COLLEGE OF ENGINEERING

THIRUVANATHAPURAM

DATA STRUCTURES
LABORATORY RECORD

DEPARTMENT OF COMPUTER SCIENCE AND

ENGINEERING

COLLEGE OF ENGINEERING

THIRUVANANTHAPURAM

…………………………………………………………

LABORATORY RECORD

University Reg. No. ………………………………………………………...

Name ………………………………………………………………...

Roll No……………………………………Batch ………………………

Class……………... from page No. ……...to page No. …………………..

CERTIFIED BONAFIDE RECORD OF WORK DONE BY

………………………………………………………………………

Thiruvananthapuram Staff Member in Charge

External Examiner

INDEX

Date:3-12-21
Experiment No: 1

IMPLEMENT STACK OPERATIONS WITH ARRAYS

AIM
Implement stack using arrays and perform push, pop and display the stack elements

Data Structure Used: Array, Stack

ALGORITHM
```
1) START
2) Create a stack using array implementation with name stack_arr
3) A menu is provided for user to choose whether to push or pop an element
   a) Push an element
   b) Pop an element
   c) Press any key to exit
4) If users choose option 1, then call function push(n) where n is the element to
   be pushed
5) Call function disp_stack() to display the contents of stack
6) If users choose option 2, then call function pop() and call function
   disp_stack() to display the contents
7) STOP
```

Definition of function push(int n)
```
   1) START
   2) Increment the value of top which is globally initialized with -1
   3) Let stack_arr[top] = n
   4) STOP
```
Definition of function pop()
```
   1) START
   2) If top=-1 then Print “Stack Underflow” and return
```

```
    3) Decrement top by 1
    4) STOP
```

Definition of function disp_stack()

```
    1) START
    2) for i = 0 to top Print stack_arr[i]
    3) STOP
```

PROGRAM

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

#define SIZE 100
int stack[SIZE], n, b, top = -1;

void push(int element);
void pop();
void display();

void main()
{

    printf("Enter the size of the stack : ");
    scanf("%d", &n);

    int response, choice, element;
    do
    {
        printf("\n\nM E N U\n\n1. Pushing elements to the Stack\n2. Popping elements from
the Stack\n3. Exit\n\n");
    ch:
        printf("\t  -> ");
        scanf("%d", &choice);
        printf("\n\n");
        switch (choice)
        {
        case 1:
            printf("Enter the element to be pushed : ");
            scanf("%d", &element);
            push(element);
            display();
            break;

        case 2:
            pop();
```

```
                display();
                break;

            case 3:
                exit(0);

            default:
                printf("Enter a valid Choice!!!!\n\n");
                go to ch;
            }
            printf("\nDo you want to continue (1 or 0) : ");
            scanf("%d", &response);
        } while (response == 1);
}

void push(int element)
{
    if (top >= n - 1)
    {
        printf("Stack Overflow!!!\n\n");
        return;
    }
    else
    {
        top++;
        stack[top] = element;
        printf("\n\n");
    }
}

void pop()
{
    if (top <= 0)
    {
        printf("\nStack Inflow!!!\n");
    }
    else
    {
        top--;
    }
}

void display()
{
    printf("Stack : \n");
    for (int i = top; i >= 0; i--)
    {
        printf("\t  %d\n", stack[i]);
    }
}
```

SAMPLE OUTPUT

```
                M E N U
1. Pushing elements into the stack
2. Popping elements from the stack
3. Exit
                -> 1
Enter the element to be pushed: 5

Stack:   5

                M E N U
1. Pushing elements into the stack
2. Popping elements from the stack
3. Exit
                -> 1
Enter the element to be pushed: 9

Stack:   5  9

        M E N U
1. Pushing elements into the stack
2. Popping elements from the stack
3. Exit
                -> 1
Enter the element to be pushed: 2

Stack:   5  9  2

        M E N U
1. Pushing elements into the stack
2. Popping elements from the stack
3. Exit
                -> 1
Enter the element to be pushed: 7
```

```
Stack:   5  9  2  7

        M E N U
1. Pushing elements into the stack
2. Popping elements from the stack
3. Exit
                -> 1
Enter the element to be pushed: 6

Stack:   5  9  2  7  6

        M E N U
1. Pushing elements into the stack
2. Popping elements from the stack
3. Exit
                -> 1
Enter the element to be pushed: 11

Stack:   5  9  2  7  6 11

        M E N U
1. Pushing elements into the stack
2. Popping elements from the stack
3. Exit
                -> 1
Enter the element to be pushed: 13

Stack:   5  9  2  7  6 11 13

        M E N U
1. Pushing elements into the stack
2. Popping elements from the stack
3. Exit
                -> 2
```

```
Stack:   5  9  2  7  6 11

        M E N U
1. Pushing elements into the stack
2. Popping elements from the stack
3. Exit
                -> 2

Stack:   5  9  2  7  6

        M E N U
1. Pushing elements into the stack
2. Popping elements from the stack
3. Exit
                -> 3
```

RESULT

Stack operations implemented successfully

Date:3-12-21
Experiment No: 2

CONVERSION OF INFIX TO POSTFIX AND EVALUATION OF POSTFIX

AIM

Convert a given infix expression to postfix and evaluate a given postfix expression

Data Structure Used: Stack

ALGORITHM

1) START
2) Create a stack 'stack' for infix to postfix conversion and evaluation of postfix
3) A menu is provided for users to choose their operation
4) If user chooses infix to postfix conversion, then call function InfixToPostfix ()
5) If user chooses evaluation of postfix expression then call the function evaluation ()
6) STOP

Definition of the function InfixToPostfix ()

1) START
2) Input the expression to the character array exp
3) Push "("onto stack and add ")" to the end of exp
4) 4) Scan exp from left to right and repeat step 5 to 8 for each element of exp until the stack is empty
5) If an operand is encountered, add it to string P
6) If a left parenthesis is encountered, push it onto stack
7) If an operator is encountered, then
    a) Repeatedly pop from the stack and add to P if the operator (on the top of stack) which has the same precedence or higher precedence than the operator

```
    b) Add the operator to stack
8) If a right parenthesis is encountered, then
    a) Repeatedly pop from stack and add to P each operator until a left
       parenthesis is encountered
    b) Remove the left parenthesis
9) STOP
```

Definition of function evaluation()

```
1) START
2) Input the postfix expression and store it in exp
3) Scan exp from left to right and repeat steps 4 and 5 for each element of exp
4) If an operand is encountered, then put it on stack
5) If an operator is encountered, then
    a) Remove the two top elements of stack where val1 is the top element and B
       is the next top element
    b) Evaluate val2 operator val1
    c) Place the result in ans push it back to stack
6) Set ans equal to the top element on the stack
7) STOP
```

PROGRAM

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

#define SIZE 100

char stack[SIZE];
int top = -1;

void push(char item)
{
    if (top >= SIZE - 1)
    {
        printf("\nStack Overflow.\n");
    }
    else
    {
        top = top + 1;
        stack[top] = item;
    }
}

char pop()
{
    char item;

    if (top < 0)
    {
        printf("stack under flow: invalid infix expression");
        getchar();
        exit(1);
    }
    else
    {
        item = stack[top];
        top = top - 1;
        return (item);
    }
}

int is_operator(char symbol)
{
    if (symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int precedence(char symbol)
{
    if (symbol == '^')
    {
        return (3);
    }
    else if (symbol == '*' || symbol == '/')
    {
        return (2);
    }
    else if (symbol == '+' || symbol == '-')
    {
        return (1);
    }
    else
    {
        return (0);
    }
}
```

```
void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
    int i, j;
    char item, x;

    push('(');
    strcat(infix_exp, ")");

    i = 0;
    j = 0;
    item = infix_exp[i];

    while (item != '\0')
    {
        if (item == '(')
        {
            push(item);
        }
        else if (isdigit(item) || isalpha(item))
        {
            postfix_exp[j] = item;
            j++;
        }
        else if (is_operator(item) == 1)
        {
            x = pop();
            while (is_operator(x) == 1 && precedence(x) >= precedence(item))
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
            push(x);
            push(item);
        }
        else if (item == ')')
        {
            x = pop();
            while (x != '(')
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
        }
        else
        {
            printf("\nInvalid infix Expression.\n");
            getchar();
            exit(1);
        }
        i++;
        item = infix_exp[i];
    }
    if (top > 0)
    {
        printf("\nInvalid infix Expression.\n");
        putchar(item);
        exit(1);
    }
    postfix_exp[j] = '\0';
}

int calculation(int a, int b, char O)
{
    int mult = 1;
    if (O == '+')
    {
        return a + b;
    }
    else if (O == '-')
    {
        return a - b;
    }
    else if (O == '*')
    {
        return a * b;
    }
    else if (O == '/' && b != 0)
    {
        return a / b;
    }
    else if (O == '^')
    {
        for (int i = 1; i <= b; i++)
        {
            mult = a * mult;
        }
        return mult;
    }
}
int evaluation(char postfix_exp[])
{
    char a;
    int i = 0, result;
    do
    {
        if (isdigit(postfix_exp[i]))
        {
            push(postfix_exp[i]);
        }
        else if (is_operator(postfix_exp[i]))
        {
            a = pop();
            push(calculation(pop() - '0', a - '0', postfix_exp[i]) + '0');
        }
        else if (isalpha(postfix_exp[i]))
        {
            printf("The expression contains a variable!!!");
            getchar();
            exit(1);
        }
        else
        {
            printf("\nInvalid Expression!!!\n\n");
            getchar();
            exit(1);
        }
        i++;
    } while (postfix_exp[i] != '\0');
    return pop();
}

int main()
{
    char infix[SIZE], postfix[SIZE];

    printf("\nEnter Infix expression : ");
    gets(infix);

    int response, choice, element;
    do
    {
        printf("\n\nM E N U\n\n1. Convert the Infix Expression into Postfix\n2. Evaluate
the postfix expression\n3. Exit\n\n");
    ch:
        printf("\t  -> ");
        scanf("%d", &choice);
        printf("\n\n");
        switch (choice)
        {
        case 1:
            InfixToPostfix(infix, postfix);
            printf("\nPostfix Expression: ");
            puts(postfix);
            break;

        case 2:
            printf("\nThe postfix expression equates to %c\n", evaluation(postfix));
            break;

        case 3:
            exit(0);

        default:
            printf("Enter a valid Choice!!!!\n\n");
            go to ch;
        }
        printf("\nDo you want to continue (1 or 0) : ");
        scanf("%d", &response);
    } while (response == 1);

    return 0;
}
```

SAMPLE OUTPUT

```
            M E N U
1.  Convert the Infix Expression into Postfix
2. Evaluate the postfix expression
3. Exit
            -> 1
Enter the expression: A*(B+C)/E-F*(G+H/K)
The postfix expression is: ABC+*E/FGHK/+*-

      M E N U
1.  Convert the Infix Expression into Postfix
2. Evaluate the postfix expression
3. Exit
            -> 2
Enter the expression to be evaluated: 2,3,1,*,+,9,-
The answer on evaluation of postfix expression: -4
```

RESULT

Infix expression is converted to postfix and postfix expression is evaluated

Date:3-12-2021
Experiment No. 3

POLYNOMIAL ADDITION

AIM

To read two polynomial and display them and their sum

Data Structure Used: Array, Stack

ALGORTHM

1) START
2) Define an array of structure with elements coef and exp of type int
3) Input the degree of polynomial A and B
4) Input the coefficient and exponent of each term and call the function readpoly (num, expon) for each terms of A and B
5) Call the function addpoly()
6) Display the polynomial A, B and their sum polynomial C using their starting and ending indices
7) STOP

Definition of function int readpoly(int, int)

1) START
2) avail, StartA and StartB are initialized with zero globally
3) Let term[avail].coef = num
4) Let term[avail].exp = expon
5) Let finish = avail
6) Let avail = avail+1 and return finish
7) STOP

Definition of the function void addpoly()

1) START
2) Let stA = StartA and stB = StartB
3) Repeat Step 4 to 6 while A and B are not empty

4) If term[stA].exp = term[stB].exp, add the coefficients of leading exponents of A and B. Attach the sum and add it to polynomial C.Remove the terms from both A and B.
5) If term[stA].exp < term[stB].exp, attach the term[stB].exp and term[stB].coef to C and remove the term from B
6) If term[stA].exp > term[stB].exp, attach term[stA].exp and its coefficient to C and remove the term from A
7) Insert any remaining terms of A or B to C
8) STOP

PROGRAM

```
#include <stdio.h>
struct polynomial
{
    int coef;
    int exp;
} term[150];

int avail = 0, startA = 0, finishA = 0, startB = 0, finishB = 0, startC, finishC;
int readpoly(int, int, int);
void addpoly();
void main()
{
    int num, expon, i, deg;
    char ans;
    printf("Enter the degree of the polynomial A: ");
    scanf("%d", &deg);
    printf("\nEnter the polynomial A\n");
    for (i = 0; i <= deg; i++)
    {
        printf("\nEnter the coefficient: ");
        scanf("%d", &num);
        printf("Enter the exponent: ");
        scanf("%d", &expon);
        finishA = readpoly(num, expon, finishA);
    }
    startB = avail;
    printf("\nEnter the degree of the polynomial B: ");
    scanf("%d", &deg);
    printf("\nEnter the polynomial B\n");
    for (i = 0; i <= deg; i++)
    {
        printf("\nEnter the coefficient: ");
        scanf("%d", &num);
        printf("Enter the exponent: ");
        scanf("%d", &expon);
        finishB = readpoly(num, expon, finishB);
    }
    addpoly();
    printf("\nPolynomial A is: ");
    for (i = startA; i < finishA; i++)
        printf("%dx^%d + ", term[i].coef, term[i].exp);
    printf("%dx^%d\n", term[i].coef, term[i].exp);
    printf("\nPolynomial B is: ");
    for (i = startB; i < finishB; i++)
        printf("%dx^%d + ", term[i].coef, term[i].exp);
    printf("%dx^%d\n", term[i].coef, term[i].exp);
    printf("\nSum of polynomial is: ");
    for (i = startC; i < finishC; i++)
        printf("%dx^%d + ", term[i].coef, term[i].exp);
    printf("%dx^%d\n", term[i].coef, term[i].exp);
}
int readpoly(int num, int expon, int finish)
{
    term[avail].coef = num;
    term[avail].exp = expon;
    finish = avail;
    avail++;
    return finish;
}
void addpoly()
{
    startC = avail;
    finishC = avail;
    int stA = startA, stB = startB, i;
    while (stA <= finishA && stB <= finishB)
    {

        if (term[stA].exp == term[stB].exp)
        {
            term[avail].coef = term[stA].coef + term[stB].coef;
            term[avail].exp = term[stA].exp;
            finishC = avail;
            avail++;
            ++stB;
            stA++;
        }
        else if (term[stA].exp < term[stB].exp)
        {
            term[avail].coef = term[stB].coef;
            term[avail].exp = term[stB].exp;
            finishC = avail;
            stB++;
            avail++;
        }
        else if (term[stA].exp > term[stB].exp)
        {
            term[avail].coef = term[stA].coef;
            term[avail].exp = term[stA].exp;
            finishC = avail;
            stA++;
            avail++;
        }
    }
    while (stA > finishA && stB <= finishB)
    {
        for (i = stB; i <= finishB; i++)
        {
            if (term[i].exp == term[stA - 1].exp)
            {
                term[avail].coef = term[i].coef + term[stA - 1].coef;
                term[avail].exp = term[i].exp;
                finishC = avail;
                avail++;
            }
            else
            {
                term[avail].coef = term[i].coef;
                term[avail].exp = term[i].exp;
                finishC = avail;
                avail++;
            }
        }
    }
    while (stA <= finishA && stB > finishB)
    {
        for (i = stA; i <= finishA; i++)
        {
            if (term[i].exp == term[stB - 1].exp)
            {
                term[avail].coef = term[i].coef + term[stB - 1].coef;
                term[avail].exp = term[i].exp;
                finishC = avail;
                avail++;
            }
            else
            {
                term[avail].coef = term[i].coef;
                term[avail].exp = term[i].exp;
                finishC = avail;
                avail++;
            }
        }
    }
}
```

