Seminario #14: C++ y Metaprogramación

Integrantes: - Carlos Aguila - Eduardo García - Ricardo Piloto

1 y 2: Metaprogramación y Templates

La *Metaprogramación* es la programación que manipula las entidades de un programa, como clases y funciones.

Los *Templates* brindan soporte directo para la programación genérica en forma de programación utilizando tipos como parámetros. El mecanismo de templates de **C++** permite que un tipo o un valor sea un parámetro en la definición de una clase, una función o un alias de un tipo. Los templates proporcionan una forma sencilla de representar una amplia gama de conceptos generales y formas sencillas de combinarlos. Las clases y funciones resultantes pueden coincidir con código escrito a mano, menos general, en tiempo de ejecución y eficiencia de espacio.

Esto lleva a la noción de *Template Metaprogramming (TMP)* como un ejercicio de escritura de programas que computan en tiempo de compilación y generan programas.

TMP surgió de forma accidental; durante el proceso de estandarización del C++ se descubrió que su sistema de templates es **Turing-completo**, o sea, capaz de computar cualquier función **Turing-computable**.

Ejemplos

Calculando el 20-ésimo número de la secuencia de Fibonacci

```
#include <iostream>
using namespace std;

template<int n>
struct Fib{
    static const int res = Fib<n-1>::res + Fib<n-2>::res;
};
template<>
struct Fib<1>{
    static const int res = 1;
};
template<>
struct Fib<2>{
    static const int res = 2;
};

int main(){
    cout << Fib<20>::res << '\n';
```

```
    return 0;
}
```

Calculando la sumatoria de 1 a 10 (*Unroll* de una expresión *for*)

Código para calcularla en tiempo de ejecución:

```
#include <iostream>
using namespace std;

int main(){
    int sum = 0;
    for (int i = 1; i <= 10;){
        sum += i++;
    }
    cout << sum << '\n';

    return 0;
}
```

Código para calcularla en tiempo de compilación:

```
#include <iostream>
using namespace std;

template<int n>
struct Sumatory{
    static const int sum = Sumatory<n-1>::sum + n;
};
template<>
struct Sumatory<1>{
    static const int sum = 1;
};

int main(){
    cout << Sumatory<10>::sum << '\n';
    return 0;
}
```

Imprimiendo la tabla de multiplicación del 2

Código para calcularla en tiempo de ejecución:

```
#include <iostream>
using namespace std;

void mult_print(int index, int value){
    cout << value << " x " << index << " = " << value*index << '\n';
```

```
}

int main(){
    for (int i = 1; i <= 10; i++){
        mult_print(i, 2);
    }

    return 0;
}
```

Código para calcularla en tiempo de compilación:

```
#include <iostream>
using namespace std;

template<int index, int end, int value, typename Function>
class FunctionExecutorLoop{
public:
    static void exec(Function f){
        f(index, value);
        FunctionExecutorLoop<index+1, end, value, Function>::exec(f);
    }
};
template<int end, int value, typename Function>
class FunctionExecutorLoop<end, end, value, Function>{
public:
    static void exec(Function f){
        f(end, value);
    }
};

void mult_print(int index, int value){
    cout << value << " x " << index << " = " << value*index << '\n';
}

int main(){
    FunctionExecutorLoop<1, 10, 2, decltype(mult_print)>::exec(mult_print);
    return 0;
}
```