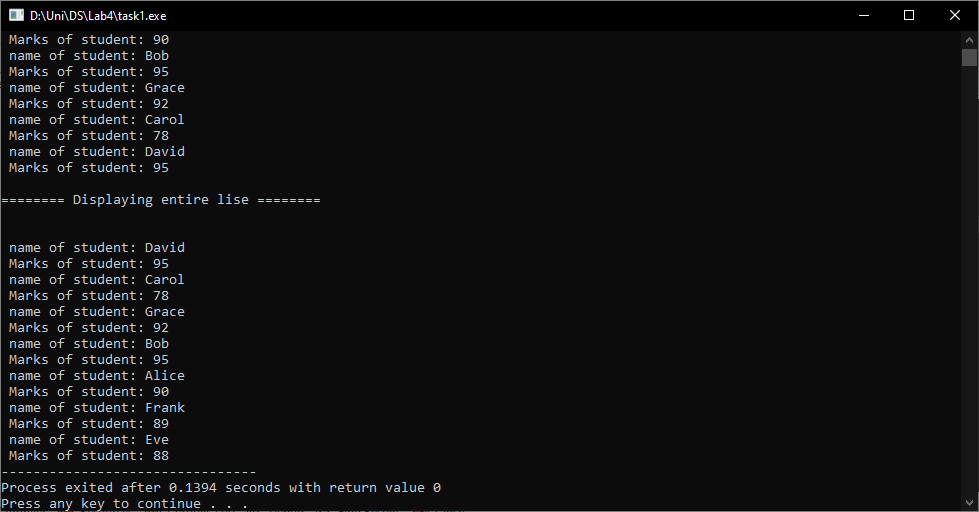
DS LAB 4

K226007

Syed Yousha

BSR-3C

Task1:



#include <iostream>

using namespace std;

class Node

{

public:

int marks;

string name;

Node \*next;

Node(int val, string name)

{

marks = val;

this->name = name;

next = NULL;

}

};

class Singli

{

public:

Node \*head;

Node \*tail;

Singli()

{

head = NULL;

tail = NULL;

}

void insert\_head(int marks, string name)

{

Node \*n = new Node(marks, name);

n->next = head;

head = n;

cout<<"\nInserted at head";

}

void insert\_tail(int marks, string name)

{

Node \*n = new Node(marks, name);

if(head == NULL)

{

head = n;

}

else

{

Node \* temp = head;

while( temp->next != NULL )

{

temp = temp->next;

}

temp->next = n;

n->next = tail;

cout<<"\nInserted at tail";

}

}

void insert\_after(string name\_pos, int marks, string name)

{

Node \*n = new Node(marks, name);

Node \*temp = head;

Node \*curr = head;

Node \*pre;

while(temp != NULL)

{

pre = curr;

curr = curr->next;

if(temp->name == name\_pos)

{

break;

}

temp = temp->next;

}

pre->next = n;

n->next = curr;

}

bool search\_marks(string key)

{

Node \*temp = head;

while( temp != NULL )

{

if( temp->name == key )

{

cout<<"\nMarks are: "<<temp->marks;

return true;

}

temp = temp->next;

}

cout<<"\n Name not found!";

return false;

}

void display()

{

Node \*temp = head;

cout<<"\n\n======== Displaying entire lise ========\n\n";

while(temp != NULL)

{

cout<<"\n name of student: "<<temp->name;

cout<<"\n Marks of student: "<<temp->marks;

temp = temp->next;

}

}

void reverse()

{

Node \*curr = head;

Node \*prev = NULL;

Node \*nex = NULL;

while(curr != NULL)

{

nex = curr->next;

curr->next = prev;

prev = curr;

curr = nex;

}

head = prev;

}

};

int main()

{

Singli obj;

obj.insert\_head(88, "Eve");

obj.insert\_tail(89, "Frank");

obj.insert\_tail(90, "Alice");

obj.insert\_tail(95, "Bob");

obj.insert\_tail(78, "Carol");

obj.insert\_tail(95, "David");

obj.insert\_after("Bob", 92, "Grace");

obj.search\_marks("Carol");

obj.display();

