeof ( *temp ) );

    if ( temp == NULL )
    {
        printf ("\nAllocation Failed\n");
        getchar (); //getch();
        exit(EXIT_FAILURE);
    }else
    {
        temp->data = data;
        temp->link = NULL;
    }

    return temp;
}
```

```

void insert(ListPointer targetNode, ListPointer newNode)
{
    newNode->link = targetNode->link;
    targetNode->link = newNode;
}

void getLastAndPrevNode( ListPointer head, ListPointer *prev, ListPointer *last)
{
    ListPointer cur, trail;

    cur = trail = head;

    if ( head->link != NULL )
    {
        cur = head->link;
        trail = NULL;
        while ( cur->link != NULL )
        {
            trail = cur;
            cur = cur->link;
        }
    }

    *prev = trail;
    *last = cur;
}

void addRear ( ListPointer head, Record data )
{
    ListPointer prev, last, temp;
    temp = getNode ( data );

    getLastAndPrevNode(head, &prev, &last);

    insert(last, temp);
}

void display ( ListPointer head )
{
    ListPointer temp = head->link;

    printf ( " %dx^%d", temp->data.coef, temp->data.expon );
    temp = temp->link;

    while ( temp != NULL )
    {
        if ( temp->data.coef > 0 )
        {
            printf(" +");
        }
        printf ( " %dx^%d", temp->data.coef, temp->data.expon );
        temp = temp->link;
    }
}

int HornerRule ( int x )
{
    ListPointer temp;
    int p;
}

```

```

    p = head1->link->data.coef;
    temp = head1->link->link;

    while ( temp != NULL )
    {
        p = x * p + temp->data.coef;
        temp = temp->link;
    }

    return p;
}

int main ( void )
{
    Record data;
    int nRecords;
    int choice;
    int x, i;

    printf("\n.. POLYNOMIAL EVALUATION ..\n");

    head1 = getNode(data);

    printf ( "\nTerms of Polynomial 1: ?\b" );
    scanf ( "%d", &nRecords );
    printf ( "\nGive 1st record details one by one\n" );
    printf ( "\nCOEF\tEXPON\n" );
    for ( i = 0; i < nRecords; ++ i )
    {
        data = getNextRecord ();
        addRear ( head1, data );
    }

    printf ( "\nEnter the value for x: ?\b");
    scanf ( "%d", &x );

    printf ( "\n\nPoly - 1\n" );
    display ( head1 );

    printf ( "\n\nResult: %d", HornerRule ( x ) );

    printf("\n\n");

    //getch();
    return 0;
}

```



---

/\*

.. POLYNOMIAL EVALUATION ..

Terms of Polynomial 1: 5

Give 1st record details one by one

COEF	EXPON
2	4
-1	3
3	2
1	1
-5	0

Enter the value for x: 3

Poly - 1

$2x^4 - 1x^3 + 3x^2 + 1x^1 - 5x^0$

Result: 160

\*/

---

## OPERATIONS ON SINGLY CIRCULAR LINKED LIST (SCLL) WITH HEADER NODES: SUM OF TWO POLYNOMIALS.

9. Design, Develop and Implement a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes
- Find the sum of two polynomials POLY1(x) and POLY2(x) and store the result in POLYSUM(x)

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

typedef struct Polynomial
{
    int coef;
    int expon;
}Record;

typedef struct Node *ListPointer;
typedef struct Node
{
    Record data;
    ListPointer link;
}Node;

ListPointer head1;
ListPointer head2;
ListPointer head3;

Record getNextRecord ( void )
{
    Record data;

    scanf ( "%d", &data.coef );
    scanf ( "%d", &data.expon );

    return data;
}

ListPointer getNode ( Record data )
{
    ListPointer temp;

    temp = ( ListPointer ) malloc ( sizeof ( *temp ) );

    if ( temp == NULL )
    {
        printf ("\nAllocation Failed\n");
        getchar (); //getch();
        exit(EXIT_FAILURE);
    }else
    {
        temp->data = data;
        temp->link = NULL;
    }
}
```

```

    return temp;
}

void insert(ListPointer targetNode, ListPointer newNode)
{
    newNode->link = targetNode->link;
    targetNode->link = newNode;
}

void getLastAndPrevNode( ListPointer head, ListPointer *prev, ListPointer *last)
{
    ListPointer cur, trail;

    cur = trail = head;

    if ( head->link != NULL )
    {
        cur = head->link;
        trail = NULL;
        while ( cur->link != NULL )
        {
            trail = cur;
            cur = cur->link;
        }
    }

    *prev = trail;
    *last = cur;
}

void addRear ( ListPointer head, Record data )
{
    ListPointer prev, last, temp;
    temp = getNode ( data );

    getLastAndPrevNode(head, &prev, &last);

    insert(last, temp);
}

void display ( ListPointer head )
{
    ListPointer temp = head->link;

    printf ( " %dx^%d", temp->data.coef, temp->data.expon );
    temp = temp->link;

    while ( temp != NULL )
    {
        if ( temp->data.coef > 0 )
        {
            printf(" +");
        }
        printf ( " %dx^%d", temp->data.coef, temp->data.expon );
        temp = temp->link;
    }
}

