

Ejercitación – Preprocesamiento y compilación

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

- ejN, si $(N \in \{1, 2, 3\})$: compila los tests correspondientes al ejercicio N.

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

1. En una consola pararse en el directorio raíz del proyecto. En este debería haber un archivo CMakeLists.txt.
2. Ejecutar el comando `$ mkdir build` para crear el directorio build.
3. Ejecutar el comando `$ cd build` para moverse al directorio recién creado.
4. Ejecutar el comando `$ cmake ..` (incluyendo el punto). Esto generará el archivo Makefile.
5. 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.
6. Ejecutar el comando `$./TARGET` siendo TARGET el nombre del target utilizado anteriormente. Esto correrá el ejecutable.

Ejercicio 1

Extraer la clase `class Periodo` fuera del archivo Fecha.cpp a dos archivos Periodo.h y Periodo.cpp.

Ejercicio 2

Intentar compilar el target ej2. Si no compila, resolverlo.

Ejercicio 3

Extraer las declaraciones de tipo `typedef` y de meses (ej.: `const Mes ENERO = 1;`) a un archivo Meses.h.

Luego, extraer las declaraciones de `bool esBisiesto(int)` y `int diasEnMes(int, int)` en un archivo Funciones.h. Extraer las definiciones en Funciones.cpp.

Recordar agregar los `#include` necesarios.

Ejercitación – Testing

Completar los tests propuestos en los siguientes ejercicios en el archivo `tests/ej_testing.cpp`. Por la característica de estos ejercicios, no podrán verificarse simplemente corriendo un test de código. Por lo tanto, en la carpeta `tests` se encontrará el archivo `solucion_testing.cpp` con las soluciones propuestas por la cátedra. Una vez resueltos los ejercicios por ustedes, revisar y consultar.

Ejercicio 4: Suma

Escribir un test en la test suit `Aritmetica`, y de nombre `suma` que evalúe que sumar $15 + 7$ da 22.

Ejercicio 5: Potencia

Escribir un test en la test suit `Aritmetica`, y de nombre `potencia` que evalúe que potenciar 10 al cuadrado da 100. Para la función potencia, usar `float pow(x, y)` de la biblioteca standard que da de resultado x^y .

Ejercicio 6: Potencia

Escribir un test en la test suit `Aritmética` y de nombre `potencia_general` que evalúe que elevar todos los números entre -5 y 5 es equivalente a multiplicar el número por sí mismo.

Ejercicio 7: Map, obtener

Escribir un test en la test suit `Diccionario` y de nombre `obtener` que defina un diccionario de enteros a enteros (`map<int, int>`), defina un valor con un significado y revise que al obtener la clave se obtiene el significado definido.

Ejercicio 8: Map, definir

Escribir un test en la test suit `Diccionario` y de nombre `definir` que defina un diccionario de enteros a enteros (`map<int, int>`), verifique que una clave no está definida, luego la defina y verifique que ahora sí está definida (usando la operación `count`).

Ejercicio 9: Truco, inicio

Considerando la clase `Truco` vista anteriormente (y presente en `src/Truco.h`), escribir un test en la test suit `Truco` y de nombre `inicio` que cree una instancia de `Truco` y verifique que el puntaje de ambos jugadores es 0.

Ejercicio 10: Truco, buenas

Escribir un test en la test suit **Truco** y de nombre **buenas** que cree una instancia de **Truco** y verifique que:

- El jugador 1 no está en buenas.
- Al sumar 15 puntos al jugador 1, este sigue en malas.
- Al sumar 1 punto al jugador 1, ahora está en buenas.
- Al sumar 2 puntos al jugador 1, sigue en buenas.