Ejercitación – Templates

Se recomienda resolver los ejercicios en orden. En CLion se encuentran disponibles los siguientes targets:

- base: compila tests para el código base
- ejN, si ($1 \leq N \leq 9$): hasta al ejercicio N inclusive.

Los targets también pueden compilarse y ejecutarse sin usar CLion. Para ello:

1. En una consola pararse en el directorio raíz del projecto. En este debería haber un archivo CMakeLists.txt.
2. Ejecutar el comando $ cmake . (incluyendo el punto). Esto generará el archivo Makefile.
3. Ejecutar el comando $ make TARGET donde TARGET es uno de los targets mencionados anteriormente. Esto creará un ejecutable con el nombre del target en el directorio actual.
4. Ejecutar el comando $ ./TARGET siendo TARGET el nombre del target utilizado anteriormente. Esto correrá el ejecutable.

Ejercicio 1

Convertir la función `int cuadrado(int)` en una función genérica que dependa de un parámetro `class T` (usando `template<class T>`) de tal manera que permita operar sobre cualquier tipo numérico. Para esto, crear un archivo Templates.hpp en la carpeta src.

Ejercicio 2

Adaptar la funcion `bool contiene(string s)` a una función genérica con dos parámetros, `class Contenedor` y `class Elem` (usando `template<class Contenedor, class Elem>`). Asumir que se encuentran definidos los métodos `Elem Contenedor::operator[](int) const` y `int Contenedor::size() const`. Por ejemplo, si `Contenedor` es `std::string`, el tipo `Elem` es `char`, y si `Contenedor` es `std::vector<int>`, el tipo `Elem` es `int`.

Ejercicio 3

Definir la función genérica `bool esPrefijo(Contenedor a, Contenedor b)` que devuelve `true` si y sólo si `a` tiene a lo sumo tantos elementos como `b` y `a` es prefijo de `b`. La función depende de un parámetro `class Contenedor`. Asumir que se encuentran definidos los métodos:

- `int Contenedor::size() const`
- `Elem Contenedor::operator[](int) const` — para algún tipo `Elem` apropiado, que puede depender de `Contenedor`.

Ejercicio 4

Definir la función genérica `Elem maximo(Contenedor c)` que devuelve el elemento más grande de `c`. Se asume que `c` tiene al menos un elemento y que se encuentran definidos los métodos:

- `int Contenedor::size() const`
- `Elem Contenedor::operator[](int) const`
- `bool operator<(Elem, Elem)`

Ejercicio 5

Los archivos `Diccionario.h` y `Diccionario.cpp` proveen una implementación básica de diccionario representado sobre un arreglo de asociaciones. Sus claves y valores son ambos de tipo `int`.

Adaptar la implementación de la clase `Diccionario` para que sea genérica y dependa de dos parámetros `class Clave` y `class Valor`. Para esto, crear un nuevo archivo `Diccionario.hpp` en la carpeta `src/`.

Ejercicio 6

En el archivo `tests/test_diccionario.cpp` hay un caso de test `dicc_int_int` para comprobar que el diccionario funciona con claves de tipo `int` y valores de tipo `int`. Agregar un caso de test `dicc_string_bool` para comprobrar que el diccionario funciona con claves de tipo `std::string` y valores de tipo `bool`.

Ejercicio 7

Crear un nuevo archivo `Multiconjunto.hpp` en la carpeta `src/` que incluya una clase genérica `Multiconjunto` que dependa de un parámetro `class T`. La clase debe implementar un multiconjunto[1] con los siguientes métodos:

1. `Multiconjunto<T>::Multiconjunto()` — construye un multiconjunto vacío,
2. `void Multiconjunto<T>::agregar(T x)` — agrega una aparición del elemento `x`,
3. `int Multiconjunto<T>::ocurrencias(T x) const` — devuelve la cantidad de ocurrencias de `x` en el multiconjunto.

Para representar el multiconjunto, utilizar un diccionario con claves de tipo `T` y valores de tipo `int`.

Ejercicio 8

Agregar un método `std::vector<Clave> Diccionario<Clave,Valor>::claves() const` en la clase `Diccionario`, que devuelve un vector con todas las claves del diccionario. Las claves deben devolverse **ordenadas** de menor a mayor. Suponer que se cuenta con una función `bool operator<=(Clave, Clave)` que permite comparar claves.

Una manera de hacer esto es la siguiente:

1. Crear un vector v_1 que contenga todas las claves del diccionario, sin importar el orden, y un vector v_2 vacío.
2. Buscar la clave más chica en el vector v_1, sacarla del vector v_1 y agregarla al vector v_2.
3. Repetir el paso 2. hasta que el vector v_1 esté vacío. El vector de claves ordenadas queda en v_2.

Este método se conoce como *Selection sort*. Más adelante en la materia veremos otros métodos de ordenamiento.

Ejercicio 9

Agregar un método `bool Multiconjunto<T>::operator<=(Multiconjunto<T> otro) const` que devuelve verdadero si el multiconjunto actual está incluido en el multiconjunto `otro`. Un multiconjunto `m1` está incluido en `m2` si para todo `x` se tiene que `m1.ocurrencias(x) <= m2.ocurrencias(x)`. Para implementar esta operación, es necesario recorrer todas las claves del diccionario que se usa para representar el multiconjunto. Suponer que se cuenta con una función `bool operator<=(T, T)` que permite comparar elementos.

[1] La especificación del TAD Multiconjunto(α) se puede encontrar en el apunte de TADs básicos.