Clase #04 de 27
Pruebas Unitarias con Assert

Abril 29, Jueves

Agenda para esta clase

- Pruebas Unitarias con Assert
- Trabajo Tipos

Pruebas Unitarias con Assert

Assert – Introducción (I)

- Ejemplo
  - `assert( 2 == 1+1 );`
- ¿Qué necesitamos para usarla?
  - `#include <cassert>`
  - Define `assert` como una macro con parámetros, similar a una función
- ¿Para qué sirve?
  - Para realizar comprobaciones internas, verificar y confirmar nuestro trabajo
  - Para aplicar TDD (Test Driven Development) y realizar pruebas automáticas
  - Durante el desarrollo, para validar aximoas (i.e., situaciones que siempre van a ser verdad) y que si no lo son significa que el desarrollo no está completo o correcto
  - Para documentar situaciones lógicamente imposibles y que si ocurren, hay algo fundamentalmente mal en el programa.
- ¿Cuál es su sintaxis?
  - `assert( expresión );`
  - Expresión es generalmente una afirmación, es decir una expresión booleana que vale true
  - La afirmación toma forma de ecuaciones e inecuaciones
    - $1 > 0$
    - $1 < 2$
    - $2+1 = 3$
    - $5 = 2+3$
    - $5 \neq 1+2$
    - $1 \neq 3+1$
  - También de expresiones booleanas como
    - edad > 18 and nacionalidad = argentina
    - not EsPrimo(n)
- ¿Qué hace?
  - En tiempo de ejecución, `assert` evalúa la expresión, si es verdadera no hace nada.
    Pero si es falso, aborta el programa con un mensaje de error transcribiendo la expresión que dio falsa y su lugar del programa.

Assert – Introducción (II)

- ¿Para qué no se usa?
  - Validar datos recibidos de usuario, archivos u otros sistemas. Por ejemplo:
    - ¿El formato de fecha recibido es el esperado?
  - Para evaluar condiciones que es natural que puedan ser verdaderas o falas. Por ejemplo:
    - ¿Es un cliente con descuento?
  - Para manejo de errores o excepciones. Al abortar el programa, no permiten recuperación, su mensaje de error es críptico, y pueden estar deshabilitadas en producción
    - ¿Se cortó la conexión?
- ¿Qué hacer cuando todas las pruebas dan bien y el desarrollo finaliza?
  - Las comprobaciones de `assert` demandan tiempos de ejecución
  - Si las pruebas finalizaron correctamente, es ineficiente que nuestro programa valide axiomas en cada ejecución en un ambiente productivo.
  - Necesitamos una forma de deshabilitar la ejecución de `asserts`.
  - Comentar o borrar los `asserts` puede introducir nuevos errores.
  - Para eso está el flag `NDEBUG` (*no debug*, no depurar), que si está presente antes de inlcuir cassert omite los `asserts`.
  - Ese flag se puede establecer en el propio código, pero es más intrusivo y flexible establecerlo o no al iniciar el proceso de creación del ejecutable con la opción -D.

30.4.2 Assertions

The standard provides:

Assertions (§iso.7)	
static_assert(e,s)	Evaluate e at compile time; give s as a compiler error message if !e
assert(e)	If the macro NDBUG is not defined, evaluate e at run time and if !e, write a message to cerr and abort(); if NDBUG is defined, do nothing

For example:

```
template<typename T>
void draw_all(vector<T*>& v)
{
        static_assert(Is_base_of<Shape,T>(),"non-Shape type for draw_all()");

        for (auto p : v) {
                assert(p!=nullptr);
                // ...
        }
}
```

The assert() is a macro found in <cassert>. The error message produced by assert() is implementation-defined but should contain the source file name (__FILE__), and the source line number (__LINE__) containing the assert().

Asserts are (as they should be) used more frequently in production code than in small illustrative textbook examples.

https://en.cppreference.com/w/cpp/error/assert
https://en.wikipedia.org/wiki/Assertion_%28software_development%29

Términos de la clase #04

Definir cada término con la bibliografía

- Pruebas Unitarias con Assert
  - assert
  - cassert
  - Pruebas automáticas
  - Comprobaciones internas
  - TDD (Test Driven Development)
  - NDEBUG

Tareas para la próxima clase

1. Entregar trabajo “Ejemplos de Valores y Operaciones de Tipos de Datos”

¿Consultas?

Fin de la clase