***DATA STRUCTURES***

***ASSIGNMENT***

***2018-2019***

**C:\Program Files (x86)\Microsoft Office\MEDIA\CAGCAT10\j0195384.wmf**

**ASSIGNMENT BY:**

**HARSHIT BARGUJAR**

**012**

**BTECH IT 2ND SEM**

**1.Arrays using pointers**

#include<stdio.h>

int main()

{ int arr[70] , i ,size;

printf("Enter the size of array: ");

scanf("%d",&size);

printf("Enter the elements: ");

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

scanf("%d",arr+i); }

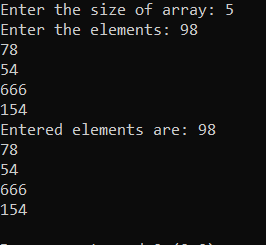
printf("Entered elements are : ");

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

printf("%d\n",\*(arr+i));

return 0;

}



**2. Stacks using arrays:**

#include <stdio.h>

#include <stdlib.h>

int ch, n, top, x, i,stack[100];

void push(){

if(top>=n-1){

printf("\nStack is overflow"); }

else{

printf("Enter the value to be pushed ");

scanf("%d",&x);

top++;

stack[top]=x; }

}

void pop(){

if(top<=-1){

printf("\nStack in underflow"); }

else{

printf("The popped element is %d",stack[top]);

top--; }

}

void topelement(){

printf("The element at the top of stack is %d",stack[top]); }

void emptystack(){

if(top<=-1){

printf("\nYes stack is empty"); }

else{

printf("\nNo stack has elements in it"); }

}

void printstack(){

if(top>=0){

printf("\nThe elements in stack are");

for(i=top;i>=0;i--){

printf("\n%d",stack[i]); }

printf("\nPress the next choice");

}

else{

printf("\nStack is empty"); }

}

int main()

{ top =-1;

printf("\n \t\t----Welcome to the stack implementation program----\n");

printf("Enter the size of stack [max=100]: ");

scanf("%d",&n);

printf("Choose from the operations to perform\n ");

printf("1.Push\n 2.Pop\n 3.Top element\n 4.Is stack empty\n 5.Print elements in the stack\n 6.Exit program\n ");

do{

printf("\nEnter the choice\t");

scanf("%d",&ch);

switch(ch){

case 1: push(); break;

case 2: pop(); break;

case 3: topelement(); break;

case 4: emptystack(); break;

case 5: printstack(); break;

case 6: printf("\n \tExit point\n"); break;

default: {printf("Invalid choice , please choose from the available choices\n");}

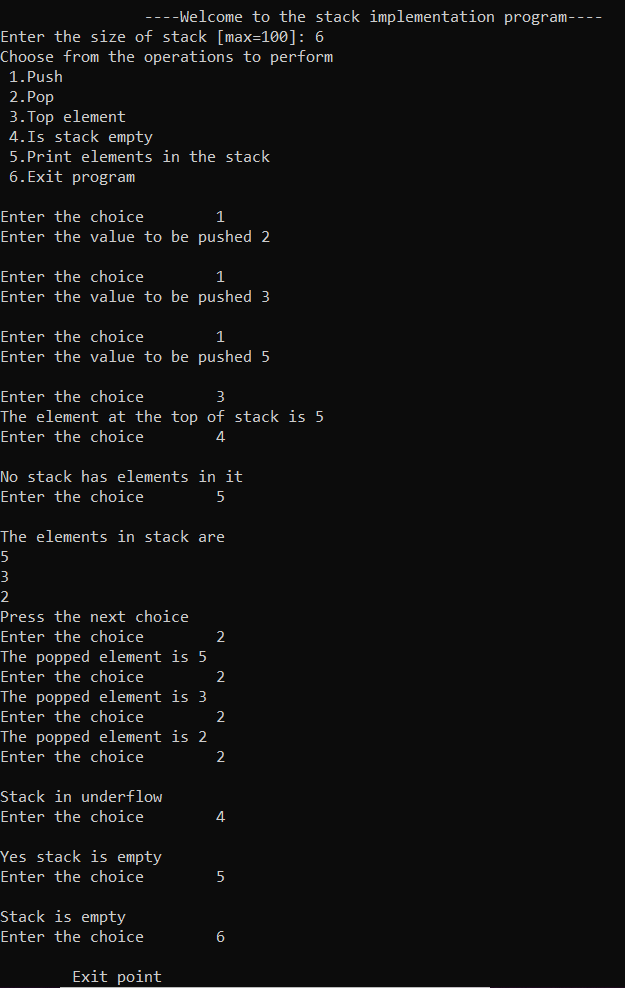
}

}

while(ch!=6);

return 0;

}



**3.Stack implementation using linked list**

#include <stdio.h>

#include <stdlib.h>

struct node{

int data;

struct node\* next;

}\*top1,\*data;

struct node\* top;

void push(int data){

struct node \*temp;

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

if(!temp){

printf("\nHeap overflow");

return;

}

temp->data=data;

temp->next=top;

top=temp;

}

void pop(){

struct node \*temp;

if(top==NULL){

printf("\nStack is underflow");

return;

}

else{

printf("\nThe popped element is %d", top->data);

temp=top;

top=top->next;

temp->next=NULL;

free(temp); }

}

void emptystack(){

if (top==NULL){

printf("Stack is empty\n"); }

else{

printf("Stack has data in it\n"); }

}

void topelement(){

if (top==NULL)

printf("Stack is empty");

else

printf("%d" ,top->data);

}

void printstack(){

top1 = top;

if (top1 == NULL)

{ printf("Stack is empty");

return; }

while (top1 != NULL)

{ printf("%d ", top1->data);

top1 = top1->next; }

}

int main()

{ int ch,x;

printf("\n \t\t----Welcome to the stack implementation program----\n");

printf("Choose from the operations to perform\n ");

printf("1.Push\n 2.Pop\n 3.Top element\n 4.Is stack empty\n 5.Print elements in the stack\n 6.Exit program\n ");

do{

printf("\nEnter the choice\t");

scanf("%d",&ch);

switch(ch){

case 1: printf("Enter data : ");

scanf("%d", &x); push(x); break;

case 2: pop(); break;

case 3: topelement(); break;

case 4: emptystack(); break;

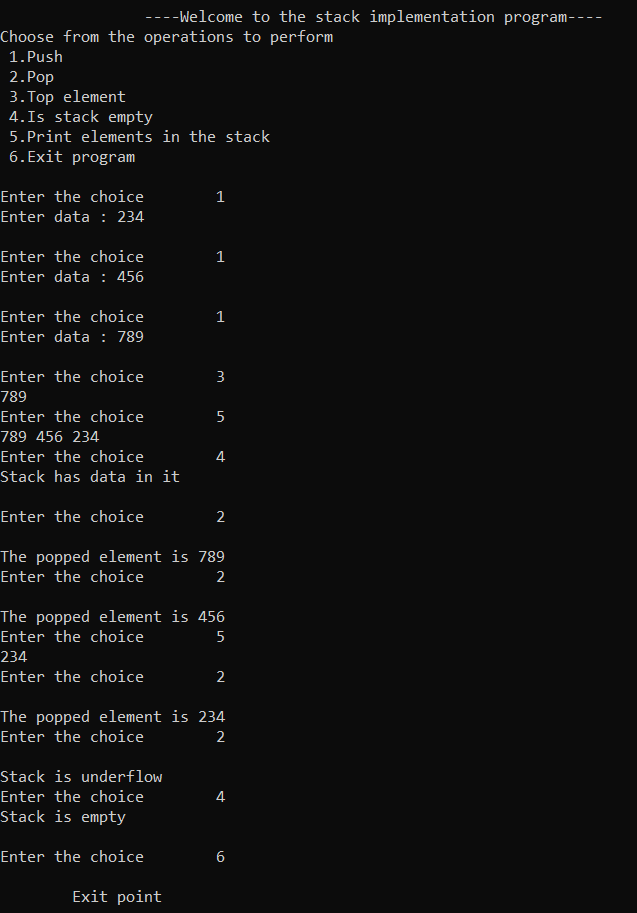
case 5: printstack(); break;

case 6: printf("\n \tExit point\n"); break;

default: {printf("Invalid choice , please choose from the available choices\n");}

} } while(ch!=6);

return 0;}



**4.Decimal to binary using stack**

#include<stdio.h>

#include<stdlib.h>

struct node

{ int data;

struct node \*next;

};

struct node \*createnode()

{ struct node \*n;

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

n->data=NULL;

n->next=NULL;

return n;

}

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

struct node \*temp;

temp=createnode();

if(\*stack!=NULL)

temp->next=\*stack;

temp->data=data;

\*stack=temp;

}

int pop(struct node \*\*stack)

{ struct node \*n;

n=\*stack;

\*stack=n->next;

return n->data;

}

int main()

{ int c=0,rem,dec,binary;

struct node \*stack;

stack=createnode();

printf("Enter the decimal number : ");

scanf("%d",&dec);

while(dec!=0){

rem=dec%2;

push(stack,rem);

dec=dec/2;

c++;

}

printf("The number in binary is : ");

while(c)

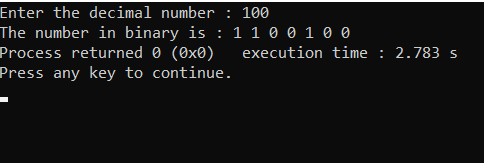
{ binary=pop(stack);

printf("%d ",binary);

c--; }

return 0;

}



**5.Infix to postfix using stack**

#include<stdio.h>

#include<ctype.h>

#include<string.h>

int priority(char);

int push(char);

int pop();

int top=0 ,s,p=0;

char stack[80],postfix[80];

int priority(char v)

{ if(v=='+' || v=='-')

return 1;

else if(v=='/' || v=='\*')

return 2;

else

return 0;

}

int push(char ch)

{ int j,k;

char z;

stack[0]='(';

j=priority(ch);

k=priority(stack[top]);

if(top==s)

printf("stack is full");

else

{ if(j>k)

{ stack[++top]=ch;

}

else{

if(ch=='(')

stack[++top]=ch;

else{

z=pop();

postfix[p++]=z;

stack[++top]=ch; }

} }

return 0;

}

int pop()

{ return stack[top--]; }

int main()

{ int c;

char infix[80];

printf("please enter your expression :\t");

scanf("%s",&infix);

s=strlen(infix);

infix[s]=')';

infix[s+1]='\0';

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

{

if(infix[i]=='(')

push(infix[i]);

else if(infix[i]=='+' || infix[i]=='/' || infix[i]=='-' || infix[i]=='\*' || infix[i]=='^')

push(infix[i]);

else if(infix[i]==')')

{ char x;

int k=1;

while(k)

{ x=pop();

if(x=='(')

k=0;

else

postfix[p++]=x; }

}

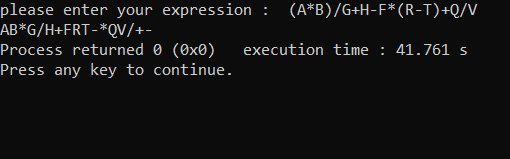
else

postfix[p++]=infix[i]; }

printf("%s",postfix);

return 0;

}



**6.Evaluate postfix expression using stack**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

struct node{

int top;

int capacity;

int \*array;

};

struct node \*createnode(int capacity)

{ struct node \*n;

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

n->top=-1;

n->capacity=capacity;

n->array=(int \*)malloc(sizeof(int)\*capacity);

return n;

}

void push(struct node \*stack,int data)

{ stack->array[++stack->top]=data;

}

int pop(struct node \*stack)

{ if(stack->top==-1)

printf("Empty");

else{

return stack->array[stack->top--]; }

}

int main()

{ struct node \*stack;

char postfix[40];

int i,result,val1,val2;

printf("Enter the expression: ");

scanf("%s",&postfix);

stack=createnode(strlen(postfix));

for (i=0;postfix[i];i++)

{ if(isdigit(postfix[i]))

push(stack,postfix[i]-'0');

else{ val1=pop(stack);

val2=pop(stack);

switch(postfix[i])

{ case '+':result= val2+val1;

push(stack,result);break;

case '-':result= val2-val1;

push(stack,result);break;

case '/':result= val2/val1;

push(stack,result);break;

case '\*':result= val2\*val1;

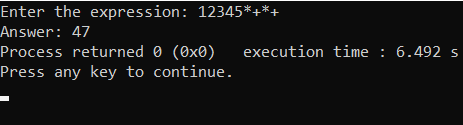
push(stack,result);break;

} }

}

printf("Answer: %d",pop(stack));

}



**7.Check for balanced parenthesis using stack**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

struct node

{ int top;

int capacity;

char \*array;

};

struct node \*createstack(int capacity)

{ struct node \*s;

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

s->top=-1;

s->array=(char \*)malloc(sizeof(char)\*capacity);

s->capacity=capacity;

return s;

};

void push(struct node \*stack,char data)

{ stack->array[++stack->top]=data;

}

char pop(struct node \*stack)

{ return stack->array[stack->top--];

}

int main()

{ struct node \*stack;

char exp[40],x;

int i, valexp=1,c=0,d=0;

printf("Enter the Expression: ");

scanf("%s",&exp);

stack=createstack(strlen(exp));

for (i=0;exp[i];i++)

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

push(stack,exp[i]);

c++; }

else

{ x=pop(stack);

if( exp[i]==')' )

