Experiment number : 1 Date :

Aim : Program to perform merging and sorting of 2 arrays operations

Algorithm :

Step 1 : Start

Step 2 : Declare 3 arrays a[10],b[10],c[20] and the size of first 2 arrays

Step 3 : Read the size and element of first array

Step 4 : Read the size and element of second array

Step 5 : Initialize x,y,z to lowerbound to 0

Step 6 : Check if a[x]<=b[y]

Then assign a[x] to c[z]

Set x=x+1 and z=z+1

Else

Then assign b[x] to c[z]

Set y=y+1 and z=z+1

Endif

Step 7 : Assign remaining elements of a[] and b[] toc[]

Step 8 : Dislay the elements of c[] by

Step 9 : initialize k=0

Step 10 : repeat for k=0 to k<z-x-1

Step 11 : Check if c[k]>c[k+1]

Swap temp=c[k]

c[k]=c[k+1]

c[k+1]=temp

Step 12 : Display the stored array c

Step 13 : stop

Code :

//ARRAY MERGE & SORT

#include<stdio.h>

void main()

{

int n,m,s,temp,i;

printf("Enter the limit of Array 1 : ");

scanf("%d",&n);

printf("Enter the limit of Array 2 : ");

scanf("%d",&m);

s=n+m;

int a[m],b[n],c[s];

//Array 1 operation

printf("\nEnter the elements of array 1 : ");

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

{

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

}

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

{

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

{

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

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

printf("\nSorted Array 1 : ");

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

{

printf("%4d",a[i]);

c[i]=a[i];

}

//Array 2 operation

printf("\nEnter the elements of array 2 : ");

int k=n;

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

{

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

c[k]=b[i];

k++;

}

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

{

for(int j=i+1;j<m;j++)

{

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

{

temp=b[i];

b[i]=b[j];

b[j]=temp;

}

}

}

printf("\nSorted Array 2 : ");

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

{

printf("%4d",b[i]);

}

printf("\n\nMERGED ARRAY-1 & ARRAY-2 :-\n");//MERGED ARRAY

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

{

printf("%4d",c[i]);

}

for(i=0;i<s;i++)//SORTED ARRAY

{

for(int j=i+1;j<s;j++)

{

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

{

temp=c[i];

c[i]=c[j];

c[j]=temp;

}

}

}

printf("\nSorted array of Merged is : \n");

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

{

printf("%4d",c[i]);

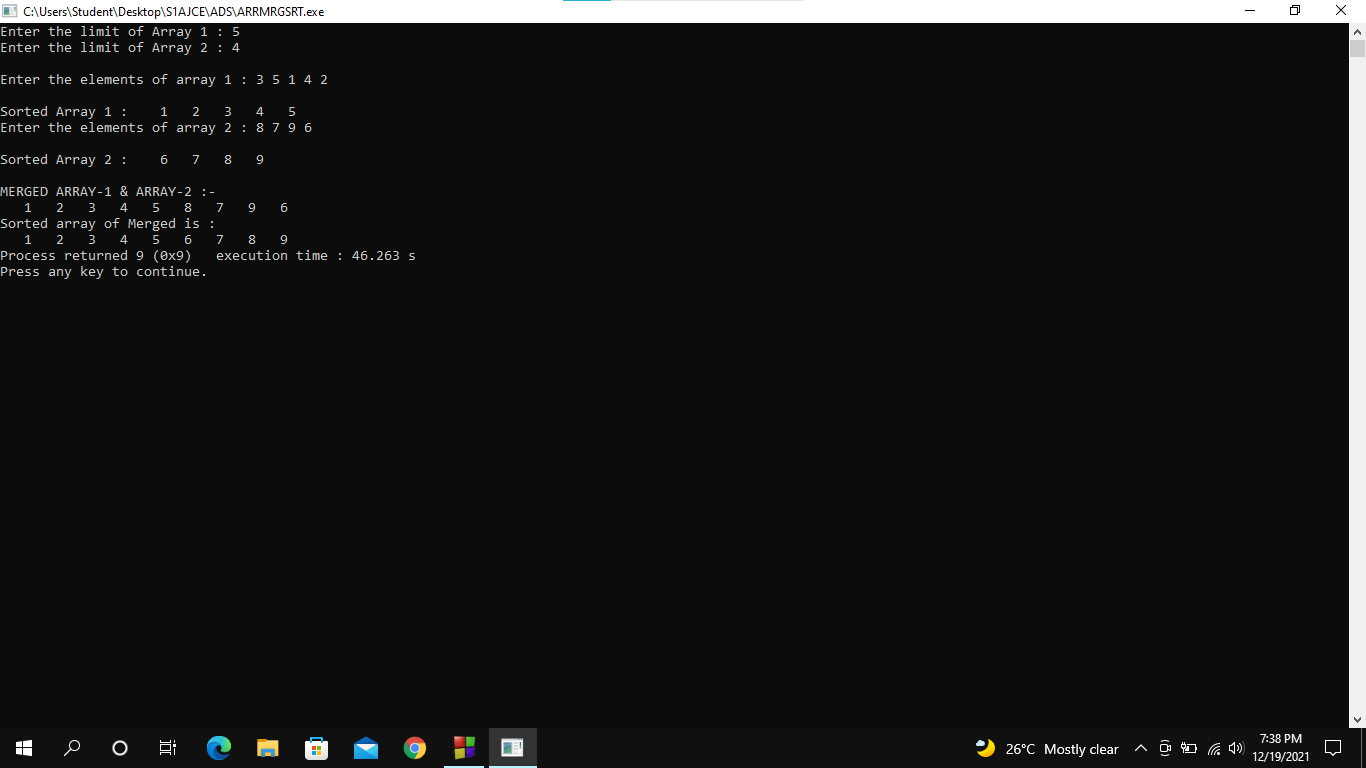
}

}

Result :

The program was executed successfully and obtained the output.

Output:



Experiment number : 2 Date :

Aim : Program to perform Stack operations using Linked List

Algorithm :

Step 1 : Start

Step 2 : Declare functions as PUSH(), POP(), DISPLAY(), SEARCH()

Step 3 : Read the choice of user as 1. PUSH(), 2. POP(), 3. DISPLAY(), 4. SEARCH(), 5. EXIT

Step 4 : if choice is 1

then goto step 5

Else

goto step 8

Step 5 : Declare new node and allocate the memory for new node

Step 6 : Assign the value to the data part of the new node

Step 7 : Check whether the TOP is NULL

Then make the link field of the new node to NULL and set TOP=new NODE

Else

Make the link field of the new node to TOP and

set TOP = new NODE and goto step 3

Step 8 : if the choice is 2

then goto step 9

Else

The goto step 11

Step 9 : Declare a temporary NODE

Step 10 : Check whether TOP==NULL

Then print UNDERFLOW

Else

Assign TOP to the temp node

Set TOP=temp->next

free(temp) and goto step 3

Step 11 : if choice is 3

Then goto step 12

Else

Goto step 15

Step 12 : Declare a node

Step 13 : Assign the top to temp

Step 14 : Repeat while first != NULL

Display first->data

Set first=first->next and goto step 3

Step 15 : if choice is 4

Then goto step 17

Else

Goto step 22

Step 16 : Declare a temporary NODE

Step 17 : Read the item to be searched

Step 18 : Assign TOP=temp

Step 19 : Repeat for temp->next!=NULL

Check if temp->data=item

Then set flag=1

Break

temp=temp->next

Step 20 : Check that flag==1 then

Print item is found

Else

Print item not found and goto step 3

Step 21 : if choice is 5, then

Goto step 24

Else

Goto step 22

Step 22 : if choice is invalid then

Display invalid choice and goto step 3

Step 23 : stop

Code :

//LINKED STACK

#include<stdio.h>

#include<stdlib.h>

void push();

void pop();

void display();

void search();

struct node

{

int val;

struct node \*next;

};

struct node \*head;

void main()

{

int ch;

do

{

printf("\n1. PUSH\n2. POP\n3. TRAVERSE\n4. SEARCH\n5. EXIT\n");

printf("Enter the Choice : ");

scanf("%d",&ch);

switch(ch)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

search();

break;

case 5:

printf("Exiting...");

break;

default:

printf("Invalid Choice made!");

}

}while(ch!=5);

}

void push ()//PUSH OPERATION

{

int val;

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

if(ptr == NULL)

{

printf("Unable to push the element into stack!");

}

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("Element %d pushed into Stack.",val);

}

}

void pop()//POP OPERATION

{

int item;

struct node \*ptr;

if(head==NULL)

{

printf("Stack is Underflow!");

}

else

{

item=head->val;

ptr=head;

head=head->next;

free(ptr);

printf("Element %d Popped from Stack.",item);

}

}

void display()//DISPLAY FUNCTION

{

struct node \*ptr;

ptr=head;

if(ptr==NULL)

{

printf("Stack is Empty!.");

}

else

{

printf("Elements of Stack are : \n");

while(ptr!=NULL)

{

printf("[%d]\n",ptr->val);

ptr=ptr->next;

}

}

}

void search()//SEARCH OPERATION

{

int item,flag=0;

struct node \*ptr;

ptr=head;

if(ptr==NULL)

{

printf("Empty Stack, search operation cannot be performed!.");

}

else

{

printf("Enter the item to search : ");

scanf("%d",&item);

while(ptr!=NULL)

{

if(item==ptr->val)

{

flag=1;

break;

}

ptr=ptr->next;

}

if(flag==0)

{

printf("Element %d not found.",item);

}

else

{

printf("Element %d found.",item);

}

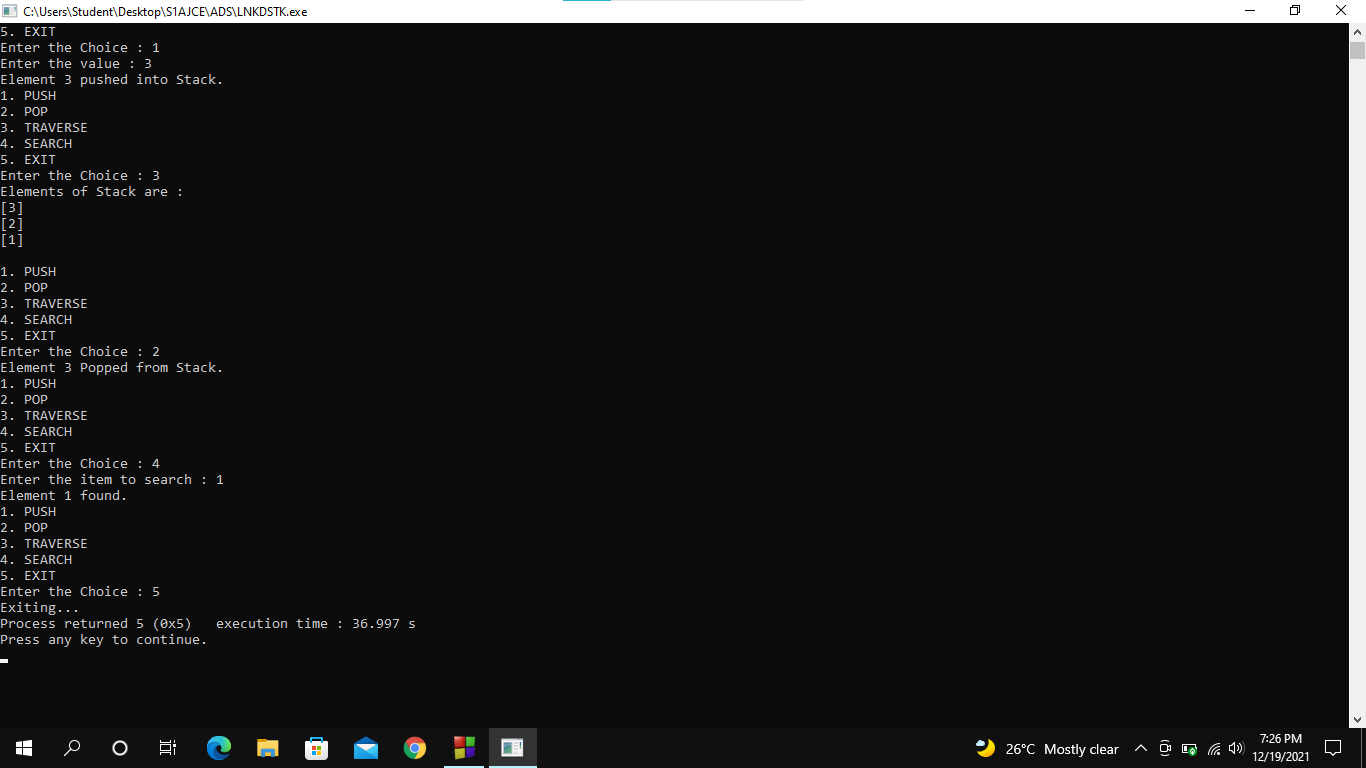
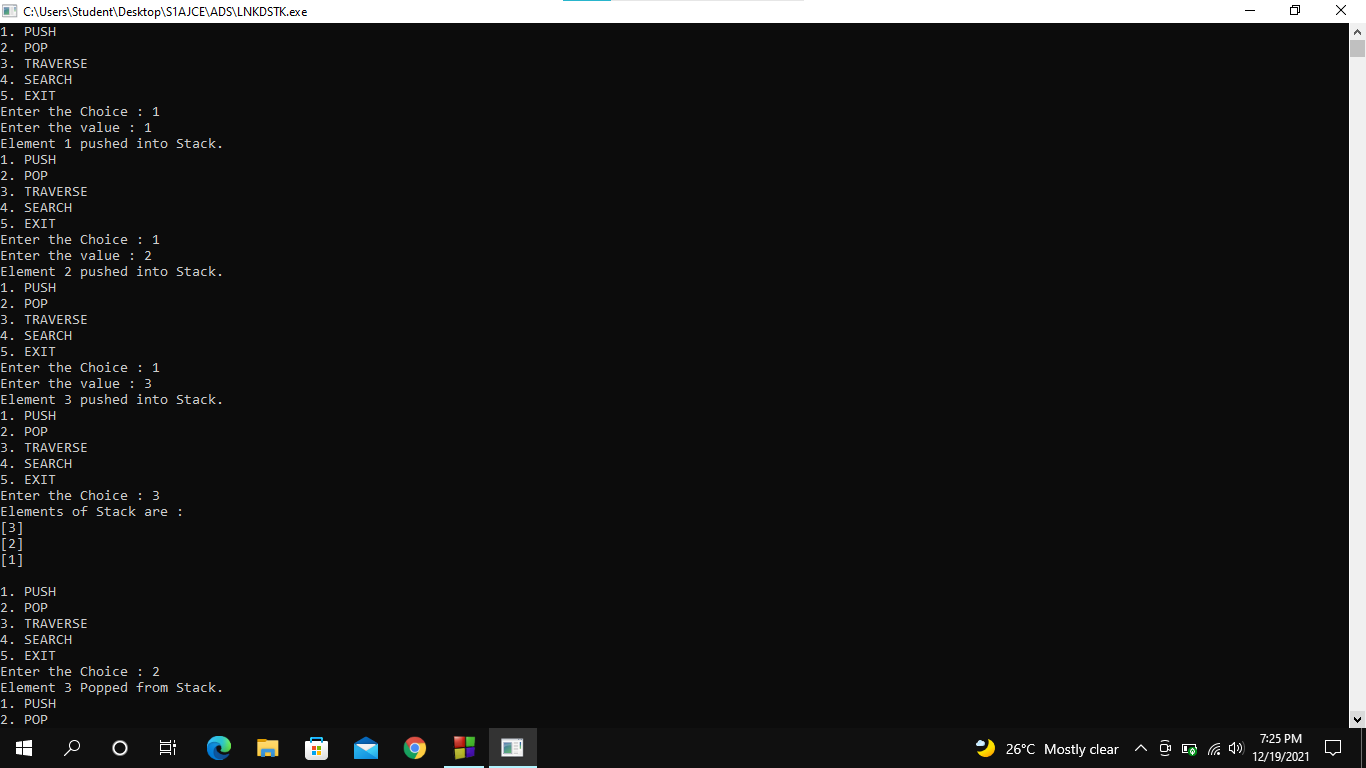
}

}

Result :

The program was executed successfully and obtained the output.

Output :



Experiment number : 3 Date :

Aim : Program to perform Doubly Linked List

Algorithm :

Step 1 : Start

Step 2 : Declare functions as insert begin(), last(), delete begin(), last(), display() and search()

Step 3 : Read the choice of user as 1. Insert begin(), 2. Insert Last(), 3. Delete Begin(), 4. Delete

Last(), 5.Search() 6. Display() 7. Exit

Step 4 : if choice is 1

then goto step 5

Else

goto step 8

Step 5 : Declare new node and allocate the memory for new node

Step 6 : Assign the value to the data part of the new node

Step 7 : Check whether the TOP is NULL

Then make the link field of the new node to NULL and set TOP=new NODE

Else

Make the link field of the new node to TOP and

set TOP = new NODE and goto step 3

Step 8 : if the choice is 2

then goto step 9

Else

The goto step 11

Step 9 : Declare a temporary NODE

Step 10 : Check whether TOP==NULL

Then print UNDERFLOW

Else

Assign TOP to the temp node

Set TOP=temp->next

free(temp) and goto step 3

Step 11 : if choice is 3

Then goto step 12

Else

Goto step 15

Step 12 : Declare a node

Step 13 : Assign the top to temp

Step 14 : Repeat while first != NULL

Display first->data

Set first=first->next and goto step 3

Step 15 : if choice is 4

Then goto step 17

Else

Goto step 22

Step 16 : Declare a temporary NODE

Step 17 : Read the item to be searched

Step 18 : Assign TOP=temp

Step 19 : Repeat for temp->next!=NULL

Check if temp->data=item

Then set flag=1

Break

temp=temp->next

Step 20 : Check that flag==1 then

Print item is found

Else

Print item not found and goto step 3

Step 21 : if choice is 5 then

Goto step 24

Else

Goto step 22

Step 22 : if choice is invalid then

Display invalid choice and goto step 3

Step 23 : stop

Code :

//DOUBLY LINKED LIST

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*prev;

int val;

struct node \*next;

};

struct node \*head;

void insbeg();

void inslast();

void delbeg();

void dellast();

void display();

void search();

void main()

{

int ch;

do

{

printf("\n1. Insert in beginning\n2. Insert at last\n3. Delete from Beginning\n4. Delete from last\n5. Search\n6. Display\n7. Exit");

printf("\nEnter your choice : ");

scanf("%d",&ch);

switch(ch)

{

case 1:

insbeg();

break;

case 2:

inslast();

break;

case 3:

delbeg();

break;

case 4:

dellast();

break;

case 5:

search();

break;

case 6:

display();

break;

case 7:

printf("Exiting....");

break;

default:

printf("Invalid choice!...");

}

}while(ch!=7);

}

//INSERT ELEMENT TO BEGINNING

void insbeg()

{

struct node \*ptr=(struct node \*)malloc(sizeof(struct node));//MEMORY ALLOCATION FOR NODE

int item;

if(ptr==NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("Enter the value : ");//GET VALUE FROM USER

scanf("%d",&item);

if(head==NULL)//IF HEAD IS NULL, SET THE DATA TO NODE

{

ptr->next=NULL;

ptr->prev=NULL;

ptr->val=item;

head=ptr;

}

else

{

ptr->val=item; //SET item TO DATA OF THE NODE

ptr->prev=NULL;//PREV WILL NULL

ptr->next=head;//NEXT WILL POINT TO HEAD

head->prev=ptr;

head=ptr;

}

printf("Node %d inserted.",item);

}

}

void inslast()//INSERT TO LAST

{

struct node \*ptr,\*temp;

int item;

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

if(ptr==NULL)

{

printf("\nUNDERFLOW...");

}

else

{

printf("Enter the value : ");

scanf("%d",&item);

ptr->val=item;

if(head==NULL)//IF HEAD IS EMPTY, THEN SET IT AS 1ST NODE

{

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("Node %d inserted",item);

}

void delbeg()//DELETE FROM BEGINNING

{

struct node \*ptr;

if(head==NULL)//IF HEAD IS NULL, NOTHING TO DELETE

{

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

}

else if(head->next==NULL)//DELETE ONLY ONE ELEMENT

{

head=NULL;

free(head);

printf("Node Deleted.");

}

else//DELETE WITH MORE THAN 1 ELEMENT

{

ptr=head;

head=head->next;

head->prev=NULL;

free(ptr);

printf("\nNode Deleted.");

}

}

void dellast()//DELETE FROM LAST

{

struct node \*ptr;

if(head==NULL)

{

printf("\nUNDERFLOW.");//NOTHING TO DELETE

}

else if(head->next==NULL)//WITH ONE ELEMENT

{

head=NULL;

free(head);

printf("Node Deleted.");

}

else//MORE THAN ONE ELEMENT

{

ptr=head;

if(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->prev->next=NULL;

free(ptr);

printf("Node Deleted.");

}

}

void display()//DISPLAY FUNCTION

{

struct node \*ptr;

printf("\nElements are : ");

ptr=head;

while(ptr!=NULL)

{

printf("[%d] <--> ",ptr->val);

ptr=ptr->next;

}

}

void search()//SEARCH FUNCTION

{

struct node \*ptr;

int item,flag=0,i=0;

ptr=head;

if(ptr==NULL)

{

printf("\nEmpty DLL.");

}

else

{

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

scanf("%d",&item);

while(ptr!=NULL)

{

if(ptr->val==item)

{

printf("Element %d is found at the position %d",item,i+1);

flag=1;

break;

}

else

{

flag=0;

}

i++;

ptr=ptr->next;

}

if(flag==0)

{

printf("Element %d is not found in DLL.",item);

}

}

}

Result :

The program was executed successfully and obtained the output.

Output :

