

**Politechnika Śląska**  
**Wydział Automatyki, Elektroniki i**  
**Informatyki**

*Programowanie*  
*Komputerów 3*

**„Mikroprojekt:**  
**Implementacja listy”**

<u>autor:</u>	Bartosz Heliński
<u>kierunek:</u>	informatyka
<u>rok akademicki:</u>	2022/2023
<u>rodzaj studiów:</u>	SSI
<u>semestr:</u>	3
<u>gr. dz.:</u>	I
<u>prowadzący:</u>	dr inż. Jolanta Kawulok
<u>termin laboratorium:</u>	pon. 14:45 – 16:15
<u>sekcja:</u>	2
<u>data sprawozdania:</u>	22-01-2023

## 1.1 Treść zadania

1) Należy zaimplementować listę jednokierunkową lub dwukierunkową lub drzewo binarne w postaci klasy jako szablon bez użycia kontenerów STL.

Należy w szablonie klasy zawrzeć:

- inteligentne wskaźniki,
- konstruktory kopiujące/przenoszące, konstruktor bezargumentowy,
- destruktor
- operatory przypisania/przeniesienia,
- dodawanie elementów do kontenera,
- usuwanie wybranego elementu,
- wyszukiwanie elementu,
- sortowanie zawartości (różne kryteria sortowania, wybór czy rosnąco, czy malejąco),
- serializacja i deserializacja (zapis i odczyt z plików binarnych);

2) Należy zaimplementować dodatkowe klasy (min 2), na których będzie testowana w/w klasa (szablon). Dla tych dodatkowych klas należy dopisać metodę umożliwiającą wczytywanie z pliku tekstowego danych (np. w pliku jest lista z opisem POJAZDOW).

Przykłady:

- Drzewo genealogiczne (w postaci drzewa) testowane na osobach, zwierzętach i roślinach
- Bazy danych w których można trzymać różne typy (osoby, pojazdy, leki)

## 1.2 Ogólny opis

Zgodnie z poleceniem zaimplementowany został szablon listy jednokierunkowej bez pomocy kontenerów STL. Zawiera on wszystkie wyżej wymienione elementy oraz większość funkcjonalności zwykłej listy takie jak: `push_front`, `push_back`, `pop_front`, itd. (nie zaimplementowane zostały m.in. funkcje `merge` oraz `swap`). Szablon jest kompatybilny ze wszystkimi podstawowymi typami jak np. `int`, `double`, `string...`, oraz dwoma bardziej złożonymi strukturami przechowującymi różne rodzaje danych, które omówione są w dalszej części sprawozdania.

## 2.1 Struktura programu

W aplikacji istnieje łącznie 5 klas w hierarchii:

-działanie listy:

\*Node

\*List

-przechowywanie struktur z danymi:

\*Tab1

\*Tab2

-interfejs programu:

\*Interfejs

## 2.2 Specyfikacja

Program został zrealizowany zgodnie z paradygmatem strukturalnym. Podzielony został na dwa pliki nagłówkowe .h (jeden zawierający szablon listy a drugi pozostałe klasy) oraz plik .cpp zawierający metody tych klas dla łatwiejszej orientacji w kodzie. Użyte zostały proste biblioteki służące m.in. do odczytu i zapisu danych (`fstream`), dostosowania wypisywania części danych w konsoli aby wyglądały lepiej (`iomanip`) oraz zatrzymywania czasu aby użytkownik mógł zapoznać się z wyskakującymi na ekran komunikatami (`windows.h`).

## 2.3 Szczegóły strukturalne

Szczegółowy opis klas, metod oraz funkcji zawarty jest w dołączonej dokumentacji stworzonej za pomocą doxygena poniżej, oraz w pliku na platformie git.

### 3.1 Szczegółowy opis

Tematem projektu jest stworzenie prostego w obsłudze rejestru więziennego, przechowującego dane osobowe więźniów, jak również różne informacje o ich pobycie w więzieniu, do czego służą dwie pierwsze zaimplementowane klasy (Tab1 oraz Tab2). Pierwsza zawiera informacje o więźniach (id, imię, nazwisko, datę urodzenia oraz płeć). Druga przechowuje informacje o karze więziennej (ponownie id, przewinienie, długość kary oraz ich cele w wybranym bloku więziennym). Przykłady danych wejściowych podane są poniżej w części „Testowanie”.

Program zapewnia użytkownikowi obszerne menu z wieloma opcjami oraz czterema stworzonymi listami. Dwie z nich są uzupełniane danymi z plików przy starcie programu, natomiast pozostałe służą do zestawiania danych oraz tworzenia różnych kombinacji informacji, które użytkownik chce w nich zapisać. Opcjami w menu są oczywiście wszystkie podstawowe opcje dostępne w szablonie listy jak np. różne formy dodawania czy usuwania elementów z listy, jak również stworzone specjalnie na potrzeby projektu funkcje takie jak: lista osób urodzonych w danym przedziale czasowym, lista osób osadzonych w konkretnym bloku więziennym, średnia długość kary, bądź średnia długość kary dłuższa niż ta zadana przez użytkownika, lista powiązań np. elementów w tymczasowej liście więziennej z listą danych osobowych, której wynikiem jest tymczasowa lista tychże osób, itd. Na podstawie tych oraz innych możliwości użytkownik jest w stanie sortować dane z list jak tylko zechce. Następnie może je wyświetlić na ekran, do czego znowu jest klika opcja, bądź zapisać je do pliku (pojedyncze listy, wszystkie, jednorazowo lub ciągiem jak w logu). Poniżej znajduje się zdjęcie wyglądu menu oraz jego opcji, wraz z przykładem użycia sortowania.

Program uruchamiany jest z linii poleceń. Podawane parametry programu mogą być w dowolnej kolejności (specjalna funkcja sprawdza ich poprawność oraz dostosowuje kolejność):

- i1 → plik wejściowy z danymi osobowymi
- i2 → plik wejściowy z danymi więziennymi
- o → plik wyjściowy z zapisywanymi informacjami

### Przykład użycia opcji „Sort”:

```

Rejestr wiezienny
1. Push
2. Pop
3. Remove
4. Get element
5. Get index
6. Sort 1
7. Count
8. Arrest length
9. Avarage arrest length
10. Avarage length greater than
11. Prison block check
12. Birth date check
13. Lists connections
14. Display
15. Clear
16. Save
0. Exit
Wybierz opcje:

Wybierz liste:
1. Tabela z danymi osobowymi 2
2. Tabela z danymi wieziennymi
3. Tabela z tymczasowymi danymi osobowymi
4. Tabela z tymczasowymi danymi wieziennymi

Jak chcesz posortowac dane?
1. Rosnaco 3
2. Malejaco

Wedlug czego chcesz posortowac?
1-id, 2-imie, 3-nazwisko, 4-data urodzenia, 5-plec
4

Lista zawiera:
991012153, Judyta, Kuc, 12.10.1999, k
991202173, Maciej, Adamczyk, 2.12.2000, m
990816274, Piotr, Kit, 16.8.1999, m

Lista zawiera:
990816274, Piotr, Kit, 16.8.1999, m
991012153, Judyta, Kuc, 12.10.1999, k
991202173, Maciej, Adamczyk, 2.12.2000, m

```

## 4.1 Testowanie

W programie przetestowano wszystkie dostępne użytkownikowi opcje pod kątem potencjalnych błędów oraz wycieków pamięci. Testy przeprowadzone zostały na różnych rozmiarach danych (od kilku do kilkuset). Wszystkie próby przebiegły pomyślnie, bez żadnych błędów. Dla większej ilości danych zauważalny był jedynie niewielki spadek wydajności sortowania (przy testach rzędu setek), coraz większy wraz z większą ich ilością (dla tysiąca był już zdecydowanie widoczny). Przykłady danych wejściowych podane zostały poniżej. Jedyną możliwą sytuacją prowadzącą do błędu aplikacji jest podanie znaku nie będącego liczbą w miejsca, gdzie należy wpisać typ int, bądź podanie nieprawidłowych parametrów wejściowych przy starcie programu (wypisywany jest wtedy komunikat o błędzie a program należy uruchomić ponownie). Aplikacja została również przetestowana pod kątem wycieków pamięci, których nie znaleziono (wynik testu znajduje się poniżej).

## Przykłady danych wejściowych:

### Plik nr 1:

991012153, Judyta, Kuc, 12.10.1999, k  
 991202173, Maciej, Adamczyk, 02.12.2000, m  
 990816274, Piotr, Kit, 16.08.1999, m

### Plik nr 2:

991012153, morderstwo, 20 lat, 21A  
 991202173, kradziez, 10 miesiecy, 15C  
 990816274, grozby, 1 miesiac, 3C

## Wynik testów wycieków pamięci:

Module Path	Size	Proces...	Timest...	Sequen...	Thread Id
<b>No leaks found</b> Check filters above: "Module", "Leak type" and "Filters"					

## 4.2 Dodatkowe informacje

Cały program wraz z dokumentacją oraz przykładowymi plikami z większą ilości danych wejściowych wrzucony został na platformę git. Dokumentacja strukturalna programu znajduje się poniżej.

## 5 Wnioski

Rejestr więzienny jako mikroprojekt z użycia szablonu listy jednokierunkowej to stosunkowo prosty w założeniach program, posiadający jednak bardzo przyjazny interfejs, który daje ogromne możliwości dobierania i sortowania danych wedle potrzeb. Zawiera wszystkie wymagane w projekcie elementy. Działa bez zarzutów a testy wykazały brak wycieków pamięci. Wszystkie klasy oraz metody zostały skomentowane, gdyby któraś z nich była mniej jasna. Jedyną wadą programu jest „mikswanie” słów w języku polskim z języki angielskim. Nie jest to niepoprawne ale w przyszłości będę zwracał na to większą uwagę, aby trzymać się jednego języka. Podczas tworzenia projektu nie napotkano większych trudności. Pozwolił on na lepsze zaznajomienie się z szablonami, inteligentnymi wskaźnikami oraz innymi ważnymi elementami programowania.

# Class Index

## Class List

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

<b>Tab1::Dane</b>	4
<b>Interfejs</b>	6
<b>List&lt; T &gt;</b>	7
<b>Node&lt; T &gt;</b>	13
<b>Tab1</b>	14
<b>Tab2</b>	16
<b>Tab2::Wiezienie</b>	18

# File Index

## File List

Here is a list of all files with brief descriptions:

<b>classes.cpp</b>	.....20
<b>classes.h</b>	.....35
<b>Rejestr_wiezienny.cpp</b>	.....38
<b>template.h</b>	.....40



# Class Documentation

## Tab1::Dane Struct Reference

```
#include <classes.h>
```

### Public Member Functions

- **Dane** ()
- **Dane** (int **id**, string **imie**, string **nazwisko**, int **dzien**, int **miesiac**, int **rok**, char **plec**)

### Public Attributes

- int **id**
- string **imie**
- string **nazwisko**
- int **dzien**
- int **miesiac**
- int **rok**
- char **plec**

---

### Detailed Description

Struktura przechowująca dane osobowe takie jak id, imie, nazwisko, data urodzenia oraz plec. Posiada 2 konstruktory - bezargumentowy z wartościami domyślnymi oraz wieloargumentowy.

Definition at line 17 of file **classes.h**.

---

### Constructor & Destructor Documentation

**Tab1::Dane::Dane ()** [inline]

Konstruktor bezargumentowy z wartościami domyślnymi.

Definition at line 28 of file **classes.h**.

**Tab1::Dane::Dane (int *id*, string *imie*, string *nazwisko*, int *dzien*, int *miesiac*, int *rok*, char *plec*)** [inline]

Konstruktor wieloargumentowy.

Definition at line 31 of file **classes.h**.

---

### Member Data Documentation

**int Tab1::Dane::dzien**

Definition at line 21 of file **classes.h**.

**int Tab1::Dane::id**

Definition at line 18 of file **classes.h**.

**string Tab1::Dane::imie**

Definition at line **19** of file **classes.h**.

**int Tab1::Dane::miesiac**

Definition at line **22** of file **classes.h**.

**string Tab1::Dane::nazwisko**

Definition at line **20** of file **classes.h**.

**char Tab1::Dane::plec**

Definition at line **24** of file **classes.h**.

**int Tab1::Dane::rok**

Definition at line **23** of file **classes.h**.

---

**The documentation for this struct was generated from the following file:**

- **classes.h**

## Interfejs Class Reference

```
#include <classes.h>
```

### Public Member Functions

- void **menu** ()
  - int **wybor** ()
  - void **blad** ()
  - void **program** (List< Tab1::Dane > &, List< Tab2::Wiezienie > &, const string &)
  - void **zapis\_clear** (const string &)
- 

### Detailed Description

Klasa odpowiedzialna za wyswietlanie interfejsu uzytkownikowi.

Definition at line **139** of file **classes.h**.

---

### Member Function Documentation

#### void Interfejs::blad ()

Metoda wyswietlajaca komunikat o bledzie wyboru.

Definition at line **137** of file **classes.cpp**.

#### void Interfejs::menu ()

Metoda wyswietlajaca liste opcji do wyboru w menu.

Definition at line **102** of file **classes.cpp**.

#### void Interfejs::program (List< Tab1::Dane > & *lista1*, List< Tab2::Wiezienie > & *lista2*, const string & *o*)

Metoda odpowiedzialna za dzialanie calego interfejsu. Mozna w niej dokonywac wyborow operacji oraz listy.

Definition at line **143** of file **classes.cpp**.

#### int Interfejs::wybor ()

Metoda wyswietlajaca wszystkie listy, na ktorych mozna dokonywac roznych operacji.

Definition at line **125** of file **classes.cpp**.

#### void Interfejs::zapis\_clear (const string & *nazwa*)

Metoda czyszczaca plik wyjsciowy.

Definition at line **961** of file **classes.cpp**.

---

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

- **classes.h**
- **classes.cpp**

## List< T > Class Template Reference

```
#include <template.h>
```

### Public Member Functions

- **List** ()
- **List** (const **List** &lista)
- **List** (**List** &&lista)
- const shared\_ptr< **Node**< T > > & **get\_head** () const
- void **push\_front** (const T &)
- void **push\_back** (const T &)
- void **pop\_front** ()
- void **pop\_back** ()
- void **pop** (const int)
- T **get\_element** (const int)
- T **get\_element2** (const int)
- int **get\_index** (const T &)
- int **get\_index** (const int)
- void **remove** (const T &)
- bool **greater** (const T &, const T &)
- bool **smaller** (const T &, const T &)
- bool **greater** (const **Tab1::Dane** &, const **Tab1::Dane** &, const int)
- bool **smaller** (const **Tab1::Dane** &, const **Tab1::Dane** &, const int)
- bool **greater** (const **Tab2::Wiezienie** &, const **Tab2::Wiezienie** &, const int)
- bool **smaller** (const **Tab2::Wiezienie** &, const **Tab2::Wiezienie** &, const int)
- void **sort** (const int)
- void **sort** (const int, const int)
- int **element\_count** (const string &)
- int **element\_count1** (const int, const string &)
- int **element\_count2** (const int, const string &)
- int **arrest\_length** (const int)
- double **arrest\_average** (const string &)
- **List**< T > **avarage\_greater** (const double, const string &)
- **List**< T > **block\_check** (const char)
- **List**< T > **born\_during** (const int, const int, const int, const int, const int, const int)
- void **clear** ()
- int **size** ()
- bool **is\_empty** ()
- void **serialization** (const string &)
- void **deserialization** (const string &)
- void **serialization\_string** (const string &)
- void **deserialization\_string** (const string &)
- void **displayElement** (const int)
- void **displayList** ()
- void **zapis** (const string &)
- void **zapis2** (const string &)
- template<typename T , typename U > **List**< U > **connections** (const **List**< U > &lista)
- **List** & **operator=** (const **List** &lista)
- **List** & **operator=** (**List** &&lista)
- ~**List** ()

### Friends

- class **Tab1**
- class **Tab2**

---

## Detailed Description

**template<typename T>**

**class List< T >**

Najwazniejsza klasa w programie. Jest to template listy jednokierunkowej. Posiada rozne metody, operatory oraz konstrukotry pozwalajace na rozno rodzaju operacje na liscie. Kompatybilna ze wszystkimi podstawowymi typami danych oraz m.in. testowanymi w tym programie dwoma klasami. Zaprzyjzniona z klasami, na ktorych jest testowana czyli **Tab1** oraz **Tab2**.

Definition at line 37 of file **template.h**.

---

## Constructor & Destructor Documentation

**template<typename T > List< T >::List () [inline]**

Konstruktor bezargumentowy z wartoscia domyslna head.

Definition at line 45 of file **template.h**.

**template<typename T > List< T >::List (const List< T > & lista) [inline]**

Konstruktor kopiujacy.

Definition at line 48 of file **template.h**.

**template<typename T > List< T >::List (List< T > && lista) [inline]**

Konstruktor przenoszacy.

Definition at line 57 of file **template.h**.

**template<typename T > List< T >::~~List () [inline]**

Destruktor klasy **Node**.

Definition at line 228 of file **template.h**.

---

## Member Function Documentation

**template<typename T > double List< T >::arrest\_average (const string & co)**

Metoda mowiaca co sredniej dlugosci pobytu w areszcie wybranego lub wszystkich elementow.

Definition at line 685 of file **template.h**.

**template<typename T > int List< T >::arrest\_length (const int id)**

Metoda wyliczajaca dlugosc pobytu w areszcie wybranego elementu.

Definition at line 667 of file **template.h**.

**template<typename T > List< T > List< T >::avarage\_greater (const double *length*, const string & *co*)**

Metoda zwracajaca liste elementow, ktorzy dlugosc pobytu w areszcie jest dluzna niz podana przez uzytkownika.

Definition at line 714 of file **template.h**.

**template<typename T > List< T > List< T >::block\_check (const char *blok*)**

Metoda zwracajaca liste elementow, ktore sa z bloku wieziennego podanego przez uzytkownika.

Definition at line 765 of file **template.h**.

**template<typename T > List< T > List< T >::born\_during (const int *from\_day*, const int *from\_month*, const int *from\_year*, const int *to\_day*, const int *to\_month*, const int *to\_year*)**

Metoda zwracajaca liste elementow, ktorzy data urodzin miesci sie w przedziale podanym przez uzytkownika.

Definition at line 777 of file **template.h**.

**template<typename T > void List< T >::clear**

Metoda czyszczaca liste.

Definition at line 819 of file **template.h**.

**template<typename T > template<typename T , typename U > List< U > List< T >::connections (const List< U > & *lista*)[inline]**

Metoda zwracajaca liste elementow powiazanych przez id z inna lista.

Definition at line 183 of file **template.h**.

**template<typename T > void List< T >::deserialization (const string & *nazwa*)**

Metoda odpowiedzialna za deserializacje danych w pliku binarnym do listy (opocz stringow).

Definition at line 857 of file **template.h**.

**template<typename T > void List< T >::deserialization\_string (const string & *nazwa*)**

Metoda odpowiedzialna za deserializacje danych w pliku binarnym do listy (tylko stringi).

Definition at line 887 of file **template.h**.

**template<typename T > void List< T >::displayElement (const int *index*)**

Metoda odpowiedzialna za wyswietlenie wybranego elementu w liscie.

Definition at line 907 of file **template.h**.

**template<typename T > void List< T >::displayList**

Metoda odpowiedzialna za wyswietlenie calej listy.

Definition at line 931 of file **template.h**.

**template<typename T > int List< T >::element\_count (const string & *obj*)**

Metoda zliczajaca wybrany element w liscie.

Definition at line 577 of file **template.h**.

**template<typename T > int List< T >::element\_count1 (const int *element*, const string & *obj*)**

Metoda zliczajaca wybrany element w liscie typu **Tab1::Dane**.

Definition at line **589** of file **template.h**.

**template<typename T > int List< T >::element\_count2 (const int *element*, const string & *obj*)**

Metoda zliczajaca wybrany element w liscie typu **Tab2::Wiezienie**.

Definition at line **632** of file **template.h**.

**template<typename T > T List< T >::get\_element (const int *index*)**

Metoda zwracajaca element w liscie o zadany indexie.

Definition at line **294** of file **template.h**.

**template<typename T > T List< T >::get\_element2 (const int *id*)**

Metoda zwracajaca element w liscie o zadany id.

Definition at line **308** of file **template.h**.

**template<typename T > const shared\_ptr< Node< T > > & List< T >::get\_head () const [inline]**

Metoda zwracajaca wskaznik na head.

Definition at line **68** of file **template.h**.

**template<typename T > int List< T >::get\_index (const int *id*)**

Metoda zwracajaca index elementu o wskazanym id.

Definition at line **338** of file **template.h**.

**template<typename T > int List< T >::get\_index (const T & *dane*)**

Metoda zwracajaca index wybranego elementu.

Definition at line **321** of file **template.h**.

**template<class T > bool List< T >::greater (const T & *x*, const T & *y*)**

Metoda mowiaca czy jeden element jest wiekszy od drugiego.

Definition at line **373** of file **template.h**.

**template<class T > bool List< T >::greater (const Tab1::Dane & *a*, const Tab1::Dane & *b*, const int *choice*)**

Metoda mowiaca czy jeden element typu **Tab1::Dane** jest wiekszy od drugiego.

Definition at line **383** of file **template.h**.

**template<class T > bool List< T >::greater (const Tab2::Wiezienie & *a*, const Tab2::Wiezienie & *b*, const int *choice*)**

Metoda mowiaca czy jeden element typu **Tab2::Wiezienie** jest wiekszy od drugiego.

Definition at line **441** of file **template.h**.

**template<typename T > bool List< T >::is\_empty**

Metoda mowiaca o tym czy lista jest pusta.

Definition at line 837 of file **template.h**.

**template<typename T > List & List< T >::operator= (const List< T > & *lista*) [inline]**

Operator przypisania.

Definition at line 199 of file **template.h**.

**template<typename T > List & List< T >::operator= (List< T > && *lista*) [inline]**

Operator przeniesienia.

Definition at line 213 of file **template.h**.

**template<typename T > void List< T >::pop (const int *index*)**

Metoda usuwajaca element z wybranego miejsca w liscie.

Definition at line 279 of file **template.h**.

**template<typename T > void List< T >::pop\_back**

Metoda usuwajaca element z konca listy.

Definition at line 264 of file **template.h**.

**template<typename T > void List< T >::pop\_front**

Metoda usuwajaca element z poczatku listy.

Definition at line 258 of file **template.h**.

**template<typename T > void List< T >::push\_back (const T & *obj*)**

Metoda dodajaca element na koniec listy.

Definition at line 242 of file **template.h**.

**template<typename T > void List< T >::push\_front (const T & *obj*)**

Metoda dodajaca element na poczatek listy.

Definition at line 232 of file **template.h**.

**template<typename T > void List< T >::remove (const T & *obj*)**

Metoda usuwajaca wybrany element z listy.

Definition at line 355 of file **template.h**.

**template<typename T > void List< T >::serialization (const string & *nazwa*)**

Metoda odpowiedzialna za serializacje danych w liscie do pliku binarnego (opocz stringow).

Definition at line 844 of file **template.h**.

**template<typename T > void List< T >::serialization\_string (const string & *nazwa*)**

Metoda odpowiedzialna za serializacje danych w liscie do pliku binarnego (tylko stringi).

Definition at line 871 of file **template.h**.

**template<typename T > int List< T >::size**

Metoda zwracajaca rozmiar listy.

Definition at line 826 of file **template.h**.



**template<class T > bool List< T >::smaller (const T & x, const T & y)**

Metoda mowiaca czy jeden element jest mniejszy od drugiego.

Definition at line 378 of file **template.h**.

**template<class T > bool List< T >::smaller (const Tab1::Dane & a, const Tab1::Dane & b, const int choice)**

Metoda mowiaca czy jeden element typu **Tab1::Dane** jest mniejszy od drugiego.

Definition at line 412 of file **template.h**.

**template<class T > bool List< T >::smaller (const Tab2::Wiezienie & a, const Tab2::Wiezienie & b, const int choice)**

Metoda mowiaca czy jeden element typu **Tab2::Wiezienie** jest mniejszy od drugiego.

Definition at line 476 of file **template.h**.

**template<typename T > void List< T >::sort (const int sort)**

Metoda sortujaca cala liste.

Definition at line 511 of file **template.h**.

**template<typename T > void List< T >::sort (const int sort, const int choice)**

Metoda sortujaca wybrany rodzaj danych w liscie.

Definition at line 544 of file **template.h**.

**template<typename T > void List< T >::zapis (const string & nazwa)**

Metoda odpowiedzialna za zapis listy do pliku.

Definition at line 954 of file **template.h**.

**template<typename T > void List< T >::zapis2 (const string & nazwa)**

Metoda odpowiedzialna za zapis listy do pliku bez czyszczenia zawartosci pliku (log).

Definition at line 981 of file **template.h**.

---

## Friends And Related Function Documentation

**template<typename T > friend class Tab1 [friend]**

Definition at line 38 of file **template.h**.

**template<typename T > friend class Tab2 [friend]**

Definition at line 39 of file **template.h**.

---

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

- **template.h**

## Node< T > Class Template Reference

```
#include <template.h>
```

### Public Member Functions

- `Node (T obj)`
- `~Node ()`

### Friends

- `template<typename U > class List`

---

### Detailed Description

`template<typename T>`

**class Node< T >**

Klasa template-owa przedstawiajaca pojedynczy element (**Node**) w liscie. Zaprzyjazniona z klasa glowna czyli Lista. Posiada element typu T zawierajaca dane oraz wskaznik na kolejny element listy.

Definition at line **14** of file **template.h**.

---

### Constructor & Destructor Documentation

`template<typename T > Node< T >::Node (T obj) [inline]`

Konstruktor jednoargumentowy.

Definition at line **23** of file **template.h**.

`template<typename T > Node< T >::~~Node () [inline]`

Destruktor klasy **Node**.

Definition at line **26** of file **template.h**.

---

### Friends And Related Function Documentation

`template<typename T > template<typename U > friend class List [friend]`

Definition at line **15** of file **template.h**.

---

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

- `template.h`

## Tab1 Class Reference

```
#include <classes.h>
```

### Classes

- struct **Dane**

### Public Member Functions

- void **odczyt** (const string &, List< **Tab1::Dane** > &)
- ~**Tab1** ()

### Friends

- template<typename U > class **List**
- bool **operator==** (const **Tab1::Dane** &dane1, const **Tab1::Dane** &dane2)
- bool **operator<** (const **Tab1::Dane** &dane1, const **Tab1::Dane** &dane2)
- istream & **operator>>** (istream &in, **Tab1::Dane** &dane)

---

### Detailed Description

Pierwsza z klas zawierajaca w sobie strukture odpowiedzialna za przechowywanie danych osobowych. Zaprzyjazona z szablonem klasy Lista.

Definition at line **10** of file **classes.h**.

---

### Constructor & Destructor Documentation

**Tab1::~Tab1 ()** [**inline**]

Destruktor klasy **Tab1**.

Definition at line **69** of file **classes.h**.

---

### Member Function Documentation

**void Tab1::odczyt** (const string & *nazwa*, List< **Tab1::Dane** > & *lista*)

Metoda odpowiedzialna za odczyt danych z pliku.

Definition at line **12** of file **classes.cpp**.

---

### Friends And Related Function Documentation

template<typename U > friend class List [**friend**]

Definition at line **11** of file **classes.h**.

**bool operator<** (const **Tab1::Dane** & *dane1*, const **Tab1::Dane** & *dane2*) [**friend**]

Operator mniejszosci.

Definition at line **44** of file **classes.h**.

**bool operator== (const Tab1::Dane & *dane1*, const Tab1::Dane & *dane2*)[friend]**

Operator porownania.

Definition at line **39** of file **classes.h**.

**istream & operator>> (istream & *in*, Tab1::Dane & *dane*)[friend]**

Operator wejscia strumieniowego.

Definition at line **63** of file **classes.h**.

---

**The documentation for this class was generated from the following files:**

- **classes.h**
- **classes.cpp**

## Tab2 Class Reference

```
#include <classes.h>
```

### Classes

- struct **Wiezienie**

### Public Member Functions

- void **odczyt** (const string &, List< **Tab2::Wiezienie** > &)
- ~**Tab2** ()

### Friends

- template<typename U > class **List**
  - bool **operator==** (const **Tab2::Wiezienie** &w11\_1, const **Tab2::Wiezienie** &w11\_2)
  - bool **operator<** (const **Tab2::Wiezienie** &w11\_1, const **Tab2::Wiezienie** &w11\_2)
  - istream & **operator>>** (istream &in, **Tab2::Wiezienie** &w11)
- 

### Detailed Description

Druga z klas zawierajaca w sobie strukture odpowiedzialna za przechowywanie danych wieziennych. Zaprzyjazniona z szablonem klasy Lista.

Definition at line **78** of file **classes.h**.

---

### Constructor & Destructor Documentation

**Tab2::~~Tab2 ()** [**inline**]

Destruktor klasy **Tab2**.

Definition at line **131** of file **classes.h**.

---

### Member Function Documentation

**void Tab2::odczyt** (const string & *nazwa*, List< **Tab2::Wiezienie** > & *lista*)

Metoda odpowiedzialna za odczyt danych z pliku.

Definition at line **43** of file **classes.cpp**.

---

### Friends And Related Function Documentation

template<typename U > friend class List [**friend**]

Definition at line **79** of file **classes.h**.

**bool operator<** (const **Tab2::Wiezienie** & *w11\_1*, const **Tab2::Wiezienie** & *w11\_2*) [**friend**]

Operator mniejszosci.

Definition at line **110** of file **classes.h**.

**bool operator== (const Tab2::Wiezienie & w11\_1, const Tab2::Wiezienie & w11\_2)[friend]**

Operator porownania.

Definition at line **105** of file **classes.h**.

**istream & operator>> (istream & in, Tab2::Wiezienie & w11)[friend]**

Operator wejscia strumieniowego.

Definition at line **125** of file **classes.h**.

---

**The documentation for this class was generated from the following files:**

- **classes.h**
- **classes.cpp**

## Tab2::Wiezienie Struct Reference

```
#include <classes.h>
```

### Public Member Functions

- **Wiezienie** ()
- **Wiezienie** (int **id**, string **przewinienie**, int **dlugosc**, string **jednostka**, string **cela**)

### Public Attributes

- int **id**
- string **przewinienie**
- int **dlugosc**
- string **jednostka**
- string **cela**

---

### Detailed Description

Struktura przechowujaca dane wiezienne takie jak id, przewinienie, dlugosc kary oraz numer celi wraz z blokiem wieziennym. Posiada 2 konstruktory - bezargumentowy z wartosciami domyslnymi oraz wieloargumentowy.

Definition at line **85** of file **classes.h**.

---

### Constructor & Destructor Documentation

#### Tab2::Wiezienie::Wiezienie () [inline]

Konstruktor bezargumentowy z wartosciami domyslnymi.

Definition at line **94** of file **classes.h**.

#### Tab2::Wiezienie::Wiezienie (int *id*, string *przewinienie*, int *dlugosc*, string *jednostka*, string *cela*) [inline]

Konstruktor wieloargumentowy.

Definition at line **97** of file **classes.h**.

---

### Member Data Documentation

#### string Tab2::Wiezienie::cela

Definition at line **90** of file **classes.h**.

#### int Tab2::Wiezienie::dlugosc

Definition at line **88** of file **classes.h**.

#### int Tab2::Wiezienie::id

Definition at line **86** of file **classes.h**.

**string Tab2::Wiezienie::jednostka**

Definition at line **89** of file **classes.h**.

**string Tab2::Wiezienie::przewinienie**

Definition at line **87** of file **classes.h**.

---

**The documentation for this struct was generated from the following file:**

- **classes.h**



# File Documentation

## classes.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <windows.h>
#include <iomanip>
#include "classes.h"
#include "template.h"
```

## Functions

- void **parameters\_check** (int argc, char \*argv[], string &i1, string &i2, string &o)
- 

## Function Documentation

**void parameters\_check** (int *argc*, char \* *argv*[], string & *i1*, string & *i2*, string & *o*)

Funkcja odpowiedzialna za sprawdzenie poprawnosci argumentow podanych w konsoli.  
Przypisuje odpowiednie nazwy plikow do danych wejsciowych/wyjsciowych.

Definition at line **971** of file **classes.cpp**.

## classes.cpp

```
Go to the documentation of this file.00001 #pragma once
00002 #include<iostream>
00003 #include<fstream>
00004 #include<sstream>
00005 #include<windows.h>
00006 #include<iomanip>
00007 #include "classes.h"
00008 #include "template.h"
00009
00010 using namespace std;
00011
00012 void Tab1::odczyt(const string& nazwa, List<Tab1::Dane>& lista) {
00013     ifstream plik(nazwa);
00014     if (plik) {
00015         Dane dane;
00016         string linia;
00017         while (getline(plik, linia)) {
00018             stringstream ss(linia);
00019
00020             getline(ss, linia, ',');
00021             dane.id = stoi(linia);
00022
00023             getline(ss >> ws, dane.imie, ',');
00024             getline(ss >> ws, dane.nazwisko, ',');
00025
00026             getline(ss, linia, '.');
00027             dane.dzien = stoi(linia);
00028             getline(ss, linia, '.');
00029             dane.miesiac = stoi(linia);
00030             getline(ss, linia, ',');
00031             dane.rok = stoi(linia);
00032
00033             ss >> dane.plec;
00034
00035             lista.push back(dane);
00036         }
00037         plik.close();
00038     }
00039 }
00040
00041
00042
00043 void Tab2::odczyt(const string& nazwa, List<Tab2::Wiezienie>& lista) {
00044     ifstream plik(nazwa);
00045     if (plik) {
00046         Wiezienie w11;
00047         string linia;
00048         while (getline(plik, linia)) {
00049             stringstream ss(linia);
00050
00051             getline(ss, linia, ',');
00052             w11.id = stoi(linia);
00053
00054             getline(ss >> ws, w11.przewinienie, ',');
00055
00056             getline(ss, linia, ',');
00057             if (linia.find("lat") != string::npos) {
00058                 w11.jednostka = "lat";
00059                 linia.erase(linia.find("lat"), 3);
00060                 w11.dlugosc = stoi(linia);
00061             }
00062             else if (linia.find("lata") != string::npos) {
00063                 w11.jednostka = "lata";
00064                 linia.erase(linia.find("lata"), 4);
00065                 w11.dlugosc = stoi(linia);
00066             }
00067             else if (linia.find("rok") != string::npos) {
00068                 w11.jednostka = "rok";
00069                 linia.erase(linia.find("rok"), 3);
00070                 w11.dlugosc = stoi(linia);
00071             }
00072             else if (linia.find("miesiecy") != string::npos) {
00073                 w11.jednostka = "miesiecy";
```

```

00074         linia.erase(linia.find("miesiacy"), 8);
00075         w11.dlugosc = stoi(linia);
00076     }
00077     else if (linia.find("miesiace") != string::npos) {
00078         w11.jednostka = "miesiace";
00079         linia.erase(linia.find("miesiace"), 8);
00080         w11.dlugosc = stoi(linia);
00081     }
00082     else if (linia.find("miesiac") != string::npos) {
00083         w11.jednostka = "miesiac";
00084         linia.erase(linia.find("miesiac"), 8);
00085         w11.dlugosc = stoi(linia);
00086     }
00087     else {
00088         w11.jednostka = "czasu";
00089         w11.dlugosc = stoi(linia);
00090     }
00091
00092     getline(ss >> ws, w11.cela, ',');
00093
00094     lista.push back(w11);
00095 }
00096 plik.close();
00097 }
00098 }
00099
00100
00101
00102 void Interfejs::menu() {
00103     system("CLS");
00104     cout << "Rejestr wiezienny" << endl;
00105     cout << "1. Push" << endl;
00106     cout << "2. Pop" << endl;
00107     cout << "3. Remove" << endl;
00108     cout << "4. Get element" << endl;
00109     cout << "5. Get index" << endl;
00110     cout << "6. Sort" << endl;
00111     cout << "7. Count" << endl;
00112     cout << "8. Arrest length" << endl;
00113     cout << "9. Avarage arrest length" << endl;
00114     cout << "10. Avarage length greater than" << endl;
00115     cout << "11. Prison block check" << endl;
00116     cout << "12. Birth date check" << endl;
00117     cout << "13. Lists connections" << endl;
00118     cout << "14. Display" << endl;
00119     cout << "15. Clear" << endl;
00120     cout << "16. Save" << endl;
00121     cout << "0. Exit" << endl;
00122     cout << "Wybierz opcje: " << endl;
00123 }
00124
00125 int Interfejs::wybor() {
00126     int choice;
00127     system("CLS");
00128     cout << "Z ktorej listy chcesz go wyswietlic? " << endl;
00129     cout << "1. Tabela z danymi osobowymi " << endl;
00130     cout << "2. Tabela z danymi wieziennymi " << endl;
00131     cout << "3. Tabela z tymczasowymi danymi osobowymi " << endl;
00132     cout << "4. Tabela z tymczasowymi danymi wieziennymi " << endl;
00133     cin >> choice;
00134     return choice;
00135 }
00136
00137 void Interfejs::blad() {
00138     system("CLS");
00139     cout << "Blad! Niepoprawna opcja! " << endl;
00140     Sleep(1000);
00141 }
00142
00143 void Interfejs::program(List<Tab1::Dane>& lista1, List<Tab2::Wiezienie>& lista2,
00144 const string& o) {
00145     List<Tab1::Dane> lista1_tmp;
00146     List<Tab2::Wiezienie> lista2_tmp;
00147     Tab1::Dane obj1;
00148     Tab2::Wiezienie obj2;
00149     string element;
00150     int choice, index;

```

```

00150     bool quit = false;
00151     char blok;
00152     int tab[6];
00153
00154     do {
00155         menu();
00156         cin >> choice;
00157         switch (choice) {
00158             case 1:
00159                 choice = wybor();
00160                 system("CLS");
00161                 cout << "Wybierz opcje: " << endl;
00162                 cout << "1. Push front " << endl;
00163                 cout << "2. Push back " << endl;
00164                 if (choice == 1) {
00165                     cin >> choice;
00166                     system("CLS");
00167                     cout << "Wpisz element: " << endl;
00168                     cin >> obj1;
00169                     if (choice == 1) {
00170                         lista1.push front(obj1);
00171                         system("CLS");
00172                         cout << "Element zostal dodany na poczatek. " << endl;
00173                         Sleep(1000);
00174                     }
00175                     else if (choice == 2) {
00176                         lista1.push back(obj1);
00177                         system("CLS");
00178                         cout << "Element zostal dodany na koniec. " << endl;
00179                         Sleep(1000);
00180                     }
00181                     else
00182                         blad();
00183                 }
00184                 else if (choice == 2) {
00185                     cin >> choice;
00186                     system("CLS");
00187                     cout << "Wpisz element: " << endl;
00188                     cin >> obj2;
00189                     if (choice == 1) {
00190                         lista2.push front(obj2);
00191                         system("CLS");
00192                         cout << "Element zostal dodany na poczatek. " << endl;
00193                         Sleep(1000);
00194                     }
00195                     else if (choice == 2) {
00196                         lista2.push back(obj2);
00197                         system("CLS");
00198                         cout << "Element zostal dodany na koniec. " << endl;
00199                         Sleep(1000);
00200                     }
00201                     else
00202                         blad();
00203                 }
00204                 else if (choice == 3) {
00205                     cin >> choice;
00206                     system("CLS");
00207                     cout << "Wpisz element: " << endl;
00208                     cin >> obj1;
00209                     if (choice == 1) {
00210                         lista1 tmp.push front(obj1);
00211                         system("CLS");
00212                         cout << "Element zostal dodany na poczatek. " << endl;
00213                         Sleep(1000);
00214                     }
00215                     else if (choice == 2) {
00216                         lista1 tmp.push back(obj1);
00217                         system("CLS");
00218                         cout << "Element zostal dodany na koniec. " << endl;
00219                         Sleep(1000);
00220                     }
00221                     else
00222                         blad();
00223                 }
00224                 else if (choice == 4) {
00225                     cin >> choice;
00226                     system("CLS");

```

```

00227         cout << "Wpisz element: " << endl;
00228         cin >> obj2;
00229         if (choice == 1) {
00230             lista2 tmp.push front(obj2);
00231             system("CLS");
00232             cout << "Element zostal dodany na poczatek. " << endl;
00233             Sleep(1000);
00234         }
00235         else if (choice == 2) {
00236             lista2 tmp.push back(obj2);
00237             system("CLS");
00238             cout << "Element zostal dodany na koniec. " << endl;
00239             Sleep(1000);
00240         }
00241         else
00242             blad();
00243     }
00244     else
00245         blad();
00246     break;
00247 case 2:
00248     choice = wybor();
00249     system("CLS");
00250     cout << "Wybierz opcje: " << endl;
00251     cout << "1. Pop front " << endl;
00252     cout << "2. Pop back " << endl;
00253     cout << "3. Pop (index) " << endl;
00254     if (choice == 1) {
00255         cin >> choice;
00256         if (choice == 1) {
00257             lista1.pop front();
00258             system("CLS");
00259             cout << "Element z poczatku zostal usuniety. " << endl;
00260             Sleep(1000);
00261         }
00262         else if (choice == 2) {
00263             lista1.pop back();
00264             system("CLS");
00265             cout << "Element z konca zostal usuniety. " << endl;
00266             Sleep(1000);
00267         }
00268         else if (choice == 3) {
00269             system("CLS");
00270             cout << "Wpisz index elementu, ktory chcesz usunac: " << endl;
00271             cin >> index;
00272             lista1.pop(index);
00273             system("CLS");
00274             cout << "Element o indexie " << index << " zostal usuniety. "
<< endl;
00275             Sleep(1000);
00276         }
00277         else
00278             blad();
00279     }
00280     else if (choice == 2) {
00281         cin >> choice;
00282         if (choice == 1) {
00283             lista2.pop front();
00284             system("CLS");
00285             cout << "Element z poczatku zostal usuniety. " << endl;
00286             Sleep(1000);
00287         }
00288         else if (choice == 2) {
00289             lista2.pop_back();
00290             system("CLS");
00291             cout << "Element z konca zostal usuniety. " << endl;
00292             Sleep(1000);
00293         }
00294         else if (choice == 3) {
00295             system("CLS");
00296             cout << "Wpisz index elementu, ktory chcesz usunac: " << endl;
00297             cin >> index;
00298             lista2.pop(index);
00299             system("CLS");
00300             cout << "Element o id " << index << " zostal usuniety. " << endl;
00301             Sleep(1000);
00302         }

```

```

00303         else
00304             blad();
00305     }
00306     else if (choice == 3) {
00307         cin >> choice;
00308         if (choice == 1) {
00309             lista1 tmp.pop front();
00310             system("CLS");
00311             cout << "Element z poczatku zostal usuniety. " << endl;
00312             Sleep(1000);
00313         }
00314         else if (choice == 2) {
00315             lista1 tmp.pop back();
00316             system("CLS");
00317             cout << "Element z konca zostal usuniety. " << endl;
00318             Sleep(1000);
00319         }
00320         else if (choice == 3) {
00321             system("CLS");
00322             cout << "Wpisz index elementu, który chcesz usunac: " << endl;
00323             cin >> index;
00324             lista1 tmp.pop(index);
00325             system("CLS");
00326             cout << "Element o id " << index << " zostal usuniety. " << endl;
00327             Sleep(1000);
00328         }
00329         else
00330             blad();
00331     }
00332     else if (choice == 4) {
00333         cin >> choice;
00334         if (choice == 1) {
00335             lista2 tmp.pop front();
00336             system("CLS");
00337             cout << "Element z poczatku zostal usuniety. " << endl;
00338             Sleep(1000);
00339         }
00340         else if (choice == 2) {
00341             lista2 tmp.pop_back();
00342             system("CLS");
00343             cout << "Element z konca zostal usuniety. " << endl;
00344             Sleep(1000);
00345         }
00346         else if (choice == 3) {
00347             system("CLS");
00348             cout << "Wpisz index elementu, który chcesz usunac: " << endl;
00349             cin >> index;
00350             lista2 tmp.pop(index);
00351             system("CLS");
00352             cout << "Element o id " << index << " zostal usuniety. " << endl;
00353             Sleep(1000);
00354         }
00355         else
00356             blad();
00357     }
00358     else
00359         blad();
00360     break;
00361 case 3:
00362     choice = wybor();
00363     system("CLS");
00364     cout << "Wpisz elementy, ktore chcesz usunac z listy: " << endl;
00365     if (choice == 1) {
00366         cin >> obj1;
00367         lista1.remove(obj1);
00368         system("CLS");
00369         cout << "Elementy zostaly usuniete. " << endl;
00370         Sleep(1000);
00371     }
00372     else if (choice == 2) {
00373         cin >> obj2;
00374         lista2.remove(obj2);
00375         system("CLS");
00376         cout << "Elementy zostaly usuniete. " << endl;
00377         Sleep(1000);
00378     }
00379     else if (choice == 3) {

```

```

00380         cin >> obj1;
00381         lista1 tmp.remove(obj1);
00382         system("CLS");
00383         cout << "Elementy zostaly usuniete. " << endl;
00384         Sleep(1000);
00385     }
00386     else if (choice == 4) {
00387         cin >> obj2;
00388         lista2 tmp.remove(obj2);
00389         system("CLS");
00390         cout << "Elementy zostaly usuniete. " << endl;
00391         Sleep(1000);
00392     }
00393     else
00394         blad();
00395     break;
00396 case 4:
00397     choice = wybor();
00398     system("CLS");
00399     cout << "Po czym chcesz wyszukac element? " << endl;
00400     cout << "1. Po indexie " << endl;
00401     cout << "2. Po id " << endl;
00402     if (choice == 1) {
00403         cin >> choice;
00404         system("CLS");
00405         if (choice == 1) {
00406             cout << "Wpisz index szukanego elementu: " << endl;
00407             cin >> index;
00408             lista1_tmp.clear();
00409             lista1 tmp.push back(lista1.get element(index));
00410             system("CLS");
00411             cout << "Element zostal zapisany w pamieci (w liscie
tymczasowej). " << endl;
00412             Sleep(1000);
00413         }
00414         else if (choice == 1) {
00415             cout << "Wpisz id szukanego elementu: " << endl;
00416             cin >> index;
00417             lista1_tmp.clear();
00418             lista1_tmp.push_back(lista1.get_element2(index));
00419             system("CLS");
00420             cout << "Element zostal zapisany w pamieci (w liscie
tymczasowej). " << endl;
00421             Sleep(1000);
00422         }
00423         else
00424             blad();
00425     }
00426     else if (choice == 2) {
00427         cin >> choice;
00428         system("CLS");
00429         if (choice == 1) {
00430             cout << "Wpisz index szukanego elementu: " << endl;
00431             cin >> index;
00432             lista2 tmp.clear();
00433             lista2 tmp.push_back(lista2.get_element(index));
00434             system("CLS");
00435             cout << "Element zostal zapisany w pamieci (w liscie
tymczasowej). " << endl;
00436             Sleep(1000);
00437         }
00438         else if (choice == 1) {
00439             cout << "Wpisz id szukanego elementu: " << endl;
00440             cin >> index;
00441             lista2 tmp.clear();
00442             lista2 tmp.push_back(lista2.get element2(index));
00443             system("CLS");
00444             cout << "Element zostal zapisany w pamieci (w liscie
tymczasowej). " << endl;
00445             Sleep(1000);
00446         }
00447         else
00448             blad();
00449     }
00450     else if (choice == 3) {
00451         cin >> choice;
00452         system("CLS");

```

```

00453         if (choice == 1) {
00454             cout << "Wpisz index szukanego elementu: " << endl;
00455             cin >> index;
00456             Tab1::Dane tmp1 = lista1 tmp.get element(index);
00457             lista1_tmp.clear();
00458             lista1_tmp.push_back(tmp1);
00459             system("CLS");
00460             cout << "Element zostal zapisany w pamieci (w liscie
tymczasowej). " << endl;
00461             Sleep(1000);
00462         }
00463         else if (choice == 1) {
00464             cout << "Wpisz id szukanego elementu: " << endl;
00465             cin >> index;
00466             Tab1::Dane tmp1 = lista1 tmp.get element(index);
00467             lista1_tmp.clear();
00468             lista1_tmp.push_back(tmp1);
00469             system("CLS");
00470             cout << "Element zostal zapisany w pamieci (w liscie
tymczasowej). " << endl;
00471             Sleep(1000);
00472         }
00473         else
00474             blad();
00475     }
00476     else if (choice == 4) {
00477         cin >> choice;
00478         system("CLS");
00479         if (choice == 1) {
00480             cout << "Wpisz index szukanego elementu: " << endl;
00481             cin >> index;
00482             Tab2::Wiezienie tmp2 = lista2 tmp.get element(index);
00483             lista2_tmp.clear();
00484             lista2_tmp.push_back(tmp2);
00485             system("CLS");
00486             cout << "Element zostal zapisany w pamieci (w liscie
tymczasowej). " << endl;
00487             Sleep(1000);
00488         }
00489         else if (choice == 1) {
00490             cout << "Wpisz id szukanego elementu: " << endl;
00491             cin >> index;
00492             Tab2::Wiezienie tmp2 = lista2 tmp.get element(index);
00493             lista2_tmp.clear();
00494             lista2_tmp.push_back(tmp2);
00495             system("CLS");
00496             cout << "Element zostal zapisany w pamieci (w liscie
tymczasowej). " << endl;
00497             Sleep(1000);
00498         }
00499         else
00500             blad();
00501     }
00502     else
00503         blad();
00504     break;
00505 case 5:
00506     choice = wybor();
00507     system("CLS");
00508     cout << "Wpisz element z szukanym indexem: " << endl;
00509     if (choice == 1) {
00510         cin >> obj1;
00511         system("CLS");
00512         cout << "Index: " << lista1.get_index(obj1) << endl;
00513         Sleep(2000);
00514     }
00515     else if (choice == 2) {
00516         cin >> obj2;
00517         system("CLS");
00518         cout << "Index: " << lista2.get_index(obj2) << endl;
00519         Sleep(2000);
00520     }
00521     else if (choice == 3) {
00522         cin >> obj1;
00523         system("CLS");
00524         cout << "Index: " << lista1 tmp.get index(obj1) << endl;
00525         Sleep(2000);

```



```

00526     }
00527     else if (choice == 4) {
00528         cin >> obj2;
00529         system("CLS");
00530         cout << "Index: " << lista2_tmp.get_index(obj2) << endl;
00531         Sleep(2000);
00532     }
00533     else
00534         blad();
00535     break;
00536 case 6:
00537     choice = wybor();
00538     system("CLS");
00539     cout << "Jak chcesz posortowac dane?" << endl;
00540     cout << "1. Rosnaco" << endl;
00541     cout << "2. Malejaco" << endl;
00542     if (choice == 1) {
00543         cin >> choice;
00544         system("CLS");
00545         cout << "Wedlug czego chcesz posortowac? " << endl;
00546         cout << "1-id, 2-imie, 3-nazwisko, 4-data urodzenia, 5-plec " <<
endl;
00547         cin >> index;
00548         lista1.sort(choice, index);
00549         system("CLS");
00550         cout << "Lista zostala posortowana. " << endl;
00551         Sleep(1000);
00552     }
00553     else if (choice == 2) {
00554         cin >> choice;
00555         system("CLS");
00556         cout << "Wedlug czego chcesz posortowac? " << endl;
00557         cout << "1-id, 2-przewinienie, 3-dlugosc odsiadki, 4-cela " << endl;
00558         cin >> index;
00559         lista2.sort(choice, index);
00560         system("CLS");
00561         cout << "Lista zostala posortowana. " << endl;
00562         Sleep(1000);
00563     }
00564     else if (choice == 3) {
00565         cin >> choice;
00566         system("CLS");
00567         cout << "Wedlug czego chcesz posortowac? " << endl;
00568         cout << "1-id, 2-imie, 3-nazwisko, 4-data urodzenia, 5-plec " <<
endl;
00569         cin >> index;
00570         lista1_tmp.sort(choice, index);
00571         system("CLS");
00572         cout << "Lista zostala posortowana. " << endl;
00573         Sleep(1000);
00574     }
00575     else if (choice == 4) {
00576         cin >> choice;
00577         system("CLS");
00578         cout << "Wedlug czego chcesz posortowac? " << endl;
00579         cout << "1-id, 2-przewinienie, 3-dlugosc odsiadki, 4-cela " << endl;
00580         cin >> index;
00581         lista2_tmp.sort(choice, index);
00582         system("CLS");
00583         cout << "Lista zostala posortowana. " << endl;
00584         Sleep(1000);
00585     }
00586     else
00587         blad();
00588     break;
00589 case 7:
00590     choice = wybor();
00591     system("CLS");
00592     cout << "Ktory typ elementu chcesz zliczyc? " << endl;
00593     if (choice == 1) {
00594         cout << "1-id, 2-imie, 3-nazwisko, 4-dzien, 5-miesiac, 6-rok, 7-plec
" << endl;
00595         cin >> choice;
00596         system("CLS");
00597         cout << "Wpisz ten element: " << endl;
00598         cin >> element;
00599         system("CLS");

```

```

00600         cout << element << " wystepuje w liscie " <<
lista1.element count1(choice, element) << " razy. " << endl;
00601         Sleep(2000);
00602     }
00603     else if (choice == 2) {
00604         cout << "1-id, 2-przewinienie, 3-dlugosc, 4-jednostka, 5-cela " <<
endl;
00605         cin >> choice;
00606         system("CLS");
00607         cout << "Wpisz ten element: " << endl;
00608         cin >> element;
00609         system("CLS");
00610         cout << element << " wystepuje w liscie " <<
lista2.element count2(choice, element) << " razy. " << endl;
00611         Sleep(2000);
00612     }
00613     else if (choice == 3) {
00614         cout << "1-id, 2-imie, 3-nazwisko, 4-dzien, 5-miesiac, 6-rok, 7-plec
" << endl;
00615         cin >> choice;
00616         system("CLS");
00617         cout << "Wpisz ten element: " << endl;
00618         cin >> element;
00619         system("CLS");
00620         cout << element << " wystepuje w liscie " <<
lista1.tmp.element count1(choice, element) << " razy. " << endl;
00621         Sleep(2000);
00622     }
00623     else if (choice == 4) {
00624         cout << "1-id, 2-przewinienie, 3-dlugosc, 4-jednostka, 5-cela " <<
endl;
00625         cin >> choice;
00626         system("CLS");
00627         cout << "Wpisz ten element: " << endl;
00628         cin >> element;
00629         system("CLS");
00630         cout << element << " wystepuje w liscie " <<
lista2.tmp.element count2(choice, element) << " razy. " << endl;
00631         Sleep(2000);
00632     }
00633     else
00634         blad();
00635     break;
00636 case 8:
00637     system("CLS");
00638     cout << "Z ktorej listy chcesz go wyswietlic? " << endl;
00639     cout << "1. Tabela z danymi wieziennymi " << endl;
00640     cout << "2. Tabela z tymczasowymi danymi wieziennymi" << endl;
00641     cin >> choice;
00642     system("CLS");
00643     cout << "Podaj id elementu: " << endl;
00644     cin >> index;
00645     system("CLS");
00646     cout << "Element o id: " << index << endl;
00647     cout << "Kara wiezienia: ";
00648     if (choice == 1) {
00649         if (lista2.arrest_length(index) > 12) {
00650             double tmp = lista2.arrest_length(index) / 12.0;
00651             cout << setprecision(3) << tmp << " (lata) " << endl;
00652         }
00653         else
00654             cout << lista2.arrest_length(index) << " (miesiace) " << endl;
00655         Sleep(3000);
00656     }
00657     else if (choice == 2) {
00658         if (lista2.tmp.arrest_length(index) > 12) {
00659             double tmp = lista2.tmp.arrest_length(index) / 12.0;
00660             cout << setprecision(3) << tmp << " (lata) " << endl;
00661         }
00662         else
00663             cout << lista2.tmp.arrest_length(index) << " (miesiace) " <<
endl;
00664         Sleep(3000);
00665     }
00666     else
00667         blad();
00668     break;

```

```

00669         case 9:
00670             system("CLS");
00671             cout << "Z ktorej listy chcesz go wyswietlic? " << endl;
00672             cout << "1. Tabela z danymi wieziennymi " << endl;
00673             cout << "2. Tabela z tymczasowymi danymi wieziennymi" << endl;
00674             cin >> choice;
00675             system("CLS");
00676             cout << "Podaj przewinienie: (all - jezeli chcesz srednia wszystkich) "
<< endl;
00677             cin >> element;
00678             system("CLS");
00679             cout << "Przewinienie: " << element << endl;
00680             cout << "Srednia kara wiezienia: ";
00681             if (choice == 1) {
00682                 if (lista2.arrest average(element) > 12) {
00683                     double tmp = lista2.arrest average(element) / 12.0;
00684                     cout << setprecision(3) << tmp << " (lata) " << endl;
00685                 }
00686                 else
00687                     cout << lista2.arrest average(element) << " (miesiace) " <<
endl;
00688                 Sleep(3000);
00689             }
00690             else if (choice == 2) {
00691                 if (lista2 tmp.arrest average(element) > 12) {
00692                     double tmp = lista2 tmp.arrest average(element) / 12.0;
00693                     cout << setprecision(3) << tmp << " (lata) " << endl;
00694                 }
00695                 else
00696                     cout << lista2 tmp.arrest average(element) << " (miesiace) " <<
endl;
00697                 Sleep(3000);
00698             }
00699             else
00700                 blad();
00701             break;
00702         case 10:
00703             system("CLS");
00704             cout << "Z ktorej listy chcesz go wyswietlic? " << endl;
00705             cout << "1. Tabela z danymi wieziennymi " << endl;
00706             cout << "2. Tabela z tymczasowymi danymi wieziennymi" << endl;
00707             cin >> choice;
00708             system("CLS");
00709             cout << "Podaj przewinienie: (all - jezeli chcesz srednia wszystkich) "
<< endl;
00710             cin >> element;
00711             cout << "oraz srednia dlugosc w miesiacach do porownania: " << endl;
00712             cin >> index;
00713             if (choice == 1) {
00714                 lista2_tmp = lista2.avarage_greater(index, element);
00715                 system("CLS");
00716                 cout << "Lista z dluzszymi karami zostala zapisana w pamieci. " <<
endl;
00717                 Sleep(2000);
00718             }
00719             else if (choice == 2) {
00720                 lista2_tmp = lista2_tmp.avarage_greater(index, element);
00721                 system("CLS");
00722                 cout << "Lista z dluzszymi karami zostala zapisana w pamieci. " <<
endl;
00723                 Sleep(2000);
00724             }
00725             else
00726                 blad();
00727             break;
00728         case 11:
00729             system("CLS");
00730             cout << "Z ktorej listy chcesz go wyswietlic? " << endl;
00731             cout << "1. Tabela z danymi wieziennymi " << endl;
00732             cout << "2. Tabela z tymczasowymi danymi wieziennymi" << endl;
00733             cin >> choice;
00734             system("CLS");
00735             cout << "Podaj blok do sprawdzenia: " << endl;
00736             cin >> blok;
00737             if (choice == 1) {
00738                 lista2 tmp = lista2.block check(blok);
00739                 system("CLS");

```

```

00740         cout << "Lista z blokami zostala zapisana w pamieci. " << endl;
00741         Sleep(1000);
00742     }
00743     else if (choice == 2) {
00744         lista2_tmp = lista2_tmp.block_check(blok);
00745         system("CLS");
00746         cout << "Lista z blokami zostala zapisana w pamieci. " << endl;
00747         Sleep(1000);
00748     }
00749     else
00750         blad();
00751     break;
00752 case 12:
00753     system("CLS");
00754     cout << "Z ktorej listy chcesz go wyswietlic? " << endl;
00755     cout << "1. Tabela z danymi osobowymi " << endl;
00756     cout << "2. Tabela z tymczasowymi danymi osobowymi" << endl;
00757     cin >> choice;
00758     system("CLS");
00759     if (choice == 1) {
00760         cout << "Podaj dolna granice daty: " << endl;
00761         cin >> tab[0] >> tab[1] >> tab[2];
00762         cout << "Teraz podaj gorna granice daty: " << endl;
00763         cin >> tab[3] >> tab[4] >> tab[5];
00764         lista1 tmp = lista1.born during(tab[0], tab[1], tab[2], tab[3],
00765         tab[4], tab[5]);
00766         system("CLS");
00767         cout << "Lista z datami zostala zapisana w pamieci. " << endl;
00768         Sleep(1000);
00769     }
00770     else if (choice == 2) {
00771         cout << "Podaj dolna granice daty: " << endl;
00772         cin >> tab[0] >> tab[1] >> tab[2];
00773         cout << "Teraz podaj gorna granice daty: " << endl;
00774         cin >> tab[3] >> tab[4] >> tab[5];
00775         lista1 tmp = lista1 tmp.born during(tab[0], tab[1], tab[2],
00776         tab[3], tab[4], tab[5]);
00777         system("CLS");
00778         cout << "Lista z datami zostala zapisana w pamieci. " << endl;
00779         Sleep(1000);
00780     }
00781     else
00782         blad();
00783     break;
00784 case 13:
00785     system("CLS");
00786     cout << "Jakie powiazania chcesz znalezc? " << endl;
00787     cout << "1. Tymczasowa lista danych wieziennych z danymi osobowymi " <<
00788     endl;
00789     cout << "2. Tymczasowa lista danych osobowych z danymi wieziennymi " <<
00790     endl;
00791     cin >> choice;
00792     if (choice == 1) {
00793         lista1 tmp = lista2 tmp.connections<Tab2::Wiezienie,
00794         Tab1::Dane>(lista1);
00795         system("CLS");
00796         cout << "Lista z powiazaniami zostala zapisana w pamieci. " << endl;
00797         Sleep(1000);
00798     }
00799     else if (choice == 2) {
00800         lista2 tmp = lista1 tmp.connections<Tab1::Dane,
00801         Tab2::Wiezienie>(lista2);
00802         system("CLS");
00803         cout << "Lista z powiazaniami zostala zapisana w pamieci. " << endl;
00804         Sleep(1000);
00805     }
00806     else
00807         blad();
00808     break;
00809 case 14:
00810     system("CLS");
00811     cout << "Chcesz wyswietlic wybrany element czy cala liste? " << endl;
00812     cout << "1. Jeden element" << endl;
00813     cout << "2. Cala lista" << endl;
00814     cin >> choice;
00815     if (choice == 1) {
00816         system("CLS");

```

```

00811         cout << "Podaj index elementu: " << endl;
00812         cin >> index;
00813         choice = wybor();
00814         system("CLS");
00815         if (choice == 1)
00816             lista1.displayElement(index);
00817         else if (choice == 2)
00818             lista2.displayElement(index);
00819         else if (choice == 3)
00820             lista1_tmp.displayElement(index);
00821         else if (choice == 4)
00822             lista2_tmp.displayElement(index);
00823         else
00824             blad();
00825         Sleep(3000);
00826     }
00827     else if (choice == 2) {
00828         system("CLS");
00829         cout << "Z ktorej listy chcesz go wyswietlic? " << endl;
00830         cout << "1. Tabela z danymi osobowymi " << endl;
00831         cout << "2. Tabela z danymi wieziennymi " << endl;
00832         cout << "3. Tabela z tymczasowymi danymi osobowymi " << endl;
00833         cout << "4. Tabela z tymczasowymi danymi wieziennymi" << endl;
00834         cout << "5. Wszystkie " << endl;
00835         cin >> choice;
00836         system("CLS");
00837         if (choice == 1)
00838             lista1.displayList();
00839         else if (choice == 2)
00840             lista2.displayList();
00841         else if (choice == 3)
00842             lista1_tmp.displayList();
00843         else if (choice == 4)
00844             lista2_tmp.displayList();
00845         else if (choice == 5) {
00846             cout << "1. Tabela z danymi osobowymi: " << endl;
00847             lista1.displayList();
00848             cout << endl << "2. Tabela z danymi wieziennymi: " << endl;
00849             lista2.displayList();
00850             cout << endl << "3. Tabela z tymczasowymi danymi osobowymi: "
00851             << endl;
00852             lista1_tmp.displayList();
00853             cout << endl << "4. Tabela z tymczasowymi danymi wieziennymi: "
00854             << endl;
00855             lista2_tmp.displayList();
00856             Sleep(7000);
00857         }
00858         else
00859             blad();
00860         Sleep(3000);
00861     }
00862     else
00863         blad();
00864     break;
00865 case 15:
00866     choice = wybor();
00867     if (choice == 1) {
00868         lista1.clear();
00869         system("CLS");
00870         cout << "Lista zostala wyczyszczona. " << endl;
00871         Sleep(1000);
00872     }
00873     else if (choice == 2) {
00874         lista2.clear();
00875         system("CLS");
00876         cout << "Lista zostala wyczyszczona. " << endl;
00877         Sleep(1000);
00878     }
00879     else if (choice == 3) {
00880         lista1_tmp.clear();
00881         system("CLS");
00882         cout << "Lista zostala wyczyszczona. " << endl;
00883         Sleep(1000);
00884     }
00885     else if (choice == 4) {
00886         lista2_tmp.clear();
00887         system("CLS");

```

```

00886         cout << "Lista zostala wyczyszczona. " << endl;
00887         Sleep(1000);
00888     }
00889     else
00890         blad();
00891     break;
00892 case 16:
00893     system("CLS");
00894     cout << "Chcesz zapisywac ciagiem (log), jest to zapis pojedynczy czy
czyszczenie pliku? " << endl;
00895     cout << "1. Ciagiem" << endl;
00896     cout << "2. Pojedynczy" << endl;
00897     cout << "3. Czyszczenie pliku" << endl;
00898     cin >> choice;
00899     system("CLS");
00900     cout << "Ktora liste chcesz zapisac? " << endl;
00901     cout << "1. Tabela z danymi osobowymi " << endl;
00902     cout << "2. Tabela z danymi wieziennymi " << endl;
00903     cout << "3. Tabela z tymczasowymi danymi osobowymi " << endl;
00904     cout << "4. Tabela z tymczasowymi danymi wieziennymi" << endl;
00905     if (choice == 1) {
00906         cout << "5. Wszystkie" << endl;
00907         cin >> choice;
00908         if (choice == 1)
00909             lista1.zapis2(o);
00910         else if (choice == 2)
00911             lista2.zapis2(o);
00912         else if (choice == 3)
00913             lista1_tmp.zapis2(o);
00914         else if (choice == 4)
00915             lista2_tmp.zapis2(o);
00916         else if (choice == 5) {
00917             lista1.zapis2(o);
00918             lista2.zapis2(o);
00919             lista1_tmp.zapis2(o);
00920             lista2_tmp.zapis2(o);
00921         }
00922         system("CLS");
00923         cout << "Dane zostaly zapisane. " << endl;
00924         Sleep(1000);
00925     }
00926     else if (choice == 2) {
00927         cin >> choice;
00928         if (choice == 1)
00929             lista1.zapis(o);
00930         else if (choice == 2)
00931             lista2.zapis(o);
00932         else if (choice == 3)
00933             lista1_tmp.zapis(o);
00934         else if (choice == 4)
00935             lista2_tmp.zapis(o);
00936         else
00937             blad();
00938         system("CLS");
00939         cout << "Dane zostaly zapisane. " << endl;
00940         Sleep(1000);
00941     }
00942     else if (choice == 3) {
00943         zapis_clear(o);
00944         system("CLS");
00945         cout << "Plik zostal wyczyszczony. " << endl;
00946         Sleep(1000);
00947     }
00948     else
00949         blad();
00950     break;
00951 case 0:
00952     quit = true;
00953     break;
00954 default:
00955     break;
00956 }
00957 } while (!quit);
00958 system("CLS");
00959 }
00960
00961 void Interfejs::zapis_clear(const string& nazwa) {

```

```

00962     ofstream clear(nazwa);
00963     if (clear) {
00964         clear << "" << endl;
00965         clear.close();
00966     }
00967 }
00968
00969
00970
00971 void parameters check(int argc, char* argv[], string& i1, string& i2, string& o) {
00972     if (argc != 7) {
00973         cout << "Blad! Bledne parametry. (i1 i2 o)" << endl;
00974         exit(0);
00975     }
00976
00977     for (int i = 1; i < argc; ++i) {
00978         string arg = argv[i];
00979         if (arg == "-i1")
00980             i1 = argv[++i];
00981         else if (arg == "-i2")
00982             i2 = argv[++i];
00983         else if (arg == "-o")
00984             o = argv[++i];
00985         else {
00986             cout << "Blad! Nieznany parametr: " << arg << endl;
00987             exit(0);
00988         }
00989     }
00990     cout << "Dane z plikow zostaly wczytane. " << endl;
00991     Sleep(1000);
00992 }

```

## classes.h File Reference

```
#include <iostream>
```

### Classes

- class **Tab1**
- struct **Tab1::Dane**
- class **Tab2**
- struct **Tab2::Wiezienie**
- class **Interfejs**

### Functions

- void **parameters\_check** (int, char \*[], string &, string &, string &)

---

### Function Documentation

**void parameters\_check** (int *argc*, char \* *argv*[], string & *i1*, string & *i2*, string & *o*)

Funkcja odpowiedzialna za sprawdzenie poprawnosci argumentow podanych w konsoli. Przypisuje odpowiednie nazwy plikow do danych wejsciowych/wyjsciowych.

Definition at line **971** of file **classes.cpp**.



## classes.h

```
Go to the documentation of this file.00001 #pragma once
00002 #include<iostream>
00003
00004 using namespace std;
00005
00010 class Tab1 {
00011     template<typename U> friend class List;
00012 public:
00017     struct Dane {
00018         int id;
00019         string imie;
00020         string nazwisko;
00021         int dzien;
00022         int miesiac;
00023         int rok;
00024         char plec;
00025
00028         Dane() : id(0), imie(" "), nazwisko(" "), dzien(0), miesiac(0), rok(0),
plec(' ') {}
00031         Dane(int id, string imie, string nazwisko, int dzien, int miesiac, int rok,
char plec) : id(id), imie(imie), nazwisko(nazwisko), dzien(dzien), miesiac(miesiac),
rok(rok), plec(plec) {}
00032     };
00033
00036     void odczyt(const string&, List<Tab1::Dane>&);
00039     friend bool operator==(const Tab1::Dane& dane1, const Tab1::Dane& dane2) {
00040         return dane1.id == dane2.id && dane1.imie == dane2.imie && dane1.nazwisko
== dane2.nazwisko && dane1.dzien == dane2.dzien && dane1.miesiac == dane2.miesiac &&
dane1.rok == dane2.rok && dane1.plec == dane2.plec;
00041     }
00044     friend bool operator<(const Tab1::Dane& dane1, const Tab1::Dane& dane2) {
00045         if (dane1.id != dane2.id)
00046             return dane1.id < dane2.id;
00047         else if (dane1.imie != dane2.imie)
00048             return dane1.imie < dane2.imie;
00049         else if (dane1.nazwisko != dane2.nazwisko)
00050             return dane1.nazwisko < dane2.nazwisko;
00051         else if (dane1.dzien != dane2.dzien)
00052             return dane1.dzien < dane2.dzien;
00053         else if (dane1.miesiac != dane2.miesiac)
00054             return dane1.miesiac < dane2.miesiac;
00055         else if (dane1.rok != dane2.rok)
00056             return dane1.rok < dane2.rok;
00057         else if (dane1.plec != dane2.plec)
00058             return dane1.plec < dane2.plec;
00059         return false;
00060     }
00063     friend istream& operator>>(istream& in, Tab1::Dane& dane) {
00064         in >> dane.id >> dane.imie >> dane.nazwisko >> dane.dzien >> dane.miesiac
>> dane.rok >> dane.plec;
00065         return in;
00066     }
00069     ~Tab1() {}
00070 };
00071
00072
00073
00078 class Tab2 {
00079     template<typename U> friend class List;
00080 public:
00085     struct Wiezienie {
00086         int id;
00087         string przewinienie;
00088         int dlugosc;
00089         string jednostka;
00090         string cela;
00091
00094         Wiezienie() : id(0), przewinienie(" "), dlugosc(0), jednostka(" "), cela("
") {}
00097         Wiezienie(int id, string przewinienie, int dlugosc, string jednostka, string
cela) : id(id), przewinienie(przewinienie), dlugosc(dlugosc), jednostka(jednostka),
cela(cela) {}
00098     };
00099 }
```

```

00099
00102     void odczyt(const string&, List<Tab2::Wiezienie>&);
00105     friend bool operator==(const Tab2::Wiezienie& w11 1, const Tab2::Wiezienie&
w11 2) {
00106         return w11_1.id == w11_2.id && w11_1.przewinienie == w11_2.przewinienie &&
w11_1.dlugosc == w11_2.dlugosc && w11_1.jednostka == w11_2.jednostka && w11_1.cela ==
w11_2.cela;
00107     }
00110     friend bool operator<(const Tab2::Wiezienie& w11 1, const Tab2::Wiezienie&
w11 2) {
00111         if (w11_1.id != w11_2.id)
00112             return w11_1.id < w11_2.id;
00113         else if (w11_1.przewinienie != w11_2.przewinienie)
00114             return w11_1.przewinienie < w11_2.przewinienie;
00115         else if (w11_1.dlugosc != w11_2.dlugosc)
00116             return w11_1.dlugosc < w11_2.dlugosc;
00117         else if (w11_1.jednostka != w11_2.jednostka)
00118             return w11_1.jednostka < w11_2.jednostka;
00119         else if (w11_1.cela != w11_2.cela)
00120             return w11_1.cela < w11_2.cela;
00121         return false;
00122     }
00125     friend istream& operator>>(istream& in, Tab2::Wiezienie& w11) {
00126         in >> w11.id >> w11.przewinienie >> w11.dlugosc >> w11.jednostka >> w11.cela;
00127         return in;
00128     }
00131     ~Tab2() {}
00132 };
00133
00134
00135
00139 class Interfejs {
00140 public:
00143     void menu();
00146     int wybor();
00149     void blad();
00153     void program(List<Tab1::Dane>&, List<Tab2::Wiezienie>&, const string&);
00156     void zapis clear(const string&);
00157 };
00158
00159
00163 void parameters_check(int, char* [], string&, string&, string&);

```

## Rejestr\_wiezienny.cpp File Reference

```
#include <iostream>
#include "classes.h"
#include "template.h"
```

### Functions

- `int main (int argc, char *argv[])`
- 

### Function Documentation

`int main (int argc, char * argv[])`

Definition at line 7 of file `Rejestr_wiezienny.cpp`.

## Rejestr\_wiezienny.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include "classes.h"
00003 #include "template.h"
00004
00005 using namespace std;
00006
00007 int main(int argc, char* argv[])
00008 {
00009     string i1="Dane1.txt", i2="Dane2.txt", o="Dane3.txt";
00010     //string i1, i2, o;
00011     //parameters_check(argc, argv, i1, i2, o);
00012
00013     List<Tab1::Dane> lista1;
00014     List<Tab2::Wiezienie> lista2;
00015     Tab1 tab1;
00016     Tab2 tab2;
00017
00018     tab1.odczyt(i1, lista1);
00019     tab2.odczyt(i2, lista2);
00020
00021     Interfejs interfejs;
00022     interfejs.program(lista1, lista2, o);
00023
00024     return 0;
00025 }
```

## template.h File Reference

```
#include <iostream>
#include <fstream>
#include <string>
```

### Classes

- class **Node**< T >
- class **List**< T >

## template.h

```
Go to the documentation of this file.00001 #pragma once
00002 #include<iostream>
00003 #include<fstream>
00004 #include<string>
00005
00006 using namespace std;
00007
00013 template<typename T>
00014 class Node {
00015     template<typename U> friend class List;
00016
00017     T data;
00018     shared_ptr<Node<T>> next;
00019
00020 public:
00023     Node(T obj) : data(obj), next(NULL) {}
00026     ~Node() {}
00027 };
00028
00036 template<typename T>
00037 class List {
00038     friend class Tab1;
00039     friend class Tab2;
00040     shared_ptr<Node<T>> head;
00041
00042 public:
00045     List() : head(NULL) {}
00048     List(const List& lista) : head(NULL) {
00049         shared_ptr<Node<T>> ptr = lista.head;
00050         while (ptr != NULL) {
00051             push_back(ptr->data);
00052             ptr = ptr->next;
00053         }
00054     }
00057     List(List&& lista) {
00058         head = NULL;
00059         shared_ptr<Node<T>> ptr = lista.head;
00060         while (ptr != NULL) {
00061             push_back(move(ptr->data));
00062             ptr = ptr->next;
00063         }
00064         lista.head = NULL;
00065     }
00068     const shared_ptr<Node<T>>& get head() const { return head; }
00071     void push_front(const T&);
00074     void push_back(const T&);
00077     void pop_front();
00080     void pop_back();
00083     void pop(const int);
00086     T get_element(const int);
00089     T get_element2(const int);
00092     int get_index(const T&);
00095     int get_index(const int);
00098     void remove(const T&);
00101     bool greater(const T&, const T&);
00104     bool smaller(const T&, const T&);
00107     bool greater(const Tab1::Dane&, const Tab1::Dane&, const int);
00110     bool smaller(const Tab1::Dane&, const Tab1::Dane&, const int);
00113     bool greater(const Tab2::Wiezienie&, const Tab2::Wiezienie&, const int);
00116     bool smaller(const Tab2::Wiezienie&, const Tab2::Wiezienie&, const int);
00119     void sort(const int);
00122     void sort(const int, const int);
00125     int element count(const string&);
00128     int element count1(const int, const string&);
00131     int element count2(const int, const string&);
00134     int arrest length(const int);
00137     double arrest average(const string&);
00140     List<T> avarage_greater(const double, const string&);
00143     List<T> block check(const char);
00146     List<T> born during(const int, const int, const int, const int, const int, const
int);
00149     void clear();
00152     int size();
```

```

00155     bool is empty();
00158     void serialization(const string&);
00161     void deserialization(const string&);
00164     void serialization_string(const string&);
00167     void deserialization_string(const string&);
00170     void displayElement(const int);
00173     void displayList();
00176     void zapis(const string&);
00179     void zapis2(const string&);
00182     template<typename T, typename U>
00183     List<U> connections(const List<U>& lista) {
00184         List<U> result;
00185         shared_ptr<Node<T>> ptr = head;
00186         while (ptr != NULL) {
00187             shared_ptr<Node<U>> ptr1 = lista.get head();
00188             while (ptr1 != NULL) {
00189                 if (ptr->data.id == ptr1->data.id)
00190                     result.push_back(ptr1->data);
00191                 ptr1 = ptr1->next;
00192             }
00193             ptr = ptr->next;
00194         }
00195         return result;
00196     }
00199     List& operator=(const List& lista) {
00200         if (this == &lista) {
00201             return *this;
00202         }
00203         clear();
00204         shared_ptr<Node<T>> ptr = lista.head;
00205         while (ptr != NULL) {
00206             push_back(ptr->data);
00207             ptr = ptr->next;
00208         }
00209         return *this;
00210     }
00213     List& operator=(List&& lista) {
00214         if (this == &lista) {
00215             return *this;
00216         }
00217         clear();
00218         shared_ptr<Node<T>> ptr = lista.head;
00219         while (ptr != NULL) {
00220             push_back(move(ptr->data));
00221             ptr = ptr->next;
00222         }
00223         lista.head = NULL;
00224         return *this;
00225     }
00228     ~List() {}
00229 };
00230
00231 template<typename T>
00232 void List<T>::push_front(const T& obj) {
00233     shared_ptr<Node<T>> node = make_shared<Node<T>>(obj);
00234     node->data = obj;
00235
00236     if (head != NULL)
00237         node->next = head;
00238     head = node;
00239 }
00240
00241 template<typename T>
00242 void List<T>::push_back(const T& obj) {
00243     shared_ptr<Node<T>> node = make_shared<Node<T>>(obj);
00244     node->data = obj;
00245
00246     if (head == NULL)
00247         head = node;
00248     else {
00249         shared_ptr<Node<T>> ptr = head;
00250         while (ptr->next != NULL) {
00251             ptr = ptr->next;
00252         }
00253         ptr->next = node;
00254     }
00255 }

```

```

00256
00257 template<typename T>
00258 void List<T>::pop front() {
00259     if (head != NULL)
00260         head = head->next;
00261 }
00262
00263 template<typename T>
00264 void List<T>::pop back() {
00265     if (head == NULL)
00266         return;
00267     else if (head->next == NULL)
00268         head = NULL;
00269     else {
00270         shared_ptr<Node<T>> ptr = head;
00271         while (ptr->next->next != NULL) {
00272             ptr = ptr->next;
00273         }
00274         ptr->next = NULL;
00275     }
00276 }
00277
00278 template<typename T>
00279 void List<T>::pop(const int index) {
00280     if (head == NULL || index < 0 || index >= size())
00281         return;
00282     else if (index == 0)
00283         pop_front();
00284     else {
00285         shared_ptr<Node<T>> ptr = head;
00286         int count = 0;
00287         while (++count != index)
00288             ptr = ptr->next;
00289         ptr->next = ptr->next->next;
00290     }
00291 }
00292
00293 template<typename T>
00294 T List<T>::get_element(const int index) {
00295     if (head != NULL) {
00296         shared_ptr<Node<T>> ptr = head;
00297         int count = 0;
00298         while (ptr != NULL) {
00299             if (count++ == index)
00300                 return ptr->data;
00301             ptr = ptr->next;
00302         }
00303     }
00304     return T();
00305 }
00306
00307 template<typename T>
00308 T List<T>::get_element2(const int id) {
00309     if (head != NULL) {
00310         shared_ptr<Node<T>> ptr = head;
00311         while (ptr != NULL) {
00312             if (ptr->data.id == id)
00313                 return ptr->data;
00314             ptr = ptr->next;
00315         }
00316     }
00317     return T();
00318 }
00319
00320 template<typename T>
00321 int List<T>::get_index(const T& dane) {
00322     if (head == NULL) {
00323         return -1;
00324     }
00325     shared_ptr<Node<T>> ptr = head;
00326     int index = 0;
00327     while (ptr != NULL) {
00328         if (ptr->data == dane) {
00329             return index;
00330         }
00331         ptr = ptr->next;
00332         ++index;

```



```

00333     }
00334     return -1;
00335 }
00336
00337 template<typename T>
00338 int List<T>::get_index(const int id) {
00339     if (head == NULL) {
00340         return -1;
00341     }
00342     shared_ptr<Node<T>> ptr = head;
00343     int index = 0;
00344     while (ptr != NULL) {
00345         if (ptr->data.id == id) {
00346             return index;
00347         }
00348         ptr = ptr->next;
00349         ++index;
00350     }
00351     return -1;
00352 }
00353
00354 template<typename T>
00355 void List<T>::remove(const T& obj) {
00356     if (head == NULL)
00357         return;
00358     else {
00359         shared_ptr<Node<T>> ptr = head;
00360         int index = 0;
00361         while (ptr != NULL) {
00362             if (obj == ptr->data) {
00363                 pop(index);
00364                 --index;
00365             }
00366             ++index;
00367             ptr = ptr->next;
00368         }
00369     }
00370 }
00371
00372 template<class T>
00373 bool List<T>::greater(const T& x, const T& y) {
00374     return !(x < y);
00375 }
00376
00377 template<class T>
00378 bool List<T>::smaller(const T& x, const T& y) {
00379     return (x < y);
00380 }
00381
00382 template<class T>
00383 bool List<T>::greater(const Tab1::Dane& a, const Tab1::Dane& b, const int choice)
00384 {
00385     switch (choice) {
00386     case 1:
00387         return a.id > b.id;
00388         break;
00389     case 2:
00390         return a.imie > b.imie;
00391         break;
00392     case 3:
00393         return a.nazwisko > b.nazwisko;
00394         break;
00395     case 4:
00396         if (a.rok != b.rok)
00397             return a.rok < b.rok;
00398         else if (a.miesiac != b.miesiac)
00399             return a.miesiac > b.miesiac;
00400         else
00401             return a.dzien > b.dzien;
00402         break;
00403     case 5:
00404         return a.plec > b.plec;
00405         break;
00406     default:
00407         return a.id > b.id;
00408         break;
00409     }
00410 }

```

```

00409 }
00410
00411 template<class T>
00412 bool List<T>::smaller(const Tab1::Dane& a, const Tab1::Dane& b, const int choice)
00413 {
00414     switch (choice) {
00415         case 1:
00416             return a.id < b.id;
00417             break;
00418         case 2:
00419             return a.imie < b.imie;
00420             break;
00421         case 3:
00422             return a.nazwisko < b.nazwisko;
00423             break;
00424         case 4:
00425             if (a.rok != b.rok)
00426                 return a.rok < b.rok;
00427             else if (a.miesiac != b.miesiac)
00428                 return a.miesiac < b.miesiac;
00429             else
00430                 return a.dzien < b.dzien;
00431             break;
00432         case 5:
00433             return a.plec < b.plec;
00434             break;
00435         default:
00436             return a.id < b.id;
00437             break;
00438     }
00439 }
00440
00441 template<class T>
00442 bool List<T>::greater(const Tab2::Wiezienie& a, const Tab2::Wiezienie& b, const int
00443 choice) {
00444     switch (choice) {
00445         case 1:
00446             return a.id > b.id;
00447             break;
00448         case 2:
00449             return a.przewinienie > b.przewinienie;
00450             break;
00451         case 3: {
00452             int months_a = a.dlugosc;
00453             if (a.jednostka == "lat" || a.jednostka == "lata" || a.jednostka == "rok")
00454                 months_a *= 12;
00455             else
00456                 months_a *= 1;
00457             int months_b = b.dlugosc;
00458             if (b.jednostka == "lat" || b.jednostka == "lata" || b.jednostka == "rok")
00459                 months_b *= 12;
00460             else
00461                 months_b *= 1;
00462             if (months_a != months_b)
00463                 return months_a > months_b;
00464             break;
00465         }
00466         case 4:
00467             return a.cela > b.cela;
00468             break;
00469         default:
00470             return a.id > b.id;
00471             break;
00472     }
00473 }
00474
00475 template<class T>
00476 bool List<T>::smaller(const Tab2::Wiezienie& a, const Tab2::Wiezienie& b, const int
00477 choice) {
00478     switch (choice) {
00479         case 1:
00480             return a.id < b.id;
00481             break;
00482         case 2:
00483             return a.przewinienie < b.przewinienie;

```

```

00483         break;
00484     case 3: {
00485         int months a = a.dlugosc;
00486         if (a.jednostka == "lat" || a.jednostka == "lata" || a.jednostka == "rok")
00487             months_a *= 12;
00488         else
00489             months a *= 1;
00490
00491         int months b = b.dlugosc;
00492         if (b.jednostka == "lat" || b.jednostka == "lata" || b.jednostka == "rok")
00493             months_b *= 12;
00494         else
00495             months b *= 1;
00496
00497         if (months a != months b)
00498             return months a < months b;
00499         break;
00500     }
00501     case 4:
00502         return a.cela < b.cela;
00503         break;
00504     default:
00505         return a.id < b.id;
00506         break;
00507     }
00508 }
00509
00510 template<typename T>
00511 void List<T>::sort(const int sort) {
00512     shared_ptr<Node<T>> ptr;
00513     T min;
00514     int count = 0;
00515     if (sort == 1)
00516         while (count != size()) {
00517             ptr = head;
00518             min = ptr->data;
00519             for (int i = 0; i < size() - count; ++i) {
00520                 if (smaller(ptr->data, min))
00521                     min = ptr->data;
00522                 ptr = ptr->next;
00523             }
00524             ++count;
00525             push_back(min);
00526             pop(get_index(min));
00527         }
00528     else if (sort == 2)
00529         while (count != size()) {
00530             ptr = head;
00531             min = ptr->data;
00532             for (int i = 0; i < size() - count; ++i) {
00533                 if (greater(ptr->data, min))
00534                     min = ptr->data;
00535                 ptr = ptr->next;
00536             }
00537             ++count;
00538             push_back(min);
00539             pop(get_index(min));
00540         }
00541 }
00542
00543 template<typename T>
00544 void List<T>::sort(const int sort, const int choice) {
00545     shared_ptr<Node<T>> ptr;
00546     T min;
00547     int count = 0;
00548     if (sort == 1)
00549         while (count != size()) {
00550             ptr = head;
00551             min = ptr->data;
00552             for (int i = 0; i < size() - count; ++i) {
00553                 if (smaller(ptr->data, min, choice))
00554                     min = ptr->data;
00555                 ptr = ptr->next;
00556             }
00557             ++count;
00558             push_back(min);
00559             pop(get_index(min));

```

```

00560     }
00561     else if (sort == 2)
00562         while (count != size()) {
00563             ptr = head;
00564             min = ptr->data;
00565             for (int i = 0; i < size() - count; ++i) {
00566                 if (greater(ptr->data, min, choice))
00567                     min = ptr->data;
00568                 ptr = ptr->next;
00569             }
00570             ++count;
00571             push_back(min);
00572             pop(get_index(min));
00573         }
00574     }
00575
00576 template<typename T>
00577 int List<T>::element_count(const string& obj) {
00578     shared_ptr<Node<T>> ptr = head;
00579     int count = 0;
00580     while (ptr != NULL) {
00581         if (to_string(ptr->data) == obj)
00582             ++count;
00583         ptr = ptr->next;
00584     }
00585     return count;
00586 }
00587
00588 template<typename T>
00589 int List<T>::element_count1(const int element, const string& obj) {
00590     shared_ptr<Node<T>> ptr = head;
00591     int count = 0;
00592     while (ptr != NULL) {
00593         switch (element) {
00594             case 1:
00595                 if (to_string(ptr->data.id) == obj)
00596                     ++count;
00597                 break;
00598             case 2:
00599                 if (ptr->data.imie == obj)
00600                     ++count;
00601                 break;
00602             case 3:
00603                 if (ptr->data.nazwisko == obj)
00604                     ++count;
00605                 break;
00606             case 4:
00607                 if (to_string(ptr->data.dzien) == obj)
00608                     ++count;
00609                 break;
00610             case 5:
00611                 if (to_string(ptr->data.miesiac) == obj)
00612                     ++count;
00613                 break;
00614             case 6:
00615                 if (to_string(ptr->data.rok) == obj)
00616                     ++count;
00617                 break;
00618             case 7:
00619                 if (ptr->data.plec == obj[0])
00620                     ++count;
00621                 break;
00622             default:
00623                 return 0;
00624                 break;
00625         }
00626         ptr = ptr->next;
00627     }
00628     return count;
00629 }
00630
00631 template<typename T>
00632 int List<T>::element_count2(const int element, const string& obj) {
00633     shared_ptr<Node<T>> ptr = head;
00634     int count = 0;
00635     while (ptr != NULL) {
00636         switch (element) {

```

```

00637         case 1:
00638             if (to_string(ptr->data.id) == obj)
00639                 ++count;
00640             break;
00641         case 2:
00642             if (ptr->data.przewinienie == obj)
00643                 ++count;
00644             break;
00645         case 3:
00646             if (to_string(ptr->data.dlugosc) == obj)
00647                 ++count;
00648             break;
00649         case 4:
00650             if (ptr->data.jednostka == obj)
00651                 ++count;
00652             break;
00653         case 5:
00654             if (ptr->data.cela == obj)
00655                 ++count;
00656             break;
00657         default:
00658             return 0;
00659             break;
00660     }
00661     ptr = ptr->next;
00662 }
00663 return count;
00664 }
00665
00666 template<typename T>
00667 int List<T>::arrest_length(const int id) {
00668     shared_ptr<Node<T>> ptr = head;
00669     int sum = 0;
00670     while (ptr != NULL) {
00671         if (ptr->data.id == id) {
00672             int months = ptr->data.dlugosc;
00673             if (ptr->data.jednostka == "lat" || ptr->data.jednostka == "lata" ||
ptr->data.jednostka == "rok")
00674                 months *= 12;
00675             else
00676                 months *= 1;
00677             sum += months;
00678         }
00679         ptr = ptr->next;
00680     }
00681     return sum;
00682 }
00683
00684 template<typename T>
00685 double List<T>::arrest_average(const string& co) {
00686     shared_ptr<Node<T>> ptr = head;
00687     int sum = 0, count = 0, tmp = 0;
00688     double sr;
00689     try {
00690         tmp = stoi(co);
00691     }
00692     catch (const invalid_argument& exp) {
00693     }
00694     while (ptr != NULL) {
00695         if (ptr->data.przewinienie == co || co == "all" || ptr->data.id == tmp) {
00696             int months = ptr->data.dlugosc;
00697             if (ptr->data.jednostka == "lat" || ptr->data.jednostka == "lata" ||
ptr->data.jednostka == "rok")
00698                 months *= 12;
00699             else
00700                 months *= 1;
00701             sum += months;
00702             ++count;
00703         }
00704         ptr = ptr->next;
00705     }
00706     if (count == 0)
00707         return 0;
00708     sr = sum / count;
00709     return sr;
00710 }
00711 }

```

```

00712
00713 template<typename T>
00714 List<T> List<T>::avarage greater(const double length, const string& co) {
00715     List<T> result;
00716     int count = 0, months = 0;
00717     sort(1, 1);
00718     shared_ptr<Node<T>> ptr = head;
00719     while (ptr != NULL) {
00720         if (ptr->data.przewinienie == co || co == "all") {
00721             if (ptr->next != NULL && ptr->data.id == (ptr->next)->data.id && count
== 0) {
00722                 shared_ptr<Node<T>> ptr2 = ptr;
00723                 int sum = 0;
00724                 double sr;
00725                 while (ptr2 != NULL) {
00726                     months = ptr2->data.dlugosc;
00727                     if (ptr2->data.jednostka == "lat" || ptr2->data.jednostka ==
"lata" || ptr2->data.jednostka == "rok")
00728                         months *= 12;
00729                     else
00730                         months *= 1;
00731                     sum += months;
00732                     ++count;
00733
00734                     if (ptr2->next == NULL || ptr2->data.id !=
(ptr2->next)->data.id)
00735                         break;
00736                     ptr2 = ptr2->next;
00737                 }
00738                 sr = sum / count * 1.0;
00739                 if (sr <= length) {
00740                     ptr = ptr2;
00741                     count = 0;
00742                 }
00743                 sum = 0;
00744             }
00745             else if (count == 0) {
00746                 months = ptr->data.dlugosc;
00747                 if (ptr->data.jednostka == "lat" || ptr->data.jednostka == "lata"
|| ptr->data.jednostka == "rok")
00748                     months *= 12;
00749                 else
00750                     months *= 1;
00751                 if (months > length)
00752                     result.push_back(ptr->data);
00753             }
00754             if (count > 0) {
00755                 result.push_back(ptr->data);
00756                 --count;
00757             }
00758         }
00759         ptr = ptr->next;
00760     }
00761     return result;
00762 }
00763
00764 template<typename T>
00765 List<T> List<T>::block check(const char blok) {
00766     List<T> result;
00767     shared_ptr<Node<T>> ptr = head;
00768     while (ptr != NULL) {
00769         if (ptr->data.cela.back() == blok && result.element_count2(1,
to_string(ptr->data.id)) == 0)
00770             result.push_back(ptr->data);
00771         ptr = ptr->next;
00772     }
00773     return result;
00774 }
00775
00776 template<typename T>
00777 List<T> List<T>::born during(const int from day, const int from month, const int
from year, const int to day, const int to month, const int to year) {
00778     List<T> result;
00779     shared_ptr<Node<T>> ptr = head;
00780     while (ptr != NULL) {
00781         if (ptr->data.rok > from year && ptr->data.rok < to year)
00782             result.push_back(ptr->data);

```

```

00783         else if (from_year == to_year && ptr->data.rok == from_year) {
00784             if (ptr->data.miesiac > from_month && ptr->data.miesiac < to_month)
00785                 result.push_back(ptr->data);
00786             else if (from_month == to_month && ptr->data.miesiac == from_month) {
00787                 if (ptr->data.dzien >= from_day && ptr->data.dzien <= to_day)
00788                     result.push_back(ptr->data);
00789             }
00790             else if (ptr->data.miesiac == from_month) {
00791                 if (ptr->data.dzien >= from_day)
00792                     result.push_back(ptr->data);
00793             }
00794             else if (ptr->data.miesiac == to_month) {
00795                 if (ptr->data.dzien <= to_day)
00796                     result.push_back(ptr->data);
00797             }
00798         }
00799         else if (ptr->data.rok == from_year) {
00800             if (ptr->data.miesiac > from_month)
00801                 result.push_back(ptr->data);
00802             else if (ptr->data.miesiac == from_month)
00803                 if (ptr->data.dzien >= from_day)
00804                     result.push_back(ptr->data);
00805         }
00806         else if (ptr->data.rok == to_year) {
00807             if (ptr->data.miesiac < to_month)
00808                 result.push_back(ptr->data);
00809             else if (ptr->data.miesiac == to_month)
00810                 if (ptr->data.dzien <= to_day)
00811                     result.push_back(ptr->data);
00812         }
00813         ptr = ptr->next;
00814     }
00815     return result;
00816 }
00817
00818 template<typename T>
00819 void List<T>::clear() {
00820     while (size() != 0) {
00821         pop_front();
00822     }
00823 }
00824
00825 template<typename T>
00826 int List<T>::size() {
00827     shared_ptr<Node<T>> ptr = head;
00828     int count = 0;
00829     while (ptr != NULL) {
00830         ++count;
00831         ptr = ptr->next;
00832     }
00833     return count;
00834 }
00835
00836 template<typename T>
00837 bool List<T>::is_empty() {
00838     if (head == NULL)
00839         return true;
00840     return false;
00841 }
00842
00843 template<typename T>
00844 void List<T>::serialization(const string& nazwa) {
00845     ofstream plik(nazwa, ios::binary);
00846     if (plik) {
00847         shared_ptr<Node<T>> ptr = head;
00848         while (ptr != NULL) {
00849             plik.write(reinterpret_cast<char*>(&ptr->data), sizeof(T));
00850             ptr = ptr->next;
00851         }
00852         plik.close();
00853     }
00854 }
00855
00856 template<typename T>
00857 void List<T>::deserialization(const string& nazwa) {
00858     ifstream plik(nazwa, ios::binary);
00859     if (plik) {

```

```

00860         while (plik) {
00861             T data;
00862             plik.read(reinterpret_cast<char*>(&data), sizeof(T));
00863             if (!plik.eof())
00864                 push_back(data);
00865         }
00866         plik.close();
00867     }
00868 }
00869
00870 template<typename T>
00871 void List<T>::serialization_string(const string& nazwa) {
00872     ofstream plik(nazwa, ios::binary);
00873     if (plik) {
00874         shared_ptr<Node<T>> ptr = head;
00875         while (ptr != NULL) {
00876             int length = ptr->data.length();
00877             plik.write(reinterpret_cast<char*>(&length), sizeof(int));
00878             plik.write(ptr->data.c_str(), length);
00879
00880             ptr = ptr->next;
00881         }
00882         plik.close();
00883     }
00884 }
00885
00886 template<typename T>
00887 void List<T>::deserialization_string(const string& nazwa) {
00888     ifstream plik(nazwa, ios::binary);
00889     if (plik) {
00890         while (plik) {
00891             int length;
00892             plik.read(reinterpret_cast<char*>(&length), sizeof(int));
00893
00894             if (!plik.eof()) {
00895                 char* tmp = new char[length];
00896                 plik.read(tmp, length);
00897                 T data(tmp, length);
00898                 push_back(data);
00899                 delete[] tmp;
00900             }
00901         }
00902         plik.close();
00903     }
00904 }
00905
00906 template <typename T>
00907 void List<T>::displayElement(const int index) {
00908     shared_ptr<Node<T>> ptr = head;
00909     if (ptr == NULL)
00910         cout << "List is empty! " << endl;
00911     else if (index >= 0 && index < size()) {
00912         cout << "Element: " << endl;
00913         int count = 0;
00914         while (count++ != index) {
00915             ptr = ptr->next;
00916         }
00917         if constexpr (is_same_v<T, Tab1::Dane>) {
00918             cout << ptr->data.id << " " << ptr->data.imie << " " << ptr->data.nazwisko
00919             << " "
00920             << ptr->data.dzien << " " << ptr->data.miesiac << " " <<
ptr->data.rok << " " << ptr->data.plec << endl;
00921         }
00922         else if constexpr (is_same_v<T, Tab2::Wiezienie>) {
00923             cout << ptr->data.id << " " << ptr->data.przewinienie << " " <<
ptr->data.dlugosc << " "
00924             << ptr->data.jednostka << " " << ptr->data.cela << endl;
00925         }
00926         else
00927             cout << ptr->data << " " << endl;
00928     }
00929 }
00930
00931 template <typename T>
00932 void List<T>::displayList() {
00933     shared_ptr<Node<T>> ptr = head;
00934     if (ptr == NULL)

```



```

00934         cout << "List is empty! " << endl;
00935     else {
00936         cout << "Lista zawiera: " << endl;
00937         while (ptr != NULL) {
00938             if constexpr (is_same_v<T, Tab1::Dane>) {
00939                 cout << ptr->data.id << ", " << ptr->data.imie << ", " <<
ptr->data.nazwisko << ", "
00940                     << ptr->data.dzien << "." << ptr->data.miesiac << "." <<
ptr->data.rok << ", " << ptr->data.plec << endl;
00941             }
00942             else if constexpr (is_same_v<T, Tab2::Wiezienie>) {
00943                 cout << ptr->data.id << ", " << ptr->data.przewinienie << ", " <<
ptr->data.dlugosc << ", "
00944                     << ptr->data.jednostka << ", " << ptr->data.cela << endl;
00945             }
00946             else
00947                 cout << ptr->data << " " << endl;
00948             ptr = ptr->next;
00949         }
00950     }
00951 }
00952
00953 template <typename T>
00954 void List<T>::zapis(const string& nazwa) {
00955     ofstream plik(nazwa);
00956     if (plik) {
00957         shared_ptr<Node<T>> ptr = head;
00958         if (ptr == NULL)
00959             plik << endl << "List is empty! " << endl;
00960         else {
00961             plik << endl << "List contains: " << endl;
00962             while (ptr != NULL) {
00963                 if constexpr (is_same_v<T, Tab1::Dane>) {
00964                     plik << ptr->data.id << ", " << ptr->data.imie << ", " <<
ptr->data.nazwisko << ", " << ptr->data.dzien << "." << ptr->data.miesiac << "." <<
ptr->data.rok << ", " << ptr->data.plec << endl;
00965                 }
00966                 else if constexpr (is_same_v<T, Tab2::Wiezienie>) {
00967                     plik << ptr->data.id << ", " << ptr->data.przewinienie << ", " <<
ptr->data.dlugosc << " " << ptr->data.jednostka << ", " << ptr->data.cela << endl;
00968                 }
00969                 else
00970                     plik << ptr->data << " " << endl;
00971                 ptr = ptr->next;
00972             }
00973         }
00974         plik.close();
00975     }
00976     else
00977         cout << "Blad! Nie mozna otworzyc pliku! " << endl;
00978 }
00979
00980 template <typename T>
00981 void List<T>::zapis2(const string& nazwa) {
00982     ofstream plik(nazwa, ios::app);
00983     if (plik) {
00984         shared_ptr<Node<T>> ptr = head;
00985         if (ptr == NULL)
00986             plik << endl << "List is empty! " << endl;
00987         else {
00988             plik << endl << "List contains: " << endl;
00989             while (ptr != NULL) {
00990                 if constexpr (is_same_v<T, Tab1::Dane>) {
00991                     plik << ptr->data.id << ", " << ptr->data.imie << ", " <<
ptr->data.nazwisko << ", " << ptr->data.dzien << "." << ptr->data.miesiac << "." <<
ptr->data.rok << ", " << ptr->data.plec << endl;
00992                 }
00993                 else if constexpr (is_same_v<T, Tab2::Wiezienie>) {
00994                     plik << ptr->data.id << ", " << ptr->data.przewinienie << ", " <<
ptr->data.dlugosc << " " << ptr->data.jednostka << ", " << ptr->data.cela << endl;
00995                 }
00996                 else
00997                     plik << ptr->data << " " << endl;
00998                 ptr = ptr->next;
00999             }
01000         }
01001         plik.close();

```

```
01002     }  
01003     else  
01004         cout << "Bład! Nie mozna otworzyc pliku! " << endl;  
01005 }
```