**Practical 1:- Revision of Array**

1. Program For Linear Search

**Code**

// linear search

#include <stdio.h>

int main()

{

    int a[20], i, n, search;

    printf("Enter the number of elements in the array (less than 20): ");

    scanf("%d", &n);

    printf("Enter the elements of the array: \n");

    for (i = 0; i < n; i++)

    {

        scanf("%d", &a[i]);

    }

    printf("Enter the element to be searched: ");

    scanf("%d", &search);

    for (i = 0; i < n; i++)

    {

        if (a[i] == search)

        {

            printf("Element found at position %d\n", i + 1);

            break;

        }

    }

    if (i == n)

    {

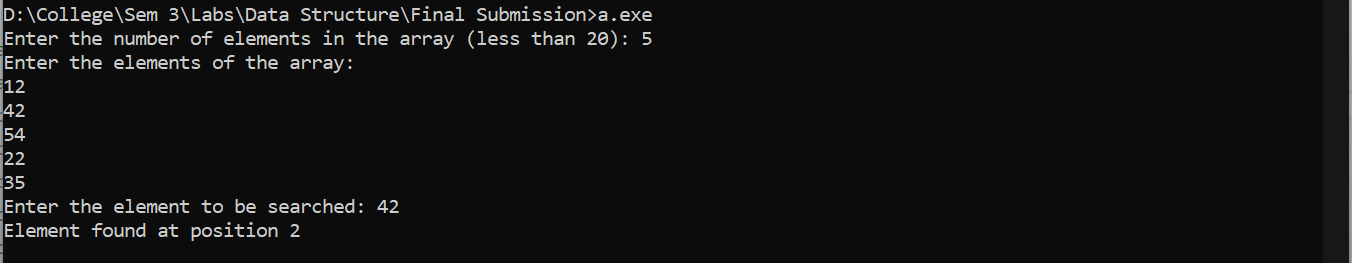
        printf("Element not found\n");

    }

    return 0;

}

**Input / Output**

****

1. Write a program in C to perform bubble sort

**Code**

#include <stdio.h>

int main()

{

    int n, i, j, temp;

    printf("Enter the size of the arrays:\n");

    scanf("%d", &n);

    int a[n];

    printf("Enter the elements of the array:\n");

    for (i = 0; i < n; i++)

    {

        scanf("%d", &a[i]);

    }

    printf("\n");

    for (i = 0; i < n; i++)

    {

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

        {

            if (a[j] > a[j + 1])

            {

                temp = a[j];

                a[j] = a[j + 1];

                a[j + 1] = temp;

            }

        }

    }

    printf("The sorted array is: ");

    for (i = 0; i < n; i++)

    {

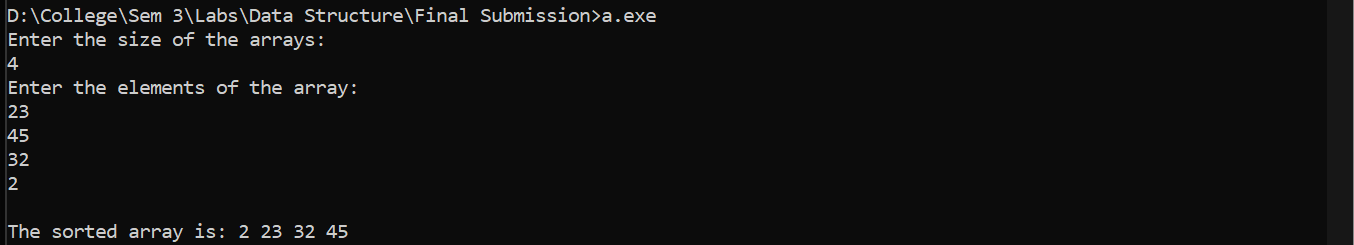
        printf("%d ", a[i]);

    }

    return 0;

}

**Input/Output**

****

1. Write a Program in C to perform selection short

**Code**

//insertion short

#include <stdio.h>

int main()

{

    int n, i, j, temp;

    printf("Enter the size of the arrays:\n");

    scanf("%d", &n);

    int a[n];

    printf("Enter the elements of the array:\n");

    for (i = 0; i < n; i++)

    {

        scanf("%d", &a[i]);

    }

    printf("\n");

    for (i = 1; i < n; i++)

    {

        temp = a[i];

        j = i - 1;

        while (temp < a[j] && j >= 0)

        {

            a[j + 1] = a[j];

            j--;

        }

        a[j + 1] = temp;

    }

    printf("The sorted array is: \n");

    for (i = 0; i < n; i++)

    {

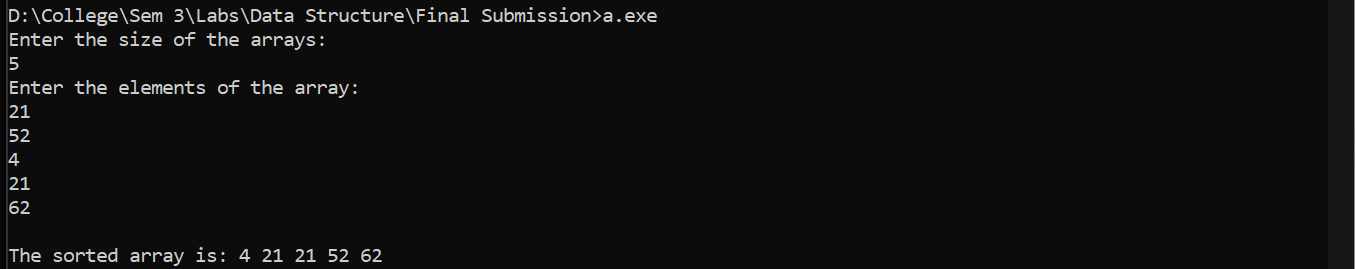
        printf("%d ", a[i]);

    }

    return 0;

}

**Input/Output**

****

1. Write a C program for Selection Sort

**Code**

//selection short

#include <stdio.h>

int main()

{

    int n, i, j, temp;

    printf("Enter the size of the arrays:\n");

    scanf("%d", &n);

    int a[n];

    printf("Enter the elements of the array:\n");

    for (i = 0; i < n; i++)

    {

        scanf("%d", &a[i]);

