



FACULTAD DE INFORMÁTICA

Plantillas

Baltasar Fernández Manjón

<http://www.fdi.ucm.es/profesor/balta/>



Plantillas

- Permiten que las funciones y las clases sean parametrizables
 - El tipo de dato sobre el que operan se recibe como parámetro
- Proporcionan reutilización de código y genericidad
 - Una misma definición se usa para crear múltiples instancias de funciones o clases que operan sobre distintos tipos de datos

Laboratorio de programación II (Facultad de Informática)

- 1



Plantillas de funciones

- Permiten ampliar la abstracción procedimental
 - generalizandola a funciones que tienen la misma lógica de operación y en lo único que se diferencian es en el tipo de objeto sobre el que se opera
- Se usa la palabra *template* para indicar que es un patrón y no una definición de función

```
template <typename ParámDeTipo1, typename ParámDeTipo2, ...>  
DefiniciónFunción
```

- Los parámetros de tipo se escriben entre corchetes angulares (<>)
- Cada parámetro de tipo debe aparecer por lo menos una vez en la lista de parámetros de la función
- Tanto el prototipo de la función como su definición deben aparecer en el mismo fichero

Laboratorio de programación II (Facultad de Informática)

- 2



Ejemplo plantilla de función

```
/* Plantilla que intercambia los valores de dos objetos de cualquier tipo
```

```
Recibe: un parámetro de tipo  
primero y segundo (objetos del mismo tipo)
```

```
Devuelve: primero y segundo intercambiados
```

```
Nota: Se supone que la asignación está definida para TipoDato
```

```
*/
```

```
template <typename TipoDato > // tipo parámetro
```

```
void intercambio(TipoDato & primero, TipoDato & segundo)
```

```
{ TipoDato temp = primero;
```

```
primero = segundo;
```

```
segundo = temp;
```

```
} //intercambio
```

Laboratorio de programación II (Facultad de Informática)

- 3



Instanciación implícita de plantillas de función

```
int main(int argc, char* argv[])
{
    int i=5; int j=7;
    cout << " i: " << i << " j: " << j << endl;
    intercambio(i,j); // instanciación con enteros
    cout << " i: " << i << " j: " << j << endl;
    string cad1="primera", cad2 ="segunda";
    cout << " cad1: " << cad1 << " cad2: " << cad2 << endl;
    intercambio(cad1,cad2); // instanciación con cadenas
    cout << " cad1: " << cad1 << " cad2: " << cad2 << endl;
}
```

Resultado

```
i: 5 j: 7
i: 7 j: 5
cad1: primera cad2: segunda
cad1: segunda cad2: primera
```



Plantillas de clases

■ Sintaxis

```
template <typename ParámDeTipo1, typename ParámDeTipo2, ...>
class ClasePlantilla
{
    //funciones miembro de la clase plantilla
}
```

- En vez de typename se puede poner class
- Las plantillas también pueden tener parámetros ordinarios (p.e. Para configuración)

■ Instanciación

```
ClasePlantilla<TipoObjetos> instanciaPlantilla;
```



Reglas para las plantillas de clases

- La definición de funciones miembro fuera de la declaración de la clase debe hacerse mediante plantillas de función
- Todos los usos del nombre de la plantilla de clase como un tipo de datos deben ser parametrizados
- Las funciones miembro deben definirse en el mismo fichero que la declaración de la clase
 - Se pone todo el código dentro del *fichero.h*



Ejemplo: nodo genérico de lista enlazada simple

```
#ifndef NodoLista_H
#define NodoLista_H
template< typename TipoInfoNodo>
class NodoLista {
public:
    NodoLista(); // constructor sin parámetros
    NodoLista( const TipoInfoNodo & ); // constructor
    void setInfo( TipoInfoNodo );
    TipoInfoNodo getInfo() const;
    void setSig( NodoLista * );
    const NodoLista *getSig() const;
private:
    TipoInfoNodo info; // datos almacenados en el nodo
    NodoLista<TipoInfoNodo> *sig; // puntero al siguiente nodo
}; // NodoLista
```

NodoLista.h



```
// constructor sin parámetros
template< typename TipoInfoNode >
NodoLista< TipoInfoNode >::NodoLista()
{
    sig = NULL;
} //constructor
// constructor
template< typename TipoInfoNode >
NodoLista< TipoInfoNode >::NodoLista( const TipoInfoNode &datos )
: info(datos), sig( NULL )
{ } //constructor
template< typename TipoInfoNode >
void NodoLista< TipoInfoNode >::setInfo( TipoInfoNode datos)
{
    info=datos;
} //setInfo
```

NodoLista.h



```
template< typename TipoInfoNode >
TipoInfoNode NodoLista< TipoInfoNode >::getInfo() const
{
    return info;
} // getInfo
template< typename TipoInfoNode >
void NodoLista< TipoInfoNode >::setSig( NodoLista * punteroSiguiente)
{
    sig=punteroSiguiente;
}
template< typename TipoInfoNode >
const NodoLista<TipoInfoNode>* NodoLista<TipoInfoNode>::getSig() const
{
    return sig;
}
#endif
```

NodoLista.h



Uso del NodoLista

```
#include <iostream>
#include <string>
#include "NodoLista.h"
using namespace std;
int main(int argc, char* argv[])
{
    NodoLista<string> nodoCadena;
    nodoCadena.setInfo("hola");
    cout <<nodoCadena.getInfo()<< endl;

    NodoLista<int> nodoEntero;
    nodoEntero.setInfo(5);
    cout <<nodoEntero.getInfo()<< endl; }
}
```