Programación Orientada a Objetos

Unidad 7:
Programación Genérica: Templates

Teoría 09 - 27/10/2011 - Pablo Novara

¿Qué es la Programación Genérica?

- Es un tipo de programación centrada mucho más en los algoritmos que en los datos.
- Se pretende generalizar el desarrollo de clases y funciones.
- El objetivo es no reescribir una misma función (por ejemplo ordenar, buscar, etc) para distintos tipos de datos.

¿Qué es Programación Genérica?

- Ejemplo: Buscar el menor entre N datos de un vector

```
??? menor (??? v[], int n) {
    ??? men(v[0]);
    for (int i=1; i<n;i++)
        if (v[i]>men)
            men=v[i];
    return men;
}
```

- Se podría reemplazar **???** por “cualquier” tipo de datos (int, float, double, string, etc.) ya que el algoritmo sería el mismo.

¿Qué es Programación Genérica?

Función genérica + argumentos

instanciación o especialización de la plantilla

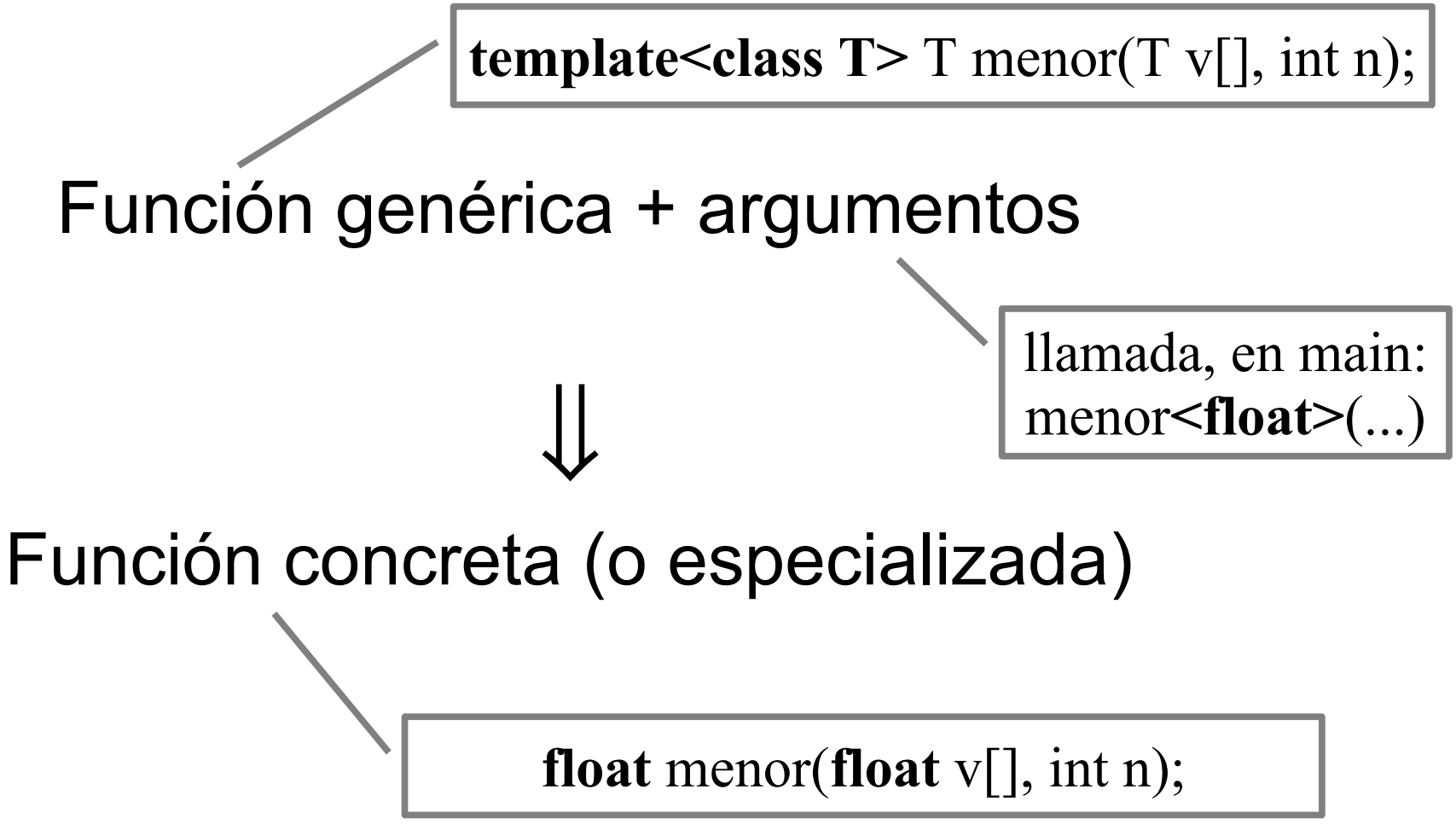
Función concreta (o especializada)

- Ejemplo: Buscar el menor entre N datos de un vector

```
template<class T>
T menor(T v[], int n) {
    T men(v[0]);
    for (int i=1; i<n;i++)
        if (v[i]<men)
            men=v[i];
    return men;
}
```

- La palabra clave **template** indica que se trata de una plantilla (función/clase genérica).
- **<class T>** indica que **T** será una clase cuando se especialice, es el argumento de la plantilla.

¿Qué es Programación Genérica?



- Ejemplo: Buscar el menor entre N datos de un vector
  - int main () {

```
…
int x[10];
for (int i=0;i<10;i++) x[i]=rand()%100;
int min_int = menor<int>(x,10);
…
string a[20];
for (int i=0;i<20;i++) cin>>a[i];
int min_str = menor<string>(a,10);
…
```

  }
- Se se indica con qué tipo se especializa al función.

– ¿Con qué tipos puedo especializar menor?

```
template<class T>
T menor(T v[], int n) {
    T men(v[0]);
    for (int i=1; i<n;i++)
        if (v[i]<men)
            men=v[i];
    return men;
}
```

– El tipo que reemplace a T debe:

- implementar un constructor de copia
- permitir comparar con el operador <

– Bonus: puntero a función

```
template<class T>
T menor(T v[], int n, bool (*f)(T t1, T t2)){
    T men(v[0]);
    for (int i=1; i<n;i++)
        if (f(v[i],men))
            men=v[i];
    return men;
}
```

– Reemplazando el operador < por una función **f** que se recibe como argumento, el mismo algoritmo busca menores según diferentes criterios.

– Ejemplo: Buscar el menor entre N datos de un vector

```
bool min_size(string s1,string s2) {
    return s1.size()<s2.size();
}
bool min_nocase(string s1, string s2) {
    for (int i=0;i<s1.size();i++) s1[i]=tolower(s1[i]);
    for (int i=0;i<s2.size();i++) s2[i]=tolower(s2[i]);
    return s1<s2;
}
…
    string m1 = menor(a,20,min_size);
    string m2 = menor(a,20,min_nocase);
…
```

– Templates de clases:

```
template<class T>
class Matriz3x3 {
    T M[3][3];
public:
    Matriz3x3();
    Matriz3x3 operator*(Matriz3x3 &m2);
    Matriz3x3 operator-(Matriz3x3 &m2);
    Matriz3x3 operator+(Matriz3x3 &m2);
    Matriz3x3 &Inverse();
    Matriz3x3 &Transpose();
    T *operator[](int i);
};
```

– Templates de clases:

```
ostream &operator<<(ostream &o, Matrix3x3 &m);
istream &operator>>(ostream &i, Matrix3x3 &m);


int main() {
    Matriz3x3<float> m1,m2;
    cin>>m1>>m2;
    cout<<m1*m2;
    return 0;
}
```

– Métodos de clases templatizadas:

```
template<class T>
Matriz3x3 &Matriz3x3<T>::Transpose() {
    T aux;
    for (int i=0;i<3;i++)
        for (int j=i+1;j<3;j++) {
            aux=M[i][j];
            M[i][j]=M[j][i];
            M[j][i]=aux;
        }
};
```

– Otros argumentos para un template:

```
template<class T, int N>
class Matriz {
    T M[N][N];
public:
    …
};

int main() {
    Matriz<float,3> mf3x3;
    Matriz<int,5> mi5x5;
    …
```