1. Write a program to implement single Link List (with function insertion at first node, insertion at last node, deletion at first node, deletion at last node, insertion and deletion at middle node and traversing of link list).

#include <iostream>

using namespace std;

class node{

public:

int data;

node\*next;

node(int data)

{

this->data=data;

this->next=NULL;

}

~node()

{

int value=this->data;

if(this->next!=NULL)

{

delete next;

this->next=NULL;

}

cout<<"memory is free for node with data : "<<value<<endl;

}

};

void insertathead(node\* &head,int d)

{

node\*temp=new node(d);

temp ->next=head;

head=temp;

}

void insertattail(node\* &tail,int d)

{

node\*temp=new node(d);

tail ->next=temp;

tail=temp;

}

void print(node\* &head)

{

node\*temp=head;

while(temp!=NULL)

{

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

temp=temp->next;

}

cout<<endl;

}

void insertatposition(node\*&tail,node\*&head,int position,int d)

{

if(position==1)

{

insertathead(head,d);

return;

}

node\*temp=head;

int cnt=1;

while(cnt<position-1)

{

temp=temp->next;

cnt++;

}

if(temp->next==NULL)

{

insertattail(tail,d);

return;

}

node\* nodetoinsert=new node(d);

nodetoinsert->next=temp->next;

temp->next=nodetoinsert;

}

void deletenode(int position,node\*&head)

{

if(position==1)

{

node\*temp=head;

head=temp->next;

temp->next=NULL;

delete temp;

}

else

{

node\* curr=head;

node\* prev=NULL;

int cnt=1;

while(cnt<position)

{

prev=curr;

curr=curr->next;

cnt++;

}

prev->next=curr->next;

curr->next=NULL;

delete curr;

}

}

int main()

{

node\*node1=new node(10);

node\*head=node1;

node\*tail=node1;

print(head);

cout<<"Adding at head"<<endl;

insertathead(head,9);

print(head);

insertathead(head,8);

print(head);

insertathead(head,7);

print(head);

insertathead(head,6);

print(head);

insertathead(head,5);

print(head);

cout<<"Adding at tail"<<endl;

insertattail(tail,11);

print(head);

insertattail(tail,12);

print(head);

insertattail(tail,13);

print(head);

cout<<"Adding at specific position"<<endl;

insertatposition(tail,head,10,14);

print(head);

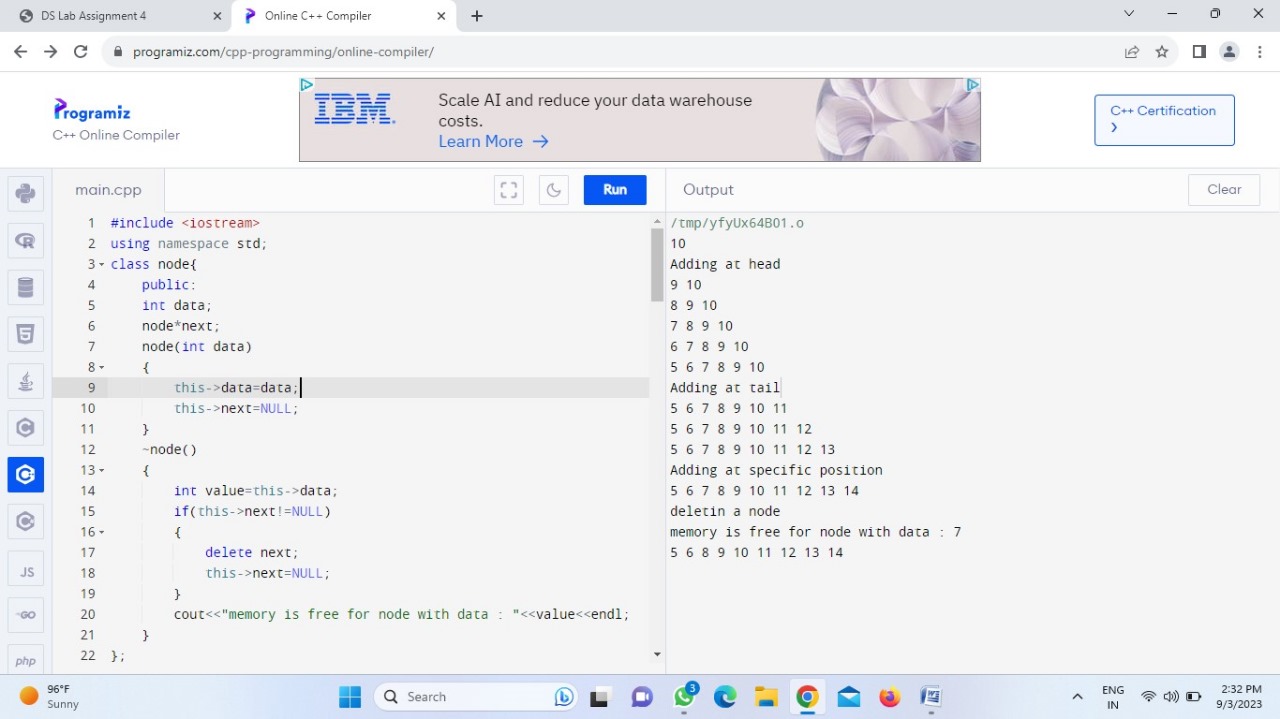
cout<<"deletin a node"<<endl;

deletenode(3,head);

print(head);

     return 0;