SAMPLE OUTPUT

```
Enter the degree of the polynomial A: 3

Enter the polynomial A

Enter the coefficient: 4
Enter the exponent: 3

Enter the coefficient: 5
Enter the exponent: 2

Enter the coefficient: 9
Enter the exponent: 1

Enter the coefficient: 0
Enter the exponent: 0

Enter the degree of the polynomial B: 2
Enter the polynomial B
Enter the coefficient: 5
Enter the exponent: 2

Enter the coefficient: 7
Enter the exponent: 1

Enter the coefficient: 9
Enter the exponent: 0

Polynomial A is: 4x^3 + 5x^2 + 9x^1 + 0x^0

Polynomial B is: 5x^2 + 7x^1 + 9x^0

Sum of polynomial is: 4x^3 + 10x^2 + 16x^1 + 9x^0
```

RESULT

Polynomials read and their sum found successfully

Date:20-12-2021
Experiment No. 4

MATRICES IN TUPLE FORM

AIM

To convert two matrices to tuple form and find their transpose and sum in tuple form

Data Structure Used: Array

ALGORITHM

1) START
2) Read two 2-D matrices in normal form
3) Define an array of structure with elements col, row and val globally
4) Call the function readtupA() and readtupB() to change the matrices A and B to tuple form of array structure a and b
5) Call the function transposeTupA() and transposetupB() to find the transpose of the matrices in tuple form in the array structure transposeA[] and transposeB[]
6) Call the function sumTup() to find the sum of matrices in tuple form in the array structure sum
7) STOP

Definition of the function void readtupA()

1) START
2) Let the number of rows and columns of A matrix be m and n
3) Let a[0].row = m, a[0].col = n and k=1
4) For each element of matrix, for i=0 to m and j=0 to n, if A[i][j] != 0 then a[k].row = i, a[k].col = j and a[k].val = A[i][j] and increment k by 1
5) Let a[0].val = k-1
6) STOP

Definition of the function void readtuA (int A[][10], int m, int n)

```
1) START
2) Let the number of rows and columns of B matrix be m and n
3) Let b[0].row = m, b[0].col = n and k=1
4) For i=0 to m and j=0 to n,if B[i][j] != 0 then b[k].row = i, b[k].col = j and b[k].val = B[i][j] and increment k by 1
5) Let b[0].val = k-1
6) STOP
```

Definition of the function transposeTupA()

```
1) START
2) Assign transposeA[0].row = a[0].row and transposeA[0].col = a[0].col and transpose[0].val = a[0].val
3) Let k=1
4) For i=0 to a[0].col repeat Step 5
5) For j=1 to a[0].val, repeat Step 6
6) If a[j].col = i then transposeA[k].row = a[j].col, transposeA[k].col = a[j].row and transposeA[k].val = a[j].val and increment k by 1
7) STOP
```

Definition of the function void transposeTupB()

```
1) START
2) Assign transposeB[0].row = b[0].row and transposeB[0].col = b[0].col and transposeB[0].val = b[0].val
3) Let k=1
4) For all j=1 to b[0].col, repeat Step 5
5) For all j=1 to b[0].val, repeat Step 6
6) If b[j].col = i, then transposeB[k].row = b[j].col , transposeB[k].col = b[j].row, transposeB[k].val = b[j].val and increment k by 1
7) STOP
```

Definition of the function void sumTup()

```
1) START
2) If a[0].row = b[0].row or a[0].col = b[0].col then STOP
3) Let k =1, i=1 and j=1
```

```
4) While i<=a[0].val and j<=b[0].val, repeat Steps 5 to 7
5) If a[i].row = b[j].row and a[j].col = b[j].col then sum[k].row = a[i].row,
   sum[k].col = a[i].col, sum[k].val = a[i].val + b[j].val and increment k, i, j
   by 1
6) If a[i].row < b[j].row or (a[i].row = b[j].row and a[i].col < b[j].col), then
         sum[k].row = a[i].row, sum[k].col = a[i].col, sum[k].val = a[i].val and
         increment k and i by 1
7) If a[i].row > b[j].row or (a[i].row = b[j].row and a[i].col > b[j].col) then,
         sum[k].row = b[j].row, sum[k].col = b[j].col, sum[k].val = b[j].val and
         increment k, j by 1
8) If i>a[0].val and j<=b[0].val then, sum[k].row = b[i].row, sum[k].col =
   b[i].col,    sum[k].val = b[i].val and increment k and j by 1
9) If i<=a[0].val and j>b[0].val the, sum[k].row = a[i].row, sum[k].col =
         a[i].col, sum[k].val = a[i].val and increment k and i by 1
10)      Let sum[0].val = k-1
11)      STOP
```

PROGRAM

```c
// Matrix in tuple
#include <stdio.h>
struct Tuple
{

    int row;
    int col;
    int val;
};

struct Tuple a[25], b[25], transposeA[25], transposeB[25], sum[50];

void changetupA(int M[][10], int m, int n);
void changetupB(int M[][10], int m, int n);
void transposeTupA();
void transposeTupB();
void sumTup();
void main()
{
    int i, j, m1, n1, m2, n2, A[10][10], B[10][10];
    printf("Enter the rows and columns of the first matrix: ");
    scanf("%d %d", &m1, &n1);
    printf("Enter the rows and columns of the second matrix: ");
    scanf("%d %d", &m2, &n2);
    printf("Enter the first matrix:\n");
    for (i = 0; i < m1; i++)
        for (j = 0; j < n1; j++)
            scanf(" %d", &A[i][j]);
    printf("Enter the second matrix:\n");
    for (i = 0; i < m2; i++)
        for (j = 0; j < n2; j++)
            scanf(" %d", &B[i][j]);
    changetupA(A, m1, n1);
    printf("\nThe first matrix in tuple form:\n");
    for (i = 0; i <= a[0].val; i++)
        printf("%4d%4d%4d\n", a[i].row, a[i].col, a[i].val);
    changetupB(B, m2, n2);
    printf("\nThe second matrix in tuple form:\n");
    for (i = 0; i <= b[0].val; i++)
        printf("%4d%4d%4d\n", b[i].row, b[i].col, b[i].val);
    transposeTupA();
    printf("\nThe transpose of first matrix:\n");
    for (i = 0; i <= transposeA[0].val; i++)
        printf("%4d%4d%4d\n", transposeA[i].row, transposeA[i].col, transposeA[i].val);
    transposeTupB();
    printf("\nThe transpose of second matrix:\n");
    for (i = 0; i <= transposeB[0].val; i++)
        printf("%4d%4d%4d\n", transposeB[i].row, transposeB[i].col, transposeB[i].val);
    sumTup();
    printf("\nThe transpose of second matrix:\n");
    for (i = 0; i <= transposeB[0].val; i++)
        printf("%4d%4d%4d\n", transposeB[i].row, transposeB[i].col, transposeB[i].val);
    printf("\nThe sum of the tuples:\n");
    for (i = 0; i <= sum[0].val; i++)
        printf("%4d%4d%4d\n", sum[i].row, sum[i].col, sum[i].val);
}

void changetupA(int M[10][10], int m, int n)
{
    int i, j, k = 1;
    a[0].row = m;
    a[0].col = n;
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            if (M[i][j] != 0)
            {
                a[k].row = i;
                a[k].col = j;
                a[k].val = M[i][j];
                k++;
            }
    a[0].val = k - 1;
}

void changetupB(int M[10][10], int m, int n)
{
    int i, j, k = 1;
    b[0].row = m;
    b[0].col = n;
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            if (M[i][j] != 0)
            {
                b[k].row = i;
                b[k].col = j;
                b[k].val = M[i][j];
                k++;
            }
    b[0].val = k - 1;
}
void transposeTupA()
{
    int i, j, k = 1;
    transposeA[0].row = a[0].col;
    transposeA[0].col = a[0].row;
    transposeA[0].val = a[0].val;
    for (i = 0; i < a[0].col; i++)
        for (j = 1; j <= a[0].val; j++)
            if (a[j].col == i)
            {
                transposeA[k].row = a[j].col;
                transposeA[k].col = a[j].row;
                transposeA[k].val = a[j].val;
                k++;
            }
}

void transposeTupB()
{
    int i, j, k = 1;
    transposeB[0].row = b[0].col;
    transposeB[0].col = b[0].row;
    transposeB[0].val = b[0].val;
    for (i = 0; i < b[0].col; i++)
        for (j = 1; j <= b[0].val; j++)
            if (b[j].col == i)
            {
                transposeB[k].row = b[j].col;
                transposeB[k].col = b[j].row;
                transposeB[k].val = b[j].val;
                k++;
            }
}
void sumTup()
{
    int i = 1, j = 1, k = 1;
    sum[0].row = a[0].row;
    sum[0].col = a[0].col;
    if (a[0].row != b[0].row || a[0].col != b[0].col)
        return;
    while (i <= a[0].val && j <= b[0].val)
    {
        if ((a[i].row == b[j].row) && (a[i].col == b[j].col))
        {
            sum[k].row = a[i].row;
            sum[k].col = a[i].col;
            sum[k].val = a[i].val + b[j].val;
            k++;
            i++;
            j++;
        }
        else if ((a[i].row < b[j].row) || ((a[i].row == b[j].row) && (a[i].col <
b[j].col)))
        {
            sum[k].row = a[i].row;
            sum[k].col = a[i].col;
            sum[k].val = a[i].val;
            k++;
            i++;
        }
        else if ((a[i].row > b[j].row) || ((a[i].row == b[j].row) && (a[i].col >
b[j].col)))
        {
            sum[k].row = b[j].row;
            sum[k].col = b[j].col;
            sum[k].val = b[j].val;
            k++;
            j++;
        }
    }
    if (i > a[0].val && j <= b[0].val)
    {
        sum[k].row = b[j].row;
        sum[k].col = b[j].col;
        sum[k].val = b[j].val;
        k++;
        j++;
    }
    if (i <= a[0].val && j > b[0].val)
    {
        sum[k].row = a[i].row;
        sum[k].col = a[i].col;
        sum[k].val = a[i].val;
        k++;
        i++;
    }
    sum[0].val = k - 1;
    printf("%d", sum[0].val);
}
```

SAMPLE OUTPUT

```
Enter the rows and columns of the first matrix: 4 4
Enter the rows and columns of the second matrix: 4 4
Enter the first matrix:
0 0 1 4
0 0 0 1
0 5 0 0
0 0 0 0
Enter the second matrix:
10 0 0 0
0 6 0 0
0 6 0 0
0 0 0 20

The first matrix in tuple form:
  4   4   4
  0   2   1
  0   3   4
  1   3   1
  2   1   5

The second matrix in tuple form:
  4   4   4
  0   0  10
  1   1   6
  2   1   6
  3   3  20
The transpose of first matrix:
  4   4   4
  1   2   5
  2   0   1
  3   0   4
  3   1   1
```

```
The transpose of second matrix:
   4   4   4
   0   0  10
   1   1   6
   1   2   6
   3   3  20

The sum of the tuples:
   4   4   7
   0   0  10
   0   2   1
   0   3   4
   1   1   6
   1   3   1
   2   1  11
   3   3  20
```

RESULT

Matrices converted to tuple form and their sum and product found in tuple form

Date: 20-12-2022
Experiment No. 5

IMPLEMENTATION OF QUEUE

AIM

To implement a queue using array

Data Structure Used: Array, Queue

ALGORITHM

```
1) START
2) Declare an array named queue_arr[] and let front = 0,rear = -1
3) Provide the user with a menu having options
        i)  Add to queue
        ii) Delete from queue
        iii)    Exit
4) If user choose option i, invoke function enqueue()
5) If user choose option ii, invoke function dequeue()
6) If user choose option iii, go to Step 7
7) STOP
```

Definition of function enqueue(int n)

```
    1) START
    2) If rear = -1 then rear=0 and queue_arr[rear] = n else go to Step 3
    3) If  rear – max-1 then Print “Queue Overflow” else go to Step 4
    4) Let rear = rear+1 and queue_arr[rear] = n
    5) STOP
```

Definition of function dequeue()

```
    1) START
    2) If front = rear then Print “Queue Underflow” else go to Step 3
    3) Let front = front+1
    4) STOP
```

Definition of function display()

```
1) START
2) For i=front to i<=rear, Print queue_arr[i]
3) STOP
```

PROGRAM

```c
#include <stdio.h>

void enqueue(int n);
void dequeue();
void display();

#define max 10
int queue_arr[max];
int front = 0, rear = -1;

void main()
{
    int opt, inp;
    while (1)
    {
        printf("\nChoose your option:\n");
        printf("1.Enqueue\n2.Dequeue\n3.Press any other key to exit\n");
        printf("Enter your option: ");
        scanf("%d", &opt);
        if (opt == 1)
        {
            printf("Enter the element to be inserted: ");
            scanf("%d", &inp);
            enqueue(inp);
            display();
        }

        else if (opt == 2)
        {
            dequeue();
            display();
        }
        else
            break;
    }
}
void enqueue(int n)
{
    if (rear == max)
```

```
        printf("\nQueue Overflow");
    else
    {
        rear++;
        queue_arr[rear] = n;
    }
}
void dequeue()
{
    if (front > rear)
    {
        printf("\nQueue Underflow");
        front = 0;
        rear = -1;
    }
    else
        front++;
}
void display()
{
    if (front <= rear)
    {
        int i;
        printf("\nThe elements of the queue: ");
        for (int i = front; i <= rear; i++)
            printf("%4d", queue_arr[i]);
        printf("\n");
    }
}
```

SAMPLE OUTPUT

```
Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 1
The elements of the queue:    1

Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 2
The elements of the queue:    1   2

Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 3
The elements of the queue:    1   2   3


Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 4
The elements of the queue:    1   2   3   4

Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 2
The elements of the queue:    2   3   4

Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 1
The elements of the queue:    2   3   4   1
```

```
Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 3
```

RESULT

Queue implemented successfully

Date: 20-12-2022
Experiment No: 6

IMPLEMENTATION OF CIRCULAR QUEUE

AIM

To implement circular queue using array

Data Structure Used: Array, Queue

ALGORIHTM

```
1) START
2) A menu is provided to the user having options
        i)  Add to queue
        ii) Delete from queue
        iii)    Exit
3) Declare an array named circ_arr[] and let front = -1, rear= -1
4) If user chooses option i, invoke function enqueue()
5) If user chooses option ii, invoke function dequeue()
6) If user chooses option iii, go to Step 7
7) STOP
```

Definition of function enqueue(int n)

```
1) START
2) If front = rear = -1 then
        i)  front = rear = 0
        ii) circ_arr[rear] = n
        iii)    go to Step 6
3) If rear = max-1 then rear=0 else rear=rear+1
4) If front = rear, Print "Queue Overflow" and go to Step 6
5) Let circ_arr[rear] = n
6) STOP
```

Definition of function dequeue()

```
1) START
2) If front = -1,then Print “Queue Empty” and go to Step 5
3) If front = rear, then front = rear = -1 and go to Step 5
4) If front = max-1, front = 0 else front = front+1
5) STOP
```

Definition of function display()

```
1) START
2) If front = -1 then go to Step 6
3) If rear >= front, for i=front to i <= rear, print circ_arr[i] and go to Step 6
4) For i=front to i< max print circ_arr[i]
5) For i=0 to i <= rear print circ_arr[i]
6) STOP
```