```

```

int COMPARE ( int a, int b )
{
    if ( a < b )
    {
        return -1;
    }
    if ( a == b )
    {
        return 0;
    }
    return 1;
}

void addPoly ( ListPointer a, ListPointer b )
{
    Record data;
    int sum;

    a = a->link; /* skip header node for a and b */
    b = b->link;

    while ( a && b )
    {
        switch ( COMPARE ( a->data.expon, b->data.expon ) )
        {
            case -1:    addRear ( head3, b->data );
                       b = b->link;
                       break;

            case 0 :    sum = a->data.coef + b->data.coef;
                       data.coef = sum;
                       data.expon = a->data.expon;
                       if ( sum )
                       {
                           addRear ( head3, data );
                       }
                       a = a->link;
                       b = b->link;
                       break;

            case 1 :    addRear ( head3, a->data );
                       a = a->link;
        }
    }

    for ( ;a; a = a->link )
    {
        addRear ( head3, a->data );
    }
    for ( ;b; b = b->link )
    {
        addRear ( head3, b->data );
    }
}

int main ( void )
{
    Record data;

```

```

int nRecords;
int choice;
int i;

printf("\n.. ADDITION OF TWO POLYNOMIALS ..\n");

head1 = getNode(data);
head2 = getNode(data);
head3 = getNode(data);

printf ( "\nTerms of Polynomial 1: ?\b" );
scanf ( "%d", &nRecords );
printf ( "\nGive 1st record details one by one\n" );
printf ( "\nCOEF\tEXPON\n" );
for ( i = 0; i < nRecords; ++ i )
{
    data = getNextRecord ();
    addRear ( head1, data );
}

printf ( "\nTerms of Polynomial 2: ?\b" );
scanf ( "%d", &nRecords );
printf ( "\nGive 2nd record details one by one\n" );
printf ( "\nCOEF\tEXPON\n" );
for ( i = 0; i < nRecords; ++ i )
{
    data = getNextRecord ();
    addRear ( head2, data );
}

printf ( "\n\nPoly - 1\n" );
display ( head1 );

printf ( "\n\nPoly - 2\n" );
display ( head2 );

addPoly ( head1, head2 );

printf ( "\n\nPoly - 3\n" );
display ( head3 );

printf("\n\n");

//getch();
return 0;
}

```

---

.. ADDITION OF TWO POLYNOMIALS ..

Terms of Polynomial 1: 3

Give 1st record details one by one

COEF	EXPON
3	14
2	8
1	0

Terms of Polynomial 2: 3

Give 2nd record details one by one

COEF	EXPON
8	14
-3	10
10	6

Poly - 1

$$3x^{14} + 2x^8 + 1x^0$$

Poly - 2

$$8x^{14} - 3x^{10} + 10x^6$$

Poly - 3

$$11x^{14} - 3x^{10} + 2x^8 + 10x^6 + 1x^0$$

.. ADDITION OF TWO POLYNOMIALS ..

Terms of Polynomial 1: 3

Give 1st record details one by one

COEF	EXPON
3	3
2	2
1	1

Terms of Polynomial 2: 3

Give 2nd record details one by one

COEF	EXPON
3	3
2	2
1	1

Poly - 1

$$3x^3 + 2x^2 + 1x^1$$

Poly - 2

$$3x^3 + 2x^2 + 1x^1$$

Poly - 3

$$6x^3 + 4x^2 + 2x^1$$

---

## OPERATIONS ON BINARY SEARCH TREE (BST) OF INTEGERS

10. Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers

- a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
- b. Traverse the BST in Inorder, Preorder and Post Order
- c. Search the BST for a given element (KEY) and report the appropriate message
- e. Exit

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

typedef struct TreeData
{
    int key;
}Record;

typedef struct Node *ListPointer;
typedef struct Node
{
    ListPointer lLink;
    Record data;
    ListPointer rLink;
}Node;

ListPointer tree;

Record getNextRecord ( void )
{
    Record data;

    scanf ( "%d", &data.key );

    return data;
}

ListPointer getNode ( Record data )
{
    ListPointer temp;

    temp = ( ListPointer ) malloc ( sizeof ( *temp ) );

    if ( temp == NULL )
    {
        printf ( "\n\n Allocation Failed\n\n" );
        getchar (); //getch();
        exit ( EXIT_FAILURE );
    }
    else
    {
        temp->data = data;
        temp->lLink = temp->rLink = NULL;
    }
}
```

```

        return temp;
    }

void inorder ( ListPointer tree )
{
    if ( tree )
    {
        inorder ( tree->lLink );
        printf ( "%d\t", tree->data.key );
        inorder ( tree->rLink );
    }
}

void preorder ( ListPointer tree )
{
    if ( tree )
    {
        printf ( "%d\t", tree->data.key );
        preorder ( tree->lLink );
        preorder ( tree->rLink );
    }
}

void postorder ( ListPointer tree )
{
    if ( tree )
    {
        postorder ( tree->lLink );
        postorder ( tree->rLink );
        printf ( "%d\t", tree->data.key );
    }
}

int search ( ListPointer tree, Record data )
{
    while ( tree )
    {
        if ( data.key == tree->data.key )
        {
            return 1;
        }
        if ( data.key < tree->data.key )
        {
            tree = tree->lLink;
        }
        else
        {
            tree = tree->rLink;
        }
    }
    return 0;
}

ListPointer modifiedSearch ( ListPointer tree, Record data )
{
    ListPointer parent = NULL;

    while ( tree )
    {

```



```

        parent = tree;
        if ( data.key == tree->data.key )
        {
            return NULL;
        }
        if ( data.key < tree->data.key )
        {
            tree = tree->lLink;
        }
        else
        {
            tree = tree->rLink;
        }
    }
    return parent;
}

void insert ( Record data )
{
    ListPointer ptr, temp = modifiedSearch ( tree, data);
    if ( temp || !tree )
    {
        ptr = getNode( data );
        if ( tree )
        {
            if ( data.key < temp->data.key )
            {
                temp->lLink = ptr;
            }
            else
            {
                temp->rLink = ptr;
            }
        }
        else
        {
            tree = ptr;
        }
    }
}

int main ( void )
{
    Record data;
    int nRecords;
    int choice;
    int i;

    printf("\n.. BINARY SEARCH TREE DEMONSTRATION ..\n");
    printf ( "\n\n1. Insert\n");
    printf ( "\n2. Inoreder \t3. Preorder \t4. Postorder\n");
    printf ( "\n5. Search for an Item\n");
    printf ( "\n6. Exit\n");

    while ( 1 )
    {
        printf ( "\nChoice: " );
        scanf ( "%d", &choice );
    }
}

```

```

switch ( choice )
{
    case 1: printf("\n.. * INSERTION ..\n");
            printf ( "\nHow many RECORDS: " );
            scanf ( "%d", &nRecords );

            printf("\nGive %d record details one by one\n",
nRecords);

            for ( i = 0; i < nRecords; ++ i )
            {
                data = getNextRecord();
                insert ( data );
            }
            break;
    case 2: if ( tree )
            {
                printf ( "\n\n Inorder Traversal\n\n" );
                inorder ( tree );
            }
            else printf ( "\n\nBST is Empty\n\n" );
            break;
    case 3: if ( tree )
            {
                printf ( "\n\nPreorder Traversal\n\n" );
                preorder ( tree );
            }
            else
            {
                printf ( "\n\nBST is Empty\n\n" );
            }
            break;
    case 4: if ( tree )
            {
                printf ( "\n\n Postorder Traversal\n\n" );
                postorder ( tree );
            }
            else
            {
                printf ( "\n\nBST is Empty\n\n" );
            }
            break;
    case 5:
            if ( tree )
            {
                printf ( "\n\nSearch for an Item: ?\n" );
                data = getNextRecord();
                if ( search ( tree, data ) )
                {
                    printf ( "\n\n Item is Present is
BST\n\n" );
                }
                else
                {
                    printf ( "\n\n Item is not present in
BST\n\n" );
                }
            }
            break;
    case 6: return 0;
}

```

---