}



1. Write a program to implement Doubly Link List (with function insertion at first node, insertion at last node, deletion at first node, deletion at last node, insertion and deletion at middle node and traversing of link list).

#include <iostream>

using namespace std;

class node{

public:

int data;

node\*next;

node\*prev;

node(int data)

{

this->data=data;

this->next=NULL;

this->prev=NULL;

}

};

void insertathead(node\* &head,node\* &tail,int d)

{ if(head==NULL)

{

node\*temp=new node(d);

head=temp;

tail=temp;

}

else{

node\*temp=new node(d);

temp ->next=head;

head->prev=temp;

head=temp;}

}

void insertattail(node\* &head,node\* &tail,int d)

{ if(tail==NULL)

{

node\*temp=new node(d);

tail=temp;

head=temp;

}

else{

node\*temp=new node(d);

tail->next=temp;

temp->prev=tail;

tail=temp;}

}

void print(node\* &head)

{

node\*temp=head;

while(temp!=NULL)

{

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

temp=temp->next;

}

cout<<endl;

}

void insertatposition(node\*&head,node\*&tail,int position,int d)

{

if(position==1)

{

insertathead(head,tail,d);

return;

}

node\*temp=head;

int cnt=1;

while(cnt<position-1)

{

temp=temp->next;

cnt++;

}

if(temp->next==NULL)

{

insertattail(head,tail,d);

return;

}

//creating a node for d

node\*nodetoinsert=new node(d);

temp->next->prev=nodetoinsert;

nodetoinsert->next=temp->next;

temp->next=nodetoinsert;

nodetoinsert->prev=temp;

}

void deletenode(int position,node\*&head)

{

if(position==1)

{

node\*temp=head;

temp->next->prev=NULL;

head=temp->next;

temp->next=NULL;

delete temp;

}

else

{

node\* curr=head;

node\* prev=NULL;

int cnt=1;

while(cnt<position)

{

prev=curr;

curr=curr->next;

cnt++;

}

curr->prev=NULL;

prev->next=curr->next;

curr->next=NULL;

delete curr;

}

}

int main()

{

node\*node1=new node(10);

node\*head=NULL;

node\*tail=NULL;

print(head);

insertattail(head,tail,13);

print(head);

cout<<"Inserting at tail"<<endl;

insertattail(head,tail,14);

print(head);

insertattail(head,tail,15);

print(head);

cout<<"Inserting at head"<<endl;

insertathead(head,tail,12);

print(head);

insertathead(head,tail,11);

print(head);

insertathead(head,tail,10);

print(head);

cout<<"Inserting at specific position"<<endl;

insertatposition(head,tail,7,56);

print(head);

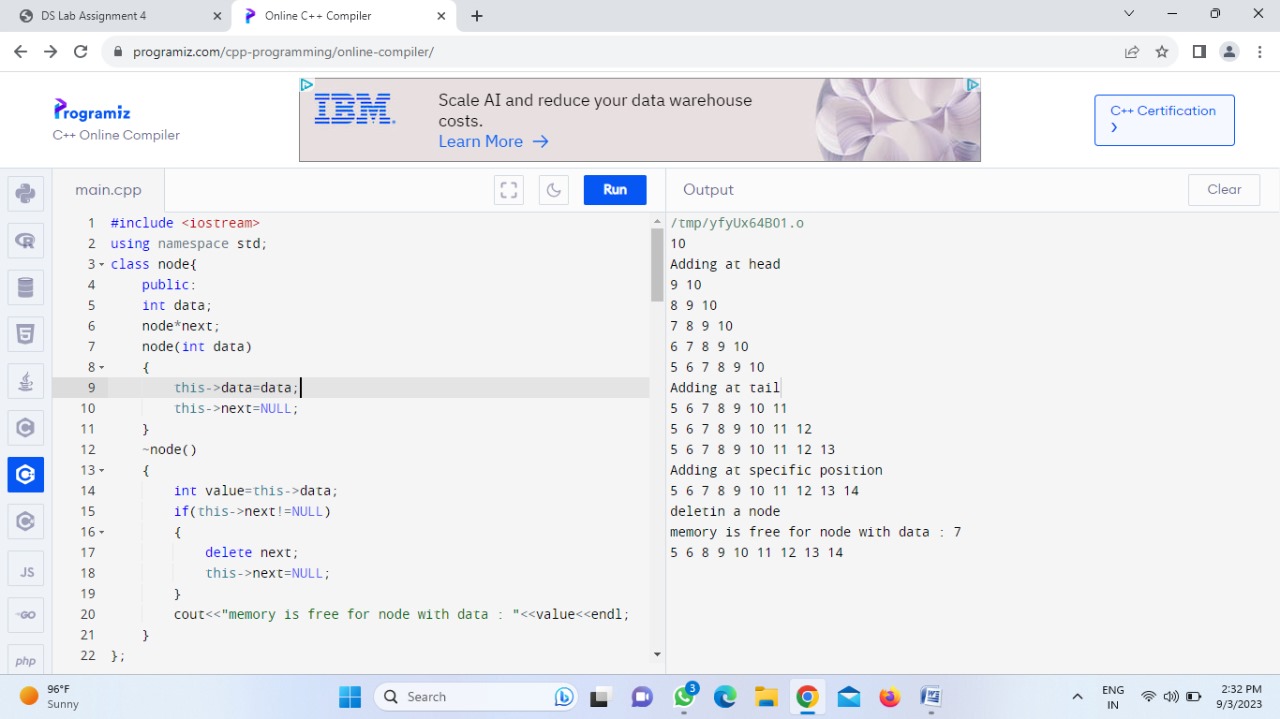
cout<<"Deleting a node"<<endl;

