**Practical File**

Of

**Data Structures using C**

**

***Submitted to: Submitted by:***

**Prof. (Dr.) Ashutosh Dixit                                                          Chetan**

Professor                                                                                       Roll No:19001003033

CE Department                                                                              B.Tech.(CE) 3rd Sem

**\_\_\_**

**Department of Computer Engineering**

**J. C. Bose University of Science and Technology YMCA, Faridabad**

**Table of Contents**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S. No.** | **Program** | **Date** | **Page No.** | **Teacher’s signature** | **Remarks** |
| **1.** | Write a Program to draw pyramid of ‘\*’ using For Loop. | 22/07/2020 | 6-7 |  |  |
| **2.** | Write a Program to implement Menu-driven calculator of basic mathematical functions using Switch-case-default. | 22/07/2020 | 8-9 |  |  |
| **3.** | Write a Program to implement Menu-driven operations of a 3x3 matrix using Switch-case-default. | 28/07/2020 | 10-15 |  |  |
| **4.** | Write a Program to implement of Insertion process at random place in 2-D Array | 28/07/2020 | 16-17 |  |  |
| **5.** | Write a Program to implement Menu-driven operations of a 1-D array using Switch-case-default. (include the following operations: create, traversal, insertion, deletion) | 04/08/2020 | 18-23 |  |  |
| **6.** | Write a program to implement linear search and binary search | 11/08/2020 | 24-27 |  |  |
| **7.** | Write a program to find address of an element of an array | 18/08/2020 | 28-29 |  |  |
| **8.** | Write a program to implement bubble sort | 25/08/2020 | 30-30 |  |  |
| **9.** | Write a program to implement selection sort | 25/08/2020 | 31-31 |  |  |
| **10.** | Write a program to implement insertion sort | 01/09/2020 | 32-32 |  |  |
| **11.** | Write a program to implement bubble sort(optimised) | 01/09/2020 | 33-34 |  |  |
| **12.** | Write a program to implement merge concept with two different sorted list | 08/09/2020 | 34-35 |  |  |
| **13.** | Write a program to implement merge sort with one unsorted list | 08/09/2020 | 35-36 |  |  |
| **14.** | Write a program to implement quick sort on a list of 10 elements | 15/09/2020 | 37-38 |  |  |
| **15.** | Write a convert Infix to Postfix expression | 22/09/2020 | 39-40 |  |  |
| **16.** | Write a program to evaluate a Postfix expression | 22/09/2020 | 41-44 |  |  |
| **17.** | Write a program to count the parenthesis in an expression | 29/09/2020 | 45-46 |  |  |
| **18.** | Write a program to implement linear queue with all necessary operations | 06/10/2020 | 47-48 |  |  |
| **19.** | Write a program to implement Double Ended queue with all necessary operations | 13/10/2020 | 49-51 |  |  |
| **20.** | Write a program to implement Linear Linked List with all necessary operations | 27/10/2020 | 52-55 |  |  |
| **21.** | Write a program to implement Doubly Linked List with all necessary operations | 03/11/2020 | 56-65 |  |  |
| **22.** | Write a program to implement Stack using Linked List with all necessary operations | 10/11/2020 | 66-72 |  |  |
| **23.** | Write a program to implement Queue using Linked List with all necessary operations | 10/11/2020 | 73-76 |  |  |
| **24.** | Write a program to implement Binary Tree with all necessary operations | 24/11/2020 | 77-88 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Program # 1**

**Aim:** Write a program to draw pyramid of ’\*’ using for loop

**Coding:**

//Simple pattern to draw.

#include <stdio.h>

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

{

    /\* code \*/

    int limit;

    printf("Enter the limit upto which you want to print the pattern: ");

    scanf("%d", &limit);

    for(int i=1; i<=limit; i++) {

        for(int j=i; j>0; j--) {

            printf("\*");

        }

        printf("\n");

    }

    return 0;

}

**Output:**

Enter the limit upto which you want to print the pattern: 10

**\***

**\*\***

**\*\*\***

**\*\*\*\***

**\*\*\*\*\***

**\*\*\*\*\*\***

**\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\***

**Program # 2**

**Aim:** Write a Program to implement Menu-driven calculator of basic mathematical functions using Switch-case-default.

**Coding:**

// Implementation of calculator using switch-case statements.

#include <stdio.h>

#include <stdlib.h>

int main() {

    float a, b;

    printf("Enter two operands: ");

    scanf("%f %f", &a, &b);

    fflush(stdin);

    char ch;

    printf("Enter an operator(+, -, \*, /): ");

    scanf("%c",&ch);

    switch(ch) {

        case '+':

            printf("%.1f + %.1f = %.1f\n", a, b, a+b);

            break;

        case '-':

            printf("%.1f - %.1f = %.1f\n", a, b, a-b);

            break;

        case '\*':

            printf("%.1f x %.1f = %.1f\n", a, b, a\*b);

            break;

        case '/':

            if(b != 0)

                printf("%.1f / %.1f = %.1f\n", a, b, a/b);

            else

                printf("Division by zero is not allowed.");

            break;

        default:

            printf("ERROR! Invalid operator.");

    }

    return 0;

}

Output:

Enter two operands: 12 10

Enter an operator(+, -, \*, /): +

12.0 + 10.0 = 22.0

Enter two operands: 12 10

Enter an operator(+, -, \*, /): -

12.0 - 10.0 = 2.0

Enter two operands: 12 10

Enter an operator(+, -, \*, /): \*

12.0 x 10.0 = 120.0

Enter two operands: 12 10

Enter an operator(+, -, \*, /): /

12.0 / 10.0 = 1.2

Program # 3

Aim: Write a Program to implement Menu-driven operations of a 3x3 matrix using Switch-case-default.

Coding:

// Basic matrix program

#include <stdio.h>

int main() {

    //declaring matrices

    int M1[3][3], M2[3][3];

    int inputIndicator = 0;

    //defining variable for choice

    int choice;

    do {

                //defining menu

                printf("Enter 1 to input matrices\n");

                printf("Enter 2 to display matrices\n");

                printf("Enter 3 to add the entered matrices\n");

                printf("Enter 4 to subtract the entered matrices\n");

                printf("Enter 0 to exit\n");

                //taking user choice

                scanf("%d", &choice);

                switch (choice) {

                    case 1:

                        //input matrix 1

                        printf("Enter matrix 1 elements\n");

                        for (int i=0; i<3; i++)

                            for(int j=0; j<3; j++)

                                scanf("%d", &M1[i][j]);

                        //input matrix 2

                        printf("Enter matrix 2 elements\n");

                        for (int i=0; i<3; i++)

                            for(int j=0; j<3; j++)

                                scanf("%d", &M2[i][j]);

                        inputIndicator = 1;

                    break;

                    case 2:

                        //case when the user wants to display the matrices without entering the

                        //elements

                          if (!inputIndicator)

                            printf("ERROR!! Input the matrices first.\n");

                          else {

                                //displaying matrix 1

                                printf("Matrix 1\n");

                                for (int i=0; i<3; i++) {

                                    for(int j=0; j<3; j++)

                                        printf("%d ", M1[i][j]);

                                    printf("\n");

                                }

                                //displaying matrix 2

                                printf("Matrix 2\n");

                                for (int i=0; i<3; i++) {

                                    for(int j=0; j<3; j++)

                                        printf("%d ", M2[i][j]);

                                    printf("\n");

                                }

                            }

                    break;

                    case 3:

                        //case when the user tries to add the matrices

                        //without entering the elements of matrices

                          if (!inputIndicator)

                            printf("ERROR!! Input the matrices first.\n");

                          else {

                                //declaring add matrix

                                int add[3][3];

                                //displaying the addition matrix

                                printf("Addition matrix\n");

                                for (int i=0; i<3; i++) {

                                    for (int j=0; j<3; j++){

                                        add[i][j] = M1[i][j] + M2[i][j];

                                        printf("%d ", add[i][j]);

                                    }

                                    printf("\n");

                                }

                          }

                    break;

                    case 4:

                        //case when the user tries to subtract the matrices

                        //without entering the elements of matrices

                          if (!inputIndicator)

                            printf("ERROR!! Input the matrices first.\n");

                          else {

                                //declaring a subtract matrix

                                int sub[3][3];

                                //displaying the addition matrix

                                printf("Subtraction matrix\n");

                                for (int i=0; i<3; i++) {

                                    for (int j=0; j<3; j++) {

                                        sub[i][j] = M1[i][j] - M2[i][j];

                                        printf("%d ", sub[i][j]);

                                    }

                                    printf("\n");

                                }

                          }

                    break;

                    case 0:

                        printf("Exiting from the do-while loop.");

                    break;

                    default:

                            printf("ERROR!! Please enter a valid choice.\n");

                }

        printf("\n");

    } while (choice != 0);

    return 0;

}

Output:

Enter 1 to input matrices

Enter 2 to display matrices

Enter 3 to add the entered matrices

Enter 4 to subtract the entered matrices

Enter 5 to multiply the entered matrices

Enter 0 to exit

1

Enter matrix 1 elements

1 2 3

4 5 6

7 8 9

Enter matrix 2 elements

1 0 0

0 1 0

0 0 1

Enter 1 to input matrices

Enter 2 to display matrices

Enter 3 to add the entered matrices

Enter 4 to subtract the entered matrices

Enter 5 to multiply the entered matrices

Enter 0 to exit

2

Matrix 1

1 2 3

4 5 6

7 8 9

Matrix 2

1 0 0

0 1 0

0 0 1

Enter 1 to input matrices

Enter 2 to display matrices

Enter 3 to add the entered matrices

Enter 4 to subtract the entered matrices

Enter 5 to multiply the entered matrices

Enter 0 to exit

3

Addition matrix

2 2 3

4 6 6

7 8 10

Enter 1 to input matrices

Enter 2 to display matrices

Enter 3 to add the entered matrices

Enter 4 to subtract the entered matrices

Enter 5 to multiply the entered matrices

Enter 0 to exit

4

Subtraction matrix

0 2 3

4 4 6

7 8 8

Enter 1 to input matrices

Enter 2 to display matrices

Enter 3 to add the entered matrices

Enter 4 to subtract the entered matrices

Enter 5 to multiply the entered matrices

Enter 0 to exit

0

Exiting from the do-while loop.

Program # 4

Aim: Write a Program to implement of Insertion process at random place in 2-D Array

Coding:

#include <stdio.h>

#include <conio.h>

#define R 3

#define C 4

void main() {

    static int a[R][C];

    int size=8;

    printf("Enter %d elements in the 2D array:\n", size);

    int \*p = a;

    for (int i=0; i<size;i++) {

        scanf("%d", p+i);

    }

    printf("Before insertion array is:\n");

    int r;

    if (size%C == 0)    r = size/C;

    else    r = size/C + 1;

    for (int i=0; i<r; i++) {

        for (int j=0; j<C; j++)

            printf("%d ", a[i][j]);

        printf("\n");

    }

    int pos, ele;

    printf("Enter the position and element: ");

    scanf("%d %d", &pos, &ele);

    for (int i=size; i>=pos; i--) {

        \*(p+i+1) = \*(p+i);

    }

    \*(p+pos) = ele;

    size++;

    if (size%C == 0)    r = size/C;

    else    r = size/C + 1;

    printf("After insertion of %d at position %d in the 2D array, array is:\n", ele, pos);

    for (int i=0; i<r; i++) {

        for (int j=0; j<C; j++)

            printf("%d ", a[i][j]);

        printf("\n");

    }

    getch();

}

Output:

Enter 8 elements in the 2D array:

1 2 3 4

5 6 7 8

Before insertion array is:

1 2 3 4

5 6 7 8

Enter the position and element: 0 10

After insertion of 10 at position 0 in the 2D array, array is:

10 1 2 3

4 5 6 7

8 0 0 0

Program # 5

Aim: Write a Program to implement Menu-driven operations of a 1-D array using Switch-case-default. ( include the following operations :create, traversal, insertion, deletion)

Coding:

// Menu-driven program for arrays

#include <stdio.h>

int main() {

                //defining the array

                int arr[20];

                int size, inputIndicator = 0;

                int choice;

                //describing menu

                do {

                        printf("Enter 1 to input array elements.\n");

                        printf("Enter 2 to display array elements.\n");

                        printf("Enter 3 to insert an element in array at a specified position.\n");

                        printf("Enter 4 to delete an element in array at a specified position.\n");

                        printf("Enter 0 to stop.\n");

                        scanf("%d", &choice);

                        switch (choice) {

                                case 0:

                                    printf("Successful exit.\n");

                                break;

                                case 1:

                                    size = 2;

                                    printf("Enter %d elements in array: ", size);

                                    for (int i=0; i<size; i++)

                                        scanf("%d", &arr[i]);

                                    inputIndicator = 1;

                                break;

                                case 2:

                                    if (!inputIndicator) {

                                        printf("ERROR!!Input array elements first.\n");

                                        break;

                                    }

                                    if (size == 0) {

                                        printf("Empty array.\n");

                                        break;

                                    }

                                     printf("Array is : [%d", arr[0]);

                                     for (int i=1; i<size; i++)

                                        printf(", %d", arr[i]);

                                     printf("]\n");

                                break;

                                case 3:

                                    if (!inputIndicator) {

                                        printf("ERROR!!Input array elements first.\n");

                                        break;

                                    }

                                     if (size == 20) {

                                        printf("ERROR!!Not more than %d elements can be inserted in array.\n", size-10);

                                        break;

                                     }

                                     int element, pos;

                                     printf("Enter the element and its position to insert in the array: ");

                                     scanf("%d %d", &element, &pos);

                                     //inserting at the desired position

                                     for (int i=size-1; i>=pos-1; i--)

                                          arr[i+1] = arr[i];

                                     arr[pos-1] = element;

                                     size++;

                                break;

                                case 4:

                                    if (!inputIndicator) {

                                        printf("ERROR!!Input array elements first.\n");

                                        break;

                                    }

                                    if (size == 0) {

                                        printf("ERROR!!Array is already empty.\n");

                                        break;

                                    }

                                    int loc;

                                    printf("Enter the position of element for deletion: ");

                                    scanf("%d", &loc);

                                    for (int i=loc-1; i<size; i++)

                                        arr[i] = arr[i+1];

                                    size--;

                                break;

                            default:

                                printf("ERROR!!Please enter a valid choice.\n");

                        }

                    printf("\n");

                } while (choice != 0);

    return 0;

}

Output:

Enter 1 to input array elements.

Enter 2 to display array elements.

Enter 3 to insert an element in array at a specified position.

Enter 4 to delete an element in array at a specified position.

Enter 0 to stop.

1

Enter 2 elements in array: 1 2

Enter 1 to input array elements.

Enter 2 to display array elements.

Enter 3 to insert an element in array at a specified position.

Enter 4 to delete an element in array at a specified position.

Enter 0 to stop.

2

Array is : [1, 2]

Enter 1 to input array elements.

Enter 2 to display array elements.

Enter 3 to insert an element in array at a specified position.

Enter 4 to delete an element in array at a specified position.

Enter 0 to stop.

3

Enter the element and its position to insert in the array: 3

3

Enter 1 to input array elements.

Enter 2 to display array elements.

Enter 3 to insert an element in array at a specified position.

Enter 4 to delete an element in array at a specified position.

Enter 0 to stop.

2

Array is : [1, 2, 3]

Enter 1 to input array elements.

Enter 2 to display array elements.

Enter 3 to insert an element in array at a specified position.

Enter 4 to delete an element in array at a specified position.

Enter 0 to stop.

4

Enter the position of element for deletion: 1

Enter 1 to input array elements.

Enter 2 to display array elements.

Enter 3 to insert an element in array at a specified position.

Enter 4 to delete an element in array at a specified position.

Enter 0 to stop.

2

Array is : [2, 3]

Enter 1 to input array elements.

Enter 2 to display array elements.

Enter 3 to insert an element in array at a specified position.

Enter 4 to delete an element in array at a specified position.

Enter 0 to stop.

3

Enter the element and its position to insert in the array: 3 1

Enter 1 to input array elements.

Enter 2 to display array elements.

Enter 3 to insert an element in array at a specified position.

Enter 4 to delete an element in array at a specified position.

Enter 0 to stop.

2

Array is : [3, 2, 3]

Enter 1 to input array elements.

Enter 2 to display array elements.

Enter 3 to insert an element in array at a specified position.

Enter 4 to delete an element in array at a specified position.

Enter 0 to stop.

0

Successful exit.

Program # 6

Aim: Write a program to implement linear search and binary search

Coding:

// Linear search and binary search algorithms implementation

#include <stdio.h>

#define SIZE 10

int main() {

        //defining an array

        int arr[SIZE], choice;

        printf ("Enter %d elements in the array(in sorted manner): ", SIZE);

        for (int i=0; i<SIZE; i++)

            scanf("%d", &arr[i]);

        //declaring an element to take input from user to search it in the array

        int element, flag = 0, i;

        printf ("Enter an element to know whether it is present in the array or not: ");

        scanf("%d", &element);

        do {

            printf ("Enter 1 for Linear search.\n");

            printf("Enter 2 for Binary search.\n");

            printf("Enter 0 to exit.\n");

            scanf("%d", &choice);

            switch(choice) {

                case 0:

                printf ("Exiting...\n");

                break;

                case 1:

                //Linear search implementation to know only if the element is present in the array or not

                for (i=0; i<SIZE; i++)

                    if (arr[i] == element) {

                        flag = 1;

                        break;

                }

                if (flag)

                    printf("%d is present in the array at position %d.\n", element, i+1);

                else

                    printf("%d is not present in the array.\n", element);

                break;

                /\*

                printf ("Enter an element to know its total occurrances in the array(if present): ");

                scanf("%d", &element);

                //Linear search to know the total occurrances of element in the array

                int count = 0;

                for (int i=0; i<SIZE; i++)

                    if (arr[i] == element)

                        count++;

                if (count)

                    printf("%d occurs %d time(s) in the array.\n", element, count);

                else

                    printf("Search unsuccessful. %d is not at all present in the array.\n", element);

                \*/

                case 2: {

                //Binary search iterative implementation

                int beg=0, end=SIZE-1, mid;

                flag = 0;

                while (beg <= end) {

                     mid = (beg + end) / 2;

                    if (arr[mid] == element) {

                        flag = 1;

                        break;

                    }

                    else if (arr[mid] > element)

                            end = mid-1;

                         else

                            beg = mid+1;

                }

                if (flag)

                    printf("%d is present in the array at %d position.\n", element, mid+1);

                else

                    printf("Search unsuccessful. %d is not at all present in the array.\n", element);

                }

                break;

                default:

                    printf("Please, enter a valid choice.\n");

            }

        printf("\n");

    }while (choice != 0);

    printf("Outside do-while loop.\n");

    return 0;

}

Output:

Enter 10 elements in the array(in sorted manner): 12 23 34 45 56 67 78 89 90 100

Enter an element to know whether it is present in the array or not: 101

Enter 1 for Linear search.

Enter 2 for Binary search.

Enter 0 to exit.

1

101 is not present in the array.

Enter 1 for Linear search.

Enter 2 for Binary search.

Enter 0 to exit.

2

Search unsuccessful. 101 is not at all present in the array.

Enter 1 for Linear search.

Enter 2 for Binary search.

Enter 0 to exit.

0

Exiting...

Outside do-while loop.

Program # 7

Aim: Write a program to find address of an element of an array

Coding:

// This program prints the address of elements and their values

#include <stdio.h>

#include <conio.h>

#define R 4

#define C 4

void main() {

        //declaring matrix array

        int mat[R][C];

        printf("Enter %d elements in the %dX%d matrix: ", R\*C, R, C);

        for (int i=0; i<R; i++)

            for (int j=0; j<C; j++)

                scanf("%d", &mat[i][j]);

        printf("Printing the addresses and values: ...");

        for (int i=0; i<R; i++)

            for (int j=0; j<C; j++)

                printf("\nAddress[%d][%d]: %u\tValue[%d][%d]: %-3d", i+1, j+1, &mat[i][j], i+1, j+1, mat[i][j]);

    getch();

}

Output:

Enter 16 elements in the 4X4 matrix: 1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

Printing the addresses and values: ...

Address[1][1]: 6422208 Value[1][1]: 1

Address[1][2]: 6422212 Value[1][2]: 2

Address[1][3]: 6422216 Value[1][3]: 3

Address[1][4]: 6422220 Value[1][4]: 4

Address[2][1]: 6422224 Value[2][1]: 5

Address[2][2]: 6422228 Value[2][2]: 6

Address[2][3]: 6422232 Value[2][3]: 7

Address[2][4]: 6422236 Value[2][4]: 8

Address[3][1]: 6422240 Value[3][1]: 9

Address[3][2]: 6422244 Value[3][2]: 10

Address[3][3]: 6422248 Value[3][3]: 11

Address[3][4]: 6422252 Value[3][4]: 12

Address[4][1]: 6422256 Value[4][1]: 13

Address[4][2]: 6422260 Value[4][2]: 14

Address[4][3]: 6422264 Value[4][3]: 15

Address[4][4]: 6422268 Value[4][4]: 16

Program # 8

Aim: Write a program to implement bubble sort

Coding:

// Bubble sort implementation

#include <stdio.h>

#define SIZE 5

#include <conio.h>

void swap(int \*x, int \*y) {

    int temp = \*x;

    \*x = \*y;

    \*y = temp;

}

void main() {

    int a[SIZE];

    printf("Enter %d elements in the array to sort them:\n", SIZE);

    for (int i=0; i<SIZE; i++)  scanf("%d", &a[i]);

    //bubble sort in action

    for (int i=0; i<SIZE-1; i++) { // worst-case complexity: O(n^2)

        for (int j=0; j<SIZE-1-i; j++)

            if (a[j] > a[j+1]) {

                swap(&a[j], &a[j+1]);

            }

    }

    printf("Sorted array: ");

    for (int i=0; i<SIZE; i++)

        printf("%d ", a[i]);

    getch();

}

Output:

Enter 5 elements in the array to sort them:

23 123 4 -324 0

Sorted array: -324 0 4 23 123

Program # 9

Aim: Write a program to implement selection sort

Coding:

// Selection sort implementation in C

#include <stdio.h>

#include <conio.h>

#define SIZE 5

void swap (int \*x, int \*y) {

    int temp = \*x;

    \*x = \*y;

    \*y = temp;

}

void main() {

    int a[SIZE];

    printf("Enter %d elements in the array to sort them:\n", SIZE);

    for (int i=0; i<SIZE; i++) scanf("%d", &a[i]);

    // selection sort in action

    for (int i=0; i<SIZE-1; i++)

        for (int j=i+1; j<SIZE; j++)

            if (a[i] > a[j])

                swap(&a[i], &a[j]);

    //displaying the sorted array

    printf("Sorted array: ");

    for (int i=0; i<SIZE;)

        printf("%d ", a[i++]);

    getch();

}

Output:

Enter 5 elements in the array to sort them:

23 2 -234 -24 0

Sorted array: -234 -24 0 2 23

Program # 10

Aim: Write a program to implement insertion sort

Coding:

// Insertion sort algorithm

#include <stdio.h>

#include <conio.h>

#define SIZE 10

void main() {

            int arr[SIZE];

            printf("Enter %d elements in the array: ", SIZE);

            for (int i=0; i<SIZE; i++) scanf("%d", &arr[i]);

            // insertion sort

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

                int key = arr[i];

                int j = i-1;

                while (j >=0 && key < arr[j]) {

                    arr[j+1] = arr[j];

                    j--;

                }

                arr[j+1] = key;

            }

            // displaying the sorted array

            printf("Sorted array: ");

            for (int i=0; i<SIZE; i++)

                printf("%d ", arr[i]);

        getch();

}

Output:

Enter 10 elements in the array: 92 23 2349 -32 -23 -34 9297 -344 0 -12

Sorted array: -344 -34 -32 -23 -12 0 23 92 2349 9297

Program # 11

Aim: Write a program to implement bubble sort(optimised)

Coding:

// Bubble sort implementation

#include <stdio.h>

#define SIZE 5

#include <conio.h>

void swap(int \*x, int \*y) {

    int temp = \*x;

    \*x = \*y;

    \*y = temp;

}

void main() {

    int a[SIZE];

    printf("Enter %d elements in the array to sort them:\n", SIZE);

    for (int i=0; i<SIZE; i++)  scanf("%d", &a[i]);

    int flag = 0;

    //bubble sort in action

    for (int i=0; i<SIZE-1; i++) { // worst-case complexity: O(n^2)

        for (int j=0; j<SIZE-1-i; j++)

            if (a[j] > a[j+1]) {

                flag = 1;

                swap(&a[j], &a[j+1]);

            }

        if (!flag) {       // best case: O(n)

            printf("Array is already sorted.\n");

            break;

        }

    }

    printf("Sorted array: ");

    for (int i=0; i<SIZE; i++)

        printf("%d ", a[i]);

    getch();

}

Output:

Enter 5 elements in the array to sort them:

122 123 124 125 154

Array is already sorted.

Sorted array: 122 123 124 125 154

Program # 12

Aim: Write a program to implement merge concept with two different sorted list

Coding:

#include <stdio.h>

#include <conio.h>

#define M 5

#define N 6

void disp(int a[], int n) {

    printf("[%d, ", a[0]);

    for (int i=1; i<n-1; i++)

        printf("%d, ", a[i]);

    printf("%d]\n", a[n-1]);

}

void main() {

    int a[M];

    int b[N];

    printf("Enter %d elements in array1(in sorted manner): ", M);

    for  (int i=0; i<M; i++)

        scanf("%d", &a[i]);

    printf("Enter %d elements in array2(in sorted manner): ", N);

    for  (int i=0; i<N; i++)

        scanf("%d", &b[i]);

    int ans[M+N];

    int i=0,j=0,k=0;

    while (i < M && j < N) {

        if (a[i] <= b[j]) {

            ans[k] = a[i];

            i++;

        }else {

            ans[k] = b[j];

            j++;

        }

        k++;

    }

    if (i >= M) {

        for (int t=j; t<N; t++)

            ans[k++] = b[t];

    }

    if (j >= N) {

        for (int t=i; t<M; t++)

            ans[k++] = a[t];

    }

    printf("Array1:\n");

    disp(a, M);

    printf("Array2:\n");

    disp(b, N);

    printf("Resultant Array:\n");

    disp(ans, M+N);

    getch();

}

Output:

Enter 5 elements in array1(in sorted manner): 12 23 34 45 56

Enter 6 elements in array2(in sorted manner): 10 20 30 40 50 60

Array1:

[12, 23, 34, 45, 56]

Array2:

[10, 20, 30, 40, 50, 60]

Resultant Array:

[10, 12, 20, 23, 30, 34, 40, 45, 50, 56, 60]

Program # 13

Aim: Write a program to implement merge sort with one unsorted list

Coding:

/\*Merge Sort Algorithm \*/

#include <stdio.h>

#include <limits.h>

#include <conio.h>

#define SIZE 10

void merge(int [],int ,int ,int );

void mergeSort(int [], int ,int);

void arrDisplay(int [], int );

void main()

{

    int arr[SIZE];

    printf("Enter %d elements in the array:\n", SIZE);

    for (int i=0; i<SIZE; i++)

        scanf("%d", &arr[i]);

    int n = sizeof(arr)/sizeof(int);

    printf("Initial array is\n");

    arrDisplay(arr,n);

    mergeSort(arr,0,n-1);

    printf("Sorted array is\n");

    arrDisplay(arr,n);

    getch();

}

void arrDisplay(int arr[],int n)

{

    for(int i=0;i<n;i++)

        printf("%d ",arr[i]);

    printf("\n");

}

void merge(int arr[],int p,int q,int r)

{

    int n1 = q-p+1;

    int n2 = r-q;

    int a[n1+1],b[n2+1];

    for(int i=0;i<n1;i++)

        a[i] = arr[p+i];

    for(int i=0;i<n2;i++)

        b[i] = arr[q+1+i];

    a[n1] = b[n2] = INT\_MAX;

    int i=0,j=0;

    for(int k=p;k<=r;k++)

    {

        if(a[i] <= b[j])

        {

            arr[k] = a[i];

            i++;

        }

        else

        {

            arr[k] = b[j];

            j++;

        }

    }

}

void mergeSort(int arr[], int p,int r)

{

    if(p<r)

    {

        int q = (p+r)/2;

        mergeSort(arr,p,q);

        mergeSort(arr,q+1,r);

        merge(arr,p,q,r);

    }

}

Output:

Enter 10 elements in the array:

292 230 223 1 234 -23 0 927 6 73

Initial array is

292 230 223 1 234 -23 0 927 6 73

Sorted array is

-23 0 1 6 73 223 230 234 292 927

Program # 14

Aim: Write a program to implement quick sort on a list of 10 elements

Coding:

#include <stdio.h>

#define SIZE 10

int partition (int [], int, int);

void quickSort (int [], int, int);

int main () {

    static int a[SIZE];

    printf("Enter %d elements in the array:\n", SIZE);

    for (int i=0; i<SIZE; i++)

        scanf("%d", &a[i]);

    quickSort (a, 0, SIZE-1);

    printf("Sorted array: [%d", a[0]);

    for (int i=1; i<SIZE; i++)

        printf(", %d", a[i]);

    printf("]\n");

    return 0;

}

int partition (int a[], int p, int q) {

    int pivot = a[p];

    int i = p;

    int j = q;

    while (i <= j) {

        while (a[i] <= pivot)

            i++;

        while (a[j] > pivot)

            j--;

        if (i < j) {

            int temp = a[i];

            a[i] = a[j];

            a[j] = temp;

        }

    }

    a[p] = a[j];

    a[j] = pivot;

    return j;

}

void quickSort (int a[], int p, int q) {

    if (p < q) {

        int j = partition (a, p, q);

        quickSort (a, p, j-1);

        quickSort (a, j+1, q);

    }

}

Output:

Enter 10 elements in the array:

10 9 8 7 6 5 4 3 2 1

Sorted array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Program # 15

Aim: WRITE A PROGRAM TO CONVERT INFIX TO POSTFIX EXPRESSION

Coding:

#include <stdio.h>

#include <conio.h>

char push(char stack[], int \*tos, char value, int n);

char pop(char stack[], int \*tos);

char peep(char stack[], int \*tos);

int isEmpty(int \*tos);

int getPriority(char op);

int isOperand(char c);

int getPriority(char op) {

    if (op == '\*' || op == '/')

        return 2;

    else if (op == '+' || op == '-')

        return 1;

    else if (op == '^')

        return 3;

    else

        return 0;

}

int isOperand(char c) {

    return ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) ? 1 : 0;

}

void main() {

    char exp[21];

    char pfix[21];

    int len;

    printf("Enter the length of the infix expression: ");

    scanf("%d", &len);

    if (len < 21) {

        printf("Enter the infix expression: ");

        for (int i=0; i<len; i++)

            scanf("%c", &exp[i]);

    } else {

        printf("Error!! Ran out of space.");

    }

    printf("Infix Expression: ");

    for (int i=0; i<len; i++)

        printf("%c", exp[i]);

    printf("\n");

    // creating a stack

    char stack[21];

    int tos = -1;

    // scanning the infix expression character-by-character

    int j=0;

    int i=0;

    while (j < len) {

        char c = exp[j];

        if (isOperand(c)) {

            // add it to the postfix exp directly

            //printf("%c", c);

            pfix[i] = c;

            i++;

        }else if (c == ')') {

            while (1) {

                char popVal = peep(stack, &tos);

                if (popVal == '(') {

                    pop(stack, &tos);

                    break;

                }else {

                    pfix[i] = pop(stack, &tos);

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

                    i++;

