**Experiment 4.1**

**Student Name:** Sahil Kaundal **UID:** 21BCS8197

**Branch:** BE CSE (Lateral Entry) **Section/Group:** 616/A

**Semester:** 5th **Date of Performance:** 06/09/2022

**Subject Name:** DAA Lab **Subject Code:** 21-CSP-312

# Aim/Overview of the practical:

Code to Insert and Delete an element at the beginning and at end in Doubly and Circular Linked List.

# Task to be done/ Which logistics used:

Insert and delete an element from a doubly circular linked list.

# Algorithm/Flowchart:

* Start.
* For insertion in the end if the list is empty start pointer points to the first node the list.If the list is non empty previous pointer of M points to last node, next pointer of M points to first node and last node’s next pointer points to this M node and first node’s previous pointer points to this M node
* For Insertion at the beginning if the list is empty T next pointer points to first node of the list, T previous pointer points to last node the list, last node’s next pointer points to this T node, first node’s previous pointer also points this T node and shift ‘Start’ pointer to this T node.
* If the list is not empty, then we define two pointers curr and prev\_1 and initialize the pointer curr points to the first node of the list, and prev\_1 = NULL.
* Traverse the list using the curr pointer to find the node to be deleted and before moving from curr to the next node, every time set prev\_1 = curr.
* If the node is found, check if it is the only node in the list. If yes, set start = NULL and free the node pointing by curr.
* If the list has more than one node, check if it is the first node of the list. The condition to check this is (curr == start). If yes, then move prev\_1 to the last node(prev\_1 = start -> prev).
* If curr is not the first node, we check if it is the last node in the list. The condition to check this is (curr -> next == start). If yes, set prev\_1 -> next = start and start -> prev = prev\_1. Free the node pointing by curr.
* If the node to be deleted is neither the first node nor the last node, declare one more pointer temp and initialize the pointer temp points to the next of curr pointer (temp = curr>next). Now set, prev\_1 -> next
  + = temp and temp ->prev = prev\_1. Free the node pointing by curr. 8.
* Stop and print the result.
* Insertion in doubly linked list
* Algorithm :
* Step 1: IF ptr = NULL
* Write OVERFLOW
* Go to Step 9
* [END OF IF]
* Step 2: SET NEW\_NODE = ptr
* Step 3: SET ptr = ptr -> NEXT
* Step 4: SET NEW\_NODE -> DATA = VAL Step 1: Let m be largest number in array
* Step 5: SET NEW\_NODE -> PREV = NULL
* Step 6: SET NEW\_NODE -> NEXT = START
* Step 7: SET head -> PREV = NEW\_NODE
* Step 8: SET head = NEW\_NODE
* Step 9: EXIT
* Deletion
* Algorithm
* STEP 1: IF HEAD = NULL
* WRITE UNDERFLOW
* GOTO STEP 6
* STEP 2: SET PTR = HEAD
* STEP 3: SET HEAD = HEAD → NEXT
* STEP 4: SET HEAD → PREV = NULL
* STEP 5: FREE PTR
* STEP 6: EXIT

# Steps for experiment/practical/Code:

#include<iostream>

using namespace std;

class Node{

public:

int data;

Node\* next;

Node\* prev;

Node(int d){

data=d;

next=NULL;

prev=NULL;

}

};

Node\* insert(Node\* head,int data){

Node\* n=new Node(data);

if(head==NULL){

head=n;

return head;

}

n->next=head;

head->prev=n;

head=n;

return head;

}

Node\* insertAtEnd(Node\* head,int data){

Node\* n=new Node(data);

if(head==NULL){

head=n;

return head;

}

Node\* temp=head;

while(temp->next!=NULL){

temp=temp->next;

}

temp->next=n;

n->prev=temp;

return head;

}

Node\* pop(Node\* head){

if(head==NULL){

return head;

}

Node\* temp=head;

head=head->next;

delete(temp);

return head;

}

Node\* deleteLast(Node\* head){

Node\* temp=head;

while(temp->next!=NULL){

temp=temp->next;

}

temp->prev->next=NULL;

delete(temp);

return head;

}

void printList(Node\* head){

Node\* temp=head;

while(temp!=NULL){

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

temp=temp->next;

}

}

int main(){

Node\* head=NULL;

head=insert(head,5);

head=insert(head,4);

head=insert(head,3);

head=insert(head,2);

head=insert(head,1);

head=insertAtEnd(head,6);

head=pop(head);

head=deleteLast(head);

printList(head);

cout<<endl;

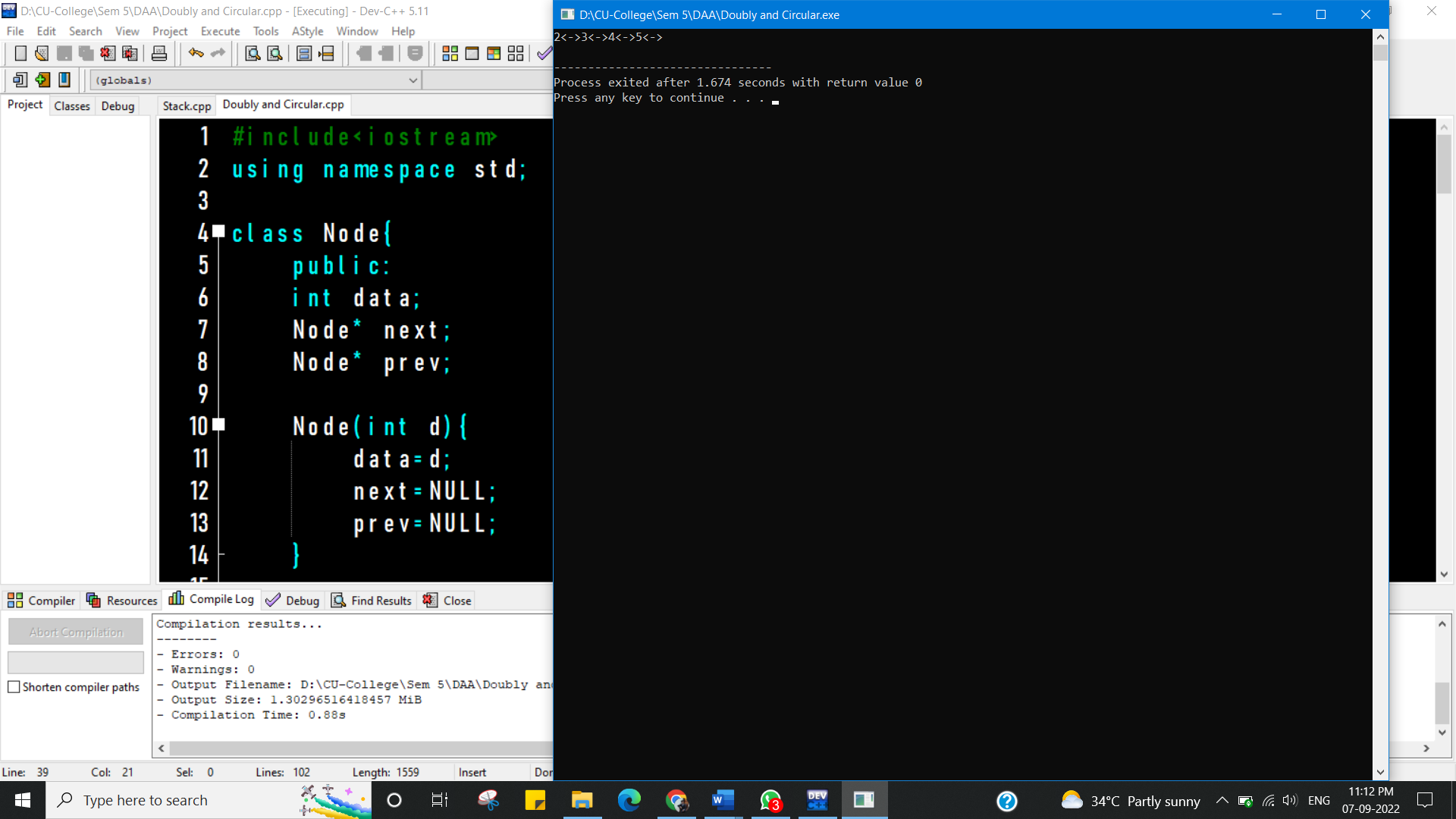
}

# Observations/Discussions/Complexity Analysis:

***Time Complexity:*** O(n)

***Space Complexity:*** O(N)

# Result/Output/Writing Summary:



**Learning outcomes (What I have learnt):**

* Learnt how to implement Doubly Linked List.
* Learnt how to insert node at beginning of doubly linked list.
* Learnt how to insert node at end of doubly linked list.
* Learnt how to delete node at beginning of doubly linked list.
* Learnt how to delete node at end of doubly linked list.

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |
|  |  |  |  |