**INPUT**

**SINGLY LINKED LIST**

#include<iostream>

using namespace std;

class Node {

public:

int data;

Node \* next;

Node(int x) {

data = x;

next = NULL; } };

class SinglyLinkedList {

Node \* first, \* last;

int count;

public:

SinglyLinkedList() {

first = last = NULL;

count = 0; }

void insertFirst(int x);

void insertLast(int x);

void deleteFirst();

void deleteLast();

void deleteAll();

void display();

void ascendingSort();

void descendingSort(); };

void SinglyLinkedList::insertFirst(int x) {

Node \* newrec = new Node(x);

if (first == NULL) {

first = last = newrec;

} else {

newrec ->next = first;

first = newrec; }

count++;

cout << "\nNew node inserted at first"; }

void SinglyLinkedList::insertLast(int x) {

Node \* newrec = new Node(x);

if (first == NULL) {

first = last = newrec;

} else {

last ->next = newrec;

last = newrec; }

count++;

cout << "\nNew node inserted at last"; }

void SinglyLinkedList::deleteFirst() {

if (first == NULL) {

cout << "\nLinkedList underflow";

} else {

Node \* delrec = first;

if (first == last) {

first = last = NULL;

} else {

first = first->next; }

delete delrec;

count--;

cout << "\nfirst node deleted"; } }

void SinglyLinkedList::deleteLast() {

if (first == NULL) {

cout << "\nLinkedList underflow";

} else {

Node \* delrec = last;

if (first == last) {

first = last = NULL;

} else {

Node \* i = first;

while (i ->next != last) {

i = i ->next; }

last = i;

last ->next = NULL; }

delete delrec;

count--;

cout << "\nLast node deleted"; } }

void SinglyLinkedList::deleteAll() {

while (first != NULL) {

Node \* delrec = first;

if (first == last) {

first = last = NULL;

} else {

first = first -> next; }

delete delrec; }

count = 0;

cout << "\nAll nodes deleted"; }

void SinglyLinkedList::display() {

if (first == NULL) {

cout << "\nLinkedList is empty";

} else {

cout << "\nSinglyLinkedList elements are : ";

for (Node \* i = first; i != NULL; i = i -> next) {

cout << "\n" << i -> data; } } }

void SinglyLinkedList::ascendingSort() {

for (int step = 1; step < count; step++) {

Node \* i = first;

for (int j = 0; j < count - step; j++) {

if (i -> data > i -> next -> data) {

int temp = i -> data;

i -> data = i -> next -> data;

i -> next -> data = temp; }

i = i -> next; } }

display(); }

void SinglyLinkedList::descendingSort() {

for (int step = 1; step < count; step++) {

Node \* i = first;

for (int j = 0; j < count - step; j++) {

if (i -> data < i -> next -> data) {

int temp = i -> data;

i -> data = i -> next -> data;

i -> next -> data = temp; }

i = i -> next; } }

display(); }

int main() {

SinglyLinkedList list;

int choice;

do {

cout << "\n\n\n\*\*\*\*\*\*\*\*\*Menu\*\*\*\*\*\*\*\*\*\*";

cout << "\n1. Insert First";

cout << "\n2. Insert Last";

cout << "\n3. Delete First";

cout << "\n4. Delete Last";

cout << "\n5. Delete All";

cout << "\n6. Ascending Sort";

cout << "\n7. Descending Sort";

cout << "\n8. Display";

cout << "\n9. Exit";

cout << "\nEnter your choice => ";

cin >> choice;

int x;

switch (choice) {

case 1:

cout << "\nEnter data : ";

cin >> x;

list.insertFirst(x);

break;

case 2:

cout << "\nEnter data : ";

cin >> x;

list.insertLast(x);

break;

case 3:

list.deleteFirst();

break;

case 4:

list.deleteLast();

break;

case 5:

list.deleteAll();

break;

case 6:

list.ascendingSort();

break;

case 7:

list.descendingSort();

break;

case 8:

list.display();

break;

case 9:

break;

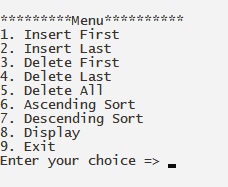
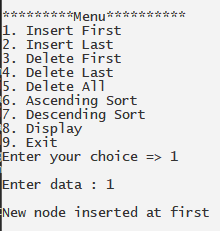
default:

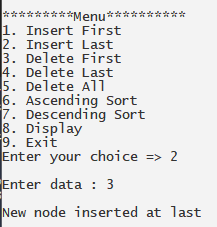
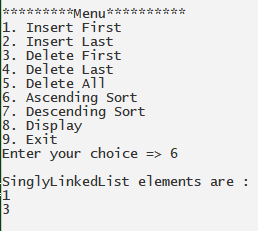
cout << "\nPlease enter proper choice"; } }

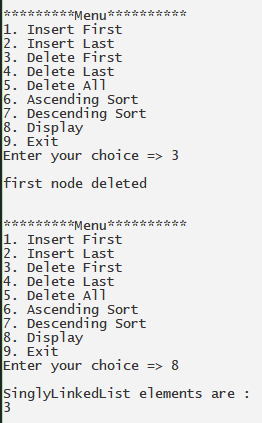
while (choice != 9);

return 0; }

**OUTPUT**



**INTPUT**

**DOUBLY LINKED LIST**

#include<iostream>

using namespace std;

class Node {

public:

int data;

Node \* next, \* prev;

Node(int x) {

data = x;

next = prev = NULL; } };

class DoublyLinkedList {

Node \* first, \* last;

int count;

public:

DoublyLinkedList() {

first = last = NULL;

count = 0; }

void insertFirst(int x);

void insertLast(int x);

void deleteFirst();

void deleteLast();

void deleteAll();

void display();

void displayReverse();

void ascendingSort();

void descendingSort(); };

void DoublyLinkedList::insertFirst(int x) {

Node \* newrec = new Node(x);

if (first == NULL) {

first = last = newrec;

} else {

newrec -> next = first;

first -> prev = newrec;

first = newrec; }

count++;

cout << "\nNew node inserted at first"; }

void DoublyLinkedList::insertLast(int x) {

Node \* newrec = new Node(x);

if (first == NULL) {

first = last = newrec;

} else {

last -> next = newrec;

newrec -> prev = last;

last = newrec; }

count++;

cout << "\nNew node inserted at last"; }

void DoublyLinkedList::deleteFirst() {

if (first == NULL) {

cout << "\nLinkedList underflow";

} else {

Node \* delrec = first;

if (first == last) {

first = last = NULL;

} else {

first = first -> next;

first -> prev = NULL; }

delete delrec;

count--;

cout << "\nfirst node deleted"; } }

void DoublyLinkedList::deleteLast() {

if (first == NULL) {

cout << "\nLinkedList underflow";

} else {

Node \* delrec = last;

if (first == last) {

first = last = NULL;

} else {

last = last -> prev;

last -> next = NULL; }

delete delrec;

count--;

cout << "\nLast node deleted"; } }

void DoublyLinkedList::deleteAll() {

while (first != NULL) {

Node \* delrec = first;

if (first == last) {

first = last = NULL;

} else {

first = first -> next;

first -> prev = NULL; }

delete delrec; }

count = 0;

cout << "\nAll nodes deleted"; }

void DoublyLinkedList::display() {

if (first == NULL) {

cout << "\nLinkedList is empty";

} else {

cout << "\nDoublyLinkedList elements are : ";

for (Node \* i = first; i != NULL; i = i -> next) {

cout << "\n" << i -> data; } } }

void DoublyLinkedList::displayReverse() {

if (first == NULL) {

cout << "\nLinkedList is empty";

} else {

cout << "\nDoublyLinkedList elements are : ";

for (Node \* i = last; i != NULL; i = i -> prev) {

cout << "\n" << i -> data; } } }

void DoublyLinkedList::ascendingSort() {

for (int step = 1; step < count; step++) {

Node \* i = first;

for (int j = 0; j < count - step; j++) {

if (i -> data > i -> next -> data) {

int temp = i -> data;

i -> data = i -> next -> data;

i -> next -> data = temp; }

i = i -> next; } }

display(); }

void DoublyLinkedList::descendingSort() {

for (int step = 1; step < count; step++) {

Node \* i = first;

for (int j = 0; j < count - step; j++) {

if (i -> data < i -> next -> data) {

int temp = i -> data;

i -> data = i -> next -> data;

i -> next -> data = temp; }

i = i -> next; } }

display(); }

int main() {

DoublyLinkedList list;

int choice;

do {

cout << "\n\n\n\*\*\*\*\*\*\*\*\*\*\*Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

cout << "\n1. Insert First";

cout << "\n2. Insert Last";

cout << "\n3. Delete First";

cout << "\n4. Delete Last";

cout << "\n5. Delete All";

cout << "\n6. Ascending Sort";

cout << "\n7. Descending Sort";

cout << "\n8. Display";

cout << "\n9. Display Reverse";

cout << "\n10. Exit";

cout << "\nEnter your choice => ";

cin >> choice;

int x;

switch (choice) {

case 1:

cout << "\nEnter data : ";

cin >> x;

list.insertFirst(x);

break;

case 2:

cout << "\nEnter data : ";

cin >> x;

list.insertLast(x);

break;

case 3:

list.deleteFirst();

break;

case 4:

list.deleteLast();

break;

case 5:

list.deleteAll();

break;

case 6:

list.ascendingSort();

break;

case 7:

list.descendingSort();

break;

case 8:

list.display();

break;

case 9:

list.displayReverse();

break;

case 10:

break;

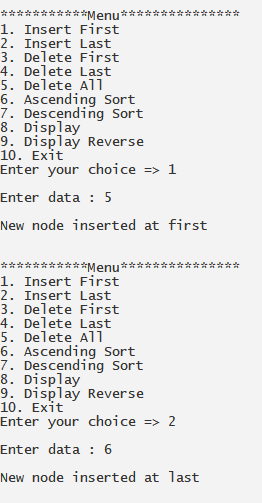
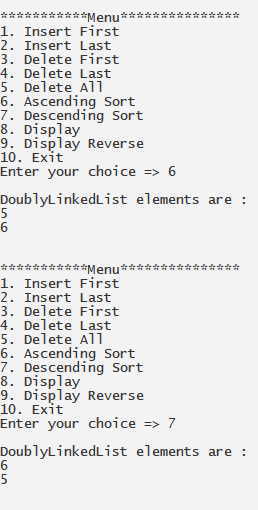
default:

cout << "\nPlease enter proper choice"; }

} while (choice != 10);

return 0; }

**OUTPUT**

**INTPUT**

**CIRCULAR LINKED LIST**

#include<iostream>

#define SIZE 100

using namespace std;

class node

{

public:

node()

{

next = NULL; }

int data;

node \*next;

}\*front=NULL,\*rear=NULL,\*n,\*temp,\*temp1;

class CircularLL

{

public:

void insertion();

void deletion();

void display(); };

void CircularLL::insertion()

{

n=new node[sizeof(node)];

cout<<"\nEnter the Element: ";

cin>>n->data;

if(front==NULL)

{

front=n;

}

else

{

rear->next=n;

}

rear=n;

rear->next=front; }

void CircularLL::deletion()

{

int x;

temp=front;

if(front==NULL)

{

cout<<"\nCircular List Empty!!!";

}

else

{

if(front==rear)

{

x=front->data;

delete(temp);

front=NULL;

rear=NULL;

}

else

{

x=temp->data;

front=front->next;

rear->next=front;

delete(temp);

}

cout<<"\nElement "<<x<<" is Deleted";

display(); } }

void CircularLL::display()

{

temp=front;

temp1=NULL;

if(front==NULL)

{

cout<<"\n\nCircular List Empty!!!"; }

else

{

cout<<"\n\nCircular List Elements are:\n\n";

while(temp!=temp1)

{

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

temp=temp->next;

temp1=front; } } }

int main()

{

CircularLL cqobj;

int ch;

do

{

cout<<"\n\n\tMain Menu Circular Link-List";

cout<<"\n##########################";

cout<<"\n1. Insert\n2. Delete\n3. Display\n4. Exit\n\nEnter Your Choice: ";

cin>>ch;

switch(ch)

{

case 1:

cqobj.insertion();

cqobj.display();

break;

case 2:

cqobj.deletion();

break;

case 3:

cqobj.display();

break;

case 4:

break;

default:

cout<<"\n\nWrong Choice!!! Try Again.";

}

}while(ch!=4);

return 0;

}

**OUTPUT**

