Standard Template Library: Classe `vector`
Tema 2: Estructures de dades dinàmiques

Standard Template Library - STL

- Standard Template Library - STL:
  - Definició de classes estàndard per utilitzar diferents tipus d'estructures dinàmiques (arrays dinàmics, llistes, piles, cues):
    - `vector`
    - `forward_list(*)` (llista simplement enllaçada)
    - `List(*)` (llista doblement enllaçada)
    - `Stack(*)` (pila)
    - `Qeue(*)` (cua)
  - Basades en *templates* per poder-se utilitzar fàcilment amb qualsevol tipus de dades com a element base
- Referència i detalls d'utilització:
  - https://es.wikipedia.org/wiki/Standard_Template_Library
  - http://en.cppreference.com/w/cpp/container

(*) Aquestes classes les veurem més endavant quan expliquem estructures dinàmiques amb nodes enllaçats

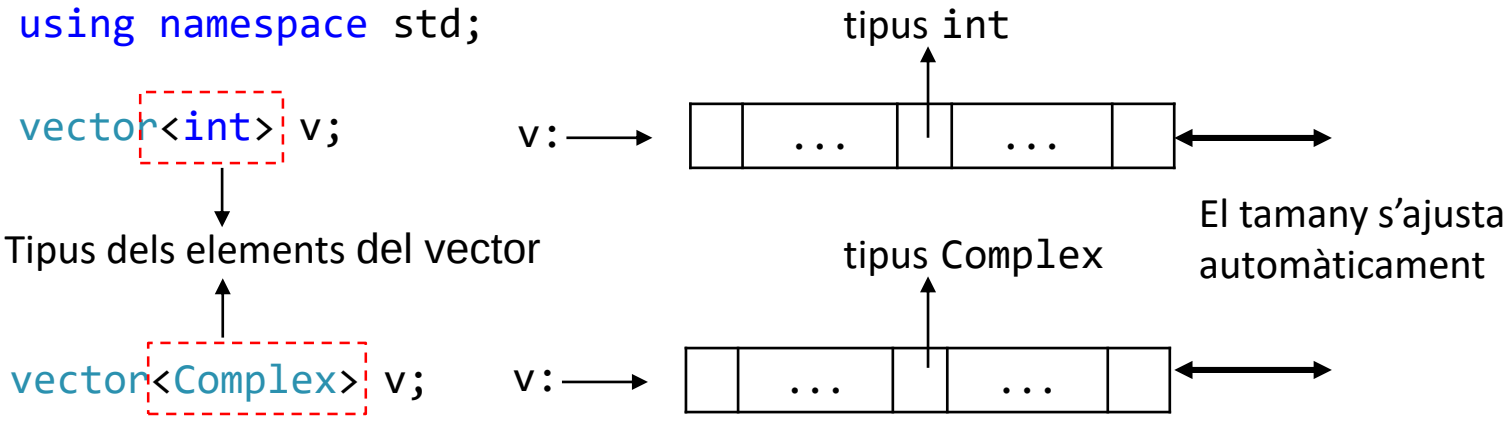
Classe vector: declaració

Classe vector https://en.cppreference.com/w/cpp/container/vector

- Permet gestionar arrays dinàmics de qualsevol tipus
  - El tipus de l'array s'especifica en el moment de la declaració de l'array utilitzant *templates*.
- El tamany de l'array dinàmic s'ajusta automàticament al nº d'elements que hi tenim guardats:
  - Si el nº d'elements creix molt automàticament es reserva més memòria i es fa una còpia de tots els elements al nou espai de memòria.

```
#include <vector>
using namespace std;

vector<int> v;
vector<Complex> v;
```

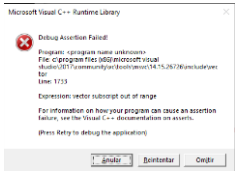


Classe vector: accés als elements

Classe vector: accés als elements

- L'accés es pot fer per índex (començant per 0) igual que als arrays.
- Només es pot accedir als índexs que corresponen a posicions vàlides (entre 0 i el nº d'elements actuals del vector).
- Inicialment, el nº d'elements del vector és 0.
- Podem recuperar el nº d'elements actual cridant al mètode `size()`
- El nº d'elements actuals del vector canvia quan:
  - Afegim un element al vector (operacions `push_back`, `insert`)
  - Eliminem un element al vector (operacions `pop_back`, `erase`)
  - Redimensionem explícitament el tamany del vector (operació `resize`)

```
vector <int> v;
v[0] = 1;
```



v: ⟶

v.size() ⟶ 0

```
vector <int> v;
v.push_back(0);
v[0] = 10;
```

v: ⟶ | 10 |

v.size() ⟶ 1

```
vector <int> v;
v.resize(5);
for (int i = 0; i < v.size(); i++)
   v[i] = i;
```

v: ⟶ | 0 | 1 | 2 | 3 | 4 |

v.size() ⟶ 5

Exercici

Recuperem l'exemple dels estudiants i les assignatures

```
class Estudiant
{
private:
   string m_nom;
   string m_NIU;
   string *m_assignatures;
   int m_maxAssignatures;
   int m_nAssignatures;
};
```

```
Estudiant::~Estudiant()
{
   if (m_assignatures != nullptr)
      delete[] m_assignatures;
}
```

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

- Com haurem de modificar la declaració de la classe `Estudiant`, del constructor i del destructor si volem utilitzar la classe `vector` per guardar l'array dinàmic d'assignatures?

Exercici: solució

```
class Estudiant
{
private:
   string m_nom;
   string m_NIU;
   string *m_assignatures;
   int m_maxAssignatures;
   int m_nAssignatures;
};
```



```
class Estudiant
{
private:
   string m_nom;
   string m_NIU;
   vector<string> m_assignatures;
};
```

- Declarem `m_assignatures` com a `vector<string>`
- Ja no cal l'atribut `m_maxAssignatures`: la classe `vector` ja gestiona de forma automàtica el tamany màxim.
- Ja no cal l'atribut `m_nAssignatures`: ho podem recuperar amb el mètode `size` de la classe vector.

Exercici: solució

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



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

- El constructor de la classe `Estudiant` crida per defecte al constructor de la classe `vector`
- No cal reservar memòria perquè ja ho fa per defecte el constructor de la classe `vector`

Exercici: solució

```
Estudiant::~Estudiant()
{
   if (m_assignatures != nullptr)
      delete[] m_assignatures;
}
```



```
Estudiant::~Estudiant()
{

}
```

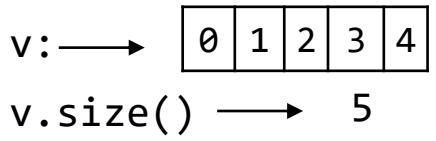
- El destructor de la classe `Estudiant` crida per defecte al destructor de la classe `vector`
- No cal alliberar memòria perquè ja ho fa per defecte el constructor de la classe `vector`

Classe vector: afegir/eliminar al final

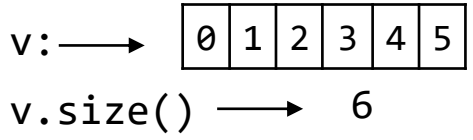
Classe vector: afegir/eliminar al final del vector

- **push_back(element)**: afegeix un element al final del vector, incrementant el nº d'elements actual.
- **pop_back()**: elimina l'últim element del vector, decrementant el nº d'elements actual

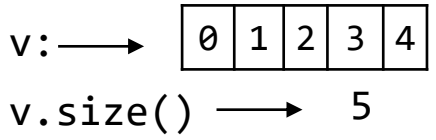
```
vector <int> v;
v.resize(5);
for (int i = 0; i < v.size(); i++)
   v[i] = i;
```




```
v.push_back(5);
```




```
v.pop_back();
```



Exercici

Afegiu a la classe **`Estudiant`** un **mètode `afegeixAssignatura`** que permeti afegir una nova assignatura a l'array dinàmic, a l'última posició de l'array.

```
void Estudiant::afegeixAssignatura(const string& assignatura)
{
   // Afegir codi
}
```

Exercici: solució

Afegiu a la classe **`Estudiant`** un **mètode `afegeixAssignatura`** que permeti afegir una nova assignatura a l'array dinàmic, a l'última posició de l'array.

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

Classe `vector`: afegir/eliminar al mig

Classe `vector`: afegir/eliminar al mig del vector

- **`insert(iterator, element)`**: afegeix un element a la posició del vector posterior a la posició indicada per un **iterador**, incrementant el nº d'elements actual i desplaçant tots els elements posteriors a l'element actual.
- **`erase(iterator)`**: elimina l'element del vector que ocupa la posició del vector indicada per un **iterador**, decrementant el nº d'elements actual i desplaçant tots els elements posteriors a l'element actual.

