# ​Array implementation of list

#include<stdio.h>

#define MAX\_SIZE 100

void createList(int list[],int \*size);

void insertElement(int list[],int \*size,int element,int position);

void deleteElement(int list[],int \*size,int element);

int searchElement(int list[],int size,int element);

void displayList(int list[],int size);

int main()

{

int list[MAX\_SIZE];

int size=0;

int choice,element,position,index;

do

{

printf("1.Create List\n");

printf("2.Insert Element\n");

printf("3.Delete Element\n");

printf("4.Search Element\n");

printf("5.Display List\n");

printf("6.Exit\n");

printf("Enter your choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

createList(list,&size);

break;

case 2:

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

scanf("%d",&element);

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

scanf("%d",&position);

insertElement(list,&size,element,position);

break;

case 3:

printf("Enter the position to delete:");

scanf("%d",&position);

deleteElement(list,&size,position);

break;

case 4:

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

scanf("%d",&element);

index=searchElement(list,size,element);

if(index!=-1)

printf("Element found at position %d\n",index);

else

printf("Element not found\n");

break;

case 5:

displayList(list, size);

break;

case 6:

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

break;

default:

printf("Invlid choice!Please try again.\n");

}

printf("\n");

}

while(choice!=6);

return 0;

}

void createList(int list[],int \*size)

{

int i,count;

printf("Enter the number of elements:");

scanf("%d",&count);

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

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

{

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

}

\*size=count;

printf("List created successfully.\n");

}

void insertElement(int list[],int \*size,int element,int position)

{

int i;

if(position < 0 || position > \*size)

{

printf("invalid position! Insertion failed.\n");

return;

}

for(i= \*size;i > position; i--)

{

list[i]=list[i-1];

}

list[position]=element;

(\*size)++;

printf("Element inserterd successfully.\n");

}

void deleteElement(int list[],int \*size,int position)

{

int i;

if (position < 0 || position>=\*size)

{

printf("Invalid position1 Deletion failed.\n");

return;

}

for(i=position;i<\*size-1;i++)

{

list[i]=list[i+1];

}

(\*size)--;

printf("Element deleted successfully.\n");

}

int searchElement(int list[],int size,int element)

{

int i;

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

{

if(list[i]==element)

{

return i;

}

}

return -1;

}

void displayList(int list[],int size)

{

int i;

if(size==0)

{

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

return;

}

printf("List elemnts:");

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

{

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

}

printf("\n");

}

1.Create List

2.Insert Element

3.Delete Element

4.Search Element

5.Display List

6.Exit

Enter your choice:1

Enter the number of elements:5

Enter the elements:

10

20

30

405

89

List created successfully.

1.Create List

2.Insert Element

3.Delete Element

4.Search Element

5.Display List

6.Exit

Enter your choice:5

List elemnts:10 20 30 40 89

# 2 .Implementation of Singly Linked List in a menu driven fashion.

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

struct Node \*head=NULL;

void insertAtBeginning(int value)

{

struct Node \*newNode;

newNode=malloc(sizeof(struct Node));

newNode->data=value;

if (head==NULL)

{

newNode->next=NULL;

head=newNode;

}

else

{

newNode->next=head;

head=newNode;

}

printf("\n Node insertion successfully at beginning\n");

}

void insertAtEnd(int value)

{

struct Node \*newNode;

newNode=malloc(sizeof(struct Node));

newNode->data=value;

newNode->next=NULL;

if(head==NULL)

head=newNode;

else

{

struct Node \*temp = head;

while(temp->next!=NULL)

temp = temp->next;

temp->next=newNode;

}

printf("\n Node inserted successfully at end\n");

}

void insertPosition(int value,int pos)

{

int i=0;

struct Node \*newNode;

newNode=malloc(sizeof(struct Node));

newNode->data = value;

if(head==NULL)

{

newNode->next=NULL;

head = newNode;

}

else

{

struct Node \*temp=head;

for(i=0;i<pos-1;i++)

temp = te mp->next;

newNode->next=temp->next;

temp->next = newNode;

}

printf("\n Node inserted successfully\n");

}

void removeBeginning()

{

if(head == NULL)

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

else

{

struct Node \*temp=head;

if(head->next == NULL)

{

head=NULL;

free(temp);

}

else

{

head=temp->next;

free(temp);

printf("\n Node deleted at the beginning \n\n");

}

}

}

void removeEnd()

{

if(head == NULL)

