AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Oprogramowanie analizujące dane zawarte w pliku csv z zastosowaniem GitHub

Autor:
Mateusz Stanek
Dawid Szołdra

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2025

Spis treści

1. Ogólne określenie wymagań **3**

2. Analiza problemu **4**

2.1. Drzewo jako struktura danych 4
2.2. Wczytanie i manipulacja plikiem 4
2.3. Git 4
2.4. Doxygen 6

3. Projektowanie **7**

3.1. Implementacja programu 7
3.2. Git 7
3.3. Doxygen 8
3.4. Google Test 8

4. Implementacja **10**

4.1. Ogólne informacje o implementacji klas 10
4.2. Measurement.hpp 10
4.3. MeasurementRecord.hpp 10
4.3.1. MeasurementsImporter.hpp oraz MeasurementsImporter.cpp . . . 11
4.4. MeasurementsTree.cpp i MeasurementsTree.hpp 16
4.5. Google Test 24

5. Wnioski **33**

Literatura **34**

Spis rysunków **35**

Spis tabel **36**

Spis listingów **37**

1. Ogólne określenie wymagań

Celem projektu jest swtorzenie programu który będzie analizował dane zawarte w pliku csv- oraz kontrolowanie jego wersji za pomocą narzędzia git. Do zadań programu będzie należało: wczytanie danych z pliku, przechowanie ich w strukturze drzewa, wykonanie operacji takich jak obliczanie sum i średnich w przedziale czasowym, poróWnywanie wyników w dwuch przedziałach czasowych, wyszukiwanie rekordów spełniających warunki, wypisywanie danych z przedziału czasowego, zapis zmienionych danych do pliku binarnego oraz ich odczyt.
Program będzie podzielony na plik main oraz pliki z funkcjami(każda funkcja będzie w osobnym pliku).
Wynikiem projektu powinno być działające oprogramowanie wczytujące dane z pliku, wykonujące na nim operacje a następnie zapisujące dane.

2. Analiza problemu

2.1. Drzewo jako struktura danych

Drzewo[1] jest hierarchiczną strukturą danych, ułożonych nieliniowo, czyli dane ułożone są w wielu poziomach. Elementy w drzewie są nazywane węzłami. Elementy połączone są ze sobą gałęziami. Każdy węzeł może mieć kilka dzieci, czyli elementów na które wskazuje, ale tylko jednego rodzica, czyli element wskazujący na dany węzeł. Między rodzeństwem, czyli elementami na tym samym poziomie nie ma żadnych połączeń bezpośrednich. Ważny w charakterystyce drzewa jest też fakt, że nie mogą w nim występować żadne cykle.

2.2. Wczytanie i manipulacja plikiem

Celem projektu jest stworzenie programu który wczyta, sprawdzi oraz przetworzy dane pomiarowe znajdujące się w pliku CSV. Bardzo ważnym jest uwzględnienie potencjalnych błędów takich jak np. puste linie lub zduplikowane wpisy, takie błędy należy odpowiednio zapisać do logów. Poprawne dane natomiast powinny być zorganizowane w strukturze typu rok -¿ miesiąc -¿ dzień -¿ ćwiartka doby. Taką strukturę można zaimplementować przy pomocy np wektorów. Konieczne jest również stworzenie metody umożliwiającej zapis i odczyt pliku f formacie binarnym oraz implementacja menu która pozwoli na wybór zadań do realizacji.

2.3. Git

Kolejnym konceptem, którym zajmuje się projekt jest narzędzie git[2]. Pozwala ono zarządzać poszczególnymi wersjami projektów. Głównym korzeniem gita jest system commitów, czyli zapisania zmian w pliku w stosunku do commita starszego. To, w połączeniu z jego innymi możliwościami pozwala na tworzenie długich i skomplikowanych osi czasu danych projektów.

Użycie gita można zademonstrować na prostym przykładzie. Tworzymy katalog a w nim repozytorium, używając komendy `git init`, jak widać na rys. 2.1.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/mattys/skrypty-i-syfy/studia/rok2/progr
amowanie-zaawansowane/p1/git-test/.git/
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % ls
drwxr-xr-x - mattys  6 Nov 15:54 .git/
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test %
```

Rys. 2.1. Puste repozytorium git

Stwórzmy jakiś plik i dodajmy go do repozytorium. Plik można dodać do repozytorium komendą `git add`

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo "commit1" > plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git add plik.txt
```

Rys. 2.2. Stworznie pliku w repozytorium

Następnie należy scommitować zmiany.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git commit -am "utworzenie pl
ik.txt"
[master (root-commit) bb3b898] utworzenie plik.txt
 1 file changed, 1 insertion(+)
 create mode 100644 plik.txt
```

Rys. 2.3. Commit nr. 1

Na rysunku 2.3 użyta komenda `git commit` commituje wszystkie dodane pliki (`-a`) z jakimś komunikatem (`-m`).

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo "commit2" >> plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git commit -am "uzupelnienie
plik.txt"
[master 40694c2] uzupelnienie plik.txt
 1 file changed, 1 insertion(+)
```

Rys. 2.4. Commit nr. 2

Na rys. 2.4, został utworzony kolejny commit, dodajacy zmiany do `plik.txt`.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git log
commit 40694c2e73758368445cb817f4524ee5c5afbece (HEAD → master)
Author: mattys1 <mattys0082@gmail.com>
Date:   Wed Nov 6 16:26:07 2024 +0100

    uzupelnienie plik.txt

commit bb3b898df760395b5763575e204c3c43b513d2f6
Author: mattys1 <mattys0082@gmail.com>
Date:   Wed Nov 6 16:12:31 2024 +0100

    utworzenie plik.txt
```

Rys. 2.5. Log gita

Jak na rys. 2.5 jest pokazane, używając komendy `git log`, można wyświetlić log commitów w repozytorium.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git checkout bb3b898df760395b
5763575e204c3c43b513d2f6
Note: switching to 'bb3b898df760395b5763575e204c3c43b513d2f6'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at bb3b898 utworzenie plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo plik.txt
plik.txt
```

Rys. 2.6. Demonstracja checkout

Jak widać na rys. 2.6, komenda `git checkout`, pozwala na przejście repozytorium w inny stan, w tym przypadku przechodzi się do commita o danym ID, pokazanym na rys. 2.5. Jako, że jest to pierwszy commit, nie ma w nim zmian z drugiego.

