

ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

Plantillas en funciones

Aquí solo vemos para las funciones pero también veremos para las clases

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Ejemplo

Queremos conseguir lo mismo que con los genéricos de java con estas plantillas.

- Implementamos una función que calcula el mínimo de dos enteros:

```
int min(int a, int b) {  
    if (a ≤ b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```

Podemos tener funciones que nos calculen lo mismo para cualquier tipo de dato

- ¿Y si quiero calcular el mínimo de dos float? ¿y el mínimo de dos double? El mínimo de otro tipo de datos vaya.
- ¿Y si quiero calcular el mínimo de dos string utilizando el orden lexicográfico? Por ejemplo: `min("AA", "AB") = "AA"`.

Ejemplo

Aquí estamos haciendo un copia y pega del método en cada tipo de dato. Pero esto no es nada eficiente. Es con este ejemplo que nosotros introducimos los genéricos.

```
int min(int a, int b) {  
    if (a ≤ b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```

```
float min(float a, float b) {  
    if (a ≤ b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```

```
double min(double a, double b) {  
    if (a ≤ b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```

```
const std::string & min(const std::string &a, const std::string &b) {  
    if (a ≤ b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```

Podemos implementar la misma función para el resto de los tipos de datos, pero esto no sería muy funcional para nosotros.

- ¡Cuanta duplicidad!
- Todas tienen la misma implementación. ¡Solo difieren en los tipos!

Programación genérica

- Sería deseable tener una única versión genérica, que pudiese funcionar con varios tipos.

```
??? min(??? a, ??? b) {  
  if (a ≤ b) {  
    return a;  
  } else {  
    return b;  
  }  
}
```

Este término es análogo o similar al concepto de genéricos que vemos en TP2.

- **Solución:** plantillas (*templates*) en C++.

Plantillas en C++

- Son definiciones con «huecos» (**parámetros de plantilla**).
- Se especifican mediante la palabra `template`, seguida de los parámetros de plantilla, y seguida de la definición de función paramétrica.

```
template <typename T>
T min(T a, T b) {
    if (a ≤ b) {
        return a;
    } else {
        return b;
    }
}
```

La T representa un tipo

Plantilla para la función de antes.

Llamada a funciones plantilla

- Basta con indicar el tipo con el que queremos «rellenar» el marcador.


```
min<int>(6, 2)
min<double>(3.3, 5.5)
min<std::string>("Pepito", "Paula")
```

Los círculos son los tipos con los que llamamos a la función plantilla.

- Cada vez que se hace una llamada a la función genérica, se hace una versión específica para el tipo indicado en el marcador. A esto se le llama instanciación de plantillas.

```
template <typename T>
T min(T a, T b) {
    if (a ≤ b) {
        return a;
    } else {
        return b;
    }
}
```

T = int



```
int min<int>(int a, int b) {
    if (a ≤ b) {
        return a;
    } else {
        return b;
    }
}
```

Instanciación de plantillas

```
template <typename T>
T min(T a, T b) {
    if (a ≤ b) {
        return a;
    } else {
        return b;
    }
}
```

T = int

T = double

T = std::string

```
const Q
std::string min(const std::string a, const std::string b) {
    if (a ≤ b) {
        return a;
    } else {
        return b;
    }
}
```

```
int min<int>(int a, int b) {
    if (a ≤ b) {
        return a;
    } else {
        return b;
    }
}
```

```
double min<double>(double a, double b) {
    if (a ≤ b) {
        return a;
    } else {
        return b;
    }
}
```

Sin embargo, esta versión no es exactamente igual a la versión que habíamos escrito antes.

Tenemos que poner la versión constante en el caso de los strings si no NO es correcto.

Instanciación de plantillas

- En esta última instanciación (con un `string`) podemos indicar un tipo más preciso:

```
std::cout << min<const std::string &>("Pepito", "Ramiro") << std::endl;
```

ESTA ES MEJOR NO HACERLA.

- O bien modificar nuestra función genérica:

```
template <typename T>
const T & min(const T &a, const T &b) {
    if (a ≤ b) {
        return a;
    } else {
        return b;
    }
}
```

De tal manera que recibe y devuelve referencias constantes.

Esta última tiene más sentido.

Deducción de argumentos de plantilla

- Cada vez que hemos llamado a una función genérica, hemos indicado el tipo con el que debe instanciarse:

```
std::cout << min<std::string>("Pepito", "Ramiro") << std::endl;
```

El compilador de C++ puede deducirlo y por tanto no es estrictamente necesario declarar el tipo

- C++ permite omitirlo en la mayoría de los casos.
 - En ese caso intenta deducir el argumento de la plantilla.

```
std::cout << min("Pepito", "Ramiro") << std::endl;
```



No es necesario poner el tipo que le estamos pasando porque aparentemente c++ lo reconoce.

¡Cuidado con las instanciaciones!

- ¿Qué pasa si instancio la plantilla con dos complejos?

```
Complejo z1(1.0, 3.0), z2(4.0, -5.0);  
std::cout << min(z1, z2) << std::endl;
```

Recordar que en el curso de IA cuántica se habla sobre estos complejos y los cúbits.

- C++ realiza esta instanciación:

Yo creo que esto es porque funcionan para los tipos primitivos.

```
template <typename T>  
const T & min(const T &a, const T &b) {  
    if (a ≤ b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```

T = Complejo

Deberíamos sobrecargarlo SUPONGO

```
const Complejo & min(const Complejo &a,  
                    const Complejo &b)  
{  
    if (a ≤ b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```



No compila porque se está aplicando a dos objetos de la clase complejo

¡Cuidado con las instanciaciones!

- Los errores provocados por instanciaciones incorrectas suelen ser crípticos, largos, y difíciles de interpretar:

Test1.cpp: En la instanciación de 'const T& min(const T&, const T&) [con T = Complejo]':

Test1.cpp:76:28: se requiere desde aquí

Test1.cpp:40:11: error: no match for 'operator<=' (operand types are 'const Complejo' and 'const Complejo')

```
40 |         if (a <= b) {  
    |             ~^~  
    |
```

Los mensajes de error son incompresibles conviene mirar cuando creamos una instancia de una clase paramétrica.