    }

    printf("\n");

    for (i = 0; i < n; i++)

    {

        for (j = i + 1; j < n; j++)

        {

            if (a[i] > a[j])

            {

                temp = a[i];

                a[i] = a[j];

                a[j] = temp;

            }

        }

    }

    printf("The sorted array is: ");

    for (i = 0; i < n; i++)

    {

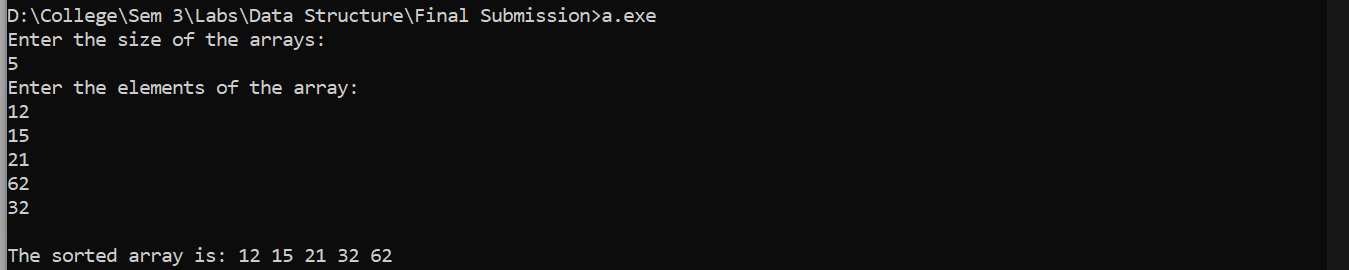
        printf("%d ", a[i]);

    }

    return 0;

}

**Input/Output**

****

1. Find Maximum Minimum Element is array

**Code**

#include <stdio.h>

int max(int a[], int n)

{

    int max = a[0];

    for (int i = 1; i < n; i++)

    {

        if (a[i] > max)

        {

            max = a[i];

        }

    }

    return max;

}

int min(int a[], int n)

{

    int min = a[0];

    for (int i = 1; i < n; i++)

    {

        if (a[i] < min)

        {

            min = a[i];

        }

    }

    return min;

}

int main()

{

    int n=10;

    int a[10] = {23,53,67,32,7,82,43,83,92,12};

    printf("The array is: ");

    for (int i = 0; i < 10; i++)

    {

        printf("%d ", a[i]);

    }

    int maximum= max(a, n);

    int minimum= min(a,n);

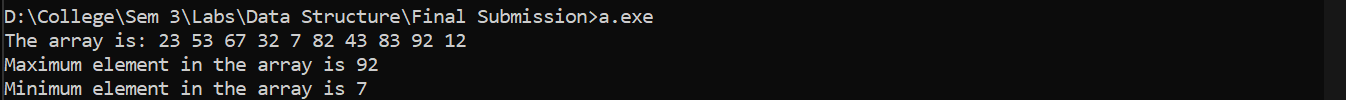
    printf("\nMaximum element in the array is %d\n",maximum);

    printf("Minimum element in the array is %d\n",minimum);

    return 0;

}

**Input/Output**

****

1. Find Second largest and Second Smallest element in the array

**Code**

// second maximum element and second minimum element in the array

#include <stdio.h>

int max(int a[], int n)

{

    int max = a[0];

    for (int i = 1; i < n; i++)

    {

        if (a[i] > max)

        {

            max = a[i];

        }

    }

    return max;

}

int min(int a[], int n)

{

    int min = a[0];

    for (int i = 1; i < n; i++)

    {

        if (a[i] < min)

        {

            min = a[i];

        }

    }

    return min;

}

int second\_max(int a[], int n)

{

    int max = a[0];

    int second\_max = a[0];

    for (int i = 1; i < n; i++)

    {

        if (a[i] > max)

        {

            second\_max = max;

            max = a[i];

        }

        else if (a[i] > second\_max)

        {

            second\_max = a[i];

        }

    }

    return second\_max;

}

// function for finding second minimum element in the array

int second\_min(int a[], int n)

{

    int min = a[0];

    int second\_min = a[0];

    for (int i = 1; i < n; i++)

    {

        if (a[i] < min)

        {

            second\_min = min;

            min = a[i];

        }

        else if (a[i] < second\_min)

        {

            second\_min = a[i];

        }

    }

    return second\_min;

}

int main()

{

    int n = 10;

    int a[10] = {23, 53, 67, 32, 7, 82, 43, 83, 92, 12};

    printf("The array is: ");

    for (int i = 0; i < 10; i++)

    {

        printf("%d ", a[i]);

    }

    int maximum = max(a, n);

    int minimum = min(a, n);

    int second\_maximum = second\_max(a, n);

    int second\_minimum = second\_min(a, n);

    printf("\nMaximum element in the array is %d\n", maximum);

    printf("Minimum element in the array is %d\n", minimum);

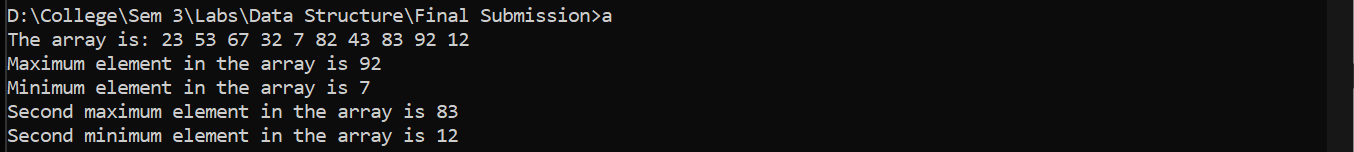
    printf("Second maximum element in the array is %d\n", second\_maximum);

    printf("Second minimum element in the array is %d\n", second\_minimum);

    return 0;

}

**Input/Output**

****

**Practical 2:- Revision of Structure**

1. Create the Student Struct

**Code**

#include <stdio.h>

struct Student{

    char name[20];

    int roll;

    char address[50];

};

int main(){

    struct Student s2[5];

    for(int i = 0; i < 5; i++){

        printf("Enter the name of the student: ");

        scanf("%s", s2[i].name);

        printf("Enter the roll number of the student: ");

        scanf("%d", &s2[i].roll);

        printf("Enter the address of the student: ");

        scanf("%s", s2[i].address);

    }

    for(int i = 0; i < 5; i++){

        printf("The name of the student is: %s\n", s2[i].name);

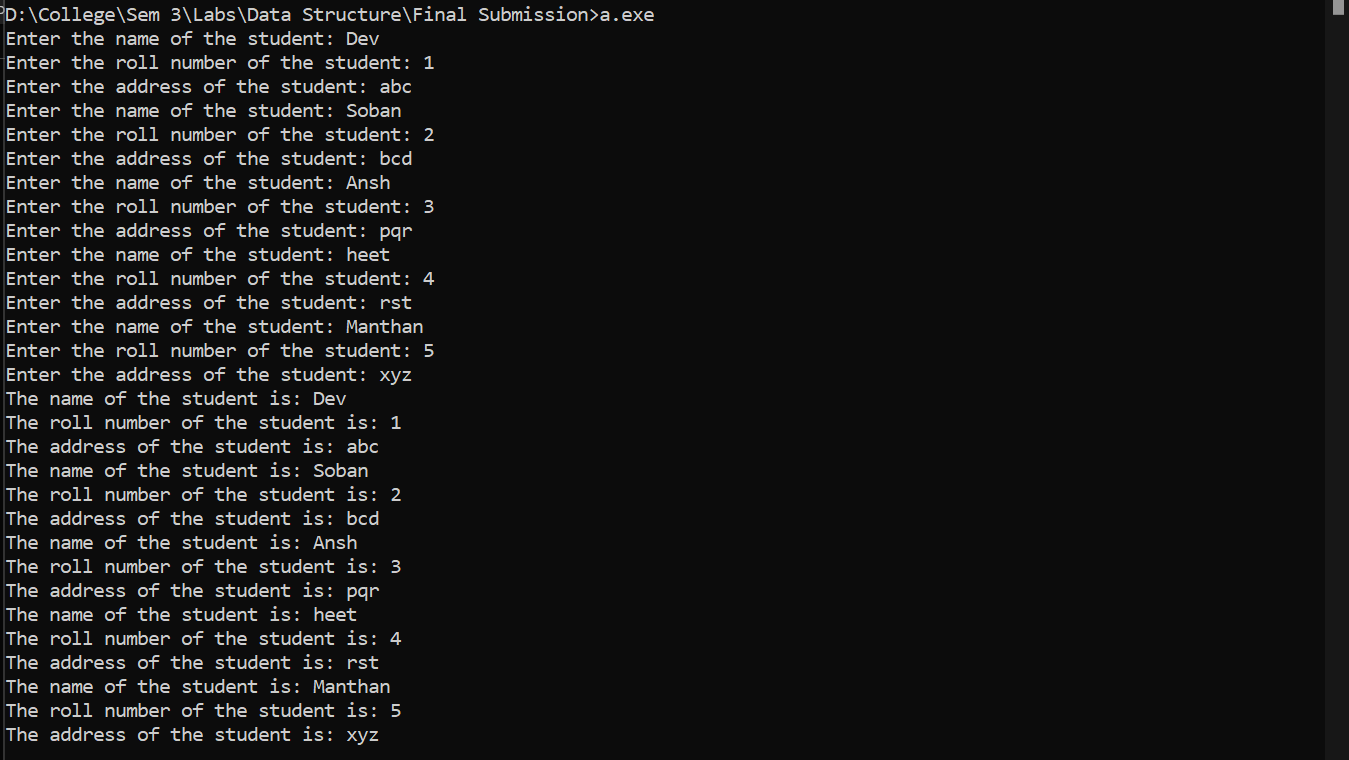
        printf("The roll number of the student is: %d\n", s2[i].roll);

        printf("The address of the student is: %s\n", s2[i].address);

    }

}

**Input/output**

****

1. Create a structure Organization with organization name and organization ID as its data members. Next, create another structure Employee that is nested in structure Organization with employee ID, employee salary and employee name as its data members. Write a program in such a way that there are two organizations and each of these contains two employees

**Code**

#include <stdio.h>

struct Organization{

    char name[20];

    int id;

    struct Employee{

        int id;

        int salary;

        char name[20];

    }Employee[2];

};

int main(){

    struct Organization o1;

    printf("Enter the name of the organization: ");

    scanf("%s", o1.name);

    printf("Enter the ID of the organization: ");

    scanf("%d", &o1.id);

    for(int i = 0; i < 2; i++){

        printf("Enter the name of the employee: ");

        scanf("%s", o1.Employee[i].name);

        printf("Enter the ID of the employee: ");

        scanf("%d", &o1.Employee[i].id);

        printf("Enter the salary of the employee: ");

        scanf("%d", &o1.Employee[i].salary);

    }

    struct Organization o2;

    printf("Enter the name of the organization: ");

    scanf("%s", o2.name);

    printf("Enter the ID of the organization: ");

    scanf("%d", &o2.id);

    for(int i = 0; i < 2; i++){

        printf("Enter the name of the employee: ");

        scanf("%s", o2.Employee[i].name);

        printf("Enter the ID of the employee: ");

        scanf("%d", &o2.Employee[i].id);

        printf("Enter the salary of the employee: ");

        scanf("%d", &o2.Employee[i].salary);

    }

    printf("The name of the organization is: %s\n", o1.name);

    printf("The ID of the organization is: %d\n", o1.id);

    for(int i = 0; i < 2; i++){

        printf("The name of the employee is: %s\n", o1.Employee[i].name);

        printf("The ID of the employee is: %d\n", o1.Employee[i].id);

        printf("The salary of the employee is: %d\n", o1.Employee[i].salary);

    }

    printf("The name of the organization is: %s\n", o2.name);

    printf("The ID of the organization is: %d\n", o2.id);

    for(int i = 0; i < 2; i++){

        printf("The name of the employee is: %s\n", o2.Employee[i].name);

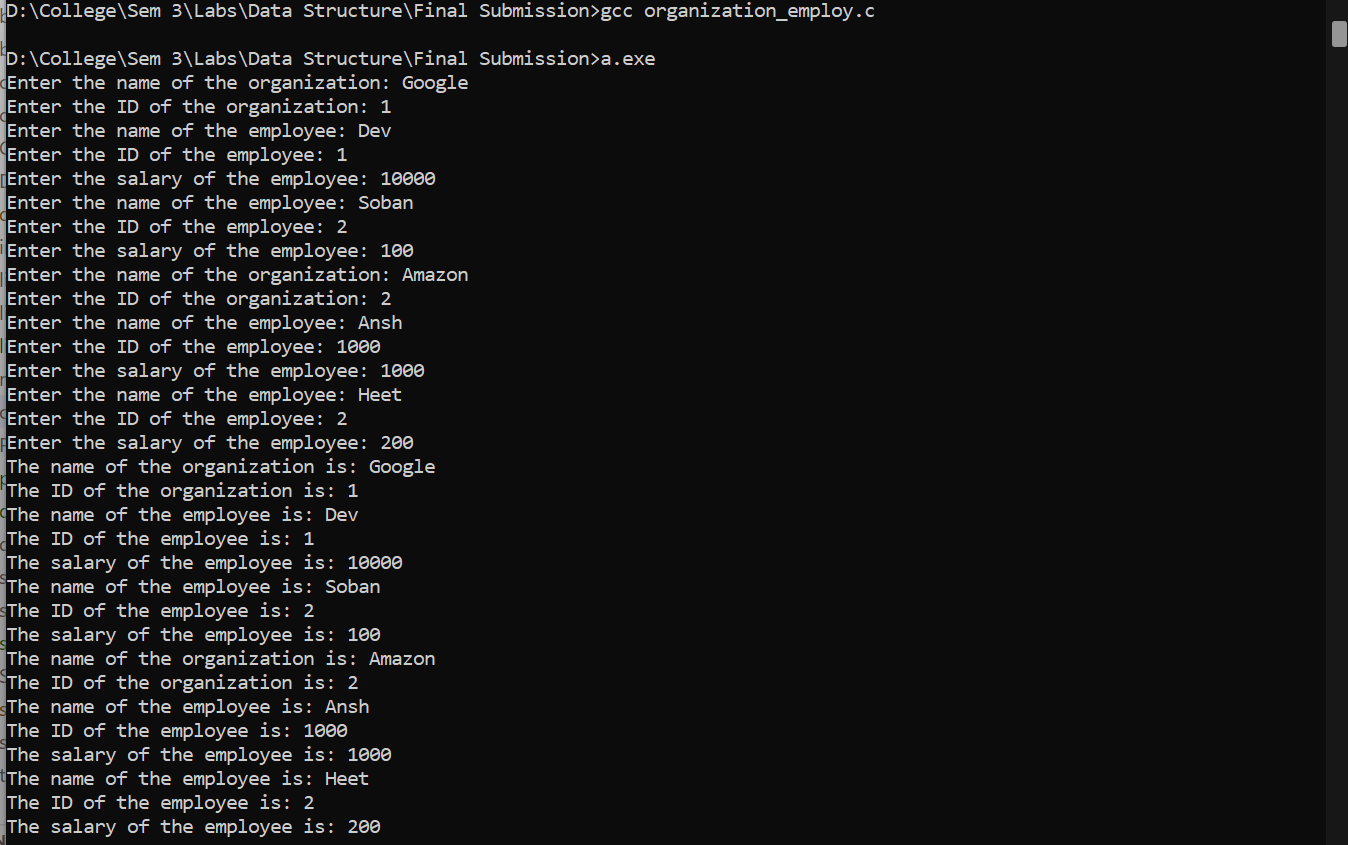
        printf("The ID of the employee is: %d\n", o2.Employee[i].id);

        printf("The salary of the employee is: %d\n", o2.Employee[i].salary);

    }

}

**Input/Ouput**

****

**Practical 2:- Revision of Pointers**

1. Write a program in C to implement arrays of pointers and pointers to arrays.

**Code**

#include <stdio.h>

int main()

{

    // pointer to array

    printf("pointer to array\n");

    int(\*a)[5];

    int b[5] = {1, 2, 3, 4, 5};

    int i = 0;

    a = &b;

    for (i = 0; i < 5; i++)

{

        printf("Value of arr[%d] = %d\n", \*(\*a + i));

}

    //    array of pointer

    printf("\n\n\n Array of pointer");

    int arr[] = {1, 2, 3};

    int \*ptr[3];

    for (i = 0; i < 3; i++)

    {

        ptr[i] = &arr[i];

    }

    for (i = 0; i < 3; i++)

    {

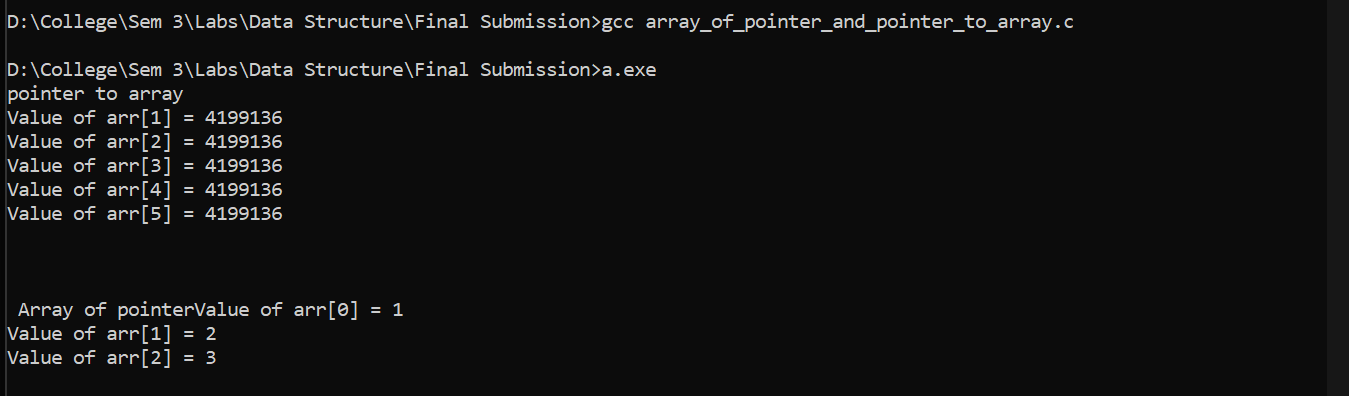
        printf("Value of arr[%d] = %d\n", i, \*ptr[i]);

    }

    return 0;

}

**Input/Output**

****

1. Write a Program to implement pointers to structure

**Code**

#include <stdio.h>

struct student

{

    int rollno;

    int marks;

    char name[20];

};

int main()

{

    struct student Dev;

    struct student \*pointer;

    pointer = &Dev;

    pointer->rollno = 1;

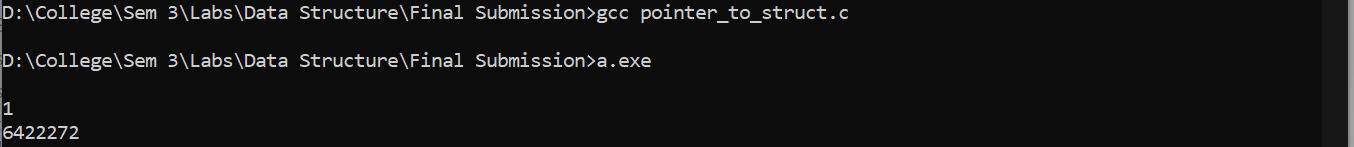
    printf("\n%d", Dev.rollno);

    printf("\n%u" , &Dev.rollno);

    return 0;

}

**Input/Output**

****

1. Write a program in C to perform swapping of two numbers by passing addresses of the variables to the functions

**Code**

#include <stdio.h>

void swap(int \*, int \*);

int main()

{

    int a = 10;

    int b = 20;

    printf("Before in main       a = %d, b = %d\n", a, b);

    swap(&a, &b);

    printf("after in main        a= %d, b = %d\n", a, b);

    return 0;

    }

void swap(int \*a, int \*b)

{

    int temp;

    temp = \*a;

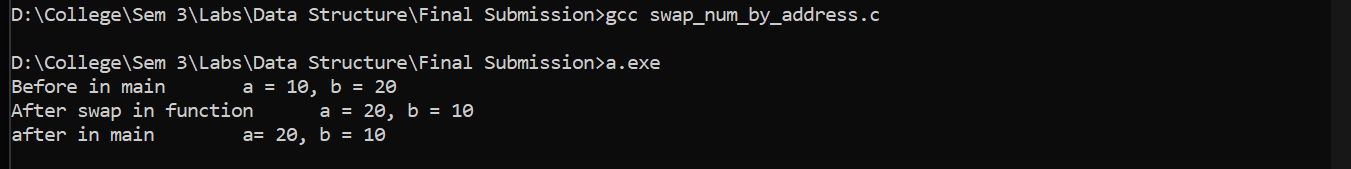
    \*a = \*b;

    \*b = temp;

    printf("After swap in function      a = %d, b = %d\n", \*a, \*b);

}

**Input/Output**

****

**Practical 5: Stack Application**

1. Stack implementation

**Code**

#include<stdio.h>

#include<stdlib.h>

#define MAX 4

int stack\_arr[MAX];

int top=-1;

int isFull(){

    if(top==MAX-1){

        return 1;

    }

    else return 0;

}

int push(int data){

    if(isFull()){

    printf("Stack Overflow!");

    }

    else

    top=top+1;

    stack\_arr[top]=data;

}

int isEmpty(){

    if(top==-1){

        return 1;

    }else

    return 0;

}

int pop(){

    int data;

    if(isEmpty()){

        printf("\nStack Underflow");

    }

    else{

        data=stack\_arr[top];

        top=top-1;

        return data;

    }

}

int peek(){

    if(isEmpty()){

        printf("Stack Underflow!\n");

        exit(1);

    }

    return stack\_arr[top];

}

void print(){

    if(top==-1){

        printf("\n Stack Underflow");

        return;

    }

    for (int i = top; i >=0; i--)

    {

        printf("%d ",stack\_arr[i]);

    }

}

int main()

{

    int choice,data;

    while(1){

        printf("\n");

        printf("1.Push\n");

        printf("2.pop\n");

        printf("3.Print the top element.\n");

        printf("4.Print all the element.\n ");

        printf("5.Quit\n");

        printf("\nEnter your choice.");

        scanf("%d",&choice);

        switch (choice)

{

case 1:

    printf("Enter the elemnt to be pushed:\n");

    scanf("%d",&data);

    push(data);

    break;

case 2:

    data=pop();

    printf("Deleted element is: %d\n",data);

    break;

case 3:

    printf("The topmost element of the stack is %d\n",peek());

    break;

case 4:

    print();

    break;

case 5:

    exit(1);

default:

    printf("Wrong Choice.\n");

    break;

}

    }

   return 0;

}

1. Infix To Postfix

**Code**

#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.");

    }

    else

    {

        top = top + 1;

        stack[top] = item;

    }

}

/\* define pop operation \*/

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;

    char 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");

            getchar();

            exit(1);

        }

        if (top > 0)

        {

            printf("\nInvalid infix Expression.\n");

            getchar();

            exit(1);

        }

        postfix\_exp[j] = '\0';

    }

    int main()

    {

        char infix[SIZE], postfix[SIZE];

        printf("\nEnter Infix expression : ");

        gets(infix);

        InfixToPostfix(infix, postfix);

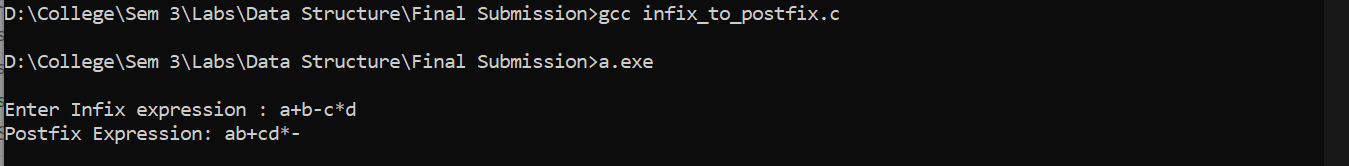
        printf("Postfix Expression: ");

        puts(postfix);

        return 0;

    }

**Input/Output**

****

1. PostFix Eavaluation

**Code**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

#define MAX 100

int stack[MAX];

char infix[MAX], postfix[MAX];

int top = -1;

int push(int val)

{

    if (top == MAX - 1)

    {

        printf("Stack Overflow.");

    }

    else

        top++;

    stack[top] = val;

    return 0;

