Laboratoria Programowania w C++

Generated by Doxygen 1.9.1

Chapter 1

Laboratorium 6 - własny typ tekstowy MyString

1.0.1 Tresc zadan dla Panstwa (bardziej szczegółowa w pliku README.md):

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:
     - mystring.h i mystring.cpp
     - proszę nie załączać: main.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: shapes↩ Tests.cpp,
6. ∗Dobrze jakby nie bylo warningow kompilacji (flagi: `-Wall -Wextra -pedantic -Werror`, a dla hardcorów jeszcze: `-Weffc++`)
7. Punkty mogą być odejmowane za wycieki pamieci (jest podpiety `valgrind`)
8. Niewykluczone jest sprawdzanie reczne - zaleznie od prowadzacego dana grupe.

1.0.2 Zadanie implementacyjne:

1. Wykorzystanie biblioteki STL i kontenerów podczas implementacji klasy MyString
2. Klasa ta ma zawierać statyczna tablice na tekst do 20 znaków, a resztę ma w razie potrzeby pobierać dynamicznie np. przy pomocy typu `std::string`.
3. Funkcjonalności są dopasowane tak, aby użyć kilku kontenerów standardowych, poćwiczyć pisanie iteratorów, oraz użyć algorytmów uogólnionych (`<algorithm>`).
4. Tresc nalezy wydedukowac w oparciu o plik z testami, nad wieloma testami jest sugestia czego można użyć.
5. Można też użyć biblioteki boost (oczywiście wtedy w razie błędów kompilacji na bobotcie proszę o maila z informacją czego Państwo używają).
6. Sygnatury funkcji do zaimplementowania (pobrane przy pomocy `ctags`):

metoda	sygnatura
MyString	MyString(const char *text)
MyString	MyString(const MyString &text)
begin	iterator begin()
begin	const_iterator begin() const
capacity	auto capacity() const
cbegin	const_iterator cbegin() const
cend	const_iterator cend() const
const_iterator	explicit const_iterator(const MyString* myString, size_t position)
empty	bool empty() const
end	iterator end()
end	const_iterator end() const
getPosition	auto getPosition() const
iterator	explicit iterator(MyString* myString, size_t position)
operator !=	bool operator!=(const MyString& rhs) const
operator !=	bool operator!=(iterator anotherIt)
operator !=	bool operator!=(const_iterator anotherIt) const
operator *	char& operator*()
operator *	char operator*() const
operator +	iterator operator+(size_t pos)
operator +	const_iterator operator+(size_t pos) const
operator ++	iterator& operator++()
operator ++	const_iterator& operator++()
operator -	size_t operator-(iterator anotherIt)
operator -	size_t operator-(const_iterator anotherIt) const
operator –	iterator& operator--()
operator –	const_iterator& operator--()
operator ==	bool operator==(iterator anotherIt)
operator ==	bool operator==(const_iterator anotherIt) const
operator []	char operator[ ](size_t i) const
operator std::string	explicit operator std::string() const
push_back	void push_back(char c)
size	auto size() const
map<MyString, size_t>	countWordsUsageIgnoringCases() const;
all_of	bool all_of(std::function<bool(char)> predicate) const
generateRandomWord	static MyString generateRandomWord(size_t length)

metoda	sygnatura
getUniqueWords	std::set<MyString> getUniqueWords() const
join	MyString join(const std::vector<MyString> &texts) const
startsWith	bool MyString::startsWith(const char *text) const
toLower	MyString& toLower()
trim	MyString& trim()

7. Tym razem kod ma się kompilować z flagami: `-Wall -Wextra -pedantic -Werror` dla hardcorów jeszcze: `-Weffc++`

1.0.3 Uwaga:

1. Konieczne jest zrobienie dwóch wersji metod begin/end -jedna stała, druga nie. Podobnie dwóch wersji iteratora.
2. Informacje `jak zdefiniować własny iterator` lub `2`.

Chapter 2

MyString

W zadaniu chodzi o to, żeby Państwo:

- zaimplementowali własny typ do obsługi tekstu, którego część pamięci będzie statycznie (tablica o stałym rozmiarze), a reszta będzie się allokować wg potrzeby,
- zależy mi też aby Państwo byli w stanie zaimplementować własny iterator
- kolejnym aspektem jest aby Państwo przećwiczyli fundamentalną umiejętność programisty - czytanie kodu. Dlatego nie mają Państwo obszernego opisu, tylko kod z testami i muszą sobie Państwo poradzić.
- wreszcie zależy mi też aby Państwo spróbowali maksymalnie dużo zrobić przy użyciu biblioteki standardowej C++ (nie wszystko na piechotę), dlatego też testy zawierają wskazówki czego można użyć aby daną funkcjonalność dostarczyć bez konieczności implementowania jej samodzielnie.
- $*$ osoby ambitne mogą spróbować zaimplementować pewne funkcjonalności przy pomocy biblioteki `boost`. Oczywiście jakby coś nie działało proszę o priv, abym doinstalował.

2.1 Opis bardziej szczegółowy:

1. Wykorzystanie biblioteki STL i kontenerów podczas implementacji klasy MyString
   - można też użyć typu `std::string` pod spodem, jak i `std::array`, nie trzeba ręcznie zarządzać pamięcią
2. Klasa ta ma zawierać statyczna tablice na tekst do 20 znaków, a resztę ma w razie potrzeby pobierać dynamicznie np. przy pomocy typu `std::string`.
3. Funkcjonalności są dopasowane tak, aby użyć kilku kontenerów standardowych, poćwiczyć pisanie iteratorów, oraz użyć algorytmów uogólnionych (`<algorithm>`).
4. Treść należy wydedukować w oparciu o plik z testami, nad wieloma testami jest sugestia czego można użyć.
5. Można też użyć biblioteki `boost` (oczywiście wtedy w razie błędów kompilacji na bobotcie proszę o maila z informacją czego Państwo używają).
6. Sygnatury funkcji do zaimplementowania (pobrane przy pomocy `ctags`):

metoda	sygnatura

metoda	sygnatura
MyString	MyString(const char ∗text)
MyString	MyString(const MyString &text)
begin	iterator begin()
begin	const_iterator begin() const
capacity	auto capacity() const
cbegin	const_iterator cbegin() const
cend	const_iterator cend() const
const_iterator	explicit const_iterator(const MyString∗ myString, size_t position)
empty	bool empty() const
end	iterator end()
end	const_iterator end() const
getPosition	auto getPosition() const
iterator	explicit iterator(MyString∗ myString, size_t position)
operator !=	bool operator!=(const MyString& rhs) const
operator !=	bool operator!=(iterator anotherIt)
operator !=	bool operator!=(const_iterator anotherIt) const
operator ∗	char& operator∗()
operator ∗	char operator∗() const
operator +	iterator operator+(size_t pos)
operator +	const_iterator operator+(size_t pos) const
operator ++	iterator& operator++()
operator ++	const_iterator& operator++()
operator -	size_t operator-(iterator anotherIt)
operator -	size_t operator-(const_iterator anotherIt) const
operator –	iterator& operator--()
operator –	const_iterator& operator--()
operator ==	bool operator==(iterator anotherIt)
operator ==	bool operator==(const_iterator anotherIt) const
operator []	char operator[ ](size_t i) const
operator std::string	explicit operator std::string() const
push_back	void push_back(char c)
size	auto size() const
map<MyString, size_t>	countWordsUsageIgnoringCases() const;
all_of	bool all_of(std::function<bool(char)> predicate) const
generateRandomWord	static MyString generateRandomWord(size_t length)
getUniqueWords	std::set<MyString> getUniqueWords() const
join	MyString join(const std::vector<MyString> &texts) const
startsWith	bool MyString::startsWith(const char ∗text) const
toLower	MyString& toLower()
trim	MyString& trim()

2.2 ## Najczęstrze problemy/wątpliwości/Uwaga:

1. Konieczne może się okazać zrobienie dwóch wersji metod begin/end -jedna stała, druga nie.
2. Należy zdefiniować dwie wersje iteratorów - stały `const_iterator` i zwykły `iterator` jako klasy zag-

nieżdżone.

(a) Informacje jak zdefiniować własny iterator lub 2. Najprościej jest dziedziczyć po `std::iterator`, niemniej jednak jest to deprecated.

(b) Powinno się to pojawić na ostatnim wykładzie.

3. Szablony muszą być zdefiniowane w całości w pliku nagłówkowym, jednakże proszę aby definicje metod dłuższych niż 1-linijkowe były pod klasą.
4. Iterator nie chce działać z kontenerem standardowym
   - Ponieważ trzeba wskazać jakiego rodzaju jest ten iterator.
5. Można użyć `std::sort` lub `std::stable_sort` - tylko trzeba wiedzieć gdzie i jak.
6. Można spróbować użyć `if constexpr` aby zmniejszyć ilość funkcji.
7. Dodałem pliki, ale testy nadal nie przechodzą - trzeba ponownie uruchomić CMake aby wykrył zmiany plików.
8. Dodałem plik `mystring.h` lub `mystring.cpp` i dalej go nie widzi.
   - Proszę się upewnić, że wielkość liter się zgadza (Linux je rozróżnia w przeciwieństwie do Windowsa)
   - Proszę po dodaniu pliku ponownie uruchomić CMake oraz kompilacje - aby wykryło nowe pliki to skompilowania.

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

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

W opisie filmu jest też częściowy spis treści.