{

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

}

else

{

struct Node \*temp1=head;

struct Node \*temp2;

if(head->next == NULL)

head=NULL;

else

{

while(temp1->next!=NULL)

{

temp2 = temp1;

temp1 =temp1->next;

}

temp2->next=NULL;

}

free(temp1);

printf("\n Node deleted at the end\n\n");

}

}

void removePosition(int pos)

{

int i,flag=1;

if (head==NULL)

printf("List is empty");

else

{

struct Node \*temp1=head;

struct Node \*temp2;

if (pos == 1)

{

head = temp1->next;

free(temp1);

printf("\n Node deleted\n\n");

}

else

{

for(i=0;i<pos-1;i++)

{

if(temp1->next!=NULL)

{

temp2 = temp1;

temp1=temp1->next;

}

else

{

flag=0;

break;

}

}

if(flag)

{

temp2->next=temp1->next;

free(temp1);

printf("\n Node deleted\n\n");

}

else

printf("Position exceeds number of elements.");

}

}

}

void display()

{

if(head==NULL)

printf("\nList is empty\n");

else

{

struct Node \*temp= head;

printf("\n\n List elements are\n");

while(temp->next!=NULL)

{

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

temp = temp->next;

}

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

}

}

int main()

{

int choice,value,pos;

while(1)

{

printf("\n\nsingly Linked list operatin\n");

printf("1.Insert at Beginning\n");

printf("2.Insert at End\n");

printf("3.Insert at position\n");

printf("4.Delete at beginning\n");

printf("5.Delete at End\n");

printf("6.Delete at position\n");

printf("7.Display\n");

printf("8.Exit\n");

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

scanf("%d",&choice);

switch(choice)

{

case 1:printf("ENTER THE VALUE TO BE INSERT:");

scanf("%d",&value);

insertAtBeginning(value);

break;

case 2:printf("Enter the value to be insert:");

scanf("%d",&value);

insertAtEnd(value);

break;

case 3:printf("Enter the value to be insert:");

scanf("%d",&value);

printf("Enter the postion after which you want to insert:");

scanf("%d",&pos);

insertPosition(value,pos);

break;

case 4:removeBeginning();

break;

case 5:removeEnd();

break;

case 6:printf("Enter the position which you want to delete:");

scanf("%d",&pos);

removePosition(pos);

break;

case 7:display();

break;

case 8:exit(0);

default:printf("\n Invalid Choice\n\n");