deletenode(3,head);

print(head);

    return 0;

}



1. Write a program to implement Circular Doubly Link List (with function insertion at first node, insertion at last node, deletion at first node, deletion at last node, insertion and deletion at middle node and traversing of link list).

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node\* prev;

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

};

class CircularDoublyLinkedList {

private:

Node\* head;

public:

CircularDoublyLinkedList() : head(nullptr) {}

// Function to insert a node at the beginning

void insertAtFirst(int value) {

Node\* newNode = new Node(value);

if (!head) {

newNode->next = newNode;

newNode->prev = newNode;

head = newNode;

} else {

newNode->next = head;

newNode->prev = head->prev;

head->prev->next = newNode;

head->prev = newNode;

head = newNode;

}

}

// Function to insert a node at the end

void insertAtLast(int value) {

Node\* newNode = new Node(value);

if (!head) {

newNode->next = newNode;

newNode->prev = newNode;

head = newNode;

} else {

newNode->next = head;

newNode->prev = head->prev;

head->prev->next = newNode;

head->prev = newNode;

}

}

// Function to delete the first node

void deleteAtFirst() {

if (head) {

if (head->next == head) {

delete head;

head = nullptr;

} else {

Node\* temp = head;

head->prev->next = head->next;

head->next->prev = head->prev;

head = head->next;

delete temp;

}

}

}

// Function to delete the last node

void deleteAtLast() {

if (head) {

if (head->next == head) {

delete head;

head = nullptr;

} else {

Node\* temp = head->prev;

temp->prev->next = head;

head->prev = temp->prev;

delete temp;

}

}

}

// Function to insert a node at the middle

void insertAtMiddle(int value, int position) {

if (position <= 0) {

insertAtFirst(value);

} else {

Node\* newNode = new Node(value);

Node\* current = head;

int count = 0;

while (current && count < position) {

current = current->next;

count++;

}

if (current) {

newNode->next = current;

newNode->prev = current->prev;

current->prev->next = newNode;

current->prev = newNode;

}

}

}

// Function to delete a node at the middle

void deleteAtMiddle(int position) {

if (position <= 0) {

deleteAtFirst();

} else {

Node\* current = head;

int count = 0;

while (current && count < position) {

current = current->next;

count++;

}

if (current && current != head) {

current->prev->next = current->next;

current->next->prev = current->prev;

delete current;

}

}

}

// Function to traverse and print the linked list

void traverse() {

Node\* current = head;

if (current) {

do {

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

current = current->next;

} while (current != head);

}

std::cout << std::endl;

}

};

int main() {

CircularDoublyLinkedList myList;

myList.insertAtFirst(1);

myList.traverse();

cout<<"INSERTING VALUES AT HEAD"<<endl;

myList.insertAtFirst(2);

myList.traverse();

cout<<"INSERTING VALUES AT TAIL"<<endl;

myList.insertAtLast(3);

myList.traverse();

cout<<"INSERTING VALUES AT MIDDLE"<<endl;

myList.insertAtMiddle(4, 1);

myList.traverse();

cout<<"DELETING VALUES AT HEAD"<<endl;

myList.deleteAtFirst();

myList.traverse();

cout<<"DELETING VALUES AT TAIL"<<endl;

myList.deleteAtLast();

myList.traverse();

cout<<"DELETING VALUES AT MIDDLE"<<endl;

myList.deleteAtMiddle(1);

myList.traverse();

    return 0;

}

