```
/*
Name:
Class:
Roll No.
*/

/*
Assignment no:7
Department of Computer Engineering has student's club named 'Pinnacle
Club'.
Students of Second, third and final year of department can be granted
membership
on request. Similarly one may cancel the membership of club. First node
is reserved
for president of club and last node is reserved for secretary of club.
Write C++
program to maintain club member‘s information using singly linked list.
Store
student PRN and Name. Write functions to
a) Add and delete the members as well as president or even secretary.
b) Compute total number of members of club
c) Display members
d) Display list in reverse order using recursion
e) Two linked lists exists for two divisions. Concatenate two lists.
*/

#include <iostream>
#include <string.h>
using namespace std;

//Node
struct node {
      int prn;
      string name;
      struct node *next;
};

//Linked List
class list {
      node *head, *temp;
      public:
            list() {
                  head = NULL;
            }
            node *create(int val, string n);
            void insertEnd();
            void insertBeg();
            void deleteAt(int i);
            void insertAt(int i);
            void display();
            int count();
            void reverse();
            void rev(node *t);
            node* readAt(int i);
            void concatenate(list A,list B);
            void op();
};
```

```
//Create
node* list::create(int val, string n) {
      temp = new(struct node);
      if (temp == NULL) {
            cout<<"Memory Allocation Failed!"<<endl;
            return 0;
      } else {
            temp -> prn = val;
            temp -> name = n;
            temp -> next = NULL;
            return temp;
      }
}

//Insert End
void list::insertEnd() {
      int val;
      string n;
      cout<<"Enter PRN: ";
      cin>>val;
      cout<<"Enter Name: ";
      cin>>n;
      struct node *t = head;
      temp = create(val,n);
      if (head == NULL) {
            head = temp;
            head -> next = NULL;
      } else {
            while ((t -> next) != NULL) {
                  t = t -> next;
            }
            temp -> next = NULL;
            t -> next = temp;
            cout<<"Element Inserted at Last"<<endl;
      }
}

//Insert At
void list::insertAt(int i) {
      int val,pos = i - 1,counter = 1;
      string n;
      struct node *ptr;
      struct node *t = head;
      while ((t -> next) != NULL) {
      //loop to count number of items in linked list.
                  t = t -> next;
                  counter++;
      }
      t = head;                                           //traverse
pointer is pointed to head again.
      if (i == 1) {
      //equivalent to insert at start.
            insertBeg();
      } else if (pos > counter || i <= 0) {                            //if
position is greater than the actual linked list.
            cout<<"Entered position is out of scope."<<endl;
      } else {                                            //insert at
required position.
```

```
            cout<<"Enter PRN: ";
            cin>>val;
            cout<<"Enter Name: ";
            cin>>n;
            temp = create(val,n);
            while (pos--) {
                  ptr = t;
                  t = t -> next;
            }
            temp -> next = t;
            ptr -> next = temp;
            cout<<"Member Inserted at Position: "<<i<<endl;
      }
}

//Delete At
void list::deleteAt(int i) {
      int val,pos = i - 1,counter = 1;
      string n;
      struct node *ptrl,*ptrr;
      struct node *t = head;
      while ((t -> next) != NULL) {
                  t = t -> next;
                  counter++;
      }
      t = head;
      if (i == 1) {
            ptrl = head;
            head = head -> next;
            delete ptrl;
      } else if (pos > counter || i <= 0) {
            cout<<"Entered member doesn't exist."<<endl;
      } else {
            while (pos--) {
                  ptrl = t;
                  t = t -> next;
                  ptrr = t -> next;
            }
            ptrl -> next = ptrr;
            delete t;
            cout<<"Member Deleted at Position: "<<i<<endl;
      }
}

//Insert Beg
void list::insertBeg() {
      int val;
      string n;
      cout<<"Enter PRN: ";
      cin>>val;
      cout<<"Enter Name: ";
      cin>>n;
      //v = val;
      struct node *t = head;
      temp = create(val,n);
      if (head == NULL) {
            head = temp;
            head -> next = NULL;
      } else {
            temp -> next = head;
            head = temp;
            cout<<"We have a New President."<<endl;
      }
}

//Display
void list::display() {
      temp = head;
      cout<<"President: ";
      cout<< temp -> prn<<" – "<<temp -> name<<" -> ";
      if(temp -> next != NULL) {
            temp = temp -> next;
      }
      while (temp -> next != NULL) {
            cout<< temp -> prn<<" – "<<temp -> name<<" -> ";
            temp = temp -> next;
      }
      cout<<"Secretary: ";
