ZADANIE 3

“Prosta baza danych”

Dokumentacja

Robert Dudziński

OPIS

Program jest bazą danych studentów. Każdy student ma imię, nazwisko, datę urodzenia i numer indeksu. Program umożliwia wyświetlenie listy wszystkich studentów uporządkowanych alfabetycznie wg imienia lub wg malejącego wieku. Dodawanie i usuwanie studentów z bazy odbywa się tylko przez ten program, gdyż rekordy są przechowywane w plikach binarnych. Wewnątrz aplikacji studenci są przechowywani w liście jednokierunkowej o podwójnych wiązaniach. Poza programem dane są zapisane w 26 plikach (w każdym z nich znajdują się studenci, których nazwiska rozpoczynają się na tą samą literą). Jeden dodatkowy plik (mapowanie.txt w folderze programu) zawiera 26 wierszy, który każdy z nich jest ścieżką do kolejnych plików z danymi (od nazwisk na A do nazwisk na Z).

PODZIAŁ KODU NA PLIKI

main.c – plik zawiera w sobie pętlę główną programu. Posiada fukcje do wyświetlania wygenerowanych list, a także funkcje do dodawania nowych studentów i sprawdzania poprawności wprowadzonych danych.

common.c / common.h – pliki zawierają definicje struktur używanych w programie, a także podstawowe funkcje używane w innych plikach.

list.c / list.h – zawierają funkcje do obsługi listy – tworzenie listy, dodawanie i usuwanie elementów.

iodata.c / iodata.h – zawierają funkcje do wczytywania/zapisywania elementów listy z/do plików.

SPECYFIKACJA STRUKTUR (w pliku common.h)

```
typedef struct Date
{
   int day;
   int month;
   int year;
};
```

Struktura **Date** przechowuje informacje o pojedynczej dacie.

int **day** – dzień miesiąca, powinien wynosić od 1 do 28 lub 29 lub 30 lub 31 (w zależności od miesiąca)
int **month** – numer miesiąca, powiniem wynosić od 1 do 12
int **year** – rok, powinien wynosić od 1800 do 9999

```
typedef struct Student
{
   char firstName[FIRSTNAME_LENGTH];
   char secondName[SECONDNAME_LENGTH];
   Date dateOfBirth;
   int indexNumber;

   struct Student* nextName;
   struct Student* nextAge;
};
```

Struktura **Student** zawiera w sobie informacje o pojedynczym studencie. Każdy obiekt tej struktury jest pojedynczym elementem jednokierunkowej listy o podwójnych wiązaniach.

char **firstName**[] – tablica znaków przechowująca imię studenta, może mieć maksymalnie FIRSTNAME_LENGTH znaków (domyślnie to 24)

char **secondName**[] - tablica znaków przechowująca nazwisko studenta, może mieć maksymalnie SECONDNAME_LENGTH znaków (domyślnie to 24).

Date **dateOfBirth –** obiekt struktury **Date**, zawierający w sobie datę urodzenia studenta

int **indexNumber** – liczba całkowita oznaczająca numer indeksu studenta, powinna być nieujemna i indywidualna dla każdego studenta

struct Student* **nextName** – wzkaźnik na następny element listy posortowanej alfabetycznie wg nazwiska

struct Student* **nextAge** – wzkaźnik na następny element listy posortowanej wg malejącego wieku

SPECYFIKACJA FUNKCJI

W pliku **main.c**:

int **DisplayOptions** (Student** studentList, int numberOfStudents, int mode) – będąc na ekranie z listą, funkcja wyświetla użytkownikowi możliwe do wykonania czynności (powrót do menu głównego lub usunięcie wybranego studenta z listy). W przypadku wybrania opcji usunięcia studenta, funkcja prosi użytkownika o podanie numeru wybranego studenta na liście. numberOfStudents to liczba studentów używana do sprawdzenia numeru studenta do usunięcia. Argument mode to tryb listy (wg imienia lub wg wieku) – MODE_NAME lub MODE_AGE. Zwraca, która czynność została wybrana.