}

char pop()

{

    int val;

    if (top == -1)

    {

        printf("Stack underflow");

    }

    else

        val = stack[top];

    top--;

    return val;

}

int post\_eval()

{

    int i;

    int a, b;

    for (i = 0; i < strlen(postfix); i++)

    {

        if (postfix[i] >= '0' && postfix[i] <= '9')

        {

            push(postfix[i] - '0');

        }

        else

        {

            a = pop();

            b = pop();

            switch (postfix[i])

            {

            case '+':

                push(b + a);

                break;

            case '-':

                push(b - a);

                break;

            case '\*':

                push(b \* a);

                break;

            case '/':

                push(b / a);

            case '^':

                push(pow(b, a));

                break;

            }

        }

    }

    return pop();

}

int main()

{

    int result;

    printf("Enter the postfix expression:");

    gets(postfix);

    post\_eval();

    result = post\_eval();

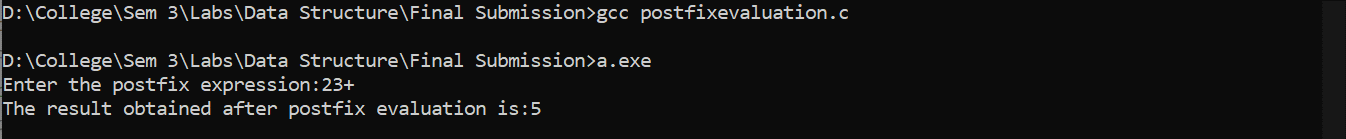
    printf("The result obtained after postfix evaluation is:");

    printf("%d\n", result);

    return 0;

}

**Input/Output**



1. Parenthesis Matching

**Code**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define max 100

int top = -1;

char stack[max];

int isEmpty()

{

    if (top == -1)

        return 1;

    else

        return 0;

}

int isFull()

{

    if (top == max - 1)

    {

        return 1;

    }

    else

        return 0;

}

void push(char c)

{

    if (isFull())

    {

        printf("Stack Overflow.");

    }

    top = top + 1;

    stack[top] = c;

}

char pop()

{

    char c;

    if (isEmpty())

    {

        printf("Stack underflow.");

    }

    else

    {

        c = stack[top];

        top--;

        return c;

    }

}

int match\_char(char a, char b)

{

    if (a == '[' && b == ']')

        return 1;

    if (a == '(' && b == ')')

        return 1;

    if (a == '{' && b == '}')

        return 1;

    else

        return 0;

}

int check\_balanced(char \*s)

{

    char temp;

    for (int i = 0; i < strlen(s); i++)

    {

        if (s[i] == '(' || s[i] == '[' || s[i] == '{')

            push(s[i]);

        if (s[i] == ')' || s[i] == ']' || s[i] == '}')

        {

            if (isEmpty())

            {

                printf("Right brackets are more than the left.\n");

                return 0;

            }

            else

            {

                temp = pop();

                if (!match\_char(temp, s[i]))

                {

                    printf("Mismatched brackets.\n");

                    return 0;

                }

            }

        }

    }

    if (isEmpty())

    {

        printf("Brackets are well balanced.\n");

        return 1;

    }

    else

    {

        printf("Left brackets are more than the right brackets.\n");

        return 0;

    }

}

int main()

{

    char expr[max];

    int balanced;

    printf("Enter the algebric expression.\n");

    fgets(expr, max, stdin);

    balanced == check\_balanced(expr);

    if (balanced == 1)

    {

        printf("The expression is valid.");

    }

    else

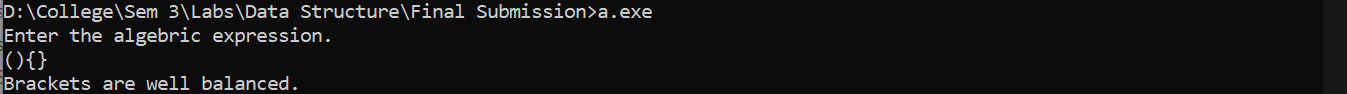
    {

        printf("The expression is not valid");

    }

    return 0;

}

****

Simple Queue

#include <stdio.h>

int queue[10], front = -1;

int is\_full(){

    if(front == 9){

        return 1;

    }

    else return 0;

}

int is\_empty(){

    if(front == -1){

        return 1;

    }

    else return 0;

}

void enQueue(int \*num){

    if(is\_full()){

        printf("Queue Overload.");

    }

    else{

        front++;

        queue[front] = \*num;

    }

}

void deQueue(){

    if(is\_empty()){

        printf("Queue is already empty!");

    }

    else{

        printf("Top element: %d\n", queue[0]);

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

            queue[i-1] = queue[i];

        }

        queue[front] = 0;

        front--;

    }

}

int main(){

    int n, num;

    while(1){

        printf("\nWhich operation would you like to use: (1) EnQueue \n(2) DeQueue\n(3) Print Queue\n");

        scanf("%d", &n);

        if(n == 1){

            printf("Enter the element to insert: ");

            scanf("%d", &num);

            enQueue(&num);

        }

        else if(n == 2){

            deQueue();

        }

        else if(n == 3){

            printf("The Queue: ");

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

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

            }

        }

        else{

            printf("Invalid input!");

        }

    }

}

Double queue

#include <stdio.h>

#define N 5

int queue[N];

int front = -1;

int rear = -1;

void enqueuerear(int x)

{

    if (rear == N - 1)

    {

        printf("Overflow");

    }

    else if (front == -1 && rear == -1)

    // queue is empty.

    {

        front = 0;

        rear = 0;

        queue[rear] = x;

    }

    else

    {

        rear++;

        queue[rear] = x;

    }

}

void dequeuefront()

{

    int x;

    if (front == -1 && rear == -1)

    {

        printf("Queue is empty.");

    }

    // only one element in the queue.

    else if (front == rear)

    {

        front = rear = -1;

    }

    else

    {

        front++;

    }

}

void enqueuefront(int x)

{

    if (front == 0 && rear == N - 1)

    {

        printf("Queue is full.");

    }

    else if (front == -1 && rear == -1)

    {

        front = rear = 0;

    }

    else

    {

        front--;

        queue[front] = x;

    }

}

void dequeuerear()

{

    int x;

    if (front == -1 && rear == -1)

    {

        printf("Queue is empty.");

    }

    if (front == rear)

    {

        front = -1;

        rear = -1;

    }

    else

    {

        queue[rear] = x;

        rear--;

    }

}

void display()

{

    if (front == -1 && rear == -1)

    {

        printf("Queue is empty.");

    }

    for (int i = front; i <= rear; i++)

    {

        printf("%d-", queue[i]);

    }

}

int main()

{ // inserting 3,4,8 & 7 to the queue...

    enqueuerear(3);

    enqueuerear(4);

    enqueuerear(8);

    enqueuerear(7);

    display();

    enqueuefront(34); // inserting 34 from the front.

    printf("\nInserting 34 from the front:");

    display();

    dequeuerear();

    printf("\nDeleting the element from the rear:");

    display();

    dequeuefront();

    printf("\nDeleting the lement from the front:");

    display();

    return 0;

}

Circular Queue

#include <stdio.h>

int queue[5], front = -1, rear = -1;

int is\_full(){

    if(rear == 4 && queue[front] != 0){

        return 1;

    }

    else return 0;

}

int is\_empty(){

    if(front > rear){

        return 1;

    }

    else return 0;

}

void enQueue(int num){

    if(is\_full()){

        printf("Queue overload.");

    }

    else if(rear == 4){

        rear = 0;

        queue[rear] = num;

    }

    else{

        rear++;

        queue[rear] = num;

    }

}

void deQueue(){

}

int main(){

    int n, num;

    while(1){

        printf("\nWhich operation would you like to use: (1) EnQueue \n(2) DeQueue\n(3) Print Queue\n");

        scanf("%d", &n);

        if(n == 1){

            printf("Enter the element to insert: ");

            scanf("%d", &num);

            enQueue(&num);

        }

        else if(n == 2){

            deQueue();

        }

        else if(n == 3){

            printf("The Queue: ");

            for(int i = front; i <= rear; i++){

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

            }

        }

        else{

            printf("Invalid input!");

        }

    }

}

Linear Linked List

#include <stdio.h>

#include <stdlib.h>

struct Linked\_List{

    int data;

    struct Linked\_List \*next;

}\*head, \*list;

typedef struct Linked\_List Node;

void insert(int val){

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

    temp->data = val;

    temp->next = NULL;

    list->next = temp;

    list = list->next;

}

int main(){

    int n, val;

    head = NULL;

    list = (Node\*)(malloc(sizeof(Node)));

    list->data = 2;

    list->next = NULL;

    head = list;

    while(1){

        printf("Do you want items (1 / 0): ");

        scanf("%d",&n);

        if(n == 1){

            printf("Value: ");

            scanf("%d",&val);

            insert(val);

        }

        else break;

    }

    Node\* temp = head;

    printf("The list: ");

    while(temp != NULL){

        printf("%d ",temp->data);

        temp = temp->next;

    }

}

Doubly linked list

#include <stdio.h>

#include <stdlib.h>

struct node

{

    struct node \*prev;

    struct node \*next;

    int data;

};

struct node \*head;

void insertion\_beginning();

void insertion\_last();

void insertion\_specified();

void deletion\_beginning();

void deletion\_last();

void deletion\_specified();

void display();

void search();

void main()

{

    int choice = 0;

    while (choice != 9)

    {

        printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n 5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n");

        printf("\nEnter your choice?\n");

        scanf("\n%d",&choice);

        switch(choice)

        {

        case 1:

            insertion\_beginning();

            break;

        case 2:

            insertion\_last();

            break;

        case 3:

            insertion\_specified();

            break;

        case 4:

            deletion\_beginning();

            break;

        case 5:

            deletion\_last();

            break;

        case 6:

            deletion\_specified();

            break;

        case 7:

            search();

            break;

        case 8:

            display();

            break;

        case 9:

            exit(0);

            break;

        default:

            printf("Please enter valid choice..");

        }

    }

}

void insertion\_beginning()

{

    struct node \*ptr;

    int item;

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

    if (ptr == NULL)

    {

        printf("\nOVERFLOW");

    }

    else

    {

        printf("\nEnter Item value");

        scanf("%d", &item);

        if (head == NULL)

        {

            ptr->next = NULL;

            ptr->prev = NULL;

            ptr->data = item;

            head = ptr;

        }

        else

        {

            ptr->data = item;

            ptr->prev = NULL;

            ptr->next = head;

            head->prev = ptr;

            head = ptr;

        }

        printf("\nNode inserted\n");

    }

}

void insertion\_last()

{

    struct node \*ptr, \*temp;

    int item;

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

    if (ptr == NULL)

    {

        printf("\nOVERFLOW");

    }

    else

    {

        printf("\nEnter value");

        scanf("%d", &item);

        ptr->data = item;

        if (head == NULL)

        {

            ptr->next = NULL;

            ptr->prev = NULL;

            head = ptr;

        }

        else

        {

            temp = head;

            while (temp->next != NULL)

            {

                temp = temp->next;

            }

            temp->next = ptr;

            ptr->prev = temp;

            ptr->next = NULL;

        }

    }

    printf("\nnode inserted\n");

}

void insertion\_specified()

{

    struct node \*ptr, \*temp;

    int item, loc, i;

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

    if (ptr == NULL)

    {

        printf("\n OVERFLOW");

    }

    else

    {

        temp = head;

        printf("Enter the location");

        scanf("%d", &loc);

        for (i = 0; i < loc; i++)

        {

            temp = temp->next;

            if (temp == NULL)

            {

                printf("\n There are less than %d elements", loc);

                return;

            }

        }

        printf("Enter value");

        scanf("%d", &item);

        ptr->data = item;

        ptr->next = temp->next;

        ptr->prev = temp;

        temp->next = ptr;

        temp->next->prev = ptr;

        printf("\nnode inserted\n");

    }

}

void deletion\_beginning()

{

    struct node \*ptr;

    if (head == NULL)

    {

        printf("\n UNDERFLOW");

    }

    else if (head->next == NULL)

    {

        head = NULL;

        free(head);

        printf("\nnode deleted\n");

    }

    else

    {

        ptr = head;

        head = head->next;

        head->prev = NULL;

        free(ptr);

        printf("\nnode deleted\n");

    }

}

void deletion\_last()

{

    struct node \*ptr;

    if (head == NULL)

    {

        printf("\n UNDERFLOW");

    }

    else if (head->next == NULL)

    {

        head = NULL;

        free(head);

        printf("\nnode deleted\n");

    }

    else

    {

        ptr = head;

        if (ptr->next != NULL)

        {

            ptr = ptr->next;

        }

        ptr->prev->next = NULL;

        free(ptr);

        printf("\nnode deleted\n");

    }

}

void deletion\_specified()

{

    struct node \*ptr, \*temp;

    int val;

    printf("\n Enter the data after which the node is to be deleted : ");

    scanf("%d", &val);

    ptr = head;

    while (ptr->data != val)

        ptr = ptr->next;

    if (ptr->next == NULL)

    {

        printf("\nCan't delete\n");

    }

    else if (ptr->next->next == NULL)

    {

        ptr->next = NULL;

    }

    else

    {

        temp = ptr->next;

        ptr->next = temp->next;

        temp->next->prev = ptr;

        free(temp);

        printf("\nnode deleted\n");

    }

}

void display()

{

    struct node \*ptr;

    printf("\n printing values...\n");

    ptr = head;

    while (ptr != NULL)

    {

        printf("%d\n", ptr->data);

        ptr = ptr->next;

    }

}

void search()

{

    struct node \*ptr;

    int item, i = 0, flag;

    ptr = head;

    if (ptr == NULL)

    {

        printf("\nEmpty List\n");

    }

    else

    {

        printf("\nEnter item which you want to search?\n");

        scanf("%d", &item);

        while (ptr != NULL)

        {

            if (ptr->data == item)

            {

                printf("\nitem found at location %d ", i + 1);

                flag = 0;

                break;

            }

            else

            {

                flag = 1;

            }

            i++;

            ptr = ptr->next;

        }

        if (flag == 1)

        {

            printf("\nItem not found\n");

        }

    }

}

Stack Using Linked list

#include <stdio.h>

#include <stdlib.h>

void push();

void pop();

void display();

struct node

{

    int val;

    struct node \*next;

};

struct node \*head;

void main()

{

    int choice = 0;

    while (choice != 4)

    {

        printf("\n\nChose one from the below options...\n");

        printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");

        printf("\n Enter your choice \n");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

        {

            push();

            break;

        }

        case 2:

        {

            pop();

            break;

        }

        case 3:

        {

            display();

            break;

        }

        case 4:

        {

            printf("Exiting....");

            break;

        }

        default:

        {

            printf("Please Enter valid choice ");

        }

        };

    }

}

void push()

{

    int val;

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

    if (ptr == NULL)

    {

        printf("not able to push the element");

    }

    else

    {

        printf("Enter the value");

        scanf("%d", &val);

        if (head == NULL)

        {

            ptr->val = val;

            ptr->next = NULL;

            head = ptr;

        }

        else

        {

            ptr->val = val;

            ptr->next = head;

            head = ptr;

        }

        printf("Item pushed");

    }

}

void pop()

{

    int item;

    struct node \*ptr;

    if (head == NULL)

    {

        printf("Underflow");

    }

    else

    {

        item = head->val;

        ptr = head;

        head = head->next;

        free(ptr);

        printf("Item popped");

    }

}

void display()

{

    int i;

    struct node \*ptr;

    ptr = head;

    if (ptr == NULL)

    {

        printf("Stack is empty\n");

    }

    else

    {

        printf("Printing Stack elements \n");

        while (ptr != NULL)

        {

            printf("%d\n", ptr->val);

            ptr = ptr->next;

        }

    }

}

Circular Linkedlist

#include <stdio.h>

#include <stdlib.h>

struct CircularLinearLinkedList{

    int val;

    struct CircularLinearLinkedList \*next;

}\*head, \*list;

typedef struct CircularLinearLinkedList Node;

void insert(int val){

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

    temp->val = val;

    temp->next = head;

    list->next = temp;

    list = list->next;

}

int main(){

    int n, val;

    head = NULL;

    list = (Node\*)(malloc(sizeof(Node)));

    list->val = 20;

    list->next = NULL;

    head = list;

    while(1){

        printf("Do you want items (1 / 0): ");

        scanf("%d",&n);

        if(n == 1){

            printf("Value: ");

            scanf("%d",&val);

            insert(val);

        }

        else break;

    }

    Node\* temp = head;

    printf("The list: ");

    do{

        printf("%d ", temp->val);

        temp = temp->next;

    } while(temp != head);

}

Queue Using Linked List

    #include<stdio.h>

    #include<stdlib.h>

    struct node

    {

        int data;

        struct node \*next;

    };

    struct node \*front;

    struct node \*rear;

    void insert();

    void delete();

    void display();

    void main ()

    {

        int choice;

        while(choice != 4)

        {

            printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");

            printf("\nEnter your choice ?");

            scanf("%d",& choice);

            switch(choice)

            {

                case 1:

                insert();

                break;

                case 2:

                delete();

                break;

                case 3:

                display();

                break;

                case 4:

                exit(0);

                break;

                default:

                printf("\nEnter valid choice??\n");

            }

        }

    }

    void insert()

    {

        struct node \*ptr;

        int item;

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

        if(ptr == NULL)

        {

            printf("\nOVERFLOW\n");

            return;

        }

        else

        {

            printf("\nEnter value?\n");

            scanf("%d",&item);

            ptr -> data = item;

            if(front == NULL)

            {

                front = ptr;

                rear = ptr;

                front -> next = NULL;

                rear -> next = NULL;

            }

            else

            {

                rear -> next = ptr;

                rear = ptr;

                rear->next = NULL;

            }

        }

    }

    void delete ()

    {

        struct node \*ptr;

        if(front == NULL)

        {

            printf("\nUNDERFLOW\n");

            return;

        }

        else

        {

            ptr = front;

            front = front -> next;

            free(ptr);

        }

    }

    void display()

    {

        struct node \*ptr;

        ptr = front;

        if(front == NULL)

        {

            printf("\nEmpty queue\n");

        }

        else

        {   printf("\nprinting values .....\n");

            while(ptr != NULL)

            {

                printf("\n%d\n",ptr -> data);

                ptr = ptr -> next;

            }

        }

    }