Sessió 11

Tema 2

Exercici 2.8:

2.8 Donades les següents declaracions de les classes Titulacio i Estudiant:

```
#include <string>
using namespace std;
#include "Estudiant.h"
#include <forward_list>
class Titulacio
{public:
   void mostraEstudiants();
   void afegeixEstudiant(Estudiant &e);
 private:
   string m_nom;
   int m_nEstudiants;
   std::forward_list<Estudiant> m_estudiants;
};
```

```
#include "Assignatura.h"
#include <string>
using namespace std;
class Estudiant
{public:
    void mostraAssignatures();
    string& getNIU() ;
    int getNAssigActual();
    Estudiant()
    Estudiant(const string &nom, const string &niu)
    bool afegeixAssignatura(int codi, int nCredits);
 private:
    string m_nom;
    string m_NIU;
    int m_nAssigActual;
    static const int MAX_ASSIGNATURES = 10;
    AssignaturaExpedient  m_assignatures[MAX_ASSIGNATURES];
};
```

Exercici 2.8:

2.8 Implementeu mètode

void eliminaEstudiants();

elimini tots els estudiants d'una titulació amb 0 assignatures.

```
#include <string>
using namespace std;
#include "Estudiant.h"
#include <forward_list>
class Titulacio
{public:
   void mostraEstudiants();
   void afegeixEstudiant(Estudiant &e);
 private:
   string m_nom;
   int m_nEstudiants;
   std::forward_list<Estudiant> m_estudiants;
};
```

```
#include "Assignatura.h"
#include <string>
using namespace std;
class Estudiant
{public:
    void mostraAssignatures();
    string& getNIU() ;
    int getNAssigActual();
    Estudiant()
    Estudiant(const string &nom, const string &niu)
    bool afegeixAssignatura(int codi, int nCredits);
 private:
    string m_nom;
    string m_NIU;
    int m_nAssigActual;
    static const int MAX_ASSIGNATURES = 10;
    AssignaturaExpedient  m_assignatures[MAX_ASSIGNATURES];
};
```

Exercici 2.8: Solucio eliminaEstudiants

- **Com obtenir primer element de la llista?**

```
m_estudiants.begin();
```

- **Si hem de recorrer la llista per eliminar un element, amb un iterador a l'element actual tenim prou?**

  NO, necessitem l'anterior

- **Com inicialitzem l'iterador anterior amb les llistes stl?**

```
m_estudiants.before_begin();
```

- **Com sabem si hem arribat al final de la llista?**

```
m_estudiants.end()
```

- **Com esborrem un element de la llista?**

```
m_estudiants.erase_after(anterior);
```

- **Com definim un iterador a una llista d'Estudiant?**

```
std::forward_list<Estudiant>::iterator
```

Exercici 2.8: Solucio eliminaEstudiants

```
void Titulacio::eliminaEstudiants()
{
  std::forward_list<Estudiant>::iterator anterior =
                                        m_estudiants.before_begin();
  std::forward_list<Estudiant>::iterator actual =
                                        m_estudiants.begin();
  while (actual != m_estudiants.end())
  {
    Estudiant estActual = *actual;
    if (estActual.getNAssigActual() == 0)
    {
       actual = m_estudiants.erase_after(anterior);
       m_nEstudiants--;
    }
    else
    {
       anterior = actual;
       actual++;
    }
  }
}
```

Exercici 2.8:

2.8 Implementeu mètode a Titulacio: Estudiants ordenats per NIU ascendentment

```
bool buscaEstudiant(const string& niu,
                    std::forward_list<Estudiant>::iterator& actual,
                    std::forward_list<Estudiant>::iterator& anterior);
```

Que cerqui a la llista d'estudiants de la titulació un Estudiant:

Si el troba retorna cert i un iterador a la seva posicio i a l'anterior

En cas contrari retorna fals.

```
#include <string>
using namespace std;
#include "Estudiant.h"
#include <forward_list>
class Titulacio
{public:
   void mostraEstudiants();
   void afegeixEstudiant(Estudiant &e);
 private:
   string m_nom;
   int m_nEstudiants;
   std::forward_list<Estudiant> m_estudiants;
};
```

Exercici 2.8:

```
bool buscaEstudiant(const string& niu,
                    std::forward_list<Estudiant>::iterator& actual,
                    std::forward_list<Estudiant>::iterator& anterior);
```

```
...
class Titulacio
{public:
   void mostraEstudiants();
   void afegeixEstudiant(Estudiant &e);
 private:
   string m_nom;
   int m_nEstudiants;
   std::forward_list<Estudiant> m_estudiants;
};
```

```
#include "Assignatura.h"
#include <string>
using namespace std;
class Estudiant
{public:
    void mostraAssignatures();
    string& getNIU() ;
    int getNAssigActual();
    Estudiant()
    Estudiant(const string &nom, const string &niu)
    bool afegeixAssignatura(int codi, int nCredits);
 private:
    string m_nom;
    string m_NIU;
    int m_nAssigActual;
    static const int MAX_ASSIGNATURES = 10;
    AssignaturaExpedient  m_assignatures[MAX_ASSIGNATURES];
};
```

Exercici 2.8:

```
bool Titulacio::buscaEstudiant(const string& niu,
                               std::forward_list<Estudiant>::iterator& actual,
                               std::forward_list<Estudiant>::iterator& anterior)
```

```
bool Titulacio::afegeixEstudiant (Estudiant &e)
{
   bool trobat = false;
   std::forward_list<Estudiant>::iterator anterior = m_estudiants.before_begin();
   std::forward_list<Estudiant>::iterator actual = m_estudiants.begin();
   while ((actual != m_estudiants.end()) && (!trobat))
   {
      Estudiant estActual = *actual;
      if (e.getNIU() < estActual.getNIU())
         trobat = true;
      else
      {
         anterior = actual;
         actual++;
      }
   }
   if (!trobat)
   { m_estudiants.insert_after(anterior, e);
     m_nEstudiants++;
   }
   return !trobat;
}
```

Exercici 2.8: Solucio buscaEstudiant

```
bool Titulacio::buscaEstudiant(const string& niu,
                               std::forward_list<Estudiant>::iterator& actual,
                               std::forward_list<Estudiant>::iterator& anterior)
{
  bool trobat = false;
  bool existeix = true;
  anterior = m_estudiants.before_begin();
  actual = m_estudiants.begin();
  while ((actual != m_estudiants.end())&&!trobat&&existeix)
  {  if ((*actual).getNIU() == niu){trobat = true;}
     else
     {    if ((*actual).getNIU() > niu)
          { existeix = false;}
          else
          {  anterior = actual;
             actual++;
          }
     }
  }
  return trobat;
}
```

Exercici 2.8: Solucio buscaEstudiant

```
bool Titulacio::buscaEstudiant(const string& niu,
                               std::forward_list<Estudiant>::iterator& actual,
                               std::forward_list<Estudiant>::iterator& anterior)
{
  bool trobat = false;
  bool existeix = true;
  anterior = m_estudiants.before_begin();
  actual = m_estudiants.begin();
  while ((actual != m_estudiants.end())&&!trobat&&existeix)
  {  if ((*actual).getNIU() == niu){trobat = true;}
     else
     {    if ((*actual).getNIU() > niu)
          { existeix = false;}
          else
          {  anterior = actual;
             actual++;
          }
     }
  }
  return trobat;
}
```

A.Si el troba:

- **actual:** iterador a actual
- **anterior**: iterador a anterior

B. Si no el troba:

- **actual:** iterador a element seguent a on hauria d'estar NIU buscat
- **anterior:** iterador a element anterior a on hauria d'estar NIU buscat.

Exercici 2.8:

2.8 Com quedaria ara el mètode afegeixEstudiant?

```
bool Titulacio::buscaEstudiant(const string& niu,
                               std::forward_list<Estudiant>::iterator& actual,
                               std::forward_list<Estudiant>::iterator& anterior)
```

```
bool Titulacio::afegeixEstudiant (Estudiant &e)
{
   bool trobat = false;
   std::forward_list<Estudiant>::iterator anterior = m_estudiants.before_begin();
   std::forward_list<Estudiant>::iterator actual = m_estudiants.begin();
   while ((actual != m_estudiants.end()) && (!trobat))
   {
      Estudiant estActual = *actual;
      if (e.getNIU() < estActual.getNIU())
         trobat = true;
      else
      {
         anterior = actual;
         actual++;
      }
   }
   if (!trobat)
   { m_estudiants.insert_after(anterior, e);
     m_nEstudiants++;
   }
   return !trobat;
}
```