PROGRAM

```c
#include <stdio.h>
void enqueue(int n);
void dequeue();
void display();

#define max 5
int circ_arr[max];
int front = -1, rear = -1;
void main()
{
    int opt, inp;
    while (1)
    {
        printf("\nChoose your option:\n");
        printf("1.Enqueue\n2.Dequeue\n3.Press any other key to exit\n");
        printf("Enter your option: ");
        scanf("%d", &opt);
        if (opt == 1)
        {
            printf("Enter the element to be inserted: ");
            scanf("%d", &inp);
            enqueue(inp);
            display();
        }
        else if (opt == 2)
        {
            dequeue();
            display();
        }
        else
            break;
    }
}
void enqueue(int n)
{
    if (rear == -1)
    {
        front = rear = 0;
        circ_arr[rear] = n;
    }
    else if (rear == max - 1 && front > 0)
    {
        rear = 0;
        circ_arr[rear] = n;
    }
    else if ((front == rear + 1) || (rear == max - 1 && front == 0))
        printf("\nQueue Overflow\n");
    else
    {
        rear++;
        circ_arr[rear] = n;
    }
}
```

SAMPLE OUTPUT

```
Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 1

The elements of queue:    1
Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 2
```

```
The elements of queue:    1   2
Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 3

The elements of queue:    1   2   3
Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 4

The elements of queue:    1   2   3   4
Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 2

The elements of queue:    2   3   4
Choose your option:
1.Enqueue
2.Dequeue
3.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 1

The elements of queue:    2   3   4   1
Choose your option:
1.Enqueue
2.Dequeue
```

```
3.Press any other key to exit
Enter your option: 3
```

RESULT

Circular Queue implemented successfully

Date: 20-12-2022
Experiment No: 7

IMPLEMENTATION OF DOUBLE ENDED QUEUE

AIM

To implement a double ended queue using array

Data Structure Used: Array, Queue

ALGORITHM

```
1) START
2) Declare an array dequeue_arr[] and let front = -1
3) and rear = -1
4) A menu is provided to the user having the following options
        i)  Add to the front
        ii) Delete from the front
        iii)    Add to rear
        iv)Delete from rear
        v) Exit
5) If user choose i, invoke function addfront(int n)
6) If user choose ii, invoke function deletefront()
7) If user choose iii, invoke function addrear(int n)
8) If user choose iv, invoke function deleterear()
9) If user choose option v, go to Step 9
10)   STOP
```

Definition of function of addfront(int n)

```
1) START
2) If front =0 and rear = max-1 then
        i)  front = max-1
        ii) dequeue_arr[front] = n
        iii)    go to Step 5
```

```
3) If front = rear+1 or (front=0 and rear=max-1) then print “Queue Overflow”
   and go to Step 5
4) Let front = front-1 and dequeue_arr[front] = n
5) STOP
```

Definition of function deletefront()

```
1) START
2) If front = -1 Print “Queue Overflow” and go to Step 6
3) If front= rear, let front=rear=-1 and go to Step 6
4) If front = max-1 and rear < front then front=0
5) Let front = front+1
6) STOP
```

Definition of addrear(int n)

```
1) START
2) If rear = -1 then
       i)  front = rear = 0
       ii) dequeue_arr[rear] = n
       iii)    go to Step 6
3) If rear = max-1 and front>0
       i)  rear = 0
       ii) dequeue_arr[rear] = n
       iii)    go to Step 6
4) If front=rear+1 or rear=max-1 and front=0, then Print “Queue Overflow” and
   go to Step 6
5) Let rear=rear+1 and dequeue_arr[rear] = n
6) STOP
```

Definition of deleterear()

```
    1) START
    2) If rear = -1, Print “Queue Underflow” and go to Step 6
    3) If front=rear let front=rear=-1 and go to Step 6
    4) If rear=max-1 and front<rear then rear=0 and go to Step 6
    5) Let rear = rear-1
    6) STOP
```

Definition of display()

```
1) START
2) If rear=-1 go to Step 6
3) If rear >= front then, for i=front to i<=rear print dequeue_arr[i] and go to
   Step 6
4) For i= front to i=max-1, Print dequeue_arr[i]
5) For i=0 to i<=rear, print dequeue_arr[i]
6) STOP
```

PROGRAM

```
#include <stdio.h>

void addfront(int n);
void addrear(int n);
void deletefront();
void deleterear();
void display();

#define max 5
int dqueue_arr[max];
int front = -1, rear = -1;

void main()
{
    int opt, inp;
    while (1)
    {
        printf("\nChoose your option:\n");
        printf("1.Add to front\n2.Add to rear\n3.Delete from front\n4.Delete from
rear\n5.Press any other key to exit\n");
        printf("Enter your option: ");
        scanf("%d", &opt);
        if (opt == 1)
        {
            printf("Enter the element to be inserted: ");
            scanf("%d", &inp);
            addfront(inp);
            display();
        }
        else if (opt == 2)
        {
            printf("Enter the element to be inserted: ");
            scanf("%d", &inp);
            addrear(inp);
            display();
        }
        else if (opt == 3)
        {
            deletefront();
            display();
        }
        else if (opt == 4)
        {
            deleterear();
            display();
        }
        else
            break;
    }
}
void addrear(int n)
{
    if (rear == -1)
    {
        front = rear = 0;
        dqueue_arr[rear] = n;
    }

    else if (rear == max - 1 && front > 0)
    {
        rear = 0;
        dqueue_arr[rear] = n;
    }
    else if ((front == rear + 1) || (rear == max - 1 && front == 0))
        printf("\nQueue Overflow\n");
    else
    {
        rear++;
        dqueue_arr[rear] = n;
    }
}
void deletefront()
{
    if (front == -1)
        printf("\nQueue Underflow\n");
    else if (front == rear)
        front = rear = -1;
    else if (front == max - 1 && rear < front)
        front = 0;
    else
        front++;
}
void addfront(int n)
{
    if (front == 0 && rear < max - 1)
    {
        front = max - 1;
        dqueue_arr[front] = n;
    }
    else if ((front == rear + 1) || (front == 0 && rear == max - 1))
        printf("\nQueue Overflow\n");
    else
    {
        front--;
        dqueue_arr[front] = n;
    }
}
void deleterear()
{
    if (rear == -1)
        printf("\nQueue Underflow\n");
    else if (front == rear)
        front = rear = -1;
    else if (rear == max - 1 && front < rear)
        rear = 0;
    else
        rear--;
}
void display()
{
    int i;
    printf("\nThe elements of queue: ");
    if (rear == -1)
        return;
    else if (rear >= front)
        for (i = front; i <= rear; i++)
            printf("%4d", dqueue_arr[i]);
    else
    {
        for (i = front; i < max; i++)
            printf("%4d", dqueue_arr[i]);
        for (i = 0; i <= rear; i++)
            printf("%4d", dqueue_arr[i]);
    }
}
```

SAMPLE OUTPUT

```
Choose your option:
1.Add to front
2.Add to rear
3.Delete from front
4.Delete from rear
5.Press any other key to exit
Enter your option: 2
Enter the element to be inserted: 1

The elements of queue:    1

Choose your option:
1.Add to front
2.Add to rear
3.Delete from front
4.Delete from rear

5.Press any other key to exit
Enter your option: 2
Enter the element to be inserted: 1

The elements of queue:    1   1

Choose your option:
1.Add to front
2.Add to rear
3.Delete from front
4.Delete from rear
5.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 2

The elements of queue:    2   1   1

Choose your option:
1.Add to front
2.Add to rear
3.Delete from front
4.Delete from rear
5.Press any other key to exit
Enter your option: 1
Enter the element to be inserted: 3
```

```
The elements of queue:    3   2   1   1

Choose your option:
1.Add to front
2.Add to rear
3.Delete from front
4.Delete from rear
5.Press any other key to exit
Enter your option: 4

The elements of queue:    3   2   1

Choose your option:
1.Add to front
2.Add to rear
3.Delete from front
4.Delete from rear
5.Press any other key to exit
Enter your option: 3

The elements of queue:    2   1

Choose your option:
1.Add to front
2.Add to rear
3.Delete from front
4.Delete from rear
5.Press any other key to exit
Enter your option: 5
```

RESULT

Double Ended Queue implemented successfully

Date: 17-01-2022
Experiment No. 8

IMPLEMENTING PRIORITY QUEUE

AIM

To implement priority queue using arrays

Data Structure Used: Array, Queue

ALGORITHM

```
1) START
2) Define a structure Priority with members value and priority and define an
   array pqueue of structure Priority and let front = rear = -1
3) A menu is provided to the user to select their choice
      i)  Enqueue
      ii) Dequeue
      iii)   Exit
4) If user choose i, input the element to be inserted and its priority as inp and pr
   and invoke the function enqueue(inp, pr)
5) If user chooses ii, invoke the function dequeue()
6) If user chooses iii, go to Step 7
7) STOP
```

Definition of function enqueue(int n, int p)

```
   1) START
   2) If front = -1, then let front = rear = 0 and go to Step 5
   3) If rear = max-1, then Print "Queue Overflow" and go to Step 6
   4) Let rear = rear+1
   5) Let pqueue[rear].value = n and pqueue[rear].priority = p
   6) STOP
```

Definition of function dequeue()

```
   1) START
```

```
2) Let tempr = pqueue[front].priority and index = front
3) If rear = -1 or front > rear Print “Queue Underflow” and go to Step 11
4) If front = rear then front = rear = -1 and go to Step 11
5) For i= front to i<=rear repeat Step 6
6) If pqueue[i].priority > tempr then tempr = pqueue[i].priority and index = i
7) For i=index to i<rear repeat Steps 8 and 9
8) Let pqueue[i].value = pqueue[i+1].value
9) Let pqueue[i].priority = pqueue[i+1].priority
10)     Let rear = rear+1
11)     STOP
```

Definition of function display()

```
1) START
2) If front>-1 and rear<max-1 then for i=front to i<=rear repeat Step 3
3) Print pqueue[i].value
4) STOP
```

PROGRAM

```
#include <stdio.h>

#define max 10
struct Priority
{
    int value;
    int priority;
} pqueue[10];

int front = -1, rear = -1;

void enqueue(int n, int p);
void dqueue();
void display();
void main()
{
    int opt, inp, pr;
    while (1)
    {
        printf("\nChoose your option\n");
        printf("1.Enqueue\t2. Dqueue\t3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &opt);
        if (opt == 1)
        {
            printf("\nEnter the element to be inserted: ");
            scanf("%d", &inp);
            printf("Enter the priority of the element: ");
            scanf("%d", &pr);
            enqueue(inp, pr);
            display();
        }
        else if (opt == 2)
        {
            dqueue();
            display();
        }
        else
            break;
    }
}
void enqueue(int n, int p)
{
    if (front == -1)
        front = rear = 0;
    else if (rear == max - 1)
    {
        printf("\nQueue Overflow!!\n");
        return;
    }
    else
        rear++;
    pqueue[rear].value = n;
    pqueue[rear].priority = p;
}
void dqueue()
{
    int tempr = pqueue[front].priority, i, index = front;
    if (rear == -1 || front > rear)
        printf("\nQueue Underflow!!\n");
    else if (front == rear)
        front = rear = -1;
    else
    {
        for (i = front; i <= rear; i++)
            if (pqueue[i].priority > tempr)
            {
                tempr = pqueue[i].priority;
                index = i;
            }
        for (i = index; i < rear; i++)
        {
            pqueue[i].value = pqueue[i + 1].value;
            pqueue[i].priority = pqueue[i + 1].priority;
        }
        rear--;
    }
}
void display()
{
    int i;
    if (front > -1 && rear < max - 1)
        for (i = front; i <= rear; i++)
            printf("%4d", pqueue[i].value);
}
```

SAMPLE OUTPUT

```
Choose your option
1.Enqueue 2. Dequeue 3. Exit
Enter your choice: 1

Enter the element to be inserted: 1
Enter the priority of the element: 1
   1
Choose your option
1.Enqueue 2. Dequeue 3. Exit
Enter your choice: 1

Enter the element to be inserted: 2
Enter the priority of the element: 1
   1   2
Choose your option
1.Enqueue 2. Dequeue 3. Exit
Enter your choice: 1

Enter the element to be inserted: 3
Enter the priority of the element: 3
   1   2   3
Choose your option
1.Enqueue 2. Dequeue 3. Exit
Enter your choice: 1
Enter the element to be inserted: 4
Enter the priority of the element: 2
   1   2   3   4
Choose your option
1.Enqueue 2. Dequeue 3. Exit
Enter your choice: 1

Enter the element to be inserted: 5
Enter the priority of the element: 3
   1   2   3   4   5
Choose your option
1.Enqueue 2. Dequeue 3. Exit
Enter your choice: 2
   1   2   4   5
Choose your option
1.Enqueue 2. Dequeue 3. Exit
Enter your choice: 2
   1   2   4
Choose your option
1.Enqueue 2. Dequeue 3. Exit
Enter your choice: 2
   1   2
Choose your option
1.Enqueue 2. Dequeue 3. Exit
Enter your choice: 3
```

RESULT

Priority Queue implemented successfully

Date:21-01-2022
Experiment No. 9

SEARCH FOR AN EMPLOYEE FROM N EMPLOYEES

AIM

To enter the details of n employees and search for an employee using employee id

ALGORITHM

1) START
2) Define a structure Employee containing the first name, middle name, last name, employee id and salary. Let p be a pointer to the structure
3) Input the number of employees and store it in n
4) Dynamically allocate memory to store the details for n employees
5) Input the details of each employees using the pointer to structure Employee p
6) Input the employee id to be searched and store it in emp
7) For i=0 to i=n-1,if (p+i)->EmpId = emp then print the employee name and salary and go to Step 9
8) If i=n then Print "Employee Not Found"
9) STOP

PROGRAM

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

typedef struct names // Defining a structure for names
{
    char First_name[20];
    char Middle_name[20];
    char Last_name[30];
} names;

typedef struct employee_details // Defining a structure for employees
{
    names Name;
    int Employee_ID;
    int Salary;
} employee_details;

int no_of_employees;
struct employee_details *details_ptr;

void Display(int i);
int search(int ID);

void main()
{
    int response, choice, employee_id;

    printf("Enter the number of employees : ");
    scanf("%d", &no_of_employees);

    details_ptr = (struct employee_details *)malloc(no_of_employees * sizeof(employee_de-
tails));
    if (details_ptr == NULL)
    {
        printf("Error!!! memory not allocated.(details_ptr)");
        exit(0);
    }

    printf("Please enter the employee details (Name Employee_ID Salary) : \n\n");
    for (int i = 0; i < no_of_employees; i++)
    {
        printf("Enter the name of employee no. %d (First_name Middle_name Last_name) :
\n", i + 1);
        scanf("%s %s %s", (details_ptr + i)->Name.First_name, (details_ptr + i)->Name.Mid-
dle_name, (details_ptr + i)->Name.Last_name);
        printf("Enter the Employee ID : ");
        scanf("%d", &(details_ptr + i)->Employee_ID);
        printf("Enter the Salary : ");
        scanf("%d", &(details_ptr + i)->Salary);
    }

    do
    {
        printf("\n\n\t M E N U\n\n");
        printf("1. Search\n2. Display details of all employees\n3. Exit\n");
    ch:
        printf("\n\t->");
        scanf("%d", &choice);
        printf("\n");

        switch (choice)
        {
        case 1:
            printf("To search the employee please enter the employee ID : ");
            scanf("%d", &employee_id);
            Display(search(employee_id));
            break;

        case 2:
            for (int i = 0; i < no_of_employees; i++)
            {
                printf("\nEmployee_ID. - %d  \nName - %s %s %s \tSalary - %d\n", (de-
tails_ptr + i)->Employee_ID, (details_ptr + i)->Name.First_name, (details_ptr + i)-
>Name.Middle_name, (details_ptr + i)->Name.Last_name, (details_ptr + i)->Salary);
            }
            break;

        case 3:
            exit(0);
            break;

        default:
            printf("\nEnter a valid Choice!!!\n");
            go to ch;
        }

        printf("\nDo you want to continue( 1 or 0 ) : ");
        scanf("%d", &response);

    } while (response == 1);
}

void Display(int i)
{
    if (i < 0)
    {
        printf("\nID not found!!!\n");
    }
    else
    {
        printf("\nEmployee_ID. - %d\nName - %s %s %s \tSalary - %d\n", (details_ptr + i)-
>Employee_ID, (details_ptr + i)->Name.First_name, (details_ptr + i)->Name.Middle_name,
(details_ptr + i)->Name.Last_name, (details_ptr + i)->Salary);
    }
}

int search(int ID)
{
    int flag, count = 0;
    for (int i = 0; i < no_of_employees; i++)
    {
        if ((details_ptr + i)->Employee_ID == ID)
        {
            flag = i;
            count++;
            break;
        }
    }
    if (count == 0)
    {
        return -1;
    }

    return flag;
}
```

SAMPLE OUTPUT

```
Enter the number of Employees: 3

Enter the first name of the Employee: Manu
Enter the middle name of the Employee: Rama
Enter the last name of the Employee: Krishna
Enter the Employee Id: 12234
Enter salary: 10000

Enter the first name of the Employee: Vishu
Enter the middle name of the Employee: Unni
Enter the last name of the Employee: Krishna
Enter the Employee Id: 45677
Enter salary: 12500

Enter the first name of the Employee: Sidha
Enter the middle name of the Employee: Bala
Enter the last name of the Employee: Krishnan
Enter the Employee Id: 37901
Enter salary: 19850

Enter the employee id to be searched: 45677

Employee Found!!!
Name of the Employee is: Vishu Unni Krishna
Salary of the Employee: 12500
```

RESULT

Details of employee entered, and Employee found successfully using Employee Id

Date: 21-01-2022
Experiment No.10

SORT AND SEARCH STRINGS FROM A FILE

AIM

To read strings from a file and sort it using Bubble Sort and search a string using Binary Search

Data Structure Used: Array

ALGORITHM

```
1) START
2) Set a file stream fp and open the required file using the fp file stream
3) Let i=0
4) While EndOfFile is not reached read each line of string and store each line
   in 2D array s and increment i by 1 when each line of string is read
5) Invoke the function BubbleSort (s,i)
6) Input the string to be searched and store it in string
7) Invoke the function BinarySearch (s, string, i)
8) Close the file open in fp stream
9) STOP
```

Definition of the function BubbleSort (char s[][20],int n)

```
1) START
2) For i=0 to i=n-2 repeat Step 3
3) For j=0 to j=n-i-1 repeat Step 4
4) If s[j] > s[j+1] then,
      i)  Copy s[j] to temp
      ii) Copy s[j+1] to s[j]
      iii)        Copy temp to s[j+1]
5) Print the contents of s
6) STOP
```

Definition of the function BinarySearch(char s[][20],string[20],int n)

```
1) START
2) Let l=0, u=n
3) While l<=u repeat Steps 4 to 7
4) Let mid = (l+u)/2
5) If s[mid]>string, then u = mid-1
6) If s[mid]<string, then l=mid+1
7) If s[mid] = string, then Print “String Found” and go to Step 9
8) If l>u then Print “String not Found”
9) STOP
```

PROGRAM

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

int first = 0, middle, last, no_of_words = 0;

void sort(char str[][100], int n)
{
    char temp[100];
    for (int i = 0; i < n; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (strcasecmp(str[i], str[j]) > 0)
            {
                strcpy(temp, str[i]);
                strcpy(str[i], str[j]);
                strcpy(str[j], temp);
            }
        }
    }
}

void binary_search(char key[100], char string[][100]);

void main()
{
    char string[10][100], a_string[100], key[100];

    FILE *file_ptr;
    file_ptr = fopen("file.txt", "r");

    if (file_ptr == NULL)
    {
        printf("\nThe file failed to open!!!\n\n");
        exit(0);
    }
    else
    {
        while (!feof(file_ptr))
        {
            fgets(a_string, 100, file_ptr);
            int len = strlen(a_string);
            if (no_of_words != 9)
            {
                a_string[len - 1] = '\0';
            }
            strcpy(string[no_of_words], a_string);
            no_of_words++;
        }
        fclose(file_ptr);
    }

    printf("Data inside the file : \n");
    for (int i = 0; i < 10; i++)
    {
        printf("%s\n", string[i]);
    }

    int response, choice;
    do
    {
        printf("\n\nM E N U\n\n1. Search\n2. Sort\n3. Exit\n\n");
    ch:
        printf("\t  -> ");
        scanf("%d", &choice);
        printf("\n\n");
        switch (choice)
        {
        case 1:
            printf("\nEnter the string to search : ");
            scanf("%s", key);
            binary_search(key, string);
            break;

        case 2:
            sort(string, no_of_words);
            for (int i = 0; i < 10; i++)
            {
                printf("%s ", string[i]);
                printf("\n");
            }
            break;

        case 3:
                exit(0);

            default:
                printf("Enter a valid Choice!!!!\n\n");
                go to ch;
            }
            printf("\nDo you want to continue (1 or 0) : ");
            scanf("%d", &response);
        } while (response == 1);
}

void binary_search(char key[100], char string[][100])
{
    last = no_of_words - 1;
    middle = (first + last) / 2;
    sort(string, no_of_words);

    while (first <= last)
    {
        if (strcasecmp(string[middle], key) < 0)
        {
            first = middle + 1;
        }
        else if (strcasecmp(string[middle], key) == 0)
        {
            printf("\n'%s' is found at location %d.\n", key, middle + 1);
            break;
        }
        else
        {
            last = middle - 1;
        }
        middle = (first + last) / 2;
    }

    if (first > last)
        printf("String not found!!! %s isn't present in the list.\n", key);
}
```

SAMPLE OUTPUT

```
The contents of the file are:
 Hello Friends
 Thank you, Friends,
 Hope You are well
 Have a nice day
 Take Care
 Good Bye
```

```
The sorted stings are:
 Good Bye
 Have a nice day
 Hello Friends
 Hope You are well
 Take Care
 Thank you Friends

Enter the string to be searched: Take Care

String found at the line 5
```

RESULT

Strings from the file read and sorted. Searched string found successfully

Date:21-01-2022
Experiment No. 11

IMPLEMENT LINKED LIST OPERATIONS

AIM

To perform insert, delete operations at the beginning, end and at specific positions of the linked list

Data Structure Used: Linked List

ALGORITHM

1) START
2) Create a structure with members data and a link of type pointer to the structure and define structure variables ptr, phead, new, ptr1 as pointers to the structure
3) A menu is provided to select the operations
    i) Display
    ii) Insert at the beginning
    iii) Insert at the end
    iv) Insert at a specific position
    v) Delete from the beginning
    vi) Delete from the end
    vii) Delete from a specific position
4) If user chooses i, then invoke the function display();
5) If user chooses ii, then input the element to be inserted as n and invoke the function insrtbeg(n)
6) If user chooses iii, then input the element to be inserted as n and invoke the function insrtend(n)
7) If user chooses iv, then input the element to be inserted as n and invoke the function insrtpos(n)
8) If user chooses v, then invoke the function delbeg()
9) If user chooses vi, then invoke the function delend()
10) If user chooses vii, then invoke the function delpos()

```
11)    If user chooses any other, go to Step 12
12)    STOP
```

Definition of the function display(int n)

```
1) START
2) Let ptr = head->link
3) While ptr!=NULL, repeat steps 3 and 4
4) Print ptr->data
5) Let ptr=ptr->link
6) STOP
```

Definition of function insrtbeg(int n)

```
1) START
2) Allocate a dynamic memory for the structure and store the pointer to new
   memory to new
3) If new != NULL, then
       i)  let new->data = n
       ii) let new->link = head->link
       iii)    head->link = new
4) STOP
```

Definition of the function insrtend(int n)

```
1) START
2) Allocate a dynamic memory for the structure and store the pointer to new
   memory to new
3) If new ==NULL, then go to Step 9
4) Let ptr = head
5) While ptr->link != NULL, ptr = ptr->link
6) Let ptr->link = new
7) Let new->data = n
8) Let new->link = NULL
9) STOP
```

Definition of the function insrtpos(int n)

```
1) START
2) Input the key element after which the insertion must take place and store it
   as key
3) Let ptr = head
4) While ptr->link != NULL && ptr->data != key, ptr = ptr->link
5) Allocate a dynamic memory for the structure and store the pointer to new
   memory to new
6) If new ==NULL, then go to Step10
7) Let new->data = n
8) Let new->link = ptr->link
9) Let ptr->link = new
10)     STOP
```

Definition of the function delbeg()

```
1) START
2) Let ptr = head
3) Let ptr = ptr->link
4) Let head->link = ptr->link
5) STOP
```

Definition of the function delend()

```
1) START
2) Let ptr = head
3) While ptr->link != NULL, repeat Steps 4 and 5
4) Let ptr1 = ptr
5) Let ptr = ptr->link
6) Let ptr1->link = NULL
7) STOP
```

Definition of the function delpos()

```
1) START
2) Input the element to be deleted as key
3) Let ptr = head
```

```
4) While ptr->link != NULL && ptr->data != key, repeat Steps 5 and 6
5) Let ptr1 = ptr
6) Let ptr = ptr->link
7) Let ptr1->link = ptr->link
8) STOP
```

PROGRAM

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

struct List
{
    int data;
    struct List *link;
} * head, *ptr, *new, *ptr1;

void display();
void insrtbeg(int n);
void insrtend(int n);
void insrtpos(int n);
void delbeg();
void delend();
void delpos();
void main()
{
    head = (struct List *)malloc(sizeof(struct List));
    head->data = NULL;
    head->link = NULL;
    int opt, inp;
    while (1)
    {
        printf("\n\nSelect your Choice");
        printf("\n1.Display\n2.Insert at the Beginning\n3.Insert at End\n4.Insert at a
specific position\n");
        printf("5.Delete from Beginning\n6.Delete from End\n7.Delete from a specific posi-
tion\n8.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &opt);
        switch (opt)
        {
        case 1:
            display();
            break;
        case 2:
            printf("\nEnter the number to be inserted: ");
            scanf("%d", &inp);
            insrtbeg(inp);
            break;
        case 3:
            printf("\nEnter the number to be inserted: ");
            scanf("%d", &inp);
            insrtend(inp);
            break;
        case 4:
            printf("\nEnter the number to be inserted: ");
            scanf("%d", &inp);
            insrtpos(inp);
            break;
        case 5:
            delbeg();
            break;
        case 6:
            delend();
            break;
        case 7:
            delpos();
        default:
            break;
        }
        if (opt > 7)
            break;
    }
    free(head);
    free(ptr);
    free(new);
    free(ptr1);
}
void display()
{
    if (head->link != NULL)
    {
        printf("The elements of the list: ");
        ptr = head->link;
        while (ptr != NULL)
        {
            printf("%4d", ptr->data);
            ptr = ptr->link;
        }
    }
    else
        printf("\n\nNo Elements to display!!\n\n");
}
void insrtbeg(int n)
{
    new = (struct List *)malloc(sizeof(struct List));
    if (new != NULL)
    {
        new->data = n;
        new->link = head->link;
        head->link = new;
    }
    else
        printf("\n\nNo Space Available!!\n\n");
}
void insrtend(int n)
{
    new = (struct List *)malloc(sizeof(struct List));
    if (new != NULL)
    {
        ptr = head;
        while (ptr->link != NULL)
            ptr = ptr->link;
        ptr->link = new;
        new->data = n;
        new->link = NULL;
    }
    else
        printf("\n\nNo Space Available!!\n\n");
}
void insrtpos(int n)
{
    int key;
    printf("Enter the key after which the element has to be inserted: ");
    scanf("%d", &key);
    ptr = head;
    while (ptr->link != NULL && ptr->data != key)
        ptr = ptr->link;
    if (ptr->link == NULL)
    {
        printf("\n\nKey Not Found!!\n\n");
        return;
    }
    new = (struct List *)malloc(sizeof(struct List));
    if (new != NULL)
    {
        new->data = n;
        new->link = ptr->link;
        ptr->link = new;
    }
    else
        printf("\n\nNo Space Available!!\n\n");
}
void delbeg()
{
    ptr = head;
    if (ptr->link == NULL)
    {
        printf("\n\nNo elements to delete!!\n\n");
        return;
    }
    ptr = ptr->link;
    head->link = ptr->link;
}
void delend()
{
    ptr = head;
    if (ptr->link == NULL)
    {
        printf("\n\nNo elements to delete!!\n\n");
        return;
    }
    while (ptr->link != NULL)
    {
        ptr1 = ptr;
        ptr = ptr->link;
    }
    ptr1->link = NULL;
}
void delpos()
{
    int key;
    printf("Enter the element to be deleted: ");
    scanf("%d", &key);
    ptr = head;
    while (ptr->link != NULL && ptr->data != key)
    {
        ptr1 = ptr;
        ptr = ptr->link;
    }
    if (ptr->link == NULL)
    {
        printf("\n\nKey Not Found!!\n\n");
        return;
    }
    ptr1->link = ptr->link;
}
```

SAMPLE OUTPUT

```
Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
```

```
Enter your choice: 2

Enter the number to be inserted: 1

Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
Enter your choice: 2

Enter the number to be inserted: 9

Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit

Enter your choice: 2
Enter the number to be inserted: 8

Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
```

```
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
Enter your choice: 1
The elements of the list:    8   9   1

Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
Enter your choice: 3

Enter the number to be inserted: 2

Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit

Enter your choice: 1
The elements of the list:    8   9   1   2

Select your Choice
1.Display
```

```
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
Enter your choice: 4

Enter the number to be inserted: 3
Enter the key after which the element has to be inserted: 1

Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
Enter your choice: 1
The elements of the list:    8   9   1   3   2

Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
Enter your choice: 5
```

```
Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
Enter your choice: 1
The elements of the list:    9   1   3   2

Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
Enter your choice: 6

Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit

Enter your choice: 1
The elements of the list:    9   1   3
```

```
Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
Enter your choice: 7
Enter the element to be deleted: 1

Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
Enter your choice: 1
The elements of the list:    9   3

Select your Choice
1.Display
2.Insert at the Beginning
3.Insert at End
4.Insert at a specific position
5.Delete from Beginning
6.Delete from End
7.Delete from a specific position
8.Exit
Enter your choice: 8
```

RESULT

Implemented insert, delete operations at the beginning, end and at specific positions of the linked list

Date:21-01-2022
Experiment No. 12

POLYNOMIAL OPERATIONS USING LINKED LIST

AIM
To do polynomial addition and multiplication using linked list

Data Structure Used: Linked List

ALGORITHM

```
1) START
2) Create a structure with members coeff, exp and a link of type pointer to the
   structure and define structure variables Pptr, Qptr, Rptr, Phead, Qhead,
   Rhead, *Aptr, Ahead, Rptr1 as pointer to the structure
3) Allocate dynamic memory of size of the structure for Phead, Qhead, Rhead
   and Ahead
4) Input the degree of first and second polynomial as deg1 and deg2
5) For i=deg1 to i>=0, repeat Steps 6 to 9
6) Allocate a dynamic memory for the structure and store the pointer to new
   memory to Pptr->link
7) Input the coefficient and store it in Pptr->coeff
8) Let Pptr->exp = i
9) Let Pptr->link = NULL
10) For i=deg2 to i>=0, repeat Steps 11 to 14
11) Allocate a dynamic memory for the structure and store the pointer to
    new memory to Qptr->link
12) Input the coefficient and store it in Qptr->coeff
13) Let Qptr->exp = i
14) Let Qptr->link = NULL
15) Call the function Multiply()
16) Call the function addition()
17) STOP
```

Definition of the function Multiply()

```
1) START
2) Let Pptr = Phead->link
3) While Pptr!= NULL, repeat Steps 4 to 25
4) Let Qptr = Qhead->link
5) While Qptr != NULL, repeat Steps 6 to 24
6) Let prod = Pptr->coef * Qptr->coef and expon = Pptr->exp + Qptr->exp
7) If Rhead->link != NULL, then go to Step 13
8) Allocate a dynamic memory for the structure and store the pointer to new
   memory to Rhead->link
9) Let Rptr = Rhead->link
10)      Let Rptr->coef = prod
11)      Let Rptr->exp = expon
12)      Let Rptr->link = NULL
13)      Let Rptr = Rhead->link
14)      While Rptr!=NULL, repeat Steps 15 and 16
15)      If Rptr->exp == expon, then Rptr->coef += prod and go to Step 17
16)      Let Rptr1 = Rptr and Rptr = Rptr->link
17)      If Rptr !=NULL then go to Step 23
18)      Allocate a dynamic memory for the structure and store the pointer to
   new memory to Rptr1->link
19)      Let Rptr1 = Rptr1->link;
20)      Let Rptr1->coef = prod;
21)      Let Rptr1->exp = expon;
22)      Let Rptr1->link = NULL
23)      Let Qptr = Qptr->link
24)      Let Pptr = Pptr->link
25)      Let Rptr = Rhead->link
26)      While Rptr != NULL, Print Rptr->coef ,"x^", Rptr->exp,"+"
27)      STOP
```

Definition of the function addition()

```
1) START
2) Let Pptr = Phead->link and Qptr = Qhead->link
3) While Pptr != NULL && Qptr != NULL, repeat Step 4
```

```
4) If Pptr->exp == Qptr->exp or Pptr->exp <Qptr, then go to Step 11
5) Allocate a dynamic memory for the structure and store the pointer to new
   memory to Aptr->link
6) Let Aptr = Aptr->link
7) Let Aptr->coef = Pptr->coef
8) Let Aptr->exp = Pptr->exp
9) Aptr->link = NULL
10)     Let Pptr = Pptr->link
11)     If Pptr->exp < Qptr->exp, then go to Step 19
12)     Allocate a dynamic memory for the structure and store the pointer to
   new memory to Aptr->link
13)     Let Aptr = Aptr->link
14)     Let Aptr->coef = Pptr->coef + Qptr->coef
15)     Let Aptr->exp = Pptr->exp
16)     Let Aptr->link = NULL
17)     Let Pptr = Pptr->link
18)     Let Qptr = Qptr->link
19)     Allocate a dynamic memory for the structure and store the pointer to
   new memory to Aptr->link
20)     Let Aptr = Aptr->link
21)     Let Aptr->coef = Qptr->coef
22)     Let Aptr->exp = Qptr->exp
23)     Let Aptr->link = NULL
24)     Let Qptr = Qptr->link
25)     Let Pptr = Phead->link
26)     Let Qptr = Qhead->link
27)     Let Aptr = Ahead->link
28)     While Pptr != NULL, Print Pptr->coef, ”x^”, Pptr->exp,”+”
29)     While Qptr != NULL, Print Qptr->coef ,”x^”, Qptr->exp,”+”
30)     While Aptr != NULL, Print Aptr->coef, ”x^”, Aptr->exp,”+”
31)     STOP
```

PROGRAM

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

```
struct Polynomial
{
    int coef;
    int exp;
    struct Polynomial *link;
} * Pptr, *Qptr, *Rptr, *Phead, *Qhead, *Rhead, *Aptr, *Ahead, *Rptr1;

void Multiply();
void addition();
void main()
{
    int i, deg1, deg2;
    Phead = (struct Polynomial *)malloc(sizeof(struct Polynomial));
    Qhead = (struct Polynomial *)malloc(sizeof(struct Polynomial));
    Rhead = (struct Polynomial *)malloc(sizeof(struct Polynomial));
    Ahead = (struct Polynomial *)malloc(sizeof(struct Polynomial));
    Phead->link = NULL;
    Pptr = Phead;
    Qhead->link = NULL;
    Qptr = Qhead;
    Rhead->link = NULL;
    Rptr = Rhead;
    Ahead->link = NULL;
    Aptr = Ahead;
    printf("Enter the degree of the first polynomial: ");
    scanf("%d", &deg1);
    printf("Enter the degree of the second polynomial: ");
    scanf("%d", &deg2);
    printf("Enter the fisrt polynomial\n");
    for (i = deg1; i >= 0; i--)
    {
        Pptr->link = (struct Polynomial *)malloc(sizeof(struct Polynomial));
        Pptr = Pptr->link;
        printf("Enter the coefficient for exponent %d: ", i);
        scanf("%d", &Pptr->coef);
        Pptr->exp = i;
        Pptr->link = NULL;
    }
    printf("Enter the second polynomial\n");
    for (i = deg2; i >= 0; i--)
    {
        Qptr->link = (struct Polynomial *)malloc(sizeof(struct Polynomial));
        Qptr = Qptr->link;
        printf("Enter the coefficient for exponent %d: ", i);
        scanf("%d", &Qptr->coef);
        Qptr->exp = i;
        Qptr->link = NULL;
    }
    Multiply();
    addition();
}
```

```c
void Multiply()
{
    int prod, expon;
    Pptr = Phead->link;
    while (Pptr != NULL)
    {
        Qptr = Qhead->link;
        while (Qptr != NULL)
        {
            prod = Pptr->coef * Qptr->coef;
            expon = Pptr->exp + Qptr->exp;
            if (Rhead->link == NULL)
            {
                Rhead->link = (struct Polynomial *)malloc(sizeof(struct Polynomial));
                Rptr = Rhead->link;
                Rptr->coef = prod;
                Rptr->exp = expon;
                Rptr->link = NULL;
            }
            else
            {
                Rptr = Rhead->link;
                while (Rptr != NULL)
                {
                    if (Rptr->exp == expon)
                    {
                        Rptr->coef += prod;
                        break;
                    }
                    Rptr1 = Rptr;
                    Rptr = Rptr->link;
                }
                if (Rptr == NULL)
                {
                    Rptr1->link = (struct Polynomial *)malloc(sizeof(struct Polynomial));
                    Rptr1 = Rptr1->link;
                    Rptr1->coef = prod;
                    Rptr1->exp = expon;
                    Rptr1->link = NULL;
                }
            }
            Qptr = Qptr->link;
        }
        Pptr = Pptr->link;
    }
    Rptr = Rhead->link;
    printf("\nThe product of the matrix: ");
    while (Rptr->link != NULL)
    {
        printf("%dx^%d+", Rptr->coef, Rptr->exp);
        Rptr = Rptr->link;
    }
    printf("%dx^%d", Rptr->coef, Rptr->exp);
    printf("\nHello\n");
}
void addition()
{
    Pptr = Phead->link;
    Qptr = Qhead->link;
    while (Pptr != NULL && Qptr != NULL)
    {
        if (Pptr->exp > Qptr->exp)
        {
            Aptr->link = (struct Polynomial *)malloc(sizeof(struct Polynomial));
            Aptr = Aptr->link;
            Aptr->coef = Pptr->coef;
            Aptr->exp = Pptr->exp;
            Aptr->link = NULL;
            Pptr = Pptr->link;
        }
        else if (Pptr->exp == Qptr->exp)
        {
            Aptr->link = (struct Polynomial *)malloc(sizeof(struct Polynomial));
            Aptr = Aptr->link;
            Aptr->coef = Pptr->coef + Qptr->coef;
            Aptr->exp = Pptr->exp;
            Aptr->link = NULL;
            Pptr = Pptr->link;
            Qptr = Qptr->link;
        }

        else if (Pptr->exp < Qptr->exp)
        {
            Aptr->link = (struct Polynomial *)malloc(sizeof(struct Polynomial));
            Aptr = Aptr->link;
            Aptr->coef = Qptr->coef;
            Aptr->exp = Qptr->exp;
            Aptr->link = NULL;
            Qptr = Qptr->link;
        }
    }
    Pptr = Phead->link;
    Qptr = Qhead->link;
    Aptr = Ahead->link;
    printf("\nThe first polynomial is: ");
    while (Pptr->link != NULL)
    {
        printf("%dx^%d+", Pptr->coef, Pptr->exp);
        Pptr = Pptr->link;
    }
    printf("%dx^%d", Pptr->coef, Pptr->exp);
    printf("\nThe Second polynomial is: ");
    while (Qptr->link != NULL)
    {
        printf("%dx^%d+", Qptr->coef, Qptr->exp);
        Qptr = Qptr->link;
    }
    printf("%dx^%d", Qptr->coef, Qptr->exp);
    printf("\nThe sum of the polynomials is: ");
    while (Aptr->link != NULL)
    {
        printf("%dx^%d+", Aptr->coef, Aptr->exp);
        Aptr = Aptr->link;
    }
    printf("%dx^%d", Aptr->coef, Aptr->exp);
}
```

SAMPLE OUTPUT

```
Enter the degree of the first polynomial: 3
Enter the degree of the second polynomial: 2
Enter the first polynomial
Enter the coefficient for exponent 3: 8
Enter the coefficient for exponent 2: 0
Enter the coefficient for exponent 1: 6
Enter the coefficient for exponent 0: 1
Enter the second polynomial
Enter the coefficient for exponent 2: 7
Enter the coefficient for exponent 1: 2
Enter the coefficient for exponent 0: 0

The product of the matrix: 56x^5+16x^4+42x^3+19x^2+2x^1+0x^0

The first polynomial is: 8x^3+0x^2+6x^1+1x^0
The Second polynomial is: 7x^2+2x^1+0x^0
The sum of the polynomials is: 8x^3+7x^2+8x^1+1x^0
```

RESULT

Polynomial addition and multiplication implemented using linked list

Date:28-01-2022
Experiment No. 13

IMPLEMENT STACK USING LINKED LIST

AIM

To perform push, pop operations in a stack using linked list

Data Structure Used: Linked List

ALGORITHM

```
1) START
2) Create a structure with members data and a link of type pointer to the
   structure and define structure variables head, top as pointers to the structure
3) A menu is provided to select the operations
      i)  Push
      ii) Pop
4) If user chooses i, then input the element to be inserted as n and invoke the
   function push(n) and display()
5) If user chooses ii, then invoke the function pop() and display()
6) If user chooses any other, go to Step 7
7) STOP
```

Definition of function push(int n)

```
1) START
2) Dynamically allocate memory of size of the structure and store its address
   in top
3) If top=NULL, then go to Step 7
4) Let top->data = n
5) Let top->link = head->link
6) Let head->link = top
7) STOP
```

Definition of the function pop()

```
1) START
2) If head->link = NULL, then go to Step 6
3) Let top = head->link
4) Let top = top->link
5) Let head->link = top
6) STOP
```

Definition of the function display()

```
1) START
2) Let top = head->link
3) If head->link = NULL then go to Step 7
4) While top!=NULL, repeat Steps 5 and 6
5) Print top->data
6) Let top = top->link
7) STOP
```

PROGRAM

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

struct Stack
{
    int data;
    struct Stack *link;
} * head, *top;

void push(int n);
void pop();
void display();
void main()
{
    int opt, inp;
    head = (struct Stack *)malloc(sizeof(struct Stack));
    head->link = NULL;
    while (1)
    {
        printf("\n\nChoose your option\n");
        printf("\n1.Push an element\n2.Pop an element\n3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &opt);
        switch (opt)
        {
```

```
        case 1:
            printf("Enter the element to be pushed: ");
            scanf("%d", &inp);
            push(inp);
            display();
            break;
        case 2:
            pop();
            display();
            break;
        default:
            return;
        }
    }
}
void push(int n)
{
    top = (struct Stack *)malloc(sizeof(struct Stack));
    if (top != NULL)
    {
        top->data = n;
        top->link = head->link;
        head->link = top;
    }
    else
        printf("No space available!!!");
}
void pop()
{
    if (head->link != NULL)
    {
        top = head->link;
        top = top->link;
        head->link = top;
    }
}
void display()
{
    top = head->link;
    if (head->link == NULL)
        printf("\nStack Underflow!!\n");
    else
    {
        printf("\nThe elements of the stack: ");
        while (top != NULL)
        {
            printf("%4d", top->data);
            top = top->link;
        }
    }
    printf("\n");
}
```

SAMPLE OUTPUT

```
Choose your option
1.Push an element
2.Pop an element
3.Exit
Enter your choice: 1
Enter the element to be pushed: 1

The elements of the stack:    1

Choose your option
1.Push an element
2.Pop an element
3.Exit
Enter your choice: 1
Enter the element to be pushed: 2

The elements of the stack:    2   1

Choose your option
1.Push an element
2.Pop an element
3.Exit
Enter your choice: 1
Enter the element to be pushed: 3

The elements of the stack:    3   2   1

Choose your option
1.Push an element
2.Pop an element
3.Exit
Enter your choice: 2
```

```
The elements of the stack:    2   1

Choose your option
1.Push an element
2.Pop an element
3.Exit
Enter your choice: 1
Enter the element to be pushed: 4

The elements of the stack:    4   2   1

Choose your option
1.Push an element
2.Pop an element
3.Exit
Enter your choice: 2

The elements of the stack:    2   1

Choose your option
1.Push an element
2.Pop an element
3.Exit
Enter your choice: 2

The elements of the stack:    1

Choose your option
1.Push an element
2.Pop an element
3.Exit
Enter your choice: 2

Stack Underflow!!
```

```
Choose your option
1.Push an element
2.Pop an element
3.Exit
Enter your choice: 3
```

RESULT

Stack Operations implemented using linked list

Date:28-01-2022
Experiment No. 14

IMPLEMENT QUEUE USING LINKED LIST

AIM
To implement queue operations using linked list

Data Structure Used: Linked List

ALGORITHM

```
1) START
2) Create a structure with members data and a Rlink and Llink of type pointer to
   the structure and define structure variables head, front, rear, temp as pointers
   to the structure
3) A menu is provided to select the operations
      i)  Enqueue
      ii) Dequeue
4) If user chooses i, then input the element to be inserted as n and invoke the
   function enqueue(n) and display()
5) If user chooses ii, then invoke the function dequeue() and display()
6) If user chooses any other, go to Step 7
7) STOP
```

Definition of function enqueue(int n)

```
   1) START
   2) Let temp = head
   3) While temp->Rlink !=NULL, then let temp = temp->Rlink
   4) Dynamically allocate memory of size of the structure and store its address
      to rear
   5) Let rear->data = n
   6) Let rear->Rlink = NULL
   7) Let rear->Llink = temp
   8) Let temp->Rlink = rear
```

```
9) If front = NULL then front = head->Rlink
10)     STOP
```

Definition of function dequeue()

```
1) START
2) If head->Rlink = NULL then go to Step6
3) Let front = front->Rlink
4) If front!=NULL then front->Llink = head
5) Let head->Rlink = front
6) STOP
```

Definition of the function display()

```
1) START
2) If head->Rlink = NULL, then go to Step 7
3) Let temp = head->Rlink
4) While temp != NULL, repeat Steps 5 and 6
5) Print temp->data
6) Let temp = temp->Rlink
7) STOP
```

PROGRAM

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

struct Queue
{
    int data;
    struct Queue *Rlink;
    struct Queue *Llink;
} * head, *front, *rear, *temp;

void enqueue(int n);
void dequeue();
void display();
void main()
{
    int opt, inp;
    head = (struct Queue *)malloc(sizeof(struct Queue));
    head->Rlink = NULL;
    head->Llink = NULL;
    while (1)
    {
        printf("\n\nChoose your option\n");
        printf("\n1.Enqueue an element\n2.Dequeue an element\n3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &opt);
        switch (opt)
        {
        case 1:
            printf("Enter the element to be pushed: ");
            scanf("%d", &inp);
            enqueue(inp);
            display();
            break;
        case 2:
            dequeue();
            display();
            break;
        default:
            return;
        }
    }
}
void enqueue(int n)
{
    temp = head;
    while (temp->Rlink != NULL)
        temp = temp->Rlink;
    rear = (struct Queue *)malloc(sizeof(struct Queue));
    rear->data = n;
    rear->Rlink = NULL;
    rear->Llink = temp;
    temp->Rlink = rear;
    if (front == NULL)
        front = head->Rlink;
}
void dequeue()
{
    if (head->Rlink != NULL)
    {
        front = front->Rlink;
        if (front != NULL)
            front->Llink = head;
        head->Rlink = front;
    }
}
void display()
{
    if (head->Rlink == NULL)
        printf("\nQueue Empty!!!\b");
    else
    {
        printf("\nElements of the queue are: ");
        temp = head->Rlink;
        while (temp != NULL)
        {
            printf("%4d", temp->data);
            temp = temp->Rlink;
        }
    }
    printf("\n");
}
```

SAMPLE OUTPUT

```
Choose your option

1.Enqueue an element
2.Dequeue an element
3.Exit
Enter your choice: 1
Enter the element to be pushed: 1

Elements of the queue are:    1

Choose your option

1.Enqueue an element
2.Dequeue an element
3.Exit
Enter your choice: 1
Enter the element to be pushed: 2

Elements of the queue are:    1   2

Choose your option

1.Enqueue an element
2.Dequeue an element
3.Exit
Enter your choice: 1
Enter the element to be pushed: 3
```

```
Elements of the queue are:    1   2   3

Choose your option

1.Enqueue an element
2.Dequeue an element
3.Exit
Enter your choice: 2

Elements of the queue are:    2   3

Choose your option

1.Enqueue an element
2.Dequeue an element
3.Exit
Enter your choice: 1
Enter the element to be pushed: 4

Elements of the queue are:    2   3   4

Choose your option

1.Enqueue an element
2.Dequeue an element
3.Exit
Enter your choice: 2

Elements of the queue are:    3   4

Choose your option

1.Enqueue an element
2.Dequeue an element
```

```
3.Exit
Enter your choice: 2

Elements of the queue are:    4


Choose your option

1.Enqueue an element
2.Dequeue an element
3.Exit
Enter your choice: 2

Queue Empty!!!

Choose your option

1.Enqueue an element
2.Dequeue an element
3.Exit
Enter your choice: 3
```

RESULT

Queue operations implemented using linked list

Date: 17-02-2022
Experiment No. 15

IMPLEMENTATION OF TREE DATA STRUCTURE

AIM

To implement tree data structure

Data Structure Used: Linked List

ALGORITHM

```
1) START
2) Create a linked list with each node having a data part of type int, and
   having a right child and left child LC and RC
3) In the main method, call the function buildtree()
4) Provide a menu to the user
   i)  Insert an element
   ii) PreOrder Traversal
   iii)    InOrder Traversal
   iv)PostOder Traversal
   v) Deletion
5) If user chooses i, input the number to be inserted as inp and input key and
   store the result of searchaddr(root, key) in ptr and invoke the function
   insrt(inp, key, ptr)
6) If user chooses ii, invoke the function preordertrav(root)
7) If user chooses iii, invoke the function inordertrav(root)
8) If user chooses iv, invoke the function postordertrav(root)
9) If user chooses v, invoke the function delete(key)
10)      For any other options, go to Step 11
11)      STOP
```

Definition of function buildtree(struct node* ptr)

```
1) START
2) If ptr=NULL, then go to Step 16
```

```
3) Input the data of a node as ptr->data
4) If this node do not have a left child go to Step 9
5) Dynamically allocate a memory of size of the node and store it in temp
6) Let ptr->LC=temp
7) Invoke the function buildtree(temp) again as recursion
8) Let ptr->LC=NULL
9) Invoke the function buildtree(ptr->RC) again as recursion
10)     If this node do not have a right child go to Step 14
11)     Dynamically allocate a memory of size of the node and store it in
    temp
12)     Let ptr->RC=temp
13)     Invoke the function buildtree(temp) again as recursion
14)     Let ptr->RC=NULL
15)     Invoke the function buildtree(ptr->RC) again as recursion
16)     STOP
```

Definition of function searchaddr(struct node* ptr, int key)

```
1) START
2) If ptr->data = key then return ptr
3) If ptr->LC = NULL, then return NULL
4) Invoke searchaddr(ptr->LC, key) again as recursion and store the return
   value in k
5) If k!=NULL, then go to Step 8
6) If ptr->RC=NULL, return NULL
7) Invoke searchaddr(ptr->RC, key) again as recursion
8) STOP
```

Definition of the function insrt(int n, int key ,struct node* ptr)

```
1) START
2) If ptr=NULL, return NULL
3) Dynamically allocate a memory of size of the node and store it in temp
4) If user want to insert as right child go to Step 6
5) If ptr->LC !=NULL,
      i)  Let ptr->LC = temp
      ii) Let temp->data = n
```

```
        iii)    Let temp->LC = NULL
        iv)Let temp->RC = NULL
    6) If ptr->RC !=NULL,
        i)  Let ptr->RC = temp
        ii) Let temp->data = n
        iii)    Let temp->RC = NULL
        iv)Let temp->LC = NULL
    7) STOP
```

Definition of function searchpar(struct node* ptr, struct node* prev, int key)

```
    1) START
    2) If ptr=NULL, then return NULL
    3) If ptr->data = key
        i)  Let k=prev
        ii) Return prev
    4) If ptr->data!=key
        i)  Invoke function searchpar(ptr->LC, ptr, key) as recursion
        ii) Invoke function searchpar(ptr->RC, ptr, key) as recursion
    5) Return k
    6) STOP
```

Definition of the function delete(int key)

```
    1) START
    2) If root=NULL, return
    3) Invoke function searchpar(root, NULL, key) as recursion and store the
       result in parent
    4) If parent !=NULL then go to Step 5
    5) Let ptrl = parent->LC and ptr = parent->RC
    6) If ptrl=NULL then go to Step 9
    7) If ptr->data = key, then go to Step 8
    8) If ptrl->LC = NULL and ptrl->RC=NULL, then let parent->LC = NULL
       else return
    9) If ptr->data = key, then go to Step 10
```

```
10)    If ptrl->LC = NULL and ptrl->RC=NULL, then let parent->RC =
       NULL else return
11)    STOP
```

Definition of function preordertrav(struct node* ptr)

```
1) START
2) Print ptr->data
3) Invoke function preordertrav(ptr->LC) as recursion
4) Invoke function preordertrav(ptr->RC) as recursion
5) STOP
```

Definition of function inordertrav(struct node* ptr)

```
1) START
2) Invoke function inordertrav(ptr->LC) as recursion
3) Print ptr->data
4) Invoke function inordertrav(ptr->RC) as recursion
5) STOP
```

Definition of function postordertrav(struct node* ptr)

```
1) START
2) Invoke function postordertrav(ptr->LC) as recursion
3) Invoke function postordertrav(ptr->RC) as recursion
4) Print ptr->data
5) STOP
```

PROGRAM

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

struct node
{
    int value;
    struct node *left;
    struct node *right;
};

struct node *root = NULL;
struct node *createNode(int data)
{
    struct node *newNode = malloc(sizeof(struct node));

    newNode->value = data;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}

// Creating the tree
struct node *createTree()
{
    int x;
    struct node *newnode = malloc(sizeof(struct node));

    printf("\nEnter the value(-1 for no node): ");
    scanf("%d", &x);

    if (x == -1)
    {
        return NULL;
    }
    newnode->value = x;
    printf("\nEnter the left child of %d: ", x);
    newnode->left = createTree();
    printf("\nEnter the right child of %d: ", x);
    newnode->right = createTree();

    return newnode;
}

// Inorder Traversal
void inorderTraversal(struct node *root)
{
    if (root == NULL)
        return;
    inorderTraversal(root->left);
    printf("%d   ", root->value);
    inorderTraversal(root->right);
}

// Preorder Traversal
void preorderTraversal(struct node *root)
{
    if (root == NULL)
        return;
    printf("%d   ", root->value);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}
```

```
// Postorder Traversal
void postorderTraversal(struct node *root)
{
    if (root == NULL)
        return;
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d   ", root->value);
}

// Search Insertion
struct node *search_ins(struct node *ptr, int data)
{
    struct node *temp;
    temp = ptr;
    if (temp == NULL)
        return NULL;

    if (temp->value != data)
    {
        if (temp->left != NULL)
        {
            search_ins(temp->left, data);
        }
        if (temp->right != NULL)
        {
            search_ins(temp->right, data);
        }
        if (temp->left == NULL && temp->right == NULL)
        {
            return NULL;
        }

    }
    else
    {
        return temp;
    }
}

// Insertion
void Insertion( int key, int data){
    struct node *ptr = search_ins(root, key);
    char option;

    if (ptr == NULL)
    {
        printf("Search Unsuccessfull : No Insertion");
        return;
    }
    if (ptr->left == NULL || ptr->right == NULL)
    {
        struct node *new = createNode(data);
    lh:
        printf("\nDo you want to insert the node as the left child(L) or right child(R) :
");
        scanf("%c", &option);
        if (option == 'L' || option == 'l')
        {
            if (ptr->left == NULL)
            {
                ptr->left = new;
            }
            else
            {
                printf("\nInsertion not possible as left child!!!\n");
                return;
            }
        }
        else if (option == 'R' || option == 'r'){
            if (ptr->right == NULL)
            {
                ptr->right = new;
            }
            else
            {
                printf("\nInsertion not possible as right child!!!\n");
            }
        }
        else
        {
            printf("\nEnter a valid choice!!!\n");
            go to lh;
        }

    }
    else
    {
        printf("\nThe key node already has both children!!!\n");
    }
}

// Search for parent node
struct node *search_parent(struct node *ptr, int data)
{
    struct node *parent;
    parent = ptr;
    if (parent == NULL)
        return NULL;

    struct node *temp1 = ptr->left, *temp2 = ptr->right;

    if (temp1->value != data && temp2->value != data)
    {

        if (temp1->left != NULL)
        {
            search_parent(temp1, data);
        }
        if (temp1->right != NULL)
        {
            search_parent(temp1, data);
        }
        if (temp2->left != NULL)
        {
            search_parent(temp2, data);
        }
        if (temp2->right != NULL)
        {
            search_parent(temp2, data);
        }
    }
    else
    {
        return parent;
    }
}

// Deletion
void deletion(int data){
    struct node *ptr = root;
    if (ptr == NULL)
    {
        printf("\nTree is Empty!!!\n");
        return;
    }
    struct node *parent = search_parent(root, data);
    // printf("\n %d", parent->value);
    if (parent != NULL)
    {
        struct node *temp1 = ptr->left, *temp2 = ptr->right;
        if (temp1->value == data)
        {
            if (temp1->left == NULL && temp1->right == NULL)
            {
                parent->left = NULL;
            }
            else
            {
                printf("\nNode is not a leaf node : No deletion!!!\n");
            }

        }
        else if (temp2->value == data)
        {
            if (temp2->left == NULL && temp2->right == NULL)
            {
                parent->right = NULL;
            }
            else
            {
                printf("\nNode is not a leaf node : No deletion!!!\n");
            }
        }

    }
    else
    {
        printf("\nNode with data %d does not exist : Deletion failed!!!\n", data);
    }
}

void main()
{
    root = createTree();

    int response = 1, choice, element, data, node;
    do
    {
        printf("\n\nM E N U\n\n1. Insert a node to the tree\n2. Delete a node to the
tree\n3. Preorder Traversal\n4. Inorder Traversal");
        printf("\n5. Postorder Traversal\n6. Exit\n\n");
    ch:
        printf("\t  -> ");
        scanf("%d", &choice);
        printf("\n\n");
        switch (choice)
        {
        case 1:
            printf("Enter the number to be inserted : ");
            scanf("%d", &data);
            printf("Enter the node after which you need to insert : ");
            scanf("%d", &node);
            Insertion(node, data);
            break;

        case 2:
            printf("Enter the node which needs to be deleted : ");
            scanf("%d", &node);
            deletion(node);
            break;

        case 3:
            preorderTraversal(root);
            break;

        case 4:
            inorderTraversal(root);
            break;

        case 5:
            postorderTraversal(root);
            break;

        case 6:
            exit(0);

        default:
            printf("Enter a valid Choice!!!!\n\n");
            go to ch;
        }
    } while (response == 1);
}
```

SAMPLE OUTPUT

```
Enter the value(-1 for no node): 12

Enter the left child of 12:
Enter the value(-1 for no node): 13

Enter the left child of 13:
Enter the value(-1 for no node): 5

Enter the left child of 5:
Enter the value(-1 for no node): -1

Enter the right child of 5:
Enter the value(-1 for no node): -1

Enter the right child of 13:
Enter the value(-1 for no node): 6

Enter the left child of 6:
Enter the value(-1 for no node): -1

Enter the right child of 6:
```

```
Enter the value(-1 for no node): -1

Enter the right child of 12:
Enter the value(-1 for no node): 23

Enter the left child of 23:
Enter the value(-1 for no node): 14

Enter the left child of 14:
Enter the value(-1 for no node): -1

Enter the right child of 14:
Enter the value(-1 for no node): -1

Enter the right child of 23:
Enter the value(-1 for no node): 9

Enter the left child of 9:
Enter the value(-1 for no node): -1

Enter the right child of 9:
Enter the value(-1 for no node): -1


M E N U

1. Insert a node to the tree
2. Delete a node to the tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Exit

        -> 1
```

```
Enter the number to be inserted : 4
Enter the node after which you need to insert : 9
Do you want to insert the node as the left child(L) or right child(R) : L


M E N U

1. Insert a node to the tree
2. Delete a node to the tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Exit

        -> 2


Enter the node which needs to be deleted : 4


M E N U

1. Insert a node to the tree
2. Delete a node to the tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Exit

        -> 3


12   13   5   6   23   14   9
```

```
M E N U

1. Insert a node to the tree
2. Delete a node to the tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Exit

          -> 4



5   13   6   12   14   23   9

M E N U

1. Insert a node to the tree
2. Delete a node to the tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Exit

          -> 5



5   6   13   14  9   23   12

M E N U

1. Insert a node to the tree
2. Delete a node to the tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
```

```
6. Exit

        -> 6
```

