LAB # 04 - ASSIGNMENT

13

NOV

2023

**GROUP 16**

MUHAMMAD YASIR | CT-22082

MUHAMMAD SHAHEER QURESHI | CT-22090

SYED SAAD WAQAR | CT-22097

AMMAR YASSER AHMED | CT-22103

DATA STRUCTURES ALGORITHMS AND APPLICATIONS (ct – 159)

TEACHER : SIR ABDUL KARIM KAZI

**EXERCISE**

QUESTION 1:

/\*

Group 16

DSAA LAB 4

13 NOV 2023

Question 1

1. Create a doubly link list and perform the mentioned tasks.

a. Insert a new node at the end of the list.

b. Insert a new node at the beginning of list.

c. Insert a new node at given position.

d. Delete any node.

e. Print the complete doubly link list.

\*/

#include<iostream>

#include<cstdlib>

using namespace std;

struct Node{

int data;

struct Node\* prev;

struct Node\* next;

};

class DoublyLL{

Node\* head;

public:

DoublyLL(int data){

Node\* newNode = (struct Node\*) malloc (sizeof(struct Node));

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

head = newNode;

}

void insert\_at\_head(int data){

Node\* newNode = (struct Node\*) malloc (sizeof(struct Node));

newNode->data = data;

newNode->prev = NULL;

newNode->next = head;

head->prev = newNode;

head = newNode;

}

void insert\_at\_end(int data){

Node\* newNode = (struct Node\*) malloc (sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

Node\* temp = head;

while(temp->next != NULL){

temp = temp->next;

}

newNode->prev = temp;

temp->next = newNode;

}

void insert\_in\_between(int data, int position){

Node\* newNode = (struct Node\*) malloc (sizeof(struct Node));

newNode->data = data;

Node \*temp = head;

int i = 0;

while(i != position-1){

temp = temp->next;

i++;

}

newNode->next = temp->next;

newNode->prev = temp;

Node\* temp2 = temp->next;

temp2->prev = newNode;

temp->next = newNode;

}

void delete\_first\_node(){

Node\* temp = head;

head = head->next;

head->prev = NULL;

free(temp);

}

void delete\_last\_node(){

Node\* temp = head;

while(temp->next != NULL){

temp = temp->next;

}

Node \*temp2 = temp->prev;

temp->prev = NULL;

temp2->next = NULL;

free(temp);

}

void delete\_in\_between(int position){

Node\* temp = head;

int i = 0;

while(i != position-1){

temp = temp->next;

i++;

}

Node\* temp2 = temp->next;

Node\* temp3 = temp->next = temp2->next;

temp3->prev = temp;

free(temp2);

}

void doubleLLTraversal(){

Node\* temp = head, \*temp2;

cout << "Doubly Linked List Traversal: ";

while(temp != NULL){

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

if(temp->next == NULL){

temp2 = temp;

}

temp = temp->next;

}

while(temp2 != NULL){

cout << temp2->data << " ";

temp2 = temp2->prev;

}

cout << endl;

}

};

int main(){

DoublyLL D1(12);

D1.insert\_at\_end(13);

cout << "D1.insert\_at\_end(13);\n\t";

D1.doubleLLTraversal();

D1.insert\_at\_end(14);

cout << "D1.insert\_at\_end(14);\n\t";

D1.doubleLLTraversal();

D1.insert\_at\_end(15);

cout << "D1.insert\_at\_end(15);\n\t";

D1.doubleLLTraversal();

D1.insert\_at\_end(16);

cout << "D1.insert\_at\_end(16);\n\t";

D1.doubleLLTraversal();

D1.insert\_at\_head(11);

cout << "D1.insert\_at\_head(11);\n\t";

D1.doubleLLTraversal();

D1.insert\_in\_between(22, 2);

cout << "D1.insert\_in\_between(22, 2);\n\t";

D1.doubleLLTraversal();

D1.delete\_first\_node();

cout << "D1.delete\_first\_node();\n\t";

D1.doubleLLTraversal();

D1.delete\_last\_node();

cout << "D1.delete\_last\_node();\n\t";

D1.doubleLLTraversal();

D1.delete\_in\_between(1);

