TEMA 2: ESTRUCTURES DE DADES DINÀMIQUES

Actualizado: 2024-04-30

Incluye ejercicios: Sesiones 12, 13, 14, 15

github.com/jkrahl/mp

Índice

Sessio 12 - Exercici classe (Data límit: 16/04)

passatger.h

```
#pragma once
#include <string>
#ifndef PASSATGER_H
#define PASSATGER_H
using namespace std;

class Passatger {
public:
    Passatger() {}
    Passatger(const string& dni, const string& nom) : m_dni(dni), m_nom(nom) {}
    void setDni(const string& dni) { m_dni = dni; }
    void setNom(const string& nom) { m_nom = nom; }
    const string& getDni() const { return m_dni; }
    const string& getNom() const { return m_nom; }
private:
    string m_dni;
    string m_nom;
};
#endif
```

seient.h

```
#pragma once
#include <string>
#include "passatger.h"
#ifndef SEIENT_H
#define SEIENT_H
```

```
class Seient {
public:
    Seient();
    Seient(const string& codi);
    ~Seient();

    void setCodi(const string& codi) { m_codi = codi; }
    void setPassatger(Passatger* p) { m_passatger = p; }

    bool assignaPassatger(const string& dni, const string& nom);
    bool eliminaPassatger();

    const string& getCodi() const { return m_codi; }
    Passatger* getPassatger() { return m_passatger; }
private:
    string m_codi;
    Passatger* m_passatger;
};
#endif
```

seient.cpp

```
#include "seient.h"

Seient::Seient() {
    m_passatger = nullptr;
    m_codi = "";
}

Seient::Seient(const string& codi) {
    m_passatger = nullptr;
    m_codi = codi;
}

Seient::~Seient() {
    m_codi = "";
    delete m_passatger;
}

bool Seient::assignaPassatger(const string& dni, const string& nom) {
    bool vacio = m_passatger == nullptr;
    if (vacio) {
        m_passatger = new Passatger(dni, nom);
    }
    return vacio;
}

bool Seient::eliminaPassatger() {
    bool reservado = m_passatger != nullptr;
    if (reservado) {
        delete m_passatger;
```

```
            m_passatger = nullptr;
        }
        return reservado;
    }
```

Sessio 13 - Exercici classe (Data límit: 23/04)

Estudiant.h

```
#pragma once
#include <string>
using namespace std;

class Estudiant {
public:
    Estudiant() : m_NIU(""), m_nom(""), m_nAssignatures(0), m_maxAssignatures(0),
m_assignatures(nullptr) {};
    Estudiant(const string& niu, const string& nom, int nAssignatures);
    ~Estudiant();

    string getNiu() const { return m_NIU; }
    string getNom() const { return m_nom; }
    void setNiu(const string& niu) { m_NIU = niu; }
    void setNom(const string& nom) { m_nom = nom; }

    void afegeixAssignatura(const string& assignatura);
    int getNAssignatures() const { return m_nAssignatures; }
    string getAssignatura(int posicio) const;

    void inicialitza(const string& niu, const string& nom, int nAssignatures);
    void allibera();
private:
    string m_nom;
    string m_NIU;
    string* m_assignatures;
    int m_maxAssignatures;
    int m_nAssignatures;
};
```

Estudiant.cpp

```
#include "Estudiant.h"

Estudiant::Estudiant(const string& niu, const string& nom, int nAssignatures){
    inicialitza(niu, nom, nAssignatures);
}

Estudiant::~Estudiant(){
    allibera();
```

```
}

void Estudiant::inicialitza(const string& niu, const string& nom, int
nAssignatures){
    m_NIU = niu;
    m_nom = nom;
    m_nAssignatures = 0;
    m_assignatures = new string[nAssignatures];
    m_maxAssignatures = nAssignatures;
}

void Estudiant::allibera() {
    if (m_assignatures != nullptr)
    {
        delete[] m_assignatures;
    }
}

void Estudiant::afegeixAssignatura(const string& assignatura) {
    m_assignatures[m_nAssignatures++] = assignatura;
}

string Estudiant::getAssignatura(int posicio) const {
    if (m_assignatures != nullptr)
        return m_assignatures[posicio];
    else return "";
}
```

Titulacio.h

```
#pragma once
#include "Estudiant.h"
const int MAX_ESTUDIANTS = 300;

class Titulacio {
public:
    Titulacio() : m_nom(""), m_nEstudiants(0), m_nMaxAssignatures(0) {}
    Titulacio(const string& nom, int maxAssignatures) : m_nom(nom),
m_nEstudiants(0), m_nMaxAssignatures(maxAssignatures) {}
    void afegeixEstudiant(const string& niu, const string& nom);
    bool eliminaEstudiant(const string& niu);
    bool consultaEstudiant(const string& niu, Estudiant& e);
private:
    string m_nom;
    int m_nMaxAssignatures;
    Estudiant m_estudiants[MAX_ESTUDIANTS];
    int m_nEstudiants;
};
```