                }

            }

        }else {

            if (isEmpty(&tos))

                push(stack, &tos, c, 21);

            else if (c == '(')

                push(stack, &tos, c, 21);

            else if (c == '+' || c == '-' || c == '^' || c == '/' || c == '\*'){

                int prty = getPriority(c);

                while (1) {

                    if (isEmpty(&tos))

                        break;

                    int prtyStack = getPriority(peep(stack, &tos));

                    if (prty > prtyStack) {

                        push(stack, &tos, c, 21);

                        break;

                    }

                    else

                    {

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

                        pfix[i] = pop(stack, &tos);

                        i++;

                    }

                }

            }else

            {

                push(stack, &tos, c, 21);

            }

        }

        j++;

    }

    while (!isEmpty(&tos)) {

        if (peep(stack, &tos) != '(') {

            pfix[i] = pop(stack, &tos);

            //printf("%c", pfix[i]);

            i++;

        }

    }

    printf("\nPostfix expression: ");

    for (int j=0; j<i; j++)

        printf("%c", pfix[j]);

    getch();

}

char push(char stack[], int \*tos, char value, int n) {

    if (\*tos == n-1) {

        printf("Stack Overflow.\n");

        return 0;

    }

    \*tos = \*tos + 1;

    stack[\*tos] = value;

    return stack[\*tos];

}

char pop(char stack[], int \*tos) {

    if (\*tos == -1) {

        printf("Stack Underflow.\n");

        return '\0';

    }

    int value = stack[\*tos];

    \*tos = \*tos - 1;

    return value;

}

char peep(char stack[], int \*tos) {

    if (\*tos == -1)

        return '\0';

    else

        return stack[\*tos];

}

int isEmpty(int \*tos) {

    return (\*tos == -1) ? 1 : 0;

}

Output:

Enter the length of the infix expression: 18

Enter the infix expression: (a+b)\*(c-d)/(e+f) -> (9+2)\*(4-5)/(6+7)

Infix Expression:

(a+b)\*(c-d)/(e+f)

Postfix expression: ab+cd-\*ef+/

Program # 16

Aim: WRITE A PROGRAM TO EVALUATE A POSTFIX EXPRESSION

Coding:

#include <stdio.h>

#include <string.h>

#define N 80

int push(int stack[], int \*tos, int value, int n) {

    if (\*tos == n-1) {

        printf("Stack Overflow.\n");

        return 0;

    }

    \*tos = \*tos + 1;

    stack[\*tos] = value;

    return stack[\*tos];

}

int pop(int stack[], int \*tos) {

    if (\*tos == -1) {

        printf("Stack Underflow.\n");

        return 0;

    }

    int value = stack[\*tos];

    \*tos = \*tos - 1;

    return value;

}

int peep(int stack[], int \*tos) {

    if (\*tos == -1)

        return 0;

    else

        return stack[\*tos];

}

int isEmpty(int \*tos) {

    return (\*tos == -1) ? 1 : 0;

}

void main() {

    char pfix[N];

    // printf("Enter the size of the postfix expression: ");

    // scanf("%d", &len);

    printf("Enter the postfix expression: ");

    scanf("%s", &pfix);

    int len;

    len = strlen(pfix);

    int stk[len];

    int tos = -1;

    int i=0;

    while (i < len) {

        char c = pfix[i];

        if (c >= '0' && c <= '9') {

            int v = c-48;

            // printf("%d", c-48);

            push(stk, &tos, v, len);

        }

        else if (c == '+' || c == '-' || c == '\*' || c == '/') {

            int op1 = pop(stk, &tos);

            // printf("\nnt: %d", op1);

            int op2 = pop(stk, &tos);

            // printf("\nnt: %d", op2);

            int res;

            switch(c) {

                case '+':

                    res = (op2 + op1);

                break;

                case '-':

                    res = (op2 - op1);

                break;

                case '/':

                    res = (op2 / op1);

                break;

                case '\*':

                    res = (op2 \* op1);

                break;

            }

            // printf("%d", res);

            push(stk, &tos, res, len);

        }

        i++;

    }

    printf("%d", pop(stk, &tos));

}

Output:

Enter the postfix expression: 93+45-\*60+/

-2

Program # 17

Aim: Write a program to count the parenthesis used in an expression

Coding:

#include <stdio.h>

#define N 80

char push(char stack[], int \*tos, char value, int n) {

    if (\*tos == n-1) {

        printf("Stack Overflow.\n");

        return 0;

    }

    \*tos = \*tos + 1;

    stack[\*tos] = value;

    return stack[\*tos];

}

char pop(char stack[], int \*tos) {

    if (\*tos == -1) {

        printf("Stack Underflow.\n");

        return '\0';

    }

    int value = stack[\*tos];

    \*tos = \*tos - 1;

    return value;

}

char peep(char stack[], int \*tos) {

    if (\*tos == -1)

        return '\0';

    else

        return stack[\*tos];

}

int isEmpty(int \*tos) {

    return (\*tos == -1) ? 1 : 0;

}

void main() {

    char symbol[N];

    printf("Enter the symbols: ");

    scanf("%s", &symbol);

    char stack[sizeof(symbol)/sizeof(char)];

    int tos = -1;

    int i=0;

    int flag = 1;

    while (i < sizeof(symbol)/sizeof(char)) {

        char c = symbol[i];

        if (isEmpty(&tos)) {

            if (c == ')') {

                flag = 0;

                break;

            }

            else if (c == '(')

                push(stack, &tos, c, sizeof(symbol)/sizeof(char));

        }else

        {

            if (c == ')') {

                pop(stack, &tos);

            }

            else if (c == '(')

                push(stack, &tos, c, sizeof(symbol)/sizeof(char));

        }

        i++;

    }

    if (isEmpty(&tos) && flag == 1)

        printf("Balanced symbols.\n");

    else

        printf("Unbalanced symbols.\n");

}

Output:

Enter the symbols: ()()(())

Balanced symbols.

Enter the symbols: ()((())())(

Unbalanced symbols.

Program # 18

Aim: Write a program to implement linear queue with all necessary operations

Coding:

#include <stdio.h>

#include <conio.h>

#include <limits.h>

void enQueue(int [], int \*, int , int );

int deQueue(int [], int \*, int \*);

void main() {

    int size = 6;

    int Q[size];

    int front = -1, rear = -1;

    int choice;

    do {

        printf("Enter 1 for enqueue\nEnter 2 for dequeue\nEnter 0 to exit\n");

        scanf("%d", &choice);

        switch (choice) {

            case 1: {

                int value;

                printf("Enter value: ");

                scanf("%d", &value);

                enQueue(Q, &rear, value, size);

                break;

            }

            case 2: {

                deQueue(Q, &rear, &front);

                break;

            }

        }

    }while(choice);

    getch();

}

void enQueue(int Q[], int \*rear, int value, int size) {

    if (\*rear == size-1) {

        printf("Queue is already full\n");

        return;

    }

    \*rear = \*rear + 1;

    Q[\*rear] = value;

    printf("%d enqueued to linear queue\n", Q[\*rear]);

}

int deQueue(int Q[], int \*rear, int \*front) {

    if (\*rear == \*front) {

        printf("Queue is empty\n");

        return INT\_MIN;

    }

    \*front = \*front + 1;

    printf("%d dequeued from the queue\n", Q[\*front]);

    return Q[\*front];

}

Output:

Enter 1 for enqueue

Enter 2 for dequeue

Enter 0 to exit

1

Enter value: 1

1 enqueued to linear queue

Enter 1 for enqueue

Enter 2 for dequeue

Enter 0 to exit

1

Enter value: 2

2 enqueued to linear queue

Enter 1 for enqueue

Enter 2 for dequeue

Enter 0 to exit

2

Enter 2 for dequeue

Enter 0 to exit

2

2 dequeued from the queue

Enter 1 for enqueue

Enter 2 for dequeue

Enter 0 to exit

2

Queue is empty

Enter 1 for enqueue

Enter 2 for dequeue

Enter 0 to exit

0

Program # 19

Aim: Write a program to implement Linear Linked List with all necessary operations

Coding:

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node \*next;

};

void traversal(struct Node\*\* head);

void addNode(struct Node\*\* head, int data);

void deleteNode(struct Node\*\* head);

void traversal(struct Node\*\* head) {

    struct Node\* temp = \*head;

    while(temp) {

        printf("%d ", temp->data);

        temp = temp->next;

    }

}

void addNode(struct Node\*\* head, int data) {

    struct Node\* toAdd = (struct Node\*)malloc(sizeof(struct Node));

    if (toAdd == NULL) {

        printf("Memory not allocated\n");

        return;

    }else {

        toAdd->data = data;

        toAdd->next = NULL;

        if (\*head == NULL) {

            \*head = toAdd;

            return;

        }

        struct Node\* temp = \*head;

        while(temp->next)

            temp = temp->next;

        temp->next = toAdd;

    }

}

void deleteNode(struct Node\*\* head) {

    if (\*head == NULL) {

        printf("Linked-list is empty\n");

    }else {

        if ((\*head)->next == NULL) {

            \*head = NULL;

        }else {

            struct Node\* prev=NULL;

            struct Node\* temp=\*head;

            while(temp->next) {

                prev = temp;

                temp = temp->next;

            }

            prev->next = NULL;

        }

    }

}

void main() {

    struct Node\* head = (struct Node\*)malloc(sizeof(struct Node));

    if (head == NULL) {

        printf("Memory not allocated\n");

        exit(0);

    }else {

        int data;

        printf("Enter head node data: ");

        scanf("%d", &data);

        head->next = NULL;

        head->data = data;

    }

    int choice;

    do {

        printf("Enter 1 for adding a node\n");

        printf("Enter 2 for deleting a node from the end\n");

        printf("Enter 3 for traversal\n");

        printf("Enter 0 for exit\n");

        scanf("%d", &choice);

        switch(choice) {

            case 1: {

                int data;

                printf("Enter node data: ");

                scanf("%d", &data);

                addNode(&head, data);

                break;

            }

            case 2: {

                deleteNode(&head);

                break;

            }

            case 3: {

                traversal(&head);

                break;

            }

        }

        puts("");

    }while(choice);

}

Output:

Enter head node data: 1

Enter 1 for adding a node

Enter 2 for deleting a node from the end

Enter 3 for traversal

Enter 0 for exit

1

Enter node data: 2

Enter 1 for adding a node

Enter 2 for deleting a node from the end

Enter 3 for traversal

Enter 0 for exit

3

1 2

Enter 1 for adding a node

Enter 2 for deleting a node from the end

Enter 3 for traversal

Enter 0 for exit

2

Enter 1 for adding a node

Enter 2 for deleting a node from the end

Enter 3 for traversal

Enter 0 for exit

3

1

Enter 1 for adding a node

Enter 2 for deleting a node from the end

Enter 3 for traversal

Enter 0 for exit

0

Program # 20

Aim: Write a program to implement Doubly Linked List with all necessary operations

Coding:

#include <stdio.h>

#include <stdlib.h>

struct dLL {

    int data;

    struct dLL \*prev;

    struct dLL \*next;

};

//function prototypes

struct dLL\* createDll();

void addHead(struct dLL\*\* start, int data);

void addTail(struct dLL \*\*start, int data);

void addNode(struct dLL \*\*start, int data, int index);

void traversal(struct dLL \*start);

void delNode(struct dLL \*\*start, int index);

// creation of doubly linked-list

struct dLL\* createDll() {

    struct dLL\* node = (struct dLL\*)malloc(sizeof(struct dLL));

    if (node == NULL) {

        printf("Memory not allocated successfully\n");

        return NULL;

    }

    else {

        node = NULL;

        return node;

    }

}

// adding a head node

void addHead(struct dLL\*\* start, int data) {

    struct dLL \*head = (struct dLL\*)malloc(sizeof(struct dLL));

    if (head == NULL)

        printf("Memory not allocated successfully\n");

    else {

        head->data = data;

        head->prev = NULL;

        head->next = NULL;

        \*start = head;

    }

}

// adding a tail node

void addTail(struct dLL \*\*start, int data) {

    struct dLL \*tail = (struct dLL\*)malloc(sizeof(struct dLL));

    if (tail == NULL)

        printf("Memory not allocated successfully\n");

    else {

        tail->data = data;

        tail->next = NULL;

        tail->prev = NULL;

        if ((\*start) == NULL) {

            \*start = tail;

            return;

        }

        // traversing till the end node

        struct dLL\* temp = (\*start);

        while (temp->next != NULL)

             temp = temp->next;

        temp->next = tail;

        tail->prev = temp;

    }

}

// adding a node at a particular position

void addNode(struct dLL \*\*start, int data, int index) {

    struct dLL \*node = (struct dLL\*)malloc(sizeof(struct dLL));

    if (node == NULL)

        printf("Memory not allocated successfully\n");

    else {

        node->data = data;

        node->next = NULL;

        node->prev = NULL;

        if ((\*start) == NULL) {

            \*start = node;

            return;

        }

        if (index == 0) {

            node->next = \*start;

            (\*start)->prev = node;

            \*start = node;

            return;

        }

        // traversing to the position node (0-based indexing)

        struct dLL\* temp = (\*start);

        for (int i=1; i<index && temp->next != NULL; i++)

            temp = temp->next;

        node->next = temp->next;

        node->prev = temp;

        if (temp->next != NULL)

            temp->next->prev = node;

        temp->next = node;

    }

}

// traversal

void traversal(struct dLL \*start) {

    while (start != NULL) {

        printf("%d ", start->data);

        start = start->next;

    }

    printf("\n");

}

// deleting a node at a particular index

void delNode(struct dLL \*\*start, int index) {

    if ((\*start) == NULL) {

        printf("Linked-list is empty\n");

        return;

    }

    if (index == 0) {

        \*start = (\*start)->next;

        return;

    }

    struct dLL \*temp = (\*start), \*node = NULL;

    for (int i=1; i<index; i++) {

        if (temp->next != NULL) {

            if (temp->next->next != NULL)

                temp = temp->next;

            else

                break;

        }else {

            break;

        }

    }

    node = temp->next;

    temp->next = node->next;

    if (node->next != NULL) {

        node->next->prev = temp;

         node->next = NULL;

    }

    node->prev = NULL;

    free(node);

}

void main() {

    int choice, flag = 0;

    struct dLL\* start;

    do {

        printf("Enter 1 to create a doubly linked-list\n");

        printf("Enter 2 to add a node\n");

        printf("Enter 3 to delete a node\n");

        printf("Enter 4 to traverse\n");

        printf("Enter 0 to exit\n");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1: {

            start = createDll();

            flag = 1;

            break;

        }

        case 2: {

            if (flag == 0) {

                printf("Error!! Create a doubly linked-list first...\n");

                break;

            }

            else {

                int option;

                do {

                    printf("Enter 21 to add a head node\n");

                    printf("Enter 22 to add a tail node\n");

                    printf("Enter 23 to add a node at a specific index(0-based indexing)\n");

                    printf("Enter 24 to traverse\n");

                    printf("Enter 20 to exit\n");

                    scanf("%d", &option);

                    switch (option)

                    {

                    case 21: {

                        int data;

                        printf("Enter data: ");

                        scanf("%d", &data);

                        addHead(&start, data);

                        break;

                    }

                    case 22: {

                        int data;

                        printf("Enter data: ");

                        scanf("%d", &data);

                        addTail(&start, data);

                        break;

                    }

                    case 23: {

                        int data, index;

                        printf("Enter data and index(0-based indexing): ");

                        scanf("%d %d", &data, &index);

                        addNode(&start, data, index);

                        break;

                    }

                    case 24: {

                        traversal(start);

                        break;

                    }

                    printf("\n");

                }

            }while(option != 20);

            break;

            }

        }

        case 4: {

            if (flag == 0) {

                printf("Error!! Create a doubly linked-list first...\n");

                break;

            }else {

                traversal(start);

                break;

            }

        }

        case 3: {

            if (flag == 0) {

                printf("Error!! Create a doubly linked-list first...\n");

                break;