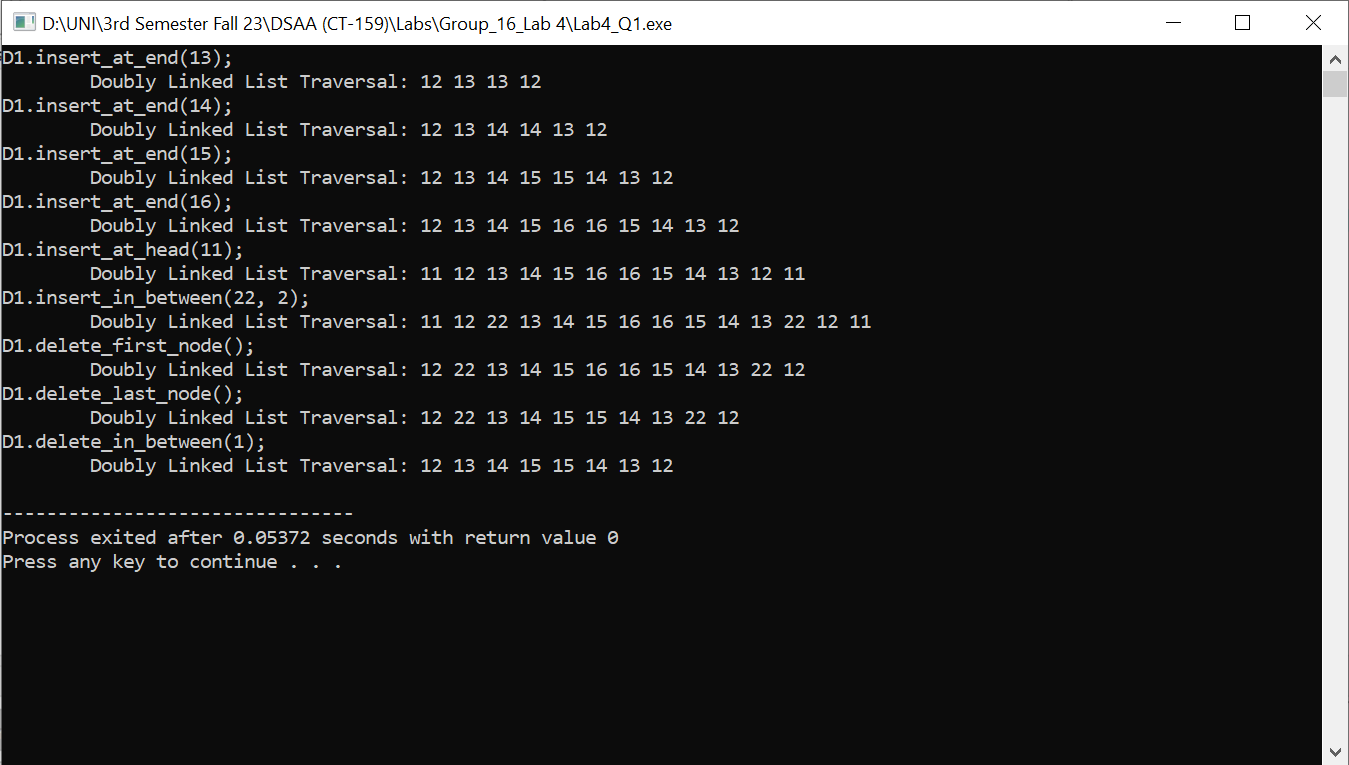
cout << "D1.delete\_in\_between(1);\n\t";

D1.doubleLLTraversal();

return 0;

}

OUTPUT:



QUESTION 2:

/\*

Group 16

DSAA LAB 4

13 NOV 2023

Question 2

2. Create two doubly link lists, say L and M . List L should contain all even elements from 2 to

10 and list M should contain all odd elements from 1 to 9. Create a new list N by

concatenating list L and M.