}

}

}

**OUTPUT**

**singly Linked list operatin**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at position**

**7.Display**

**8.Exit**

**Enter your choice:1**

**ENTER THE VALUE TO BE INSERT:10**

**Node insertion successfully at beginning**

**singly Linked list operatin**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at position**

**7.Display**

**8.Exit**

**Enter your choice:2**

**Enter the value to be insert:20**

**Node inserted successfully at end**

**singly Linked list operatin**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at position**

**7.Display**

**8.Exit**

**Enter your choice:3**

**Enter the value to be insert:30**

**Enter the postion after which you want to insert:2**

**Node inserted successfully**

**singly Linked list operatin**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at position**

**7.Display**

**8.Exit**

**Enter your choice:7**

**List elements are**

**data:10**

**data:20**

**data:30**

**singly Linked list operatin**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at position**

**7.Display**

**8.Exit**

**Enter your choice:4**

**Node deleted at the beginning**

**singly Linked list operatin**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at position**

**7.Display**

**8.Exit**

**Enter your choice:7**

**List elements are**

**data:20**

**data:30**

**singly Linked list operatin**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at position**

**7.Display**

**8.Exit**

**Enter your choice:6**

**Enter the position which you want to delete:2**

**Node deleted**

**singly Linked list operatin**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at position**

**7.Display**

**8.Exit**

**Enter your choice:7**

**List elements are**

**data:20**

**singly Linked list operatin**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at position**

**7.Display**

**8.Exit**

**Enter your choice:5**

**Node deleted at the end**

**singly Linked list operatin**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at position**

**7.Display**

**8.Exit**

**Enter your choice:7**

**List is empty**

**singly Linked list operatin**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at position**

**7.Display**

**8.Exit**

**Enter your choice:8**

**=== Code Execution Successful ===**

# ​Doubly linked list implementation in C

# #include<stdio.h>

# #include<stdlib.h>

# struct Node

# {

# int data;

# struct Node \*previous;

# struct Node \*next;

# };

# struct Node \*head=NULL;

# void insertAtBeginning(int value)

# {

# struct Node \*newNode;

# newNode=malloc(sizeof(struct Node));

# newNode->data=value;

# newNode->previous=NULL;

# if(head==NULL)

# {

# newNode->next=NULL;

# head=newNode;

# }

# else

# {

# newNode->next=head;

# head->previous=newNode;

# head=newNode;

# }

# printf("\n Insertion successful");

# display();

# }

# void insertAtEnd(int value)

# {

# struct Node \*newNode;

# newNode=malloc(sizeof(struct Node));

# newNode-> data=value;

# newNode->next=NULL;

# if(head==NULL)

# {

# newNode->previous=NULL;

# head = newNode;

# }

# else

# {

# struct Node \*temp = head;

# while(temp->next!=NULL)

# temp = temp -> next;

# newNode->previous=temp;

# temp->next=newNode;

# }

# printf("\n Insertion successful at End");

# display();

# }

# void insertAfter(int value,int pos)

# {

# int i,flag=1;

# struct Node \*newNode;

# struct Node \*temp=head;

# newNode=malloc(sizeof(struct Node));

# newNode->data = value;

# if(head == NULL)

# {

# newNode->previous=NULL;

# newNode->next=NULL;

# head = newNode;

# }

# else

# {

# for(i=0;i<pos-1;i++)

# {

# temp=temp->next;

# if(temp->next == NULL)

# {

# flag=0;

# break;

# }

# }

# if(flag)

# {

# newNode->next = temp->next;

# temp->next->previous=newNode;

# temp->next=newNode;

# newNode->previous=temp;

# printf("\n Insertion succesful at position");

# }

# else

# printf("Number of elements is less than position enterd");

# }

# display();

# }

# void deleteBeginning()

# {

# if(head == NULL)

# printf("List is Empty");

# else

# {

# struct Node \*temp=head;

# if(temp->previous==temp->next)

# {

# head=NULL;

# free(temp);

# }

# else

# {

# head=temp->next;

# head->previous=NULL;

# free(temp);

# }

# printf("\n Delettion successful");

# }

# display();

# }

# void deleteEnd()

# {

# if(head == NULL)

# printf("List is Empty");

# else

# {

# struct Node \*temp=head;

# if(temp->previous==temp->next)

# {

# head=NULL;

# free(temp);

# }

# else

# {

# while(temp->next!=NULL)

# temp = temp -> next;

# temp->previous->next=NULL;

# }

# printf("\n Deletion succesful at End");

# }

# display();

# }

# void deleteSpecific(int delvalue)

# {

# if(head == NULL)

# printf("List is Empty");

# else

# {

# struct Node \*temp = head;

# while(temp->data!=delvalue)

# {

# if(temp->next==NULL)

# {

# printf("\n Given node is not found in the list");

# return;

# }

# else

# temp=temp->next;

# }

# if(temp==head)

# {

# head=NULL;

# free(temp);

# }

# else

# {

# temp->previous->next=temp->next;

# temp->next->previous=temp->previous;

# free(temp);

# }

# printf("\n Deletion successful at specific");

# }

# display();

# }

# void display()

# {

# if(head == NULL)

# printf("\n List is Empty");

# else

# {

# struct Node \*temp = head;

# printf("\n List elements are:\n");

# printf("START<---");

# while(temp->next!=NULL)

# {

# printf("%d<----->",temp->data);

# temp = temp -> next;

# }

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

# }

# }

# int main()

# {

# int choice,value,pos;

# while(1)

# {

# printf("\n Doubly Linked List Operations\n");

# printf("1. Insert at Beginning\n");

# printf("2.Insert at End\n");

# printf("3.Insert at position\n");

# printf("4.Delete at beginning\n");

# printf("5.Delete at End\n");

# printf("6.Delete at position\n");

# printf("7.Display\n");

# printf("8.Exit\n");

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

# scanf("%d",&choice);

# switch(choice)

# {

# case 1:printf("ENTER THE VALUE TO BE INSERT:");

# scanf("%d",&value);

# insertAtBeginning(value);

# break;

# case 2:printf("Enter the value to be insert:");

# scanf("%d",&value);

# insertAtEnd(value);

# break;

# case 3:printf("Enter the value to be insert:");

# scanf("%d",&value);

# printf("Enter the postion after which you want to insert:");

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

# insertAfter(value,pos);

# break;

# case 4:deleteBeginning();

# break;

# case 5:deleteEnd();

# break;

# case 6:printf("Enter the value you want to delete:");

# scanf("%d",&value);

# deleteSpecific(value);

# break;

# case 7:display();

# break;

# case 8:exit(0);

# default:printf("\n Invalid Choice\n");

# }

# }

# }

**OUTPUT**

**Doubly Linked List Operations**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at Specific**

**7.Display**

**8.Exit**

**Enter your choice:1**

**ENTER THE VALUE TO BE INSERT:22**

**Insertion successful**

**List elements are:**

**START<---22--->NULL**

**Doubly Linked List Operations**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at Specific**

**7.Display**

**8.Exit**

**Enter your choice:2**

**Enter the value to be insert:67**

**Insertion successful at End**

**List elements are:**

**START<---22<----->67--->NULL**

**Doubly Linked List Operations**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at Specific**

**7.Display**

**8.Exit**

**Enter your choice:3**

**Enter the value to be insert:18**

**Enter the postion after which you want to insert:1**

**Insertion succesful at position**

**List elements are:**

**START<---22<----->18<----->67--->NULL**

**Doubly Linked List Operations**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at Specific**

**7.Display**

**8.Exit**

**Enter your choice:7**

**List elements are:**

**START<---22<----->18<----->67--->NULL**

**Doubly Linked List Operations**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at Specific**

**7.Display**

**8.Exit**

**Enter your choice:6**

**Enter the value you want to delete:22**

**Deletion successful at specific**

**List is Empty**

**Doubly Linked List Operations**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at position**

**4.Delete at beginning**

**5.Delete at End**

**6.Delete at Specific**

**7.Display**

**8.Exit**

**Enter your choice:8**

**=== Code Execution Successful ===**

**Circular linked list implementation in C**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct Node**

**{**

**int data;**

**struct Node \*next;**

**};**

**struct Node \*head=NULL;**

**void insertAtBeginning(int value)**

**{**

**struct Node \*newNode;**

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

**newNode->data=value;**

**if(head==NULL)**

**{**

**head=newNode;**

**newNode->next=head;**

**}**

**else**

**{**

**struct Node \*temp=head;**

**while(temp->next!=head)**

**temp=temp->next;**

**newNode->next=head;**

**head=newNode;**

**temp->next=head;**

**}**

**printf("\n Insertion successful\n");**

**display();**

**}**

**void insertAtEnd(int value)**

**{**

**struct Node \*newNode;**

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

**newNode->data=value;**

**if(head==NULL)**

**{**

**head=newNode;**

**newNode->next=head;**

**}**

**else**

**{**

**struct Node \*temp=head;**

**while(temp->next!=head)**

**temp=temp->next;**

**temp->next=newNode;**

**newNode->next=head;**

**}**

**printf("\n insertion Successful\n");**

**display();**

**}**

**void insertAfter(int value,int location)**

**{**

**struct Node \*newNode;**

**int i;**

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

**newNode->data=value;**

**if(head==NULL)**

**{**

**head=newNode;**

**newNode->next=head;**

**}**

**else**

**{**

**struct Node \*temp=head;**

**for(i=1;i<location-1;i++)**

**{**

**temp=temp->next;**

**}**

**newNode->next=temp->next;**

**temp->next=newNode;**

**printf("\n Insertion Successful");**

**}**

**display();**

**}**

**void deleteBeginning()**

**{**

**if(head==NULL)**

**printf("List is Empty");**

**else**

**{**

**struct Node \*temp=head;**

**struct Node \*last=NULL;**

**if(temp->next==head)**

**{**

**head=NULL;**

**free(temp);**

**}**

**else**

**{**

**while(temp->next!=head)**

**temp=temp->next;**

**last=temp;**

**temp=head;**

**head=head->next;**

**free(temp);**

**last->next=head;**

**}**

**printf("\n Deletion Successful");**

**}**

**display();**

**}**

**void deleteEnd()**

**{**

**if(head==NULL)**

**printf("List is Empty");**

**else**

**{**

**struct Node \*temp1=head;**

**struct Node \*temp2;**

**if(temp1->next==head)**

**{**

**head=NULL;**

**free(temp1);**

**}**

**else**

**{**

**while(temp1->next!=head)**

**{**

**temp2=temp1;**

**temp1=temp1->next;**

**}**

**temp2->next=head;**

**free(temp1);**

**}**

**printf("\n Deletion Successful");**

**}**

**display();**

**}**

**void deleteSpecific(int location)**

**{**

**int i;**

**if(head == NULL)**

**printf("List is Empty");**

**else**

**{**

**struct Node \*temp1=head;**

**struct Node \*temp2;**

**for(i=1;i<location-1;i++)**

**{**

**if(temp1->next==head)**

**printf("\n Element is not found in the list");**

**else**

**{**

**temp2=temp1;**

**temp1=temp1->next;**

**}**

**}**

**if(temp1->next==head)**

**{**

**head = NULL;**

**free(temp1);**

**}**

**else**

**{**

**if(temp1==head)**

**{**

**temp2=head;**

**while(temp2->next!=head)**

**temp2= temp2 -> next;**

**head = head -> next;**

**temp2->next=head;**

**}**

**else**

**{**

**if(temp1->next==head)**

**temp2->next=head;**

**else**

**temp2->next=temp1->next;**

**free(temp1);**

**}**

**}**

**printf("\n Deletion Successful");**

**}**

**display();**

**}**

**void display()**

**{**

**if(head == NULL)**

**printf("\nList is Empty");**

**else**

**{**

**struct Node \*temp = head;**

**printf("\n List elements are:\n");**

**while(temp -> next != head)**

**{**

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

**temp = temp -> next;**

**}**

**printf("%d------>%d",temp->data,head->data);**

**}**

**}**

**int main()**

**{**

**int choice,value,pos;**

**while(1)**

**{**

**printf("\n\nCircular Linked List operation\n");**

**printf("1.Insert at Beginning\n");**

**printf("2.Insert at End\n");**

**printf("3.Insert at location\n");**

**printf("4.Delete at Beginning\n");**

**printf("5.Delete at End\n");**

**printf("6.Delete at location\n");**

**printf("7.Display\n");**

**printf("8.Exit\n\n");**

**printf("Enter the chioce:");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1:printf("Enter the value to insert:");**

**scanf("%d",&value);**

**insertAtBeginning(value);**

**break;**

**case 2:printf("Enter the value to insert:");**

**scanf("%d",&value);**

**insertAtEnd(value);**

**break;**

**case 3:printf("Enter the value to insert:");**

**scanf("%d",&value);**

**printf("Enter the position to insert the value:");**

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

**insertAfter(value,pos);**

**break;**

**case 4:deleteBeginning();**

**break;**

**case 5:deleteEnd();**

**break;**

**case 6:printf("Enter the location to delete:");**

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

**deleteSpecific(pos);**

**break;**

**case 7:display();**

**break;**

**case 8:exit(0);**

**default:printf("\nInvalid choice\n\n");**

**}**

**}**

**}**

**Output:**

**Circular Linked List operation**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at location**

**4.Delete at Beginning**

**5.Delete at End**

**6.Delete at location**

**7.Display**

**8.Exit**

**Enter the chioce:1**

**Enter the value to insert:19**

**Insertion successful**

**List elements are:**

**19------>19**

**Circular Linked List operation**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at location**

**4.Delete at Beginning**

**5.Delete at End**

**6.Delete at location**

**7.Display**

**8.Exit**

**Enter the chioce:1**

**Enter the value to insert:20**

**Insertion successful**

**List elements are:**

**20-------->19------>20**

**Circular Linked List operation**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at location**

**4.Delete at Beginning**

**5.Delete at End**

**6.Delete at location**

**7.Display**

**8.Exit**

**Enter the chioce:2**

**Enter the value to insert:10**

**insertion Successful**

**List elements are:**

**20-------->19-------->10------>20**

**Circular Linked List operation**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at location**

**4.Delete at Beginning**

**5.Delete at End**

**6.Delete at location**

**7.Display**

**8.Exit**

**Enter the chioce:4**

**Deletion Successful**

**List elements are:**

**19-------->10------>19**

**Circular Linked List operation**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at location**

**4.Delete at Beginning**

**5.Delete at End**

**6.Delete at location**

**7.Display**

**8.Exit**

**Enter the chioce:6**

**Enter the location to delete:2**

**Deletion Successful**

**List elements are:**

**10------>10**

**Circular Linked List operation**

**1.Insert at Beginning**

**2.Insert at End**

**3.Insert at location**

**4.Delete at Beginning**

**5.Delete at End**

**6.Delete at location**

**7.Display**

**8.Exit**

**Enter the chioce:1**

**Enter the value to insert:19**

**Insertion successful**

**5.STACK**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct node**

**{**

**int info;**

**struct node \*ptr;**

**};**

**struct node \*top,\*top1,\*temp;**

**int count = 0;**

**void create()**

**{**

**top = NULL;**

**}**

**void stack\_count()**

**{**

**printf("\n No. of elements in stack : %d", count);**

**}**

**void push(int data)**

**{**

**if(top == NULL)**

**{**

**top =malloc(sizeof(struct node));**

**top->ptr= NULL;**

**top->info = data;**

**}**

**else**

**{**

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

**temp->ptr = top;**

**temp->info = data;**

**top=temp;**

**}**

**count++;**

**display();**

**}**

**void display()**

**{**

**top1=top;**

**if (top1 == NULL)**

**{**

**printf("Stack is empty");**

**return;**

**}**

**printf("\n Elements in the stack are:");**

**while(top1 != NULL)**

**{**

**printf("\t %d",top1 -> info);**

**top1 = top1 -> ptr;**

**}**

**}**

**void pop()**

**{**

**top1 = top;**

**if(top1 ==NULL)**

**{**

**printf("\n Error : Typing to pop from empty stack");**

**return;**

**}**

**else**

**top1 = top1->ptr;**

**printf("\n Popped value : %d", top->info);**

**free(top);**

**top = top1;**

**count--;**

**display();**

**}**

**int topelement()**

**{**

**return(top->info);**

**}**

**void empty()**

**{**

**if (top == NULL)**

**printf("\n Stack is Empty");**

**else**

**printf("\n Stack is not Empty with %d elements",count);**

**}**

**void destroy()**

**{**

**top1 = top;**

**while(top1 != NULL)**

**{**

**top1 = top->ptr;**

**free(top);**

**top = top1;**

**top1 = top1 -> ptr;**

**}**

**free(top1);**

**top = NULL;**

**printf("\n All stack elements destroyed");**

**count = 0;**

**}**

**void main()**

**{**

**int no,ch,val;**

**printf("\n 1.Push");**

**printf("\n 2.Pop");**

**printf("\n 3.Top");**

**printf("\n 4. Empty");**

**printf("\n 5.Display");**

**printf("\n 6.Stack Count");**

**printf("\n 7.Destroy stack");**

**printf("\n 8.Exit");**

**create();**

**while (1)**

**{**

**printf("\n Enter choice :");**

**scanf("%d", &ch);**

**switch (ch)**

**{**

**case 1:**

**printf("Enter data :");**

**scanf("%d", &no);**

**push(no);**

**break;**

**case 2:**

**pop();**

**break;**

**case 3:**

**if (top == NULL)**

**printf("No elements in stack");**

**else**

**{**

**val = topelement();**

**printf("\n Top element : %d",val);**

**}**

**break;**

**case 4:**

**empty();**

**break;**

**case 5:**

**display();**

**break;**

**case 6:**

**stack\_count();**

**break;**

**case 7:**

**destroy();**

**break;**

**case 8:**

**exit(0);**

**default:**

**printf("Invalid choice ");**

**break;**

**}**

**}**

**}**

**OUTPUT:**

**1.Push**

**2.Pop**

**3.Top**

**4. Empty**

**5.Display**

**6.Stack Count**

**7.Destroy stack**

**8.Exit**

**Enter choice :1**

**Enter data :10**

**Elements in the stack are: 10**

**Enter choice :1**

**Enter data :22**

**Elements in the stack are: 22 10**

**Enter choice :1**

**Enter data :33**

**Elements in the stack are: 33 22 10**

**Enter choice :1**

**Enter data :21**

**Elements in the stack are: 21 33 22 10**

**Enter choice :6**

**No. of elements in stack : 4**

**6.QUEUE:**

**#include<stdio.h>**

**#include<stdlib.h>**

**int count=0;**

**struct node**

**{**

**int info;**

**struct node \*ptr;**

**};**

**struct node \*front,\*rear,\*temp,\*front1;**

**void create()**

**{**

**front=rear=NULL;**

**}**

**void queuesize()**

**{**

**printf("\n Queue size:%d",count);**

**}**

**int frontelement()**

**{**

**if((front!=NULL)&&(rear!=NULL))**

**return (front->info);**

**else**

**return 0;**

**}**

**void empty()**

**{**

**if((front==NULL)&&(rear==NULL))**

**printf("\n Queue empty");**

**}**

**void enq(int data)**

**{**

**if(rear==NULL)**

**{**

**rear=(struct node\*)malloc(1\*sizeof(struct node));**

**rear->ptr=NULL;**

**rear->info=data;**

**front=rear;**

**}**

**else**

**{**

**temp=(struct node\*)malloc(1\*sizeof(struct node));**

**rear->ptr=temp;**

**temp->info=data;**

**temp->ptr=NULL;**

**rear=temp;**

**}**

**count++;**

**printf("\n List of elements in the enqueue\n");**

**display();**

**}**

**void deq()**

**{**

**front1 = front;**

**if(front1 == NULL)**

**{**

**printf("\n Error:Trying to display elements from empty queue");**

**return;**

**}**

**else**

**if(front1->ptr!=NULL)**

**{**

**front1=front1->ptr;**

**printf("\n Dequeue value:%d",front->info);**

**free(front);**

**front=front1;**

**}**

**else**

**{**

**printf("\n Dequeue value:%d",front->info);**

**free(front);**

**front=NULL;**

**rear=NULL;**

**}**

**count--;**

**printf("\n List of Elements in the queue:\n");**

**display();**

**}**

**void display()**

**{**

**front1=front;**

**if((front1==NULL)&&(rear==NULL))**

**{**

**printf("Queue is empty");**

**return;**

**}**

**while(front1!=rear)**

**{**

**printf("%d\n",front1->info);**

**front1=front1->ptr;**

**}**

**if(front1==rear)**

**printf("%d",front1->info);**

**}**

**void main()**

**{**

**int ch,value;**

**printf("\n 1:Enqueue");**

**printf("\n 2:Dequeue");**

**printf("\n 3:front element");**

**printf("\n 4:empty");**

**printf("\n 5:Display");**

**printf("\n 6:Queue");**

**printf("\n 7:Exit");**

**create();**

**while(1)**

**{**

**printf("\n Enter choice:");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1:**

**printf("ENTER DATA:");**

**scanf("%d",&value);**

**enq(value);**

**break;**

**case 2:**

**deq();**

**break;**

**case 3:**

**value=frontelement();**

**if(value!=0)**

**printf("Front element:%d",value);**

**else**

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

**break;**

**case 4:**

**empty();**

**break;**

**case 5:**

**display();**

**break;**

**case 6:**

**queuesize();**

**break;**

**case 7:**

**exit(0);**

**default:**

**printf("Invalid choice:");**

**break;**

**}**

**}**

**}**

**OUTPUT:**

**1:Enqueue**

**2:Dequeue**

**3:front element**

**4:empty**

**5:Display**

**6:Queue**

**7:Exit**

**Enter choice:1**

**ENTER DATA:10**

**List of elements in the enqueue**

**10**

**Enter choice:1**

**ENTER DATA:34**

**List of elements in the enqueue**

**10**

**34**

**Enter choice:1**

**ENTER DATA:18**

**List of elements in the enqueue**

**10**

**34**

**18**

**Enter choice:3**

**Front element:10**

**Enter choice:5**

**10**

**34**

**18**

**Enter choice:6**

**Queue size:3**

**7.HEAP**

**#include<stdio.h>**

**#include<stdlib.h>**

**int array[100],n;**

**void display();**

**void insert(int,int);**

**int main()**

**{**

**int choice,num;**

**n=0;**

**while(1)**

**{**

**printf("1.Insert the element \n");**

**printf("2.Display all elements \n");**

**printf("3.Quit \n");**

**printf("Enter your choice :");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1:**

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

**scanf("%d",&num);**

**insert(num,n);**

**n = n+1;**

**break;**

**case 2:**

**display();**

**break;**

**case 3:**

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

**exit(0);;**

**deafault:**

**printf("Invalid choice\n");**

**}**

**}**

**}**

**void display()**

**{**

**int i;**

**if (n == 0)**

**{**

**printf("Heap is empty\n");**

**return;**

**}**

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

**printf("%d", array[i]);**

**printf("\n");**

**}**

**void insert(int num,int location)**

**{**

**int parentnode;**

**while(location > 0)**

**{**

**parentnode =(location - 1)/2;**

**if(num <= array[parentnode])**

**{**

**array[location] = num;**

**return;**

**}**

**array[location] = array[parentnode];**

**location = parentnode;**

**}**

**array[0] = num;**

**}**

**OUTPUT:**

**1.Insert the element**

**2.Display all elements**

**3.Quit**

**Enter your choice :1**

**Enter the element to be inserted to the list:12**

**1.Insert the element**

**2.Display all elements**

**3.Quit**

**Enter your choice :1**

**Enter the element to be inserted to the list:22**

**1.Insert the element**

**2.Display all elements**

**3.Quit**

**Enter your choice :2**

**2212**

**1.Insert the element**

**2.Display all elements**

**3.Quit**

**Enter your choice :3**

**Exiting...**

**8.INSERTION**

**#include <stdio.h>**

**void insertionSort(int arr[], int n)**

**{**

**int i, j, key;**

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

**{**

**key = arr[i];**

**j = i - 1;**

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

**{**

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

**j = j - 1;**

**}**

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

**}**

**}**

**int main()**

**{**

**int n;**

**printf("Enter the number of elements: ");**

**scanf("%d", &n);**

**int arr[n];**

**printf("Enter %d elements: ", n);**

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

**{**

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

**}**

**printf("Original array: ");**

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

**{**

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

**}**

**printf("\n");**

**insertionSort(arr, n);**

**printf("Sorted array: ");**

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

**{**

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

**}**

**printf("\n");**

**return 0;**

**}**

**Output:**

**Enter the number of elements: 5 Enter 5 elements: 3 2 4 1 6**

**Original array: 3 2 4 1 6**

**Sorted array: 1 2 3 4 6**

**9. Merge Sort in C**

**#include <stdio.h>**

**void merge(int arr[], int l, int m, int r)**

**{**

**int i, j, k;**

**int n1 = m - l + 1;**

**int n2 = r - m;**

**int L[n1], R[n2];**

**for (i = 0; i < n1; i++)**

**L[i] = arr[l + i];**

**for (j = 0; j < n2; j++)**

**R[j] = arr[m + 1 + j];**

**i = 0;**

**j = 0;**

**k = l;**

**while (i < n1 && j < n2)**

**{**

**if (L[i] <= R[j]) {**

**arr[k] = L[i]; i++;**

**}**

**else {**

**arr[k] = R[j]; j++;**

**} k++;**

**}**

**while (i < n1) {**

**arr[k] = L[i];**

**i++;**

**k++;**

**}**

**while (j < n2) {**

**arr[k] = R[j];**

**j++;**

**k++;**

**}**

**}**

**void mergeSort(int arr[], int l, int r) { if (l < r) {**

**int m = l + (r - l) / 2;**

**mergeSort(arr, l, m);**

**mergeSort(arr, m + 1, r);**

**merge(arr, l, m, r);**

**}**

**}**

**int main()**

**{**

**int n;**

**printf("Enter the number of elements: ");**

**scanf("%d", &n);**

**int arr[n];**

**printf("Enter %d elements: ", n);**

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

**{**

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

**}**

**printf("Original array: ");**

**for(int i=0; i<n; i++){**

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

**}**

**printf("\n");**

**mergeSort(arr, 0, n - 1);**

**printf("Sorted array: ");**

**for(int i=0; i<n; i++){**

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

**}**

**printf("\n");**

**return 0;**

**}**

**Output:**

**Enter the number of elements: 5 Enter 5 elements: 90 50 30 40 10**

**Original array: 90 50 30 40 10**

**Sorted array: 10 30 40 50 90**

**10.Infix to postfix conversion using stack**

**#include<stdio.h>**

**#include<ctype.h>**

**char stack[100];**

**int top = -1;**

**void push(char x)**

**{**

**stack[++top] = x;**

**}**

**char pop()**

**{**

**if(top == -1)**

**return -1;**

**else**

**return stack[top--];**

**}**

**int priority(char x)**

**{**

**if(x == '(')**

**return 0;**

**if(x == '+' || x == '-')**

**return 1;**

**if(x == '\*' || x == '/')**

**return 2;**

**return 0;**

**}**

**int main()**

**{**

**char exp[100];**

**char \*e, x;**

**printf("Enter the expression : ");**

**scanf("%s",exp);**

**printf("\n");**

**e = exp;**

**while(\*e != '\0')**

**{**

**if(isalnum(\*e))**

**printf("%c ",\*e);**

**else if(\*e == '(')**

**push(\*e);**

**else if(\*e == ')')**

**{**

**while((x = pop()) != '(')**

**printf("%c ", x);**

**}**

**else**

**{**

**while(priority(stack[top]) >= priority(\*e))**

**printf("%c ",pop());**

**push(\*e);**

**}**

**e++;**

**}**

**while(top != -1)**

**{**

**printf("%c ",pop());**

**}**

**return 0;**

**}**

**Output:**

**Enter the expression : a(b-c)+(d/e)**

**a b c - d e / +**