2.4. Doxygen

Doxygen[3] jest narzędziem automatycznie generującym dokumentację programu z komentarzy w kodzie źródłowym. Potrafi on generować strony HTML, gdzie można dynamicznie nawigować się miedzy rożnymi częściami kodu oraz pliki LaTeX, które można konwertować na różne, statyczne formaty.

3. Projektowanie

3.1. Implementacja programu

Do zaimplementowania programu zostanie użyty Język `C++` z kompilatorem `gcc` oraz `MVSC`. Wersja standardu `C++`to `C++23`. Wersja ta została użyta, ze względu na zawartą w niej funkcję `std::print()`. Jako, że projekt ma być rozdzielony na dwa pliki, zostanie zastosowany `CMake` w celu automatyzacji procesu budowania. `CMake` pozwala na generowanie plików budujących dany projekt, zgodnie z określoną konfiguracją. Oszczędza to programiście, szczególnie przy większych projektach, manualne pisanie Makefileów. Plik konfiguracyjny `CMakeLists.txt` może wyglądać jak na rysunku

```
1  cmake_minimum_required(VERSION 3.5)
2
3  set(PROJECT_NAME proj1)
4
5  project(${PROJECT_NAME} VERSION 0.1 LANGUAGES CXX)
6
7  set(CMAKE_CXX_STANDARD 23)
8  set(CMAKE_CXX_STANDARD_REQUIRED ON)
9  set(CMAKE_EXPORT_COMPILE_COMMANDS True)
10 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_CURRENT_LIST_DIR}/out)
11
12 add_subdirectory(src)
13
```

Listing 1. Plik konfiguracyjny CMake

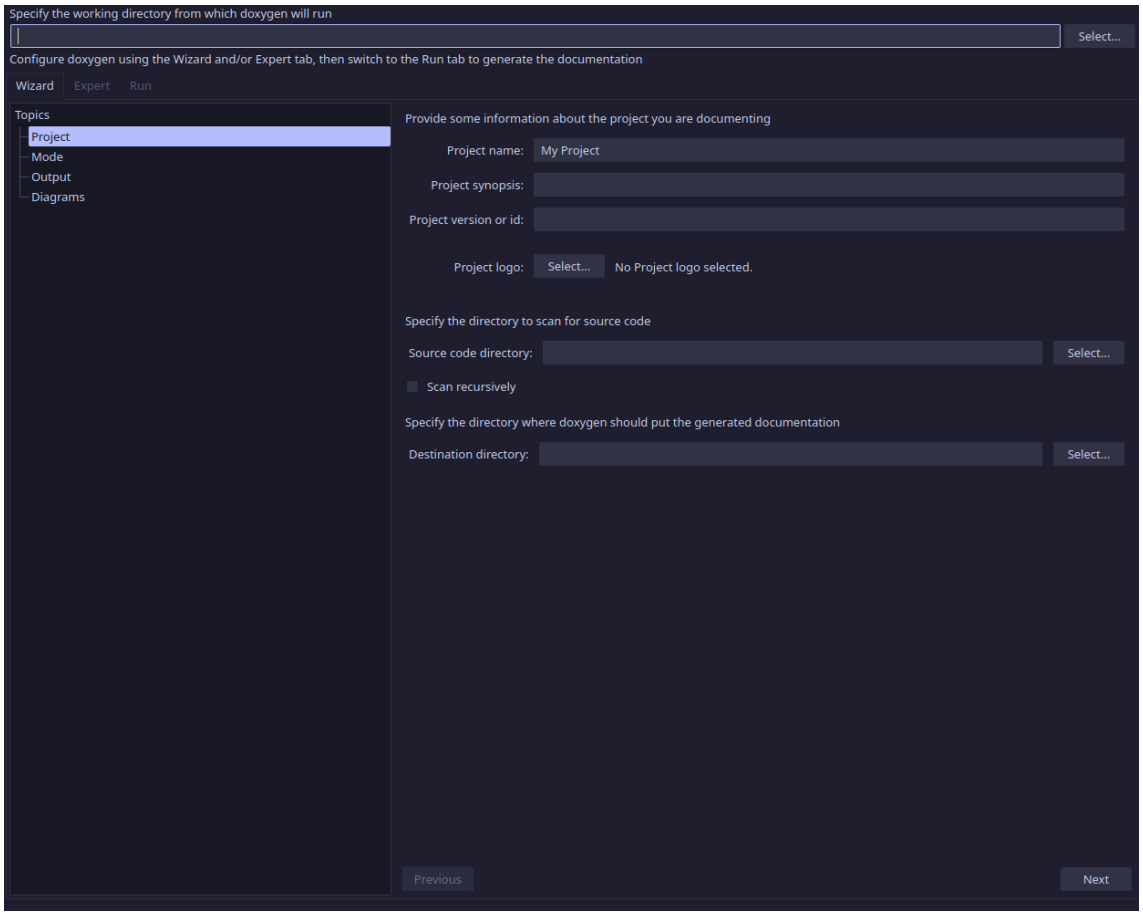
Edytorem będzie program Neovim oraz Visual Studio 2022. Jest to terminalowy edytor tekstu z możliwością poszerzenia funkcjonalności przy użyciu wszelkiego rodzaju pluginów. Wybrany został, dlatego że jest on już skonfigurowany na moim komputerze zgodnie z moimi preferencjami.

3.2. Git

Dla ułatwienia pracy, zastosowany został front-end dla gita o nazwie lazygit. Jest to terminalowy program, którego główną zaletą jest łatwa nawigacja przy użyciu klawiatury. Ponadto, jest on lekki i szybki.

3.3. Doxygen

Konfiguracja dla Doxygena jest wygenerowana przy użyciu programu doxywizard, pokazany na rys. 3.1, pozwalającego na graficzne zmienianie ustawień. Po wygenerowaniu konfiguracji, Doxygen wywoływany jest przy użyciu komendy.



Rys. 3.1. Interfejs programu doxywizard

3.4. Google Test

Google Test został dodany do projektu jako biblioteka w `CMake`, co zostało ukazane na listingu nr. 2. Framework jest automatycznie pobierany i instalowany przy konfiguracji.

```
1 include(FetchContent)
2 FetchContent_Declare(
3 googletest
4 URL https://github.com/google/googletest/
   archive03597a01ee50ed33e9dfd640b249b4be3799d395.zip
5 )
6 FetchContent_MakeAvailable(googletest)
```

```
7
```

Listing 2. Dodanie Google Test do projektu

Pliki źródłowe testów znajdują się we własnym folderze, więc też trzeba do niego dodać `CMakeLists`, jak widać na listingu nr. 3

```
1  file(GLOB_RECURSE SRC_FILES *.cpp *.h)
2  set(TESTS_EXECUTABLE ${PROJECT_NAME}_tests)
3
4  enable_testing()
5
6  add_executable(${TESTS_EXECUTABLE} ${SRC_FILES})
7
8  target_link_libraries(
9  ${TESTS_EXECUTABLE} PUBLIC
10 ${PROJECT_NAME}_lib
11 GTest::gtest_main
12 )
13
```

Listing 3. Dodanie Google Test do projektu

4. Implementacja

4.1. Ogólne informacje o implementacji klas

Program podzielony jest na wiele plików, w tym rozdziale zostaną opisane funkcjonalności każdego z nich.

4.2. Measurement.hpp

W tym pliku znajduje się struktura definiująca pojedynczy pomiar. Kod struktury został przedstawiony na listingu nr 4.

```
1  #pragma once
2
3  #include <compare>
4  struct Measurement {
5    double autoconsumption;
6    double gridExport;
7    double gridImport;
8    double consumption;
9    double production;
10   int timeMinutes;
11
12   auto operator<=>(const Measurement&) const = default;
13 };
```

Listing 4. Zawartość pliku `Measurement.hpp`

Struktura służy do przechowywania danych z pliku.

4.3. MeasurementRecord.hpp

W tym pliku znajduje się struktura `MeasurementRecord` wraz ze strukturą wewnętrzną `Time`. Zadaniem tej struktury jest przechowywanie mementu w czasie wraz z wartościami pomiarowymi. Struktura jest przedstawiona na listingu nr 5.

```
1  #pragma once
2
3  #include <compare>
4  struct Measurement {
5    double autoconsumption;
6    double gridExport;
7    double gridImport;
8    double consumption;
```

```
    double production;
    int timeMinutes;

    auto operator<=>(const Measurement&) const = default;
};
```

Listing 5. Zawartość pliku `MeasurementRecord.hpp`

4.3.1. MeasurementsImporter.hpp oraz MeasurementsImporter.cpp

Klasa została podzielona na dwa pliki: .hpp oraz .cpp. Na listingu nr 6 przedstawiona została zawawrtość pliku `MeasurementsImporter.hpp`.

```
#pragma once
#include "MeasurementRecord.hpp"
#include <string_view>
#include <vector>
#include <fstream>

class MeasurementsImporter {
  private:
  std::vector<MeasurementRecord> records;
  public:
  void read_measurements(const std::string_view fileName);
  std::vector<MeasurementRecord> get_records() const;
};
```

Listing 6. Zawartość pliku `MeasurementsImporter.hpp`

Na listingu nr 7 przedstawiona została zawartość pliku `MeasurementsImporter.cpp`.

```
#include "MeasurementsImporter.hpp"
#include <algorithm>
#include <cassert>
#include <chrono>
#include <ranges>
#include <variant>

void MeasurementsImporter::read_measurements(const std::
 string_view fileName) {
  const auto check_for_double { [this](const MeasurementRecord&
 record) -> bool {
      return std::ranges::find(records, record) != records.end();
  }};

  const auto log { [](std::string_view line, std::fstream& file)
 {
```

```
14          const auto time { std::chrono::system_clock::now() };
15
16          file << std::format("{}: Loading record: {}\n", time, line)
    ;
17      }};
18
19      const auto log_err { [](std::string_view line, const std::
    string_view message, std::fstream& file) {
20          const auto time { std::chrono::system_clock::now() };
21
22          file << std::format("{}: {}: {}\n", time, message, line);
23      }};
24
25      std::fstream measurementsFile;
26      measurementsFile.open(fileName.data(), std::fstream::in);
27
28      if(!measurementsFile.is_open()) {
29        throw std::runtime_error("Could not open input file");
30      }
31
32      std::fstream errorLogs; errorLogs.open("log_error_data_godzina.
    txt", std::fstream::app);
33      if(!errorLogs.is_open()) {
34        throw std::runtime_error("Could not open error log file");
35      }
36
37      std::fstream allLogs; allLogs.open("log_data_godzina.txt", std
    ::fstream::app);
38      if(!allLogs.is_open()) {
39        throw std::runtime_error("Could not open log file");
40      }
41
42      std::vector<std::string> lines;
43      std::string line;
44      while(std::getline(measurementsFile, line)) {
45        if(line.empty() || std::find_if(line.begin(), line.end(), [](
    const auto ch) { return !isspace(ch); }) == line.end()) {
46          continue;
47        };
48
49        log(line, allLogs);
50
51        if(const auto invalid = std::find_if(line.begin(), line.end()
    , [](const auto x) {
52          return !((x >= '0' && x <= '9') || x == '.' || x == ',' ||
```

```
    x == ':' || x == '\"' || isspace(x));
53     }); invalid != line.end()) {
54       log_err(line, "Line with invalid characters", errorLogs);
55       continue;
56     }
57
58     lines.emplace_back(line);
59   }
60
61   if(lines.size() == 0) {
62     return;
63   }
64
65   records.reserve(lines.size());
66   for(const auto& line : lines) {
67     auto entries { std::views::split(line, ',') | std::ranges::to
    <std::vector<std::string>>() };
68
69     if(entries.size() != 6) {
70       log_err(line, "Invalid entry size", errorLogs);
71       continue;
72     }
73
74     const auto DateAndHourTime { entries[0] |
75       std::views::split(' ') |
76       std::ranges::to<std::vector<std::string>>() };
77
78     const auto& date { DateAndHourTime[0] }, hourTime {
    DateAndHourTime[1] };
79
80     const auto splitDate { date |
81       std::views::split('.') |
82       std::ranges::to<std::vector<std::string>>() };
83
84     const auto& day { splitDate[0] }, month { splitDate[1] },
    year { splitDate[2] };
85
86     for(auto& x : entries) {
87       x.erase(remove(x.begin(), x.end(), '\"'), x.end());
88     }
89
90     const auto minutesPerQuarter { 24 * 60 / 4 };
91
92     const auto hoursMinutes { hourTime |
93       std::views::split(':') |
```

```
94          std::ranges::to<std::vector<std::string>>()
95          /* std::ranges::to<std::vector<std::string>>() }; */
96      };
97
98      const auto timeInMinutes { std::stoi(hoursMinutes[0]) * 60 +
    std::stoi(hoursMinutes[1]) };
99
100     /* return timeInMinutes / minutesPerQuarter + 1; */
101
102     const auto record { [&]() -> std::variant<MeasurementRecord,
    std::exception> const {
103         try {
104           return MeasurementRecord {
105             .time = {
106               .year = std::stoi(year),
107               .month = std::stoi(month),
108               .day =  std::stoi(day),
109               .inMinutes = timeInMinutes,
110               .quarter =  timeInMinutes / minutesPerQuarter + 1
111             },
112
113             .autoconsumption = std::stod(entries[1]),
114             .gridExport = std::stod(entries[2]),
115             .gridImport = std::stod(entries[3]),
116             .consumption = std::stod(entries[4]),
117             .production = std::stod(entries[5])
118           };
119         } catch(std::exception& e) {
120           return e;
121         }
122       }()};
123
124     if(std::holds_alternative<std::exception>(record)) {
125       log_err(line, "Error converting line to values", errorLogs);
126       continue;
127     }
128
129     if(check_for_double(std::get<MeasurementRecord>(record))) {
130       log_err(line, "Line already exists", errorLogs);
131       continue;
132     }
133
134     records.push_back(std::get<MeasurementRecord>(record));
135   }
```

```
136 }
137
138 std::vector<MeasurementRecord> MeasurementsImporter::get_records
     () const {
139 return records;
140 }
```

Listing 7. Zawartość pliku `MeasurementsImporter.cpp`

Głównym zadaniem klasy jest odczytywanie zawartości pliku oraz zapisywanie ich we własnym kontenerze. Za odczyt danych z pliku odpowiedzialna jest metoda `read_measurements` znajdująca się w wierszach od 8 do 136 na listingu nr 6. Instrukcja w wiersach od 8 do 11 odpowiedzialna jest za sprawdzanie czy nie ma duplikatów. Instrukcja w wierszach od 13 do 17 jest odpowiedzialna za logowanie poprawnych danych. Instrukcja w wierszach od 19 do 23 jest odpowiedzialna za logowanie błędów. Instrukcje w wierszach od 25 do 26 są odpowiedzialne za otwarcie pliku wejściowego. Instrukcja if w wierszach od 28 do 30 jest odpowiedzialna za sprawdzanie czy plik został otwarty. Instrukcje w wierszach od 32 do 35 są odpowiedzialne za otwarcie pliku z logami błędów. Instrukcje w wierszach od 37 do 40 są odpowiedzialne za otwarcie pliku z logami. Instrukcje w wierszach od 42 do 43 są odpowiedzialne za stworzenie kontenera na wszystkie linie. Instrukcje w wierszach od 44 do 59 są odpowiedzialne za utworzenie pętli wczytującej dane z pliku. Na początku, w wierszach od 45 do 47 są instrukcje odpowiedzialne za pomijanie pustych lini lub linii z samymi białymi znakami. Następnie w wierszu 49 jest instrukcja która loguje każdą linię do pliku z logami zwykłymi. Następne instrukcje w wierszach od 61 do 57 sprawdzają czy linia zawiera dodatkowe znaki, jeżeli wystąpił błąd to logujemy do loga z błędami i przechodzimy dalej. Ostatnia instrukcja pętli w wierszu 58 dodaje linię do wektora `lines`. Instrukcja in w wierszach od 61 do 63 sprawdza czy wczytaliśmy linie, jeżeli nie to kończy się działanie funkcji. Instrukcje w wierszu 65 jest odpowiedzialna za rezerwację pamięci dla `records`. Pętla for w wierszach od 66 do 135 jest odpowiedzialna za przetwarzanie każdej lini oraz konwersję do `MeasurementRecord`. Pętla zaczyna działanie od rozbicia lini po przecinku na 6 fragmentów w wierszu 67. Następnie w wierszach od 69 do 72 sprawdzane jest czy są 6 fragmentów, jeżeli nie to logujemy do logów z błędami i linia jest pomijana. Instrukcja w wierszach 74 do 78 wyciagają datę. Instrukcje znajdujące się w wierszach od 80 do 82 rozbijają datę po kropkach na dzień, miesiąc i rok. Pętla for w wierszach od 86 do 88 jest odpowiedzialna za usuwanie cudzysłowów. Instrukcja w wierszu 90 ustala liczbę minut przypadających na ćwiartkę doby. Instrukcje w wierszach od 92 do 96 rozdzielają godziny po dwukropku na godziny i minuty. Instrukcja w wierszu 98 jest odpowie-

dzialna za konwersję godziny i minuty na jedną wartość w minutach od północy. Instrukcje w wierszach od 102 do 122 są odpowiedzialne za tworzenie rekordu w bloku lambda. Instrukcja if w wiersach od 124 do 127 jest odpowiedzialna za czy w rekord jest poprawny rekord czy wyjątek. Instrukcja if w wierszach od 129 do 132 jest odpowiedzialna za sprawdzenie czy wczytany rekord już istnieje. Instrukcja w wierszu 134 dodaje nowy rekord do kontenera `records`.

4.4. MeasurementsTree.cpp i MeasurementsTree.hpp

MeasurementsTree to klasa przechowująca i porządkująca pomiary, jest podzielona na dwa pliki. W pliku `MeasurementsTree.hpp` określone jest co będzie zawierać klasa `MeasurementsTree`, w pliku `MeasurementsTree.cpp` określone jest faktyczna implementacja tych metod. Na listingu nr 8 predstawiona została zawartość pliku `MeasurementsTree.hpp`.

```
1  #pragma once
2
3  #include <cassert>
4  #include <span>
5  #include <vector>
6  #include "Measurement.hpp"
7  #include "MeasurementRecord.hpp"
8
9  class MeasurementsTree {
10   private:
11   using TreeType = std::vector<std::vector<std::vector<std::
    vector<std::vector<Measurement>>>>;
12   TreeType tree;
13
14   constexpr static TreeType endDummy { TreeType {} };
15   public:
16   class Iterator {
17     private:
18     TreeType* tree;
19     size_t yearIdx;
20     size_t monthIdx;
21     size_t dayIdx;
22     size_t quarterIdx;
23     size_t measurementIdx;
24
25
26     int incrementSafe() {
27       assert(monthIdx < (*tree)[yearIdx].size());
```

```
28        assert(dayIdx < (*tree)[yearIdx][monthIdx].size());
29        assert(quarterIdx < (*tree)[yearIdx][monthIdx][dayIdx].size
    ());
30        /* assert(measurementIdx < (*tree)[yearIdx][monthIdx][
    dayIdx][quarterIdx].size()); */
31
32        auto& yearVec { (*tree)[yearIdx] };
33        auto& monthVect { yearVec[monthIdx] };
34        auto& dayVec { monthVect[dayIdx] };
35        auto& quarterVec { dayVec[quarterIdx] };
36
37        /* assert(quarterVec.empty()); */
38        measurementIdx++;
39        if (measurementIdx >= quarterVec.size()) {
40          measurementIdx = 0;
41          quarterIdx++;
42
43          if (quarterIdx >= dayVec.size()) {
44            quarterIdx = 0;
45            dayIdx++;
46
47            if (dayIdx >= monthVect.size()) {
48              dayIdx = 0;
49              monthIdx++;
50
51              if (monthIdx >= yearVec.size()) {
52                monthIdx = 0;
53                yearIdx++;
54
55                if (yearIdx >= tree->size()) {
56                  return -1;
57                }
58              }
59            }
60          }
61        }
62        /* Measurement& measurement { quarterVec[measurementIdx] };
     */
63        return 0;
64      }
65
66      void goToNextValid() {
67        //FIXME: get the reference to the actual member vectors not
     the copies
68        std::span yearVec { (*tree)[yearIdx] };
```

```
69          std::span monthVec { yearVec[monthIdx] };
70          std::span dayVec { monthVec[dayIdx] };
71          std::span quarterVec { dayVec[quarterIdx] };
72
73          while(quarterVec.empty() && yearIdx < tree->size()) {
74            const auto end { incrementSafe() };
75
76            if(end < 0) {
77              break;
78            }
79
80            yearVec = (*tree)[yearIdx];
81            monthVec = yearVec[monthIdx];
82            dayVec = monthVec[dayIdx];
83            quarterVec = dayVec[quarterIdx];
84          }
85
86        }
87        void decrementSafe() {
88          std::span yearVec { tree[yearIdx] };
89          std::span monthVec { yearVec[monthIdx] };
90          std::span dayVec { monthVec[dayIdx] };
91          std::span quarterVec { dayVec[quarterIdx] };
92          std::span record { quarterVec[measurementIdx] };
93
94          if(measurementIdx == 0) {
95            measurementIdx = quarterVec.size() - 1;
96            quarterIdx--;
97          }
98
99          if(quarterIdx == 0) {
100           quarterIdx = dayVec.size() - 1;
101           dayIdx--;
102         }
103
104         if(dayIdx == 0) {
105           dayIdx = monthVec.size() - 1;
106           monthIdx--;
107         }
108
109         if(monthIdx == 0) {
110           monthIdx = yearVec.size() - 1;
111           yearIdx--;
112         }
113       }
```

```
114     public:
115     Iterator(TreeType* _tree, size_t year = 0, size_t month = 0,
    size_t day = 0, size_t quarter = 0, size_t measurement = 0):
116     tree(_tree),
117     yearIdx(year),
118     monthIdx(month),
119     dayIdx(day),
120     quarterIdx(quarter),
121     measurementIdx(measurement) {
122       if(tree != nullptr) {
123         goToNextValid();
124       }
125     }
126
127     Measurement& operator*() {
128       auto measurements { (*tree)[yearIdx][monthIdx][dayIdx][
    quarterIdx] };
129
130       assert(!measurements.empty());
131
132       return (*tree)[yearIdx][monthIdx][dayIdx][quarterIdx][
    measurementIdx];
133     }
134
135     std::strong_ordering operator<=>(const Iterator& other) const
     {
136       if(yearIdx == tree->size() && yearIdx == other.yearIdx) {
    // for end iterator
137         return std::strong_ordering::equal;
138       }
139
140       if(yearIdx != other.yearIdx) {
141         return yearIdx <=> other.yearIdx;
142       } else if(monthIdx != other.monthIdx) {
143         return monthIdx <=> other.monthIdx;
144       } else if(dayIdx != other.dayIdx) {
145         return dayIdx <=> other.dayIdx;
146       } else if(quarterIdx != other.quarterIdx) {
147         return quarterIdx <=> other.quarterIdx;
148       } else {
149         return measurementIdx <=> other.measurementIdx;
150       }
151
152       return std::strong_ordering::equal;
153     }
```

```
154
155       bool operator!=(const Iterator& other) const {
156         return (*this <=> other) != std::strong_ordering::equal;
157       }
158
159       Iterator operator++() {
160         incrementSafe();
161         goToNextValid();
162
163         if(yearIdx >= tree->size()) {
164           return Iterator(nullptr, tree->size());
165         }
166
167         return(*this);
168       }
169
170       Iterator operator--() {
171         decrementSafe();
172
173         return Iterator(
174         tree,
175         yearIdx,
176         monthIdx,
177         dayIdx,
178         quarterIdx,
179         measurementIdx
180         );
181       }
182     };
183
184     public:
185     MeasurementsTree();
186
187     Iterator begin();
188     Iterator end();
189
190     void generate_measurement_tree(std::vector<MeasurementRecord>
    records);
191     TreeType get_tree(void) const;
192   };
```

Listing 8. Zawartość pliku `MeasurementsTree.hpp`

W pliku znajduje się definicja klasy `MeasurementsTree` oraz klasa wewnętrzna `Iterator`. Klasa `Iterator` znajdująca się na wierszach od 16 do 182 służy do prze-

chodzenia przez wszystkie pomiary w pięciopoziomowym zagnieżdżeniu wektorów. Zmienne w wierszach od 19 do 23 określają aktualną pozycję w drzewie. Metoda `IncrementSafe` w wierszu od 26 do 63 jest metodą pomocniczą do bezpiecznego przejścia do następnego elementu. W wierszach od 27 do 29 znajdują się insstrukcje `assert`, odpowiedzialne za sprawdzanie poprawności bieżących indeksów. Instrukcje w wierszach od 32 do 35 są odpowiedzialne za odniesienia do odpowiednich wektorów dla aktualnie rozpatrywanego przypadku. Instrukcja w wierszu 38 przesuwa na kolejny element w `quarteridx`. Zagnieżdżony if w wierszach od 43 do 59 jest odpowiedzialny za zarządzanie iteracją po strukturze `tree`.

Metoda `GoToNextValid` w wierszach od 66 do 88 służy do przeskoku do następnego niepustego wektora pomiarów. W wierszach od 68 do 71 znajdują się instrukcje odpowiedzialne za tworzenie wycinków do wglądu w dane. Następnie pętla `while` w wierszach od 73 do 84 sprawdza czy wektor `quarterVec` jest pusty i przechodzi aż do znalezienia niepustego lub skończenia się drzewa.

Metoda `decrementSafe` w wierszach od 87 do 113 służy do cofania się do poprzedniego elementu. W wierszach od 88 do 92 tworzone są wycinki do wglądu w dane. W wierszach od 94 do 97 jest instrukcja if która sprawdza czy musimy cofnąć się do poprzedniej kwarty. Instrukcja if w wierszach od 99 do 102 sprawdza czy `quarterIdx` wynosi 0, jeżeli tak to cofamy się do poprzedniego dnia. Instrukcja if w wierszach od 104 do 107 sprawdza czy `dayIdx` wynosi 0, jeżeli tak to cofamy się do poprzedniego miesiąca. Instrukcja if w wierszach od 110 do 112 sprawdza czy `monthIdx` wynosi 0, jeżeli tak to cofamy się do poprzedniego roku. W wierszach od 115 do 125 znajduje się konstruktor inicjujący `Iterator`. Instrukcje w wierszach od 127 do 133 są odpowiedzialne za zwrócenie referencji do aktualnie wskazywanego pomiaru. Instrukcje w wierszach od 127 do 153 są odpowiedzialne za porównanie dwóch iteratorów w celu sprawdzenia ich kolejności. Instrukcja bool w wierszach od 155 do 157 jest odpowiedzialna za sprawdzanie czy iterator nie wskazuje na to samo co inny iterator. Instrukcje w wierszach od 159 do 165 są odpowiedzialne za przesuwanie się do następnego pomiaru w drzewie. Instrukcje w wierszach od 173 do 181 są odpowiedzialne za przesunięcie się do poprzedniego elementu w drzewie.

W wierszach od 185 do 191 mamy deklaracje metod publicznych dla klasy `MeasurementTree`. Pełny kod tych metod znajduje się w listingu nr 9.

```
1  #include "MeasurementsTree.hpp"
2
3  MeasurementsTree::MeasurementsTree(): tree {
4    2, std::vector {
5      12, std::vector {
```

```
      30, std::vector {
        4, std::vector<Measurement> {}
      }
    }
  },
} {
  for(size_t i = 0; i < tree.size(); i++) {
    auto& year = tree[i];
    const auto yearInCalendar { 2020 + i };

    for(size_t j = 0; j < year.size(); j++) {
      auto& month = year[j];
      const auto monthInCalendar = j + 1;

      if(monthInCalendar % 2 != 0) {
        if(monthInCalendar < 8) {
          month.push_back(std::vector {
            4, std::vector<Measurement> {}
          });
        }
      } else {
        if(monthInCalendar == 2) {
          if(yearInCalendar == 2020) {
            month.pop_back();
          } else {
            month.pop_back();
            month.pop_back();
          }
        } else if(monthInCalendar >= 8) {
          month.push_back(std::vector {
            4, std::vector<Measurement> {}
          });
        }
      }
    }
  }
}

void MeasurementsTree::generate_measurement_tree(std::vector<
 MeasurementRecord> records) {
  for(const auto& record : records) {
    const auto time { record.time };

    auto& toFill { tree[time.year - 2020][time.month - 1][time.
 day - 1][time.quarter - 1] };
```

```
49
50        toFill.push_back({
51          .autoconsumption = record.autoconsumption ,
52          .gridExport = record.gridExport      ,
53          .gridImport = record.gridImport      ,
54          .consumption = record.consumption    ,
55          .production = record.production      ,
56          .timeMinutes = record.time.inMinutes
57        });
58      }
59    }
60    MeasurementsTree::TreeType MeasurementsTree::get_tree(void) const
        {
61      return tree;
62    }
63    MeasurementsTree::Iterator MeasurementsTree::begin() {
64      return MeasurementsTree::Iterator(&tree, 0, 0, 0, 0, 0);
65    }
66
67    MeasurementsTree::Iterator MeasurementsTree::end() {
68      return Iterator(nullptr, tree.size());
69    }
```

Listing 9. Zawartość pliku `MeasurementsTree.cpp`

W wierszach od 3 do 42 znajduje się konstruktor. Tutaj następuje inicjalizacja drzewa `tree`. Najpierw mamy wektor roku o rozmiarze 2, czyli 2 lata. Każdy element wektora roku ma wektor miesiąc o wielkości 12 czyli 12 miesięcy w roku. Następnie każdy element wektora miesiąca ma wektor dnia o rozmiarze 30, czyli 30 dni w roku. Na koniec, każdy element wektora miesiąca ma wektor dnia o rozmiarze 4, czyli 4 kwarty dnia po 6 godzin. Pętla w wierszach od 12 do 44 jest odpowiedzialna za przechodzenie po latach i miesiącach oraz modyfikowanie dni w zależności od numeru miesiąca. Metoda w wierszach od 44 do 59 jest odpowiedzialna za wypełnianie struktury `tree` na podstawie wektora pomiarów. Najpierw za pomocą pętli for przechodzimy po każdym rekordzie, następnie znajdujemy odpowiednie miejsce w drzewie, wrzucamy `Measurement` i na koniec zwracamy całe drzewo. Metoda w wierszach od 60 do 62 jest odpowiedzialna za zwracanie całego drzewa. Metoda w wierszach od 63 do 65 jest odpowiedzialna za zwracanie iteratora na początek. Metoda w wierszach od 67 do 69 jest odpowiedzialna za zwracanie iteratora na koniec.

4.5. Google Test

Na listingu nr 10

```
#include <gtest/gtest.h>
#include "MeasurementsImporter.hpp"
#include "MeasurementsTree.hpp"
#include <print>

TEST(CsvImport, ReadTest) {
  MeasurementsImporter importer;

  importer.read_measurements("empty.csv");
}

TEST(CsvImport, FullReadTest) {
  MeasurementsImporter importer;

  importer.read_measurements("Chart Export.csv");
}

TEST(CsvImport, CorrectConversionTest) {
  std::vector compare = {
    MeasurementRecord {
      .time = {
        .year = 2020,
        .month = 10,
        .day = 1,
        .inMinutes = 0, // 0:00
        .quarter = 1,
      },
      .autoconsumption = 0.0,
      .gridExport = 0.0,
      .gridImport = 406.8323,
      .consumption = 406.8323,
      .production = 0.0
    },
    MeasurementRecord {
      .time = {
        .year = 2020,
        .month = 10,
        .day = 1,
        .inMinutes = 15, // 0:15
        .quarter = 1,
      },
      .autoconsumption = 0.0,
```

```
43          .gridExport = 0.0,
44          .gridImport = 403.5656,
45          .consumption = 403.5656,
46          .production = 0.0
47        },
48        MeasurementRecord {
49          .time = {
50            .year = 2020,
51            .month = 10,
52            .day = 1,
53            .inMinutes = 30, // 0:30
54            .quarter = 1,
55          },
56          .autoconsumption = 0.0,
57          .gridExport = 0.0,
58          .gridImport = 336.7334,
59          .consumption = 336.7334,
60          .production = 0.0
61        },
62        MeasurementRecord {
63          .time = {
64            .year = 2020,
65            .month = 10,
66            .day = 1,
67            .inMinutes = 975, // 16:15
68            .quarter = 3,
69          },
70          .autoconsumption = 119.3333,
71          .gridExport = 0.0,
72          .gridImport = 1871.7124,
73          .consumption = 1991.0458,
74          .production = 119.3333
75        },
76        MeasurementRecord {
77          .time = {
78            .year = 2021,
79            .month = 10,
80            .day = 31,
81            .inMinutes = 750, // 12:30
82            .quarter = 3,
83          },
84          .autoconsumption = 416.3987,
85          .gridExport = 3064.2681,
86          .gridImport = 0.0,
87          .consumption = 416.3987,
```

```
88          .production = 3480.6667
89        }
90        /* { 01.10.2020 0:00,"0","0","406.8323","406.8323","0" }, */
91        /* { 01.10.2020 0:15,"0","0","403.5656","403.5656","0" }, */
92        /* { 01.10.2020 0:30,"0","0","336.7334","336.7334","0" }, */
93        /* { 01.10.2020
      16:15,"119.3333","0","1871.7124","1991.0458","119.3333" } */
94        /* { 31.10.2021
      12:30,"416.3987","3064.2681","0","416.3987","3480.6667" }*/
95      };
96
97      MeasurementsImporter importer;
98      importer.read_measurements("tests.csv");
99      const auto measurements = importer.get_records();
100
101     ASSERT_EQ(measurements, compare);
102   }
103
104   TEST(MeasurementsTree, ConversionTest) {
105     std::vector input = {
106       MeasurementRecord {
107         .time = {
108           .year = 2020,
109           .month = 10,
110           .day = 1,
111           .inMinutes = 0, // 0:00
112           .quarter = 1,
113         },
114         .autoconsumption = 0.0,
115         .gridExport = 0.0,
116         .gridImport = 406.8323,
117         .consumption = 406.8323,
118         .production = 0.0
119       },
120       MeasurementRecord {
121         .time = {
122           .year = 2020,
123           .month = 10,
124           .day = 1,
125           .inMinutes = 15, // 0:15
126           .quarter = 1,
127         },
128         .autoconsumption = 0.0,
129         .gridExport = 0.0,
130         .gridImport = 403.5656,
```

```
131         .consumption = 403.5656,
132         .production = 0.0
133       },
134       MeasurementRecord {
135         .time = {
136           .year = 2020,
137           .month = 10,
138           .day = 1,
139           .inMinutes = 30, // 0:30
140           .quarter = 1,
141         },
142         .autoconsumption = 0.0,
143         .gridExport = 0.0,
144         .gridImport = 336.7334,
145         .consumption = 336.7334,
146         .production = 0.0
147       },
148       MeasurementRecord {
149         .time = {
150           .year = 2020,
151           .month = 10,
152           .day = 1,
153           .inMinutes = 975, // 16:15
154           .quarter = 3,
155         },
156         .autoconsumption = 119.3333,
157         .gridExport = 0.0,
158         .gridImport = 1871.7124,
159         .consumption = 1991.0458,
160         .production = 119.3333
161       },
162       MeasurementRecord {
163         .time = {
164           .year = 2021,
165           .month = 10,
166           .day = 31,
167           .inMinutes = 750, // 12:30
168           .quarter = 3,
169         },
170         .autoconsumption = 416.3987,
171         .gridExport = 3064.2681,
172         .gridImport = 0.0,
173         .consumption = 416.3987,
174         .production = 3480.6667
175       }
```

```
176     };
177
178     std::vector compare {
179       Measurement {
180         .autoconsumption = 0.0,
181         .gridExport = 0.0,
182         .gridImport = 406.8323,
183         .consumption = 406.8323,
184         .production = 0.0,
185         .timeMinutes = 0 // 0:00
186       },
187       Measurement {
188         .autoconsumption = 0.0,
189         .gridExport = 0.0,
190         .gridImport = 403.5656,
191         .consumption = 403.5656,
192         .production = 0.0,
193         .timeMinutes = 15 // 0:15
194       },
195       Measurement {
196         .autoconsumption = 0.0,
197         .gridExport = 0.0,
198         .gridImport = 336.7334,
199         .consumption = 336.7334,
200         .production = 0.0,
201         .timeMinutes = 30 // 0:30
202       },
203       Measurement {
204         .autoconsumption = 119.3333,
205         .gridExport = 0.0,
206         .gridImport = 1871.7124,
207         .consumption = 1991.0458,
208         .production = 119.3333,
209         .timeMinutes = 975 // 16:15
210       },
211       Measurement {
212         .autoconsumption = 416.3987,
213         .gridExport = 3064.2681,
214         .gridImport = 0.0,
215         .consumption = 416.3987,
216         .production = 3480.6667,
217         .timeMinutes = 750 // 12:30
218       }
219     };
220
```

```
221     MeasurementsTree tree;
222     tree.generate_measurement_tree(input);
223 
224     std::vector<Measurement> result;
225     for(const auto& yearArr : tree.get_tree()) {
226       for(const auto& monthArr : yearArr) {
227         for(const auto& dayArr : monthArr) {
228           for(const auto& quarter : dayArr) {
229             for(const Measurement& measurement : quarter) {
230               if(measurement != Measurement {}) {
231                 result.push_back(measurement);
232               }
233             }
234           }
235         }
236       }
237     }
238 
239     std::sort(result.begin(), result.end());
240     std::sort(compare.begin(), compare.end());
241     EXPECT_EQ(result, compare);
242   }
243 
244   TEST(MeasurementsTree, IteratorTest) {
245     std::vector input = {
246       MeasurementRecord {
247         .time = {
248           .year = 2020,
249           .month = 10,
250           .day = 1,
251           .inMinutes = 0, // 0:00
252           .quarter = 1,
253         },
254         .autoconsumption = 0.0,
255         .gridExport = 0.0,
256         .gridImport = 406.8323,
257         .consumption = 406.8323,
258         .production = 0.0
259       },
260       MeasurementRecord {
261         .time = {
262           .year = 2020,
263           .month = 10,
264           .day = 1,
265           .inMinutes = 15, // 0:15
```

```
266           .quarter = 1,
267         },
268         .autoconsumption = 0.0,
269         .gridExport = 0.0,
270         .gridImport = 403.5656,
271         .consumption = 403.5656,
272         .production = 0.0
273       },
274       MeasurementRecord {
275         .time = {
276           .year = 2020,
277           .month = 10,
278           .day = 1,
279           .inMinutes = 30, // 0:30
280           .quarter = 1,
281         },
282         .autoconsumption = 0.0,
283         .gridExport = 0.0,
284         .gridImport = 336.7334,
285         .consumption = 336.7334,
286         .production = 0.0
287       },
288       MeasurementRecord {
289         .time = {
290           .year = 2020,
291           .month = 10,
292           .day = 1,
293           .inMinutes = 975, // 16:15
294           .quarter = 3,
295         },
296         .autoconsumption = 119.3333,
297         .gridExport = 0.0,
298         .gridImport = 1871.7124,
299         .consumption = 1991.0458,
300         .production = 119.3333
301       },
302       MeasurementRecord {
303         .time = {
304           .year = 2021,
305           .month = 10,
306           .day = 31,
307           .inMinutes = 750, // 12:30
308           .quarter = 3,
309         },
310         .autoconsumption = 416.3987,
```

```
311         .gridExport = 3064.2681,
312         .gridImport = 0.0,
313         .consumption = 416.3987,
314         .production = 3480.6667
315       }
316     };
317
318     MeasurementsTree tree;
319     tree.generate_measurement_tree(input);
320
321     std::vector<Measurement> compare;
322     for(const auto& yearArr : tree.get_tree()) {
323       for(const auto& monthArr : yearArr) {
324         for(const auto& dayArr : monthArr) {
325           for(const auto& quarter : dayArr) {
326             for(const auto& measurement : quarter) {
327               if(measurement != Measurement {}) {
328                 compare.push_back(measurement);
329               }
330             }
331           }
332         }
333       }
334     }
335
336     std::vector<Measurement> result;
337
338     for(auto iter { tree.begin() }; iter != tree.end(); ++iter) {
339       result.push_back(*iter);
340     }
341
342     std::sort(result.begin(), result.end());
343     std::sort(compare.begin(), compare.end());
344     EXPECT_EQ(result, compare);
345   }
346
347   int main(int argc, char **argv) {
348     ::testing::InitGoogleTest(&argc, argv);
349     return RUN_ALL_TESTS();
350   }
```

Listing 10. Zawartość pliku `tests.cpp`

W wierszach od 6 do 16 są testy importera csv. Pierwszy przypadek to odczyt pustego pliku, drugi przypadek to pełny odczyt. W wierszach od 18 do 102 znajduje

się test poprawnej konwersji danych z CSV do `MeasurementRecord`. W wierszach od 104 do 242 znajduje się test konwersji `MeasurementRecord` do `MeasurementsTree`. W wierszach od 244 do 316 znajduje się test iteratora w `MeasurementsTree`. W wierszach 318 i 319 tworzymy i generujemy instancję drzewa pomiarów. W zagnieżdżonej pętli for w wierszach od 322 do 334 iterujemy po całym drzewie oraz zbieramy wszystkie pomiary. W wektorze w wierszu 336 przechowywane są pomiary. W wierszach od 342 do 344 wektory są sortowane oraz jest sprawdzane czy wynik zgadza się z oczekiwanym. W wierszach od 347 do 3550 znajduje się główny punkt wyjścia do uruchamiania testów.

5. Wnioski

- Konstrukcja drzewa binarnego przydaje się wtedy, kiedy mamy zamiar kilkakrotnie wyszukiwać dane z nieposortowanego zbioru
- Branchowanie w git bardzo pomaga w pracy kolaboracyjnej
- Rekurencja w pewnych algorytmach jest prostszym rozwiązaniem do zaimplementowania niż metody iteracyjne

Bibliografia

[1] *Artykuł Wikipedii o drzewie*. URL: https://en.wikipedia.org/wiki/Tree_(abstract_data_type).

[2] *Strona Gita*. URL: https://git-scm.com/.

[3] *Strona Doxygena*. URL: https://www.doxygen.nl/.

Spis rysunków

2.1. Puste repozytorium git 5
2.2. Stworznie pliku w repozytorium 5
2.3. Commit nr. 1 5
2.4. Commit nr. 2 5
2.5. Log gita 6
2.6. Demonstracja checkout 6
3.1. Interfejs programu doxywizard 8

Spis tabel

Spis listingów

1. Plik konfiguracyjny CMake 7
2. Dodanie Google Test do projektu 8
3. Dodanie Google Test do projektu 9
4. Zawartość pliku `Measurement.hpp` 10
5. Zawartość pliku `MeasurementRecord.hpp` 10
6. Zawartość pliku `MeasurementsImporter.hpp` 11
7. Zawartość pliku `MeasurementsImporter.cpp` 11
8. Zawartość pliku `MeasurementsTree.hpp` 16
9. Zawartość pliku `MeasurementsTree.cpp` 21
10. Zawartość pliku `tests.cpp` 24