      cout<< temp -> prn<<" – "<<temp -> name<<" -> ";
      cout<<"NULL"<<endl;
}

//Count
int list::count() {
      temp = head;
      int ct = 0;
      while (temp != NULL) {
            ct++;
            temp = temp -> next;
      }
      return ct;
}

//Concatenate
void list::concatenate(list A,list B) {
      struct node * last,*last1;
      node* t = A.head;
      while (t != NULL) {
            int val = t -> prn;
            string n = t -> name;
            temp = create(val,n);
            if (head == NULL) {
                  head = temp;
                  head -> next = NULL;
                  last=head;
            } else {
                  //temp -> next = NULL;
                  last -> next = t;
                  last=t;
            }
            t = t -> next;
      }
      last -> next = B.head;
      t = B.head;
      while (t != NULL) {
            int val = t -> prn;
            string n = t -> name;
            temp = create(val,n);

                  last -> next = temp;
                  last= temp;

            t = t -> next;
      }
      last->next=NULL;
}
//Accept
void list::op() {
      while(1) {
            int choice;
            cout<<"\nEnter: \n1. Add \n2. Delete \n3. Member's Count \n4.
Display \n5. Reverse the List \n0. Prev Menu"<<endl;
            cin>>choice;
            switch(choice) {
                  case 1: { //Add
                        char c;
                        cout<<"\nEnter: \nA. Add President \nB. Add
Secretary \nC. Add Member"<<endl;
                        cin>>c;
                        switch(c) {
                              case 'A':
                              case 'a':{
                                    insertBeg();
                                    break;
                              }
                              case 'B':
                              case 'b': {
                                    insertEnd();
                                    break;
                              }
                              case 'C':
                              case 'c': {
                                    insertAt(2);
                                    break;
                              }
                        }
                        break;
                  }
                  case 2: { //Delete
                        char c;
                        cout<<"\nEnter: \nA. Delete President \nB. Delete
Secretary \nC. Delete Member"<<endl;
                        cin>>c;
                        switch(c) {
                              case 'A': {
                                    deleteAt(1);
                                    cout<<"Club must have a President.
Enter Details"<<endl;
                                    insertBeg();
                                    break;
                              }
                              case 'B': {
                                    deleteAt(count());
                                    cout<<"Club must have a Secretary.
Enter Details"<<endl;
                                    insertEnd();
                                    break;
                              }
                              case 'C': {
                                    int j;
                                    cout<<"Enter Position for
Deletion"<<endl;
                                    cin>>j;
                                    deleteAt(j);
                                    break;
                              }
                        }
                        break;
                  }
                  case 3: { //Count
                        cout<<"Count: "<<count()<<endl;
                        break;
                  }
                  case 4: { //Display
                        if (head == NULL) {
                              cout<<"NULL"<<endl;
                              break;
                        } else {
                              display();
                              break;
                        }

                  }
                  case 5: { //Reverse
                        reverse();
                        break;
                  }
                  case 0: { //Prev Menu
                        return;
                  }
            }
      }
}

//Reverse Recursion
void list::rev(node *t) {
      if(t -> next != NULL) {
            rev (t -> next);
      }
      if(t == head)
            cout<<"Secretary: "<<t -> prn<<" – "<<t -> name<<endl;
      else if(t -> next == NULL)
            cout<<"President: "<<t -> prn<<" – "<<t -> name<<" -> ";
      else
            cout<<"Member: "<<t -> prn<<" – "<<t -> name<<" -> ";
}

//Reverse
void list::reverse() {
      rev(head);
}
```

```
//Read At
node* list::readAt(int i) {
      struct node *t = head;
      int c = count();
      while(c--) {
            t = t-> next;
      }
}

//Main
int main() {
      list L,X,Y;
      int c;
      while(1) {
            cout<<"Enter: \n1. List A \n2. List B \n3. Concatenate\n0.
Exit"<<endl;
            cin>>c;
            switch(c) {
                  case 1: cout<<"\nList A:"; X.op(); break;
                  case 2: cout<<"\nList B:"; Y.op(); break;
                  case 3: L.concatenate(X,Y); L.display(); break;
                  case 0: return 0;
            }
      }
}
```

```
/*Write C++ program for storing binary number using doubly linked lists.
Write functions a) To compute 1‘s and 2‘s complement b) Add two binary
numbers*/
#include<iostream>
using namespace std;
class binary;
class node
{
	node *prev;
	bool n;
	node*next;
public:
	node()
	{
		prev=next=NULL;
	}
	node(bool b)
	{
		n=b;
		prev=next=NULL;
	}
	friend class binary;
};

class binary
{
	node *start;

	public:
		binary()
		{
			start=NULL;
		}
		void generateBinary(int no);
		void displayBinary();
		void onesComplement();
		void twoscomplement();
			binary operator +(binary n1);
	bool addBitAtBegin(bool val)
	{
		node *nodee=new node(val);
		if(start==NULL)
		{
			start=nodee;
		}
		else
		{
			nodee->next=start;
			start->prev=nodee;
			start=nodee;
		}
		return true;
	}
};

void binary::generateBinary(int no)
{
	bool rem;
	node *p;
	rem=no%2;
	start=new node(rem);
	no=no/2;
	while(no!=0)
	{
		rem=no%2;
		no=no/2;

	/*
		if(start==NULL)
		{
			start=new node(rem);
		//	cout<<" Start prev: "<<start->prev;
		//	cout<<" Start next: "<<start->next ;

		}
		else
		{
		*/
			p=new node(rem);
			p->next=start;
			start->prev=p;
		//	cout<<" Start prev: "<<start->prev->n;
		//	cout<<"   p->n"<<p->n;
			start=p;

		//}
	}
}
void binary::displayBinary()
{
	node *t;
	t=start;
	while(t!=NULL)
	{
		cout<<t->n;
		t=t->next;
	}

}
void binary::onesComplement()
{
	node *t;
	t=start;

	while(t!=NULL)
	{
		if(t->n==0)
			t->n=1;
		else
			t->n=0;

		t=t->next;

	}
}
binary binary::operator +(binary n1)
```

```
{
      binary sum;
      node *a=start;
      node *b=n1.start;
//    bit *s=sum.start;
      bool carry=false;
      while(a->next!=NULL)
            a=a->next;
      while(b->next!=NULL)
            b=b->next;

      while(a!=NULL && b!=NULL)
      {
            sum.addBitAtBegin((a->n)^(b->n)^carry);
            carry=((a->n&& b->n) || (a->n&& carry) || (b->n && carry));

            a=a->prev;
            b=b->prev;
      }
      while(a!=NULL)
      {
            sum.addBitAtBegin(a->n^carry);
            a=a->prev;
      }
      while(b!=NULL)
      {
            sum.addBitAtBegin(b->n^carry);
            b=b->prev;
      }
      sum.addBitAtBegin(carry);
      return sum;
}
void binary::twoscomplement()
{
      onesComplement();
      bool carry=1;
      node *t;
      t=start;
      while(t->next!=NULL)
      {
            t=t->next;
      }
      while(t!=NULL)
      {
      if(t->n==1&& carry==1)
      {
            t->n=0;
            carry=1;
      }
      else
      if(t->n==0&& carry==1)
      {
            t->n=1;
            carry=0;
      }
      else
      if(carry==0)
      break;
      t=t->prev;
}
displayBinary();
}
int main()
{
      int num,num1;
      binary n1,n3,n2;
      int choice=1;
      do
      {
            cout<<"\n\n=========Binary Number Operations========\n";
