Kolokwium/

Generated by Doxygen 1.9.1

Chapter 1

Laboratorium 3 - Vector agregujacy Fraction

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: 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

Vector

2.1 Klasa Fraction:

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(...)`.

2.2 Najczestrze pytania do klasy Fraction:

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.

2.3 Klasa Vector:

Klasy Vector zarzadzająca dynamiczną tablicą na elementy

- i rozszerzająca się wg potrzeb z obsługą wyjątków.
- **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.

2.4 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`**
   - Proszę pamiętać, że należy zdefiniować dwie wersje tego operatora - constową i zwykłą.

Informacje o co chodzi w paczce, na co zwrócić uwagę, jak czytać testy znajdują się w materiale `wideo`.

2.5 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. Wszystkie metody, które mogą być stałe proszę aby były

1. Mozna tworzyc dowolna ilosc metod pomocniczych, jednakze aby byly one prywatne.
2. Gettery i settery operujace na liczbach, ktore nie rzucaja wyjatku, warto zadeklarowac jako `noexcept`.
3. Co się da na listę inicjalizacyjną konstruktora.
4. Za złe zarządzanie pamięcią (wycieki, pisanie poza pamięcią) powodują odejmowanie punktów
5. Obiekt, z którego przenosimy też powinien się nadawać do użytku!

`Bardziej szczegółowe informacje jak pisać programy w ładnym stylu` dla zaawansowanych.

2.6 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
- Nasza implementacja w razie automatycznego zwiekszania rozmiaru ma allokowac pamiec tylko o 1 wieksza!
  - Nie powinno sie tak robic!
- Deklaracje klasy powinny znaleźć się w odpowiednich plikach nagłówkowych, definicje metod i konstruktorów - w plikach źródłowych.

2.7 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!

2.8 Najczestrze pytania/błędy/problemy:

1. Może się pojawią...

2.9 Pytania po implementacji ćwiczenia:

1. (Jak macie pomysł to podrzućcie)

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

1. (Jak macie pomysł to podrzućcie)

2.11 Jak skonfigurować sobie pracę nad paczką:

W formie `wideo` do poprzedniej paczki (link do projektu inny, reszta analogiczna).
Alternatywnie poniżej jest to spisane w kolejnej sekcji

2.11.1 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`

2.11.2 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.

2.11.3 Project Structure

```
.
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 implement class declaration
fraction.cpp            # file to implement methods
vector.h                # another file to implement class declaration
vector.cpp              # another 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
   lib                 # directory containing GTest library
   fractionTests.cpp   # tests v1
   vectorTests.cpp     # tests v1
doxyfiles               # here is logo for documentation generated by Doxygen
   cppLogo.png         # logo
Doxyfile                # here is prepared file for Doxygen, to generate documentation when we type `doxygen
Dockerfile              # this file contains instructions how to run tests in embedded Ubuntu
README.md               # this file
```

Chapter 3

Todo List

Member FIRSTNAME

Uzupelnij swoje dane:

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

MyOwnMemoryManagement . . . . . ??

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- Fraction . . . . . **??**
- testing::Test
  - FractionTester . . . . . **??**
  - VectorTest . . . . . **??**
- Vector . . . . . **??**

Chapter 6

Class Index

6.1 Class List

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

Fraction . . . . . ??
FractionTester . . . . . ??
Vector . . . . . ??
VectorTest . . . . . ??

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

fraction.cpp . . . . . ??

fraction.h

Bardziej zlozona implementacja ułamka na podstawie klasy Fraction (bardziej szczegolowa tresc w README.md): . . . . . ??

main.cpp . . . . . ??

vector.cpp . . . . . ??

vector.h

Klasy Vector zarzadzająca dynamiczną tablicą na elementy (bardziej szczegolowa tresc w README.md): i rozszerzająca się wg potrzeb z obsługą wyjątków . . . . . ??

tests/fractionTests.cpp . . . . . ??

tests/vectorTests.cpp . . . . . ??

Chapter 8

Namespace Documentation

8.1 MyOwnMemoryManagement Namespace Reference

Variables

- unsigned deletions = 0

8.1.1 Variable Documentation

8.1.1.1 deletions

```
unsigned MyOwnMemoryManagement::deletions = 0
```

Definition at line 19 of file vectorTests.cpp.

Chapter 9

Class Documentation

9.1 Fraction Class Reference

`#include <fraction.h>`

9.1.1 Detailed Description

Definition at line 42 of file fraction.h.
The documentation for this class was generated from the following file:

- fraction.h

9.2 FractionTester Struct Reference

Inheritance diagram for FractionTester:
Collaboration diagram for FractionTester:

9.2.1 Detailed Description

Definition at line 17 of file fractionTests.cpp.
The documentation for this struct was generated from the following file:

- tests/fractionTests.cpp

9.3 Vector Class Reference

`#include <vector.h>`

9.3.1 Detailed Description

---------- ********************
Nasza implementacja w razie automatycznego zwiekszania rozmiaru ma allokowac pamiec tylko o 1 wieksza! Ale w implementacjach profesjonalnych nie powinno tak być!
Definition at line 69 of file vector.h.
The documentation for this class was generated from the following file:

- vector.h

9.4 VectorTest Class Reference

Inheritance diagram for VectorTest:

Chapter 10

File Documentation

10.1 CMakeLists.txt File Reference

10.2 tests/CMakeLists.txt File Reference

Functions

- project (tests) add_subdirectory(lib) include_directories($

10.2.1 Function Documentation

10.2.1.1 project()

```
project (
            tests  )
```

Definition at line 1 of file CMakeLists.txt.

10.3 fraction.cpp File Reference

```
#include <iostream>
#include <stdexcept>
#include <numeric>
#include "fraction.h"
```

Include dependency graph for fraction.cpp:

10.4 fraction.h File Reference

Bardziej zlozona implementacja ułamka na podstawie klasy Fraction (bardziej szczegolowa tresc w `README.md`):
This graph shows which files directly or indirectly include this file:

Classes

- class Fraction

10.4.1 Detailed Description

Bardziej zlozona implementacja ułamka na podstawie klasy Fraction (bardziej szczegolowa tresc w `README.md`):

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(...)`.

10.4.1.1 Uwaga (bardziej wiazaca tresc jest w pliku <tt>README.md</tt>):

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`.

10.4.1.2 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.

10.5 main.cpp File Reference

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

Include dependency graph for main.cpp:

Functions

- void validateStudentsInfo ()
- int main ()
- constexpr size_t compileTimeStrlen (const char *text) noexcept
- constexpr size_t compileTimeCountFirstDigits (const char *text) noexcept
- constexpr bool compileTimeIsDigit (const char *text) noexcept
- constexpr bool compileTimeContains (const char *text, char letter) noexcept

Variables

- constexpr const char *const FIRSTNAME = ""
- constexpr const char *const SURNAME = ""
- constexpr const char *const MAIL = ""
- constexpr const char *const BOOK_ID = ""
- constexpr const char *const TEACHER_MAIL = "bazior[at]agh.edu.pl"

10.5.1 Function Documentation

10.5.1.1 compileTimeContains()

```
constexpr bool compileTimeContains (
            const char * text,
            char letter )  [inline], [constexpr], [noexcept]
```

Definition at line 70 of file main.cpp.
Here is the caller graph for this function:

10.5.1.2 compileTimeCountFirstDigits()

```
constexpr size_t compileTimeCountFirstDigits (
            const char * text )  [inline], [constexpr], [noexcept]
```

Definition at line 60 of file main.cpp.
Here is the caller graph for this function:

10.5.1.3 compileTimeIsDigit()

```
constexpr bool compileTimeIsDigit (
            const char * text )  [inline], [constexpr], [noexcept]
```

Definition at line 65 of file main.cpp.
Here is the call graph for this function: Here is the caller graph for this function:

10.5.1.4 compileTimeStrlen()

```
constexpr size_t compileTimeStrlen (
            const char * text )  [inline], [constexpr], [noexcept]
```

Definition at line 55 of file main.cpp.
Here is the caller graph for this function:

10.5.1.5 main()

```
int main ( )
```

Definition at line 46 of file main.cpp.
Here is the call graph for this function:

10.5.1.6 validateStudentsInfo()

```
void validateStudentsInfo ( )
```

Definition at line 78 of file main.cpp.
Here is the call graph for this function: Here is the caller graph for this function:

10.5.2 Variable Documentation

10.5.2.1 BOOK_ID

```
constexpr const char* const BOOK_ID = ""  [constexpr]
```

Definition at line 39 of file main.cpp.

10.5.2.2 FIRSTNAME

```
constexpr const char* const FIRSTNAME = ""  [constexpr]
```

Todo Uzupelnij swoje dane:

Definition at line 36 of file main.cpp.

10.5.2.3 MAIL

```
constexpr const char* const MAIL = ""  [constexpr]
```
Definition at line 38 of file main.cpp.

10.5.2.4 SURNAME

```
constexpr const char* const SURNAME = ""  [constexpr]
```
Definition at line 37 of file main.cpp.

10.5.2.5 TEACHER_MAIL

```
constexpr const char* const TEACHER_MAIL = "bazior[at]agh.edu.pl"  [constexpr]
```
Definition at line 41 of file main.cpp.

10.6 README.md File Reference

10.7 tests/fractionTests.cpp File Reference

```
#include <iostream>
#include <sstream>
#include <gtest/gtest.h>
```
Include dependency graph for fractionTests.cpp:

Classes

- struct FractionTester

Functions

- TEST_F (FractionTester, constructionAndGetters_expectedValuesFromConstructorReadableByGetters)
- TEST_F (FractionTester, settersForMembers_expectedValuesChangesWithoutReduction)
- TEST_F (FractionTester, fractionMultiplyResultFractionImpossibleForReduction_expectedSimpleMultiplying↩ Working)
- TEST_F (FractionTester, fractionAdditionResultFractionImpossibleForReduction_expectedSimpleAddition↩ Working)
- TEST_F (FractionTester, settingInvalidDenominarrInConstructor_expectedThrow)
- TEST_F (FractionTester, fractionAdditionWithCommonDenominator_expectedReduction)

10.7.1 Function Documentation

10.7.1.1 TEST_F() [1/6]

```
TEST_F (
            FractionTester ,
            constructionAndGetters_expectedValuesFromConstructorReadableByGetters  )
```
1/2
4/1
Definition at line 21 of file fractionTests.cpp.

10.7.1.2 TEST_F() [2/6]

```
TEST_F (
            FractionTester ,
            fractionAdditionResultFractionImpossibleForReduction_expectedSimpleAddition↩
Working  )
```

Definition at line 68 of file fractionTests.cpp.

10.7.1.3 TEST_F() [3/6]

```
TEST_F (
            FractionTester ,
            fractionAdditionWithCommonDenominator_expectedReduction  )
```

Definition at line 90 of file fractionTests.cpp.

10.7.1.4 TEST_F() [4/6]

```
TEST_F (
            FractionTester ,
            fractionMultiplyResultFractionImpossibleForReduction_expectedSimpleMultiplying↩
Working  )
```

Definition at line 57 of file fractionTests.cpp.

10.7.1.5 TEST_F() [5/6]

```
TEST_F (
            FractionTester ,
            settersForMembers_expectedValuesChangesWithoutReduction  )
```

Definition at line 42 of file fractionTests.cpp.

10.7.1.6 TEST_F() [6/6]

```
TEST_F (
            FractionTester ,
            settingInvalidDenominarrInConstructor_expectedThrow  )
```

Definition at line 79 of file fractionTests.cpp.

10.8 tests/vectorTests.cpp File Reference

```
#include <gtest/gtest.h>
```

Include dependency graph for vectorTests.cpp:

Classes

- class VectorTest

Namespaces

- MyOwnMemoryManagement

Functions

- void * operator new[ ] (size_t size)
- void operator delete[ ] (void *memory2remove) noexcept

- void operator delete[] (void *memory2remove, size_t) noexcept
- TEST_F (VectorTest, constructionAndGetters_expectedValuesFromConstructorReadableByGetters)
- TEST_F (VectorTest, destructOfNotEmptyVector_expectedMemoryFreedOnce)
- TEST_F (VectorTest, addingMultipleElementsWithoutReallocation_expectedSizeIncreasesButCapacity↩ DoesNot)
- TEST_F (VectorTest, accessingWithIndexOperator_expectedProperElementsInProperPlaces)
- TEST_F (VectorTest, copyAndMoveConstructor_expectedSuccessfulCopyAndMove)
- TEST_F (VectorTest, assignmentOperatorCopyingAndMoving_expectedSuccessfulCopyAndMove)
- TEST_F (VectorTest, accessingWithIndexOperatorOutOfRange_expectedExceptionWhenOutOfRange)
- TEST_F (VectorTest, addingElementsWithReallocationNecessairy)

Variables

- unsigned MyOwnMemoryManagement::deletions = 0

10.8.1 Function Documentation

10.8.1.1 operator delete[]() [1/2]

```
void operator delete[] (
            void * memory2remove )  [noexcept]
```

Definition at line 28 of file vectorTests.cpp.

10.8.1.2 operator delete[]() [2/2]

```
void operator delete[] (
            void * memory2remove,
            size_t  )  [noexcept]
```

Definition at line 34 of file vectorTests.cpp.

10.8.1.3 operator new[]()

```
void* operator new[] (
            size_t size )
```

Definition at line 22 of file vectorTests.cpp.

10.8.1.4 TEST_F() [1/8]

```
TEST_F (
            VectorTest ,
            accessingWithIndexOperator_expectedProperElementsInProperPlaces  )
```

Definition at line 92 of file vectorTests.cpp.

10.8.1.5 TEST_F() [2/8]

```
TEST_F (
            VectorTest ,
            accessingWithIndexOperatorOutOfRange_expectedExceptionWhenOutOfRange  )
```

Definition at line 189 of file vectorTests.cpp.

10.8.1.6 TEST_F() [3/8]

```
TEST_F (
            VectorTest ,
            addingElementsWithReallocationNecessairy  )
```

Definition at line 207 of file vectorTests.cpp.

10.8.1.7 TEST_F() [4/8]

```
TEST_F (
            VectorTest ,
            addingMultipleElementsWithoutReallocation_expectedSizeIncreasesButCapacityDoesNot
)
```

Definition at line 78 of file vectorTests.cpp.

10.8.1.8 TEST_F() [5/8]

```
TEST_F (
            VectorTest ,
            assignmentOperatorCopyingAndMoving_expectedSuccessfulCopyAndMove  )
```

copying:
moving:
Definition at line 148 of file vectorTests.cpp.

10.8.1.9 TEST_F() [6/8]

```
TEST_F (
            VectorTest ,
            constructionAndGetters_expectedValuesFromConstructorReadableByGetters  )
```

default constructor
constructor with single argument
Definition at line 44 of file vectorTests.cpp.

10.8.1.10 TEST_F() [7/8]

```
TEST_F (
            VectorTest ,
            copyAndMoveConstructor_expectedSuccessfulCopyAndMove  )
```

copying:
moving:
Definition at line 107 of file vectorTests.cpp.

10.8.1.11 TEST_F() [8/8]

```
TEST_F (
            VectorTest ,
            destructOfNotEmptyVector_expectedMemoryFreedOnce  )
```

Definition at line 65 of file vectorTests.cpp.

10.9 vector.cpp File Reference

```
#include <algorithm>
#include <stdexcept>
#include <utility>
```

`#include "vector.h"`
Include dependency graph for vector.cpp:

10.10 vector.h File Reference

Klasy Vector zarzadzająca dynamiczną tablicą na elementy (bardziej szczegolowa tresc w README.md): i rozszerzająca się wg potrzeb z obsługą wyjątków.
`#include <cstddef>`
`#include <memory>`
`#include "fraction.h"`
Include dependency graph for vector.h: This graph shows which files directly or indirectly include this file:

Classes

- class Vector

10.10.1 Detailed Description

Klasy Vector zarzadzająca dynamiczną tablicą na elementy (bardziej szczegolowa tresc w README.md): 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.

10.10.1.1 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.

10.10.1.2 Uwaga (bardziej wiazaca tresc jest w pliku <tt>README.md</tt>):

Wszystkie atrybuty powinny być prywatne. Deklaracje klasy powinny znaleźć się w odpowiednich plikach nagłówkowych, definicje metod i konstruktorów - w plikach źródłowych. Wszystkie atrybuty powinny być prywatne, konstruktory i metody - publiczne, metody/konstruktory/destruktory większe niż 1-linijkowe powinny być zadeklarowane w klasie, zdefiniowane poza klasą, Obiekty typów klasowych powinny być przekazywane do funkcji/metod przez referencje (zwykłe lub stałe), metody niemodyfikujące zawartości klasy powinny być oznaczane jako const. Wszystkie metody, które mogą być stałe proszę aby były

1. Co się da na listę inicjalizacyjną konstruktora.

   Note

   Za złe zarządzanie pamięcią (wycieki, pisanie poza pamięcią) powodują odejmowanie punktów Obiekt, z którego przenosimy też powinien się nadawać do użytku!

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

10.10.1.3 Punktacja:

Na maksa przejście wszystkich testów i niepoprawnych operacji na pamieci (m.in. wyciekow pamieci)

10.10.1.4 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