LAB # 03 - ASSIGNMENT

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**

/\*

DSAA LAB 3

11 NOV 2023

1. Implement a singly linked list class with the following functions:

a) Insert a node at head

b) Insert  a node at tail/end/back

c) Insert a node at any position

d) Delete a node by value

e) Delete head

f) Delete tail

g) Delete a node at any position.

\*/

#include<iostream>

#include<cstdlib>

using namespace std;

struct Node{

    int data;

    struct Node\* next;

};

class LinkedList{

    struct Node\* head;

    public:

    LinkedList(int newdata){

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

        newNode->data = newdata;

        newNode->next = NULL;

        head = newNode;

    }

    void insert\_at\_head(int newdata){

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

        newNode->data = newdata;

        newNode->next = head;

        head = newNode;

    }

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

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

        newNode->data = newdata;

        Node \*temp = head;

        int i = 0;

        while(i != position-1){

            temp = temp->next;

            i++;

        }

        newNode->next = temp->next;

        temp->next = newNode;

    }

    void insert\_in\_end(int newdata){

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

        newNode->data = newdata;

        newNode->next = NULL;

        Node\* temp = head;

        while(temp->next != NULL){

            temp = temp->next;

        }

        temp->next = newNode;

    }

    void insert\_after\_node(Node\* prevNode, int newdata){

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

        newNode->data = newdata;

        newNode->next = prevNode->next;

        prevNode->next = newNode;

    }

    void delete\_First\_Node(){

        Node \*temp = head;

        head = head->next;

        free(temp);

    }

    void delete\_End\_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){ // position is index, will not run on index = 0, as it will be the starting one

        Node \*temp = head;

        int i = 1;

        while(i != position-1){

            temp = temp->next;

            i++;

        }

        Node \*temp2 = temp->next;

        temp->next = temp2->next;

        free(temp2);

    }

    void delete\_Node\_With\_Given\_Value\_Which\_Comes\_FIrst(int value){

        Node \*temp = head, \*temp2 = head->next;

        while(temp2->data != value){        // loop for traversing

            if(temp->data == value){        // if given data is in 0 index i.e. head

                delete\_First\_Node();        // first index, call function delete first node

                free(temp);                 // free mempry

                return;                     // end function

            }

            temp = temp->next;

            temp2 = temp2->next;

        }

        if(temp2->next == NULL){            // if given data is in last index

            temp->next = NULL;              // same as delete\_end\_node function

            free(temp2);                    // free memory

            return;                         // end function

        }

        temp->next = temp2->next;

        free(temp2);

    }

    struct Node\* getNode(int index){

        Node \*temp = head;

        for(int i = 0; i < index; i++)

        temp = temp->next;

        return temp;

    }

    void display(){

        Node\* temp = head;

        cout << "Linked List: ";

        while(temp != NULL){

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

            temp = temp->next;

        }

        cout << endl;

    }

};

int main(){

    LinkedList L1(2);

    cout << "LinkedList L1(2);" << endl << "\t";

    L1.display();

    L1.insert\_in\_end(5);

    cout << "L1.insert\_in\_end(5);" << endl << "\t";

    L1.display();

    L1.insert\_at\_head(1);

    cout << "L1.insert\_at\_head(1);" << endl << "\t";

    L1.display();

    L1.insert\_in\_between(3, 2);

    cout << "L1.insert\_in\_between(3, 2); 3 after position 2" << endl << "\t";

    L1.display();

    L1.insert\_in\_end(6);

    cout << "L1.insert\_in\_end(6);" << endl << "\t";

    L1.display();

    L1.insert\_in\_end(7);

    cout << "L1.insert\_in\_end(7);" << endl << "\t";

    L1.display();

    L1.insert\_in\_end(8);

    cout << "L1.insert\_in\_end(8);" << endl << "\t";

    L1.display();

    L1.insert\_in\_end(9);

    cout << "L1.insert\_in\_end(9);" << endl << "\t";

    L1.display();

    L1.delete\_First\_Node();

    cout << "L1.delete\_First\_Node();" << endl << "\t";

    L1.display();

    L1.delete\_End\_Node();

    cout << "L1.delete\_End\_Node();" << endl << "\t";

    L1.display();

    L1.delete\_Middle\_Node(3);

    cout << "L1.delete\_Middle\_Node(3);" << endl << "\t";

    L1.display();

    L1.delete\_Node\_With\_Given\_Value\_Which\_Comes\_FIrst(7);

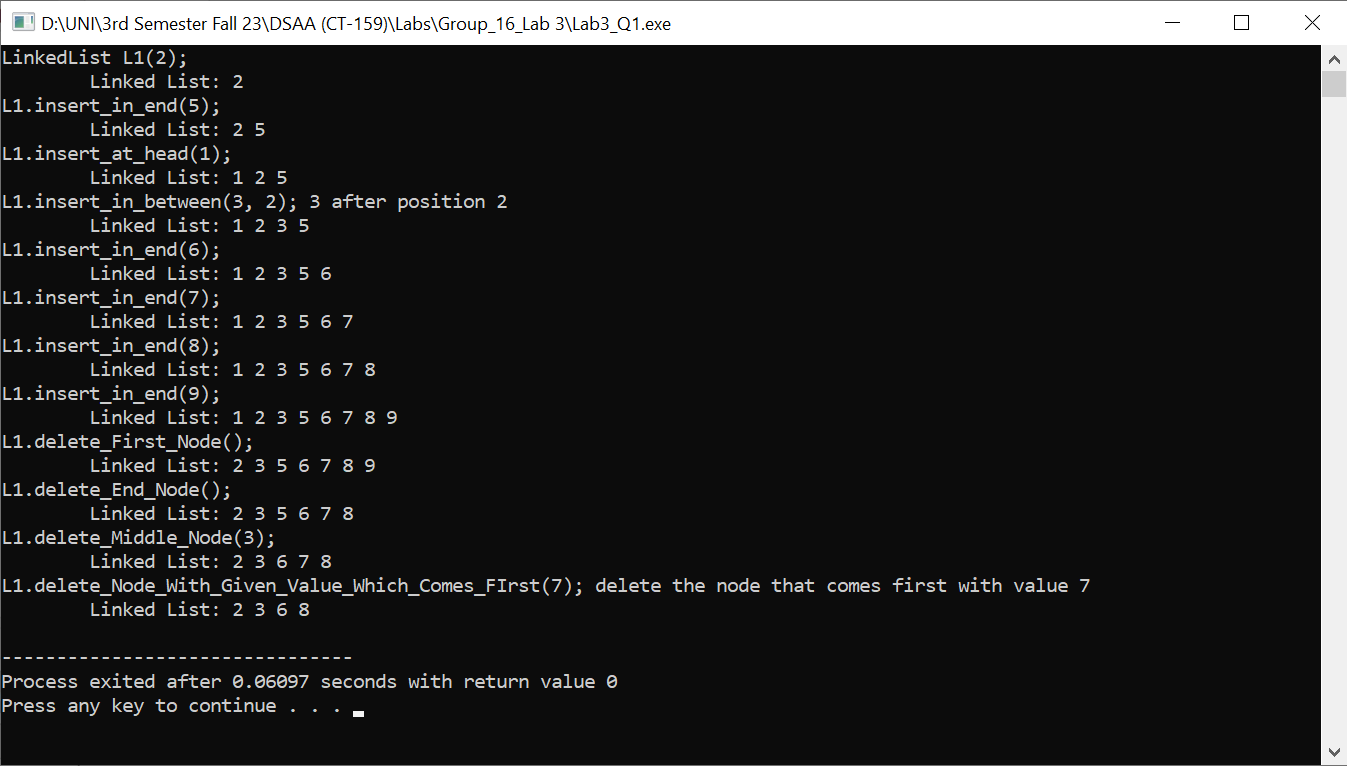
    cout << "L1.delete\_Node\_With\_Given\_Value\_Which\_Comes\_FIrst(7); delete the node that comes first with value 7" << endl << "\t";

    L1.display();

    return 0;

}

OUTPUT:



/\*

DSAA LAB 3

11 NOV 2023

2. Solve the following problem using a Singly Linked List. Given a singly linked list of

characters, write a function to make word out of given letters in the list. Test Case:

Input:C->S->A->R->B->B->E->L->NULL,

Output:S->C->R->A->B->B->L->E->NULL

\*/

#include <iostream>

#include <string.h>

#include<cstdlib>

using namespace std;

void swap(char \*a, char \*b)

{ // swapping of variables, also can be done with that method temp = a; a = b; b = temp;

    \*a = \*a + \*b;

    \*b = \*a - \*b;

    \*a = \*a - \*b;

}

struct Node

{

    char data;

    struct Node \*next;

};

class LinkedList

{

    struct Node \*head;

public:

    LinkedList(char newdata)

    {

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

        newNode->data = newdata;

        newNode->next = NULL;

        head = newNode;

    }

    void insert\_at\_head(char newdata)

    {

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

        newNode->data = newdata;

        newNode->next = head;

        head = newNode;

    }

    void insert\_in\_between(char newdata, int position)

    {

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

        newNode->data = newdata;

        Node \*temp = head;

        int i = 0;

        while (i != position - 1)

        {

            temp = temp->next;

            i++;

        }

        newNode->next = temp->next;

        temp->next = newNode;

    }

    void insert\_in\_end(char newdata)

    {

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

        newNode->data = newdata;

        newNode->next = NULL;

        Node \*temp = head;

        while (temp->next != NULL)

        {

            temp = temp->next;

        }

        temp->next = newNode;

    }

    void insert\_after\_node(Node \*prevNode, char newdata)

    {

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

        newNode->data = newdata;

        newNode->next = prevNode->next;

        prevNode->next = newNode;

    }

    void delete\_First\_Node()

    {

        Node \*temp = head;

        head = head->next;

        free(temp);

    }

    void delete\_End\_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)

    { // position is index, will not run on index = 0, as it will be the starting one

        Node \*temp = head;

        int i = 1;

        while (i != position - 1)

        {

            temp = temp->next;

            i++;

        }

        Node \*temp2 = temp->next;

        temp->next = temp2->next;

        free(temp2);

    }

    void delete\_Node\_With\_Given\_Value\_Which\_Comes\_FIrst(char value)

    {

        Node \*temp = head, \*temp2 = head->next;

        while (temp2->data != value)

        { // loop for traversing

            if (temp->data == value)

            {                        // if given data is in 0 index i.e. head

                delete\_First\_Node(); // first index, call function delete first node

                free(temp);          // free mempry

                return;              // end function

            }

            temp = temp->next;

            temp2 = temp2->next;

        }

        if (temp2->next == NULL)

        {                      // if given data is in last index

            temp->next = NULL; // same as delete\_end\_node function

            free(temp2);       // free memory

            return;            // end function

        }

        temp->next = temp2->next;

        free(temp2);

    }

    void sort\_word(int size)

    {

        if (size == 4)

        {

            char targettedArray[size] = "GAZA";

            sort\_part\_2(head, targettedArray, size);

        }

        else if (size == 5)

        {

            char targettedArray[size] = "SYRIA";

            sort\_part\_2(head, targettedArray, size);

        }

        else if (size == 6)

        {

            char targettedArray[size] = "PYTHON";

            sort\_part\_2(head, targettedArray, size);

        }

        else if (size == 7)

        {

            char targettedArray[size] = "LEBANON";

            sort\_part\_2(head, targettedArray, size);

        }

        else if (size == 8)

        {

            char targettedArray[size] = "SCRABBLE";

            sort\_part\_2(head, targettedArray, size);

        }

        else if (size == 9)

        {

            char targettedArray[size] = "PALESTINE";

            sort\_part\_2(head, targettedArray, size);

        }

        else if (size == 10)

        {

            char targettedArray[size] = "STRUCTURES";

            sort\_part\_2(head, targettedArray, size);

        }

        else if (size == 12)

        {

            char targettedArray[size] = "APPLICATIONS";

            sort\_part\_2(head, targettedArray, size);

        }

        else if (size == 13)

        {

            char targettedArray[size] = "INTERNATIONAL";

            sort\_part\_2(head, targettedArray, size);

        }

        else if (size == 15)

        {

            char targettedArray[size] = "DEV\_C\_PLUS\_PLUS";

            sort\_part\_2(head, targettedArray, size);

        }

        else if (size == 18)

        {

            char targettedArray[size] = "VISUAL\_STUDIO\_CODE";

            sort\_part\_2(head, targettedArray, size);

        }

        else

        {

            cout << "Invalid Word." << endl;

            return;

        }

    }

    void sort\_part\_2(struct Node \*h, char targetedArray[], int size)

    {

        char b;

        for (int k = 0; k < size; k++)

        {

            Node \*temp = head;

            for (int i = 0; i < size; i++)

            {

                b = targetedArray[i];

                Node \*temp2 = head;

                for (int j = 0; j < size; j++)

                {

                    if (temp2->data == b)

                    {

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

                        break;

                    }

                    temp2 = temp2->next;

                }

                temp = temp->next;

            }

        }

    }

    struct Node \*getNode(int index)

    {

        Node \*temp = head;

        for (int i = 0; i < index; i++)

            temp = temp->next;

        return temp;

    }

    void display()

    {

        Node \*temp = head;

        cout << "Linked List: ";

        while (temp != NULL)

        {

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

            temp = temp->next;

        }

        if (temp == NULL)

            cout << "NULL";

        cout << endl;

    }

    void swapNodes(struct Node \*a, struct Node \*b)

    {

        a->data = a->data + b->data;

        b->data = a->data - b->data;

        a->data = a->data - b->data;

    }

};

int main()

{

    cout << "Enter No: ";

    int no;

    cin >> no;

    char a[no];

    cout << "Enter characters:";

    for (int i = 0; i < no; i++)

        cin >> a[i];

    LinkedList L1(a[0]);

    for (int i = 1; i < no; i++)

        L1.insert\_in\_end(a[i]);

    cout << "Input: ";

    L1.display();

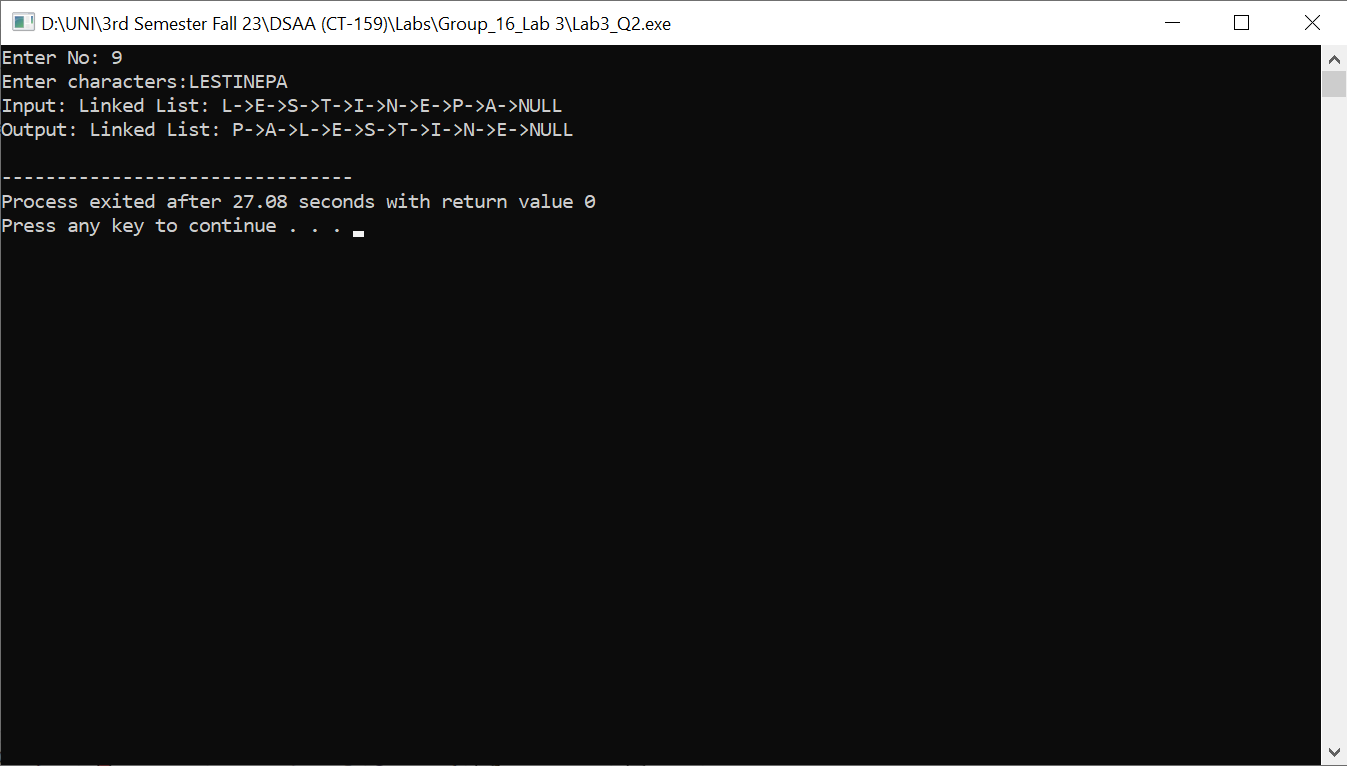
    L1.sort\_word(no);

    cout << "Output: ";

    L1.display();

}

OUTPUT:



/\*

DSAA LAB 3

11 NOV 2023

3. Use the class of SLL created by you during the lab task 1. Do the following:

a) Reverse the linked list

b) Sort the contents of linked list

c) Find the duplicates in the linked list

\*/

#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 \*next;

};

class LinkedList

{

    struct Node \*head;

public:

    LinkedList(int newdata)

    {

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

        newNode->data = newdata;

        newNode->next = NULL;

        head = newNode;

    }

    void insert\_at\_head(int newdata)

    {

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

        newNode->data = newdata;

        newNode->next = head;

        head = newNode;

    }

    void insert\_in\_between(int newdata, int position)

    {

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

        newNode->data = newdata;

        Node \*temp = head;

        int i = 0;

        while (i != position - 1)

        {

            temp = temp->next;

            i++;

        }

        newNode->next = temp->next;

        temp->next = newNode;

    }

    void insert\_in\_end(int newdata)

    {

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

        newNode->data = newdata;

        newNode->next = NULL;

        Node \*temp = head;

        while (temp->next != NULL)

        {

            temp = temp->next;

        }

        temp->next = newNode;

    }

    void insert\_after\_node(Node \*prevNode, int newdata)

    {

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

        newNode->data = newdata;

        newNode->next = prevNode->next;

        prevNode->next = newNode;

    }

    void delete\_First\_Node()

    {

        Node \*temp = head;

        head = head->next;

        free(temp);

    }

    void delete\_End\_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)

    { // position is index, will not run on index = 0, as it will be the starting one

        Node \*temp = head;

        int i = 1;

        while (i != position - 1)

        {

            temp = temp->next;

            i++;

        }

        Node \*temp2 = temp->next;

        temp->next = temp2->next;

        free(temp2);

    }

    void delete\_Node\_With\_Given\_Value\_Which\_Comes\_FIrst(int value)

    {

        Node \*temp = head, \*temp2 = head->next;

        while (temp2->data != value)

        { // loop for traversing

            if (temp->data == value)

            {                        // if given data is in 0 index i.e. head

                delete\_First\_Node(); // first index, call function delete first node

                free(temp);          // free mempry

                return;              // end function

            }

            temp = temp->next;

            temp2 = temp2->next;

        }

        if (temp2->next == NULL)

        {                      // if given data is in last index

            temp->next = NULL; // same as delete\_end\_node function

            free(temp2);       // free memory

            return;            // end function

        }

        temp->next = temp2->next;

        free(temp2);

    }

    struct Node \*getNode(int index)

    {

        Node \*temp = head;

        for (int i = 0; i < index; i++)

            temp = temp->next;

        return temp;

    }

    void display()

    {

        Node \*temp = head;

        cout << "Linked List: ";

        while (temp != NULL)

        {

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

            temp = temp->next;

        }

        cout << endl;

    }

    void reverseList()

    {

        Node \*next = NULL, \*prev = NULL;

        while (head != NULL)

        {

            next = head->next;

            head->next = prev;

            prev = head;

            head = next;

        }

        head = prev;

    }

    void sortList()

    {

        Node \*current = head, \*next = head->next, \*count = head;

        while (count != NULL)

        {

            next = head->next;

            current = head;

            while (next != NULL)

            {

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

                {

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

                }

                current = current->next;

                next = next->next;

            }

            count = count->next;

        }

    }

    void findDuplicates()

    {

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

        int count = 0, i;

        while(temp != NULL){

            temp = temp->next;

            count++;

        }

        int duplicates[count];

        for(i = 0; i < count; i++)

        duplicates[i] = 0;

        i = 0;

        temp = head;

        while(temp != NULL){

            temp2 = head;

            while(temp2 != NULL){

                if((temp2->data == temp->data) && (temp != temp2)){

                    duplicates[i]++;

                }

                temp2 = temp2->next;

            }

            temp = temp->next;

            i++;

        }

        temp = head;

        for(int j = 0; j < count; j++){

            if(duplicates[j] != 0){

                cout << "THer is/are " << duplicates[j] << " duplicate of value " << temp->data << endl;

            }

            temp = temp->next;

        }

    }

};

int main()

{

    LinkedList L1(10);

    L1.insert\_in\_end(2);

    L1.insert\_in\_end(5);

    L1.insert\_in\_end(3);

    L1.insert\_in\_end(4);

    L1.insert\_in\_end(4);

    L1.insert\_in\_end(1);

    cout << "Input: ";

    L1.display();

    L1.reverseList();

    cout << "Reverse List: ";

    L1.display();

    L1.sortList();

    cout << "Sort List: ";

    L1.display();

    cout << "Find Duplicates:" << endl;

    L1.findDuplicates();

    return 0;

}

OUTPUT:

