My Project

Generated by Doxygen 1.9.1

Chapter 1

Laboratorium 1 - ułamki

1.0.1 Tresc zadan dla Panstwa (aktualniejsza jest w <tt>README.md</tt>)

Zadanie 0: absolutnie obowiazkowe, chociaz bez punktow

1. Pierwsza rzecza jest poprawa bledow kompilacji, czyli wpisanie poprawnych Panstwa danych w pliku: main.↩ cpp
2. Oddanne zadanie musi sie bezwzglednie kompilowac na systemie Linux:
   - Jesli sie nie skompiluje to jest to 0 punktow za zadanie!
   - Oczywiscie w razie problemow z kompilacja prosze się zgłaszać/pisać.
   - Dobrze, jesli nie byloby warningow kompilacji, ale za to nie obnizam punktow.
   - Aby się upewnić, że się kompiluje można skorzystać z `narzędzia Bobot`
3. Oddane zadanie nie powinno crashować na żadnym teście, jeśli crashuje proszę zrobic implementacje -fake, ktora nie dopusci do crasha nawet jesli test bedzie failowal, ale za to testy nie beda sie crashowaly. W przypadku crasha biorę pod uwagę tylko tyle testów, ile przejdzie do czasu crasha!
4. Mam program antyplagiatowy, dlatego prosze pracowac samodzielnie!
   - Osoby ktore udostepniaja swoje rozwiazania rowniez beda mialy kare!
   - Na ukaranie prowadzący ma czas 2 tygodnie po terminie oddania, czyli nawet jak ktoś otrzyma punkty wcześniej ma pewność, że za oszustwa/łatwowierność dosięgnie go niewidzialna ręka sprawiedliwości.
5. Zadanie z zalozenia bedzie sprawdzane automatycznie, przez testy jednostkowe dostepne w pliku: fraction↩ Tests.cpp,
6. ∗Dobrze jakby nie bylo warningow kompilacji (flagi: `-Wall -Wextra -pedantic -Werror`, a dla hardcorów jeszcze: `-Weffc++`)
7. Punkty beda odejmowane za wycieki pamieci (jest podpiety `valgrind`)
8. Niewykluczone jest sprawdzanie reczne - zaleznie od prowadzacego dana grupe.

Tresc do implementacji - szukaj w plikach ∗.h

Chapter 2

Ułamek

2.1 Implementacja prostej klasy - Ułamek

W zadaniu chodzi o to, żeby zaimplementować wg treści podaną klasę `Fraction` wraz z metodami podanymi w treści. Tym sposobem chcemy wejść w świat programowania obiektowego w C++ robiąc pierwszą prostą klasę.

2.1.1 Utrudnienie - makra preprocesora

Zadanie zawiera testy automatyczne, które testują czy dana metoda jest zaimplementowana poprawnie. Jeśli metody nie ma a byłaby testowana to byłby błąd kompilacji. Tym samym zadanie można by oddać wyłącznie jako wszystko lub jako nic. Dlatego też korzystam z makr preprocesora, które zależnie od tego czy dana funkcjonalność jest zaimplementowana czy nie aktywują (w trakcie kompilacji) odpowiednie części kodu. Czyli Państwo po zaimpelemtowaniu danej metody będą musieli zmazać odpowiednią linijkę `UNIMPLEMENTED_abcFunction`.

2.1.2 Do zrobienia

Zaimplementuj podaną na zajęciach klasę reprezentującą ulamek:

1. Nazwa klasy `Fraction`.
   (a) Następnie usuń makro: `UNIMPLEMENTED_classFraction`
2. Do klasy dodaj pola protected typu całkowitego:
   - `numerator_` (licznik)
   - `denominator_` (mianownik).

   (a) Następnie usuń makro `UNIMPLEMENTED_numeratorAndDenominator`
3. Do klasy dodaj konstruktor bezparametrowy (może być też z wartościami domyślnymi), który ustawi wartości licznika na `0` i mianownika na `1`.
   (a) Następnie usuń makro `UNIMPLEMENTED_defaultConstructor`
4. Do klasy dodaj konstruktor z parametrami (może być modyfikacja powyższego), który ustawi licznik i mianownik na podstawie podanych argumentów funkcji,
   - wartością domyślną dla mianownika ma być `1`.

   (a) Następnie usuń makro `UNIMPLEMENTED_constructorWhichInitialiseFields`

5. Do klasy dodaj zestaw metod dostępowych/modyfikujących obiekty klasy -tzw. gettery i settery, które umożliwią modyfikacje i pobranie wartości licznika i mianownika

   (a) Następnie usuń makro `UNIMPLEMENTED_gettersAndSetters`,

6. Do klasy dodaj metodę `print()`, wypisującą atrybuty obiektu na konsolę w formie `"licznik/mianownik\n"`

   (a) Następnie usuń makro `UNIMPLEMENTED_printFunction`

7. Do klasy dodaj statyczny atrybut `removedFractions_` i metodę statyczną: `removedFractions()` zwracającą ten atrybut Skladowa ta powinna byc incrementowana w destruktorze.

   (a) Następnie usuń makro `UNIMPLEMENTED_counterOfDestructedFractions`

8. Dodaj do klasy metody `zapisz(std::ostream& os)` i `wczytaj(std::istream& is)`, zapisujące/odczytujące zawartość obiektu do przekazanego jako parametr strumienia wyjściowego/wejściowego, w formacie `"licznik/mianownik"`. Metoda wczytująca może założyć, że format danych będzie poprawny (nie trzeba obslugiwac bledow).

   (a) Następnie usuń makro `UNIMPLEMENTED_readWrite`.

9. Proszę dodanie stałej odzwierciedlającej nazwę ułamka o nazwie `fractionName_`, powinna być `protected`. Do niej proszę dodać metodę stałą: `getFractionName()`.

   (a) Następnie usuń makro `UNIMPLEMENTED_fractionNameSettableFromConstructor`

10. Do klasy dodaj dwie stałe:

    - stała statyczna odzwierciedlajaca niedopuszczalna wartość mianownika: `invalidDenominator↩Value`
    - metodę statyczną zwracającą powyższe: `getInvalidDenominatorValue()`
    - stałą statyczną czasu kompilacji (constexpr) odzwierciedlajaca domyślna wartość mianownika↩: `defaultDenominatorValue`
    - metodę constexpr `getDefaultDenominatorValue()` zwracającą powyższe.

    (a) Następnie usuń makro `UNIMPLEMENTED_fractionConstStaticFields`
    Po implementowaniu powyższych poleceń i zmiany wartości poniższych makr powinno przechodzić coraz więcej testów dostępnych w pliku `fractionTests.cpp`.

2.2 Uwaga:

Wszystkie atrybuty powinny być prywatne, konstruktory i metody - publiczne, metody większe niż 1-linijkowe powinny być zadeklarowane w klasie, zdefiniowane poza klasą, obiekty typów klasowych powinny być w miarę możliwości przekazywane w argumentach funkcji przez referencję, proszę też stosować słówko "const" w odpowiednich miejscach.

Mozna tworzyc dowolna ilosc metod pomocniczych, jednakze aby byly one prywatne.

2.3 Ocenianie:

1. Ocenia `Bobot`, na ten moment w następujący sposób:
   (a) Kompilacja nadesłanego rozwiązania - bez tego zero punktów. Bobot pracuje na Linuxie, używa kompilatora g++.
   (b) Uruchamianie testów - za każdy test, który przejdzie są punkty, ale mogą być odjęte w kolejnych krokach.
   (c) Jeśli program się wywala na którymś z testów (to się pojawia często u osób pracujących na Windowsie - ten system pozwala pisać po nie-swojej pamięci, Linux nie pozwala) lub jest timeout - wtedy będzie przyznane tyle punktów ile przechodzi testów **minus dwa za karę**.
   (d) Jest odpalane narzędzie `valgrind`, które sprawdza czy umiemy obsługiwać pamięć w praktyce - jeśli nie to **minus punkt**.
   (e) Odpalane są też inne narzędzia takie jak `cppcheck`, czy `fawfinde` i inne. One nie odejmują punktów, no ale mają pomóc w pisaniu porządnych programów. Nie olewajmy tego.
   (f) Antyplagiat - za wykrycie plagiatu (jest specjalne narzędzie) otrzymuje się 0 punktów. Róbmy więc samemu!

2.4 Najczestrze pytania/błędy/problemy:

1. Są rozbieżności między treścią `README.md`, a treściami w plikach nagłówkowych!
   - W tym roku przechodzimy na `README.md`, więc ta treść jest wiążąca.
2. Zrobiłem wg treści i się nie kompiluje.
   - Poza samym robieniem wg instrukcji należy pamiętać, że piszemy w C++, nie podaje dokładnych typów zwracanych funkcji, a trzeba pamiętać o nich definiując sygnaturę funkcji.
3. Podczas kompilacji pojawia się dziwny błąd z tekstem `removedFractions_` i dużą ilością dziwnych znaków, mimo iż został zdefiniowany w klasie.
   - Zapewne chodzi o błąd linkowania - kod się kompiluje, ale trzeba go jeszcze połączyć w binarkę. C++ wymaga aby składowa statyczna (równocześnie nie `const`/`constexpr`) była zdefiniowana poza klasą. Alternatywnie można dołożyć słówko `inline` (to dodał C++17).

2.5 Pytania po implementacji ćwiczenia:

1. Jaka jest różnica między składowymi: `const static` a `static`?
2. Jaka jest różnica między składowymi: `const static` a `constexpr static`?

2.6 Zadania, które warto zrobić (uwaga: nie będzie za to punktów, tylko coś cenniejszego - umiejętności)

1. Warto rozbudować naszą klasę-ułamek o podstawowe operacji na ułamkach takich jak mnożenie, dodawanie, skracanie.

2.7 Jak skonfigurować sobie pracę nad paczką:

W formie `wideo`.
Alternatywnie poniżej jest to spisane w kolejnej sekcji

2.7.1 Grading (section copied from Mateusz Ślażyński, of course he agreed):

- [ ] Make sure, you have a **private** group
  - how to create a group
- [ ] Fork this project into your private group
  - how to create a fork
- [ ] Add @bobot-is-a-bot as the new project's member (role: **maintainer**)
  - how to add an user

2.7.2 How To Submit Solutions

1. [ ] Clone repository: `git clone` (clone only once the same repository):
   ```bash git clone <repository url> ```
2. [ ] Solve the exercises
3. [ ] Commit your changes
   ```bash git add <path to the changed files> git commit -m <commit message> ```
4. [ ] Push changes to the gitlab main branch
   ```bash git push -u origin main ```

The rest will be taken care of automatically. You can check the `GRADE.md` file for your grade / test results. Be aware that it may take some time (up to one hour) till this file. Details can be found in `./logs/` directory where You can check compilation results, tests logs etc.

2.7.3 Project Structure

```
.
 zaj1Fraction                # directory containing exercises
|    CMakeLists.txt          # CMake configuration file - the file is to open out project in our IDE
|    main.cpp                # main file - here we can test out solution manually, but it is not required
|    fraction.h              # file to create class declaration and methods' declaration
|    fraction.cpp            # file to implement methods
|    trescPdf.pdf            # documentation in PDF (generated by Doxygen)
|    tests                   # here are tests for exercise, inner CMakeLists.txt, GTest library used by tests
       CMakeLists.txt      # iner CMake for tests - it is included by outter CMake
       fractionTests.cpp   # files with tests for exercise
       lib                 # directory containing GTest library
|    Doxyfile                # here is prepared file for Doxygen, to generate documentation when we type 'doxy
|    doxyfiles               # here is logo for documentation generated by Doxygen
|    Dockerfile              # this file contains instructions how to run tests in embedded Ubuntu
|    README.md               # this file
```
```

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

File Documentation

4.1 fraction.h File Reference

Zaimplementuj podaną na zajęciach klasę reprezentującą ulamek:

```
#include <iosfwd>
#include <string>
```

Include dependency graph for fraction.h: