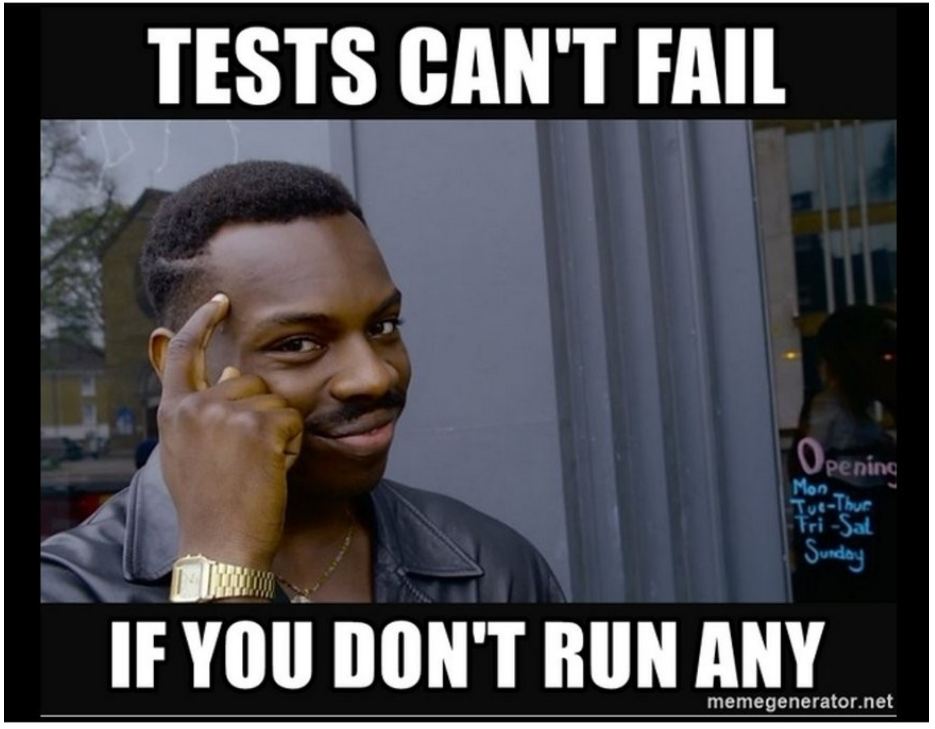
Семинар 8 по ООП

Exception handling, Unit тестване



Exception Handling

- Защото, ако нещо може да се обърка, то ще се обърка!
- Следствие: Ако знаеш, че нещо може да се обърка по 4 начина и си се подготвил, то непременно ще се появи пети, където въобще не си очаквал

- Exception-и възникват, когато се случи нещо, което пречи на програмата да продължи нормалния си ход и от нас зависи след като се появи такъв, как ще го обработим и как ще възстановим програмата.
- Когато възникне Exception програмата гледа дали текущата функция го обработва, ако не го подава на извикващата я. Ако и тя не го обработи продължава нагоре по веригата и накрая, ако никой не го обработи програмата спира.

Try / Catch / Throw

- Try блок се използва за да се огради мястото, където може да възникне Exception(нещата да се объркат).
- Ако в този блок възникне Exception той се предава на Catch блоковете
- Catch блоковете улавят даден тип Exception-и като трябва да са наредени от частен случай, към общ.
- Можем да създадем наши собствени Exception-и и също да хвърляме низове. Това става чрез throw.

Пример

```cpp
#include <iostream>
#include <exception>

void sthThatThrows()
{
    throw std::exception();
}
int main()
{
    try
    {
        sthThatThrows();
    }
    catch(std::exception& e)
    {
        std::cout << "Catch block" << std::endl;
    }
    std::cout << "Normal flow" << std::endl;
}
```

Популярни exception-и

- logic_error: invalid_argument, out_of_range, domain_error, length_error
- runtime_error: range_error, overflow_error, underflow_error
- bad_cast
- Всеки може да си прави свои exception-и

Unit Tests

- В практиката хора като нас редовно пишат големи количества код, но трябва и да сме сигурни, че този код работи както се очаква.
- Има два начина, по които можем да гарантираме, че кодът работи: Тестване и Формално доказателство
- Масово се предпочитат тестовете, защото са лесни за писане и позволяват разширяване. (НАСА е изключение по обясними причини...)

Какво представляват Unit тестовете

- Тестват малка част от кода (тази която разработвате в момента).
- Представляват извикване на функциите, които имплементирате с примерни данни и следене дали поведението им е както се очаква в спецификацията.

Test driven development

- Методология, при която преди започване на работа по кода на даден софтуер, изискванията се превеждат до тестове.
- Три основни фази: Red, Green, Refactor
- Тестовете често се използват като документация на даден код, защото хората не обичат да пишат документация, а те предоставят информация за поведението на дадена функция и колкото повече са, толкова по-детайлна информация имаме.

Doctest

- За писане на тестове ще използваме библиотеката Doctest.
- https://github.com/doctest/doctest/
- Doctest е свободна библиотека за тестване, която предоставя много прост интерфейс за нашите цели

Инициализация

- #include <doctest.h>
- #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN – ако искаме предефинирания от библиотеката main
- #define DOCTEST_CONFIG_IMPLEMENT – ако искаме да напишем свой main

Писане на тестове

- След инициализацията можем да пристъпим към писането на тестове.
- Това става с макроса TEST_CASE, който приема име на теста, като е добре в един тест да се тества една функционалност, а отделните примери за тази функционалност да са в SUBCASE

Макроси

- За да проверим нещо ще използваме CHECK. Той приема булев израз и проверява дали е истина.
- Има още много, като най-честите са:

  CHECK_EQ, REQUIRE, CHECK_THROWS, CHECK_THROWS_AS, CHECK_NOTHROW

Време за пример:

Въпроси?

Задачи

- Напишете Unit-тестове за свързания списък от предния път
- Задача Car (Г. Атанасов): Да се реализира клас Car, който представлява кола, която има марка, модел, идентификационен номер (номер на рамата), средна скорост и изминато разстояние. Идентификационният номер на всяка кола се определя от реда ѝ на създаване. Да се реализира метод drive, който пресмята за колко часа колата ще измине подаденото разстояние. След всяко извикване на drive да се актуализира изминатото разстояние. Освен това за всеки 100 изминати километра, средната скорост на колата намалява с 1 км/час, но не може да падне под 50% от първоначалната скорост. Да се реализират оператор ==, който проверява дали 2 коли са от един и същи модел и марка, и операторите за вход и изход. Да се напишат автоматизирани тестове за класа, без операторите за вход и изход.