Programación Orientada a Objetos

Unidad 7: Programación genérica

UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

FICH UNL

EJEMPLO

Buscar el menor entre N datos de un `vector<int>`

```
int menor (const vector<int> &v) {
    int men = v[0];
    for (size_t i=1; i<v.size(); i++)
        if (v[i]<men)
            men = v[i];
    return men;
}
```

EJEMPLO

Buscar el menor entre N datos de un `vector<double>`

```
double menor (const vector<double> &v) {
    double men = v[0];
    for (size_t i=1; i<v.size(); i++)
        if (v[i]<men)
            men = v[i];
    return men;
}
```

EJEMPLO

Buscar el menor entre N datos de un vector<string>

```
string menor (const vector<string> &v) {
    string men = v[0];
    for (size_t i=1; i<v.size(); i++)
        if (v[i]<men)
            men = v[i];
    return men;
}
```

EJEMPLO

Buscar el menor entre N datos de un `vector<Foo>`

```
Foo menor (const vector<Foo> &v) {
    Foo men = v[0];
    for (size_t i=1; i<v.size(); i++)
        if (v[i]<men)
            men = v[i];
    return men;
}
...
bool operator<(Foo f1, Foo f2) { ... }
```

EJEMPLO

Buscar el menor entre N datos de un `vector<???>`

```
??? menor (const vector<???> &v) {
    ??? men = v[0];
    for (size_t i=1; i<v.size(); 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 siempre el mismo.

¿QUÉ ES PROGRAMACIÓN GENÉRICA?

Técnica de programación que **se enfoca en los algoritmos** e *ignora* los tipos de datos sobre los cuales se aplican.

El objetivo es no tener que reescribir un mismo algoritmo (por ejemplo ordenar, buscar, etc) o clase para distintos tipos de datos.

PROGRAMACIÓN GENÉRICA EN C++

Ejemplo:

```
template<typename T>
T menor(const vector<T> &v) {
    T men = v[0];
    for (size_t i=1; i<v.size(); i++)
        if (v[i]<men)
            men=v[i];
    return men;
}
```

- `template` indica que se trata de una plantilla (función/clase genérica).
- `<typename T>` (o `<class T>`) es el argumento de la plantilla.

PROGRAMACIÓN GENÉRICA EN C++

```
template<typename T>
T menor(const vector<T> &v) {
    ...
}
```

Función genérica (plantilla) + argumentos


```
vector<float> v;
...
float men =
    menor<float>(v);
```

Función concreta (especializada)

```
float menor(const vector<float> &v) {
    ...
}
```

PROGRAMACIÓN GENÉRICA EN C++

Ejemplo: Buscar el *menor* entre N datos de un vector de enteros y de otro de palabras

```
int main () {
    ...
    vector<int> x(10);
    for (int i=0;i<10;i++) x[i] = rand()%100;
    int min_int = menor<int>(x);
    ...
    vector<string> a(20);
    for (int i=0;i<20;i++) cin >> a[i];
    string min_str = menor<string>(a);
    ...
}
```

Se indica con qué tipo se especializa la plantilla.

PROGRAMACIÓN GENÉRICA EN C++

Ejemplo: Buscar el *menor* entre N datos de un vector de enteros y de otro de palabras

```
int main () {
    ...
    vector<int> x(10);
    for (int i=0;i<10;i++) x[i] = rand()%100;
    int min_int = menor(x); // menor recibe vector<T>
                            // y x es vector<int>,
                            // entonces T=int
    vector<string> a(20);
    for (int i=0;i<20;i++) cin >> a[i];
    string min_str = menor(a); // a es vector<string>,
                               // entonces T=string
}
```

Para una función, el tipo generalmente se puede deducir a partir de sus parámetros actuales

PROGRAMACIÓN GENÉRICA EN C++

```
template<typename T>
T menor(const vector<T> &v) {
    T men = v[0];
    for (size_t i=1; i<v.size(); 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 <
- permitir asignar con el operador =

TEMPLATES DE CLASES

```
template<typename T>
class Vector3D {
    T m_datos[3];
public:
    Vector3D();
    Vector3D operator-(const Vector3D &v2);
    Vector3D operator+(const Vector3D &v2);
    T operator*(const Vector3D &v2);
    T &operator[](int i);
};
```

```
template<typename T>
T &Vector3D<T>::operator[](int i) {
    return m_datos[i];
}
```

TEMPLATES DE CLASES

```
int main() {
    Vector3D<float> v1,v2;
    cin >> v1 >> v2;
    cout << v1*v2 << endl;
}
```

¿y los operadores << y >>?

```
istream &operator>>(istream &i, Vector3D<float> &v) {
    ...
}
```

Ó

```
template<typename T>
istream &operator>>(istream &i, Vector3D<T> &v) {
    ...
}
```

EJEMPLO

1. Escriba una clase genérica para gestionar registros directamente sobre un archivo binario.

   La clase debe tener métodos para:

   - Obtener la cantidad de registros
   - Consultar un registro
   - Añadir un registro
   - Modificar un registro