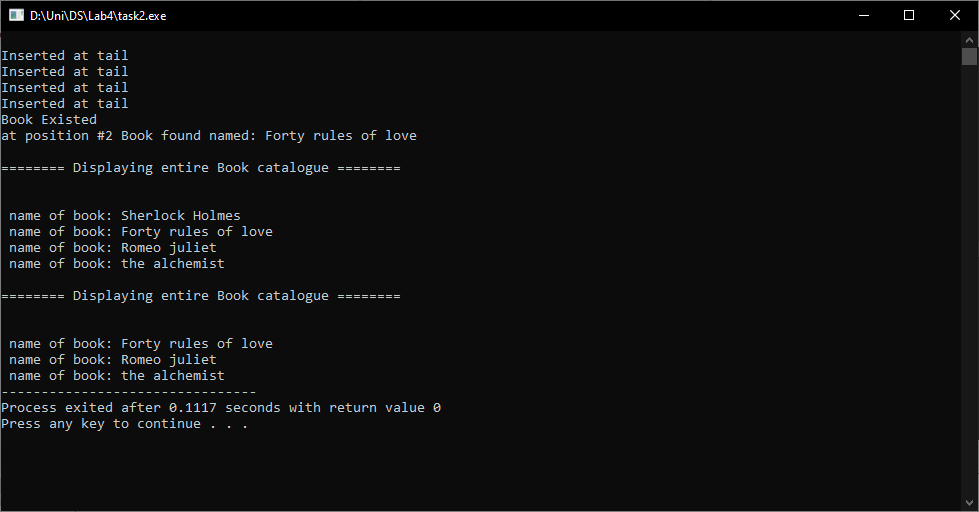
obj.reverse();

obj.display();

return 0;

}

Task2:



#include <iostream>

using namespace std;

class Node

{

public:

string book;

Node \*next;

Node(string book) : book(book), next(NULL) {}

};

class Library

{

public:

Node \*head;

Node \*tail;

Library()

{

head = NULL;

tail = NULL;

}

void insert\_tail(string book)

{

Node \*n = new Node(book);

if (head == NULL)

{

head = n;

tail = n;

}

else

{

tail->next = n;

tail = n;

}

cout << "\nInserted at tail";

}

void del\_front()

{

Node \*d = head;

head = head->next;

d->next = NULL;

delete d;

}

bool search\_book(string key)

{

Node \*temp = head;

while (temp != NULL)

{

if (temp->book == key)

{

cout << "\nBook Existed";

return true;

}

temp = temp->next;

}

cout << "\n Name not found!";

return false;

}

bool search\_book(int key)

{

Node \*temp = head;

int i = 0;

while (temp != NULL)

{

if (i == key)

{

cout << "\nat position #" << i + 1 << " Book found named: " << temp->book;

return true;

}

temp = temp->next;

i++;

}

cout << "\nBook not found at position #" << key + 1;

return false;

}

void display()

{

Node \*temp;

temp = head;

cout << "\n\n======== Displaying entire Book catalogue ========\n\n";

while (temp != NULL)

{

cout << "\n name of book: " << temp->book;

temp = temp->next;

}

}

};

int main()

{

Library obj;

//Adding book

obj.insert\_tail("Sherlock Holmes");

obj.insert\_tail("Forty rules of love");

obj.insert\_tail("Romeo juliet");

obj.insert\_tail("the alchemist");

//Searching

obj.search\_book("Forty rules of love");

obj.search\_book(1);

obj.display();

//Deleting a book

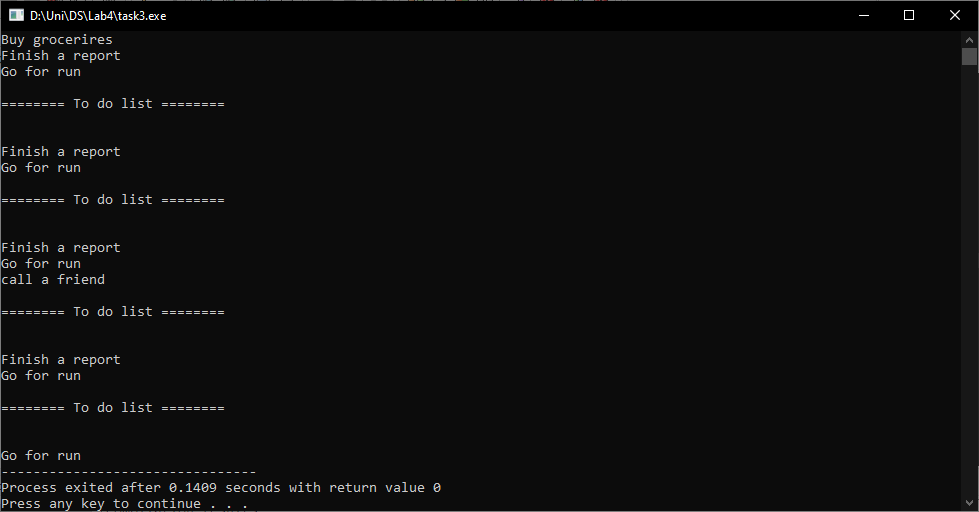
obj.del\_front();

obj.display();

return 0;