            cout<<"1. Generate binary\n2.One's Complement\n3.Two's 
Complement\n4. Addition\n0.Exit\nEnter your choice: ";
            cin>>choice;
            switch(choice)
            {
                  case 1: cout<<"\nENter Number in decimal form: ";
                              cin>>num;
                              n1.generateBinary(num);
                              cout<<"\nBinary Representation: ";
                              n1.displayBinary();
                              break;
                  case 2:cout<<"\nENter Number in decimal form: ";
                              cin>>num;
                              n1.generateBinary(num);
                              cout<<"\nBinary Representation: ";
                              n1.displayBinary();
                              cout<<"\nOnes Complement: ";
                              n1.onesComplement();
                              n1.displayBinary();
                              break;
                  case 3:cout<<"\nENter Number in decimal form: ";
                              cin>>num;
                              n1.generateBinary(num);
                              cout<<"\nBinary Representation: ";
                              n1.displayBinary();
                              cout<<"\nTwos complement; ";
                              n1.twoscomplement();
                              break;
                  case 4: cout<<"\nENter Two Numbers: ";
                              cin>>num>>num1;
                              n1.generateBinary(num);
                              n2.generateBinary(num1);
                              n1.displayBinary();
                              cout<<" + ";
                              n2.displayBinary();
                              cout<<"= ";
                              n3=n1+n2;
                              n3.displayBinary();




            }
      }while(choice!=0);
      n1.generateBinary(7);
```

```
        cout<<"\nBinary Representation: ";
        n1.displayBinary();
//
//      cout<<"\nOnes Complement: ";
//      n1.displayBinary();
        cout<<"\nTwos complement; ";
        n1.twoscomplement();
        return 0;
}
```

```
/*  Assignment No: 9
    In any language program mostly syntax error occurs due to unbalancing
delimiter such as (),{},[]. Write C++ program using stack to check
whether given expression is well parenthesized or not.
   */

#include <iostream>
using namespace std;
#define size 10

class stackexp
{
    int top;
    char stk[size];
public:
    stackexp()
    {
     top=-1;
    }
    void push(char);
    char pop();
    int isfull();
    int isempty();
};

void stackexp::push(char x)
{
    top=top+1;
    stk[top]=x;
}

char stackexp::pop()
{
    char s;
    s=stk[top];
    top=top-1;
    return s;
}

int stackexp::isfull()
{
    if(top==size)
        return 1;
    else
        return 0;
}

int stackexp::isempty()
{
    if(top==-1)
        return 1;
    else
        return 0;
}

int main()
{
    stackexp s1;
    char exp[20],ch;
    int i=0;
    cout << "\n\t!! Parenthesis Checker..!!!!" << endl; // prints
!!!Hello World!!!