Titulacio.cpp

```cpp
#include "Titulacio.h"

void Titulacio::afegeixEstudiant(const string& niu, const string& nom){
    m_estudiants[m_nEstudiants++].inicialitza(niu, nom, m_nMaxAssignatures);
}

bool Titulacio::eliminaEstudiant(const string& niu){
    bool trobat = false;
    int i = 0;
    while ((i < m_nEstudiants) && !trobat)
    {
        if (niu == m_estudiants[i].getNiu())
            trobat = true;
        else
            i++;
    }
    if (trobat)
    {
        // Allibera estudiant
        m_estudiants[i].allibera();
        // Movem tots els apuntador per evitar tenir un buit a l'array.
        for (int j = i; j < (m_nEstudiants - 1); j++)
            m_estudiants[j] = m_estudiants[j + 1];
        m_nEstudiants--;
    }
    return trobat;
}

bool Titulacio::consultaEstudiant(const string& niu, Estudiant& e){
    bool trobat = false;
    int i = 0;
    while ((i < m_nEstudiants) && !trobat)
    {
        if (niu == m_estudiants[i].getNiu())
            trobat = true;
        else
            i++;
    }
    if (trobat)
        e = m_estudiants[i];
    return trobat;
}
```

Sessio 14 - Exercici classe (Data límit: 26/04)

matriu.cpp

```cpp
#include "matriu.h"
#include <fstream>
```

```
int** creaMatriu(int nFiles, int nColumnes) {
    int** m = new int*[nFiles];
    for (int i = 0; i < nFiles; i++)
        m[i] = new int[nColumnes];
    return m;
}

void destrueixMatriu(int** m, int nFiles) {
    for (int i = 0; i < nFiles; i++)
        delete[] m[i];
    delete[] m;
}

void llegeixMatriu(int** m, int nFiles, int nColumnes, const string& nomFitxer) {
    ifstream fitxer;
    fitxer.open(nomFitxer);
    for (int i = 0; i < nFiles; i++)
        for (int j = 0; j < nColumnes; j++)
            fitxer >> m[i][j];
    fitxer.close();
}

void sumaMatrius(int** m1, int** m2, int** suma, int nFiles, int nColumnes) {
    for (int i = 0; i < nFiles; i++)
        for (int j = 0; j < nColumnes; j++)
            suma[i][j] = m1[i][j] + m2[i][j];
}
```

matriu.h

```
#ifndef MATRIU_H
#define MATRIU_H
#include <string>
using namespace std;

int** creaMatriu(int nFiles, int nColumnes);
void llegeixMatriu(int** m, int nFiles, int nColumnes, const string& nomFitxer);
void sumaMatrius(int** m1, int** m2, int** suma, int nFiles, int nColumnes);
void destrueixMatriu(int** m, int nFiles);
#endif
```

Sessio 15 - Exercici classe (Data límit: 07/05)

Titulacio.h

```
#pragma once
#include "Estudiant.h"
const int MAX_ESTUDIANTS = 300;
```

```
class Titulacio {
public:
    Titulacio() : m_nom(""), m_nEstudiants(0), m_nMaxAssignatures(0),
m_nMaxEstudiants(0), m_estudiants(nullptr) {}
    Titulacio(const string& nom, int maxAssignatures, int maxEstudiants) :
m_nom(nom), m_nEstudiants(0), m_nMaxAssignatures(maxAssignatures),
        m_nMaxEstudiants(maxEstudiants) {
        m_estudiants = new Estudiant[maxEstudiants];
    }
    Titulacio(const Titulacio& t);
    Titulacio& operator=(const Titulacio& t);
    ~Titulacio() { if (m_estudiants != nullptr) delete[] m_estudiants; }

    void afegeixEstudiant(const string& niu, const string& nom);
    bool eliminaEstudiant(const string& niu);
    Estudiant* consultaEstudiant(const string& niu);

    Estudiant getEstudiant(int posicio) { return m_estudiants[posicio]; }
    string getNom() const { return m_nom; }
    int getMaxAssignatures() const { return m_nMaxAssignatures; }
    int getMaxEstudiants() const { return m_nMaxEstudiants; }
    int getNEstudiants() const { return m_nEstudiants; }
private:
    string m_nom;
    int m_nMaxAssignatures;
    Estudiant* m_estudiants;
    int m_nMaxEstudiants;
    int m_nEstudiants;
};
```

Titulacio.cpp

```
#include "Titulacio.h"
#pragma warning(disable:6385)

Titulacio& Titulacio::operator=(const Titulacio& t) {
    if (this != &t) {
        m_nom = t.m_nom;
        m_nMaxAssignatures = t.m_nMaxAssignatures;
        m_nMaxEstudiants = t.m_nMaxEstudiants;
        m_nEstudiants = t.m_nEstudiants;
        if (m_estudiants != nullptr)
            delete[] m_estudiants;

        if (t.m_estudiants != nullptr) {
            m_estudiants = new Estudiant[m_nMaxEstudiants];
            for (int i = 0; i < m_nEstudiants; i++)
                m_estudiants[i] = t.m_estudiants[i];
        }
        else
            m_estudiants = nullptr;
```

```
        return *this;
    }
}


Titulacio::Titulacio(const Titulacio& t) {
    m_nom = t.m_nom;
    m_nMaxAssignatures = t.m_nMaxAssignatures;
    m_nMaxEstudiants = t.m_nMaxEstudiants;
    m_nEstudiants = t.m_nEstudiants;

    if (t.m_estudiants != nullptr) {
        m_estudiants = new Estudiant[m_nMaxEstudiants];
        for (int i = 0; i < m_nEstudiants; i++)
            m_estudiants[i] = t.m_estudiants[i];
    }
    else
        m_estudiants = nullptr;
}

void Titulacio::afegeixEstudiant(const string& niu, const string& nom) {
    Estudiant e(niu, nom, m_nMaxAssignatures);
    m_estudiants[m_nEstudiants] = e;
    m_nEstudiants++;
}

bool Titulacio::eliminaEstudiant(const string& niu) {
    bool trobat = false;
    int i = 0;
    while ((i < m_nEstudiants) && !trobat) {
        if (niu == m_estudiants[i].getNiu())
            trobat = true;
        else {
            i++;
        }
    }
    if (trobat) {
        m_estudiants[i].allibera();
        for (int j = i; j < (m_nEstudiants - 1); j++)
            m_estudiants[j] = m_estudiants[j + 1];
        m_nEstudiants--;
    }
    return trobat;
}

Estudiant* Titulacio::consultaEstudiant(const string& niu) {
    Estudiant* estudiant = nullptr;
    bool trobat = false;
    int i = 0;
    while ((i < m_nEstudiants) && !trobat) {
        if (niu == m_estudiants[i].getNiu())
            trobat = true;
        else {
            i++;
```

```
        }
    }
    if (trobat)
        estudiant = &m_estudiants[i];
    return estudiant;
}
```

Estudiant.h

```
#pragma once
#include <string>
#include <vector>
using namespace std;

class Estudiant {
public:
    Estudiant() : m_NIU(""), m_nom("") {};
    Estudiant(const string& niu, const string& nom, int nAssignatures);
    Estudiant(const Estudiant& e);
    ~Estudiant();

    string getNiu() const { return m_NIU; }
    string getNom() const { return m_nom; }
    void setNiu(const string& niu) { m_NIU = niu; }
    void setNom(const string& nom) { m_nom = nom; }

    void afegeixAssignatura(const string& assignatura);
    void insereixAssignatura(const string& assignatura);
    void eliminaAssignatura(const string& assignatura);
    void mostraAssignatures();
    int getNAssignatures() const { return m_assignatures.size(); }
    string getAssignatura(int posicio) const;

    void inicialitza(const string& niu, const string& nom);
    void allibera();
private:
    string m_nom;
    string m_NIU;
    vector<string> m_assignatures;
};
```

Estudiant.cpp

```
#include "Estudiant.h"
#include <iostream>

Estudiant::Estudiant(const string& niu, const string& nom, int nAssignatures) {
    m_NIU = niu;
    m_nom = nom;
```

```
    m_assignatures.clear();
}

Estudiant::~Estudiant() {
    m_assignatures.clear();
}

void Estudiant::inicialitza(const string& niu, const string& nom) {
    m_NIU = niu;
    m_nom = nom;
    m_assignatures.clear();
}

void Estudiant::allibera()
{
    m_assignatures.clear();
}

void Estudiant::afegeixAssignatura(const string& assignatura) {
    m_assignatures.push_back(assignatura);
}

void Estudiant::insereixAssignatura(const string& assignatura) {
    vector<string>::iterator iterator = m_assignatures.begin();
    vector<string>::iterator final = m_assignatures.end();
    bool trobat = 0;
    while (iterator != final && !trobat) {
        if (*iterator > assignatura)
            trobat = 1;
        else
            iterator++;
    }
    m_assignatures.insert(iterator, assignatura);
}

void Estudiant::eliminaAssignatura(const string& assignatura) {
    int i = 0;
    bool trobat = 0;
    while (i < m_assignatures.size() && !trobat) {
        if (m_assignatures[i] == assignatura) {
            m_assignatures.erase(m_assignatures.begin() + i);
            trobat = 1;
        }
        else
            i++;
    }
}

void Estudiant::mostraAssignatures() {
    for (int i = 0; i < m_assignatures.size(); i++)
        cout << m_assignatures[i] << endl;
}

string Estudiant::getAssignatura(int posicio) const {
```

```
    if ((posicio >= 0) && (posicio < m_assignatures.size()))
        return m_assignatures[posicio];
    else return "";
}

Estudiant::Estudiant(const Estudiant& e) {
    m_NIU = e.m_NIU;
    m_nom = e.m_nom;
    m_assignatures = e.m_assignatures;
}
```