My Project

Generated by Doxygen 1.8.17

Chapter 1

Laboratorium 3 - Vector agregujacy Fraction

1.0.1 Tresc zadan dla Panstwa:

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 online judge` (VPN AGH konieczny). Aby wyslac zadanie nalezy wybrac odpowiednie dla zajec: konkurs (context), problem, oraz jezyk programowania. prosze zalaczyc pliki fraction.h, fraction.cpp i vector.h i vector.cpp
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: matrix↩ 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

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 Fraction Class Reference

The documentation for this class was generated from the following file:

- fraction.h

4.2 Vector Class Reference

```
#include <vector.h>
```

4.2.1 Detailed Description

---------- **********

Nasza implementacja w razie automatycznego zwiekszania rozmiaru ma allokowac pamiec tylko o 1 wieksza!

```
Deklaracje klasy powinny znaleźć się w odpowiednich plikach nagłówkowych, definicje metod i konstruktorów - w
Wszystkie atrybuty powinny być prywatne, konstruktory i metody - publiczne, metody/konstruktory/destruktory wi

Obiekty typów klasowych powinny być przekazywane do funkcji/metod przez referencje (zwykłe lub stałe), metody
Wszystkie metody, które mogą być stałe proszę aby były
```

The documentation for this class was generated from the following file:

- vector.h

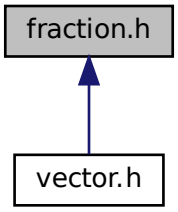
Chapter 5

File Documentation

5.1 fraction.h File Reference

Bardziej zlozona implementacja klasy Fraction (Ulamek):

This graph shows which files directly or indirectly include this file:



Classes

- class Fraction

5.1.1 Detailed Description

Bardziej zlozona implementacja klasy Fraction (Ulamek):

1. Klasa powinna posiadac pola `numerator_` (licznik) i `denominator_` (mianownik). Najlepiej aby byly to zmienne calkowite.
2. Powinna zawierac jeden konstruktor ustawiajacy licznik (domyslnie na 0) i mianownik (domyslne na 1)
3. Gettery i settery do wartosci licznika i mianownika m.in.: `denominator()` i `setDenominator(...)`.

- Prosze pamietac, ze gettery, jako metody nic nie zmieniajace powinny byc oznaczone jako metody stale.
- W mysl zasady aby w razie potrzeby kod modyfikowac w mniejszej ilosci miejsc sugeruje aby typem zwracanym getterow bylo `auto`.

4. `operator+` dla ulamka zwracajacy ulamek przez kopie. Metoda stala.
5. `operator*` dla ulamka zwracajacy ulamek przez kopie. Metoda stala.
6. Niepoprawny mianownik (=0) powinien byc zglaszany przez wyjatek `std::invalid_argument`. Dotyczy to wszystkich miejsc, gdzie jest ustawiany mianownik.
7. Prosze o automatyczne skracanie ulamkow po operacji + i * Pomocny moze sie okazac algorytm euklidesa, oczywiscie tutaj robimy tylko dla przypadkow dodatnich. Zachecam do uzycia `std::gcd(...)`.

5.1.1.1 Najczestrze pytania:

1. Czy w setterach skracać ułamki? Setter swoją nazwą mówi -ustawX, więc powinien to zrobić i nic więcej. Trochę dziwne byłoby zachowanie gdy użytkownik ustawia 1/4 na 2/4 i by nagle się mu zrobiło 1/2, mimo iż ustawiał tylko licznik na 2.

5.1.1.2 Uwaga:

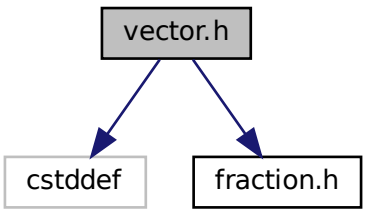
Wszystkie atrybuty powinny być prywatne, konstruktory i metody - publiczne, metody większe niż 1-linijkowe powinny być zadeklarowane w klasie, zdefiniowane poza klasą, obiekty typów klasowych powinny być w miarę możliwości przekazywane w argumentach funkcji przez referencję, proszę też stosować słówko "const" w odpowiednich miejscach.
Mozna tworzyc dowolna ilosc metod pomocniczych, jednakze aby byly one prywatne.
Gettery i settery operujace na liczbach, ktore nie rzucaja wyjatku, warto zadeklarowac jako `noexcept`.

5.2 vector.h File Reference

Klasy Vector zarzadzająca dynamiczną tablicą na elementy i rozszerzająca się wg potrzeb z obsługą wyjątków.

```
#include <cstddef>
#include "fraction.h"
```

Include dependency graph for vector.h:



Classes

- class Vector

5.2.1 Detailed Description

Klasy Vector zarzadzająca dynamiczną tablicą na elementy i rozszerzająca się wg potrzeb z obsługą wyjątków.

Note

UWAGA: To bardzo wazne zadanie, jeśli ktoś chce być programista C++ to w srodku nocy powinien umiec takie zadania robic!

Nasza implementacja wzorowana C++owym `std::vector`, ale wystepuja różnice.

Nie wolno uzyc w srodku std::vector! Zaawansowani moga uzyc uzyc inteligetnych wskaznikow, jesli chca.

5.2.1.1 Tresc zadania:

1. Proszę aby klasa miała następujące składowe:
   - `Fraction* data_` - dynamiczna tablica na dane. Osobom zaawansowanym sugeruję użyć inteligętnych wskaźników np. `std::unique_ptr<Fraction[]> data_**`
   - `std::size_t size_` - aktualna ilość elementów na tablicy
   - `std::size_t capacity_` - ile elementow pomieści aktualnie zaalokowana tablica.
2. Proszę o zaimplementowanie metod - getterów zwracających powyższe składowe - `size()`, `capacity()`, `data()`.
3. Proszę o zaimplementowanie konstuktora przyjmującego liczbę do wstępnej allokacji (z wartoscia domyslna 0)
4. Proszę o zaimplementowanie destruktora. Musi on koniecznie zwalniać pamięć (chyba, że używamy inteligętnych wskaźników, wtedy się zwolni automatycznie i nie musimy go implementować).
5. Proszę o zdefiniowanie konstuktora kopiującego, który będzie wykonywał tzw. "głęboką kopię" (czyli allokował nową pamięć i kopiował zawartość). Osoby zaawansowane mogą to rozwiązać przez copy-on-write.**
6. Proszę o zdefiniowanie `operator=` wersji kopiującej głęboko i przenoszącej
7. Proszę zdefiniować metodę dodającą obiekt na koncu tablicy `push_back()`. W razie braku miejsca metoda ta powinna dokonać reallokacji pamięci aby nowy element się zmieścił.**
8. Proszę o zdefiniowanie operatora indeksowania: `operator[](std::size_t index)` zwracający wskazany element tablicy. Dostep po indeksie poza rozmiar tablicy (size) powinny być zgłaszane poprzez wyjątki `std::out_of_range**`

   Note

   Proszę pamiętać, że należy zdefiniować dwie wersje tego operatora - constową i zwykłą.

5.2.1.2 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.

Note

Co się da na listę inicjalizacyjną konstruktora.

Za złe zarządzanie pamięcią (wycieki, pisanie poza pamięcią) mogą być odejmowane punkty

Obiekt, z którego przenosimy też powinien się nadawać do użytku!

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

5.2.1.3 Podpowiedzi:

- polecam uzycie operatora :?
- mozna allokowac zero elementor: new int[0];
- dla wygody mozna zastosowac idiom: `copy&swap`, podkreslam jednak, ze jest to mniej wydajne

Index