int **DisplayByName** (Student* studentList) – funkcja wyświetla wszystkich studentów z listy studentList posortowanych wg imienia lub informuje o braku studentów w bazie. Zwraca liczbę wyświetlonych studentów.

int **DisplayByAge** (Student* studentList) – funkcja wyświetla wszystkich studentów z listy studentList posortowanych wg wieku lub informuje o braku studentów w bazie. Zwraca liczbę wyświetlonych studentów.

int **CheckName** (const char* s) – funkcja służy do sprawdzenia poprawności w prowadzonego imienia lub nazwiska – sprawdza, czy każdy znak jest literą. Zwraca 1 w przypadku poprawności lub 0, gdy dane nie są poprawne.

int **LeapYear** (int y) – zwraca 0, gdy podany rok y jest zwykły lub 1, gdy podany rok jest przestępny.

int **CheckDate** (Date d) – zwraca poprawność danych w obiekcie d struktury Date. Uwzględnia warunki podane w specyfikacji elementów struktury Date. Zwraca 1 w przypadku poprawności lub 0, gdy dane nie są poprawne.

int **CheckIndexNumber** (Student* studentList, int n) – sprawdza, czy podany numer indeksu n jest poprawny. Najpierw sprawdza, czy jest liczbą nieujemną, a następnie sprawdza, czy przypadkiem taki numer indeksu nie jest przez kogoś innego wykorzystywany. Zwraca 1 w przypadku poprawności lub 0, gdy dane nie są poprawne.

void **AddStudent** (Student** studentList) – funkcja służy do dodawania nowego studenta do bazy. Prosi użytkownika o kolejne dane o studencie i sprawdza ich poprawność. Jeżeli nie udało się wczytać, którejś informacji lub któraś z nich była niepoprawna informuje o błędzie i przerywa swoje dalsze działanie. Jeżeli dane są poprawne to przekazuje je do zapisu. Jeżeli wszystko się udało zwraca komunikat o pomyślności.

void **DisplayMenu** () - wyświetla główne menu programu.

void **ListByName** (Student** studentList) – zawiera wewnętrzną pętlę do obsługi listy posortowanej wg imienia.

void **ListByAge** (Student** studentList) - zawiera wewnętrzną pętlę do obsługi listy posortowanej wg wieku.

int **main** () - główna funkcja programu. W pierwszej kolejności tworzy pustą listę i wczytuje dane z plików i zapisuje je do tej listy. Następnie uruchamiana jest główna pętla programu. W przypadku chęci wyjścia użytkownika z programu usuwa listę z pamięci i kończy program.

W pliku **common.c**:

int **CompareStrings** (char* s1, char* s2) – funkcja porównuje dwa ciągi znaków i stwierdza, który z nich jest alfabetycznie pierwszy (nie zwraca uwagi na wielkość znaków). Zwraca 1, gdy s1 jest alfabetycznie przed s2 lub 0 w innych przypadkach.

int **CompareDates** (Date d1, Date d2) – porównuje dwie daty. Zwraca 0, gdy d1 jest starszy od d2 lub 1 w innych przypadkach.

void **ClearInputBuffer** () - czyści bufor wejścia. Funkcja jest używana w przypadku, gdy użytkownik wprowadził błędne dane i trzeba się ich pozbyć z buforu.

void **DisplayRecord** (Student* student, int it) – funkcja wyświetla informację o pojedynczym studencie. Jeżeli it nie jest równe 0 to funkcja wyświetla przed rekordem liczbę porządkową równą it.

void **DisplayHeader** () - funkcja czyści ekran i wyświetla na górze tytuł i autora programu.

W pliku **list.c**:

void **CreateEmptyList** (Student** studentList) – funkcja tworzy pustą listę. Ponieważ lista jest o podwójnych wiązaniach to pierwszy element nie przechowuje informacji o żadnym studencie. Pierwszy element ma jedynie wzkaźniki do pierwszego elementu listy wg imienia oraz do pierwszego elementu listy wg wieku.

void **ClearStudentList** (Student** studentList) – całkowicie czyści podaną listę.

void **InsertToNameList** (Student** studentList, Student* temp, Student* optLastStudent) – wstawia nowego studenta temp do listy wg imienia. Argument optLastStudent jest opcjonalny (nie używany powinien mieć wartość NULL). Służy on do znacznie szybszego wstawiania studentów podczas wczytywania danych z pliku. Więcej informacji w dalszej części.

void **InsertToAgeList** (Student** studentList, Student* temp) – wstawia nowego studenta temp do listy wg wieku.

Student* **AddStudentToList** (Student** studentList, Student student, Student* optLastStudent) – wstawia nowego studenta student do listy studentList. Dodaje go do listy wg imienia i wg wieku. Argument optLastStudent jest opcjonalny (nie używany powinien być NULLem) i jest dla funkcji InsertToNameList(). Po wstawieniu studenta do listy, rekord jest przekazany do zapisu do odpowiedniego pliku. Zwraca adres nowo dodanego rekordu.

void **RemoveFromList** (Student** studentList, int number, int mode) – usuwa studenta o liczbie porządkowej number na liście wg imienia lub wg wieku (w zależności od argumentu mode – MODE_NAME lub MODE_AGE). Przed samym usunięciem, pyta się użytkownika czy na pewno chce usunąć dany rekord. Po pomyślnym usunięciu z pamięci, aktualizuje plik, w którym znajdował się usunięty student.

W pliku **iodata.c**:

void **SaveStudent** (Student* student, const char* fileName) – dopisuje pojedynczego studenta student do odpowiedniego pliku. Funkcja używana podczas dodawania nowego studenta.

void **SaveStudents** (Student* studentList, const char* fileName, char whichDataFile) – aktualizuje plik, zawierający studentów o nazwiskach zaczynających się od litery whichDataFile. Funkcja używana podczas usuwania studenta z bazy.

int **LoadStudents** (Student** studentList, const char* fileName) – wczytuje studentów z plików do pamięci. Najpierw otwiera główny plik z mapowaniem do innych plików i po kolei wczytuje kolejne rekordy. Zwraca liczbę wczytanych studentów lub -1, gdy nie udało się wczytać wszystkich ścieżek do plików z danymi.

DODAWANIE NOWEGO STUDENTA

Aby dodać nowego studenta należy wybrać numer 3 w menu głównym. Program będzie prosił o kolejne informacje o studencie, jednocześnie sprawdza ich poprawność, aby były zgodne ze specyfikacjami struktur.

Jeżeli nie udało się wczytać danej lub była ona niepoprawna, będzie wypisany stosowny komunikat.

Jeżeli wszystkie podane dane są prawidłowe, rekord zostaje dodany do listy, a następnie dopisany na koniec odpowiedniego pliku. Na koniec pojawia się komunikat o powodzeniu.

WCZYTYWANIE DANYCH

Na początku swojego działania, program wczytuje rekordy studentów z pliku. Najpierw otwiera plik (mapowanie.txt) zawierający ścieżki do konkretnych plików binarnych z danymi. Z każdego takiego pliku są wczytywani po kolei studenci i wstawiani do listy.

Ponieważ kolejni studenci z listy są często[(1)] uporządkowani alfabetycznie, a listę przegląda się od początku to każdemu kolejnemu wstawieniu towarzyszyła by większa ilość iteracji, dlatego w funkcjach AddStudentToList() oraz InsertToNameList() jest dodatkowy, opcjonalny argument optLastStudent, za pomocą którego można podać adres ostatniego elementu i pierwsze porównanie wykonać z nim (dotyczy to oczywiście tylko listy wg imion).

Dzięki temu przy początkowym wczytywaniu danych, liczba iteracji przy wstawianiu rekordu do listy wg imienia może być dużo mniejsza[(1)].

[(1)] gdy użytkownik jedynie dodaje nowe rekordy to są one zapisywane w kolejności dodania, ale jeżeli usunie jakiegoś studenta to plik jest przepisywany, a rekordy tam się znajdujące są od tego momentu w kolejności alfabetycznej (do dodania nowego studenta na koniec pliku).

WYŚWIETLANIE LIST I USUWANIE REKORDU

Aby wyświetlić listę należy wybrać 1 lub 2 w menu głównym. Pojawi się wtedy lista wszystkich studentów uporządkowana wg wybranej formy (wg imienia lub wg wieku). Jeden rekord jest wyświetlany w podanej postaci:

[NumerNaLiście]. [Nazwisko] [Imie] [DataUrodzenia DD-MM-RRRR] [NrIndeksu]

Pod listą znajduje się informacja o możliwych czynnościach.

Po wybraniu opcji 0, użytkownik wróci do menu głównego.

Po wybraniu opcji 1, użytkownik zostanie poproszony o podanie pozycji z listy, którą chce usunąć z bazy danych. Jeżeli nie udało się wczytać numeru lub nie należy on do żadnej pozycji z listy, z powrotem wyświetla się lista. W przeciwnym wypadku, gdy podany numer zgadza się z którymś elementem na liście, program pyta się czy na pewno chce usunąć dany rekord (dla pewności wyświetla go na ekranie).

Jeżeli użytkownik wpisze 1, rekord zostanie usunięty z pamięci, a następnie cały plik, w którym znajdował się rekord zostanie wyczyszczony i od nowa zapisany odpowiednimi rekordami.

SCHEMAT BLOKOWY DODAWANIA STUDENTA DO BAZY

Elementy struktury Student:

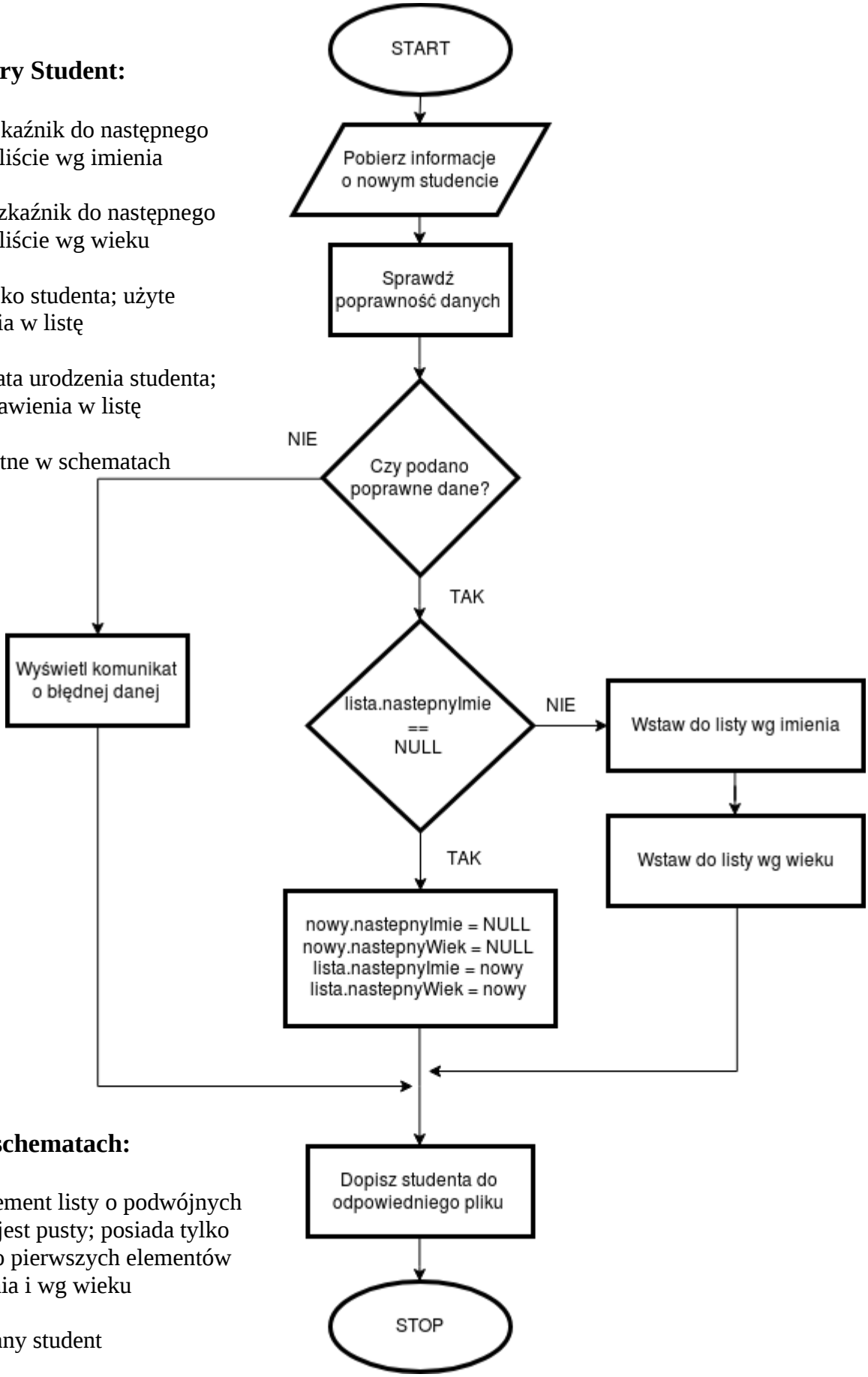
- nastepnyImie – wzkaźnik do następnego elementu na liście wg imienia

- nastepnyWiek – wzkaźnik do następnego elementu na liście wg wieku

- nazwisko – nazwisko studenta; użyte do wstawiania w listę

- dataUrodzenia – data urodzenia studenta; użyte do wstawienia w listę

- inne dane – nieistotne w schematach



Obiekty użyte w schematach:

- lista – pierwszy element listy o podwójnych wiązaniach; jest pusty; posiada tylko wzkaźniki do pierwszych elementów list wg imienia i wg wieku

- nowy – nowo dodany student

Powyższe elementy są adresami obiektów struktury Student.

Dla uproszczenia schematów:

- do zmiennych każdego elementu (nawet adresów do obiektów) odwołuje się jak do zmiennych zwykłych obiektów struktur (znak ‘.’ zamiast ‘→’)

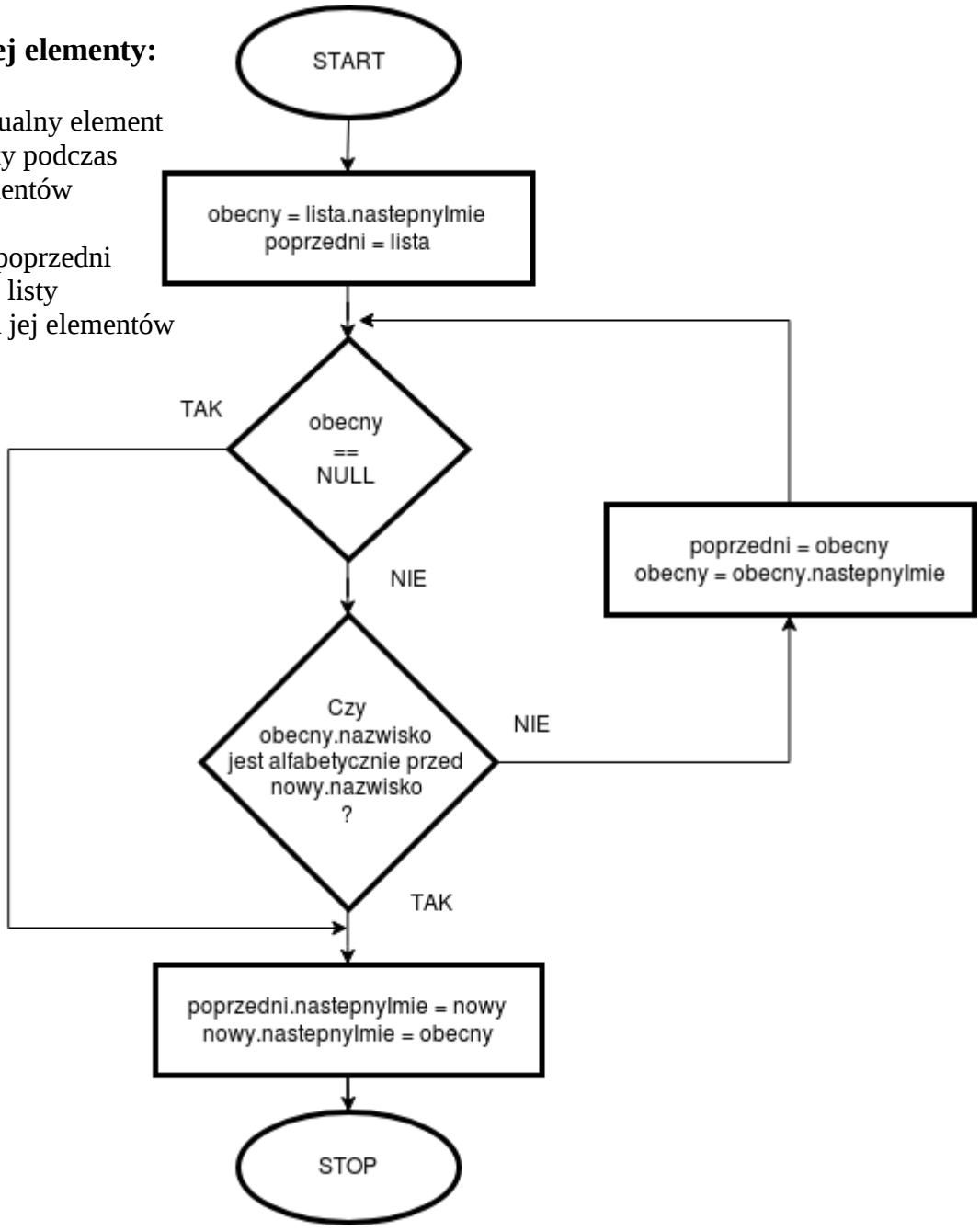
- przypisując adres elementu do wzkaźników (np. do nastepnyImie, nastepnyWiek) podaje się po prostu ten element, nie zwracając uwagi czy jest on obiektem czy adresem obiektu

SCHEMAT WSTAWIANIA DO LISTY WG IMIENIA

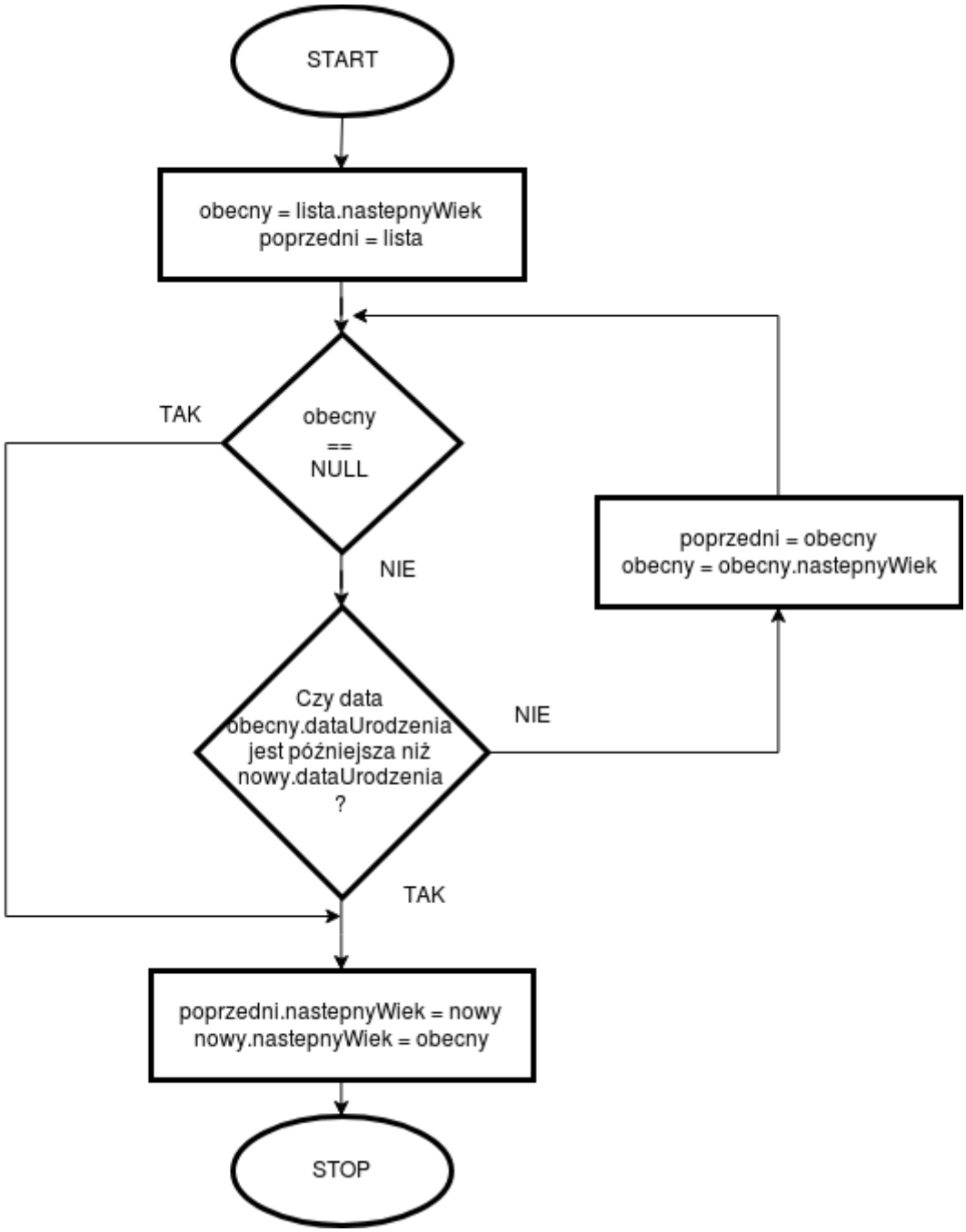
Nieopisane wcześniej elementy:

- obecny – zawiera aktualny element (jego adres) listy podczas iteracji jej elementów

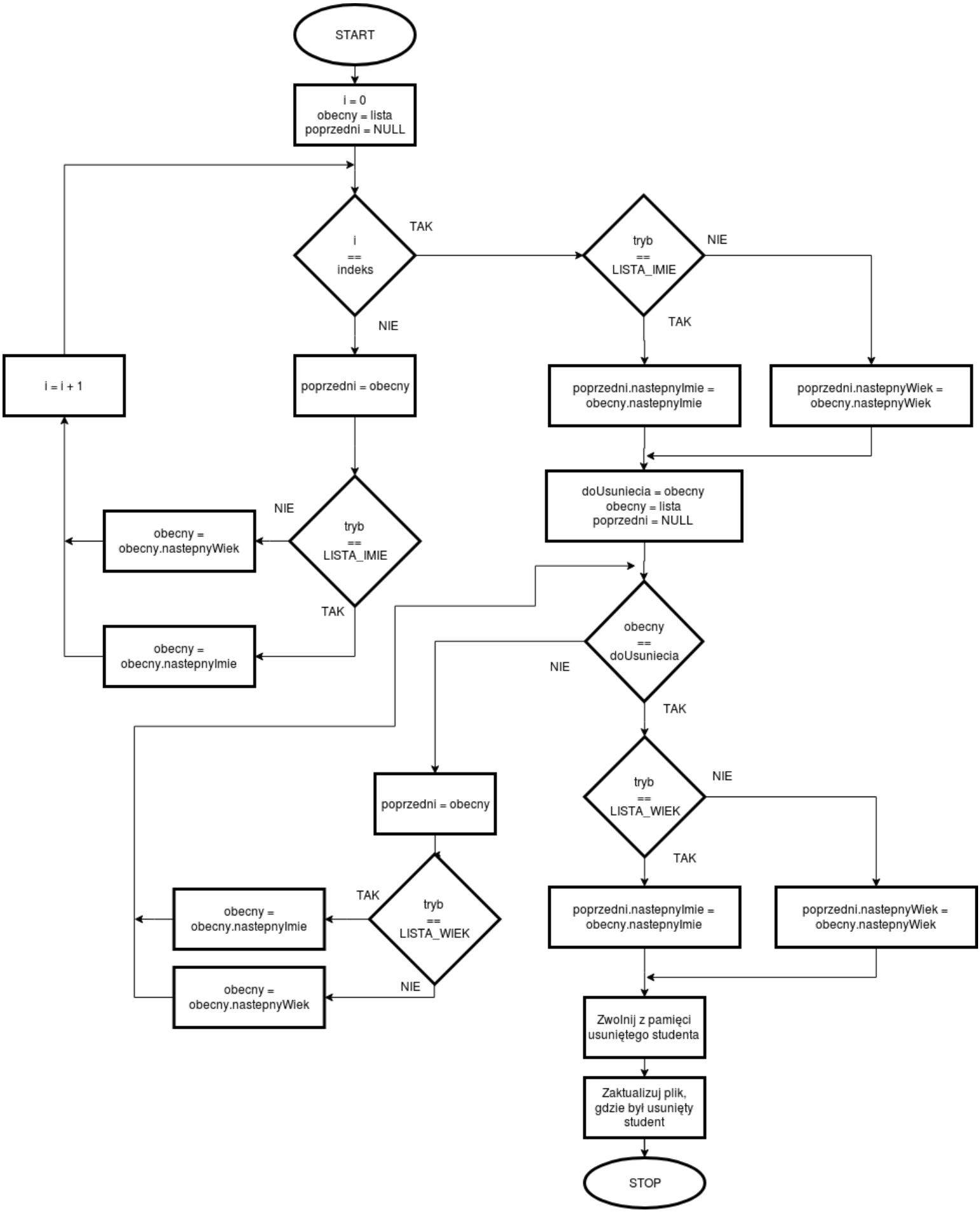
- poprzedni – zawiera poprzedni element (adres) listy podczas iteracji jej elementów



SCHEMAT WSTAWIANIA DO LISTY WG WIEKU



SCHEMAT USUWANIA STUDENTA Z BAZY



Opis schematu usuwania z bazy:

Usuwa studenta, który ma pozycję na liście równą zmiennej indeks. Pozycja na dwóch listach może być różna, dlatego zmienna tryb przechowuje informację wg której listy „odliczać” rekordy.

Nieopisane wcześniej elementy:

- i – iterator służący do sprawdzania, którym w kolejności na liście jest aktualny element

- indeks – zawiera informację, który z kolei element jest do usunięcia

- tryb – może przyjmować wartości: LISTA_WIEK lub LISTA_IMIE; jest to informacja wg której listy liczyć numer elementu

- doUsuniecia – zawiera adres elementu po usunięciu go z pierwszej listy; za pomocą jego można odnaleźć element na drugiej liście