|  |  |
| --- | --- |
| download | COMSATS University Islamabad, Vehari Campus Department of Computer Science |

**Class: BCS-SP22 Submission Deadline: 9 Oct 2023**

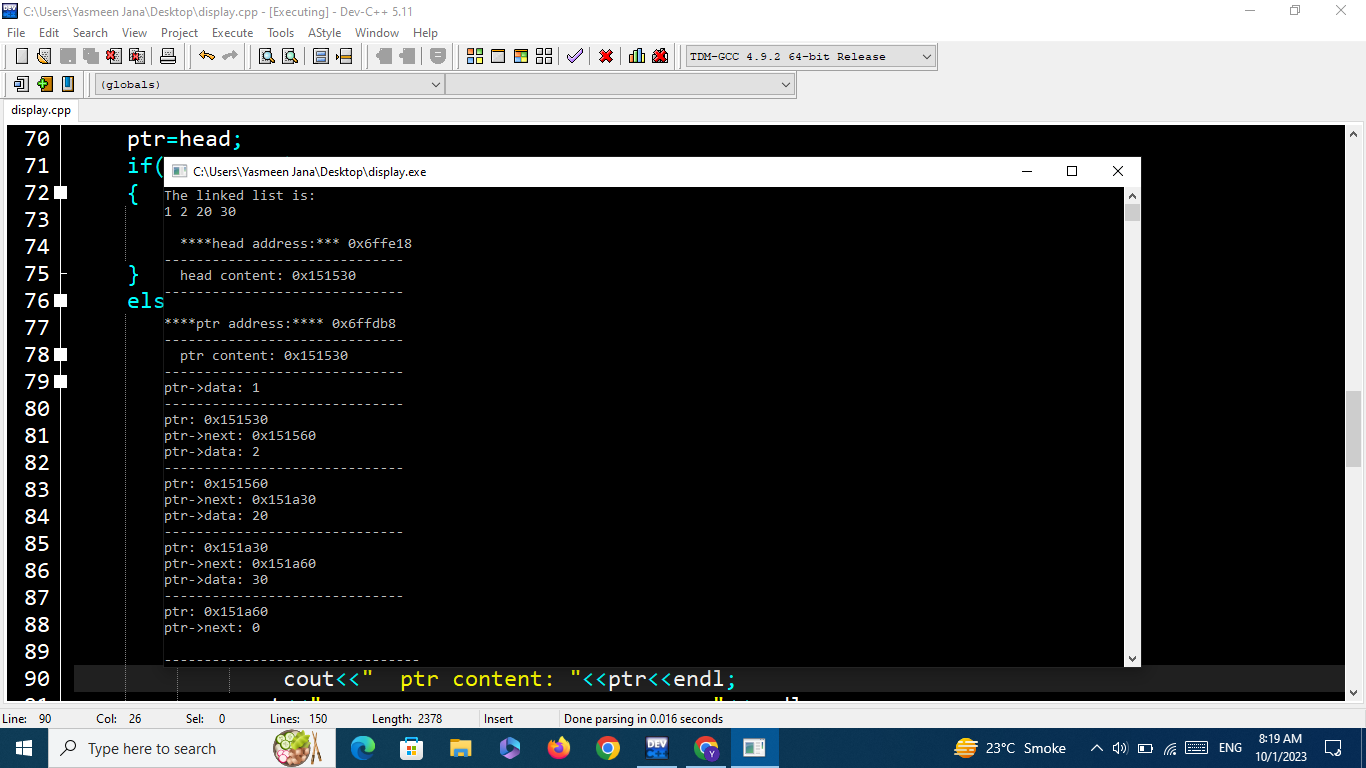
**Subject: Data Structures and Algorithms-Lab Instructor: Yasmeen Jana Max Marks: 20 Reg. No:**

**Email:** [**yasmeenjana@cuivehari.edu.pk**](mailto:yasmeenjana@cuivehari.edu.pk)

**You can ask queries related to Lab Activities on the above email.**

**Activity 1:**

Create a function to display linked list output as below:



#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

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

};

void displayLinkedList(Node\* head) {

cout << "The linked list is:" << endl;

Node\* ptr = head;

while (ptr != NULL) {

cout << ptr->data << " ";

ptr = ptr->next;

}

cout << endl<<endl;

cout << "\*\*\*\*head address:\*\*\*";

cout << &head << endl;

cout<<"-----------------------------\n";

cout << "head content: " << head->data << endl;

cout <<"-----------------------------\n";

ptr = head;

cout << "\*\*\*ptr address\*\*\* ";

cout << &ptr << endl<<endl;

while (ptr != NULL) {

cout <<"-----------------------------\n";

cout << "ptr->data: " << ptr->data << endl;

cout << "ptr: " << ptr << endl;

cout << "ptr->next: " << ptr->next << endl;

ptr = ptr->next;

}

}

int main() {

Node\* head = new Node(1);

head->next = new Node(2);

head->next->next = new Node(20);

head->next->next->next = new Node(30);

displayLinkedList(head);

// Free the allocated memory

Node\* current = head;

while (current != NULL) {

Node\* temp = current;

current = current->next;

delete temp;

}

return 0;

}

**Activity 2:**

Write a program that will implement single, doubly, and circular linked link list operations by showing a menu to the user.

