**D-LINKED LIST 1**

**CREATE+DISPLAY+LENGTH+REVERSE+SORT+INSERT+DELETE+SEARCH+COPY**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node \*prev;

};

class list:public Node

{

Node \*dlist,\*temp;

public:

list()

{

dlist=NULL;

temp=NULL;

}

void create();

void display();

void insert\_start(int n);

void insert\_last(int n);

void insert\_between(int n);

void delete\_start();

void delete\_last();

void delete\_between();

void length();

void reverse();

int search(int n);

void sort();

void copyall(int n);

};

int main()

{

list l;

int c,k,i,n,r;

while(1)

{

cout<<"\n1.Create\n2.Insert at first\n3.Insert at last\n4.Insert inbetween\n5.Delete at start\n6.Delete at last\n7.Delete inbetween\n8.Reverse\n9.Length of linked list\n10.Search\n11.Sort\n12.Reverse\n13.Copy all\n14.Display\nExit\n";

cout<<"Enter your choice:\n";

cin>>c;

switch(c)

{

case 1:cout<<"enter number of nodes:";

cin>>k;

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

{

l.create();

}

break;

case 2:cout<<"Enter value to insert at the start of list:";

cin>>n;

l.insert\_start(n);

break;

case 3:cout<<"Enter value to insert at the end of list:";

cin>>n;

l.insert\_last(n);

break;

case 4:cout<<"Enter value to insert in middle of list:";

cin>>r;

l.insert\_between(r);

break;

case 5:l.delete\_start();

break;

case 6:l.delete\_last();

break;

case 7:l.delete\_between();

break;

case 8:cout<<"reverse";

break;

case 9:l.length();

break;

case 10:cout<<"Enter element to search:";

cin>>n;

i=l.search(n);

cout<<"index="<<i;

break;

case 11:l.sort();

break;

case 12:l.reverse();

break;

case 13:l.copyall(k);

break;

case 14:l.display();

break;

case 15:return 0;

}

}

return 0;

}

void list::create()

{

Node \*newnode=new Node;

int n;

cout<<"Enter an element:";

cin>>n;

newnode->data=n;

newnode->next=NULL;

newnode->prev=NULL;

if(dlist==NULL)

{

dlist=temp=newnode;

}

else

{

temp->next=newnode;

newnode->prev=temp;

temp=temp->next;

}

}

void list::display()

{

Node \*newnode=dlist;

if(newnode==NULL)

{

cout<<"List is empty";

}

while(newnode!=NULL)

{

cout<<newnode->data;

cout<<"-->";

newnode=newnode->next;

}

cout<<"NULL\n";

}

void list::insert\_start(int n)

{

Node \*newnode=new Node;

newnode->data=n;

newnode->next=dlist;

dlist=newnode;

}

void list::insert\_last(int n)

{

Node \*newnode=new Node;

Node \*temp;

temp=dlist;

newnode->data=n;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

newnode->prev=temp;

}

void list::insert\_between(int n)

{

Node \*newnode=new Node;

newnode->data=n;

Node \*temp=dlist;

newnode->next=temp->next;

temp->next->prev=newnode;

newnode->prev=temp->next;

temp->next=newnode;

}

void list::delete\_start()

{

Node \*temp;

temp=dlist;

dlist=dlist->next;

free(temp);

temp=temp->next;

}

void list::delete\_last()

{

Node \*temp2;

temp2=dlist;

while(temp2->next!=NULL)

{

temp2=temp2->next;

}

temp2->next=NULL;

temp2->prev=NULL;

free(temp2);

}

void list::delete\_between()

{

Node \*temp;

temp=dlist;

temp=temp->next;

temp->prev->next=temp->next;

temp->next->prev=temp->prev;

free(temp);

temp=temp->next;

}

void list::length()

{

int c = 0;

Node \*p;

p=dlist;

if(dlist==NULL)

{

cout<<"Size="<<c;

}

while(p!=NULL)

{

p=p->next;

c++;

}

cout<<"Size="<<c;

}

int list::search(int n)

{

int index = 0;

Node \*current = dlist;

do

{

if (current->data == n)

return index;

current = current->next;

index++;

} while (current != dlist);

return -1;

}

void list::reverse()

{

Node \*forw,\*back,\*curr;

forw=dlist;

curr=NULL;

back=NULL;

while(forw!=NULL)

{

back=curr;

curr=forw;

forw=forw->next;

curr->next=back;

}

dlist=curr;

}

void list::sort()

{

Node \*temp,\*temp1;

temp=dlist;

temp1=NULL;

int t;

while(temp!=NULL)

{

temp1=temp->next;

while(temp1!=NULL)

{

if(temp->data>temp1->data)

{

t=temp->data;

temp->data=temp1->data;

temp1->data=t;

}

temp1=temp1->next;

}

cout<<temp->data<<"-->";

temp=temp->next;

}

cout<<"NULL";

}

void list::copyall(int n)

{

temp=dlist;

Node \*q;

q=dlist;

int i;

Node \*t;

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

{

Node \*newnode=new Node;

newnode->data=temp->data;

newnode->next=temp->next;

if(q==NULL)

{

q=newnode;

}

temp=temp->next;

}

//display;

t=q;

while(t->next!=NULL)

{

cout<<t->data<<"-->";

t=t->next;

}

cout<<t->data;

cout<<"-->";

cout<<"NULL";

}

**OUTPUT**

**D-LINKED LIST 2**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node \*prev;

};

class list:public Node

{

Node \*dlist1,\*temp1,\*dlist2,\*temp2;

public:

list()

{

dlist1=NULL;

temp1=NULL;

dlist2=NULL;

temp2=NULL;

}

void createA();

void displayA();

void createB();

void displayB();

void concate();

void merge();

};

int main()

{

list l;

int c,k,i,n,r;

while(1)

{

cout<<"\n1.Create linked list A\n2.Display linked list A\n3.Create linked list B\n4.Display linked list B\n5.Concate both linked lists\n6.Merge\nExit\n";

cout<<"Enter your choice:\n";

cin>>c;

switch(c)

{

case 1:cout<<"enter number of nodes:";

cin>>k;

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

{

l.createA();

}

break;

case 2:l.displayA();

break;

case 3:cout<<"enter number of nodes:";

cin>>k;

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

{

l.createB();

}

break;

case 4:l.displayB();

break;

case 5:l.concate();

break;

case 6:l.merge();

break;

case 7:return 0;

}

}

return 0;

}

void list::createA()

{

Node \*newnode=new Node;

int n;

cout<<"Enter an element:";

cin>>n;

newnode->data=n;

newnode->next=NULL;

newnode->prev=NULL;

if(dlist1==NULL)

{

dlist1=temp1=newnode;

}

else

{

temp1->next=newnode;

newnode->prev=temp1;

temp1=temp1->next;

}

}

void list::displayA()

{

Node \*newnode=dlist1;

if(newnode==NULL)

{

cout<<"List is empty";

}

while(newnode!=NULL)

{

cout<<newnode->data;

cout<<"-->";

newnode=newnode->next;

}

cout<<"NULL\n";

}

void list::createB()

{

Node \*newnode=new Node;

int n;

cout<<"Enter an element:";

cin>>n;

newnode->data=n;

newnode->next=NULL;

newnode->prev=NULL;

if(dlist2==NULL)

{

dlist2=temp2=newnode;

}

else

{

temp2->next=newnode;

newnode->prev=temp2;

temp2=temp2->next;

}

}

void list::displayB()

{

Node \*newnode=dlist2;

if(newnode==NULL)

{

cout<<"List is empty";

}

while(newnode!=NULL)

{

cout<<newnode->data;

cout<<"-->";

newnode=newnode->next;

}

cout<<"NULL\n";

}

void list::concate()

{

Node \*head1,\*head2,\*p;

head1=dlist1;

head2=dlist2;

p=head1;

while(p->next!=NULL)

{

p=p->next;

}

p->next=head2;

cout<<"concatenate linked list:\n";

Node \*newnode=dlist1;

if(newnode==NULL)

{

cout<<"List is empty";

}

while(newnode!=NULL)

{

cout<<newnode->data;

cout<<"-->";

newnode=newnode->next;

}

cout<<"NULL\n";

}

void list::merge()

{

Node \*t1,\*t2,\*q1,\*q2,\*r1,\*r2;

t1=dlist1;

t2=dlist2;

q1=t1->next;

q2=t2->next;

r1=q1->next;

r2=q2->next;

t1->next=t2;

t2->prev=t1;

t2->next=q1;

q1->prev=t2;

q1->next=q2;

q2->prev=q1;

q2->next=r1;

r1->prev=q2;

r1->next=r2;

r2->prev=r1;

cout<<"merged double linked list:";

Node \*newnode=dlist1;

if(newnode==NULL)

{

cout<<"List is empty";

}

while(newnode!=NULL)

{

cout<<newnode->data;

cout<<"-->";

newnode=newnode->next;

}

cout<<"NULL\n";

}