... però, què són els **iteradors**?

- Objectes que permeten fer referència a una posició del vector (s'utilitzaran també més endavant en altres classes de la STL, com per exemple les llistes).
- Encapsula el concepte de referència a un element de qualsevol estructura (vector, llista, etc.), independentment de la implementació concreta de l'estructura i de les referències als elements.
- Serveixen per indicar una posició dins del vector i recórrer tots els elements de la vector.

Classe vector: *iteradors*

Declaració:

```
vector<Estudiant>::iterator actual;
```

Tipus del vector amb el que volem utilitzar l'iterador

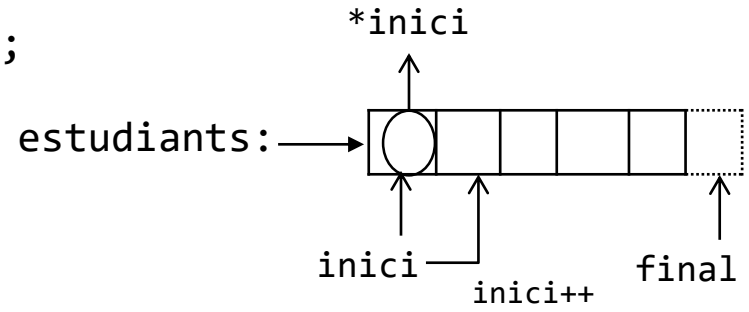
Definim un iterador sobre el vector

Operacions amb iteradors

- `begin()`: retorna un iterador a la primera posició d'un vector
- `end()`: retorna un iterador a la posició posterior a l'últim element d'un vector. Serveix per detectar si en un recorregut del vector ja hem arribat al final del vector
- `*actual`: accedeix al contingut de la posició indicada per l'iterador. Mateixa sintaxi que els apuntadors
- `actual++`: avança l'iterador a la següent posició dins del vector

```
vector<Estudiant> estudiants;
vector<Estudiant>::iterator inici, final;

inici = estudiants.begin();
final = estudiants.end();
inici++;
```




```
string nom = (*inici).getNom();  <---->  inici->getNom();
```

Exercici

1. **Afegiu** a la classe **`Estudiant`** un **mètode `insereixAssignatura`** que permeti afegir una nova assignatura al vector, mantenint l'ordre alfabètic.

2. **Afegiu** a la classe **`Estudiant`** un **mètode `eliminaAssignatura`** que permeti eliminat una assignatura del vector, a partir del seu nom.

3. **Afegiu** a la classe **`Estudiant`** un **mètode `mostraAssignatures`** que mostri per pantalla el nom de totes les assignatures de l'estudiant.

Exercici: solució

```
void Estudiant::insereixAssignatura(const string& assignatura)
{
   // Declarar iterador
   // Inicialitzar iterador al primer element del vector

   bool trobat = false;

   while (!trobat && iterador != final del vector)
   {
      if ( element actual de l'iterador > assignatura)
         trobat = true;
      else
         // incrementar iterador
   }
   // afegir element a partir de l'iterador actual
}
```

Exercici: solució

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

Inicialització iterador al principi del vector

Comprovació de si hem arribat al final del vector

Accés a l'assignatura referenciada per l'iterador

Inserim a la posició anterior on hem detectat una assignatura més gran

Exercici: solució

```
void Estudiant::eliminaAssignatura(const string& assignatura)
{
   vector<string>::iterator actual = m_assignatures.begin();
   bool trobat = false;
   while (!trobat && (actual != m_assignatures.end()))
   {
      if (*actual == assignatura)
         trobat = true;
      else
         actual++;
   }
   if (trobat)
     m_assignatures.erase(actual);
}
```

Inicialització iterador al principi del vector

Comprovació de si hem arribat al final del vector

Accés a l'assignatura referenciada per l'iterador

Si l'hem trobat eliminem l'assignatura referenciada per l'iterador

Exercici: solució

Amb iteradors:

```
void Estudiant::mostraAssignatures()
{
   vector<string>::iterator actual;
   for (actual = m_assignatures.begin(); actual != m_assignatures.end();
         actual++)
      cout << *actual << endl;
}
```

Amb índexos:

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