\*/

#include <iostream>

#include <cstdlib>

using namespace std;

void swap(int \*a, int\*b){

\*a = \*a + \*b;

\*b = \*a - \*b;

\*a = \*a - \*b;

}

struct Node{

int data;

struct Node\* prev;

struct Node\* next;

};

class DoublyLL{

struct Node\* head;

public:

DoublyLL(int data){

Node\* newnode = (struct Node\*)malloc(sizeof(struct Node));

newnode->data = data;

newnode->next = NULL;

newnode->prev = NULL;

head = newnode;

}

void insert\_at\_head(int data){

Node\* newnode = (struct Node\*)malloc(sizeof(struct Node));

newnode->data = data;

newnode->next = head;

newnode->prev = NULL;

head->prev = newnode;

head = newnode;

}

void insert\_at\_end(int data){

Node\* newnode = (struct Node\*)malloc(sizeof(struct Node));

newnode->data = data;

newnode->next = NULL;

Node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

newnode->prev = temp;

temp->next = newnode;

}

void concatenateLL(DoublyLL List){

Node\* temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = List.head;

List.head->prev = temp;

}

void DoublyTraversal(){

Node\* temp = head;

cout << "Doubly Linked List: ";

while(temp->next != NULL){

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

temp = temp->next;

}

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

while(temp != head){

temp = temp->prev;

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

}

cout << endl;

}

};

int main() {

DoublyLL L(2), M(1);

for(int i = 4; i <= 10; i+=2){

L.insert\_at\_end(i);

}

for(int i = 3; i <= 9; i+=2){

M.insert\_at\_end(i);

}

cout << "L list: ";

L.DoublyTraversal();

cout << "M list: ";

M.DoublyTraversal();

L.concatenateLL(M);

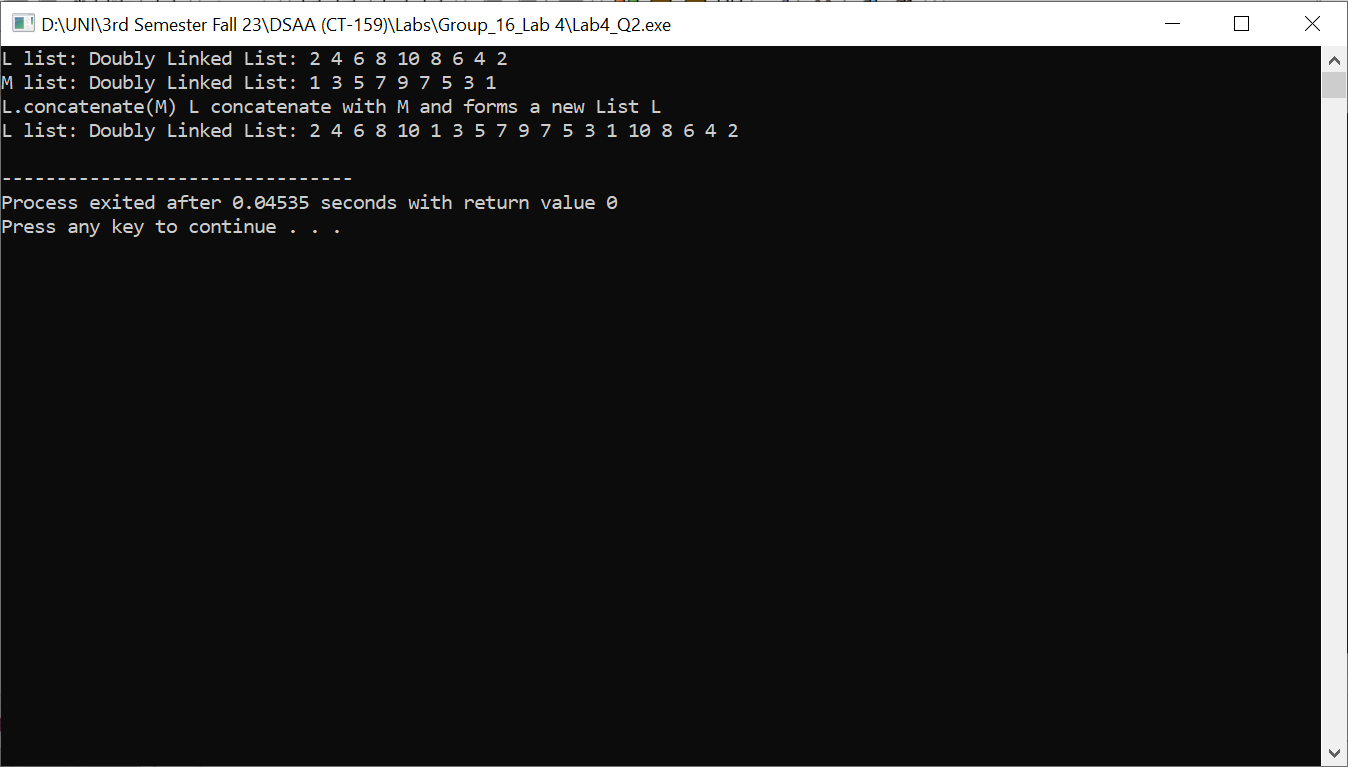
cout << "L.concatenate(M) L concatenate with M and forms a new List L\nL list: ";

L.DoublyTraversal();

return 0;

}

OUTPUT:



QUESTION 3:

/\*

Group 16

DSAA LAB 4

13 NOV 2023

Question 3

3. Using the above created list N, sort the contents of list N is descending order.

\*/

#include <iostream>

#include <cstdlib>

using namespace std;

void swap(int \*a, int\*b){

\*a = \*a + \*b;

\*b = \*a - \*b;

\*a = \*a - \*b;

}

struct Node{

int data;

struct Node\* prev;

struct Node\* next;

};

class DoublyLL{

struct Node\* head;

public:

DoublyLL(int data){

Node\* newnode = (struct Node\*)malloc(sizeof(struct Node));

newnode->data = data;

newnode->next = NULL;

newnode->prev = NULL;

head = newnode;

}

void insert\_at\_head(int data){

Node\* newnode = (struct Node\*)malloc(sizeof(struct Node));

newnode->data = data;

newnode->next = head;

newnode->prev = NULL;

head->prev = newnode;

head = newnode;

}

void insert\_at\_end(int data){

Node\* newnode = (struct Node\*)malloc(sizeof(struct Node));

newnode->data = data;

newnode->next = NULL;

Node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

newnode->prev = temp;

temp->next = newnode;

}

void concatenateLL(DoublyLL List){

Node\* temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = List.head;

List.head->prev = temp;

}

void sort(){

Node\* temp = head, \*temp2, \*temp3 = head;

while(temp3->next != NULL){

while(temp->next != NULL){

temp2 = temp;

temp = temp->next;

if(temp->data > temp2->data){

swap(temp->data, temp2->data);

}

}

temp = head;

temp3 = temp3->next;

}

}

void DoublyTraversal(){

Node\* temp = head;

cout << "Doubly Linked List: ";

while(temp->next != NULL){

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

temp = temp->next;

}

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

while(temp != head){

temp = temp->prev;

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

}

cout << endl;

}

};

int main() {

DoublyLL L(2), M(1);

for(int i = 4; i <= 10; i+=2){

L.insert\_at\_end(i);

}

for(int i = 3; i <= 9; i+=2){

M.insert\_at\_end(i);

}

cout << "L list: ";

L.DoublyTraversal();

cout << "M list: ";

M.DoublyTraversal();

L.concatenateLL(M);

cout << "L.concatenate(M) L concatenate with M and forms a new List L\nL list: ";

L.DoublyTraversal();

L.sort();