}

Task3:



#include <iostream>

using namespace std;

class Node

{

public:

string task;

Node \*next;

Node(string t) : task(t), next(NULL) {}

};

class To\_do\_list

{

public:

Node \*head;

Node \*tail;

To\_do\_list() : head(NULL), tail(NULL) {}

void insert\_tail(string task)

{

Node \*n = new Node(task);

if (head == NULL)

{

head = n;

tail = n;

}

else

{

tail->next = n;

tail = n;

}

// cout << "\nInserted at tail";

}

void del\_front()

{

Node \*d = head;

head = head->next;

d->next = NULL;

delete d;

}

void delete\_end()

{

if (head == NULL)

{

cout << "List is empty. Cannot delete from an empty list.";

return;

}

Node \*temp = head;

Node \*pre = NULL;

while (temp->next != NULL)

{

pre = temp;

temp = temp->next;

}

if (temp == head)

{

delete temp;

head = NULL;

tail = NULL;

}

else

{

tail = pre;

tail->next = NULL;

delete temp;

}

}

void delete\_after(int key)

{

if (head == NULL || key < 0)

{

cout << "Invalid position or list is empty. Cannot delete.";

return;

}

Node \*curr = head;

Node \*pre = NULL;

for (int i = 1; i <= key && curr != NULL; i++)

{

if (i == key)

{

if (pre == NULL)

{

head = curr->next;

delete curr;

return;

}

else

{

pre->next = curr->next;

delete curr;

return;

}

}

pre = curr;

curr = curr->next;

}

cout << "Position " << key << " not found in the list. Cannot delete.";

}

void display()

{

Node \*temp;

temp = head;

cout << "\n\n======== To do list ========\n\n";

while (temp != NULL)

{

cout << endl

<< temp->task;

temp = temp->next;

}

}

};

int main()

{

To\_do\_list obj;

// insertion

obj.insert\_tail("Buy grocerires");

obj.insert\_tail("Finish a report");

obj.insert\_tail("Go for run");

obj.display();

// Deletion

obj.del\_front();

obj.display();

obj.insert\_tail("call a friend");

obj.display();

obj.delete\_end();

obj.display();

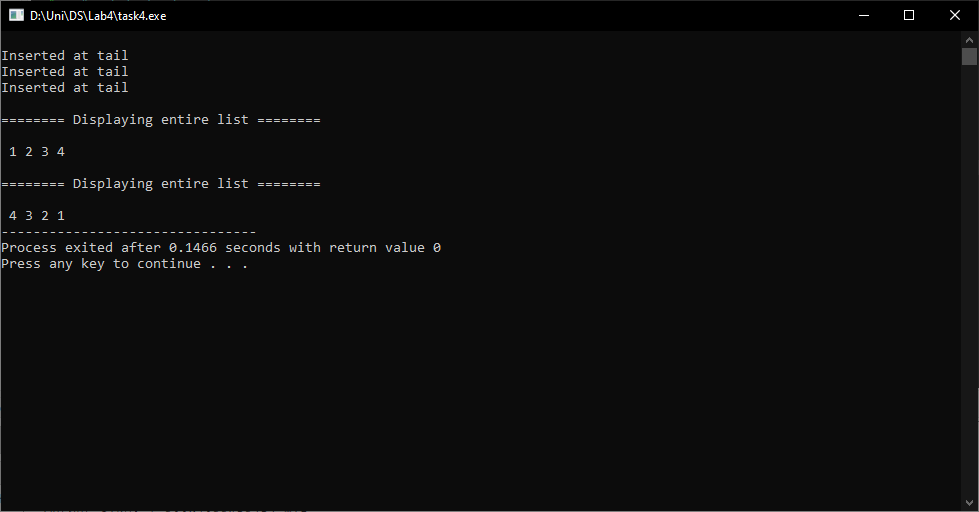
obj.delete\_after(1);

obj.display();

return 0;

}

Task4:



#include <iostream>

using namespace std;

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int val)

{

data = val;

next = NULL;

}

};

class Singli

{

public:

Node \*head;

Node \*tail;

Singli()

{

head = NULL;

tail = NULL;

}

void insert\_head(int data)

{

Node \*n = new Node(data);

n->next = head;

head = n;

cout<<"\nInserted at head";

}

void insert\_tail(int data)

{

Node \*n = new Node(data);

if(head == NULL)

{

head = n;

}

else

{

Node \* temp = head;

while( temp->next != NULL )

{

temp = temp->next;

}

temp->next = n;

n->next = tail;

cout<<"\nInserted at tail";

}

}

void display()

{

Node \*temp = head;

cout<<"\n\n======== Displaying entire list ========\n\n";

while(temp != NULL)

{

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

temp = temp->next;

}

}

void reverse()

{

Node \*curr = head;

Node \*prev = NULL;

Node \*nex = NULL;

while(curr != NULL)

{

nex = curr->next;

curr->next = prev;

prev = curr;

curr = nex;

}

head = prev;

}

};

int main()

{

Singli obj;

obj.insert\_tail(1);

obj.insert\_tail(2);

obj.insert\_tail(3);

obj.insert\_tail(4);

obj.display();

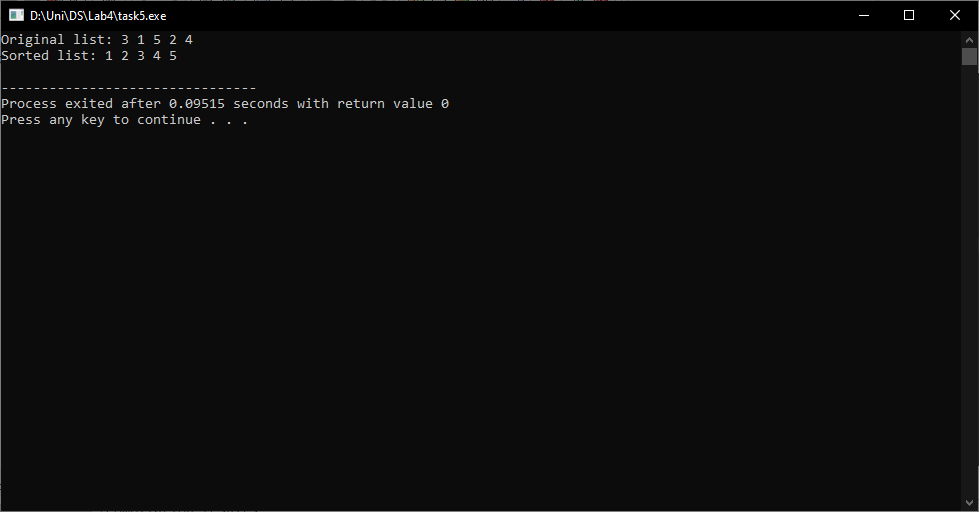
obj.reverse();

obj.display();

return 0;

}

Task5:



#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int val) : data(val), next(NULL) {}

};

class LinkedList

{

private:

Node \*head;

Node \*tail;

public:

LinkedList() : head(NULL), tail(NULL) {}

void insert(int val)

{

Node \*newNode = new Node(val);

if (head == NULL)

{

head = newNode;

tail = newNode;

}

else

{

tail->next = newNode;

tail = newNode;

}

}

void display()

{

Node \*temp = head;

while (temp != NULL)

{

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

temp = temp->next;

}

cout << endl;

}

void sort()

{

if (head == NULL)

return;

Node \*current;

Node \*nextNode = NULL;

bool sorted = false;

while (!sorted)

{

sorted = true;

current = head;

while (current->next != nextNode)

{

if (current->data > current->next->data)

{

swap(current->data, current->next->data);

sorted = false;

}

current = current->next;

}

nextNode = current;

}

}

};

int main()

{

LinkedList list;

list.insert(3);

list.insert(1);

list.insert(5);

list.insert(2);

list.insert(4);

cout << "Original list: ";

list.display();

list.sort();

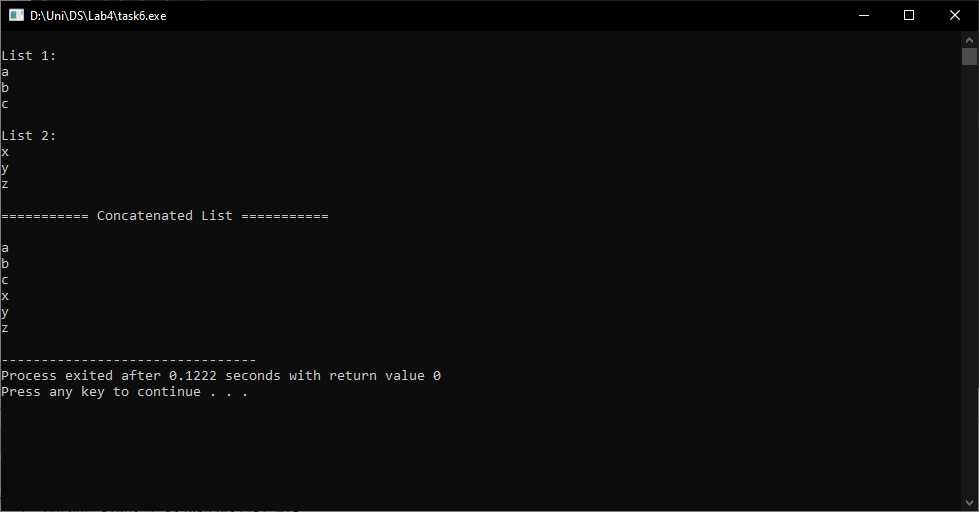
cout << "Sorted list: ";

list.display();

return 0;

}

Task6:



#include <iostream>

using namespace std;

class Node

{

public:

string str;

Node \*next;

Node(string s) : str(s), next(NULL) {}

};

class List

{

public:

Node \*head;

Node \*tail;

List() : head(NULL), tail(NULL) {}

void insert\_tail(string item)

{

Node \*n = new Node(item);

if (head == NULL)

{

head = n;

tail = n;

}

else

{

tail->next = n;

tail = n;

}

}

void display()

{

Node \*temp = head;

while (temp != NULL)

{

cout << temp->str << endl;

temp = temp->next;

}

}

};

void concatenate\_list(List &list1, List &list2)

{

if (list1.head == NULL)

{

list1.head = list2.head;

list1.tail = list2.tail;

}

else if (list2.head != NULL)

{

list1.tail->next = list2.head;

list1.tail = list2.tail;

}

}

int main()

{

List list1;

list1.insert\_tail("a");

list1.insert\_tail("b");

list1.insert\_tail("c");

List list2;

list2.insert\_tail("x");

list2.insert\_tail("y");

list2.insert\_tail("z");

cout << "\nList 1: \n";

list1.display();

cout << "\nList 2: \n";

list2.display();

concatenate\_list(list1, list2);

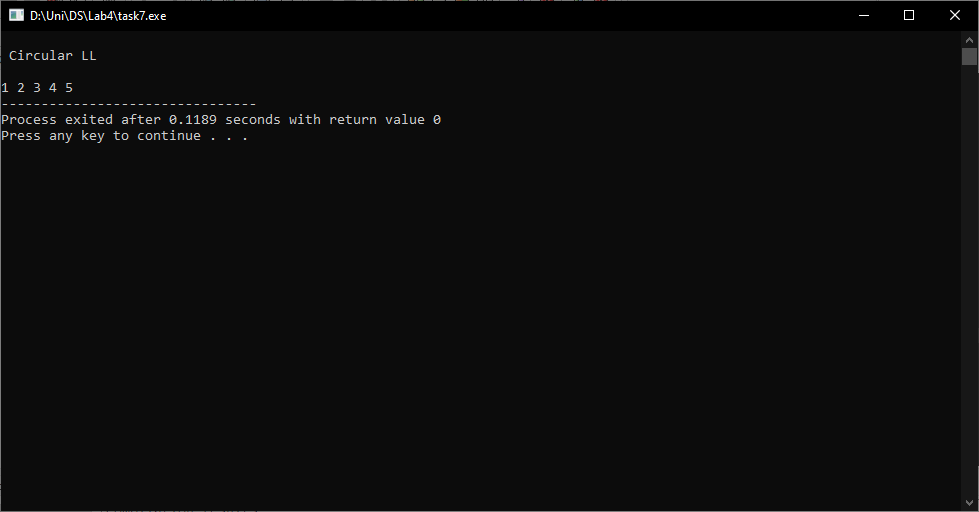
cout << "\n=========== Concatenated List ===========\n" << endl;

list1.display();

return 0;

}

Task 7:



#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int val)

{

data = val;

next = NULL;

}

};

class Circular

{

public:

Node \*head;

Node \*tail;

Circular()

{

head = NULL;

tail = NULL;

}

void insert\_end(int val)

{

Node \*n = new Node(val);

if(head == NULL)

{

head = n;

tail = n;

tail->next=head;

}

tail->next=n;

tail=tail->next;

tail->next=head;

}

void display()

{

cout<<"\n Circular LL\n\n";

Node \*tmp;

tmp = tail->next;

cout<<tmp->data<<" ";

tmp = tmp->next;

while(tmp != tail->next)

{

cout<< tmp->data<<" ";

tmp = tmp->next;

}

}

};

int main()

{

Circular obj;

obj.insert\_end(1);

obj.insert\_end(2);

obj.insert\_end(3);

obj.insert\_end(4);

obj.insert\_end(5);

obj.display();

return 0;

}