Exercici 2.8:

2.8 Com quedaria ara el mètode afegeixEstudiant?

```
bool Titulacio::buscaEstudiant(const string& niu,
                               std::forward_list<Estudiant>::iterator& actual,
                               std::forward_list<Estudiant>::iterator& anterior)
```

```
void Titulacio::afegeixEstudiant (Estudiant &e)
{
   bool trobat = false;
   std::forward_list<Estudiant>::iterator anterior = m_estudiants.before_begin();
   std::forward_list<Estudiant>::iterator actual = m_estudiants.begin();
   while ((actual != m_estudiants.end()) && (!trobat))
   {
      Estudiant estActual = *actual;
      if (e.getNIU() < estActual.getNIU())
         trobat = true;
      else
      {
         anterior = actual;
         actual++;
      }
   }
   if (!trobat)
   { m_estudiants.insert_after(anterior, e);
     m_nEstudiants++;
   }
   return !trobat;
}
```

Exercici 2.8: Sol afegeixEstudiant amb buscarEstudiant

2.8 Com quedaria ara el mètode afegirEstudiant?

```
bool Titulacio::afegeixEstudiant(Estudiant &e)
{
    bool trobat = false;
    std::forward_list<Estudiant>::iterator anterior;
    std::forward_list<Estudiant>::iterator actual;

    trobat = buscaEstudiant(e.getNIU(),actual,anterior);

    if (!trobat)
    {
        m_estudiants.insert_after(anterior, e);
        m_nEstudiants++;
    }
    return !trobat;
}
```

Exercici 2.8:

2.8 Implementeu mètode a Titulacio

```
bool afegeixAssignaturaEstudiant(const string& niu,
                                 int codiAssig, int nCredits)
```

Que cerqui a la llista d'estudiants de la titulació un Estudiant i si existeix i te espai al seu array d'assignatures li afegeixi una asignatura i retorni true i si o no existeix o existeix però no té espai a l'array d'assignatures retorni false.

Suposeu que teniu un mètode a Estudiant:

```
bool afegeixAssignatura(int codi, int nCredits)
```

```
#include <string>
using namespace std;
#include "Estudiant.h"
#include <forward_list>
class Titulacio
{public:
   void mostraEstudiants();
   void afegeixEstudiant(Estudiant &e);
   bool buscaEstudiant(const string& niu,
                       std::forward_list<Estudiant>::iterator& actual,
                       std::forward_list<Estudiant>::iterator& anterior)
 private:
   string m_nom;
   int m_nEstudiants;
   std::forward_list<Estudiant> m_estudiants;
};
```

Exercici 2.8: Solucio afegeixAssignaturaEstudiant

```
bool Titulacio::afegeixAssignaturaEstudiant(const string& niu,
                                            int codiAssig,
                                            int nCredits)
{
    bool trobat = false;
    bool inserida = false;
    std::forward_list<Estudiant>::iterator anterior;
    std::forward_list<Estudiant>::iterator actual;
    trobat = buscaEstudiant(niu, actual,anterior);
    if (trobat)
    {
     inserida =(*actual).afegeixAssignatura(codiAssig, nCredits);
    }
    return inserida;
}
```

Exercici 2.8: Solucio afegeixAssignatura

```
bool Estudiant::afegeixAssignatura(int codi, int nCredits)
{
  bool inserit = false;

  if (m_nAssigActual < MAX_ASSIGNATURES)
  {
    bool trobat = false;
    int pos;
    trobat = buscaAssignatura(codi, pos);
    if (!trobat)
    {
        m_assignatures[m_nAssigActual].setCodi(codi);
        m_assignatures[m_nAssigActual].setCredits(nCredits);
        m_nAssigActual++;
        inserit = true;
    }
   }

   return inserit;
}
```

Exercici 2.8: Solucio buscaAssignatura

```
bool Estudiant::buscaAssignatura(int codi, int& pos)
{
  bool trobat = false;
  pos = 0;

  while ((pos < m_nAssigActual) && (!trobat))
  {
    if (codi == m_assignatures[pos].getCodi())
    {
      trobat = true;
    }
    else
    {
      pos++;
    }
  }
  return trobat;
}
```