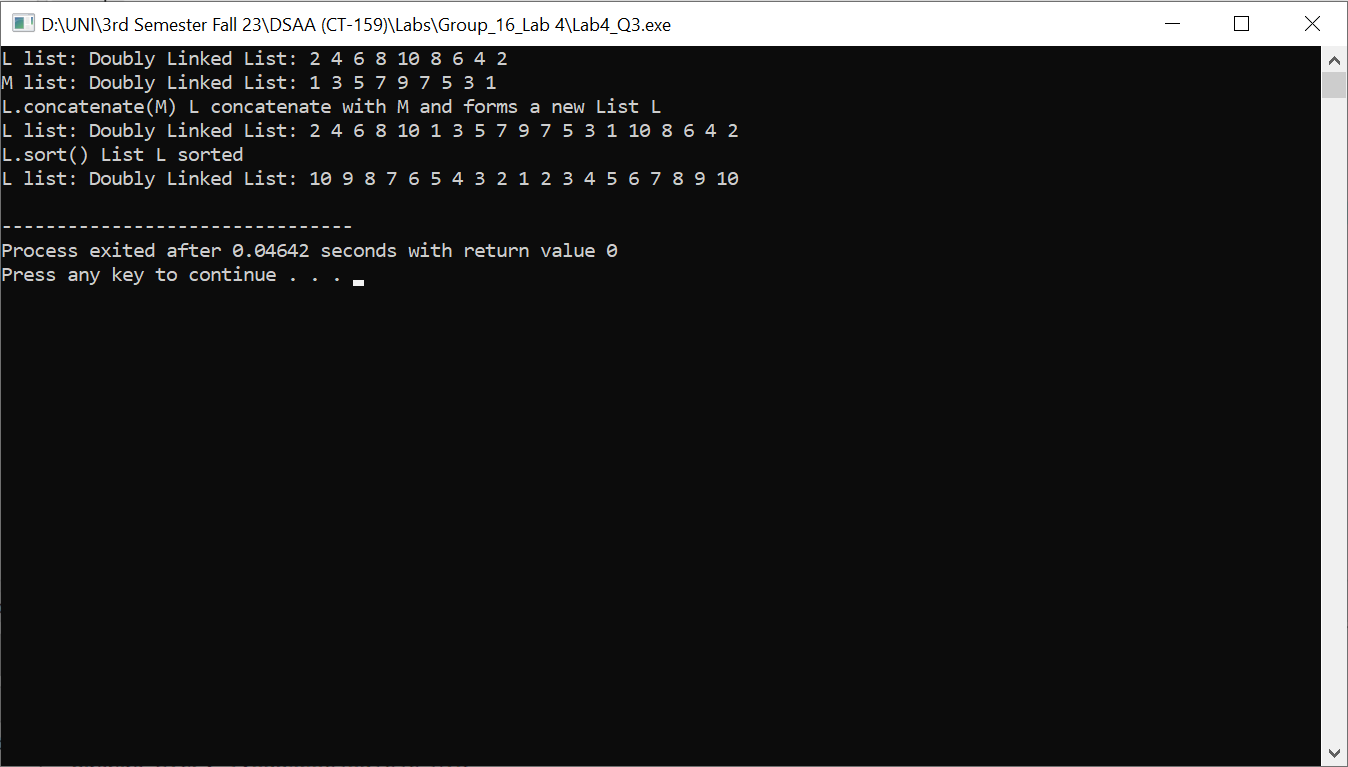
cout << "L.sort() List L sorted\nL list: ";

L.DoublyTraversal();

return 0;

}

OUTPUT:



QUESTION 4:

/\*

Group 16

DSAA LAB 4

13 NOV 2023

Question 4

4. Create a circular link list and perform the mentioned tasks.

a. Insert a new node at the end of the list.

b. Insert a new node at the beginning of list.

c. Insert a new node at given position.

d. Delete any node.

e. Print the complete circular link list.

\*/

#include<iostream>

#include<cstdlib>

using namespace std;

struct Node{

int data;

struct Node\* next;

};

class LinkedList{

struct Node\* head;

public:

LinkedList(int data){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

head = newNode;

}

void insert\_at\_head(int data){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = head;

head = newNode;

}

void insert\_at\_end(int data){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

Node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = newNode;

}

void insert\_in\_between(int data, int position){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

Node \*temp = head;

int i = 0;

while(i != position-1){

temp = temp->next;

i++;

}

newNode->next = temp->next;

temp->next = newNode;

}

void insert\_after\_node(int data, struct Node\* node){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = node->next;

node->next = newNode;

}

void delete\_first\_node(){

Node\* temp = head;

head = head->next;

free(temp);

}

void delete\_last\_node(){

Node\* temp = head;

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

temp = temp->next;

}

Node \*temp2 = temp->next;

temp->next = NULL;

free(temp2);

}

void delete\_middle\_node(int position){

Node\* temp = head;

int i = 0;

while(i != position - 1){

temp = temp->next;

i++;

}

Node\* temp2 = temp->next;

temp->next = temp2->next;

free(temp2);

}

void delete\_given\_value\_node(int data){

Node\* temp = head, \*temp2;

while((temp->data != data) && (temp->next != NULL)){

temp2 = temp;

temp = temp->next;

}

if(temp->data == data){

temp2->next = temp->next;

free(temp);

}

}

void singlyLinkedListTraversal(){

Node\* temp = head;

cout << "Singly Linked List Traversal: ";

while(temp != NULL){

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

temp = temp->next;

}

cout << endl;

}

void circularLLConversion(){

Node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = head;

}

void circularLinkedListTraversal(){

Node \*temp = head;

cout << "Circular Linked List Traversal: ";

do{

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

temp = temp->next;

}while(temp != head);

cout << endl;

}

void CCinsert\_at\_head(int data){

Node \*newNode = (struct Node\*)malloc (sizeof(struct Node));

newNode->data = data;

newNode->next = head;

Node \*ptr = head;

while(ptr->next != head){

ptr = ptr->next;

}

ptr->next = newNode;

head = newNode;

}

void CCinsert\_at\_end(int data){

Node \*newNode = (struct Node\*)malloc (sizeof(struct Node));

newNode->data = data;

newNode->next = head;

Node \*ptr = head;

while(ptr->next != head){

ptr = ptr->next;

}

ptr->next = newNode;

}

void CCinsert\_in\_between(int data, int position){

Node \*newNode = (struct Node\*) malloc (sizeof(struct Node));

newNode->data = data;

Node\* temp = head;

for(int i = 0; i < position-1; i++){

temp = temp->next;

}

newNode->next = temp->next;

temp->next = newNode;

}

void CCinsert\_after\_node(int data, struct Node\* node){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = node->next;

node->next = newNode;

}

void CCdelete\_first\_node(){

Node\* temp = head;

while(temp->next != head){

temp = temp->next;

}

Node \*temp2 = head;

temp->next = temp2->next;

head = temp->next;

free(temp2);

}

void CCdelete\_end\_node(){

Node \*temp = head, \*temp2;

while(temp->next != head){

temp2 = temp;

temp = temp->next;

}

temp2->next = head;

free(temp);

}

void CCdelete\_in\_between(int position){

Node \*temp = head;

int i = 0;

while(i != position-1){

temp = temp->next;

i++;

}

Node\* temp2 = temp->next;

temp->next = temp2->next;

free(temp2);

}

};

int main(){

LinkedList L1(12);

L1.insert\_at\_end(13);

L1.insert\_at\_end(14);

L1.insert\_at\_end(15);

L1.insert\_at\_end(16);

L1.insert\_at\_end(17);

L1.insert\_at\_end(18);

L1.insert\_at\_end(19);

L1.singlyLinkedListTraversal();

cout << "Circular Linked list Conversion...\n";

L1.circularLLConversion();

cout << "Circular Linked list Traversal:\n\t";

L1.circularLinkedListTraversal();

L1.CCinsert\_at\_head(11);

cout << "CC insert at head (11):\n\t";

L1.circularLinkedListTraversal();

L1.CCinsert\_at\_end(20);

cout << "CC insert at end (20):\n\t";

L1.circularLinkedListTraversal();

L1.CCinsert\_in\_between(22, 2);

cout << "CC insert in between (22, 2):\n\t";

L1.circularLinkedListTraversal();

