

# AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynierskich  
Katedra Informatyki

## DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

### **Parsowanie plików**

Autor:  
Adrian Gargula  
Michał Bernardy

Prowadzący:  
mgr inż. Dawid Kotlarski

Nowy Sącz 2025

---

# Spis treści

<b>1. Ogólne określenie wymagań</b>	<b>3</b>
1.1. Cel Projektu . . . . .	3
1.2. Implementacja . . . . .	3
<b>2. Analiza problemu</b>	<b>5</b>
2.1. Parsowanie danych . . . . .	5
2.2. CSV . . . . .	5
2.3. Opis procesu parsowania . . . . .	6
<b>3. Projektowanie</b>	<b>7</b>
3.1. Wykorzystane Narzędzia . . . . .	7
3.1.1. Visual Studio 2022 . . . . .	7
3.1.2. Kompilator C++ . . . . .	7
3.1.3. Git . . . . .	7
3.1.4. GitHub Copilot . . . . .	7
<b>4. Implementacja</b>	<b>8</b>
4.1. Point.h . . . . .	8
4.2. Point.cpp . . . . .	9
4.3. Tree.h . . . . .	10
4.4. Tree.cpp . . . . .	11
4.5. FileManager.h . . . . .	14
4.6. FileManager.cpp . . . . .	14
4.7. Analizer.h . . . . .	18
4.8. Analizer.cpp . . . . .	20
4.9. Menu . . . . .	29
4.10. Działanie programu . . . . .	31
<b>Literatura</b>	<b>34</b>
<b>Spis rysunków</b>	<b>35</b>
<b>Spis listingów</b>	<b>36</b>

# 1. Ogólne określenie wymagań

## 1.1. Cel Projektu

Celem projektu jest stworzenie programu, który będzie parsować pliki o rozszerzeniu csv, a następnie wykona analizę danych zwartych w tych plikach. Pliki są zapisane w określonym formacie. Pierwsza linia jest linią informacyjną, która opisuje jakie są dane w pliku. Dane w pliku:

- Data i godzina pomiaru
- Autokonsumpcja [W]
- Eksport [W]
- Import [W]
- Pobór [W]
- Produkcja [W]

## 1.2. Implementacja

Wszystkie klasy w programie muszą zostać zaimplementowane w osobnych plikach.

Program powinien zawierać klasy odpowiedzialne za:

- Klasa, gdzie każda pojedyncza linijka zasila w dane jeden obiekt (punkt)
- Kklasa jest obiektem, który wygląda jak drzewo. Korzeniem drzewa jest rok, następnym poziomem jest miesiąc, następnie dzień oraz ćwiartka (6 godzin). Ćwiartki są cztery, a do każdej wczytywane są dane. Dane muszą zostać poukładane od najmniejszej godziny i minuty do największej godziny i minuty.
- Klasa zawiera całą mechanikę analizy danych

Przy poruszaniu się po węzłach należy zastosować wzorzec projektowy iterator. Dane mogą być niepełne.

Po uruchomieniu programu wyświetla się menu gdzie użytkownik wybiera dostępne opcje. Dodatkowo w menu dostępne są opcje wczytania pliku csv, zapisu do pliku binarnego wczytanych danych, odczyt z pliku binarnego danych (odczyt do pustego programu).

Przy wczytywaniu pliku z csv należy prowadzić bieżącą analizę i wychwytywać błędne linie. Przy wczytywaniu pliku csv program nie może zawiesić się. Program powinien tworzyć plik `log_data_godzina.txt` gdzie zapisuje postęp wczytywania (poprawne i niepoprawne rekordy) oraz plik `log_error_data_godzina.txt` gdzie zapisuje same niepoprawne rekordy. Niepoprawny rekord powinien być odrzucony i zapisany w obu plikach (`log_data_godzina.txt`, `log_error_data_godzina.txt`), szczególnie w pliku z błędami.

Data i godzina w nazwie pliku musi być pobierana z komputera automatycznie (tak aby nie nadpisywać plików). Po wczytaniu pliku program powinien wypisać podsumowanie na ekran (ile było poprawnych i niepoprawnych rekordów).

## 2. Analiza problemu

### 2.1. Parsowanie danych

Parsowanie danych (ang. data parsing) to proces analizy i konwersji danych z jednego formatu na inny. Jest to powszechnie stosowana operacja w programowaniu i przetwarzaniu danych, gdzie dane otrzymywane są w jednym formacie, a następnie są przetwarzane i interpretowane w celu ekstrakcji potrzebnych informacji.

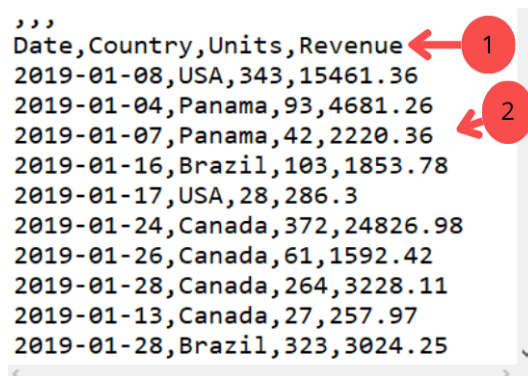
Podczas parsowania danych, dane wejściowe są analizowane w celu identyfikacji ich struktury i elementów składowych. To może obejmować analizę składniową, dzielenie na tokeny, wyciąganie kluczowych pól, rozpoznawanie wartości, typowanie danych itp. W zależności od rodzaju danych i formatu parsowanie może być stosowane do tekstu, plików binarnych, strumieni danych, dokumentów XML, JSON, CSV i wielu innych.<sup>[1]</sup>

### 2.2. CSV

To format przechowywania danych w formie tekstowej, gdzie wartości są oddzielone przecinkami. Jest to jedna z najpopularniejszych form tabelarycznego przechowywania danych, często używana w arkuszach kalkulacyjnych, bazach danych i różnych aplikacjach. W pliku CSV każdy rekord danych reprezentowany jest jako jedna linia, a poszczególne wartości w rekordzie są oddzielone przecinkami widoczne jest to na przykładowym rysunku 2.1(s.6).

---

<sup>1</sup>Parsowanie danych - słownik pojęć [\[1\]](#)



```
'''
Date,Country,Units,Revenue
2019-01-08,USA,343,15461.36
2019-01-04,Panama,93,4681.26
2019-01-07,Panama,42,2220.36
2019-01-16,Brazil,103,1853.78
2019-01-17,USA,28,286.3
2019-01-24,Canada,372,24826.98
2019-01-26,Canada,61,1592.42
2019-01-28,Canada,264,3228.11
2019-01-13,Canada,27,257.97
2019-01-28,Brazil,323,3024.25
'''
```

Rys. 2.1. Plik CSV

1. W tym przykładzie, pierwszy wiersz zawiera nagłówki kolumn.
2. Każdy kolejny wiersz reprezentuje dane oddzielone przecinkiem

## 2.3. Opis procesu parsowania

- Otwarcie pliku: Rozpoczynamy od otwarcia pliku, który zawiera dane CSV, za pomocą odpowiednich operacji wejścia/wyjścia.
- Odczyt linii: Wczytujemy kolejne linie z pliku, reprezentujące poszczególne rekordy w pliku CSV.
- Podział na pola: Dla każdej wczytanej linii przeprowadzamy proces podziału na pola. Najczęściej pola w pliku CSV są oddzielone konkretnym znakiem, najczęściej przecinkiem (,).
- Przetwarzanie pól: Po podziale linii na poszczególne pola, przetwarzamy te pola zgodnie z naszymi potrzebami.
- Zamykanie pliku: Po przetworzeniu wszystkich danych z pliku zamykamy go, aby zwolnić zasoby systemowe.

## 3. Projektowanie

### 3.1. Wykorzystane Narzędzia

#### 3.1.1. Visual Studio 2022

Visual Studio 2022 to zaawansowane zintegrowane środowisko programistyczne (IDE) opracowane przez firmę Microsoft. Jest to potężne narzędzie do tworzenia różnego rodzaju aplikacji, w tym aplikacji konsolowych w języku C++.

#### 3.1.2. Kompilator C++

Kompilator C++ jest niezbędny do przekształcenia kodu źródłowego napisanego w języku C++ na kod maszynowy, który może być uruchamiany na komputerze. Visual Studio zawiera wbudowany kompilator C++, który wykonuje to zadanie.

#### 3.1.3. Git

Git to rozproszony system kontroli wersji (VCS), który został stworzony przez Linusa Torvaldsa w 2005 roku. Git jest szeroko stosowanym narzędziem w dziedzinie programowania i zarządzania projektem, umożliwiającym śledzenie zmian w kodzie źródłowym oraz skuteczne zarządzanie projektem.

#### 3.1.4. GitHub Copilot

GitHub Copilot to narzędzie do generowania kodu, które korzysta z sztucznej inteligencji. Bazuje na modelu językowym stworzonym przez OpenAI (GPT-4) i działa jako rozszerzenie do środowisk programistycznych. GitHub Copilot może wspomagać programistów, dostarczając sugestie kodu w czasie rzeczywistym na podstawie wpisywanego tekstu i kontekstu kodu.

## 4. Implementacja

### 4.1. Point.h

```
1 #ifndef POINT_H
2 #define POINT_H
```

**Listing 1.** Header Guard

Na listingu 1 (s. 8) widzimy strażnik nagłówka (header guard). Zapobiegają one wielokrotnemu dołączaniu tego samego pliku nagłówkowego do jednostki translacyjnej. Jeśli POINT\_H nie jest zdefiniowane, zawartość pliku zostanie dołączona, a POINT\_H zostanie zdefiniowane. Jeśli już jest zdefiniowane, zawartość zostanie pominięta. Pomaga to uniknąć problemów z wielokrotnym dołączaniem tego samego nagłówka.

```
1 #include<iostream>
2 #include<string>
```

**Listing 2.** Dyrektywy include

Na listingu 2 (s. 8) widzimy dyrektywy pozwalają na korzystanie z funkcji i klas z bibliotek standardowych C++, takich jak iostream i string.

```
1 class Point {
2 public:
3     Point(const std::string & date, double autokonsumpcja, double
        eksport, double import, double pobor, double produkcja);
4
5     std::string date;
6     double autokonsumpcja;
7     double eksport;
8     double import;
9     double pobor;
10    double produkcja;
11};
```

**Listing 3.** Deklaracja klasy Point

Ten listing 3 (s. 8) definiuje klasę Point. Klasa ta ma publiczne pola (zmienne) i konstruktor. Pola obejmują datę (date) oraz różne wartości liczbowe związane z punktem danych. Konstruktor inicjalizuje te pola podczas tworzenia obiektu klasy.



```

1 class Point {
2 public:
3     Point(const std::string & date, double autokonsumpcja, double
      eksport, double import, double pobor, double produkcja);
4
5     std::string date;
6     double autokonsumpcja;
7     double eksport;
8     double import;
9     double pobor;
10    double produkcja;
11 };

```

**Listing 4.** Zakończenie strażnika nagłówka

Na listing 4 (s. 9) widać kończąca linie strażnik nagłówka dla kompilatora, informującym, że plik nagłówkowy został już dołączony.

## 4.2. Point.cpp

```

1 #include "Point.h"

```

**Listing 5.** Dyrektywa include

Na listingu 5 (s. 9) widzimy włączenie definicji klasy Point z pliku nagłówkowego "Point.h", co pozwala na korzystanie z tej klasy w bieżącym pliku źródłowym.

```

1 Point::Point(const std::string &date, double autokonsumpcja, double
      eksport, double import, double pobor, double produkcja)
2 : date(date), autokonsumpcja(autokonsumpcja), eksport(eksport),
      import(import), pobor(pobor), produkcja(produkcja){}

```

**Listing 6.** Implementacja konstruktora

Na listingu 6 (s. 9) widzimy:

- Point::Point - Rozpoczyna definicję konstruktora klasy Point.
- (const std::string &date, double autokonsumpcja, double eksport, double import, double pobor, double produkcja) sa to parametry konstruktora, które są przekazywane podczas tworzenia obiektu klasy Point
- : date(date), autokonsumpcja(autokonsumpcja), eksport(eksport), import(import), pobor(pobor), produkcja(produkcja) jest to lista inicjalizacyjna, ustawiająca pola klasy na wartości przekazane jako argumenty konstruktora.

### 4.3. Tree.h

```
1 #ifndef TREE_H
2 #define TREE_H
3 #include <map>
4 #include <vector>
5 #include "Point.h"
```

**Listing 7.** Dyrektywa include i strażnik nagłówka:

Na listingu 7 (s. 10) jest widoczny strażnik nagłówka (`#ifndef`, `#define`, `#endif`) jest używany do zapobiegania wielokrotnemu dołączaniu pliku nagłówkowego. Dyrektywy `include` zawierają deklaracje potrzebnych bibliotek i pliku nagłówkowego "Point.h", który zawiera definicję klasy `Point`.

```
1 class Tree {
2 private:
3     std::map<int, std::map<int, std::map<int, std::map<int, std::
4         vector<Point>>>>> tree;
5     int getCwiartka(int hour, int minute) const;
6 public:
7     Tree();
8
9     void dodanieDanych(const Point& dataPoint);
10    void pokazDrzewo() const;
11    std::vector<Point> getDataPoint(const std::string&
12        startDateTime, const std::string& endDateTime) const;
13};
```

**Listing 8.** Deklaracja klasy `Tree`:

Na listingu 8 (s. 10) jest widoczna definicja klasy `Tree`. Prywatna sekcja (`private`) zawiera strukturę drzewa (`tree`) oraz prywatną metodę `getCwiartka`. Publiczna sekcja (`public`) zawiera deklaracje konstruktora `Tree()`, metody `dodanieDanych`, `pokazDrzewo` i `getDataPoint`.

## 4.4. Tree.cpp

```

1 void Tree::dodanieDanych(const Point& dataPoint) {
2     int year, month, day, quarter;
3
4     std::istringstream dateStream(dataPoint.date.substr(0, 10)); //
5     Pobranie pierwszych 10 znak w jako data
6     dateStream.imbue(std::locale("pl_PL.utf8")); // Ustawienie
7     lokalizacji dla polskiego formatu daty
8
9     char dot; // Odczytuj kropk Ż oddzielaj c dzie , miesi c i
10    rok
11    dateStream >> day >> dot >> month >> dot >> year;
12
13    if (dateStream.fail()) {
14        std::cerr << "Error reading date: " << dataPoint.date <<
15        std::endl;
16        return;
17    }
18
19    quarter = getCwiartka(std::stoi(dataPoint.date.substr(11, 2)),
20    std::stoi(dataPoint.date.substr(14, 2)));
21
22    tree[year][month][day][quarter].push_back(dataPoint);
23 }

```

**Listing 9.** Metoda dodawania danych

Na listingu 9 (s. 11) widzimy metode `dodanieDanych` przyjmuje obiekt klasy `Point` i dodaje go do drzewa (`tree`). Analizuje datę, dzieląc ją na rok, miesiąc, dzień i ćwiartkę, a następnie dodaje punkt do odpowiedniego węzła drzewa.

```

1 void Tree::pokazDrzewo() const {
2     for (const auto& yearNode : tree) {
3         std::cout << "Rok: " << yearNode.first << std::endl;
4
5         for (int i = 0; i < 12; i++) {
6             if (yearNode.second.find(i + 1) != yearNode.second.end()) {
7                 std::cout << " Miesi c : " << i + 1 << std::endl;
8
9                 for (int j = 0; j < 31; j++) {
10                    auto& monthNode = yearNode.second.at(i + 1);
11                    if (monthNode.find(j + 1) != monthNode.end()) {
12                        std::cout << " Dzie : " << j + 1 << std::endl;
13
14                    for (int k = 0; k < 4; k++) {

```

```

15     auto& dayNode = monthNode.at(j + 1);
16     if (dayNode.find(k + 1) != dayNode.end()) {
17         std::cout << "           Ęwiartka : " << k + 1 << std::endl;
18
19         for (const auto& dataPoint : dayNode.at(k + 1)) {
20             // Wypisz dane punktu (dataPoint)
21             std::cout << "           Data: " << dataPoint.date
22                 << ", Autokonsumpcja: " << dataPoint.autokonsumpcja
23                 << ", Eksport: " << dataPoint.eksport
24                 << ", Import: " << dataPoint.import
25                 << ", Pob r: " << dataPoint.pobor
26                 << ", Produkcja: " << dataPoint.produkcja << std::
endl;
27         }
28     }
29 }
30 }

```

Listing 10. Metoda wyświetlająca drzewo

Na listingu 10 (s. 11) widzimy metode pokazDrzewo służy do wyświetlania drzewa danych na standardowym wyjściu. Iteruje przez drzewo, wyświetlając informacje o roku, miesiącu, dniu, ćwiartce i danych punktu.

```

1 std::vector<Point> Tree::getDataPoint(const std::string&
   startDateTime, const std::string& endDateTime) const {
2     std::vector<Point> dataPoints;
3
4     // Convert startDateTime and endDateTime to std::tm objects
5     std::tm startTm = {};
6     std::istringstream startStream(startDateTime);
7     startStream >> std::get_time(&startTm, "%d.%m.%Y %H:%M");
8
9     std::tm endTm = {};
10    std::istringstream endStream(endDateTime);
11    endStream >> std::get_time(&endTm, "%d.%m.%Y %H:%M");
12
13    // Iterate over the tree data structure
14    for (const auto& yearNode : tree) {
15        for (int i = 0; i < 12; i++) {
16            if (yearNode.second.find(i + 1) != yearNode.second.end()) {
17                for (int j = 0; j < 31; j++) {
18                    auto& monthNode = yearNode.second.at(i + 1);
19                    if (monthNode.find(j + 1) != monthNode.end()) {
20                        for (int k = 0; k < 4; k++) {
21                            auto& dayNode = monthNode.at(j + 1);

```

```
22     if (dayNode.find(k + 1) != dayNode.end()) {
23         for (const auto& dataPoint : dayNode.at(k + 1)) {
24             std::tm dataTm = {};
25             std::istringstream dataStream(dataPoint.date);
26             dataStream >> std::get_time(&dataTm, "%d.%m.%Y %H:%M");
27             if (std::mktime(&dataTm) >= std::mktime(&startTm) && std::mktime(&dataTm) <= std::mktime(&endTm)) {
28                 dataPoints.push_back(dataPoint);
29             }
30         }
31     }
32 }
33
34 return dataPoints;
35 }
```

**Listing 11.** Metoda pobierająca punkty danych w określonym zakresie czasowym

Na listingu 11 (s. 12) widzimy metodę `getDataPoint` zwraca wektor punktów danych, które mieszczą się w określonym zakresie czasowym. Konwertuje daty na obiekty `std::tm` i porównuje je z zakresem czasowym, a następnie zwraca pasujące punkty.

## 4.5. FileManager.h

```
1 #pragma once
```

**Listing 12.** Dyrektywa #pragma once

Na listingu 12 (s. 14) widzimy dyrektywa #pragma once jest używana do zapobiegania wielokrotnemu dołączaniu pliku nagłówkowego, podobnie jak tradycyjny strażnik nagłówka #ifndef, #define, #endif.

```
1 class FileManager {
2 private:
3     std::vector<std::string> split(const std::string& str, char
        delimiter);
4     std::string removeQuotes(const std::string& str);
5
6 public:
7     std::vector<Point> loadDataFromCsv(const std::string& filePath)
        ;
8     void saveDataToBinary(const Tree& dataTree, const std::string&
        filePath);
9     Tree loadDataFromBinary(const std::string& filePath);
10};
```

**Listing 13.** klasy FileManager

Na listingu 13 (s. 14) widzimy klasę FileManager służy do obsługi operacji związanych z zarządzaniem danymi, takimi jak wczytywanie i zapisywanie danych w różnych formatach.

## 4.6. FileManager.cpp

```
1 vector<string> FileManager::split(const string& str, char delimiter
    ) {
2     vector<string> tokens;
3     string token;
4     istringstream tokenStream(str);
5
6     while (getline(tokenStream, token, delimiter)) {
7         tokens.push_back(token);
8     }
9
10    return tokens;
11}
```

**Listing 14.** Metoda prywatna split

Na listingu 14 (s. 14) widzimy metode ktora dzieli łańcuch znaków (str) na wektor podłańcuchów, używając określonego separatora (delimiter). Jest używana do analizy linii danych CSV.

```
1 string FileManager::removeQuotes(const string& str) {
2     string newStr = str;
3
4     for (size_t i = 0; i < newStr.size(); ++i) {
5         if (newStr[i] == ',') {
6             newStr.erase(i, 1);
7         }
8     }
9
10    return newStr;
11 }
```

**Listing 15.** Metoda prywatna removeQuotes

Na listingu 15 (s. 15) widzimy metode która usuwa cudzysłowy z łańcucha znaków (str). Jest używana do usunięcia ewentualnych cudzysłowów z danych CSV. Na listingu 14 (s. 14) widzimy metode ktora dzieli łańcuch znaków (str) na wektor podłańcuchów, używając określonego separatora (delimiter). Jest używana do analizy linii danych CSV.

```
1 vector<Point> FileManager::loadDataFromCsv(const string& filePath)
2 {
3     vector<Point> dataPoints;
4
5     time_t rawtime;
6     struct tm* timeinfo;
7     char buffer[80];
8
9     time(&rawtime);
10    timeinfo = localtime(&rawtime);
11
12    strftime(buffer, sizeof(buffer), "%d-%m-%Y_%H", timeinfo);
13    std::string logTime(buffer);
14
15    string logFilePath = "log_" + logTime + ".txt";
16    string errorFilePath = "log_error_" + logTime + ".txt";
17
18    ofstream logFile(logFilePath);
19    ofstream errorFile(errorFilePath);
20    ifstream csvFile(filePath);
21
22    if (!logFile.is_open() || !errorFile.is_open()) {
```

```
22     cerr << "Wystapil blad przy otwieraniu pliku log" << endl;  
23     throw exception("Wystapil blad przy otwieraniu pliku log");  
24 }  
25  
26     if (!csvFile.is_open()) {  
27         cerr << "Wystapil blad przy otwieraniu " << filePath <<  
28         endl;  
29         errorFile << "Wystapil blad przy otwieraniu " << filePath  
30         << endl;  
31         return dataPoints;  
32     }  
33  
34     string line;  
35  
36     while (getline(csvFile, line)) {  
37         if (line.find("Time") != string::npos) {  
38             clog << "Pominieto naglowek: " << line << endl;  
39             logFile << "Pominieto naglowek: " << line << endl;  
40             continue;  
41         }  
42  
43         string dateTime;  
44         double autokonsumpcja, eksport, importD, pobor, produkcja;  
45  
46         vector<string> dataFromCsv = split(line, ',');  
47  
48         if (dataFromCsv.size() != 6) {  
49             cerr << "Bledna linia: " << line << endl;  
50             errorFile << "Bledna linia: " << line << endl;  
51             continue;  
52         }  
53  
54         bool isLineInvalid = false;  
55         for (size_t i = 0; i < dataFromCsv.size(); ++i) {  
56             dataFromCsv[i] = removeQuotes(dataFromCsv[i]);  
57  
58             if (dataFromCsv[i].empty()) {  
59                 cerr << "Brak wartosci w linii: " << line << endl;  
60                 errorFile << "Brak wartosci w linii: " << line << endl;  
61                 isLineInvalid = true;  
62             }  
63         }  
64  
65         if (isLineInvalid) {  
66             continue;  
67         }  
68     }  
69 }
```



```
65     }
66
67     dateTime = dataFromCsv[0];
68     autokonsumpcja = stod(dataFromCsv[1]);
69     eksport = stod(dataFromCsv[2]);
70     importD = stod(dataFromCsv[3]);
71     pobor = stod(dataFromCsv[4]);
72     produkcja = stod(dataFromCsv[5]);
73
74     if (dateTime.empty()) {
75         cerr << "Brak daty w linii: " << line << endl;
76         errorFile << "Brak daty w linii: " << line << endl;
77         continue;
78     }
79
80     Point* point = new Point(dateTime, autokonsumpcja, eksport,
81     importD, pobor, produkcja);
82     dataPoints.push_back(*point);
83     delete point;
84
85     if (dataPoints.empty()) {
86         cerr << "Nie wczytano danych" << endl;
87         errorFile << "Nie wczytano danych" << endl;
88         return dataPoints;
89     }
90
91     csvFile.close();
92     logFile.close();
93     errorFile.close();
94     return dataPoints;
95 }
```

**Listing 16.** Metoda publiczna loadDataFromCsv

Na listingu 16 (s. 15) widzimy metoda która wczytuje dane z pliku CSV (filePath) i zwraca wektor obiektów klasy Point. Obsługuje również błędy, logując je do pliku dziennika (logFile) oraz pliku błędów (errorFile).

```
1 void FileManager::saveDataToBinary(const Tree& dataTree, const
    string& filePath)
```

**Listing 17.** Metoda saveDataToBinary

Na listingu 17 (s. 18) widzimy metoda która odpowiedzialna jest za zapisywanie danych do pliku binarnego, jednak jej implementacja nie jest przedstawiona w dostarczonym kodzie.

```
1 Tree FileManager::loadDataFromBinary(const string& filePath)
```

**Listing 18.** Metoda loadDataFromBinary

Na listingu 18 (s. 18) widzimy metoda odpowiedzialna za odczytywanie danych z pliku binarnego, jednak jej implementacja nie jest przedstawiona w dostarczonym kodzie.

## 4.7. Analizer.h

```
1 #ifndef ANALYZER_H
2 #define ANALYZER_H
3 #include "Tree.h"
```

**Listing 19.** Dyrektywa include i strażnik nagłówka

Na listingu 19 (s. 18) widzimy strażnika nagłówka (#ifndef, #define, #endif) jest używany do zapobiegania wielokrotnemu dołączaniu pliku nagłówkowego. Dyrektywa include zawiera deklarację klasy Tree, której obiekty są przekazywane do analizy przez obiekt klasy DataAnalyzer.

```
1 #class DataAnalyzer {
2 private:
3     const Tree* tree;
4
5 public:
6     DataAnalyzer(const Tree* tree);
7
8     double sumAutokonsumpcja(const std::string& startDateTime,
9     const std::string& endDateTime) const;
10    double sumEksport(const std::string& startDateTime, const std::
11    string& endDateTime) const;
12    double sumImport(const std::string& startDateTime, const std::
13    string& endDateTime) const;
14    double sumPobor(const std::string& startDateTime, const std::
15    string& endDateTime) const;
16    double sumProdukcja(const std::string& startDateTime, const std
17    ::string& endDateTime) const;
```

```
13
14     double averageAutokonsumpcja(const std::string& startDateTime,
15     const std::string& endDateTime) const;
16     double averageEksport(const std::string& startDateTime, const
17     std::string& endDateTime) const;
18     double averageImport(const std::string& startDateTime, const
19     std::string& endDateTime) const;
20     double averagePobor(const std::string& startDateTime, const std
21     ::string& endDateTime) const;
22     double averageProdukcja(const std::string& startDateTime, const
23     std::string& endDateTime) const;
24
25     void compareAutokonsumpcja(const std::string& startDateTime1,
26     const std::string& endDateTime1,
27     const std::string& startDateTime2, const std::string&
28     endDateTime2) const;
29
30     void compareEksport(const std::string& startDateTime1, const
31     std::string& endDateTime1,
32     const std::string& startDateTime2, const std::string&
33     endDateTime2) const;
34
35     void compareImport(const std::string& startDateTime1, const std
36     ::string& endDateTime1,
37     const std::string& startDateTime2, const std::string&
38     endDateTime2) const;
39
40     void comparePobor(const std::string& startDateTime1, const std
41     ::string& endDateTime1,
42     const std::string& startDateTime2, const std::string&
43     endDateTime2) const;
44
45     void compareProdukcja(const std::string& startDateTime1, const
46     std::string& endDateTime1,
47     const std::string& startDateTime2, const std::string&
48     endDateTime2) const;
49
50     void searchAndPrintRecords(double searchedValue, double
51     tolerance, const std::string& startDateTime, const std::string&
52     endDateTime) const;
53
54     void printDataInRange(const std::string& startDateTime, const
55     std::string& endDateTime) const;
```

```
38 };
```

### Listing 20. Klasy DataAnalyzer

Na listingu 20 (s. 18) widzimy klasa DataAnalyzer służy do analizy danych zawartych w obiekcie klasy Tree. Przechowuje wskaźnik do obiektu Tree, który jest przekazywany przez konstruktor. Klasa zawiera funkcje do obliczania sum, średnich, porównań oraz wyszukiwania i wyświetlania rekordów w określonym zakresie czasowym.

## 4.8. Analizer.cpp

```
1 DataAnalyzer::DataAnalyzer(const Tree* tree) : tree(tree) {}
```

### Listing 21. Konstruktor

Na listingu 21 (s. 20) widzimy konstruktor klasy DataAnalyzer inicjalizuje pole prywatne tree obiektem klasy Tree, który zawiera dane do analizy. Wartość obiektu Tree jest przekazywana jako argument konstruktora.

```
1 double DataAnalyzer::sumAutokonsumpcja(const string& startDateTime,
    const string& endDateTime) const {
2     // Implementacja sumy autokonsumpcji
3     double suma = 0;
4
5     vector<Point> dataPoints = tree->getDataPoint(startDateTime,
        endDateTime);
6
7     for (auto& point : dataPoints) {
8         suma += point.autokonsumpcja;
9     }
10
11     return suma;
12 }
```

### Listing 22. Funkcja sumAutokonsumpcja

Na listingu 22 (s. 20) widzimy funkcja oblicza sumę wartości autokonsumpcji w określonym zakresie czasowym.

```
1 double DataAnalyzer::sumEksport(const string& startDateTime, const
    string& endDateTime) const {
2     // Implementacja sumy eksportu
3     double suma = 0;
4
5     vector<Point> dataPoints = tree->getDataPoint(startDateTime,
        endDateTime);
```

```
6
7     for (auto& point : dataPoints) {
8         suma += point.eksport;
9     }
10
11     return suma;
12 }
13
14 double DataAnalyzer::sumImport(const string& startDateTime, const
15     string& endDateTime) const {
16     // Implementacja sumy importu
17     double suma = 0;
18
19     vector<Point> dataPoints = tree->getDataPoint(startDateTime,
20     endDateTime);
21
22     for (auto& point : dataPoints) {
23         suma += point.import;
24     }
25
26     return suma;
27 }
28
29 double DataAnalyzer::sumPobor(const string& startDateTime, const
30     string& endDateTime) const {
31     // Implementacja sumy poboru
32     double suma = 0;
33
34     vector<Point> dataPoints = tree->getDataPoint(startDateTime,
35     endDateTime);
36
37     for (auto& point : dataPoints) {
38         suma += point.pobor;
39     }
40
41     return suma;
42 }
43
44 double DataAnalyzer::sumProdukcja(const string& startDateTime,
45     const string& endDateTime) const {
46     // Implementacja sumy produkcji
47     double suma = 0;
48
49     vector<Point> dataPoints = tree->getDataPoint(startDateTime,
50     endDateTime);
```

```

45
46     for (auto& point : dataPoints) {
47         suma += point.produkcja;
48     }
49
50     return suma;
51 }

```

**Listing 23.** Funkcje sumEksport, sumImport, sumPobor, sumProdukcja

Na listingu 23 (s. 20) widzimy funkcje obliczające sumę wartości dla sumEksport, sumImport, sumPobor, sumProdukcja w określonym zakresie czasowym.

```

1  double DataAnalyzer::averageAutokonsumpcja(const string&
   startDateTime, const string& endDateTime) const {
2      // Implementacja Średniej autokonsumpcji
3      double suma = 0;
4      int counter = 0;
5
6      vector<Point> dataPoints = tree->getDataPoint(startDateTime,
   endDateTime);
7
8      for (auto& point : dataPoints) {
9          suma += point.autokonsumpcja;
10         counter++;
11     }
12
13     return suma / counter;
14 }
15
16 double DataAnalyzer::averageEksport(const string& startDateTime,
   const string& endDateTime) const {
17     // Implementacja Średniej eksportu
18     double suma = 0;
19     int counter = 0;
20
21     vector<Point> dataPoints = tree->getDataPoint(startDateTime,
   endDateTime);
22
23     for (auto& point : dataPoints) {
24         suma += point.eksport;
25         counter++;
26     }
27
28     return suma / counter;
29 }

```

```
30
31 double DataAnalyzer::averageImport(const std::string& startDateTime
    , const std::string& endDateTime) const {
32     double suma = 0;
33     int counter = 0;
34
35     vector<Point> dataPoints = tree->getDataPoint(startDateTime,
    endDateTime);
36
37     for (auto& point : dataPoints) {
38         suma += point.import;
39         counter++;
40     }
41
42     return suma / counter;
43 }
44
45 double DataAnalyzer::averagePobor(const string& startDateTime,
    const string& endDateTime) const {
46     // Implementacja Źredniej poboru
47     double suma = 0;
48     int counter = 0;
49
50     vector<Point> dataPoints = tree->getDataPoint(startDateTime,
    endDateTime);
51
52     for (auto& point : dataPoints) {
53         suma += point.pobor;
54         counter++;
55     }
56
57     return suma / counter;
58 }
59
60 double DataAnalyzer::averageProdukcja(const string& startDateTime,
    const string& endDateTime) const {
61     // Implementacja Źredniej produkcji
62     double suma = 0;
63     int counter = 0;
64
65     vector<Point> dataPoints = tree->getDataPoint(startDateTime,
    endDateTime);
66
67     for (auto& point : dataPoints) {
68         suma += point.produkcja;
```

```

69     counter++;
70 }
71
72     return suma / counter;
73 }

```

**Listing 24.** Funkcje averageAutokonsumpcja, averageEksport, averageImport, averagePobor, averageProdukcja

Na listingu 24 (s. 22) widzimy funkcje która obliczają średnią wartość dla averageAutokonsumpcja, averageEksport, averageImport, averagePobor, averageProdukcja w określonym zakresie czasowym.

```

1 void DataAnalyzer::compareAutokonsumpcja(const string&
    startDateTime1, const string& endDateTime1,
2     const string& startDateTime2, const string& endDateTime2) const
    {
3     // Implementacja porównania autokonsumpcji
4     double suma1 = 0, suma2 = 0;
5     vector<Point> dataPoints1 = tree->getDataPoint(startDateTime1,
    endDateTime1);
6     vector<Point> dataPoints2 = tree->getDataPoint(startDateTime2,
    endDateTime2);
7     for (auto& point : dataPoints1) {
8         suma1 += point.autokonsumpcja;
9     }
10    for (auto& point : dataPoints2) {
11        suma2 += point.autokonsumpcja;
12    }
13    cout << "Suma autokonsumpcji w pierwszym przedziale: " << suma1
    << endl;
14    cout << "Suma autokonsumpcji w drugim przedziale: " << suma2 <<
    endl;
15    cout << "Różnica: " << suma1 - suma2 << endl;
16    cout << "Stosunek: " << suma1 / suma2 << endl;
17 }
18
19 void DataAnalyzer::compareEksport(const string& startDateTime1,
    const string& endDateTime1,
20     const string& startDateTime2, const string& endDateTime2) const
    {
21     // Implementacja porównania eksportu
22     double suma1 = 0, suma2 = 0;
23     vector<Point> dataPoints1 = tree->getDataPoint(startDateTime1,
    endDateTime1);
24     vector<Point> dataPoints2 = tree->getDataPoint(startDateTime2,

```



```
    endDateTime2);
25     for (auto& point : dataPoints1) {
26         suma1 += point.eksport;
27     }
28     for (auto& point : dataPoints2) {
29         suma2 += point.eksport;
30     }
31     cout << "Suma eksportu w pierwszym przedziale: " << suma1 <<
endl;
32     cout << "Suma eksportu w drugim przedziale: " << suma2 << endl;
33     cout << "R    nica: " << suma1 - suma2 << endl;
34     cout << "Stosunek: " << suma1 / suma2 << endl;
35 }
36
37 void DataAnalyze::compareImport(const string& startDateTime1,
    const string& endDateTime1,
38     const string& startDateTime2, const string& endDateTime2) const
    {
39     // Implementacja por wnanie importu
40     double suma1 = 0, suma2 = 0;
41     vector<Point> dataPoints1 = tree->getDataPoint(startDateTime1,
endl;
42     endDateTime1);
43     vector<Point> dataPoints2 = tree->getDataPoint(startDateTime2,
44     endDateTime2);
45     for (auto& point : dataPoints1) {
46         suma1 += point.import;
47     }
48     for (auto& point : dataPoints2) {
49         suma2 += point.import;
50     }
51     cout << "Suma importu w pierwszym przedziale: " << suma1 <<
endl;
52     cout << "Suma importu w drugim przedziale: " << suma2 << endl;
53     cout << "R    nica: " << suma1 - suma2 << endl;
54     cout << "Stosunek: " << suma1 / suma2 << endl;
55 }
56
57 void DataAnalyze::comparePobor(const string& startDateTime1, const
    string& endDateTime1,
58     const string& startDateTime2, const string& endDateTime2) const
    {
59     // Implementacja por wnanie poboru
60     double suma1 = 0, suma2 = 0;
    vector<Point> dataPoints1 = tree->getDataPoint(startDateTime1,
```

```

        endDateTime1);
61     vector<Point> dataPoints2 = tree->getDataPoint(startDateTime2,
        endDateTime2);
62     for (auto& point : dataPoints1) {
63         suma1 += point.pobor;
64     }
65     for (auto& point : dataPoints2) {
66         suma2 += point.pobor;
67     }
68     cout << "Suma poboru w pierwszym przedziale: " << suma1 << endl
;
69     cout << "Suma poboru w drugim przedziale: " << suma2 << endl;
70     cout << "R   nica: " << suma1 - suma2 << endl;
71     cout << "Stosunek: " << suma1 / suma2 << endl;
72
73 }
74
75 void DataAnalyze::compareProdukcja(const string& startDateTime1,
        const string& endDateTime1,
76     const string& startDateTime2, const string& endDateTime2) const
        {
77     // Implementacja porównania produkcji
78     double suma1 = 0, suma2 = 0;
79     vector<Point> dataPoints1 = tree->getDataPoint(startDateTime1,
        endDateTime1);
80     vector<Point> dataPoints2 = tree->getDataPoint(startDateTime2,
        endDateTime2);
81     for (auto& point : dataPoints1) {
82         suma1 += point.produkcja;
83     }
84     for (auto& point : dataPoints2) {
85         suma2 += point.produkcja;
86     }
87     cout << "Suma produkcji w pierwszym przedziale: " << suma1 <<
        endl;
88     cout << "Suma produkcji w drugim przedziale: " << suma2 << endl
;
89     cout << "R   nica: " << suma1 - suma2 << endl;
90     cout << "Stosunek: " << suma1 / suma2 << endl;
91 }

```

Listing 25. Funkcje porównujące

Na listingu 25 (s. 24) widzimy funkcje która porównują sumy wartości w dwóch różnych okresach czasowych dla odpowiednich parametrów.

```
1 void DataAnalyze::searchAndPrintRecords(double searchedValue,
2     double tolerance,
3     const string& startDateTime, const string& endDateTime) const {
4     // Implementacja wyszukiwania i wypisywania rekord w
5     vector<Point> dataPoints = tree->getDataPoint(startDateTime,
6     endDateTime);
7     for (auto& point : dataPoints) {
8         if (point.autokonsumpcja >= searchedValue - tolerance &&
9         point.autokonsumpcja <= searchedValue + tolerance) {
10         cout << "Data: " << point.date << endl;
11         cout << "Autokonsumpcja: " << point.autokonsumpcja <<
12         endl;
13         cout << "Eksport: " << point.eksport << endl;
14         cout << "Import: " << point.import << endl;
15         cout << "Pobor: " << point.pobor << endl;
16         cout << "Produkcja: " << point.produkcja << endl;
17     }
18     else if (point.eksport >= searchedValue - tolerance &&
19     point.eksport <= searchedValue + tolerance) {
20         cout << "Data: " << point.date << endl;
21         cout << "Autokonsumpcja: " << point.autokonsumpcja <<
22         endl;
23         cout << "Eksport: " << point.eksport << endl;
24         cout << "Import: " << point.import << endl;
25         cout << "Pobor: " << point.pobor << endl;
26         cout << "Produkcja: " << point.produkcja << endl;
27     }
28     else if (point.import >= searchedValue - tolerance && point.
29     .import <= searchedValue + tolerance) {
30         cout << "Data: " << point.date << endl;
31         cout << "Autokonsumpcja: " << point.autokonsumpcja <<
32         endl;
33         cout << "Eksport: " << point.eksport << endl;
34         cout << "Import: " << point.import << endl;
35         cout << "Pobor: " << point.pobor << endl;
```

```

36         cout << "Produkcja: " << point.produkcja << endl;
37     }
38     else if (point.produkcja >= searchedValue - tolerance &&
point.produkcja <= searchedValue + tolerance) {
39         cout << "Data: " << point.date << endl;
40         cout << "Autokonsumpcja: " << point.autokonsumpcja <<
endl;
41         cout << "Eksport: " << point.eksport << endl;
42         cout << "Import: " << point.import << endl;
43         cout << "Pobor: " << point.pobor << endl;
44         cout << "Produkcja: " << point.produkcja << endl;
45     }
46 }
47 }

```

**Listing 26.** Funkcja searchAndPrintRecords

Na listingu 26 (s. 26) widzimy funkcję która wyszukuje i wypisuje rekordy spełniające określone kryteria (wartość, tolerancja, zakres czasowy) dla wszystkich parametrów.

```

1 void DataAnalyzer::printDataInRange(const string& startDateTime,
const string& endDateTime) const {
2     // Implementacja wypisywania danych w przedziale
3     vector<Point> dataPoints = tree->getDataPoint(startDateTime,
endDateTime);
4     for (auto& point : dataPoints) {
5         cout << "Data: " << point.date << endl;
6         cout << "Autokonsumpcja: " << point.autokonsumpcja << endl;
7         cout << "Eksport: " << point.eksport << endl;
8         cout << "Import: " << point.import << endl;
9         cout << "Pobor: " << point.pobor << endl;
10        cout << "Produkcja: " << point.produkcja << endl;
11    }
12 }

```

**Listing 27.** Funkcja printDataInRange

Na listingu 27 (s. 28) widzimy funkcję która wypisuje wszystkie dane zawarte w określonym zakresie czasowym dla wszystkich parametrów.

## 4.9. Menu

```
1 Menu::Menu() : tree(new Tree()), analyzer(new DataAnalyzer(tree)),
   choice(0) {}
```

**Listing 28.** Konstruktor

Na listingu 28 (s. 29) widzimy konstruktor klasy Menu inicjalizuje obiekty tree (klasa Tree), analyzer (klasa DataAnalyzer), fileManager (klasa FileManager) oraz pole choice.

```
1 Menu::~~Menu() {
2     delete tree;
3     delete analyzer;
4 }
```

**Listing 29.** Destruktor

Na listingu 29 (s. 29) widzimy destruktora który zwalnia pamięć zajmowaną przez obiekty tree i analyzer

```
1 void Menu::printMenu() {
2     cout << "-----Parsowanie CSV-----" << endl;
3     cout << "1. Wczytaj plik CSV" << endl;
4     cout << "2. Zapisz do pliku binarnego" << endl;
5     cout << "3. Wczytaj z pliku binarnego" << endl;
6     cout << "4. Wyświetl wszystkie dane" << endl;
7     cout << "5. Wypisz dane w przedziale" << endl;
8     cout << "6. Wypisz sume autokonsumpcji" << endl;
9     cout << "7. Wypisz sume eksportu" << endl;
10    cout << "8. Wypisz sume importu" << endl;
11    cout << "9. Wypisz sume poboru" << endl;
12    cout << "10. Wypisz sume produkcji" << endl;
13    cout << "11. Wypisz srednia autokonsumpcji" << endl;
14    cout << "12. Wypisz srednia eksportu" << endl;
15    cout << "13. Wypisz srednia importu" << endl;
16    cout << "14. Wypisz srednia poboru" << endl;
17    cout << "15. Wypisz srednia produkcji" << endl;
18    cout << "16. Porównaj autokonsumpcje wyznaczony w dwóch
   przedziałach" << endl;
19    cout << "17. Porównaj eksport wyznaczony w dwóch przedziałach"
   << endl;
20    cout << "18. Porównaj import wyznaczony w dwóch przedziałach"
   << endl;
21    cout << "19. Porównaj pobor wyznaczony w dwóch przedziałach" <<
   endl;
22    cout << "20. Porównaj produkcje wyznaczona w dwóch przedziałach
   " << endl;
```

```

23     cout << "21. Wyszukaj i wypisz rekordy" << endl;
24     cout << "0. Wyjdź z programu" << endl;
25     cout << "Podaj numer opcji: " << endl;
26 }

```

**Listing 30.** Funkcja printMenu

Na listingu 30 (s. 29) widzimy funkcja która wypisuje menu na ekranie, prezentując dostępne opcje użytkownikowi.

```

1 void Menu::start() {
2
3 }

```

**Listing 31.** Funkcja printMenu

Na listingu 31 (s. 30) widzimy funkcja która rozpoczyna działanie programu, obsługując wybory użytkownika.

```

1 void Menu::loadDataFromCsv();
2 void Menu::saveDataToBinary();
3 void Menu::loadDataFromBinary();

```

**Listing 32.** Funkcje do wczytywania i zapisywania danych

Na listingu 32 (s. 30) widzimy funkcje która obsługuje wczytywanie danych z pliku CSV, zapisywanie danych do pliku binarnego oraz wczytywanie danych z pliku binarnego.

```

1 void Menu::printAllData();
2 void Menu::printDataInRange();

```

**Listing 33.** Funkcje do wypisywania danych

Na listingu 33 (s. 30) widzimy funkcje która wypisują wszystkie dostępne dane oraz dane zawarte w określonym zakresie czasowym.

```

1 void Menu::sumAutoconsumption();
2 void Menu::sumExport();
3 void Menu::sumImport();
4 void Menu::sumDemand();
5 void Menu::sumProduction();
6 void Menu::avarageAutoconsumption();
7 void Menu::avarageExport();
8 void Menu::avarageImport();
9 void Menu::avarageDemand();
10 void Menu::avarageProduction();

```

**Listing 34.** Funkcje obliczające sumy i średnie

Na listingu 34 (s. 30) widzimy funkcje która obliczają sumy i średnie dla różnych parametrów.

```
1 void Menu::compareAutoconsumption();
2 void Menu::compareExport();
3 void Menu::compareImport();
4 void Menu::compareDemand();
5 void Menu::compareProduction();
```

**Listing 35.** Funkcje porównujące

Na listingu 35 (s. 31) widzimy funkcje która porównuje sumy wartości w dwóch różnych okresach czasowych dla różnych parametrów.

```
1 void Menu::searchAndPrintRecords();
```

**Listing 36.** Funkcja searchAndPrintRecords

Na listingu 36 (s. 31) widzimy funkcje która obsługuje wyszukiwanie i wypisywanie rekordów spełniających określone kryteria.

## 4.10. Działanie programu

Rysunek 4.1 (s. 33) ukazuje menu programu. Program oferuje następujące opcje:

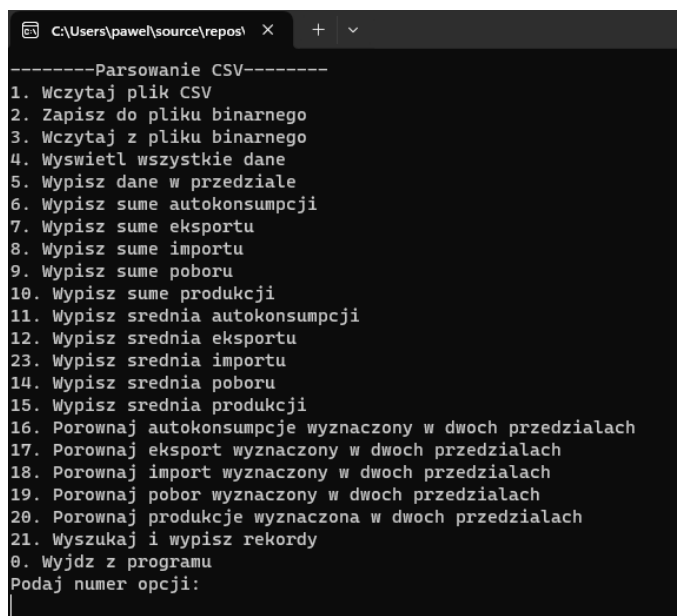
- 1. Wczytaj plik CSV - opcja ta umożliwia użytkownikowi załadowanie pliku CSV do bazy danych.
- 2. Zapisz do pliku binarnego - opcja ta umożliwia użytkownikowi zapisanie danych z bazy danych do pliku binarnego.
- 3. Wczytaj z pliku binarnego - opcja ta umożliwia użytkownikowi załadowanie danych z pliku binarnego do bazy danych.
- 4. Wyświetl wszystkie dane - opcja ta umożliwia użytkownikowi wyświetlenie wszystkich danych z bazy danych.
- 5. Wypisz dane w przedziale - opcja ta umożliwia użytkownikowi wyświetlenie danych z bazy danych w określonym przedziale.
- 6. Wypisz sumę autokonsumpcji - opcja ta umożliwia użytkownikowi wyświetlenie sumy autokonsumpcji z bazy danych.
- 7. Wypisz sumę eksportu - opcja ta umożliwia użytkownikowi wyświetlenie sumy eksportu z bazy danych.

- 8. Wypisz sumę importu - opcja ta umożliwia użytkownikowi wyświetlenie sumy importu z bazy danych.
- 9. Wypisz sumę poboru - opcja ta umożliwia użytkownikowi wyświetlenie sumy poboru z bazy danych.
- 10. Wypisz sumę produkcji - opcja ta umożliwia użytkownikowi wyświetlenie sumy produkcji z bazy danych.
- 11. Wypisz średnią autokonsumpcji - opcja ta umożliwia użytkownikowi wyświetlenie średniej autokonsumpcji z bazy danych.
- 12. Wypisz średnią eksportu - opcja ta umożliwia użytkownikowi wyświetlenie średniej eksportu z bazy danych.
- 13. Wypisz średnią importu - opcja ta umożliwia użytkownikowi wyświetlenie średniej importu z bazy danych.
- 14. Wypisz średnią poboru - opcja ta umożliwia użytkownikowi wyświetlenie średniej poboru z bazy danych.
- 15. Wypisz średnią produkcji - opcja ta umożliwia użytkownikowi wyświetlenie średniej produkcji z bazy danych.
- 16. Porównaj autokonsumpcje wyznaczone w dwóch przedziałach - opcja ta umożliwia użytkownikowi porównanie sumy autokonsumpcji w dwóch określonych przedziałach.
- 17. Porównaj eksport wyznaczony w dwóch przedziałach - opcja ta umożliwia użytkownikowi porównanie sumy eksportu w dwóch określonych przedziałach.
- 18. Porównaj import wyznaczony w dwóch przedziałach - opcja ta umożliwia użytkownikowi porównanie sumy importu w dwóch określonych przedziałach.
- 19. Porównaj pobór wyznaczony w dwóch przedziałach - opcja ta umożliwia użytkownikowi porównanie sumy poboru w dwóch określonych przedziałach.
- 20. Porównaj produkcje wyznaczona w dwóch przedziałach - opcja ta umożliwia użytkownikowi porównanie sumy produkcji w dwóch określonych przedziałach.
- 21. Wyszukaj i wypisz rekordy - opcja ta umożliwia użytkownikowi wyszukanie i wyświetlenie rekordów z bazy danych.



- 0. Wyjdź z programu - opcja ta umożliwia użytkownikowi wyjście z programu.

W przypadku uruchomienia programu bez podania żadnej opcji, program wyświetli komunikat i ponownie wypisze menu.



```

C:\Users\pawel\source\repos\ x + v
-----Parsowanie CSV-----
1. Wczytaj plik CSV
2. Zapisz do pliku binarnego
3. Wczytaj z pliku binarnego
4. Wyświetl wszystkie dane
5. Wypisz dane w przedziale
6. Wypisz sume autokonsumpcji
7. Wypisz sume eksportu
8. Wypisz sume importu
9. Wypisz sume poboru
10. Wypisz sume produkcji
11. Wypisz srednia autokonsumpcji
12. Wypisz srednia eksportu
13. Wypisz srednia importu
14. Wypisz srednia poboru
15. Wypisz srednia produkcji
16. Porównaj autokonsumpcje wyznaczone w dwóch przedziałach
17. Porównaj eksport wyznaczony w dwóch przedziałach
18. Porównaj import wyznaczony w dwóch przedziałach
19. Porównaj pobor wyznaczony w dwóch przedziałach
20. Porównaj produkcje wyznaczona w dwóch przedziałach
21. Wyszukaj i wypisz rekordy
0. Wyjdź z programu
Podaj numer opcji:

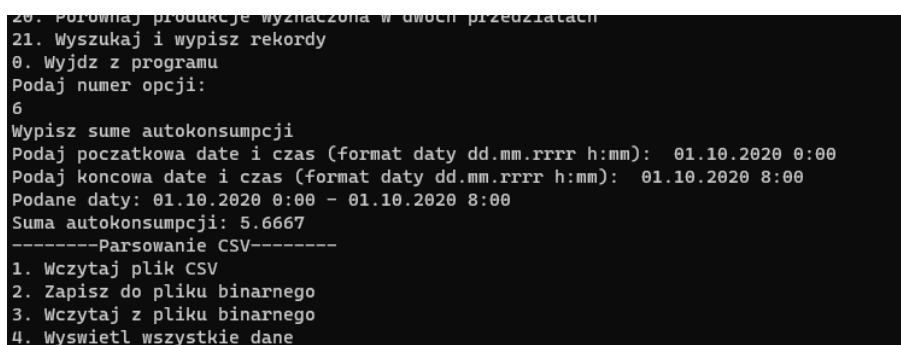
```

Rys. 4.1. Menu

Program na rysunku 4.2 (s. 33) najpierw prosi użytkownika o podanie daty i godziny początkowej (w formacie dd.mm.rrrr h:mm). W tym przypadku użytkownik podał datę 01.10.2020 0:00.

Następnie program prosi użytkownika o podanie daty i godziny końcowej. W tym przypadku użytkownik podał datę 01.10.2020 8:00.

Po wprowadzeniu tych danych program oblicza sumę autokonsumpcji dla danych z podanego przedziału czasowego. W tym przypadku suma autokonsumpcji wynosi 5.6667.



```

20. Porównaj produkcje wyznaczona w dwóch przedziałach
21. Wyszukaj i wypisz rekordy
0. Wyjdź z programu
Podaj numer opcji:
6
Wypisz sume autokonsumpcji
Podaj początkowa date i czas (format daty dd.mm.rrrr h:mm): 01.10.2020 0:00
Podaj końcowa date i czas (format daty dd.mm.rrrr h:mm): 01.10.2020 8:00
Podane daty: 01.10.2020 0:00 - 01.10.2020 8:00
Suma autokonsumpcji: 5.6667
-----Parsowanie CSV-----
1. Wczytaj plik CSV
2. Zapisz do pliku binarnego
3. Wczytaj z pliku binarnego
4. Wyświetl wszystkie dane

```

Rys. 4.2. Działanie programu

## Bibliografia

- [1] *Parsowanie danych*. URL: <https://twojedrzwidoit.pl/slownik-pojec-technicznych/parsowanie-danych>.

## Spis rysunków

2.1. Plik CSV . . . . .	6
4.1. Menu . . . . .	33
4.2. Działanie programu . . . . .	33

## Spis listingów

1.	Header Guard . . . . .	8
2.	Dyrektywy include . . . . .	8
3.	Deklaracja klasy Point . . . . .	8
4.	Zakończenie strażnika nagłówka . . . . .	9
5.	Dyrektywa include . . . . .	9
6.	Implementacja konstruktora . . . . .	9
7.	Dyrektywa include i strażnik nagłówka: . . . . .	10
8.	Deklaracja klasy Tree: . . . . .	10
9.	Metoda dodawania danych . . . . .	11
10.	Metoda wyświetlająca drzewo . . . . .	11
11.	Metoda pobierająca punkty danych w określonym zakresie czasowym	12
12.	Dyrektywa #pragma once . . . . .	14
13.	klasy FileManager . . . . .	14
14.	Metoda prywatna split . . . . .	14
15.	Metoda prywatna removeQuotes . . . . .	15
16.	Metoda publiczna loadDataFromCsv . . . . .	15
17.	Metoda saveDataToBinary . . . . .	18
18.	Metoda loadDataFromBinary . . . . .	18
19.	Dyrektywa include i strażnik nagłówka . . . . .	18
20.	Klasy DataAnalyzer . . . . .	18
21.	Konstruktor . . . . .	20
22.	Funkcja sumAutokonsumpcja . . . . .	20
23.	Funkcje sumEksport, sumImport, sumPobor, sumProdukcja . . . . .	20
24.	Funkcje averageAutokonsumpcja, averageEksport, averageImport, ave- ragePobor, averageProdukcja . . . . .	22
25.	Funkcje porównujące . . . . .	24
26.	Funkcja searchAndPrintRecords . . . . .	26
27.	Funkcja printDataInRange . . . . .	28
28.	Konstruktor . . . . .	29
29.	Destruktor . . . . .	29
30.	Funkcja printMenu . . . . .	29

31.	Funkcja printMenu . . . . .	30
32.	Funkcje do wczytywania i zapisywania danych . . . . .	30
33.	Funkcje do wypisywania danych . . . . .	30
34.	Funkcje obliczające sumy i średnie . . . . .	30
35.	Funkcje porównujące . . . . .	31
36.	Funkcja searchAndPrintRecords . . . . .	31