The menu should be:

**Which linked list you want:**

1: Single

2: Double

3: Circular

After the option is chosen by the user:

**Which operation you want to perform:**

1: Insertion

2: Deletion

3: Display

4: Reverse

4: Seek

5: Exit

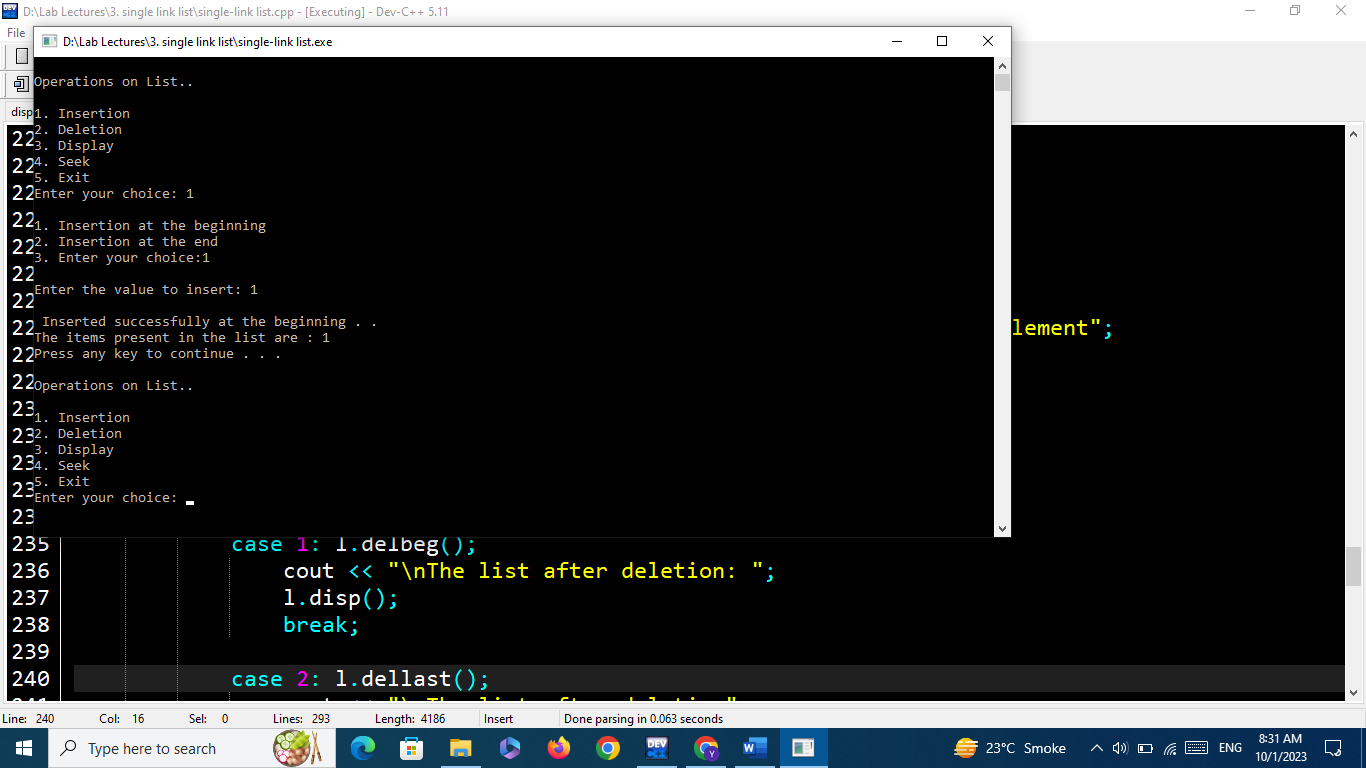
**Let's suppose, the user has chosen the insertion option then the following menu should be displayed:**

1: insertion at beginning

2: insertion at end

3: insertion at the specific data node

A sample output screenshot is below:



#include <iostream>

using namespace std;

class List {

struct Node {

int data;