L1.CCdelete\_in\_between(2); // index starts from 0, will delete 3rd element

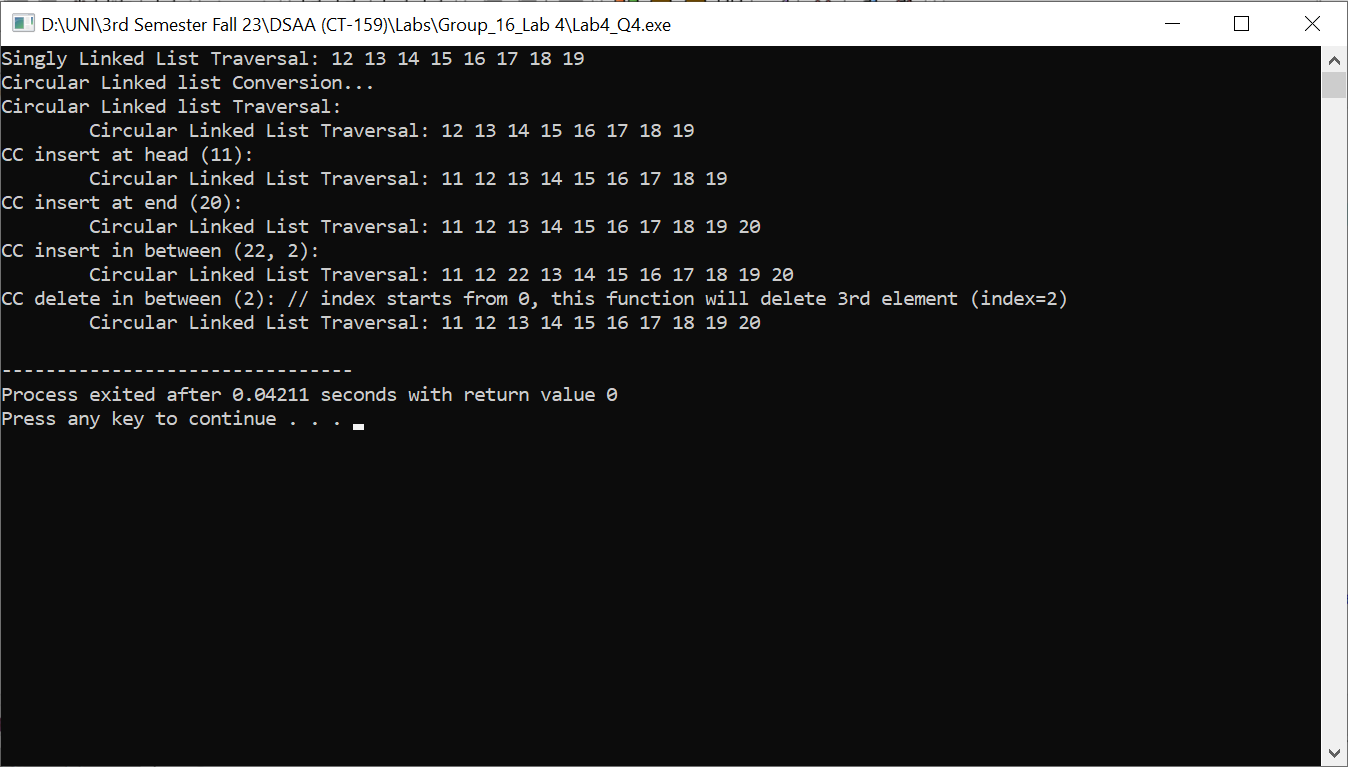
cout << "CC delete in between (2): // index starts from 0, this function will delete 3rd element (index=2)\n\t";

L1.circularLinkedListTraversal();

return 0;

}

OUTPUT:



QUESTION 5:

/\*

Group 16

DSAA LAB 4

13 NOV 2023

Question 5

5. Break the above-created circular linked list into two halves.

\*/

#include<iostream>

#include<cstdlib>

using namespace std;

struct Node{

int data;

struct Node\* next;

};

class LinkedList{

struct Node\* head;

public:

LinkedList(int data){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

head = newNode;

}

void insert\_at\_head(int data){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = head;

head = newNode;

}

void insert\_at\_end(int data){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

Node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = newNode;

}

void insert\_in\_between(int data, int position){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

Node \*temp = head;

int i = 0;

while(i != position-1){

temp = temp->next;

i++;

}

newNode->next = temp->next;

temp->next = newNode;

}

void insert\_after\_node(int data, struct Node\* node){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = node->next;

node->next = newNode;

}

void delete\_first\_node(){

Node\* temp = head;

head = head->next;

free(temp);

}

void delete\_last\_node(){

Node\* temp = head;

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

temp = temp->next;

}

Node \*temp2 = temp->next;

temp->next = NULL;

free(temp2);

}

void delete\_middle\_node(int position){

Node\* temp = head;

int i = 0;

while(i != position - 1){

temp = temp->next;

i++;

}

Node\* temp2 = temp->next;

temp->next = temp2->next;

free(temp2);

}

void delete\_given\_value\_node(int data){

Node\* temp = head, \*temp2;

while((temp->data != data) && (temp->next != NULL)){

temp2 = temp;

temp = temp->next;

