Liste Dublu Inlănțuite

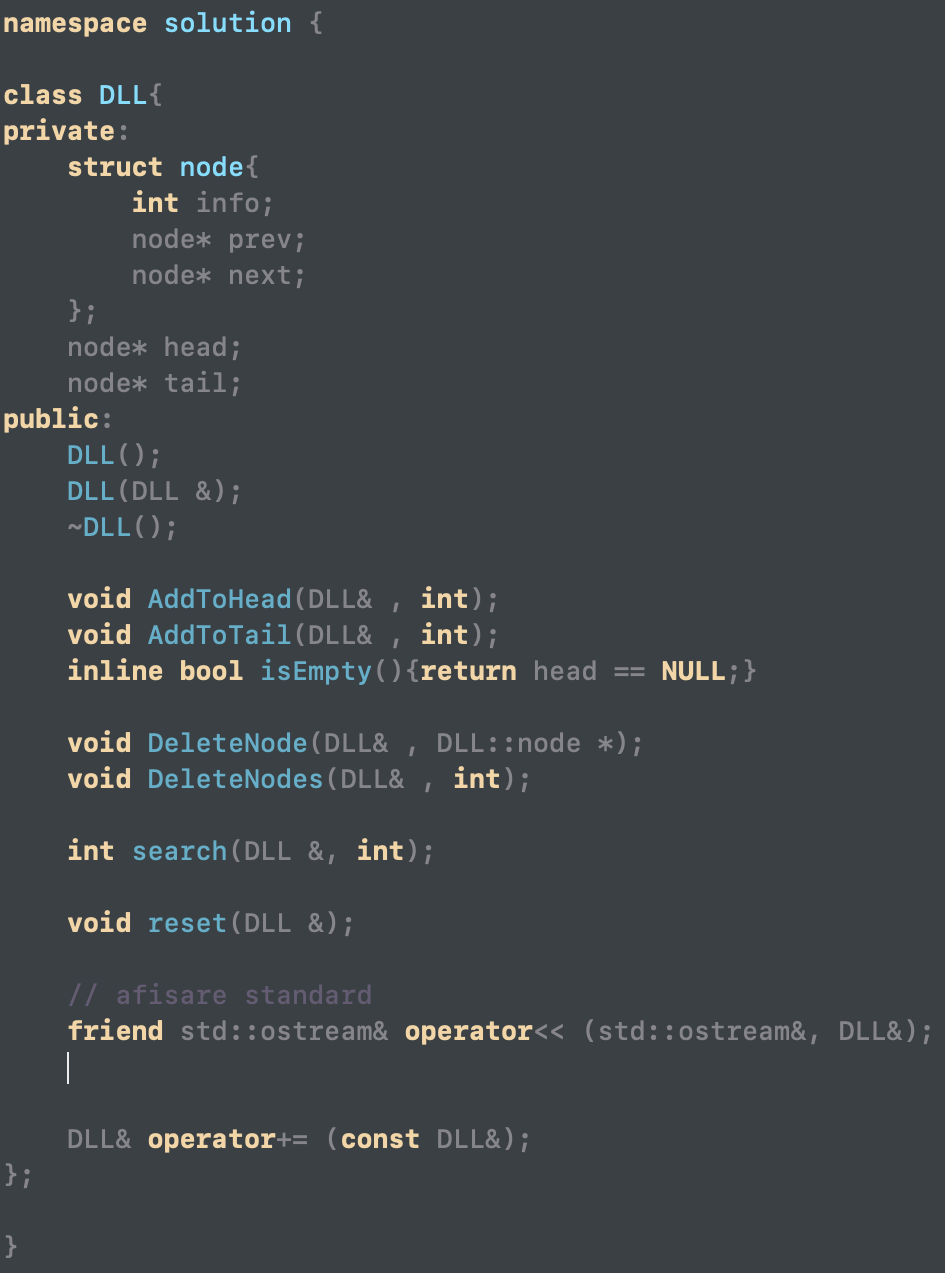
Proiect Poo - Programare Orientată Obiect

## Proiect realizat de Ștefan Munteanu,

## Clasa a XI - A, LIIS

## Profesor coordonator: Oana Butnărașu

Argument

Sa se conceapă o clasă adecvată pentru implementarea structurii dinamice de date de tip listă liniară dublu înlănţuită cu informații de tip întreg. Clasa va conţine urmatoarele metode: constructor, constructor de copiere, destructor, verificarea listei vide (inline), căutarea unui nod de cheie dată, adăugarea unui nod nou la începutul listei,  adăugarea unui nod nou la sfârşitul listei,  ștergerea nodurilor cu o anumită informație, operatorul += pentru concatenarea a două liste, operatori supraîncărcaţi pentru citire/scriere.

Prezentarea clasei (DLL.hpp)

## Clasa DLL conține o parte privată unde am definiția listei dublu inlănțuite si 2 noduri head si tail, reprezentând adresele primului nod, respectiv ultimului nod. In partea publică se gasesc un constructor, constructor de copiere, distructor, operatorii de supraîncărcare pentru afișare, concatenare si functiile care îndeplinesc operațiile cu liste dublu inlănțuite.

## Structura fișierului main.cpp:



Exemple de implementări ale unor metode din clasa DLL.hpp:

DLL::DLL()

{

head = tail = **NULL**;

}

DLL::DLL(DLL& p)

{

head = p.head;

tail = p.tail;

}

DLL::~DLL()

{

DLL::node \* q = **new** DLL::node;

**for**(q = head; q != tail; q = q -> next)

{

head = q -> next;

**delete** q;

}

q = **new** DLL::node;

tail = q -> next;

**delete** q;

}

**void** DLL::AddToHead(DLL& p, **int** x)

{

**if**(p.head == **NULL**)

{

p.head = **new** DLL::node;

p.head -> info = x;

p.head -> prev = **NULL**;

p.head -> next = **NULL**;

p.tail = p.head;

}

**else**

{

DLL::node\* q = **new** DLL::node;

q -> info = x;

q -> prev = **NULL**;

q -> next = p.head;

p.head -> prev = q;

p.head = q;

}

}

**void** DLL::AddToTail(DLL& p, **int** x)

{

DLL::node \*q = **new** DLL::node;

q -> info = x;

q -> next = **NULL**;

q -> prev = **NULL**;

**if**(p.head == **NULL**)

{

p.head = q;

p.tail = q;

}

**else** {

q -> prev = p.tail;

p.tail -> next = q;

p.tail = q;

}

}

**void** DLL::DeleteNode(DLL& p, DLL::node\* temp)

{

**if**(p.head == **NULL** || temp == **NULL**)

{**return**;}

**if**(p.head == temp)

{

p.head = temp -> next;

}

**if**(temp -> prev != **NULL**)

{ temp -> prev -> next = temp -> next; }

**if**(temp -> next != **NULL**)

{ temp -> next -> prev = temp -> prev; }

**delete** temp;

**return**;}

**void** DLL::DeleteNodes(DLL& p, **int** x)

{

**if**(p.head == **NULL**)

**return**;

DLL::node\* temp = **new** DLL::node;

temp = p.head;

DLL::node\* urm = **new** DLL::node;

**while**(temp)

{

**if**(temp -> info == x)

{

urm = temp -> next;

DeleteNode(p, temp);

temp = urm;

}

**else**

temp = temp -> next;

}

**return**;

}

**int** DLL::search(DLL & p , **int** x)

{

**int** poz = 0;

DLL::node \*q = **new** DLL::node;

q = p.head;

**while**(q)

{

++poz;

**if**(q -> info == x)

**return** poz;

q = q -> next;

}

**return** -1;

}

**void** DLL::reset(DLL & p)

{

p.head = **NULL**;

p.tail = **NULL**;}

std::ostream& **operator**<< (std::ostream& out, DLL& p)

{

DLL::node\* q = **new** DLL::node;

**for**(q = p.head; q; q = q -> next)

out << q -> info << ' ';

**return** out;

}

DLL& DLL::**operator** += (**const** DLL& p)

{

DLL::node\* temp = **new** DLL::node;

temp = p.head;

**while**(temp)

{

AddToTail(\***this**, temp -> info);

temp = temp -> next;

}

**return** \***this**;

}}

Concluzie

Acest proiect m-a ajutat să aprofundez tainele programării orientate obiect și a modului de gestionare a alocării dinamice a memoriei în contextul-suport.

Webografie:**1)** Lucrarea mea