            }else {

                int index;

                printf("Enter index(0-based indexing): ");

                scanf("%d", &index);

                delNode(&start, index);

                break;

            }

        }

    }

    printf("\n");

    }while(choice);

}

Output:

Enter 1 to create a doubly linked-list

Enter 2 to add a node

Enter 3 to delete a node

Enter 4 to traverse

Enter 0 to exit

1

Enter 1 to create a doubly linked-list

Enter 2 to add a node

Enter 3 to delete a node

Enter 4 to traverse

Enter 0 to exit

2

Enter 21 to add a head node

Enter 22 to add a tail node

Enter 23 to add a node at a specific index(0-based indexing)

Enter 24 to traverse

Enter 20 to exit

21

Enter data: 1

Enter 21 to add a head node

Enter 22 to add a tail node

Enter 23 to add a node at a specific index(0-based indexing)

Enter 24 to traverse

Enter 20 to exit

24

1

Enter 21 to add a head node

Enter 22 to add a tail node

Enter 23 to add a node at a specific index(0-based indexing)

Enter 24 to traverse

Enter 20 to exit

22

Enter data: 2

Enter 21 to add a head node

Enter 22 to add a tail node

Enter 23 to add a node at a specific index(0-based indexing)

Enter 24 to traverse

Enter 20 to exit

24

1 2

Enter 21 to add a head node

Enter 22 to add a tail node

Enter 23 to add a node at a specific index(0-based indexing)

Enter 24 to traverse

Enter 20 to exit

23

Enter data and index(0-based indexing): 0 0

Enter 21 to add a head node

Enter 22 to add a tail node

Enter 23 to add a node at a specific index(0-based indexing)

Enter 24 to traverse

Enter 20 to exit

24

0 1 2

Enter 21 to add a head node

Enter 22 to add a tail node

Enter 23 to add a node at a specific index(0-based indexing)

Enter 24 to traverse

Enter 20 to exit

22

Enter data: 3

Enter 21 to add a head node

Enter 22 to add a tail node

Enter 23 to add a node at a specific index(0-based indexing)

Enter 24 to traverse

Enter 20 to exit

24

0 1 2 3

Enter 21 to add a head node

Enter 22 to add a tail node

Enter 23 to add a node at a specific index(0-based indexing)

Enter 24 to traverse

Enter 20 to exit

20

Enter 1 to create a doubly linked-list

Enter 2 to add a node

Enter 3 to delete a node

Enter 4 to traverse

Enter 0 to exit

3

Enter index(0-based indexing): 0

Enter 1 to create a doubly linked-list

Enter 2 to add a node

Enter 3 to delete a node

Enter 4 to traverse

Enter 0 to exit

4

1 2 3

Enter 1 to create a doubly linked-list

Enter 2 to add a node

Enter 3 to delete a node

Enter 4 to traverse

Enter 0 to exit

3

Enter index(0-based indexing): 0

Enter 1 to create a doubly linked-list

Enter 2 to add a node

Enter 3 to delete a node

Enter 4 to traverse

Enter 0 to exit

4

2 3

Enter 1 to create a doubly linked-list

Enter 2 to add a node

Enter 3 to delete a node

Enter 4 to traverse

Enter 0 to exit

0

Program # 21

Aim: Write a program to implement Stack using Linked List with all necessary operations

Coding:

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

//creating a structure for creating a stack

struct stack {

    int data;

    struct stack\* next;

};

//function prototype

struct stack\* createStack();

void push(struct stack\*\* head, int data);

int pop(struct stack\*\* head);

//function for creating stack

struct stack\* createStack() {

    struct stack\* temp = (struct stack\*)malloc(sizeof(struct stack));

    if (temp == NULL)

        printf("Memory not allocated successfully\n");

    else {

        temp = NULL;

        return temp;

    }

}

//push functionality

void push(struct stack\*\* head, int data) {

    //creating a stack

    struct stack\* stack = (struct stack\*)malloc(sizeof(struct stack));

 stack->data = data;

 stack->next = NULL;

    if ((\*head) == NULL) {

        \*head = stack;

    }else {

     stack->next = \*head;

        \*head = stack;

    }

}

//pop functionality

int pop(struct stack\*\* head) {

    //underflow condition

    if (\*head == NULL) {

        printf("Stack Underflow\n");

        return INT\_MIN;

    }

    int data = (\*head)->data;

    \*head = (\*head)->next;

    return data;

}

//element at the top of the stack

int tos(struct stack\*\* head) {

    if (\*head == NULL) {

        printf("Stack Underflow\n");

        return INT\_MIN;

    }

    return (\*head)->data;

}

//driver method

void main() {

    int choice, flag = 0;

    struct stack\* S = NULL;

    do {

        printf("Enter 1 to create a stack\n");

        printf("Enter 2 to add an element onto the top of stack\n");

        printf("Enter 3 to pop out the element at the top of the stack\n");

        printf("Enter 4 to get the element at the top of the stack\n");

        printf("Enter 0 to exit\n");

        scanf("%d", &choice);

        switch(choice) {

            case 1: {

                S = createStack();

                flag = 1;

                break;

            }

            case 2: {

                if (flag == 0) {

                    printf("ERROR!! Create stack first\n");

                }else {

                    int data;

                    printf("Enter data: ");

                    scanf("%d", &data);

                    push(&S, data);

                }

                break;

            }

            case 3: {

                if (flag == 0) {

                    printf("ERROR!! Create stack first\n");

                }else {

                    int value = pop(&S);

                    if (value != INT\_MIN)

                        printf("%d\n", value);

                }

                break;

            }

            case 4: {

                if (flag == 0) {

                    printf("ERROR!! Create stack first\n");

                }else {

                    int value = tos(&S);

                    if(value != INT\_MIN)

                        printf("%d\n", value);

                }

                break;

            }

        }

    }while(choice);

}

Output:

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

1

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

2

Enter data: 1

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

2

Enter data: 2

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

4

2

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

2

Enter data: 3

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

4

3

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

3

3

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

4

2

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

3

2

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

4

1

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

3

1

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

4

Stack Underflow

Enter 1 to create a stack

Enter 2 to add an element onto the top of stack

Enter 3 to pop out the element at the top of the stack

Enter 4 to get the element at the top of the stack

Enter 0 to exit

0

Program # 22

Aim: Write a program to implement Queue using Linked List with all necessary operations

Coding:

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

struct Q{

    int data;

    struct Q\* next;

};

//function prototypes

struct Q\* createQ();

void nQ(int);

int dQ();

// queue specific variables

struct Q\* front=NULL;

struct Q\* rear=NULL;

struct Q\* createQ() {

    struct Q\* node = (struct Q\*)malloc(sizeof(struct Q));

    if (node == NULL)

        printf("Memory not allocated successfully\n");

    else

        node = NULL;

    return node;

}

void nQ(int data) {

    struct Q\* node = (struct Q\*)malloc(sizeof(struct Q));

    node->next = NULL;

    node->data = data;

    if (front == NULL)

        front = rear = node;

    else {

        rear->next = node;

        rear = rear->next;

    }

}

int dQ() {

    // underflow condition

    if (front == NULL) {

        printf("Queue underflow\n");

        return INT\_MIN;

    }

    int data = front->data;

    if (front == rear)

        front = rear = NULL;

    else

        front = front->next;

    return data;

}

//driver method

void main() {

    int choice, flag = 0;

    struct Q\* S = NULL;

    do {

        printf("Enter 1 to create a queue\n");

        printf("Enter 2 to enque\n");

        printf("Enter 3 to deque\n");

        printf("Enter 0 to exit\n");

        scanf("%d", &choice);

        switch(choice) {

            case 1: {

                S = createQ();

                flag = 1;

                break;

            }

            case 2: {

                if (flag == 0) {

                    printf("ERROR!! Create queue first\n");

                }else {

                    int data;

                    printf("Enter data: ");

                    scanf("%d", &data);

                    nQ(data);

                }

                break;

            }

            case 3: {

                if (flag == 0) {

                    printf("ERROR!! Create queue first\n");

                }else {

                    int value = dQ();

                    if (value != INT\_MIN)

                        printf("%d\n", value);

                }

                break;

            }

        }

    }while(choice);

}

Output:

Enter 1 to create a queue

Enter 2 to enque

Enter 3 to deque

Enter 0 to exit

1

Enter 1 to create a queue

Enter 2 to enque

Enter 3 to deque

Enter 0 to exit

2

Enter data: 1

Enter 1 to create a queue

Enter 2 to enque

Enter 3 to deque

Enter 0 to exit

2

Enter data: 2

Enter 1 to create a queue

Enter 2 to enque

Enter 3 to deque

Enter 0 to exit

3

1

Enter 1 to create a queue

Enter 2 to enque

Enter 3 to deque

Enter 0 to exit

3

2

Enter 1 to create a queue

Enter 2 to enque

Enter 3 to deque

Enter 0 to exit

3

Queue underflow

Enter 1 to create a queue

Enter 2 to enque

Enter 3 to deque

Enter 0 to exit

0

Program # 23

Aim: Write a program to implement binary tree with all necessary operations.

