Entendiendo C++

Samuel Rodríguez Sevilla

11 de marzo de 2009

Contenidos

Wikipedia

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

El lenguaje C++ tiene todas las características mínimas de los lenguajes orientados a objetos:

- Encapsulación
- Herencia
- Polimorfismo

Pero añade otras características interesantes:

- Espaciones de nombres (*namespaces*).
- Integración con C (C++ no es un lenguaje orientado a objetos puro).
- Funciones y métodos *inline*.
- Plantillas (*templates*).
- Sobrecarga de operadores.
- Parametros por referencia, devolución por referencia, devolución constante, métodos constantes, etc.

Pros:

- Uso de una sintaxis estándar (la de C).
- Generación de ejecutables binarios en código máquina.
- Mezcla de código no orientado a objetos y sí orientado a objetos.

Contras:

- No dispone de recolector de basura.
- Sintaxis algo pesada en ocasiones:

```
vector<int>::iterator i = v.begin();
```

- No dipone de una bibliotecas ricas en funcionalidades como Java o .Net.
- Es un lenguaje que puede ser muy complejo.
- Errores ininteligibles.

Sobrecarga de operadores

Introducción

- Mecanismo que permite redefinir los operadores de modo que puede adaptarse su uso a nuestra necesidad. Esto puede simplificar mucho el código y mejorar la legibilidad si se hace bien.
- Operadores sobrecargables:

+	-	*	/	%	^	&	|	~
!	=	<	>	+=	-=	*=	/=	%=
^	&=	|=	<<	>>	<<=	>>=	==	!=
<=	>=	&&	||	++	- -	,	->*	->
( )	[ ]	new	delete	new[]	delete[]			

Sobrecarga de operadores

Utilización

Hay dos formas de realizar sobrecarga de operadores:

- Como métodos dentro de clases:

```
class string
{
    string& operator+(const string& str);
};
```

- como funciones:

```
string& operator+(string& str1,
        const string& str2);
```

Functores

Introducción

- Los functores son clases que sobrecargan el operador paréntesis para que puedan ser utilizadas como funciones.
- Permite crear “funciones” que pueden ser inicializadas para que se adapten mejor a las necesidades del programador.

Functores

Utilización

```
template<typename T>
class comparar {
    int _dif = 0;
public:
    comparar(int dif) {
        _dif = dif;
    }
    bool operator()(const T& op1, const T& op2) {
        return op1+_dif-op2 > 0;
    }
};
/* ... */
vector<int> v(20);
sort(v.begin(), v.end(), comparar(3));
```

Plantillas

Introducción

- Las plantillas son el mecanismo que tiene C++ para realizar clases que se ajusten a cualquier tipo de dato (como los genéricos de Java o .Net).
- Eliminan la necesidad de utilizar *void** como en C y añaden comprobación de tipos.
- Se hace una compilación específica por cada tipo de dato de la plantilla (ocupa más espacio).
- Las plantillas pueden ser de clases, estructuras, métodos y funciones.

Plantillas

Utilización

```
template<typename TIPO, size_type SIZE = 10>
class array {
    TIPO _array[SIZE];
public:
    TIPO& operator[](size_type idx) {
        if(idx < 0 || idx >= SIZE)
            throw out_of_range();
        return _array[idx];
    }
    size_type size() {
        return SIZE;
    }
};
```

Plantillas

Utilización

```
template<typename T>
T square(T v) {
    return v*v;
}
```

Los conceptos nacen con el objetivo de facilitar el desarrollo con plantillas. Funcionan igual que las interfaces de otros lenguajes de programación, pero es en la plantilla en la que se dice que se va a usar, no en la clase que hipotéticamente heredaría de él.

```
concept C<typename T> { void f(int); }
template<typename T> requires C<T> class A {
    void h(const T& x) {
        x.f(5);
    }
};
```

Permite crear funciones directamente donde se las necesita y heredando las ventajas de los functores.

```
vector<int> v(20);
int _dif = 3;
sort(v.begin(), v.end(),
[_dif](const int op1, const int op2)
{ return op1+_dif-op2; });
```

El tipo auto hace que sea el compilador el que decida el tipo de una variable. Es muy útil en el caso de funciones lambda o cuando se trabaja con algunas plantillas.

```
vector<int> v(20);
int _dif = 3;
auto c = [_dif](const int op1, const int op2)
    { return op1+_dif-op2; };
sort(v.begin(), v.end(), c );
for(auto it = v.begin(); it != v.end(); it ++)
{ /* hacerl algo */ }
```

Más recursos

- Boost++, una de las mejores bibliotecas de clases para C++
- Standard Template Library, la biblioteca estandar de C++

Dudas

. . .

Fin