```
        default: printf ( "\nWrong Choice\n" );
    }
}
```

**OUTPUT:**

.. BINARY SEARCH TREE DEMONSTRATION ..

1. Insert

2. Inoreder      3. Preorder      4.Postorder

5. Search for an Item

6. Exit

Choice: 1

.. \* INSERTION ..

How many RECORDS: 12

Give 12 record details one by one

6

9

5

2

8

15

24

14

7

8

5

2

Choice: 2

Inorder Traversal

2      5      6      7      8      9  
14      15      24

Choice: 3

Preorder Traversal

6      5      2      9      8      7  
15      14      24

Choice: 4

Postorder Traversal

2      5      7      8      14      24  
15      9      6

Choice: 5

Search for an Item: ?

14

Item is Present is BST

Choice: 5

Search for an Item: ?

10

Item is not present in BST

Choice: 6 Exit

---

## IMPLEMENTATION OF HASH TABLE.

12. Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table(HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function  $H: K \rightarrow L$  as  $H(K)=K \text{ mod } m$  (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

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

#define NO_OF_BUCKETS 13

typedef struct EmployeeRecord
{
    char SSN[20];
    char name[20];
    char dept[20];
    char salary[20];
}Record;

typedef struct Node *ListPointer;
typedef struct Node
{
    Record data;
    ListPointer link;
}Node;

typedef enum
{
    available, duplicate, bucket_full
}Status;

ListPointer ht[NO_OF_BUCKETS];

int homeBucket, currentBucket;

Record getNextRecord ( void )
{
    Record data;

    scanf ( "%s", data.SSN );
    scanf ( "%s", data.name );
    scanf ( "%s", data.dept );
    scanf ( "%s", data.salary );

    return data;
}

ListPointer getNode ( Record data )
{
    ListPointer temp;
```

```

temp = ( ListPointer ) malloc ( sizeof ( *temp ) );

if ( temp == NULL )
{
    printf ( "\n\n Allocation Failed\n\n" );
    getchar (); //getch();
    exit ( EXIT_FAILURE );
}
else
{
    temp->data = data;
}

return temp;
}

int stringToInt ( char *key )
{
    int number = 0;

    while ( *key )
        number += *key ++;

    return number;
}

int myHash ( char *key )
{
    int k;

    k = stringToInt ( key );

    return k % NO_OF_BUCKETS;
}

Status search ( char *key )
{
    homeBucket = myHash ( key );

    currentBucket = homeBucket;

    while ( ht[currentBucket]
            &&
            strcmp ( ht[currentBucket]->data.SSN, key ) != 0 )
    {
        currentBucket = ( currentBucket + 1 ) % NO_OF_BUCKETS;
        /* treat the table as circular */

        if ( currentBucket == homeBucket )
            return bucket_full; /* back to start point */
    }

    if ( ht[currentBucket] )
    {
        if ( strcmp ( ht[currentBucket]->data.SSN, key ) == 0 )
            return duplicate;
    }

    return available;
}

```

```

void insert ( Record data )
{
    Status state;

    state = search ( data.SSN);

    switch ( state )
    {
        case available:    if ( ht[homeBucket] != NULL )
                            {
                                printf ( "\n\n Collision is Detected at the
Index:                                %d", homeBucket );
                                printf ( "\n\n Issue is solved by Linear
probing                                with the new Index: %d\n\n", currentBucket );
                                    }
                                ht[currentBucket] = getNode ( data );
                                    break;
        case duplicate:    printf("\nDuplicate key: %s\n", data.SSN);
                            break;
        case bucket_full: printf("\nAll buckets are occupied\n");
    }
}

void display ( void )
{
    int i;

    printf("\nEmployee details...\n");
    printf("\nINDEX\tSSN\tNAME\tDEPT.\tSALARY\n");
    for ( i = 0; i < NO_OF_BUCKETS; ++ i )
    {
        if ( ht[i])
        {
            printf ( "\n[%d] - %s\t", i, ht[i]->data.SSN );
            printf ( "%s\t", ht[i]->data.name );
            printf ( "%s\t", ht[i]->data.dept );
            printf ( "%s\n", ht[i]->data.salary );
        }
        else
        {
            printf ( "\n[%d] - %s\n", i, " " );
        }
    }
}

int main ( void )
{
    Record data;
    int nRecords;
    int choice;
    int i;
}

```

---

```

printf("\n..HASH DEMONSTRATION WITH LINEAR PROBING..\n");
printf ( "\n\n1. Insertion\n\n2. Display\n\n3. Exit\n");

while ( 1 )
{
    printf ( "\nEnter your Choice: " );
    scanf ( "%d", &choice );

    switch ( choice )
    {
        case 1: printf ( "\nHow many records: " );
                scanf ( "%d", &nRecords );

                printf("\nGive %d record details one by one\n",
nRecords);

                printf("\nSSN\tNAME\tDEPT.\tSALARY\n");

                for ( i = 0; i < nRecords; ++ i )
                {
                    data = getNextRecord();
                    insert ( data );
                }
                break;
        case 2: display ( );
                break;
        case 3: return 0;

        default: printf ( "\nWrong Choice\n" );
    }
}
}

```



**output:**

..HASH DEMONSTRATION WITH LINEAR PROBING..

1. Insertion

2. Display

3. Exit

Enter your Choice: 1

How many records: 6

Give 6 record details one by one

SSN	NAME	DEPT.	SALARY
One	Martin	CSE	1111
Two	Kate	CSE	2222
Three	Brian	ISE	5555
Four	Kathy	ECE	4444
Five	Daniel	ECE	3333

Collision is Detected at the Index: 10  
Issue is solved by Linear probing with the new Index: 11

six Emily ISE 8888

Collision is Detected at the Index: 2  
Issue is solved by Linear probing with the new Index: 4

Enter your Choice: 2

Employee details...

INDEX	SSN	NAME	DEPT.	SALARY
-------	-----	------	-------	--------

[0] -

[1] -

[2] - four Kathy ECE 4444

[3] - three Brian ISE 5555

[4] - six Emily ISE 8888

[5] -

[6] -

[7] -

[8] - two Kate CSE 2222

[9] -

[10] - one Martin CSE 1111

[11] - five Daniel ECE 3333

[12] -

Enter your Choice: 3  
Exit

..HASH DEMONSTRATION WITH LINEAR PROBING..

1. Insertion

2. Display

3. Exit

Enter your Choice: 1

How many records: 3

Give 3 record details one by one

SSN	NAME	DEPT.	SALARY
one	John	CSE	1111
two	Kathy	ISE	2222
one	John	CSE	1111

Duplicate key: one

Enter your Choice: 3  
Exit

..HASH DEMONSTRATION WITH LINEAR PROBING..

1. Insertion

2. Display

3. Exit

Enter your Choice: 1

How many records: 14

Give 14 record details one by one

SSN	NAME	DEPT.	SALARY
one	John	CSE	1111
two	Kathy	CSE	2222

three Martin ISE 3333  
 four Emily Ise 4444  
 five Helena ECE 5555

Collision is Detected at the Index: 10  
 Issue is solved by Linear probing with the new Index:  
 11

six Joan ECE 6666

Collision is Detected at the Index: 2  
 Issue is solved by Linear probing with the new Index:  
 4

seven Daniel CSE 2222  
 eight Brook ISE 5555  
 nine Bobby ISE 6666

Collision is Detected at the Index: 10  
 Issue is solved by Linear probing with the new Index:  
 0

ten Jack CSE 7777

Collision is Detected at the Index: 2  
 Issue is solved by Linear probing with the new Index:  
 5

eleven Tim ISE 8888

Collision is Detected at the Index: 2  
 Issue is solved by Linear probing with the new Index:  
 6

twelve Roger CSE 9999

Collision is Detected at the Index: 0  
 Issue is solved by Linear probing with the new Index:  
 1

thirty Sarah ISE 1111

Collision is Detected at the Index: 0  
 Issue is solved by Linear probing with the new Index:  
 7

fourty Curry ECE 3333

All buckets are occupied

Enter your Choice: 2

Employee details...

INDEX	SSN	NAME	DEPT.	SALARY
[0] -	nine	Boby	ISE	6666
[1] -	twelve	Roger	CSE	9999
[2] -	four	Emily	Ise	4444
[3] -	three	Martin	ISE	3333
[4] -	six	Joan	ECE	6666
[5] -	ten	Jack	CSE	7777
[6] -	eleven	Tim	ISE	8888
[7] -	thirty	Sarah	ISE	1111
[8] -	two	Kathy	CSE	2222
[9] -	eight	Brook	ISE	5555
[10] -	one	John	CSE	1111
[11] -	five	Helena	ECE	5555
[12] -	seven	Daniel	CSE	2222

Enter your Choice: 3  
 Exit

---

## OPERATION ON GRAPHS.

**12 . Design, Develop and Implement a program in C for the following operations on Graph (G) of cities**

- a. Create a Graph of N cities using Adjacency Matrix
- b. Print all the Nodes Reachable from a given starting node in a digraph using BFS method
- c. Check whether a given graph is connected or not using DFS method.

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

int adj[50][50];
int n; /* no. of vertex */
int visited[50] = {0};

int max = 50;
int count = 0;

void createMatrix ( void )
{
    int i, j;

    printf ( "\n\nEnter the adjacency matrix\n\n" );
    for ( i = 1; i <= n; ++ i )
        for ( j = 1; j <= n; ++ j )
            scanf ( "%d", &adj[i][j] );
}

void displayMatrix ( void )
{
    int i, j;
```

```

printf ( "\n\nAdjacency matrix\n\n\n" );
for ( i = 1; i <= n; ++ i )
{
    for ( j = 1; j <= n; ++ j )
        printf ( "%d\t", adj[i][j] );
    printf ( "\n\n" );
}

void dfs ( int v )
{
    int w;
    count = count + 1;
    visited[v] = count;
    printf ( "%-4d", v );
    for ( w = 1; w <= n; ++ w )
    {
        if ( adj[v][w] && !visited[w] )
        {
            dfs ( w );
        }
    }
}

int main ( void )
{
    int ch;
    int i;
    int sVertex;

    do{
        system("cls");
        printf ( "\n\n\t\t\tGRAPH DEMONSTRATION\n\n" );
        printf ( "\n\n\t\t1. Create Matrix\t\t2. Display Matrix\n\n" );
        printf ( "\n\n\t\t3. DFS\t\t4. Exit\n\n" );

        printf ( "\n\nEnter your choice: ?\b" );
        scanf ( "%d", &ch );

        switch ( ch )
        {
            case 1: printf ( "\n\nEnter No. of Vertex: ?\b" );
                    scanf ( "%d", &n );
                    createMatrix ( );
                    break;
            case 2: displayMatrix ( );
                    break;
            case 3: printf ( "\n\nEnter the Source Vertex: ?\b" );
                    scanf ( "%d", &sVertex );

                    /* Resetting visited[] and count */
                    for ( i = 1; i <= n; ++ i ) visited[i] = 0;
                    count = 0;

                    dfs ( sVertex );

                    break;
            case 4: return 0;
        }
    } while ( ch < 4 );
}

```

---

```
        default: printf ( "\n\nInvalid Option\n\n" );
    }
    fflush(stdin);
    getchar();
}while ( 1 );
}
```

---

## GRAPH DEMONSTRATION

1. Create Matrix 2. Display Matrix

3. DFS 4. Exit

Enter your choice: 1

Enter No. of Vertex: 8

Enter the adjacency matrix

```
0 1 1 0 1 0 0 0
1 0 1 1 1 0 1 0
1 1 0 0 0 0 0 0
0 1 0 0 1 0 1 1
1 1 0 1 0 0 0 1
0 0 0 0 0 1 1
0 1 0 1 0 1 0 1
0 0 1 1 1 1 0
```

## GRAPH DEMONSTRATION

1. Create Matrix 2. Display Matrix

3. DFS 4. Exit

Enter your choice: 2

Adjacency matrix

```
0 1 1 0 1 0 0 0
1 0 1 1 1 0 1 0
1 1 0 0 0 0 0 0
0 1 0 0 1 0 1 1
1 1 0 1 0 0 0 1
0 0 0 0 0 0 1 1
0 1 0 1 0 1 0 1
0 0 0 1 1 1 1 0
```

## GRAPH DEMONSTRATION

1. Create Matrix 2. Display Matrix

3. DFS 4. Exit

Enter your choice: 3

Enter the Source Vertex: 1

1 -> 2 -> 3 -> 4 -> 5 -> 8 -> 6 -> 7 ->