PUNTEROS A FUNCIONES

```
template<typename T>
T menor(const vector<T> &v) {
    T men(v[0]);
    for (size_t i=1; i<v.size(); i++)
        if (v[i]<men)  // if ( operator<(v[i],men) )...
            men = v[i];
    return men;
}
```

```
struct Alumno { ... };
bool operator<(const Alumno &a1, const Alumno &a2) { ... }
```

```
int main() {
    vector<Alumno> v;
    ...
    Alumno m = menor(v);
    ...
```

PUNTEROS A FUNCIONES

```
template<typename T, typename PFnc>
T menor(const vector<T> &v, PFnc comparador) {
    T men(v[0]);
    for (size_t i=1; i<v.size(); i++)
        if (comparador(v[i],men))
            men = v[i];
    return men;
}
```

Antes que intentar especificar el tipo de un puntero a función, conviene dejar que el compilador lo deduzca

PUNTEROS A FUNCIONES

```
template<typename T, typename PFnc>
T menor(const vector<T> &v, PFnc comparador);
```

```
bool cmp_edad(const Alumno &a1, const Alumno &a2) {
    return a1.edad < a2.edad;
}

bool cmp_nombre(const Alumno &a1, const Alumno &a2) {
    return a1.nombre < a2.nombre;
}

bool cmp_promedio(const Alumno &a1, const Alumno &a2) {
    return a1.prom < a2.prom;
}
```

PUNTEROS A FUNCIONES

```
template<typename T, typename PFnc>
T menor(const vector<T> &v, PFnc comparador);
```

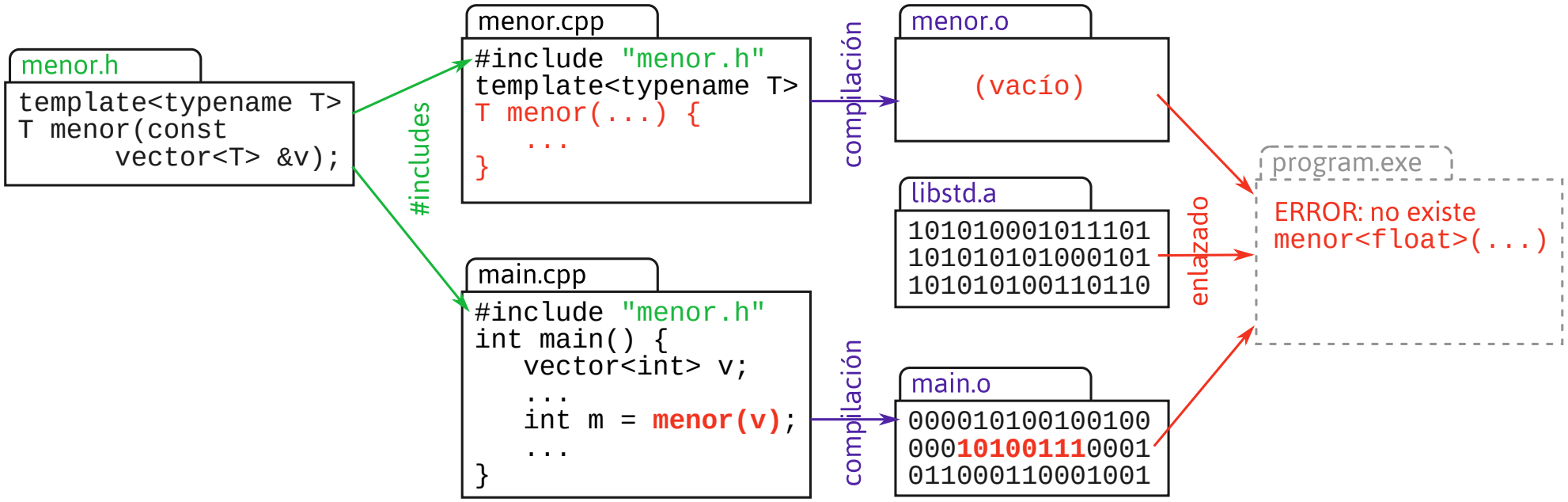
```
bool cmp_edad(...) { ... }
bool cmp_nombre(...) { ... }
bool cmp_promedio(...) { ... }
```

```
int main() {
    vector<Alumno> v;
    ...
    Alumno men_edad = menor(v,cmp_edad);
    Alumno men_nomb = menor(v,cmp_nombre);
    Alumno men_prom = menor(v,cmp_prom);
    ...
```

DESVENTAJAS DEL USO DE TEMPLATES

- Los errores que arroja el compilador suelen ser mucho más largos y confusos.

- Una biblioteca de templates debe estar definida por completo en un .h (no se puede separar en .h y .cpp)



TEMPLATES VS. "POLIMORFISMO"

Métodos virtuales

- Polimorfismo **dinámico**
  - en tiempo de ejecución
  - más flexible

Templates

- Polimorfismo **estático**
  - en tiempo de compilación
  - más eficiente