RESULT

Tree data structure implemented

Date: 04-03-2022
Experiment No. 16

IMPLEMENTATION OF GRAPH DATA STRUCTURE

AIM

To implement graph data structure

Data Structure Used: Linked List

ALGORITHM

1) START
2) Create a node list for the graph using linked list with each node has a data part, one part pointing to the adjacent node in Edgelist and another part pointing to the next vertex of the node list with start, vrtx, ptr1, ptr2 as pointers of this type
3) Create an edgelist for the graph using linked list with each node having one part pointing to a node in nodelist and another part pointing to the next adjacent node in edgelist with adjvrtx as pointer of this type
4) Create two linked lists of same type visit with each node containing a data part and link part with a head nodes heada and headb as head nodes and ptra and ptrb as other nodes
5) Create a Stack stack and Queue queue and let mat[10][10] be 2D matrix
6) In the main method and input the all the values of vertices of the graph and store them in an array A[] and let the size of the array be i
7) Invoke the function buildmat(i)
8) For each elements in array A, if the element has an adjacent element input each adjacent element as adj and invoke the function adjmat(A, A[j], adj, i) and adjlist(A[j],adj)
9) A menu is provided for the user
    i) Adjacent Matrix Representation
    ii) Adjacent List Representation
    iii) Depth First Search
    iv) Breadth First Search

```
10)     If user choose i, invoke dispmat(i)
11)     If user choose ii, invoke displist()
12)     If user choose iii, invoke dfs(i) and display the elements of the linked
        list visit1
13)     If user choose iv, invoke bfs(i) and display the elements of the linked
        list visit2
14)     STOP
```

Definition of the function buildlist(char curr)

```
1) START
2) Dynamically allocate a memory of size of the node in nodelist and store it
   in   vrtx->nxtvtx
3) Let vrtx = vrtx->nxtvtx
4) Let vrtx->data = curr
5) Let vrtx->adjnode=NULL and vrtx->nxtvtx=NULL
6) NULL
```

Definition of the function buildmat(int m)

```
1) START
2) For i=0 to m repeat Step 3
3) For j=0 to m mat[i][j] = 0
4) STOP
```

Definition of function adjmat(char A[10],char curr, char adj, int i)

```
1) START
2) For k=0 to k<i repeat Steps 3 and 4
3) If A[k] = curr then let indc=k
4) If A[k]==adj then let indad=k
5) Increment mat[indc][indad] by 1
6) STOP
```

Definition of function adjlist(char curr, char adj)

```
1) START
2) Let vrtx=start->nxtvtx
3) Let curr in the nodelist be ptr1 and adj in nodelist be ptr2
```

```
4) Let vrtx=vrtx->nxtvtx
5) Let adjvrtx=ptr1->adjnode
6) If adjvrtx==NULL, then dynamically allocate a memory of size of the node
   in edgelist and store it in ptr1->adjnode and let adjvrtx=ptr1->adjnode and
   go to Step 11
7) While adjvrtx->adjlink!=NULL, let adjvrtx=adjvrtx->adjlink
8) Dynamically allocate a memory of size of the node in edgelist and store it
   in adjvrtx->adjlink and let adjvrtx = adjvrtx->adjlink
9) Let adjvrtx->vlink = ptr2 and adjvrtx->adjlink=NULL
10)      STOP
```

Definition of the function dfs()

```
1) START
2) Push the first element of nodelist to stack
3) While stack is not empty repeat Steps 5 to 8
4) Pop from the stack and store it in ch
5) If ch is present in visit 1 go to Step 5
6) Insert ch to visit1
7) Push the adjacent elements of ch to the stack
8) STOP
```

Definition of the function bfs()

```
1) START
2) Enqueue the first element of nodelist to queue
3) While queue is not empty repeat Steps 5 to 8
4) Dequeue from the queue and store it in ch
5) If ch is present in visit 2 go to Step 5
6) Insert ch to visit1
7) Dequeue the adjacent elements of ch to the queue
8) STOP
```

PROGRAM

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

struct AdjList
{
    struct NodeGraph *dest;
    struct AdjList *link;
};
struct NodeGraph
{
    int data;
    struct NodeGraph *next;
    struct AdjList *adj;
};
struct Node
{
    int data;
    struct Node *next;
};
struct Node *top = NULL;
struct Node *front = NULL;
struct Node *rear = NULL;

struct Node *getNode()
{
    struct Node *p = (struct Node *)malloc(sizeof(struct Node));
    if (p == NULL)
    {
        printf("Memory overflow\n");
        exit(0);
    }
    p->next = NULL;
    return p;
}
void push(int n)
{
    if (top == NULL)
    {
        top = getNode();
        top->data = n;
        return;
    }
    struct Node *newNode = getNode();
    newNode->data = n;
    newNode->next = top;
    top = newNode;
}
int peek()
{
    if (top == NULL)
    {
        return -1;
    }
    return top->data;
}
int pop()
{
    if (top == NULL)
    {
        printf("The stack is empty\n");
        return 0;
    }
    struct Node *temp = top;
    int x = temp->data;
    top = top->next;
    free(temp);
    return x;
}

void enqueue(int n)
{
    if (front == NULL)
    {
        front = getNode();
        front->data = n;
        rear = front;
        return;
    }
    struct Node *newNode = getNode();
    newNode->data = n;
    rear->next = newNode;
    rear = newNode;
}
int dequeue()
{
    if (front == NULL)
    {
        return -1;
    }
    struct Node *currNode = front;
    int x = front->data;
    front = front->next;
    if (front == NULL)
        rear = NULL;
    free(currNode);
    return x;
}

int isEmpty()
{
    return front == NULL;
```

```
}

struct NodeGraph *getNodeGraph()
{
    struct NodeGraph *p = (struct NodeGraph *)malloc(sizeof(struct NodeGraph));
    if (p == NULL)
    {
        printf("Error in creating new NodeGraph\n");
        exit(0);
    }
    p->adj = NULL;
    p->next = NULL;
    return p;
}
struct AdjList *getAdjList()
{
    struct AdjList *p = (struct AdjList *)malloc(sizeof(struct AdjList));
    if (p == NULL)
    {
        printf("Error in creating new NodeGraph\n");
        exit(0);
    }
    p->dest = NULL;
    p->link = NULL;
    return p;
}

int arr[10];
int AdjMatrix[10][10];

void DisplayLinked(struct NodeGraph *header)
{
    header = header->next;

    while (header != NULL)
    {
        struct AdjList *ping = getAdjList();
        printf("%d : ", header->data);
        ping = header->adj;
        while (ping != NULL)
        {
            printf("%d ", ping->dest->data);
            ping = ping->link;
        }
        printf("\n");
        header = header->next;
    }
}
void DisplayMatrix(int n)
{
    int i, j;
    printf("  ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
    for (i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
        for (j = 0; j < n; j++)
        {
            printf("%d ", AdjMatrix[i][j]);
        }
        printf("\n");
    }
}
void updateMatrix(int i, int x, int n)
{
    for (int j = 0; j < n; j++)
    {
        if (arr[j] == x)
        {
            AdjMatrix[i][j] += 1;
            return;
        }
    }
}
void setAdjMat()
{
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            AdjMatrix[i][j] = 0;
        }
    }
}
struct NodeGraph *search(int n, struct NodeGraph *header)
{
    header = header->next;
    while (header != NULL)
    {
        if (header->data == n)
        {
            return header;
        }
        header = header->next;
    }
    printf("No NodeGraph exist with the entered data\n");
    return NULL;
}
struct NodeGraph *takeInputGraph(int n)
{
    int i, a, x;
    struct NodeGraph *header = getNodeGraph();
    struct NodeGraph *currNodeGraph = getNodeGraph();
    header->next = currNodeGraph;

    for (i = 0; i < n; i++)
    {
        if (i != 0)
        {
            struct NodeGraph *newNodeGraph = getNodeGraph();
            currNodeGraph->next = newNodeGraph;
            currNodeGraph = newNodeGraph;
        }
        printf("Enter the data in Node%d : ", i + 1);
        scanf("%d", &arr[i]);
        currNodeGraph->data = arr[i];
    }
    printf("\n");

    currNodeGraph = header->next;

    for (i = 0; i < n; i++)
    {
        printf("Enter the number of links in Node %d : ", arr[i]);
        scanf("%d", &a);
        struct AdjList *adjlistHeader = getAdjList();
        struct AdjList *currAdj = adjlistHeader;
        if (a != 0)
        {
            printf("Enter the links\n");
        }
        if (a == 0)
        {
            adjlistHeader = NULL;
        }

        for (int j = 0; j < a; j++)
        {
            if (j != 0)
            {
                struct AdjList *newAdj = getAdjList();
                currAdj->link = newAdj;
                currAdj = newAdj;
            }
            scanf("%d", &x);
            currAdj->dest = search(x, header);

            updateMatrix(i, x, n);
        }
        currNodeGraph->adj = adjlistHeader;
        currNodeGraph = currNodeGraph->next;
    }
    return header;
};

int searchVisited(int arrVisited[], int len, int n)
{
    for (int i = 0; i < len; i++)
    {
        if (arrVisited[i] == n)
        {
            return 1;
        }
    }
    return -1;
}

void DFS(struct NodeGraph *header, int n)
{
    if (header == NULL || header->next == NULL)
    {
        printf("The graph is empty\n");
        return;
    }
    struct NodeGraph *currNode = getNodeGraph();
    currNode = header->next;
    int arrVisit[n];

    int u = currNode->data;
    push(u);
    int count = 0;

    while (peek() != -1)
    {
        u = pop();
        if (searchVisited(arrVisit, count, u) == -1)
        {
            arrVisit[count++] = u;
            struct NodeGraph *newNode = getNodeGraph();
            newNode = search(u, header);

            struct AdjList *ptr = newNode->adj;

            while (ptr != NULL)
            {
                push(ptr->dest->data);
                ptr = ptr->link;
            }
        }
    }
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arrVisit[i]);
    }
```

```
}

void BFS(struct NodeGraph *header, int n)
{
    if (header == NULL || header->next == NULL)
    {
        printf("The graph is empty\n");
        return;
    }
    struct NodeGraph *currNode = getNodeGraph();
    currNode = header->next;
    int arrVisit[n];

    int u = currNode->data;
    enqueue(u);
    int count = 0;

    while (!isEmpty())
    {
        u = dequeue();
        if (searchVisited(arrVisit, count, u) == -1)
        {
            arrVisit[count++] = u;
            struct NodeGraph *newNode = getNodeGraph();
            newNode = search(u, header);

            struct AdjList *ptr = newNode->adj;

            while (ptr != NULL)
            {
                enqueue(ptr->dest->data);
                ptr = ptr->link;
            }
        }
    }
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arrVisit[i]);
    }
}

void main()
{

    printf("Graph\n");
    printf("Enter the number of Nodes in the graph: ");
    int n;
    scanf("%d", &n);
    setAdjMat();

    struct NodeGraph *header = takeInputGraph(n);
    int response = 1, choice, element, data, node;
    do
    {
        printf("\n\nM E N U\n\n1. Adajcency Matrix Representation\n2. Linked Representa-
tion\n3. Depth First Search\n4. Breadth First Search");
        printf("\n5. Exit\n\n");
    ch:
        printf("\t  -> ");
        scanf("%d", &choice);
        printf("\n\n");
        switch (choice)
        {
        case 1:
            printf("Adjacency Matrix\n");
            DisplayMatrix(n);
            break;

        case 2:
            printf("Linked Representation\n");
            DisplayLinked(header);
            break;

        case 3:
            printf("Depth First Search\n");
            DFS(header, n);
            break;

        case 4:
            printf("Breadth First Search\n");
            BFS(header, n);
            break;

        case 5:
            exit(0);

        default:
            printf("Enter a valid Choice!!!!\n\n");
            go to ch;
        }
    } while (response == 1);
}
```

SAMPLE OUTPUT

```
Enter the number of Nodes in the graph: 5
Enter the data in Node1 : 1
Enter the data in Node2 : 2
Enter the data in Node3 : 3
Enter the data in Node4 : 4
Enter the data in Node5 : 5

Enter the number of links in Node 1 : 2
Enter the links
2 3
Enter the number of links in Node 2 : 1
Enter the links
3
Enter the number of links in Node 3 : 4
Enter the links
1 2 3 4
Enter the number of links in Node 4 : 0
Enter the number of links in Node 5 : 2
Enter the links
1 3


M E N U

1. Adjacency Matrix Representation
2. Linked Representation
3. Depth First Search
4. Breadth First Search
5. Exit

        -> 1

Adjacency Matrix
  1 2 3 4 5
1 0 1 1 0 0
2 0 0 1 0 0
3 1 1 1 1 0
4 0 0 0 0 0
5 1 0 1 0 0
```

```
Depth First Search
1 3 4 2 5

M E N U

1. Adjacency Matrix Representation
2. Linked Representation
3. Depth First Search
4. Breadth First Search
5. Exit

        -> 4

Breadth First Search
1 2 3 4 5

M E N U

1. Adjacency Matrix Representation
2. Linked Representation
3. Depth First Search
4. Breadth First Search
5. Exit

        -> 5
```

RESULT

Graph data structure implemented

Experiment no.17
Date: 04-03-22

SORTING ALGORITHMS

AIM

To implements all sorting on the data read from a file

Data Structures Used: Arrays

ALGORITHM

```
1.START
2.Open the file sort.txt containing the numbers
3.Pass all the values in the file into an integer array A[]
4.Take user input choice and do as follows:
    (a) If choice is to perform insertion sort:
        (i) Set i = 1 and till i < n do:
            (A) Set key = A[i] and j = i - 1
            (B) While j >= 0 and A[j] > key set A[j+1] = A[j] and decrement
            j by 1
            (C) Set A[j + 1] = key
    (b) If choice is to perform Selection sort:
        (i) Set i = 0 and till i < n do:
            (a) Set j = i + 1 and till j < n if A[j] < min then set min = A[j]
            and set pos = j
            (b) If min != A[i] then set temp = A[i] and A[i] = A[pos], A[pos]
            = temp
    (c) If choice is to perform merge sort then do:
        (i) Define function merge_sort() with parameters as integer: array A[],
        b, and e
        (ii) If b >= e then return
        (iii) set m = (b + e) / 2
        (iv) call merge_sort(A, b, m)
        (v) call merge_sort(A, m+1, e)
        (vii) merge the halves
    (d) If choice is to perform quick sort then do:
        (i) Define the function partition() having parameters integer: array[],
        b, e
        (ii)Set pivot = arr[e] and index = b - 1
        (iii) Set i = b and till i < e do:
            (A)If pivot >= arr[i] then swap arr[index + 1] with arr[i]
            (B)Increment index by 1
        (iv) Swap arr[index + 1] with arr[e] and return index + 1
        (v) Define the function quicksort() having parameters integer: array
        A[], b, e
        (vi).If b < e then set p = partition(arr, b, e)
        (vii) Call quicksort(arr, b, p - 1)
        (viii) call quicksort(arr, p+1, e)
    (e) If choice is to perform heap sort then do:
        (i) Define heapify() function with parameters integer: arr[], n, i
        (ii) Set largest = i and l = 2 * i + 1, set r = 2 * i + 2
        (iii) If l<n and arr[l] > arr[largest] then set largest = l
        (iv) If r < n and arr[r] > arr[largest]
        (v) If largest != -1 then swap arr[i] and arr[largest] also call
        heapify(arr, n,
        largest)
        (vi) Define heapsort() function with parameters integer: arr[], n
        (vii) Set i = n / 2 -1 and till i >= 0, call heapify(arr, n, i), increment i
        by 1
        (viii) Set i = n - 1 and till i > 0, swap arr[0] and arr[i] and call
        heapify(arr, i, 0)
        increment i by 1
        (f) Display all elements in A[]
5.STOP
```

PROGRAM

```c
#include <stdlib.h>
#include <stdio.h>
void swap(int *a, int *b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i)
    {
        swap(&arr[i], &arr[largest]);
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i = n - 1; i > 0; i--)
    {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}
int partition(int arr[], int b, int e)
{
    int pivot = arr[e];
    int index = b - 1;
    for (int i = b; i < e; i++)
    {
        if (pivot >= arr[i])
        {
            swap(&arr[index + 1], &arr[i]);
            index++;
        }
    }
    swap(&arr[index + 1], &arr[e]);
    return (index + 1);
}
void quick_sort(int arr[], int b, int e)
{
    if (b < e)
    {
        int p = partition(arr, b, e);
        quick_sort(arr, b, p - 1);
        quick_sort(arr, p + 1, e);
    }
}
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }

        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
void merge_sort(int A[], int b, int e)
```

```
{
    if (b >= e)
        return;
    int m = (b + e) / 2;
    merge_sort(A, b, m);
    merge_sort(A, m + 1, e);
    merge(A, b, m, e);
}
void selection_sort(int A[], int n)
{
    for (int i = 0; i < n; i++)
    {
        int min = A[i], pos = i;
        for (int j = i + 1; j < n; j++)
        {
            if (A[j] < min)
            {
                min = A[j];
                pos = j;
            }
        }
        if (min != A[i])
        {
            int temp = A[i];
            A[i] = A[pos];
            A[pos] = temp;
        }
    }
}
void insertion_sort(int A[], int n)
{
    int j, key;
    for (int i = 1; i < n; i++)
    {
        int key = A[i];
        int j = i - 1;
        while (j >= 0 && A[j] > key)
        {
            A[j + 1] = A[j];
            --j;
        }
        A[j + 1] = key;
    }
}
int main()
{
    FILE *fptr1, *fptr2;
    char file1[100], c;
    fptr1 = fopen("sort.txt", "r");
    if (fptr1 == NULL)
    {
        printf("Cannot open file %s \n", file1);
        return (0);
    }
    int num[10], i = 0;
    char x[5];
    int k = 0;
    while ((c = fgetc(fptr1)) != EOF)
    {
        if (c != '\n')
        {
            x[k] = c;
            k++;
        }
        else
        {
            x[k] = '\0';
            num[i] = atoi(x);
            i++;
            k = 0;
        }
    }
    x[k] = '\0';
    num[i] = atoi(x);
    printf("\nContents:\n");
    for (int j = 0; j <= i; j++)
    {
        printf("%d ", num[j]);
    }
    fclose(fptr1);
    int choice;
    printf("\nEnter your choice of sort\n1.Insertion sort\n2.Selection
sort\n       3.Merge sort\n4.Quick sort\n5.Heap sort\n");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
        printf("Insertion sort\n");
        insertion_sort(num, i + 1);
        break;
    case 2:
        printf("Selection sort\n");
        selection_sort(num, i + 1);
        break;
    case 3:
        printf("Merge sort\n");
        merge_sort(num, 0, i);
        break;
    case 4:
        printf("Quick sort\n");
        quick_sort(num, 0, i);
        break;
    case 5:
        printf("Heap sort\n");
```

```
            heapSort(num, i + 1);
            break;
        default:
            printf("Invalid choice\n");
            break;
        }
        printf("\nAfter sort\n");
        for (int j = 0; j <= i; j++)
        {
            printf("%d ", num[j]);
        }
    }
```

SAMPLE OUTPUT

```
Contents:
10 1 24 13 5 17 -1 -5
Enter your choice of sort
1.Insertion sort
2.Selection sort
3.Merge sort
4.Quick sort
5.Heap sort
3
Merge sort
After sort
-5 -1 1 5 10 13 17 24
```

RESULT

Sorting algorithms were implemented, and output is verified

Experiment no.18
Date: 04-03-22

HASH TABLE USING CHAINING

AIM
To implement a hash table using chaining method with size 10

Data Structure Used: Linked list

ALGORITHM

```
1. START
2. Define a struct node with data members key and pointer *next of type struct node
3. Declare variables *table[TABLE_SIZE] = {NULL} and *temp
4. Enter values into the hash table such that:
    (a)Input value and set it as data
    (b)Set i = data % TABLE_SIZE
    (c)Dynamically allocate memory for newnode
    (d)Set newnode->key = data
    (e)Set newnode->next = NULL
    (f)If table[i] = new node
    (g)Else set temp = table[i] and while temp->next != NULL set temp =
    temp->next also set temp->next = newnode
5. To display elements in the hashtable do:
    (a)Set i = 0 and till i <TABLE_SIZE do:
        (i) If table[i] == NULL then display i
        (ii) Else display i and set temp = table[i]
        (iii) Set temp = table[i] and till temp != NULL display temp->key and
        set temp = temp->next
6. STOP
```

PROGRAM

```c
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10
struct node
{
    int key;
    struct node *next;
};
struct node *table[TABLE_SIZE] = {NULL};
struct node *temp;
void search()
{
    printf("Enter the data you want to search for : ");
    int data;
    scanf("%d", &data);
    int i = data % TABLE_SIZE;
    if (table[i] == NULL)
    {
        printf("The element is not present !!!");
    }
    else
    {
        for (temp = table[i]; temp != NULL; temp = temp->next)
        {
            if (temp->key == data)
            {
                printf("The elements has been found at index %d", i);
                break;
            }
        }
        if (temp == NULL)
        {
            printf("The element is not present !!!");
        }
    }
}
void display()
{
    printf("Displaying the elements of the hash map :\n\n\n\n\n");
    printf("INDEX\tKEY\n");
    for (int i = 0; i < TABLE_SIZE; i++)
    {
        if (table[i] == NULL)
        {
            printf("%d", i);
            printf("\t");
            printf("\n");
            continue;
        }
        else
        {
            printf("%d", i);
            printf("\t");
            temp = table[i];
            for (temp = table[i]; temp != NULL; temp = temp->next)
            {
                printf("%d->", temp->key);
            }
            printf("\n");
        }
    }
}
void main()
{
    printf("Enter the 10 values into the hash table \n");
    int a = 10;
    int data;
    while (a)
    {
        printf("Enter the data to enter into the hash table : ");
        scanf("%d", &data);
        int i = data % TABLE_SIZE;
        struct node *newnode = (struct node *)malloc(sizeof(struct node));
        newnode->key = data;
        newnode->next = NULL;
        if (table[i] == NULL)
        {
            table[i] = newnode;
        }
        else
        {
            temp = table[i];
            while (temp->next != NULL)
            {
                temp = temp->next;
            }
            temp->next = newnode;
        }
        a--;
    }
    while (1)
    {
        printf("\n\n1.Search the hash map\n2.Display the hash map\n3.Exit\nEnter your
choice : ");
        int choice;
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            search();
            break;
        case 2:
            display();
            break;
        case 3:
            exit(0);
        }
    }
}
```

SAMPLE OUTPUT

```
Enter the 10 values into the hash table
Enter the data to enter into the hash table : 1
Enter the data to enter into the hash table : 2
Enter the data to enter into the hash table : 3
Enter the data to enter into the hash table : 4
Enter the data to enter into the hash table : 5
Enter the data to enter into the hash table : 6
Enter the data to enter into the hash table : 7
Enter the data to enter into the hash table : 8
Enter the data to enter into the hash table : 9
Enter the data to enter into the hash table : 10
1.Search the hash map
2.Display the hash map
3.Exit
Enter your choice : 2


Displaying the elements of the hash map :
168INDEX KEY
0 10->
1 1->
2 2->
3 3->
4 4->
5 5->
6 6->
7 7->
```

```
8 8->
9 9->

1.Search the hash map
2.Display the hash map
3.Exit
Enter your choice : 3
```

RESULT

Hash table was created using chaining method and output is verified

Experiment no.19
Date: 04-03-22

HASH TABLE USING LINEAR PROBING

AIM
To implement Hash table that uses linear probing for collision resolution.

Data Structure Used: array

ALGORITHM
```
1.START
2.Define an integer array table[TABLE_SIZE] = {NULL}
3.Input values into the hashtable such that:
      (a) Input value and set it as data
      (b) Set index = data % TABLE_SIZE
      (c) If table[index] = data
      (d) Else set i = index + 1 and if i == TABLE_SIZE then set i = 0
      (e) while i != index, if table[i] == NULL then set table[i] = data and if i ==
            TABLE_SIZE then set i = 0, increment i by 1
      (f) If i == index display the table is full
      (g) Display elements in the hash table
4.STOP
```

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10
int table[TABLE_SIZE] = {NULL};
void insert()
{
    int data;
    int i;
    printf("Enter data into the hash table : ");
    scanf("%d", &data);
    int index = data % TABLE_SIZE;
    if (table[index] == NULL)
    {
        table[index] = data;
    }
    else
    {
        i = index + 1;
        if (i == TABLE_SIZE)
            i = 0;
        while (i != index)
        {
            if (table[i] == NULL)
            {
                table[i] = data;
                break;
            }
            i++;
            if (i == TABLE_SIZE)
            {
                i = 0;
            }
        }
    }
    if (i == index)
    {
        printf("The table was full and thus element couldnt be inserted ");
    }
}
void search()
{
    int data;
    int i;
    printf("Enter the data to search for : ");
    scanf("%d", &data);
    int index = data % TABLE_SIZE;
    if (table[index] == data)
        printf("Data entered found at index %d", index);
    else
    {
        i = index + 1;
        while (i != index)
        {
            if (table[i] == data)
            {
                printf("Data found at index %d", index);
                break;
            }
            i++;
            if (i == TABLE_SIZE)
            {
                i = 0;
            }
        }
    }
    if (i == index)
    {
        printf("The element was not found within the given hash table ");
    }
}
void display()
{
    printf("Displaying the elements of the hash table :\n\n\n\n\n");
    printf("INDEX\tKEY\n");
    for (int i = 0; i < TABLE_SIZE; i++)
    {
        if (table[i] == NULL)
        {
            printf("%d", i);
            printf("\t");
            printf("\n");
            continue;
        }
        else
        {
            printf("%d", i);
            printf("\t");
            printf("%d", table[i]);
            printf("\n");
        }
    }
}
int main()
{
    while (1)
    {
        printf("\n\n1.Enter into the hash table\n2.Search the hash table\n3.Display the
hash table\n4.Exit\nEnter your choice : ");
        int choice;
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            insert();
            break;
        case 2:
            search();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        }
    }
}
```

SAMPLE OUTPUT

```
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
4.Exit
Enter your choice : 1
Enter data into the hash table : 1
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
4.Exit
Enter your choice : 1
Enter data into the hash table : 2
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
4.Exit
Enter your choice : 1
Enter data into the hash table : 3
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
4.Exit
Enter your choice : 1
Enter data into the hash table : 4
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
4.Exit
Enter your choice : 1
Enter data into the hash table : 5
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
```

```
4.Exit
Enter your choice : 1
Enter data into the hash table : 6
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
4.Exit
Enter your choice : 1
Enter data into the hash table : 7
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
4.Exit
Enter your choice : 1
Enter data into the hash table : 8
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
4.Exit
Enter your choice : 1
Enter data into the hash table : 9
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
4.Exit
Enter your choice : 1
Enter data into the hash table : 10
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
4.Exit
Enter your choice : 3
Displaying the elements of the hash table :
```

```
INDEX KEY
0 10
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
178
1.Enter into the hash table
2.Search the hash table
3.Display the hash table
4.Exit
Enter your choice: 4
```

RESULT

Hash table was created using linear probing and output is verified

Experiment no.20
Date: 04-03-22

MEMORY ALLOCATOR AND GARBAGE COLLECTOR

AIM
To simulate the memory allocator and garbage collector using doubly linked list

Data Structure Used: Doubly linked list

ALGORITHM
```
1. START
2. Define struct node with data elements integer data and pointers *next
and *prev
3. Declare variable *head = NULL
3. Input n the no of memory blocks available and set cur = n
4. Set *p = NULL
5. Input memory block sizes and dynamically allocate memory for newnode
6. Set block size as newnode->data
7. Set newnode->prev = p and newnode->next = NULL
8. If head == NULL then set head = newnode and set p = newnode
9. Else set p->next = newnode and set p = newnode
10. Take user input choice and do as follows:
    (a) If choice is to allocate memory, then do:
        (i) set *p = head
        (ii) If head == NULL then display no memory blocks to be allocated
        and return
        (iii) Input memory block to be allocated and set it as val
        (iv) Display the memory >= val and remove it from the list
    (b) If choice is to deallocate memory, then do:
        (i) Input memory to be deallocated
        (ii) Position it at the last of the list containing memory blocks
11. STOP
```

PROGRAM

```c
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
    struct node *prev;
};
struct node *head = NULL;
int n, cur;
void display()
{
    struct node *p = head;
    if (p == NULL)
    {
        return;
    }
    printf("Memory blocks available is/are: ");
    while (p != NULL)
    {
        printf("%d ", p->data);
        p = p->next;
    }
    printf("\n");
}
void allocate()
{
    int val;
    struct node *p = head;
    if (head == NULL)
    {
        printf("There is no memory blocks to be allocated!!!!\n");
        return;
    }
    printf("Enter the memory block size to be allocated: ");
    scanf("%d", &val);
    if (head->data >= val)
    {
        if (head->next != NULL)
        {
            head = head->next;
            head->prev = NULL;
        }
        else
            head = NULL;
        printf("A memory block of size %d was allocated\n", p->data);
        --cur;
        display();
    }
    else if (head->data < val && head->next == NULL)
    {
        printf("No memory blocks are available to be allocated!!!\n");
        return;
    }
    else
    {
        while (p->next != NULL)
        {
            if (p->next->data >= val)
            {
                --cur;
                printf("A memory block of size %d was allocated\n", p->next->data);
                if (p->next->next != NULL)
                {
                    p->next->next->prev = p;
                    p->next = p->next->next;
                }
                else
                    p->next = NULL;
                display();
                return;
            }
            p = p->next;
        }
        printf("No memory blocks are available to be allocated!!!\n");
        return;
    }
}
void deallocate()
{
    if (cur == n)
    {
        printf("No memory blocks available to be deallocated!!!\n");
        return;
    }
    printf("Enter the memory block size to be deallocated: ");
    int val;
    scanf("%d", &val);
    struct node *p = head;
    struct node *newnode;
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = val;
    newnode->prev = p;
    newnode->next = NULL;
    if (p == NULL)
    {
        head = newnode;
        p = newnode;
        ++cur;
        display();
        return;
    }
    while (p->next != NULL)
        p = p->next;
    newnode->prev = p;
    p->next = newnode;
    ++cur;
    display();
}
int main()
{
    printf("Enter number of memory blocks available: ");
    scanf("%d", &n);
    cur = n;
    struct node *p = NULL;
    printf("Enter %d memory block sizes\n", n);
    for (int i = 0; i < n; i++)
    {
        struct node *newnode;
        newnode = (struct node *)malloc(sizeof(struct node));
        int val;
        printf("Enter block size: ");
        scanf("%d", &val);
        newnode->data = val;
        newnode->prev = p;
        newnode->next = NULL;
        if (head == NULL)
            head = newnode, p = newnode;
        else
            p->next = newnode, p = newnode;
    }
    do
    {
        printf("\n1. Allocate memory block\n2. Deallocate memory blocks\n3. Exit\nChoose
your option:");
        int opt;
        scanf("%d", &opt);
        switch (opt)
        {
        case 1:
            allocate();
            break;
        case 2:
            deallocate();
            break;
        case 3:
            exit(0);
        default:
            printf("Enter a valid option!!!!\n");
        }
    } while (1);
}
```

SAMPLE OUTPUT

```
Enter number of memory blocks available: 3
Enter 3 memory block sizes
Enter block size: 200
Enter block size: 300
Enter block size: 400
1. Allocate memory block
2. Deallocate memory blocks
3. Exit
Choose your option:1
Enter the memory block size to be allocated: 275
A memory block of size 300 was allocated
Memory blocks available is/are: 200 400
1. Allocate memory block
2. Deallocate memory blocks
3. Exit
Choose your option:2
Enter the memory block size to be deallocated: 300
Memory blocks available is/are: 200 400 300
1. Allocate memory block
2. Deallocate memory blocks
3. Exit
Choose your option:3
```

RESULT

Garbage collector was implemented using doubly linked list and the output was verified.