Data Structures

NICOLETA RADU C++

# Problema 1

#include <iostream>

#include <fstream>

#include <tgmath.h>

using namespace std;

// A linked list node

class Node

{

public:

float real, imaginar;

Node\* next;

};

/\* Given a reference (pointer to pointer)

to the head of a list and an int, inserts

a new node on the front of the list. \*/

void push(Node\*\* head\_ref, float new\_data\_1, float new\_data\_2)

{

/\* 1. allocate node \*/

Node\* new\_node = new Node();

/\* 2. put in the data \*/

new\_node->real = new\_data\_1;

new\_node->imaginar = new\_data\_2;

/\* 3. Make next of new node as head \*/

new\_node->next = (\*head\_ref);

/\* 4. move the head to point to the new node \*/

(\*head\_ref) = new\_node;

}

/\* Given a node prev\_node, insert a new node after the given

prev\_node \*/

void insertAfter(Node\* prev\_node, float new\_data\_1, float new\_data\_2)

{

/\*1. check if the given prev\_node is NULL \*/

if (prev\_node == NULL)

{

cout << "The given previous node cannot be NULL";

return;

}

/\* 2. allocate new node \*/

Node\* new\_node = new Node();

/\* 3. put in the data \*/

new\_node->real = new\_data\_1;

new\_node->imaginar = new\_data\_2;

/\* 4. Make next of new node as next of prev\_node \*/

new\_node->next = prev\_node->next;

/\* 5. move the next of prev\_node as new\_node \*/

prev\_node->next = new\_node;

}

/\* Given a reference (pointer to pointer) to the head

of a list and an int, appends a new node at the end \*/

void append(Node\*\* head\_ref, float new\_data\_1, float new\_data\_2)

{

/\* 1. allocate node \*/

Node\* new\_node = new Node();

Node\* last = \*head\_ref; /\* used in step 5\*/

/\* 2. put in the data \*/

new\_node->real = new\_data\_1;

new\_node->imaginar = new\_data\_2;

/\* 3. This new node is going to be

the last node, so make next of

it as NULL\*/

new\_node->next = NULL;

/\* 4. If the Linked List is empty,

then make the new node as head \*/

if (\*head\_ref == NULL)

{

\*head\_ref = new\_node;

return;

}

/\* 5. Else traverse till the last node \*/

while (last->next != NULL)

{

last = last->next;

}

/\* 6. Change the next of last node \*/

last->next = new\_node;

return;

}

// This function prints contents of

// linked list starting from head

void printList(Node\* node)

{

while (node != NULL)

{

cout << " " << node->real << "," << node->imaginar;

node = node->next;

}

}

float modul(Node\* node)

{

float rezultat;

rezultat = sqrt(pow(static\_cast<double>(node->real),2) + pow(static\_cast<double>(node->imaginar), 2));

return rezultat;

}

/\* Driver code\*/

int main()

{

/\* Start with the empty list \*/

Node\* head = NULL;

// Insert 6. So linked list becomes 6->NULL

append(&head, 64,2);

// Insert 7 at the beginning.

// So linked list becomes 7->6->NULL

push(&head, 7,8);

// Insert 1 at the beginning.

// So linked list becomes 1->7->6->NULL

push(&head, 12,4);

// Insert 4 at the end. So

// linked list becomes 1->7->6->4->NULL

append(&head, 4,10);

// Insert 8, after 7. So linked

// list becomes 1->7->8->6->4->NULL

insertAfter(head->next, 8,1);

cout << "List 1" << endl;

printList(head);

cout << endl;

cout << "Modul: " << modul(head) << endl;

Node\* head\_2 = NULL;

float userInput{0};

while (head != NULL)

{

cout << "Introduceti un numar: ";

cin >> userInput;

if (modul(head) == userInput)

{

cout << "Conditie indeplinita" << endl;

push(&head\_2, head->real, head->imaginar);

}

head = head->next;

}

cout << endl;

cout << "List 2" << endl;

printList(head\_2);

/\*

Node\* head = new Node();

Node\* second = new Node();

Node\* third = new Node();

head->data = 99;

head->next = second;

second->data = 100;

second->next = third;

third->data = 101;

third->next = nullptr;

insertNewHead(&head, -99);

printLinkedList(head);

std::cout << std::endl;

insertNewHead(&head, -88);

printLinkedList(head);

std::cout << std::endl;

deleteHead(&head);

printLinkedList(head);

std::cout << std::endl;

removeLastNode(head);

printLinkedList(head);

std::cout << std::endl;

\*/

return 0;

}

# Problema 2

# Problema 3

# Problema 4

# Problema 5