Node\* next;

Node\* prev;

Node(int value) : data(value), next(NULL), prev(NULL) {}

};

Node\* head\_singly;

Node\* head\_doubly;

Node\* head\_circular;

public:

List() {

head\_singly = NULL;

head\_doubly = NULL;

head\_circular = NULL;

}

// Functions for inserting into singly, doubly, and circular linked lists

void insert\_beg\_singly(int n) {

Node\* newNode = new Node(n);

newNode->next = head\_singly;

head\_singly = newNode;

}

void insert\_end\_singly(int n) {

Node\* newNode = new Node(n);

if (head\_singly == NULL) {

head\_singly = newNode;

} else {

Node\* current = head\_singly;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

}

void insert\_beg\_doubly(int n) {

Node\* newNode = new Node(n);

newNode->next = head\_doubly;

newNode->prev = NULL;

if (head\_doubly != NULL) {

head\_doubly->prev = newNode;

}

head\_doubly = newNode;

}

void insert\_end\_doubly(int n) {

Node\* newNode = new Node(n);

newNode->next = NULL;

if (head\_doubly == NULL) {

newNode->prev = NULL;

head\_doubly = newNode;

} else {

Node\* current = head\_doubly;

while (current->next != NULL) {

current = current->next;

}

newNode->prev = current;

current->next = newNode;

}

}

void insert\_beg\_circular(int n) {

Node\* newNode = new Node(n);

if (head\_circular == NULL) {

head\_circular = newNode;

head\_circular->next = head\_circular;

} else {

Node\* current = head\_circular;

while (current->next != head\_circular) {

current = current->next;

}

newNode->next = head\_circular;

head\_circular = newNode;

current->next = head\_circular;

}

}

void insert\_end\_circular(int n) {

Node\* newNode = new Node(n);

if (head\_circular == NULL) {

head\_circular = newNode;

head\_circular->next = head\_circular;

} else {

Node\* current = head\_circular;

while (current->next != head\_circular) {

current = current->next;

}

newNode->next = head\_circular;

current->next = newNode;

}

}

// Functions for deleting from singly, doubly, and circular linked lists

void delete\_beg\_singly() {

if (head\_singly == NULL) {

return; // List is empty

}

Node\* temp = head\_singly;

head\_singly = head\_singly->next;

delete temp;

}

void delete\_end\_singly() {

if (head\_singly == NULL) {

return; // List is empty

} else if (head\_singly->next == NULL) {

delete head\_singly;

head\_singly = NULL;

} else {

Node\* current = head\_singly;

while (current->next->next != NULL) {

current = current->next;

}

delete current->next;

current->next = NULL;

}

}

void delete\_beg\_doubly() {

if (head\_doubly == NULL) {

return; // List is empty

}

Node\* temp = head\_doubly;

head\_doubly = head\_doubly->next;

if (head\_doubly != NULL) {

head\_doubly->prev = NULL;

}

delete temp;

}

void delete\_end\_doubly() {

if (head\_doubly == NULL) {

return; // List is empty

} else if (head\_doubly->next == NULL) {

delete head\_doubly;

head\_doubly = NULL;

} else {

Node\* current = head\_doubly;

while (current->next->next != NULL) {

current = current->next;

}

delete current->next;

current->next = NULL;

}

}

void delete\_beg\_circular() {

if (head\_circular == NULL) {

return; // List is empty

}

Node\* current = head\_circular;

while (current->next != head\_circular) {

current = current->next;

}

Node\* temp = head\_circular;

head\_circular = head\_circular->next;

current->next = head\_circular;

delete temp;

}

void delete\_end\_circular() {

if (head\_circular == NULL) {

return; // List is empty

} else if (head\_circular->next == head\_circular) {

delete head\_circular;

head\_circular = NULL;

} else {

Node\* current = head\_circular;

Node\* previous = NULL;

while (current->next != head\_circular) {

previous = current;

current = current->next;

}

previous->next = head\_circular;

delete current;

}

}

// Functions for seeking in singly, doubly, and circular linked lists

int seek\_singly(int value) {

Node\* current = head\_singly;

int index = 0;

while (current != NULL) {

if (current->data == value) {

return index; // Value found at this index

}

current = current->next;

index++;

}

return -1; // Value not found in the list

}

int seek\_doubly(int value) {

Node\* current = head\_doubly;

int forwardIndex = 0;

// Forward traversal

while (current != NULL) {

if (current->data == value) {

return forwardIndex; // Value found at this index

}

current = current->next;

forwardIndex++;

}

// Value not found in the forward direction, let's try backward

current = head\_doubly;

int backwardIndex = 0;

while (current->next != NULL) {

current = current->next;

}

while (current != NULL) {

if (current->data == value) {

return backwardIndex; // Value found at this index

}

current = current->prev;

backwardIndex--;

}

return -1; // Value not found in the list

}

int seek\_circular(int value) {

if (head\_circular == NULL) {

return -1; // List is empty

}

Node\* current = head\_circular;

int index = 0;

do {

if (current->data == value) {

return index; // Value found at this index

}

current = current->next;

index++;

} while (current != head\_circular);

return -1; // Value not found in the list

}

// Functions for reversing singly, doubly, and circular linked lists

void reverse\_singly() {

Node\* prev = NULL;

Node\* current = head\_singly;

Node\* nextNode = NULL;

while (current != NULL) {

nextNode = current->next;

current->next = prev;

prev = current;

current = nextNode;

}

head\_singly = prev;

}

void reverse\_doubly() {

Node\* current = head\_doubly;

Node\* temp = NULL;

while (current != NULL) {

temp = current->prev;

current->prev = current->next;

current->next = temp;

current = current->prev;

}

if (temp != NULL) {

head\_doubly = temp->prev;

}

}

void reverse\_circular() {

if (head\_circular == NULL) {

return; // List is empty

}

Node\* current = head\_circular;

Node\* prev = NULL;

Node\* nextNode = NULL;

do {

nextNode = current->next;

current->next = prev;

prev = current;

current = nextNode;

} while (current != head\_circular);

head\_circular->next = prev;

head\_circular = prev;

}

// Display functions for singly, doubly, and circular linked lists

void display\_singly() {

Node\* current = head\_singly;

while (current != NULL) {

cout << current->data << " -> ";

current = current->next;

}

cout << "NULL" << endl;

}

void display\_doubly() {

Node\* current = head\_doubly;

while (current != NULL) {

cout << current->data << " <-> ";

current = current->next;

}

cout << "NULL" << endl;

}

void display\_circular() {

if (head\_circular == NULL) {

cout << "Empty Circular Linked List" << endl;

return;

}

Node\* current = head\_circular;

do {

cout << current->data << " -> ";

current = current->next;

} while (current != head\_circular);

cout << "Head" << endl;

}

};

int main() {

List l;

int ch, val;

do {

cout << "\nOperations on link list" << endl;

cout << "1-Insertion \n2-Deletion \n3-Seek \n4-Reverse \n5-Display \n6-Exit" << endl;

cout << "\nEnter Your Choice: ";

cin >> ch;

int ps, ds;

switch (ch) {

case 1:

cout << "\n1-Insertion in Singly \n2-Insertion in Doubly \n3-Insertion in Circular" << endl;

cout << "\nEnter your choice: ";

cin >> ps;

switch (ps) {

case 1:

cout << "\n1-Insertion at beginning \n2-Insertion at end\n";

cout << "\nEnter Your Choice: ";

cin >> ds;

cout << "\nEnter Value to insert: ";

cin >> val;

switch (ds) {

case 1:

l.insert\_beg\_singly(val);

cout<<"\nData after insertion"<<endl;

l.display\_singly();

break;

case 2:

l.insert\_end\_singly(val);

cout<<"\nData after insertion"<<endl;

l.display\_singly();

break;

default:

cout << "\nInvalid choice." << endl;

}

break;

case 2:

cout << "\n1-Insertion at beginning \n2-Insertion at end" << endl;

cout << "\nEnter Your Choice: ";

cin >> ds;

cout << "\nEnter Value to insert: ";

cin >> val;

switch (ds) {

case 1:

l.insert\_beg\_doubly(val);

cout<<"\nData after insertion"<<endl;

l.display\_doubly();

break;

case 2:

l.insert\_end\_doubly(val);

cout<<"\nData after insertion"<<endl;

l.display\_doubly();

break;

default:

cout << "Invalid choice." << endl;

}

break;

case 3:

cout << "\n1-Insertion at beginning \n2-Insertion at end" << endl;

cout << "\nEnter Your Choice: ";

cin >> ds;

cout << "\nEnter Value to insert: ";

cin >> val;

switch (ds) {

case 1:

l.insert\_beg\_circular(val);

cout<<"\nData after insertion"<<endl;

l.display\_circular();

break;

case 2:

l.insert\_end\_circular(val);

cout<<"\nData after insertion"<<endl;

l.display\_circular();

break;

default:

cout << "Invalid choice." << endl;

}

break;

default:

cout << "Invalid choice." << endl;

}

break;

case 2:

cout << "\n1-Deletion in Singly \n2-Deletion in Doubly \n3-Deletion in Circular" << endl;

cout << "\nEnter your choice: ";

cin >> ps;

switch (ps) {

case 1:

cout << "\n1-Deletion at beginning \n2-Deletion at end ";

cout << "\nEnter Your Choice: ";

cin >> ds;

switch (ds) {

case 1:

l.delete\_beg\_singly();

cout<<"\nData after deletion"<<endl;

l.display\_singly();

break;

case 2:

l.delete\_end\_singly();

cout<<"\nData after deletion"<<endl;

l.display\_singly();

break;

default:

cout << "Invalid choice." << endl;

}

break;

case 2:

cout << "\n1-Deletion at beginning \n2-Deletion at end" << endl;

cout << "\nEnter Your Choice: ";

cin >> ds;

switch (ds) {

case 1:

l.delete\_beg\_doubly();

cout<<"\nData after deletion"<<endl;

l.display\_doubly();

break;

case 2:

l.delete\_end\_doubly();

cout<<"\nData after deletion"<<endl;

l.display\_doubly();

break;

default:

cout << "Invalid choice." << endl;

break;

}

break;

case 3:

cout << "\n1-Deletion at beginning \n2-Deletion at end" << endl;

cout << "\nEnter Your Choice: ";

cin >> ds;

switch (ds) {

case 1:

l.delete\_beg\_circular();

cout<<"\nData after deletion"<<endl;

l.display\_circular();

break;

case 2:

l.delete\_end\_circular();

cout<<"\nData after deletion"<<endl;

l.display\_circular();

break;

default:

cout << "Invalid choice." << endl;

break;

}

break;

default:

cout << "Invalid choice." << endl;

break;

}

break;

case 3:

cout << "\n1-Seek in Singly \n2-Seek in Doubly \n3-Seek in Circular" << endl;

cout << "\nEnter your choice: ";

cin >> ps;

switch (ps) {

case 1:

cout << "Enter the value to seek: ";

cin >> val;

int position\_singly;

position\_singly = l.seek\_singly(val);

if (position\_singly != -1) {

cout << "Value found at position " << position\_singly << endl;

} else {

cout << "Value not found in the singly linked list." << endl;

}

break;

case 2:

cout << "Enter the value to seek: ";

cin >> val;

int position\_doubly;

position\_doubly = l.seek\_doubly(val);

if (position\_doubly != -1) {

cout << "Value found at position " << position\_doubly << endl;

} else {

cout << "Value not found in the doubly linked list." << endl;

}

break;

case 3:

cout << "Enter the value to seek: ";

cin >> val;

int position\_circular;

position\_circular = l.seek\_circular(val);

if (position\_circular != -1) {

cout << "Value found at position " << position\_circular << endl;

} else {

cout << "Value not found in the circular linked list." << endl;

}

break;

default:

cout << "Invalid choice." << endl;

break;

}

break;

case 4:

cout<<"\n1-Reverse in singly \n2-Reverse in Doubly \n3-Reverse in Circular\n";

cout << "\nEnter your choice: ";

cin >> ps;

switch(ps){

case 1:

l.reverse\_singly();

cout<<"\nData after reversal"<<endl;

l.display\_singly();

break;

case 2:

l.reverse\_doubly();

cout<<"\nData after reversal"<<endl;

l.display\_doubly();

break;

case 3:

l.reverse\_circular();

cout<<"\nData after reversal"<<endl;

l.display\_circular();

break;

default:

cout<<"Invalid Command"<<endl;

}

break;

case 5:

cout << "\n1-Display Singly \n2-Display Doubly \n3-Display Circular" << endl;

cout << "\nEnter your choice: ";

cin >> ps;

switch (ps) {

case 1:

cout << "Data in Singly Linked List:" << endl;

l.display\_singly();

break;

case 2:

cout << "Data in Doubly Linked List:" << endl;

l.display\_doubly();

break;

case 3:

cout << "Data in Circular Linked List:" << endl;

l.display\_circular();

break;

default:

cout << "Invalid choice." << endl;

break;

}

break;

case 6:

return 0; // Exit the program

default:

cout << "Invalid choice." << endl;

break;

}

} while (ch != 6);

return 0;

}

**You can get help from the below link:**

[**https://github.com/programming-debug/Data-Structure-Lab/blob/main/Lab3/single-link%20list.cpp**](https://github.com/programming-debug/Data-Structure-Lab/blob/main/Lab3/single-link%20list.cpp)

In this Word file, you should place the code and its output screenshot.

After completing the activities, Upload the final pdf and cpp code files to the “**DSA\_Lab”** repository.