1. O co chodzi w zadaniu
2. Jak to zrobić nie mając pliku z implementacją.
3. Ponowne uruchamienie pliku CMake.
4. Omówienie testów, sugestii i testów iteratorów.
5. Reklama `<algorithm>` i innych algorytmów.
6. Algorytmy tekstowe: `boost::algorithm`
7. Koniec paczek, proszę sobie dopilnować

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 Pytania po implementacji ćwiczenia:

1. (Jak macie pomysł to podrzućcie)

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

1. Użyć biblioteki boost

2.6 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

2.6.1 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`

2.6.2 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.

2.6.3 Project Structure

```
.
 zaj6MyString
    CMakeLists.txt          # CMake configuration file - the file is to open out project in our IDE
    Dockerfile              # this file contains instructions how to run tests in embedded Ubuntu
    Doxyfile                # here is prepared file for Doxygen, to generate documentation when we type `doxyge
    main.cpp                # main file - here we can test out solution manually, but it is not required, it co
    mystring.h              # this file should be created and contain declarations
    mystring.cpp            # this file should be created and contain definitions for declared methods
    README.md               # this file
    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
       myStringTests.cpp  # tests for our class
    trescWygenerowan_dlaStudentowa.pdf
```
```

Chapter 3

Todo List

Member FIRSTNAME

Uzupelnij swoje dane:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

- testing::Test
  - MyStringTester . . . . . **??**

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Class Documentation

7.1 MyStringTester Class Reference

Inheritance diagram for MyStringTester:

Chapter 8

File Documentation

8.1 CMakeLists.txt File Reference

8.2 tests/CMakeLists.txt File Reference

Functions

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

8.2.1 Function Documentation

8.2.1.1 project()

```
project (
            tests  )
```

Definition at line 1 of file CMakeLists.txt.

8.3 main.cpp File Reference

```
#include <iostream>
```

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"

8.3.1 Function Documentation

8.3.1.1 compileTimeContains()

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

Definition at line 134 of file main.cpp.
Here is the caller graph for this function:

8.3.1.2 compileTimeCountFirstDigits()

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

Definition at line 124 of file main.cpp.
Here is the caller graph for this function:

8.3.1.3 compileTimeIsDigit()

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

Definition at line 129 of file main.cpp.
Here is the call graph for this function: Here is the caller graph for this function:

8.3.1.4 compileTimeStrlen()

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

Definition at line 119 of file main.cpp.
Here is the caller graph for this function:

8.3.1.5 main()

```
int main ( )
```

Definition at line 110 of file main.cpp.
Here is the call graph for this function:

8.3.1.6 validateStudentsInfo()

```
void validateStudentsInfo ( )
```

Definition at line 142 of file main.cpp.
Here is the call graph for this function: Here is the caller graph for this function:

8.3.2 Variable Documentation

8.3.2.1 BOOK_ID

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

Definition at line 104 of file main.cpp.

8.3.2.2 FIRSTNAME

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

Todo Uzupelnij swoje dane:

Definition at line 101 of file main.cpp.

8.3.2.3 MAIL

```
constexpr const char* const MAIL = ""  [constexpr]
```
Definition at line 103 of file main.cpp.

8.3.2.4 SURNAME

```
constexpr const char* const SURNAME = ""  [constexpr]
```
Definition at line 102 of file main.cpp.

8.3.2.5 TEACHER_MAIL

```
constexpr const char* const TEACHER_MAIL = "bazior[at]agh.edu.pl"  [constexpr]
```
Definition at line 105 of file main.cpp.

8.4 README.md File Reference

8.5 tests/myStringTests.cpp File Reference

```
#include <vector>
#include <algorithm>
#include <type_traits>
#include <functional>
#include <cctype>
#include <cstring>
#include <gtest/gtest.h>
```
Include dependency graph for myStringTests.cpp:

Classes

- class MyStringTester

Functions

- TEST_F (MyStringTester, emptyStringConstruction_expectedSizeIsZero)
- TEST_F (MyStringTester, constructionFromConstChar_expectedTextCopied)
- TEST_F (MyStringTester, clear_expectedEmptyStringAfterClearing)
- TEST_F (MyStringTester, ostreamOperator_readTextShorterThanBufferSize_expectedTextInStream)
- TEST_F (MyStringTester, ostreamOperator_readTextEqualToBufferSize_expectedTextInStream)
- TEST_F (MyStringTester, ostreamOperator_readTextLongerThanBufferSize_expectedTextInStream)
- TEST_F (MyStringTester, indexAccessOperator_expectedPossibilityToAccesElementsWithIndex)
- TEST_F (MyStringTester, istreamOperator_expectedTextCopiedFromStreamToString)
- TEST_F (MyStringTester, trimming_expectedSpacesRemoved)
- TEST_F (MyStringTester, operatorPlusEqual_expectedTextConcatenation)
- TEST_F (MyStringTester, countingUniqueWords)
- TEST_F (MyStringTester, countingWordsNumbersInText)
- TEST_F (MyStringTester, countingWordsNumbersInTextIgnoringWordCases)
- TEST_F (MyStringTester, wordsIntoLowerCase)
- TEST_F (MyStringTester, randomWordGeneration)
- TEST_F (MyStringTester, startsWith)
- TEST_F (MyStringTester, endsWith)

- TEST_F (MyStringTester, comparingTexts)
- TEST_F (MyStringTester, joiningContainer)
- TEST_F (MyStringTester, all_of)

8.5.1 Function Documentation

8.5.1.1 TEST_F() [1/20]

```
TEST_F (
            MyStringTester ,
            all_of  )
```

Definition at line 396 of file myStringTests.cpp.

8.5.1.2 TEST_F() [2/20]

```
TEST_F (
            MyStringTester ,
            clear_expectedEmptyStringAfterClearing  )
```

Definition at line 51 of file myStringTests.cpp.

8.5.1.3 TEST_F() [3/20]

```
TEST_F (
            MyStringTester ,
            comparingTexts  )
```

na koncu nie ma kropki:
Definition at line 360 of file myStringTests.cpp.

8.5.1.4 TEST_F() [4/20]

```
TEST_F (
            MyStringTester ,
            constructionFromConstChar_expectedTextCopied  )
```

Definition at line 36 of file myStringTests.cpp.

8.5.1.5 TEST_F() [5/20]

```
TEST_F (
            MyStringTester ,
            countingUniqueWords  )
```

Definition at line 186 of file myStringTests.cpp.

8.5.1.6 TEST_F() [6/20]

```
TEST_F (
            MyStringTester ,
            countingWordsNumbersInText  )
```

Definition at line 199 of file myStringTests.cpp.

8.5.1.7 TEST_F() [7/20]

```
TEST_F (
            MyStringTester ,
            countingWordsNumbersInTextIgnoringWordCases  )
```

Definition at line 261 of file myStringTests.cpp.

8.5.1.8 TEST_F() [8/20]

```
TEST_F (
            MyStringTester ,
            emptyStringConstruction_expectedSizeIsZero  )
```

Definition at line 28 of file myStringTests.cpp.

8.5.1.9 TEST_F() [9/20]

```
TEST_F (
            MyStringTester ,
            endsWith  )
```

Definition at line 348 of file myStringTests.cpp.

8.5.1.10 TEST_F() [10/20]

```
TEST_F (
            MyStringTester ,
            indexAccessOperator_expectedPossibilityToAccesElementsWithIndex  )
```

Definition at line 92 of file myStringTests.cpp.

8.5.1.11 TEST_F() [11/20]

```
TEST_F (
            MyStringTester ,
            istreamOperator_expectedTextCopiedFromStreamToString  )
```

Definition at line 104 of file myStringTests.cpp.

8.5.1.12 TEST_F() [12/20]

```
TEST_F (
            MyStringTester ,
            joiningContainer  )
```

Definition at line 377 of file myStringTests.cpp.

8.5.1.13 TEST_F() [13/20]

```
TEST_F (
            MyStringTester ,
            operatorPlusEqual_expectedTextConcatenation  )
```

Definition at line 166 of file myStringTests.cpp.

8.5.1.14 TEST_F() [14/20]

```
TEST_F (
            MyStringTester ,
```

```
            ostreamOperator_readTextEqualToBufferSize_expectedTextInStream  )
```

Definition at line 74 of file myStringTests.cpp.

8.5.1.15 TEST_F() [15/20]

```
TEST_F (
            MyStringTester ,
            ostreamOperator_readTextLongerThanBufferSize_expectedTextInStream  )
```

Definition at line 83 of file myStringTests.cpp.

8.5.1.16 TEST_F() [16/20]

```
TEST_F (
            MyStringTester ,
            ostreamOperator_readTextShorterThanBufferSize_expectedTextInStream  )
```

Definition at line 65 of file myStringTests.cpp.

8.5.1.17 TEST_F() [17/20]

```
TEST_F (
            MyStringTester ,
            randomWordGeneration  )
```

Definition at line 322 of file myStringTests.cpp.

8.5.1.18 TEST_F() [18/20]

```
TEST_F (
            MyStringTester ,
            startsWith  )
```

Definition at line 333 of file myStringTests.cpp.

8.5.1.19 TEST_F() [19/20]

```
TEST_F (
            MyStringTester ,
            trimming_expectedSpacesRemoved  )
```

Definition at line 121 of file myStringTests.cpp.

8.5.1.20 TEST_F() [20/20]

```
TEST_F (
            MyStringTester ,
            wordsIntoLowerCase  )
```

Definition at line 307 of file myStringTests.cpp.