Dokumentacja Programu: System Zarządzania Bazą Adresów Osób

Sformułowanie zadania

Projekt i implementacji systemu wspomagającego zarządzanie zbiorem adresów osób. Automatyczne grupowanie po zainteresowaniach.

Funkcjonalności systemu

1. **Dodawanie osób**: Umożliwia użytkownikowi wprowadzenie danych osobowych (imię, nazwisko, adres, email) i listy zainteresowań. Dane przechowywane w odpowiedniej strukturze danych.
2. **Usuwanie osób:** Pozwala użytkownikowi usunąć osobę z bazy na podstawie unikalnego identyfikatora.
3. **Edycja danych osoby:** Możliwość aktualizacji danych osobowych i zainteresowań.
4. **Wyświetlanie listy osób:** Wyświetla listę wszystkich zapisanych osób w przejrzysty sposób.
5. **Automatyczne grupowanie po zainteresowaniach:** Tworzenie grup osób na podstawie wspólnych zainteresowań. Algorytm analizuje listy zainteresowań i grupuje osoby z co najmniej jednym wspólnym zainteresowaniem.
6. **Wyszukiwanie osób po parametrze:** Umożliwia wyświetlenie osób z szukanym przez nas parametrem (np. imię Anna).
7. **Eksport i import danych:** Zapis i odczyt bazy danych z pliku w formacie tekstowym, co pozwala na trwałe przechowywanie informacji.

Kluczowe komponenty implementacji

1. **Struktury danych:** Jako struktura zostanie użyty struct. Dane o osobach przechowywane będą w formie obiektu.
2. **Logika grupowania:** Wyszukiwanie osób z co najmniej jednym wspólnym zainteresowaniem. Tworzenie dynamicznych grup na podstawie podobieństwa zainteresowań.
3. **Obsługa plików:** Zapisywanie danych w pliku txt, gdzie każda linia zawiera dane osoby (imię, nazwisko, adres, email, zainteresowania rozdzielone przecinkami). Wczytywanie danych z pliku.
4. **Interfejs użytkownika:** Menu tekstowe umożliwiające wybór opcji (dodanie osoby, usunięcie, edycja, wyświetlanie grup, zapis/odczyt z pliku). Obsługa wejścia i częściowa walidacja wprowadzonych danych.
5. **Moduł wyszukiwania:** Filtrowanie osób na podstawie wybranych kryteriów, np. wyszukiwanie wszystkich osób nazywających się “Jan”.

Struktura programu

Struktura Person

```
struct Person {
   int id;              // Unikalny identyfikator osoby.
   string name;         // Imię.
   string surname;      // Nazwisko.
   string address;      // Adres zamieszkania.
   string email;        // Adres e-mail.
   string* hobbies = new string[20]; // Lista zainteresowań (maksymalnie 20).
};
```

Poza podstawowymi danymi obecnymi w strukturze jest tam też obecne ID. Jest ono niezbędne przy niektórych funkcjach które operują na konkretnych rekordach w pliku (np. usuwanie). ID jest unikalnym elementem dla każdej osoby w bazie co pozwala na działanie na konkretnym rekordzie i zapobiega np. usunięciu złego rekordu.

Struktura danych w pliku database.cpp

"ID;Imię;Nazwisko;Adres;Email;[zainteresowanie1, zainteresowanie2]"

Przykładowy rekord w bazie

1;Mateusz;Kowalski;ul. Lipowa 5, Warszawa;jan.kowalski@example.com;[Czytanie, Programowanie, Turystyka]

Podział na pliki w projekcie

Projekt został podzielony na kilka plików w celu zapewnienia modularności, przejrzystości i łatwości zarządzania kodem. Oto szczegółowy opis podziału i zawartości poszczególnych plików:

1. main.cpp

Opis: Plik główny programu, który zawiera funkcję main oraz obsługę menu użytkownika.

Zawartość:

- Funkcja int main() – punkt wejścia programu.
- Funkcja MenuDisplay() – wyświetla menu główne i umożliwia wybór opcji.

2. utility.cpp

Opis: Plik implementujący funkcje pomocnicze, używane w różnych częściach programu.

Zawartość:

- void handleExit(const string& input) – obsługuje wyjście z programu na podstawie wprowadzonego tekstu.
- string getInput(const string& prompt) – pobiera dane wejściowe od użytkownika w postaci tekstu.
- int getIntInput(const string& prompt) – pobiera i waliduje dane wejściowe w postaci liczby całkowitej.
- void GetPersonId(int& id) – pobiera i waliduje ID osoby wprowadzone przez użytkownika.

3. database_operations.cpp

Opis: Plik implementujący funkcje zarządzania bazą danych.

Zawartość:

- Person GeneratePerson() – tworzy nowy obiekt Person na podstawie danych wprowadzonych przez użytkownika.
- void AddPerson() – dodaje nową osobę do bazy danych.
- ProcessLineToDelete(const string& line, ofstream& outfile, int id, bool& found) - przetwarza bazę w celu usunięcia rekordu o podanym ID
- void DeletePerson() – usuwa osobę z bazy danych na podstawie ID.
- void updatePersonData(Person& person) – aktualizuje dane osoby.
- Person ParsePerson(const string& line) – konwertuje linię z pliku tekstowego na obiekt Person.
- bool ProcessLine(const string& line, int choice, const string& searchTerm) – sprawdza, czy linia z pliku pasuje do określonego kryterium wyszukiwania.
- void EditPerson() – edytuje dane istniejącej osoby w bazie.
- void SearchPerson() - wyszukuje rekordy posiadające podane kryterium.
- void DisplayDatabase() – wyświetla wszystkie rekordy w bazie danych.

4. hobby_operations.cpp

Opis: Plik implementujący funkcje związane z grupowaniem i wyświetlaniem zainteresowań.

Zawartość:

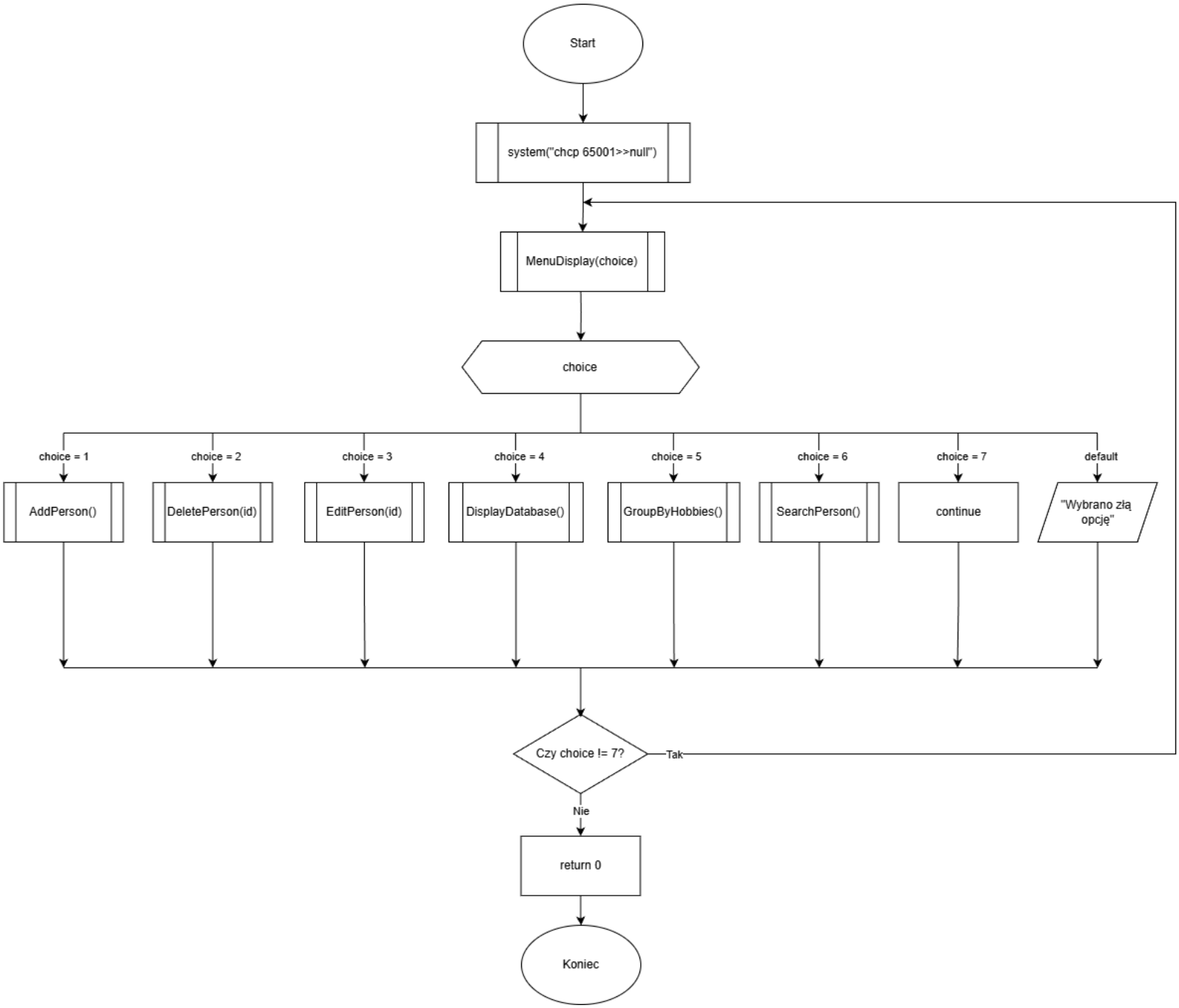
- void GroupByHobbies() – grupuje osoby według ich zainteresowań i wyświetla wyniki.
- void ParseLine(const string& line, string hobbies[], string groupedPeople[][100], int hobbyCounts[], int& hobbyIndex, int maxPeople) – przetwarza linię tekstu i grupuje osoby według zainteresowań.
- void DisplayGroupedHobbies(const string hobbies[], const string groupedPeople[][100], const int hobbyCounts[], int hobbyIndex) – wyświetla grupy osób przypisane do konkretnych zainteresowań.

Funkcje

Plik main.cpp

main

Schemat blokowy:



Opis: Jest punktem wejścia do programu, obsługuje główną pętlę programu oraz logikę sterującą działaniem aplikacji.

Parametry wejściowe: Brak.

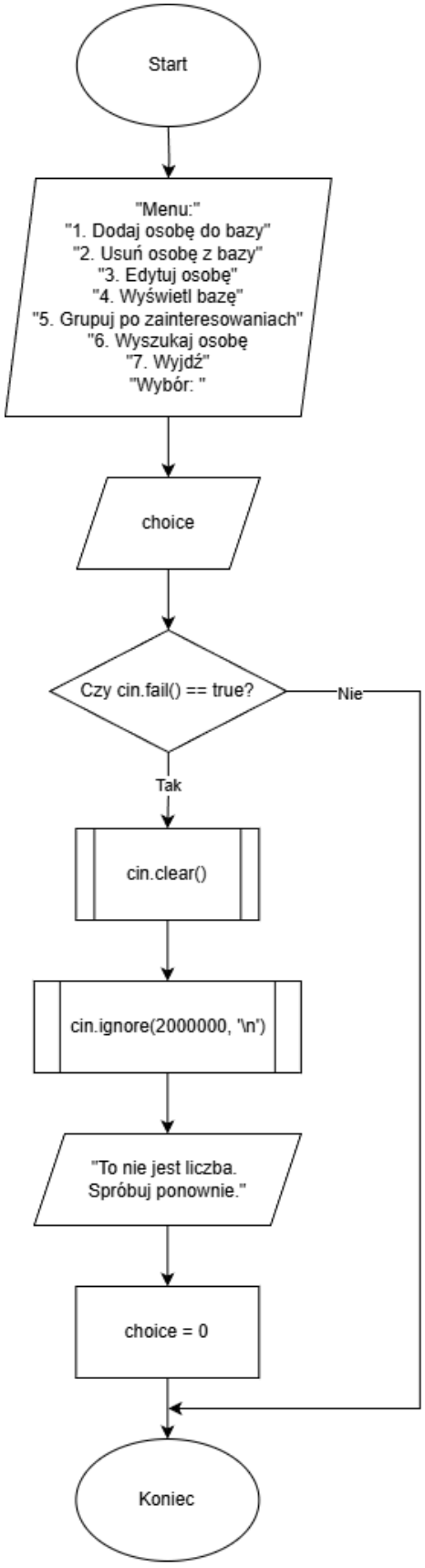
Wartość zwracana: int.

Działanie:

1. Wywołuje system("chcp 65001>>null"), aby ustawić kodowanie UTF-8.
2. Wywołuje MenuDisplay, aby wyświetlić menu główne i pobrać wybór.
3. Wykorzystuje instrukcję switch do wykonania odpowiednich funkcji:
    a. AddPerson – dodawanie nowej osoby.
    b. DeletePerson – usuwanie osoby.
    c. EditPerson – edycja danych osoby.
    d. DisplayDatabase – wyświetlanie całej bazy danych.
    e. GroupByHobbies – grupowanie według zainteresowań.
    f. SearchPerson – wyszukiwanie osoby w bazie danych.
4. Obsługa błędów:
    a. Wyłapuje wyjątki runtime_error i wyświetla komunikat o powrocie do menu.
5. Zakończenie programu:
    a. Program kończy się, gdy użytkownik wybierze opcję wyjścia (choice == 7).

MenuDisplay

Schemat blokowy:



Opis: Wyświetla menu główne programu i pobiera wybór użytkownika.

Parametry wejściowe: int& choice.

Wartość zwracana: Brak.

Działanie:

1. Wyświetla listę dostępnych opcji:
   1. Dodaj osobę do bazy
   2. Usuń osobę z bazy
   3. Edytuj osobę
   4. Wyświetl bazę
   5. Grupuj po zainteresowaniach
   6. Wyszukaj osobę
   7. Wyjdź
2. Pobiera wybór użytkownika za pomocą cin.
3. Sprawdza, czy użytkownik wprowadził poprawną liczbę:
   a. Jeśli nie (cin.fail()), czyści strumień wejściowy i wyświetla komunikat o błędzie.
   b. Ustawia choice na 0, aby wymusić ponowne wyświetlenie menu.

Plik utility.cpp

HandleExit

Opis: Obsługuje wyjście z programu na podstawie wprowadzonego tekstu.

Parametry: input (string).

Wartość zwracana: Brak.

Działanie:

1. Sprawdza, czy wprowadzone dane są równe "wyjdź".
2. Jeśli tak, rzuca wyjątek runtime_error.
3. W przeciwnym razie kończy działanie bez dodatkowych operacji.

GetInput

Opis: Pobiera dane wejściowe od użytkownika w postaci tekstu.

Parametry: prompt (string): Komunikat wyświetlany przed oczekiwaniem na dane wejściowe.

Wartość zwracana: string.

Działanie:

1. Wyświetla komunikat zawarty w zmiennej prompt.
2. Odczytuje linię tekstu wprowadzonego przez użytkownika.
3. Zwraca wprowadzony ciąg znaków.

GetIntInput

Opis: Pobiera dane wejściowe od użytkownika w postaci liczby całkowitej.

Parametry: prompt (string): Komunikat wyświetlany przed oczekiwaniem na dane wejściowe.

Wartość zwracana: int.

Działanie:

1. Wyświetla komunikat prompt.
2. Oczekuje na wprowadzenie danych przez użytkownika.
3. Odczytuje dane.
4. Wywołuje funkcję HandleExit().
5. Zwraca dane przekonwertowane na typ int.

GetPersonId

Opis: Pobiera i waliduje ID osoby wprowadzone przez użytkownika.

Parametry: id (int&).

Wartość zwracana: Brak.

Działanie:

1. Wyświetla komunikat "Podaj id osoby: ".
2. Próbuje odczytać dane jako liczbę całkowitą.
3. Jeśli odczyt się nie powiedzie:
4. Czyści strumień wejściowy.
5. Usuwa pozostałe dane z bufora.
6. Wyświetla komunikat o błędzie.
7. Rzuca wyjątek runtime_error.
8. Jeśli odczyt się powiedzie, przypisuje wprowadzone ID do zmiennej id.

Plik database_operations.cpp

GeneratePerson

Opis: Tworzy nową osobę na podstawie danych wprowadzonych przez użytkownika.

Parametry: Brak.

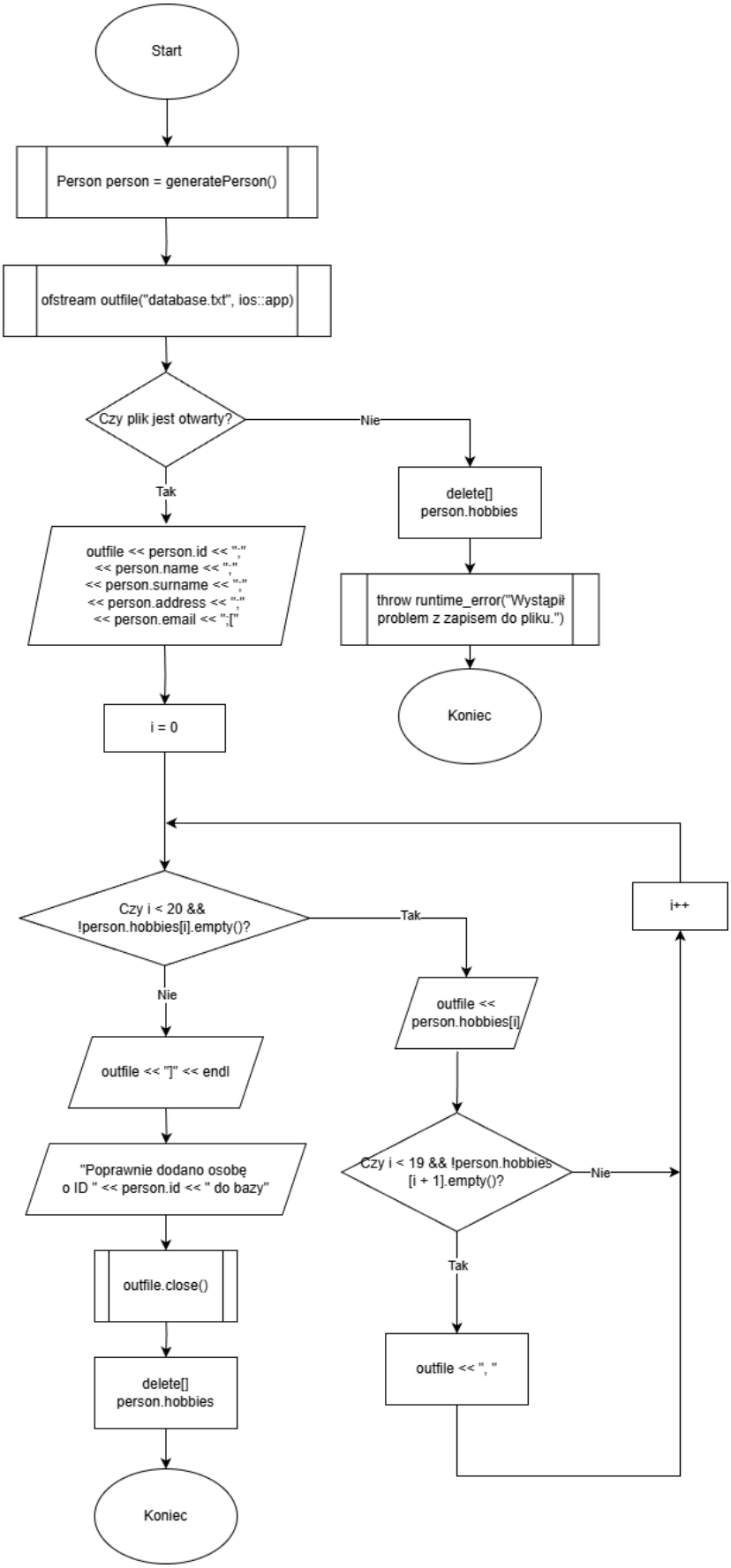
Wartość zwracana: Person.

Działanie:

1. Inicjalizuje obiekt Person i alokuje pamięć na zainteresowania.
2. Wyświetla informację, że można wpisać "wyjdź", aby przerwać.
3. Pobiera od użytkownika dane: imię, nazwisko, adres, email, liczbę zainteresowań oraz ich listę.
4. Odczytuje ostatnie ID z pliku database.txt i generuje nowe ID jako lastID + 1.
5. Wypełnia obiekt Person danymi i zwraca go.

AddPerson

Schemat blokowy:



Opis: Dodaje nową osobę do bazy danych.

Parametry: Brak.

Wartość zwracana: Brak.

Działanie:

1. Wywołuje funkcję generatePerson, aby utworzyć nową osobę.
2. Otwiera plik database.txt w trybie dopisywania.
3. Zapisuje dane nowej osoby w pliku w odpowiednim formacie.
4. Wyświetla komunikat o sukcesie lub obsługuje błędy związane z dostępem do pliku.

ProcessLineToDelete

Opis: Przetwarza pojedynczą linię tekstu z pliku bazy danych w celu sprawdzenia, czy odpowiada podanemu ID. Jeśli tak, umożliwia użytkownikowi usunięcie lub pozostawienie rekordu.

Parametry wejściowe:

- const string& line – Pojedyncza linia tekstu z pliku bazy danych.
- ofstream& outfile – Strumień wyjściowy, do którego zapisywane są przetworzone linie.
- int id – ID rekordu, który ma zostać sprawdzony.
- bool& found – Referencja do zmiennej, która informuje, czy rekord o podanym ID został znaleziony.

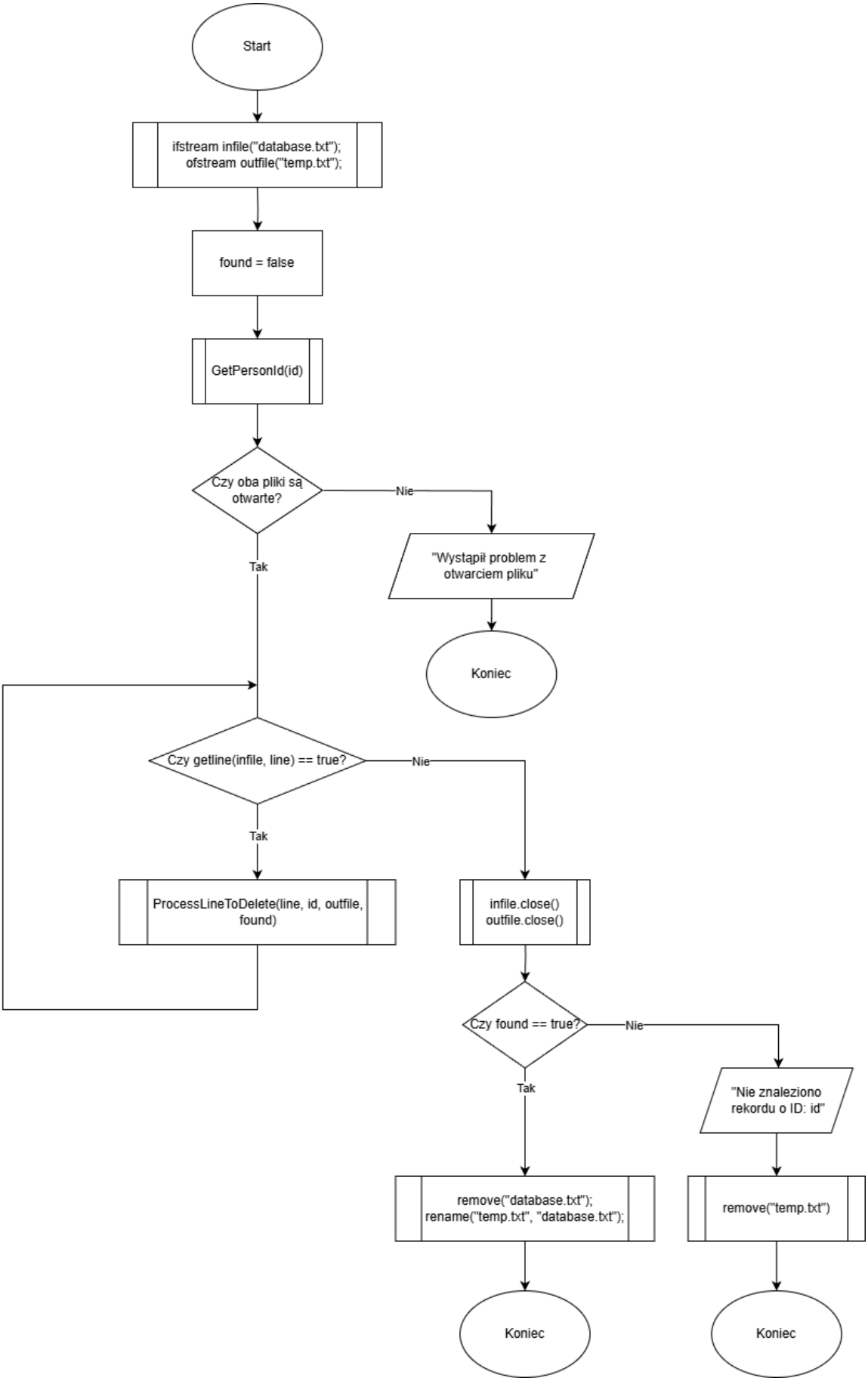
Wartość zwracana: Brak

Działanie:

1. Pobiera numer ID rekordu z początku linii, wykorzystując find do znalezienia pozycji średnika (;) i substr do wyodrębnienia identyfikatora.
2. Porównuje wyodrębniony ID z podanym id.
3. Jeśli ID się zgadza:
    - Oznacza rekord jako znaleziony (found = true).
    - Wyświetla treść rekordu na ekranie i pyta użytkownika, czy chce go usunąć.
    - Pozwala użytkownikowi na wybór:
        - "t"/"T": Usunięcie rekordu (rekord nie jest zapisywany do pliku wyjściowego).
        - "n"/"N": Zatrzymanie rekordu (rekord jest zapisywany do pliku wyjściowego).
4. Jeśli ID się nie zgadza:
    - Zapisuje linię do pliku wyjściowego bez zmian.

DeletePerson

Schemat blokowy:



Opis: Usuwa osobę z bazy danych na podstawie ID.

Parametry: Brak.

Wartość zwracana: Brak.

Działanie:

1. Pobiera ID osoby do usunięcia za pomocą funkcji GetPersonId.
2. Otwiera plik database.txt w trybie odczytu i tworzy plik tymczasowy temp.txt.
3. Iteruje przez każdy rekord w pliku i wywołuje funkcję ProcessLineToDelete().
4. Jeśli szukany rekord został znaleziony zastępuje plik database.txt plikiem temp.txt.
5. Wyświetla komunikat o sukcesie lub informuje, że rekord nie został znaleziony.

UpdatePersonData

Opis: Aktualizuje dane istniejącego obiektu Person na podstawie wprowadzonych przez użytkownika wartości.

Parametry wejściowe: Person& person.

Wartość zwracana: Brak.

Działanie:

1. Dla każdego pola (name, surname, address, email): Pyta użytkownika o nowe dane. Jeśli pole pozostanie puste, zachowuje poprzednią wartość.
2. Pyta użytkownika o nowe zainteresowania w formacie oddzielonym przecinkami. Jeśli użytkownik poda nowe zainteresowania, zastępuje istniejące w tablicy hobbies.
3. Aktualizuje tablicę hobbies zgodnie z nowymi danymi.

ParsePerson

Opis: Konwertuje pojedynczą linię tekstu z pliku database.txt na obiekt Person.

Parametry wejściowe: const string& line – pojedyncza linia tekstu z pliku database.txt.

Wartość zwracana: Person.

Działanie (krok po kroku):

1. Rozdziela linię tekstu na poszczególne pola (id, name, surname, address, email, hobbies) za pomocą średników (;) i nawiasów kwadratowych ([]).
2. Wypełnia strukturę Person na podstawie wyodrębnionych danych:
3. ID jest konwertowane na liczbę całkowitą (stoi).
4. Hobbies są przetwarzane jako lista oddzielona przecinkami i zapisywane w tablicy.
5. Zwraca wypełniony obiekt Person.

EditPerson

Opis: Umożliwia edycję danych osoby w bazie danych.

Parametry: Brak.

Wartość zwracana: Brak.

Działanie:

1. Pobiera ID osoby do edycji za pomocą funkcji GetPersonId.
2. Otwiera plik database.txt w trybie odczytu i tworzy plik tymczasowy temp.txt.
3. Iteruje przez każdy rekord w pliku:
4. Jeśli ID pasuje do wprowadzonego ID, wywołuje funkcję parsePerson, aby wczytać dane osoby.
5. Wyświetla dane osoby i umożliwia ich edycję za pomocą updatePersonData.
6. Zapisuje zaktualizowane dane do pliku temp.txt.
7. Jeśli ID nie pasuje, kopiuje rekord bez zmian do pliku temp.txt.
8. Zastępuje plik database.txt plikiem temp.txt.
9. Wyświetla komunikat o sukcesie lub informuje, że rekord nie został znaleziony.

DisplayDatabase

Opis: Wyświetla wszystkie rekordy w bazie danych.

Parametry: Brak.

Wartość zwracana: Brak.

Działanie:

1. Otwiera plik database.txt w trybie odczytu.
2. Jeśli plik otwiera się poprawnie:
3. Wyświetla nagłówek tabeli (ID, Imię, Nazwisko, Adres, Email, Zainteresowania).
4. Iteruje przez każdą linię w pliku i wyświetla ją na ekranie.
5. Zamyka plik po zakończeniu odczytu.
6. Jeśli plik nie otworzy się, wyświetla komunikat o błędzie.

ProcessLine

Opis: Sprawdza, czy dana linia tekstu spełnia określone kryterium wyszukiwania.

Parametry wejściowe:

const string& line – linia tekstu z pliku database.txt.
int choice – wybrane kryterium wyszukiwania.
const string& searchTerm – szukany termin.

Wartość zwracana: true/false.

Działanie:

1. Rozdziela linię tekstu na poszczególne pola (id, name, surname, address, email, hobbies) za pomocą średników (;) i nawiasów kwadratowych ([]).
2. W zależności od wybranego kryterium (choice), porównuje wartość pola z wyszukiwanym terminem (searchTerm):
3. ID: Sprawdza, czy id jest równe searchTerm.
4. Imię: Sprawdza, czy name jest równe searchTerm.
5. Nazwisko: Sprawdza, czy surname jest równe searchTerm.
6. E-mail: Sprawdza, czy email jest równe searchTerm.
7. Inne: Sprawdza, czy address lub hobbies zawierają searchTerm.
8. Zwraca true, jeśli warunek jest spełniony, w przeciwnym razie false.

SearchPerson

Opis: Wyszukuje osoby w bazie danych na podstawie wybranego kryterium.

Parametry: Brak.

Wartość zwracana: Brak.

Działanie:

1. Pyta użytkownika o kryterium wyszukiwania (ID, Imię, Nazwisko, Email, Adres/Zainteresowania).
2. Pobiera wartość wyszukiwaną od użytkownika.
3. Otwiera plik database.txt w trybie odczytu.
4. Iteruje przez każdą linię w pliku:
5. Wywołuje funkcję ProcessLine, aby sprawdzić, czy linia spełnia kryterium wyszukiwania.
6. Jeśli linia pasuje, wyświetla ją na ekranie.
7. Jeśli żadne rekordy nie spełniają kryteriów, wyświetla komunikat o braku wyników.
8. Obsługuje błędy związane z otwieraniem pliku lub wprowadzaniem danych.

Plik hobby_operations.cpp

ParseLine

Opis: Przetwarza pojedynczą linię tekstu z bazy danych, grupując osoby według zainteresowań.

Parametry wejściowe:

const string& line – linia tekstu z pliku database.txt.
string hobbies[] – tablica przechowująca unikalne zainteresowania.
string groupedPeople[][100] – tablica przechowująca osoby przypisane do zainteresowań.
int hobbyCounts[] – tablica liczników osób dla każdego zainteresowania.
int& hobbyIndex – licznik unikalnych zainteresowań.
int maxPeople – maksymalna liczba osób na jedno zainteresowanie.

Wartość zwracana: Brak (funkcja działa poprzez modyfikację przekazanych tablic).

Działanie:

1. Rozdziela linię na pola: identyfikator, imię, nazwisko i listę zainteresowań.
2. Dla każdego zainteresowania sprawdza, czy już istnieje w tablicy hobbies:
3. Jeśli nie istnieje, dodaje je do tablicy i przypisuje mu nowy indeks.
4. Jeśli istnieje, wykorzystuje jego istniejący indeks.
5. Dodaje imię i nazwisko osoby do odpowiedniej grupy w groupedPeople na podstawie indeksu zainteresowania.
6. Aktualizuje licznik osób przypisanych do każdego zainteresowania.

DisplayGroupedHobbies

Opis: Wyświetla grupy osób przypisane do zainteresowań.

Parametry wejściowe:

const string hobbies[] – tablica przechowująca unikalne zainteresowania.
const string groupedPeople[][100] – tablica przechowująca osoby przypisane do zainteresowań.
const int hobbyCounts[] – tablica liczników osób dla każdego zainteresowania.
int hobbyIndex – liczba unikalnych zainteresowań.

Wartość zwracana: Brak (funkcja wyświetla dane na standardowe wyjście).

Działanie:

1. Iteruje przez tablicę hobbies, wyświetlając każde zainteresowanie.

2. Dla każdego zainteresowania iteruje przez tablicę groupedPeople, wyświetlając przypisane do niego osoby.
3. Jeśli dane zainteresowanie nie ma przypisanych osób, pomija jego wyświetlanie.

GroupByHobbies

Opis: Grupuje osoby według ich zainteresowań i wyświetla wyniki.

Parametry: Brak.

Wartość zwracana: Brak.

Działanie:

1. Inicjalizuje tablice do przechowywania zainteresowań, grup osób i liczników osób.
2. Otwiera plik database.txt w trybie odczytu.
3. Iteruje przez każdą linię w pliku i wywołuje funkcję ParseLine w celu przetworzenia danych.
4. Po zakończeniu przetwarzania danych wywołuje funkcję DisplayGroupedHobbies, aby wyświetlić wyniki.
5. Jeśli plik nie otworzy się, wyświetla komunikat o błędzie.

Obsługa błędów w programie

Obsługa błędów w programie obejmuje następujące aspekty:

1. **Nieprawidłowe dane wejściowe:**
   - Sprawdzanie poprawności formatu wprowadzanych danych, np. liczbowych wartości (`getIntInput`, `GetPersonId`).
   - Obsługa błędów wprowadzania poprzez `cin.fail()`, `clear()` i ponowne pobieranie danych.
2. **Błędy plików:**
   - Weryfikacja poprawnego otwarcia plików (`database.txt`, `temp.txt`).
   - Wyświetlanie komunikatów w przypadku problemów z odczytem lub zapisem.
3. **Wyjątki w logice programu:**
   - Obsługa wyjątków (`runtime_error`) w funkcjach, takich jak:
     - Wyjście użytkownika (`handleExit`).
     - Nieprawidłowe wartości w danych wejściowych (np. konwersja `stoi`).
4. **Wyjście z operacji:**
   - Umożliwienie użytkownikowi przerwania operacji, np. przez wpisanie "wyjdź", co powoduje rzucenie wyjątku i powrót do menu.

5. **Brak wyników wyszukiwania:**
    - Informowanie użytkownika, gdy nie znaleziono rekordów spełniających kryteria wyszukiwania.

Ograniczenia programu od strony użytkownika

Brak walidacji poprawności danych wejściowych:

- **Imię, nazwisko, e-mail, adres:** Program nie sprawdza, czy wprowadzone dane są poprawne pod względem formatowania (np. e-mail zawierający "@" i domenę).

Brak obsługi błędów plików:

- Jeśli plik database.txt jest uszkodzony, nie istnieje lub zawiera nieprawidłowe dane, program może działać nieprawidłowo.

Brak zaawansowanego zarządzania błędami:

- Niektóre wyjątki są obsługiwane w sposób ogólny, np. wyrzucając użytkownika do menu głównego, bez szczegółowego komunikatu o błędzie.

Brak obsługi danych w złym formacie:

- Dane w pliku database.txt muszą być we właściwym formacie, aby program funkcjonował poprawnie

Przykłady działania programu

1. **Dodanie osoby do bazy danych**
   Użytkownik wprowadza dane osobowe oraz zainteresowania. Program zapisuje nową osobę w pliku database.txt.
   Przykład zapisu:
   1;Jan;Kowalski;ul. Lipowa 10, Warszawa;jan.kowalski@example.com;[Czytanie, Podróże, Programowanie]
2. **Usunięcie osoby z bazy danych**
   Użytkownik podaje ID osoby do usunięcia. Program wyświetla rekord i prosi o potwierdzenie. Po potwierdzeniu rekord zostaje usunięty, w przeciwnym razie operacja zostanie anulowana.

3. **Wyszukiwanie osoby po kryterium**
   Użytkownik wybiera kryterium wyszukiwania (np. nazwisko) i podaje wartość do wyszukania. Program wyświetla wszystkie rekordy spełniające kryterium.
   Przykład wyniku:
   1;Jan;Kowalski;ul. Lipowa 10, Warszawa;jan.kowalski@example.com;[Czytanie, Podróże, Programowanie]
4. **Edycja osoby**
   Użytkownik podaje ID osoby do edycji. Program wyświetla istniejące dane, a użytkownik może wprowadzić nowe wartości lub pozostawić pola puste, aby zachować stare dane.
   Przykład:
   Po edycji rekord w bazie zostaje zaktualizowany.
5. **Grupowanie według zainteresowań**
   Program przetwarza bazę danych i wyświetla listę osób przypisanych do każdego zainteresowania.
   Przykład wyniku:
   Czytanie:

   - Jan Kowalski
   - Marek Wiśniewski

   Programowanie:

   - Jan Kowalski
   - Anna Nowak

6. **Wyświetlenie całej bazy danych**
   Program wyświetla zawartość pliku database.txt w formie tabeli.

Autor: Mateusz Bonat