{if(x!='('){

valexp=0;}}

else if( exp[i]==']' )

{if( x!='[' ){

valexp=0;}}

else if( exp[i]=='}' )

{if( x!='{' ){

valexp=0;}}

d++;

} }

if(c!=d)

valexp=0;

if(valexp==1)

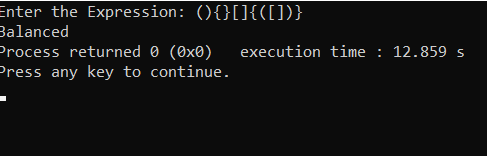
printf("Balanced");

else

printf("Unbalanced");

return 0;

}



**8.Implement queue using array**

#include <stdio.h>

#include <stdlib.h>

int ch, n, x, i,queue[100], rear =-1,front=-1;

void enqueue(){

int add\_item;

if(rear==n-1){

printf("Queue is overflow\n"); }

else{ if(front==-1)

{front=0;}

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

scanf("%d",&add\_item);

rear=rear+1;

queue[rear]=add\_item; }

}

void dequeue(){

if(front==-1 || front>rear){

printf("Queue is underflow\n");}

else{

printf("Element deleted from queue is: %d",queue[front]);

front=front+1; }

}

void top(){

printf("The element at front of the queue is %d :",queue[front]);

}

void printqueue(){

int i;

if (front==-1)

{printf("Queue is empty\n");}

else{

printf("Queue is: \n");

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

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

printf("\n"); }

} }

int main()

{ printf("\n \t\t----Welcome to the queue implementation program----\n");

printf("Enter the size of queue [max=100]: ");

scanf("%d",&n);

printf("Choose from the operations to perform\n ");

printf("1.Enqueue\n 2.Dequeue\n 3.Top element\n 4.Print elements in the queue\n 5.Exit program\n ");

do{

printf("\nEnter the choice\t");

scanf("%d",&ch);

switch(ch){

case 1: enqueue(); break;

case 2: dequeue(); break;

case 3: top(); break;

case 4: printqueue(); break;

case 5: printf("\n \tExit point\n"); break;

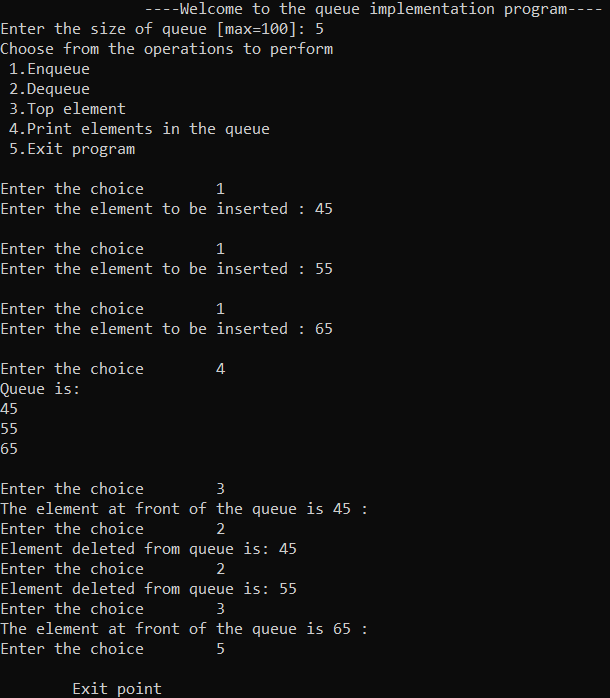
default: {printf("Invalid choice , please choose from the available choices\n");}

} }

while(ch!=5);

return 0;

}



**9.Implement queue using linked list**

#include<stdio.h>

#include<conio.h>

struct Node

{ int data;

struct Node \*next;

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

int ch,x;

void enqueue(int value)

{ struct Node \*newNode;

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

newNode->data = value;

newNode -> next = NULL;

if(front == NULL)

front = rear = newNode;

else{

rear -> next = newNode;

rear = newNode; }

}

void dequeue()

{ if(front == NULL)

printf("\nQueue is Empty\n");

else{

struct Node \*temp = front;

front = front -> next;

printf("\nDeleted element: %d\n", temp->data);

free(temp); }

}

void display()

{ temp=front;

if(front==NULL) {

printf("Queue is empty\n"); }

else{ printf("\n");

for(;temp!=rear;temp=temp->next)

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

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

}

void top(){

if(front==NULL){

printf("Queue is empty\n"); }

else{ printf("Element at top is:%d\n",front->data); }

}

int main()

{ printf("\n \t\t----Welcome to the queue implementation program----\n");

printf("Choose from the operations to perform\n ");

printf("1.Enqueue\n 2.Dequeue\n 3.Top element\n 4.Print elements in the queue\n 5.Exit program\n ");

do{ printf("\nEnter the choice\t");

scanf("%d",&ch);

switch(ch){

case 1: printf("Enter data : ");

scanf("%d", &x);

enqueue(x); break;

case 2: dequeue(); break;

case 3: top(); break;

case 4: display(); break;

case 5: printf("\n \tExit point\n"); break;

default: {printf("Invalid choice , please choose from the available choices\n");}

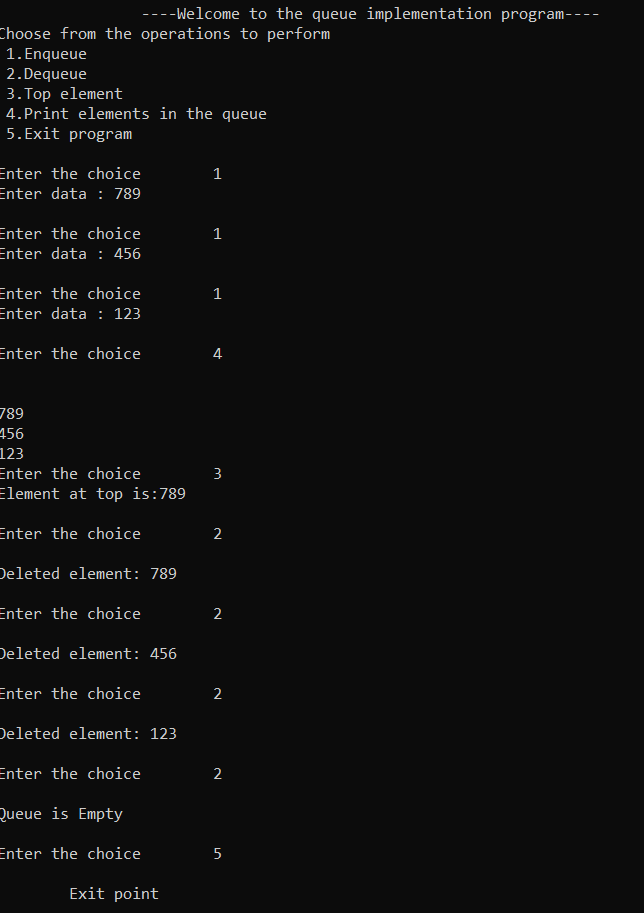
}

}

while(ch!=5);

return 0;

}



**10.Implement circular queue using array**

#include <stdio.h>

#include <stdlib.h>

int ch, n, x, i,capacity,queue[100], rear =-1,front=-1;

void enqueue(){

int add\_item;

if(n==capacity){

printf("Queue is overflow\n"); }

else{

if(front==-1)

{front=0;}

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

scanf("%d",&add\_item);

rear = (rear + 1) % capacity;

queue[rear]=add\_item;

n++; }

}

void dequeue(){

if(front==-1 || n==0){

printf("Queue is underflow\n");}

else{

n--;

printf("Element deleted from queue is: %d",queue[front]);

front = (front + 1) % capacity; }

}

void top(){

if(front==-1 || n==0)

printf("Empty");

else

printf("Element at top is: %d",queue[front]);

}

int main()

{ printf("\n \t\t----Welcome to the queue implementation program----\n");

printf("Enter the size of queue [max=100]: ");

scanf("%d",&n);

printf("Choose from the operations to perform\n ");

printf("1.Enqueue\n 2.Dequeue\n 3.Top element\n 4.Exit program\n ");

capacity=n;

n=0;

do{ printf("\nEnter the choice\t");

scanf("%d",&ch);

switch(ch){

case 1: enqueue(); break;

case 2: dequeue(); break;

case 3: top(); break;

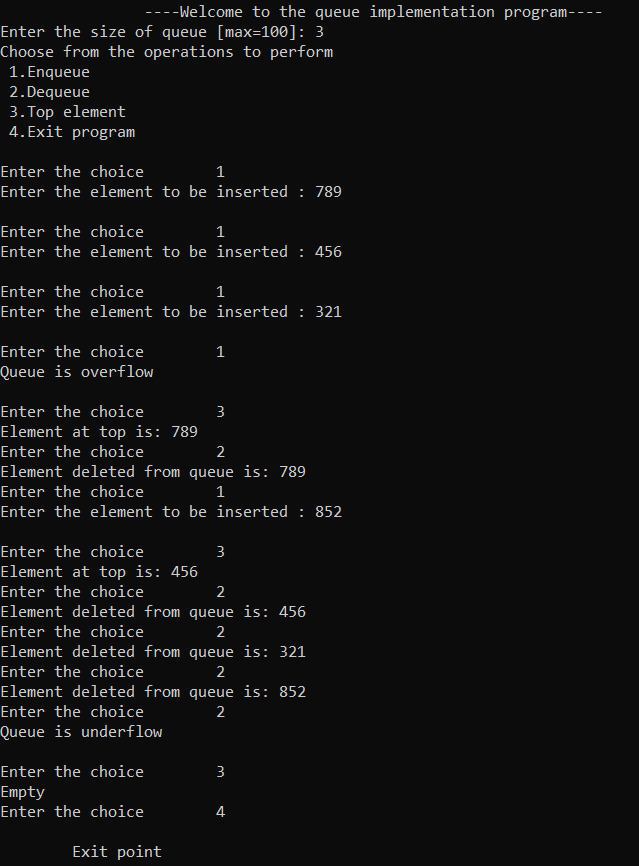
case 4: printf("\n \tExit point\n"); break;

default: {printf("Invalid choice , please choose from the available choices\n");}

}

} while(ch!=4);

return 0; }



**11.Implement circular queue using linked list**

#include<stdio.h>

#include<stdlib.h>

typedef struct cqueue{

int info;

struct cqueue \*next;

}node;

node \*front=NULL, \*rear=NULL , \*temp;

int ch;

void enqueue()

{ node \*newnode;

newnode=(node\*)malloc(sizeof(node));

printf("Enter the element to be inserted in queue ");

scanf("%d",&newnode->info);

newnode->next=NULL;

if(rear==NULL)

{ front=rear=newnode; }

else{ rear->next=newnode;

rear=newnode; }

}

void dequeue()

{ temp=front;

if(front ==NULL)

{ printf("Queue is empty\n"); }

else{ if(front==rear){

printf("Popped element is: %d\n", front->info);

front=rear=NULL; }

else{ printf("Popped element is:%d\n",front->info);

front=front->next;

rear->next=front; }

temp->next=NULL;

free(temp); }

}

void display(){

temp=front;

if(front==NULL) {

printf("Queue is empty\n"); }

else{ printf("\n");

for(;temp!=rear;temp=temp->next)

printf("\n%d",temp->info);

printf("\n%d",temp->info); }

}

void top(){

if(front==NULL){

printf("Queue is empty\n"); }

else{ printf("Element at top is:%d\n",front->info); }

}

int main()

{ printf("\n \t\t----Welcome to the queue implementation program----\n");

printf("Choose from the operations to perform\n ");

printf("1.Enqueue\n 2.Dequeue\n 3.Top element\n 4.Print elements in the queue\n 5.Exit program\n ");

do{ printf("\nEnter the choice\t");

scanf("%d",&ch);

switch(ch){

case 1: enqueue(); break;

case 2: dequeue(); break;

case 3: top(); break;

case 4: display(); break;

case 5: printf("\n \tExit point\n"); break;

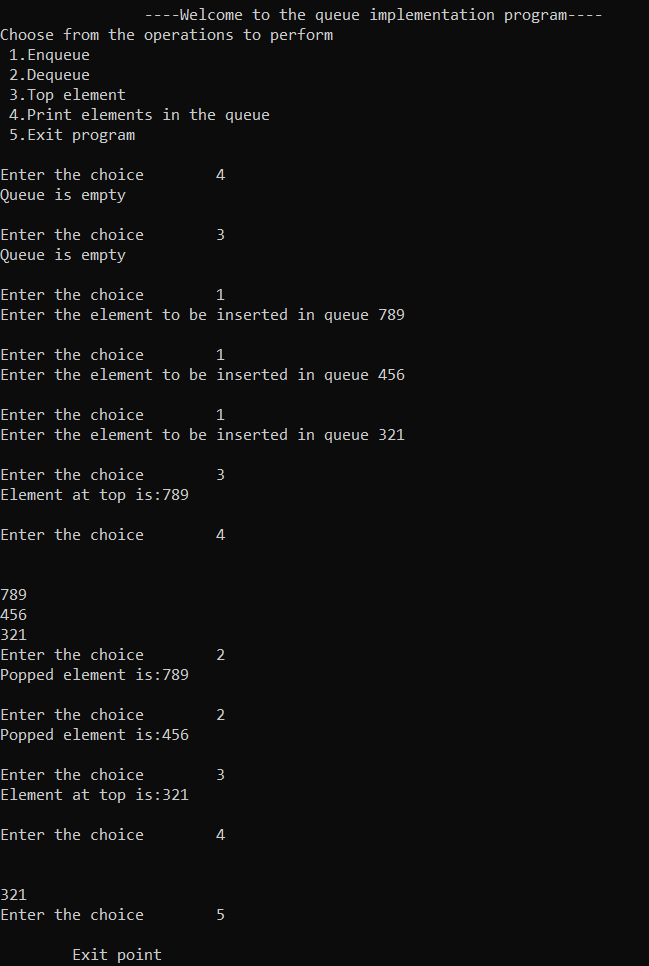
default: {printf("Invalid choice , please choose from the available choices\n");}

} }

while(ch!=5);

return 0;

}



**12.Singly linked list**

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

struct node {

int value;

struct node \*next;

};

typedef struct node DATA\_NODE;

DATA\_NODE \*head\_node, \*first\_node, \*temp\_node = 0, \*prev\_node, next\_node;

int ch ,data;

void insert() {

printf("Enter Element for Insert Linked List : \n");

scanf("%d", &data);

temp\_node = (DATA\_NODE \*) malloc(sizeof (DATA\_NODE));

temp\_node->value = data;

if (first\_node == 0) {

first\_node = temp\_node;

} else {

head\_node->next = temp\_node; }

temp\_node->next = 0;

head\_node = temp\_node;

fflush(stdin);

}

void del() {

int countvalue, pos, i = 0;

countvalue = count();

temp\_node = first\_node;

printf("Enter Position for Delete Element : \n");

scanf("%d", &pos);

if (pos > 0 && pos <= countvalue) {

if (pos == 1) {

temp\_node = temp\_node -> next;

first\_node = temp\_node;

printf("Deleted Successfully \n");

} else {

while (temp\_node != 0) {

if (i == (pos - 1)) {

prev\_node->next = temp\_node->next;

if(i == (countvalue - 1))

{ head\_node = prev\_node;

}

printf("Deleted Successfully \n");

break;

} else { i++;

prev\_node = temp\_node;

temp\_node = temp\_node -> next; } }

}

} else printf("Invalid Position \n");

}

void display() {

int count = 0;

temp\_node = first\_node;

printf("Elements in Linked List : \n");

while (temp\_node != 0) {

printf(" %d ", temp\_node->value);

count++;

temp\_node = temp\_node -> next;

}

printf("No Of Items In Linked List : %d\n", count);

}

int count() {

int count = 0;

temp\_node = first\_node;

while (temp\_node != 0) {

count++;

temp\_node = temp\_node -> next; }

printf("No Of Items In Linked List : %d\n", count);

return count;

}

int main()

{ printf(" \t\t----Welcome to the linked list implementation program----\n");

printf("Choose from the operations to perform\n ");

printf("1.Insert\n 2.Delete\n 3.Count elements\n 4.Print elements in the linked list\n 5.Exit program\n ");

do{ printf("\nEnter the choice\t");

scanf("%d",&ch);

switch(ch){

case 1: insert(); break;

case 2: del(); break;

case 3: count(); break;

case 4: display(); break;

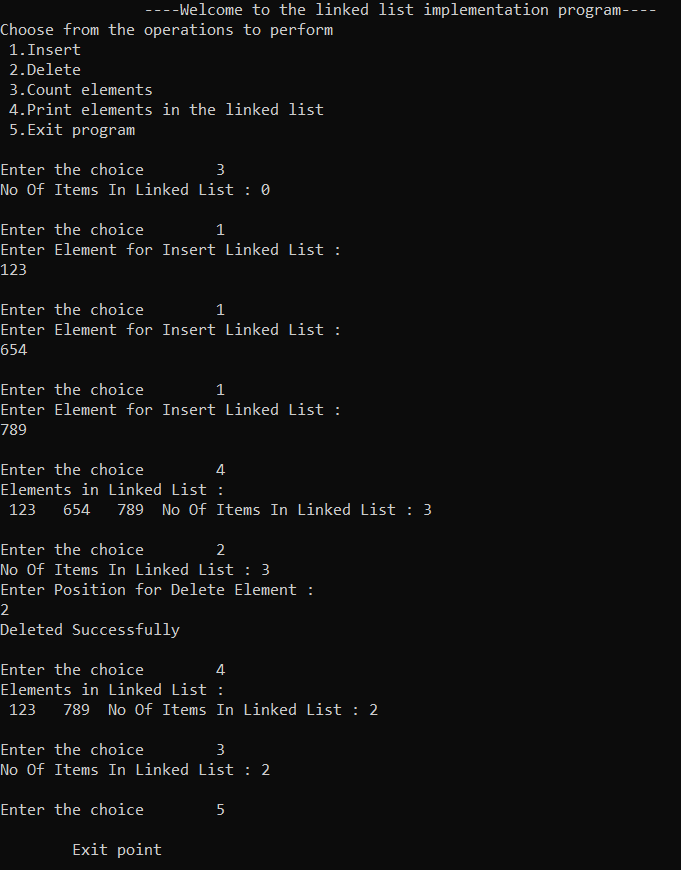
case 5: printf("\n \tExit point\n"); break;

default: {printf("Invalid choice , please choose from the available choices\n");}

} }

while(ch!=5);

return 0;}



**13.Doubly linked list**

#include<stdio.h>

#include<stdlib.h>

struct Node;

typedef struct Node \* PtrToNode;

typedef PtrToNode List;

typedef PtrToNode Position;

struct Node

{ int e;

Position previous;

Position next;

};

void Insert(int x, List l, Position p)

{ Position TmpCell;

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

if(TmpCell == NULL)

printf("Memory out of space\n");

else

{ TmpCell->e = x;

TmpCell->previous = p;

TmpCell->next = p->next;

p->next = TmpCell; }

}

int isLast(Position p)

{ return (p->next == NULL);

}

Position Find(int x, List l)

{ Position p = l->next;

while(p != NULL && p->e != x)

p = p->next;

return p;

}

void Delete(int x, List l)

{ Position p, p1, p2;

p = Find(x, l);

if(p != NULL)

{ p1 = p -> previous;

p2 = p -> next;

p1 -> next = p -> next;

if(p2 != NULL)

p2 -> previous = p -> previous; }

else

printf("Element not found\n");

}

void Display(List l)

{ printf("The list element are : ");

Position p = l->next;

while(p != NULL)

{ printf("%d ", p->e);

p = p->next; }

}

int main()

{ int x, pos, ch, i; List l, l1;

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

l->previous = NULL; l->next = NULL;

List p = l;

printf(" \t\t----Welcome to the linked list implementation program----\n");

printf("Choose from the operations to perform\n ");

printf("1.Insert\n 2.Delete\n 3.Search elements\n 4.Print elements in the linked list\n 5.Exit program\n ");

do{ printf("\nEnter the choice\t");

scanf("%d",&ch);

switch(ch){

case 1: p = l;

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

scanf("%d",&x);

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

scanf("%d",&pos);

for(i = 1; i < pos; i++)

{ p = p->next; }

Insert(x,l,p); break;

case 2: p = l;

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

scanf("%d",&x); Delete(x,p); break;

case 3: p = l;

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

scanf("%d",&x);

p = Find(x,p);

if(p == NULL)

printf("Element does not exist!!!\n");

else

printf("Element exist!!!\n");

break;

case 4: Display(l); break;

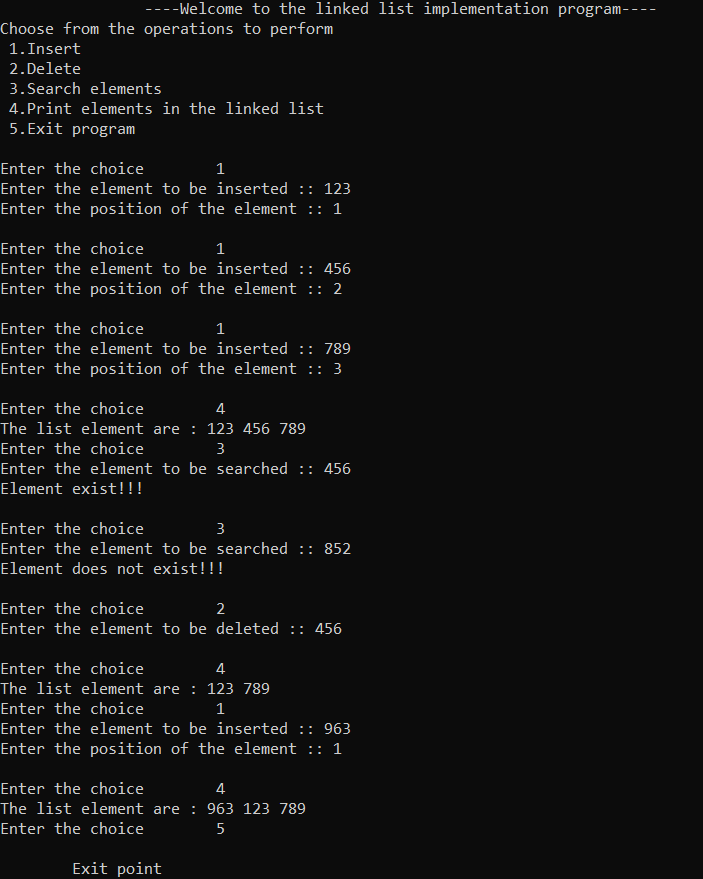
case 5: printf("\n \tExit point\n"); break;

default: {printf("Invalid choice , please choose from the available choices\n");}

} }

while(ch!=5);

return 0;}



**14.Singly circular linked list**

#include<stdio.h>

#include<stdlib.h>

struct node

{ int data;

struct node \*next;

};

struct node \*head;

int main()

{ int ch;

printf(" \t\t----Welcome to the linked list implementation program----\n");

printf("Choose from the operations to perform\n ");

printf("1.Insert\n 2.Delete from beginning\n 3.Search for element\n 4.Print elements in the linked list\n 5.Exit program\n ");

do{ printf("\nEnter the choice\t");

scanf("%d",&ch);

switch(ch){

case 1: insert(); break;

case 2: begin\_delete(); break;

case 3: search(); break;

case 4: display(); break;

case 5: printf("\n \tExit point\n"); break;

default: {printf("Invalid choice , please choose from the available choices\n");}

} }

while(ch!=5);

return 0;}

void insert()

{ struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{ printf("OVERFLOW\n");

}

else

{ printf("Element to be inserted\n");

scanf("%d",&item);

ptr -> data = item;

if(head == NULL)

{ head = ptr;

ptr -> next = head;

}

else

{ temp = head;

while(temp->next != head)

temp = temp->next;

ptr->next = head;

temp -> next = ptr;

head = ptr;

}

printf("Node inserted\n"); }

}

void begin\_delete()

{ struct node \*ptr;

if(head == NULL)

{ printf("UNDERFLOW\n");

}

else if(head->next == head)

{ head = NULL;

free(head);

printf("Node deleted\n");

}

else

{ ptr = head;

while(ptr -> next != head)

ptr = ptr -> next;

ptr->next = head->next;

free(head);

head = ptr->next;

printf("Node deleted\n"); }

}

void search()

{ struct node \*ptr;

int item,i=0,flag=1;

ptr = head;

if(ptr == NULL)

{ printf("Empty List\n");

}

else

{ printf("Element to search for\n");

scanf("%d",&item);

if(head ->data == item)

{

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

flag=0; }

else

{ while (ptr->next != head)

{ if(ptr->data == item)

{ printf("Item found at location %d\n",i+1);

flag=0;

break; }

else

{ flag=1; }

i++;

ptr = ptr -> next; }

}

if(flag != 0)

{ printf("Item not found\n"); }

} }

void display()

{ struct node \*ptr;

ptr=head;

if(head == NULL)

{ printf("List is empty\n"); }

else { printf("Elements in list are \n");

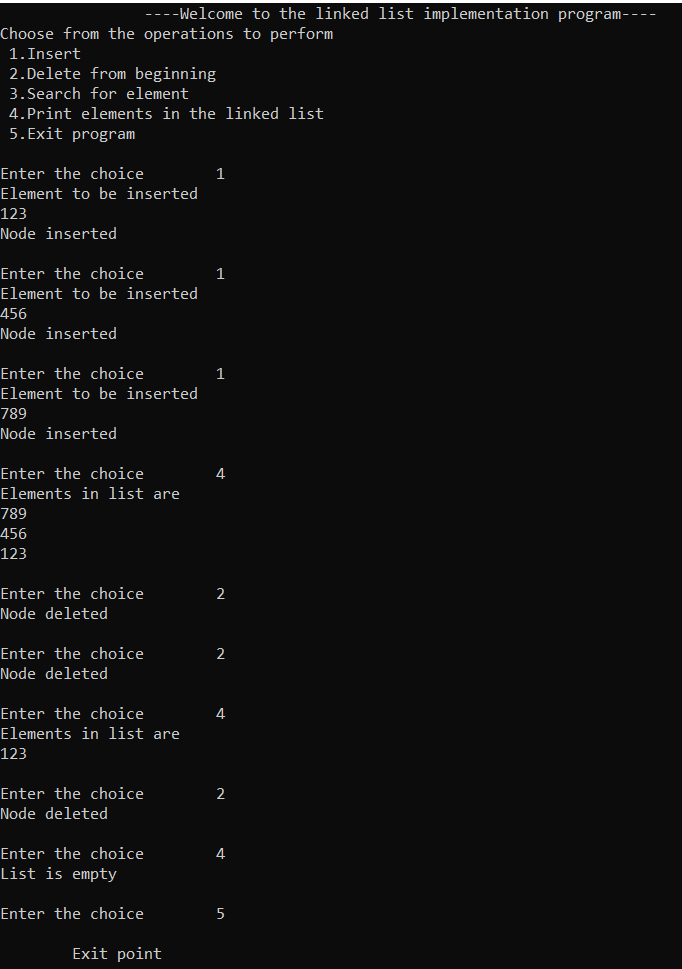
while(ptr -> next != head)

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

ptr = ptr -> next; }

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

}



**15.Doubly circular linked list**

#include<stdio.h>

#include<stdlib.h>

struct node

{ struct node \*prev;

struct node \*next;

int data;

};

struct node \*head;

int main()

{ int ch;

printf(" \t\t----Welcome to the linked list implementation program----\n");

printf("Choose from the operations to perform\n ");

printf("1.Insert\n 2.Delete from beginning\n 3.Search for element\n 4.Print elements in the linked list\n 5.Exit program\n ");

do{ printf("\nEnter the choice\t");

scanf("%d",&ch);

switch(ch){

case 1: insert(); break;

case 2: begin\_delete(); break;

case 3: search(); break;

case 4: display(); break;

case 5: printf("\n \tExit point\n"); break;

default: {printf("Invalid choice , please choose from the available choices\n");}

} }

while(ch!=5);

return 0;}

void insert()

{ struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{ printf("OVERFLOW\n");

}

else

{ printf("Element to be inserted\n");

scanf("%d",&item);

ptr->data=item;

if(head==NULL)

{ head = ptr;

ptr -> next = head;

ptr -> prev = head; }

else

{ temp = head;

while(temp -> next != head)

{ temp = temp -> next; }

temp -> next = ptr;

ptr -> prev = temp;

head -> prev = ptr;

ptr -> next = head;

head = ptr; }

printf("Node inserted\n");}

}

void begin\_delete()

{ struct node \*temp;

if(head == NULL)

{ printf("UNDERFLOW\n");

}

else if(head->next == head)

{ head = NULL;

free(head);

printf("Node deleted\n"); }

else

{ temp = head;

while(temp -> next != head)

{ temp = temp -> next;

}

temp -> next = head -> next;

head -> next -> prev = temp;

free(head);

head = temp -> next; }

}

void display()

{ struct node \*ptr;

ptr=head;

if(head == NULL)

{ printf("List is empty\n");

}

else

{ printf("Elements in list are \n");

while(ptr -> next != head)

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

ptr = ptr -> next; }

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

}

void search()

{ struct node \*ptr;

int item,i=0,flag=1;

ptr = head;

if(ptr == NULL)

{ printf("List is empty\n"); }

else

{ printf("Element to search ");

scanf("%d\n",&item);

if(head ->data == item)

{ printf("Item found at location %d",i+1);

flag=0; }

else

{ while (ptr->next != head)

{ if(ptr->data == item)

{ printf("Item found at location %d \n",i+1);

flag=0;

break; }

else { flag=1; }

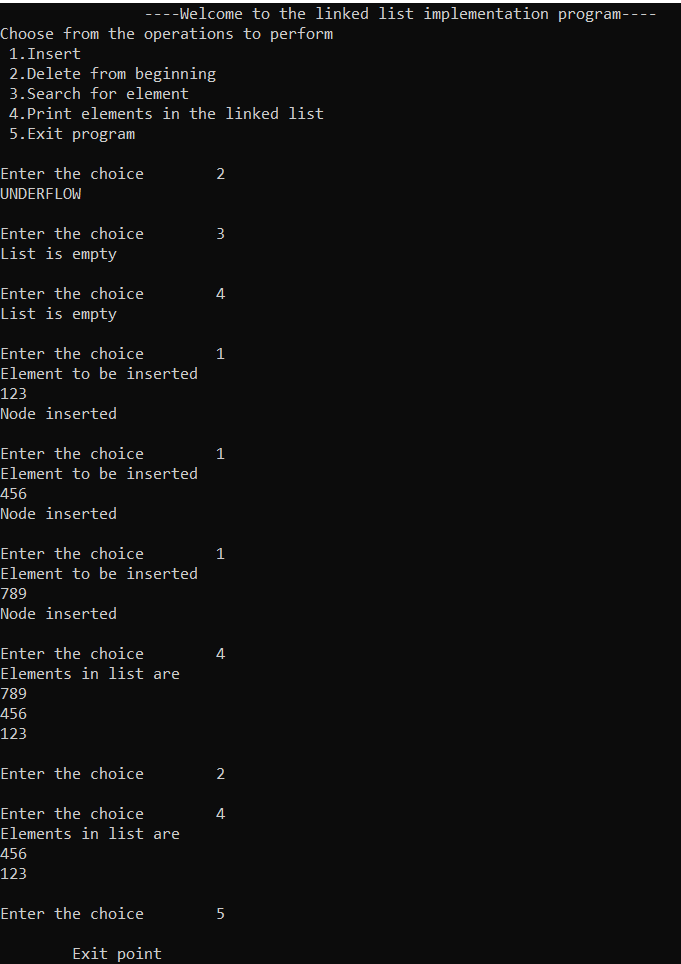
i++; ptr = ptr -> next; }

}

if(flag != 0)

{ printf("Item not found\n"); } }

}



**16.Selection sort**

#include <stdio.h>

void selection\_sort();

int a[30], n;

void main()

{ int i;

printf("\t\t----Selection sort program----\n");

printf("\nEnter size of an array: ");

scanf("%d", &n);

printf("\nEnter elements of an array:\n");

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

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

selection\_sort();

printf("\n\nAfter sorting:\n");

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

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

printf("\nExit point");

return 0;

}

void selection\_sort()

{ int i, j, min, temp;

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

{ min = i;

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

{ if (a[j] < a[min])

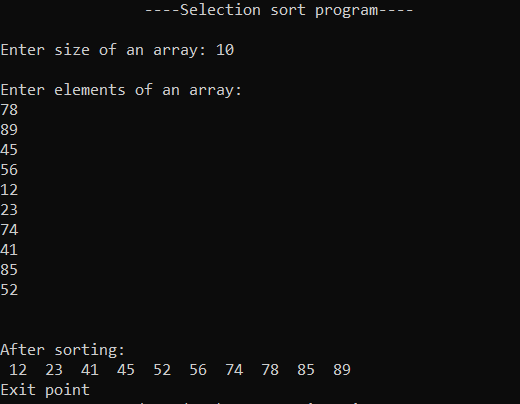
min = j; }

temp = a[i];

a[i] = a[min];

a[min] = temp; }

}



**17.Insertion sort**

int main()

{ int n, array[100], c, d, t;

printf("\t\t----Insertion sort program----\n");

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

for (c = 0; c < n; c++)

scanf("%d", &array[c]);

for (c = 1 ; c <= n - 1; c++) {

d = c;

while ( d > 0 && array[d-1] > array[d]) {

t = array[d];

array[d] = array[d-1];

array[d-1] = t;

d--; }

}

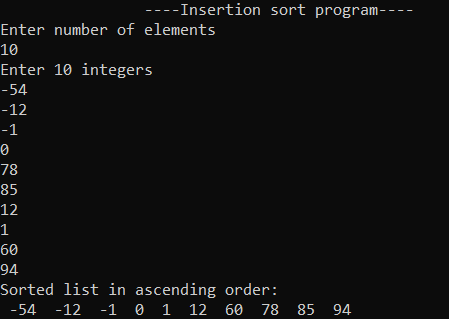
printf("Sorted list in ascending order:\n");

for (c = 0; c <= n - 1; c++) {

printf(" %d ", array[c]); }

return 0;

}



**18.Bubble sort**

#include<stdio.h>

int main()

{ int a[50],n,i,j,temp;

printf("\t\t----Bubble sort program----\n");

printf("Enter the size of array max[50]: ");

scanf("%d",&n);

printf("Enter the array elements: ");

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

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

for(i=1;i<n;++i)

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

if(a[j]>a[j+1])

{ temp=a[j];

a[j]=a[j+1];

a[j+1]=temp; }

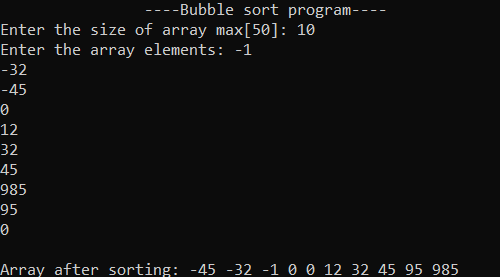
printf("\nArray after sorting: ");

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

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

return 0;

}



**19.Merge Sort**

#include<stdio.h>

void mergesort(int a[],int i,int j);

void merge(int a[],int i1,int j1,int i2,int j2);

int main()

{ int a[50],n,i;

printf("\t\t----Merge sort program----\n");

printf("Enter size of array max[50]:");

scanf("%d",&n);

printf("Enter array elements:");

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

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

mergesort(a,0,n-1);

printf("\nSorted array is :");

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

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

return 0;

}

void mergesort(int a[],int i,int j)

{ int mid;

if(i<j)

{ mid=(i+j)/2;

mergesort(a,i,mid);

mergesort(a,mid+1,j);

merge(a,i,mid,mid+1,j); }

}

void merge(int a[],int i1,int j1,int i2,int j2)

{ int temp[50];

int i,j,k; i=i1;

j=i2; k=0;

while(i<=j1 && j<=j2)

{ if(a[i]<a[j])

temp[k++]=a[i++];

else

temp[k++]=a[j++]; }

while(i<=j1)

temp[k++]=a[i++];

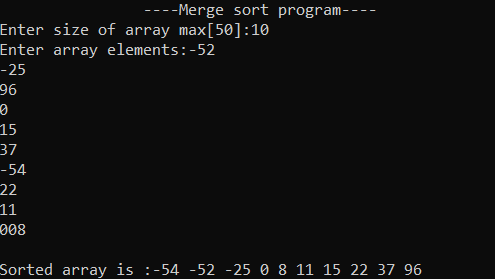
while(j<=j2)

temp[k++]=a[j++];

for(i=i1,j=0;i<=j2;i++,j++)

a[i]=temp[j];

}



**20.Quick Sort**

#include <stdio.h>

void quick\_sort(int[],int,int);

int partition(int[],int,int);

int main()

{ int a[50],n,i;

printf("\t\t----Quick sort program----\n");

printf("Enter size of array max[50]:");

scanf("%d",&n);

printf("\nEnter array elements:");

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

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

quick\_sort(a,0,n-1);

printf("\nArray after sorting:");

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

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

return 0;

}

void quick\_sort(int a[],int l,int u)

{ int j;

if(l<u)

{ j=partition(a,l,u);

quick\_sort(a,l,j-1);

quick\_sort(a,j+1,u); }

}

int partition(int a[],int l,int u)

{ int v,i,j,temp;

v=a[l]; i=l;

j=u+1;

do

{ do

i++;

while(a[i]<v&&i<=u);

do

j--;

while(v<a[j]);

if(i<j)

{ temp=a[i];

a[i]=a[j];

a[j]=temp; }

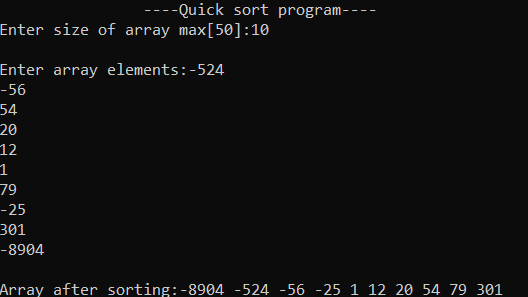
}while(i<j);

a[l]=a[j];

a[j]=v;

return(j);

}



**21.Linear search**

#include<stdio.h>

int main()

{ int a[50],i,x,n;

printf("\t\t----Linear search program----\n");

printf("Enter size of array max[50]:");

scanf("%d",&n);

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

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

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

printf("\nEnter element to search:");

scanf("%d",&x);

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

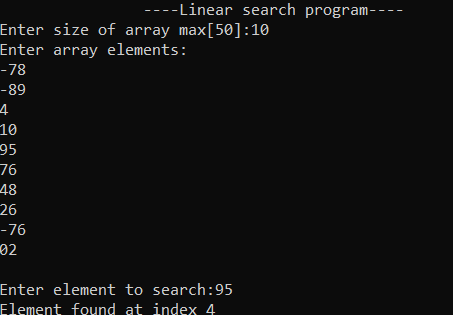
if(a[i]==x) break;

if(i<n)

printf("Element found at index %d",i);

else printf("Element not found");

return 0; }



**22.Binary search**

#include <stdio.h>

int main()

{ int c, first, last, middle, n, search, array[50];

printf("\t\t----Binary search program----\n");

printf("Enter size of array max[50]:");

scanf("%d",&n);

printf("Enter %d integers\n", n);

for (c = 0; c < n; c++)

scanf("%d",&array[c]);

printf("Enter value to find\n");

scanf("%d", &search);

first = 0; last = n - 1;

middle = (first+last)/2;

while (first <= last) {

if (array[middle] < search)

first = middle + 1;

else if (array[middle] == search) {

printf("%d found at location %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

}

if (first > last)

printf("Not found! %d isn't present in the list.\n", search);

return 0;

}

