**PROGRAM:1**

**Write a program to sort an integer array**

#include<conio.h>

#include<stdio.h>

void main()

{

int i,j,a,n, number[30];

clrscr();

printf("enter the value of N \n");

scanf("%d",&n);

printf("enter the numbers \n");

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

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

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

{

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

{

if (number[i]>number[j])

{

a=number[i];

number[i]=number[j];

number[j]=a;

}

}

}

printf("Sorted array \n");

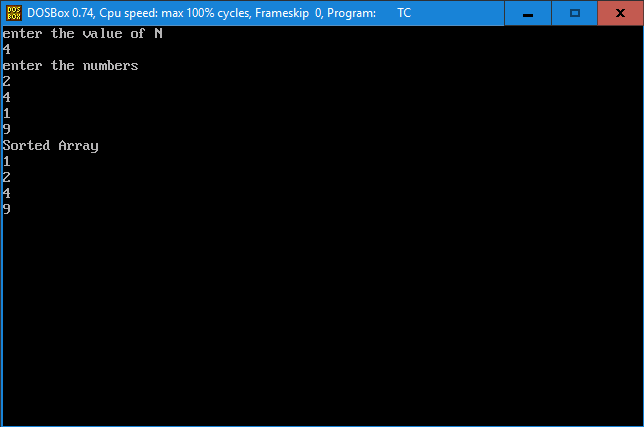
for(i=0;i<n;++i)

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

getch();

}

**OUTPUT**

****

**PROGRAM:2**

**Write a program to merge 2 sorted arrays**

#include<stdio.h>

#include<conio.h>

void main()

{

int array1[50],array2[50],array3[100],m,n,i,j,k=0;

clrscr();

printf("\n Enter the size of array1:");

scanf("%d",&m);

printf("\n Enter sorted element of array1:\n");

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

{

scanf("%d",&array1[i]);

}

printf("\n Enter size of array2:");

scanf("%d",&n);

printf("\n Enter sorted element of arra2:\n");

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

{

scanf("%d",array2[i]);

}

i=0;

j=0;

while(i<m&&j<n)

{

if(array1[i]<array2[j])

{

array3[k]=array1[i];

i++;

}

else

{

array3[k]=array2[j];

j++;

}

k++;

}

if(i>=m)

{

while(j<n)

{

array3[k]=array2[j];

j=0;

while(i<m&&j<n)

{

if(array1[i]<array2[j])

{

array3[k]=array2[i];

i++;

}

else

{

array3[k]=array2[j];

j++;

}

k++;

}

if(i>=m)

{

while(j<n)

{

array3[k]=array1[i];

i++;

k++;

}

}

printf("\n After merging:\n");

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

{

printf("\n%d",array3[i]);

}

getch();

}

}

}

**OUTPUT**

**PROGRAM:3**

**Write a program to implement stack operation**

#include<stdio.h>

#include<conio.h>

#define MAXSIZE 5

struct stack

{

int stk[MAXSIZE];

int top;

};

typedef struct stack STACK;

STACK s;

void push(void);

int pop(void);

void display(void);

void main ()

{

int choice;

int option = 1;

clrscr();

s.top = -1;

printf ("STACK OPERATION\n");

while (option)

{

printf ("------------------------------------------\n");

printf (" 1 --> PUSH \n");

printf (" 2 --> POP \n");

printf (" 3 --> DISPLAY \n");

printf (" 4 --> EXIT \n");

printf ("------------------------------------------\n");

printf ("Enter your choice\n");

scanf ("%d", &choice);

switch (choice)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

return;

}

fflush (stdin);

printf ("Do you want to continue(0 or 1)?\n");

scanf ("%d", &option);

}

}

/\* Function to add an element to the stack \*/

void push ()

{

int num;

if (s.top == (MAXSIZE - 1))

{

printf ("Stack is Full\n");

return;

}

else

{

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

scanf ("%d", &num);

s.top = s.top + 1;

s.stk[s.top] = num;

}

return;

}

/\* Function to delete an element from the stack \*/

int pop ()

{

int num;

if (s.top == - 1)

{

printf ("Stack is Empty\n");

return (s.top);

}

else

{

num = s.stk[s.top];

printf ("poped element is = %d\n", s.stk[s.top]);

s.top = s.top - 1;

}

return(num);

}

/\* Function to display the status of the stack \*/

void display ()

{

int i;

if (s.top == -1)

{

printf ("Stack is empty\n");

return;

}

else

{

printf ("\n The status of the stack is \n");

for (i = s.top; i >= 0; i--)

{

printf ("%d\n", s.stk[i]);

}

}

printf ("\n");

}

**OUTPUT**

**PROGRAM:4**

**Write a program to implement circular stack**

#include<stdio.h>

#include<conio.h>

#define MAX 5

int cqueue\_arr[MAX];

int front=-1;

int rear=-1;

void insert(int item)

{

if((front == 0 && rear == MAX-1)||(front == rear+1))

{

printf("queue overflow \n");

return;

}

if(front == -1)

{

front = 0;

rear = 0;

}

else

{

if(rear == MAX-1)

rear = 0;

else

rear = rear+1;

}

cqueue\_arr[rear] = item;

}

void deletion()

{

if(front == -1)

{

printf("queue underflow\n");

return;

}

printf("Element deleted from queue is:%d/n", cqueue\_arr[front]);

if(front == rear)

{

front=-1;

rear=-1;

}

else

{

if(front == MAX-1)

front = 0;

else

front = front+1;

}

}

void display()

{

int front\_pos = front,rear\_pos = rear;

if(front == -1)

{

printf("queue is empty\n");

return;

}

printf("queue elements:");

if(front\_pos <= rear\_pos)

while(front\_pos <= rear\_pos)

{

printf("%d",cqueue\_arr[front\_pos]);

front\_pos++;

}

else

{

while(front\_pos<=MAX-1)

{

printf("%d",cqueue\_arr[front\_pos]);

front\_pos++;

}

front\_pos = 0;

while(front\_pos <= rear\_pos)

{

printf("%d",cqueue\_arr[front\_pos]);

front\_pos++;

}

}

printf("\n");

}

void main()

{

int choice,item;

clrscr();

do

{

printf("1.insert\n");

printf("2.delete\n");

printf("3.display\n");

printf("4.quit\n");

printf("Enter your choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("input the element for insertion in queue:");

scanf("%d",&item);

insert(item);

break;

case 2:

deletion();

break;

case 3:

display();

break;

case 4:

break;

default:

printf("wrong choice\n");

}

}while(choice!=4);

getch();

}

**OUTPUT**

**PROGRAM:5**

**Write a program to implement linked stack**

|  |
| --- |
|  |
|  | #include<stdio.h> |
|  | #include<stdlib.h> |
|  | #include<limits.h> |
|  | #define CAPACITY 1000 |
|  | struct stack |
|  | { |
|  | int data; |
|  | struct stack \*next; |
|  | }\*top; |
|  | int size = 0; |
|  | void push(int element); |
|  | int pop(); |
|  | void main() |
|  | { |
|  | int choice, data; |
|  | while(1) |
|  | { |
|  | printf("----------------\n"); |
|  | printf("STACK IMPLEMENTATION PROGRAM\n"); |
|  | printf("1.push\n"); |
|  | printf("2.pop\n"); |
|  | printf("3.size\n"); |
|  | printf("4.exit\n"); |
|  | printf("enter your choice\n"); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1: |
|  | printf("enter data to push into stack\n"); |
|  | scanf("%d",&data); |
|  | push(data); |
|  | break; |
|  | case 2: |
|  | data = pop(); |
|  | if (data != INT\_MIN) |
|  | printf("Data =>%d\n", data); |
|  | break; |
|  | case 3: |
|  | printf("stack size:%d\n", size); |
|  | break; |
|  | case 4: |
|  | printf("exiting\n"); |
|  | break; |
|  | default: |
|  | printf("invalid choice, please try again.\n"); |
|  | } |
|  | printf("\n\n"); |
|  | } |
|  | } |
|  | void push(int element) |
|  | { |
|  | struct stack \* newNode = (struct stack \*)malloc(sizeof(struct stack)); |
|  | if(size >= CAPACITY) |
|  | { |
|  | printf("stack overflow\n"); |
|  | return; |
|  | } |
|  | newNode->data = element; |
|  | newNode->next = top; |
|  | top = newNode; |
|  | size++; |
|  | printf("data pushed into stack\n"); |
|  | } |
|  | int pop() |
|  | { |
|  | int data = 0; |
|  | struct stack \* topNode; |
|  | if (size <=0 || !top) |
|  | { |
|  | printf("stack is empty\n"); |
|  | return INT\_MIN; |
|  | } |
|  | topNode = top; |
|  | data = top->data; |
|  | top = top->next; |
|  | free(topNode); |
|  | size--; |
|  | return data; |
|  | } |

**OUTPUT**