Coding:

#include <stdio.h>

#include <conio.h>

#include <stdbool.h>

#include <stdlib.h>

struct QueueNode

{

    struct Node \*val;

    struct QueueNode \*next;

}\*front=NULL,\*rear=NULL;

bool isEmpty()

{

    if(front) return false;

    return true;

}

void enqueue(struct Node\* element)

{

    if(rear)

    {

        struct QueueNode \*temp=(struct QueueNode\*)malloc(sizeof(struct QueueNode));

        temp->val=element;

        temp->next=NULL;

        rear->next=temp;

        rear=temp;

    }

    else

    {

        struct QueueNode \*temp=(struct QueueNode\*)malloc(sizeof(struct QueueNode));

        temp->val=element;

        temp->next=NULL;

        front=temp;

        rear=temp;

    }

}

struct Node\* dequeue()

{

    struct QueueNode \*ptr=front;

    front=front->next;

    ptr->next=NULL;

    struct Node \*data=ptr->val;

    free(ptr);

    if(!front)

    {

        front=NULL;

        rear=NULL;

    }

    return data;

}

void displayQueue()

{

    struct QueueNode \*temp=front;

    int count=0;

    while(temp)

    {

        ++count;

        temp=temp->next;

    }

    printf("%d",count);

}

struct Node

{

    int value;

    struct Node \*left;

    struct Node \*right;

}\*root=NULL;

void createTree()

{

    int first,data,y;

    printf("\nEnter root value of tree\n");

    scanf("%d",&first);

    root=(struct Node\*)malloc(sizeof(struct Node));

    root->value=first;

    root->left=NULL;

    root->right=NULL;

    enqueue(root);

    while(!isEmpty())

    {

        struct Node \*insert=dequeue();

        data=insert->value;

        printf("\nEnter left child of %d\n",data);

        y=0;

        scanf("%d",&y);

        if(y!=-1)

        {

            struct Node \*temp1=(struct Node\*)malloc(sizeof(struct Node));

            temp1->value=y;

            temp1->left=NULL;

            temp1->right=NULL;

            insert->left=temp1;

            enqueue(temp1);

        }

        printf("\nEnter right child of %d\n",data);

        y=0;

        scanf("%d",&y);

        if(y!=-1)

        {

            struct Node \*temp=(struct Node\*)malloc(sizeof(struct Node));

            temp->value=y;

            temp->left=NULL;

            temp->right=NULL;

            insert->right=temp;

            enqueue(temp);

        }

    }

}

void inorder(struct Node \*initial)

{

    if(initial)

    {

        inorder(initial->left);

        printf("%d ",initial->value);

        inorder(initial->right);

    }

}

void preorder(struct Node \*initial)

{

    if(initial)

    {

        printf("%d ",initial->value);

        preorder(initial->left);

        preorder(initial->right);

    }

}

void postorder(struct Node \*initial)

{

    if(initial)

    {

        postorder(initial->left);

        postorder(initial->right);

        printf("%d ",initial->value);

    }

}

int flag;

void search(struct Node \*initial,int key)

{

    if(initial)

    {

        search(initial->left,key);

        search(initial->right,key);

        if(initial->value==key){

            printf("%d is found in tree!!",key);

            flag=1;

            return;

        }

    }

}

void deleteTree(struct Node \*start)

{

    if (start == NULL)

     return;

    deleteTree(start->left);

    deleteTree(start->right);

    free(start);

}

void main()

{

    int choice;

    printf("Trees Data Structure Using Linked List\n");

    while(1)

    {

        printf("\nEnter Operation to be performed\n");

        printf("1. Create Tree\n");

        printf("2. Inorder Traversal\n");

        printf("3. Preorder Traversal\n");

        printf("4. Postorder Traversal\n");

        printf("5. Search\n");

        printf("6. Delete Tree\n");

        printf("7. Exit\n");

        choice=0;

        printf("\n");

        scanf("%d",&choice);

        if(choice==1)

        {   int total;

            if(root)

            {

                printf("\nError! Tree is already created\n");

            }

            else

            {   printf("Enter total number of nodes:");

                scanf("%d",&total);

                createTree();

                printf("\nYour Tree has been created successfully\n");

            }

        }

        else if(choice==2)

        {

            if(root)

            {

                printf("\nYour Inorder Traversal of tree is: \n");

                inorder(root);

                printf("\n");

            }

            else

            {

                printf("\nError! Create Tree first\n");

            }

        }

        else if(choice==3)

        {

            if(root)

            {

                printf("\nYour Preorder Traversal of tree is: \n");

                preorder(root);

                printf("\n");

            }

            else

            {

                printf("\nError! Create Tree first\n");

            }

        }

        else if(choice==4)

        {

            if(root)

            {

                printf("\nYour Postorder Traversal of tree is: \n");

                postorder(root);

                printf("\n");

            }

            else

            {

                printf("\nError! Create Tree first\n");

            }

        }

        else if(choice==5)

        {

            if(root)

            {   int key;

                printf("\nEnter value to be searched: \n");

                scanf("%d",&key);

                flag=0;

                search(root,key);

                if(flag==0){

                    printf("Element not found!!");

                }

                printf("\n");

            }

            else

            {

                printf("\nError! Create Tree first\n");

            }

        }

        else if(choice==6)

        {

            if(root)

            {

                deleteTree(root);

                printf("Tree Deleted");

                root=NULL;

                printf("\n");

            }

            else

            {

                printf("\nError! Create Tree first\n");

            }

        }

        else

        {

            exit(0);

        }

    }

    getch();

}

Output:

Trees Data Structure Using Linked List

Enter Operation to be performed

1. Create Tree

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Search

6. Delete Tree

7. Exit

1

Enter total number of nodes:5

Enter root value of tree

1

Enter left child of 1

2

Enter right child of 1

3

Enter left child of 2

4

Enter right child of 2

5

Enter left child of 3

-1

Enter right child of 3

-1

Enter left child of 4

-1

Enter right child of 4

-1

Enter left child of 5

-1

Enter right child of 5

-1

Your Tree has been created successfully

Enter Operation to be performed

1. Create Tree

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Search

6. Delete Tree

7. Exit

2

Your Inorder Traversal of tree is:

4 2 5 1 3

Enter Operation to be performed

1. Create Tree

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Search

6. Delete Tree

7. Exit

3

Your Preorder Traversal of tree is:

1 2 4 5 3

Enter Operation to be performed

1. Create Tree

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Search

6. Delete Tree

7. Exit

4

Your Postorder Traversal of tree is:

4 5 2 3 1

Enter Operation to be performed

1. Create Tree

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Search

6. Delete Tree

7. Exit

5

Enter value to be searched:

1

1 is found in tree!!

Enter Operation to be performed

1. Create Tree

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Search

6. Delete Tree

7. Exit

5

Enter value to be searched:

-1

Element not found!!

Enter Operation to be performed

1. Create Tree

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Search

6. Delete Tree

7. Exit

6

Tree Deleted

Enter Operation to be performed

1. Create Tree

2. Inorder Traversal

3. Preorder Traversal

4. Postorder Traversal

5. Search

6. Delete Tree

7. Exit

7