}

if(temp->data == data){

temp2->next = temp->next;

free(temp);

}

}

void singlyLinkedListTraversal(){

Node\* temp = head;

cout << "Singly Linked List Traversal: ";

while(temp != NULL){

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

temp = temp->next;

}

cout << endl;

}

void circularLLConversion(){

Node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = head;

}

void circularLinkedListTraversal(){

Node \*temp = head;

cout << "Circular Linked List Traversal: ";

do{

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

temp = temp->next;

}while(temp != head);

cout << endl;

}

void CCinsert\_at\_head(int data){

Node \*newNode = (struct Node\*)malloc (sizeof(struct Node));

newNode->data = data;

newNode->next = head;

Node \*ptr = head;

while(ptr->next != head){

ptr = ptr->next;

}

ptr->next = newNode;

head = newNode;

}

void CCinsert\_at\_end(int data){

Node \*newNode = (struct Node\*)malloc (sizeof(struct Node));

newNode->data = data;

newNode->next = head;

Node \*ptr = head;

while(ptr->next != head){

ptr = ptr->next;

}

ptr->next = newNode;

}

void CCinsert\_in\_between(int data, int position){

Node \*newNode = (struct Node\*) malloc (sizeof(struct Node));

newNode->data = data;

Node\* temp = head;

for(int i = 0; i < position-1; i++){

temp = temp->next;

}

newNode->next = temp->next;

temp->next = newNode;

}

void CCinsert\_after\_node(int data, struct Node\* node){

Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = node->next;

node->next = newNode;

}

void CCdelete\_first\_node(){

Node\* temp = head;

while(temp->next != head){

temp = temp->next;

}

Node \*temp2 = head;

temp->next = temp2->next;

head = temp->next;

free(temp2);

}

void CCdelete\_end\_node(){

Node \*temp = head, \*temp2;

while(temp->next != head){

temp2 = temp;

temp = temp->next;

}

temp2->next = head;

free(temp);

}

void CCdelete\_in\_between(int position){

Node \*temp = head;

int i = 0;

while(i != position-1){

temp = temp->next;

i++;

}

Node\* temp2 = temp->next;

temp->next = temp2->next;

free(temp2);

}

LinkedList CCdivide\_into\_2\_halves(){

Node \*temp = head;

int count = 0;

while(temp->next != head){

temp = temp->next;

count++;

}

temp->next = NULL;

temp = head;

int i = 0;

while(i != (count/2)){

temp = temp->next;

i++;

}

Node\* temp2 = temp->next;

temp->next = head;

LinkedList newCC(temp2->data);

newCC.head = temp2;

while(temp2->next != NULL){

temp2 = temp2->next;

}

temp2->next = newCC.head;

return newCC;

}

};

int main(){

LinkedList L1(12);

L1.insert\_at\_end(13);

L1.insert\_at\_end(14);

L1.insert\_at\_end(15);

L1.insert\_at\_end(16);

L1.insert\_at\_end(17);

L1.insert\_at\_end(18);

L1.insert\_at\_end(19);

L1.singlyLinkedListTraversal();

cout << "Circular Linked list Conversion...\n";

L1.circularLLConversion();

cout << "Circular Linked list Traversal:\n\t";

L1.circularLinkedListTraversal();

L1.CCinsert\_at\_head(11);

cout << "CC insert at head (11):\n\t";

L1.circularLinkedListTraversal();

L1.CCinsert\_at\_end(20);

cout << "CC insert at end (20):\n\t";

L1.circularLinkedListTraversal();

L1.CCinsert\_in\_between(22, 2);

cout << "CC insert in between (22, 2):\n\t";

L1.circularLinkedListTraversal();

L1.CCdelete\_in\_between(2); // index starts from 0, will delete 3rd element

cout << "CC delete in between (2): // index starts from 0, this function will delete 3rd element (index=2)\n\t";

L1.circularLinkedListTraversal();

LinkedList L2 = L1.CCdivide\_into\_2\_halves();

cout << "LinkedList L2 = L1.CCdivide\_into\_2\_halves();\n";

cout << "L1 List: ";

L1.circularLinkedListTraversal();

cout << "L2 List: ";

L2.circularLinkedListTraversal();

return 0;

}

OUTPUT:

