C++ Templates

DPTO. INFORMATICA - UNIVERSIDAD DE CORDOBA

Una breve introducción

Ventajas/Inconvenientes

- Ventajas:
  - Permiten:
    - Definir algoritmos genéricos.
    - Definir TAD genéricos.

- Inconvenientes:
    - Una sintaxis un poco complicada.
    - Mayor tiempo de compilación.
    - Complican el polimorfismo y la herencia.

La STL

- El lenguaje C++ incorpora una Librería Estándar de Templates (STL).
- Ya la habrás usado:


```
#include <vector>
#include <algorithm>
...
std::vector<float> miVector(10);
std::list<int> miLista;

std::sort(miVector.begin(), miVector.end());
std::sort(miLista.begin(), miLista.end());
```

Instancia de un template

algoritmo genérico

Sintaxis

- Para un tipo genérico:

lista.hpp

```
template< class T>
class Lista:
{
public:
  typedef T DataType;
  T const& item() const { return _item;}
  Void setItem (T const& it) {_item=it};
private:
   T _item;
};
```

main.cpp

```
#include “lista.hpp”
Lista<int> list;
Lista<int>::DataType a = list.item();
```

Sintaxis

- Para un algoritmo genérico:

algoritmos.hpp

```
template< class T>
bool isEqual (T const& a, T const& b)
{
    return a==b;
};
```


main.cpp

```
#include “algoritmos.hpp”

int main() {
int a, b;
...
if (isEqual(a, b))
```

Sintaxis

- Para un Functor:

algorithm.hpp

```
#include <list.hpp>

template< class T>
class Accumulator
{
Public:
    Acuumulator() : _d(0) {}
    void operator()(T const& v) { _d += v;}
    T value() const {return _d;}
private:
    T _d;
};

template<class T, class Functor>
void processList(List<T> const& l, Functor& f)
{
   List<T>::constIterator it=l.begin();
   while (it != l.end())
     f(*it)
}
```

main.cpp

```
#include <list.hpp>
#include <algorithm.hpp>

int main()
{
  List<int> l;
...
  Accumulator<int> f;
  processList(l, Accumulator<int>);
  std::cout << “acc=” << f.value()
            << std::endl;
}
```

Dónde definir un template

- ¿En un .hpp o en un .cpp?
  - Un template representa un conjunto infinito de tipos: ejemplo una lista<T>: lista<int>, lista<char>, lista< lista<int> > .....
  - No podemos compilar las infinitas instancias de un template.
  - Solución: definir los templates en línea en fichero .hpp e incluir este fichero allí donde haga falta.
  - El compilador compilará sólo aquellas instancias para los tipos usados (y sólo los métodos realmente usados).
  - Inconveniente: compilación lenta.

Dónde definir un template

- Especializaciones.
  - Podemos especializar un template para tipos concretos e implementarlo en un .cpp para compilarlo una sóla vez. (Si todos los tipos genéricos están resueltos).

algoritmos.cpp

```
#include <cmath>
#include “algoritmos.hpp”
const double EPS=1.0e-5;

template<>
bool isEqual<double> (double const&a,
double const& b)
{
   return fabs(a-b)<=EPS;
};
```

main.cpp

```
…
double a, b;
if (isEqual(a,b)) {
```

```
g++ -Wall algoritmos.cpp
g++ -Wall main.cpp
g++ -o miprog main.o algoritmos.o
```

Algunas cosillas más

- Podemos tener parámetros por defecto:

lista.hpp

```
template<class T=int>
class Lista
{
public:
  typedef T DataType;
  T const& item() const { return _item;}
   ...
private:
   T _item;
};
```

main.cpp

```
#include “lista.hpp”
Lista list;
Lista::DataType a = list.item();
```

Algunas cosillas más

- Podemos tener parámetros con valor:

vector.hpp

```
template<class T, int _dim>
class Vector
{
public:
   typedef T DataType;
   int dim() const { return _dim;}
   ...
private:
   T _data[_dim];
};

typedef Vector<unsigned char, 3> Vec3b;
typedef vector<float, 4> Vec4f;
```

main.cpp

```
#include “vector.hpp”
int main() {
Vec3b unVector;
```

Referencias

- B. Stroustrup, “The C++ Programming Language (4th Edition)”, Addison-Wesley ISBN 978-0321563842. May 2013.