    cout<<"\nEnter the expression to check whether it is in well form or
not :  ";
    cin>>exp;
    if((exp[0]==')')||(exp[0]==']')||(exp[0]=='}'))
    {
        cout<<"\n Invalid Expression.....\n";
        return 0;
    }
    else
    {
        while(exp[i]!='\0')
        {
            ch=exp[i];
            switch(ch)
            {
            case '(':s1.push(ch);break;
            case '[':s1.push(ch);break;
            case '{':s1.push(ch);break;
            case ')':s1.pop();break;
            case ']':s1.pop();break;
            case '}':s1.pop();break;
            }
            i=i+1;
        }
    }
    if(s1.isempty())
    {
        cout<<"\nExpression is well parenthesised...\n";
    }
    else
    {
        cout<<"\nSorry !!! Invalid Expression or not in well
parenthesized....\n";
    }
    return 0;
}
```

```
#include<iostream>
#include<string.h>
using namespace std;
class stackop
{  char  st[20],st1[20];  int top,top1,ss[10],e1,e2,e3,flag;
   public:
    void input();
    void push(char a);
    void pop();
    int pri(char b);
    void eval();
    void push1(int d);
    void pop1();
 };
 int stackop::pri(char b)
 {   if(b=='-')
       return 1;
        if(b=='+')
       return 2;
        if(b=='/')
       return 3;
        if(b=='*')
       return 4;
 }

void stackop::input()
{   char ch[20];   top=-1;   int f=1,l,i=0,j=0;   flag=0;
     cout<<"\n enter the expression\n";
     cin>>ch;
     l=strlen(ch);



     while(i<l)
     {     f=1;
         if(isalpha(ch[i]))
           {    cout<<ch[i]; st1[j]=ch[i]; j++;  flag=1;   }
           if(isdigit(ch[i]))
           {  cout<<ch[i];  st1[j]=ch[i]; j++;   }
         if(ch[i]=='(')
           {    push(ch[i]); }
          if(ch[i]==')')
           {
             while(st[top]!='(')
              {   cout<<st[top]; st1[j]=st[top]; j++; pop();}
                  pop();
           }
           if((ch[i]=='+')||(ch[i]=='-')||(ch[i]=='*')||(ch[i]=='/'))
           {    while(f==1)
                {
                  if(top==-1)
                  {   push(ch[i]); f=0;   }
                  else
                  {   if(st[top]=='(')
                      {   push(ch[i]);  f=0; }
                      else
                      {
                             if((pri(ch[i]))>(pri(st[top])))
```

```
                              {         push(ch[i]); f=0;       }
                              else
                              {      cout<<st[top]; st1[j]=st[top]; j++;
pop();       }

                       }
                    }
                  }
             }
      i++;
     }
    while(top!=-1)
     { cout<<st[top]; st1[j]=st[top]; j++; pop();   }  cout<<"\n";
     cout<<st1;
}

void stackop::eval()
{  int j=0;  top1=-1;
  if(flag==0)
{
 while(j<strlen(st1))
{
    if(st1[j]=='1')
      push1(1);
if(st1[j]=='2')
      push1(2);
if(st1[j]=='3')
      push1(3);
if(st1[j]=='4')
      push1(4);
if(st1[j]=='5')
      push1(5);
if(st1[j]=='6')
      push1(6);
if(st1[j]=='7')
      push1(7);
if(st1[j]=='8')
      push1(8);
if(st1[j]=='9')
      push1(9);
if(st1[j]=='0')
      push1(0);
if(st1[j]=='+')
{      e1=ss[top1]; pop1();
       e2=ss[top1]; pop1();
       e3=e2+e1;
       push1(e3);
}
if(st1[j]=='-')
 {      e1=ss[top1]; pop1();
       e2=ss[top1]; pop1();
       e3=e2-e1;
       push1(e3);
}

if(st1[j]=='*')
{      e1=ss[top1]; pop1();
       e2=ss[top1]; pop1();
       e3=e2*e1;
       push1(e3);
}

if(st1[j]=='/')
{      e1=ss[top1]; pop1();
       e2=ss[top1]; pop1();
       e3=e2/e1;
       push1(e3);
} j++;
}
cout<<"\n evaluated value:";
cout<<ss[0];
}
else
{  cout<<"\n cannot evaluate given input";
}
}

void stackop::push(char a)
{    top++; st[top]=a;   }
void stackop::pop()
{  top--;         }
void stackop::push1(int d)
{    top1++; ss[top1]=d;   }
void stackop::pop1()
{  top1--;         }

int main()
{  stackop s;
   s.input();
   s.eval();
   cout<<"\n";
   return 0;
}
```

```
/* Assignment No.11
    Queues are frequently used in computer programming, and a typical
example is the creation of a job queue by an operating system. If the
operating system does not use priorities, then the jobs are processed in
the order they enter the system. Write C++ program for simulating job
queue. Write functions to add job and delete job from queue.
    4*/

#include <iostream>
#define MAX 10
using namespace std;
struct queue
{       int data[MAX];
      int front,rear;
};
class Queue
{    struct queue q;
   public:
      Queue(){q.front=q.rear=-1;}
      int isempty();
      int isfull();
      void enqueue(int);
      int delqueue();
      void display();
};
int Queue::isempty()
{
      return(q.front==q.rear)?1:0;
}
int Queue::isfull()
{    return(q.rear==MAX-1)?1:0;}
void Queue::enqueue(int x)
{q.data[++q.rear]=x;}
int Queue::delqueue()
{return q.data[++q.front];}
void Queue::display()
{   int i;
    cout<<"\n";
    for(i=q.front+1;i<=q.rear;i++)
           cout<<q.data[i]<<" ";
}
int main()
{      Queue obj;
      int ch,x;
      do{    cout<<"\n 1. insert job\n 2.delete job\n 3.display\n
4.Exit\n Enter your choice:";
             cin>>ch;
      switch(ch)
      {  case 1: if (!obj.isfull())
                 {   cout<<"\n Enter data:";
                    cin>>x;
                    obj.enqueue(x);
                 }
                  else
                    cout<< "Queue is overflow";
                   break;
         case 2: if(!obj.isempty())
                      cout<<"\n Deleted Element="<<obj.delqueue();
```

```
                else
                  {   cout<<"\n Queue is underflow";  }
                cout<<"\nremaining jobs :";
                obj.display();
                 break;
        case 3: if (!obj.isempty())
              {  cout<<"\n Queue contains:";
                   obj.display();
              }
              else
                     cout<<"\n Queue is empty";
             break;
        case 4: cout<<"\n Exit";
        }
      }while(ch!=4);
return 0;
}
```

```
/*
 * C++ Program to Implement Priority Queue
 Priority: 1-Low to 10-High
 */
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
using namespace std;

/*
 * Node Declaration
 */
struct node
{
      int priority;
      int info;
      struct node *link;
};
/*
 * Class Priority Queue
 */
class Priority_Queue
{
    private:
        node *front;
    public:
        Priority_Queue()
        {
            front = NULL;
        }
        /*
         * Insert into Priority Queue
         */
        void insert(int item, int priority)
        {
            node *tmp, *q;
            tmp = new node;
            tmp->info = item;
            tmp->priority = priority;
            if (front == NULL || priority < front->priority)
            {
                tmp->link = front;
                front = tmp;
            }
            else
            {
                q = front;
                while (q->link != NULL && q->link->priority <= priority)
                    q=q->link;
                tmp->link = q->link;
                q->link = tmp;
            }
        }
        /*
         * Delete from Priority Queue
         */
        void del()
        {
            node *tmp;
            if(front == NULL)
                cout<<"Queue Underflow\n";
            else
            {
                tmp = front;
                cout<<"Deleted item is: "<<tmp->info<<endl;
                front = front->link;
                free(tmp);
            }
        }
        /*
         * Print Priority Queue
         */
        void display()
        {
            node *ptr;
            ptr = front;
            if (front == NULL)
                cout<<"Queue is empty\n";
            else
            {    cout<<"Queue is :\n";
                cout<<"Priority       Item\n";
                while(ptr != NULL)
                {
                    cout<<ptr->priority<<"                 "<<ptr-
>info<<endl;
                    ptr = ptr->link;
                }
            }
        }
};
/*
 * Main
 */
int main()
{
    int choice, item, priority;
    Priority_Queue pq;
    do
    {
        cout<<"1.Insert\n";
        cout<<"2.Delete\n";
        cout<<"3.Display\n";
        cout<<"4.Quit\n";
        cout<<"Enter your choice : ";
        cin>>choice;
        switch(choice)
        {
        case 1:
            cout<<"Input the item value to be added in the queue : ";
            cin>>item;
            cout<<"Enter its priority : ";
            cin>>priority;
            pq.insert(item, priority);
            break;
        case 2:
            pq.del();
            break;
        case 3:
            pq.display();
            break;
        case 4:
            break;
        default :
            cout<<"Wrong choice\n";
        }
    }
    while(choice != 4);
    return 0;
}
```

```
/* Assignment No.=13: A double-ended queue (deque) is a linear list in
which additions and deletions may be made at either end. Obtain a data
representation mapping a deque into a onedimensional array. Write C++
program to simulate deque with functions to add and delete elements from
either end of the deque*/
#include<iostream>
//#include
//#include
using namespace std;
#define SIZE 5
// ERROR HANDLINH NOT DOne

//    program is not working correct.
//
class dequeue
{

      int a[10],front,rear,count;

public:
      dequeue();
      void add_at_beg(int);
      void add_at_end(int);
      void delete_fr_front();
      void delete_fr_rear();
      void display();
};


dequeue::dequeue()
{
      front=-1;
      rear=-1;
      count=0;
}


void dequeue::add_at_beg(int item)
{
      int  i;
      if(front==-1)
      {
            front++;
            rear++;
            a[rear]=item;
            count++;
      }
      else if(rear>=SIZE-1)
      {
            cout<<"\nInsertion is not possible,overflow!!!!";
      }
      else
      {
            for(i=count;i>=0;i--)
            {
                  a[i]=a[i-1];
            }
            a[i]=item;
            count++;
            rear++;
      }
}



void dequeue::add_at_end(int item)
{

      if(front==-1)
      {
            front++;
            rear++;
            a[rear]=item;
            count++;
      }
      else if(rear>=SIZE-1)
      {
            cout<<"\nInsertion is not possible,overflow!!!";
            return;
      }
      else
      {
            a[++rear]=item;
      }


}



void dequeue::display()
{

      for(int i=front;i<=rear;i++)
      {
            cout<<a[i]<<" "; }
}


void dequeue::delete_fr_front()
{
      if(front==-1)
      {
            cout<<"Deletion is not possible:: Dequeue is empty";
            return;
      }
      else
      {
            if(front==rear)
            {
                  front=rear=-1;
                  return;
            }
            cout<<"The deleted element is "<<a[front];
            front=front+1;
      }
```

```
}

void dequeue::delete_fr_rear()
{
      if(front==-1)
      {
            cout<<"Deletion is not possible:Dequeue is empty";
            return;
      }
      else
      {
            if(front==rear)
            {
                  front=rear=-1;
            }
            cout<<"The deleted element is "<< a[rear];
            rear=rear-1;
      }


}



int main()
{
      int c,item;
      dequeue d1;

      do
      {
            cout<<"\n\n****DEQUEUE OPERATION****\n";
            cout<<"\n1-Insert at beginning";
            cout<<"\n2-Insert at end";
            cout<<"\n3_Display";
            cout<<"\n4_Deletion from front";
            cout<<"\n5-Deletion from rear";
            cout<<"\n6_Exit";
            cout<<"\nEnter your choice<1-4>:";
            cin>>c;

            switch(c)
            {
            case 1:
                  cout<<"Enter the element to be inserted:";
                  cin>>item;
                  d1.add_at_beg(item);
                  break;

            case 2:
                  cout<<"Enter the element to be inserted:";
                  cin>>item;
                  d1.add_at_end(item);
                  break;

            case 3:
                  d1.display();
                  break;

            case 4:
                  d1.delete_fr_front();
                  break;
            case 5:
                  d1.delete_fr_rear();
                  break;

            case 6:
                  exit(1);
                  break;

            default:
                  cout<<"Invalid choice";
                  break;
            }

      }while(c!=7);
      return 0;

 }
```