Система за управление на БД със самолети за авиобаза

от: Росен Витлянов

Проектът предоставя възможност за създаване, редактиране, преглед, търсене и изтриване на голям брой записи от семпла база данни за самолети. Програмата се поддържа от REPL за осъществяване на обозначените по-горе действия. Информацията за всички самолети се съхранява чрез текстов файл в постоянната памет при изход от програмата.

1. Архитектура

1.1 Plane

Класът “Plane” описва запис на един самолет от базата данни. Той поддържа следните *“private”* член данни:

- **id** - Уникален идентификационен номер на самолет от тип *“unsigned long long”*;
- **name** - име на самолет от тип *std::string*;
- **type** - тип на самолет от тип *std::string*;
- **flights** - брой извършени полети от тип *“unsigned long long”*.

Класът използва методите по долу, за поддръжка на данните:

- Конструктор **Plane(unsigned long long id, std::string Plane, std::string Type, unsigned long long Flights)** създава запис на самолет по дадените атрибути: Id - идентификатор, Plane - име на самолет, Type - тип на самолет, Flights - брой полети;

- **unsigned long long getID() const** позволява безопасен достъп на външни функции и класове до идентификатора *Id* на самолет;

- **void setID(unsigned long long id)** - “сетър” който служи за контрол на подадените до *“Id”* стойности. Методът модифицира невалидните данни във валидни без прекратяване на програмата;

- **void setFlights(unsigned long long flights)** - “сетър” който служи за контрол на подадените до *“flights”* стойности. Методът модифицира невалидните данни във валидни без прекратяване на програмата;

- **void setName(const std::string& name)** - “сетър” който служи за контрол на подадените до *“name”* стойности. Методът модифицира невалидните данни във валидни без прекратяване на програмата;

- **void setType(const std::string& type)** - “сетър” който служи за контрол на подадените до *“type”* стойности. Методът модифицира невалидните данни във валидни без прекратяване на програмата;

- **void printPlane()** отпечатва запис в конзолата във формат *id name type flights*;

- **friend std::ostream& operator<< (std::ostream& os, const Plane& plane)** предефенира оператора за изход **<<**;

1.2 PlaneDatabase

Класът *PlaneDatabase* служи за управление на базата данни. Той поддържа следните *“private”* член данни и методи:

- **std::vector<Plane> db** - динамичен масив от самолети;

- **bool isOptimized** - булев идентификатор оказващ дали вектора е оптимизиран или не.

- **void heapify(unsigned long long n, unsigned long long i)** - помощна рекурсивна функция на сортиращият алгоритъм HeapSort. Функцията подрежда елементите на вектора 3 по 3, така че най-големият елемент да отиде в началото.

- **long long binSearch(unsigned long long left, unsigned long long right, unsigned long long id)** - тази функция рекурсивно намира елемент в сортираният вектора *“db”*. Изпълнява се при вдигнат флаг *“isOptimized”.*

“*public”* методите на класа *PlaneDatabase* са:

- **PlaneDatabase()** - конструктор по подразбиране. Инициализира празен вектор и дава нулева стойност на *“isOptimized”;*

- **PlaneDatabase(std::string& fileName)** - този конструктор приема име на файл. Ако такъв файл съществува, създава самолет за всеки валиден запис във файла. Ако не съществува - инициализира празен вектор и дава нулева стойност на *“isOptimized”;*

- **unsigned long long search(unsigned long long id)** - приема стойност на идентификатор. Ако *isOptimized* е 0 търси идентификатора със сложност *O(n).* При вдигнат флаг *isOptimized* , вика функцията **binSearch;**

- **void cratePlane(unsigned long long id, std::string name, std::string Type, unsigned long long Flights)** създава нов самолет в края на вектора, ако даденият идентификатор “*Id”* не съществува базата данни.

- **void deletePlane(unsigned long long id)** - премахва самолет от базата данни;

- **void updatePlane(unsigned long long id, std::string attribute, std::string newValue)** - променя даденият “*attribute”* с подадената стойност в “*newValue”* на идентификатора. Не прави промени при некоректно подадени данни;

- **void optimize()** - сортира елементите във вектора с *HeapSort* алгоритъм;

- **void show(unsigned long long offset, unsigned long long limit);** - извежда на стандартният изход *limit* на брой самолета считани от елементът *offset;*

- **unsigned long long getSize() const** - връща големината на вектора *db*;

- **void saveToFile(std::string& fileName)** - запазва вектора db във файл с име *“fileName”;*

- **void printElement(unsigned long long index)** - извежда на стандартния изход елементът с индекс *“index”;*

1.3 Menu

Класът “Menu” служи за комуникация между потребителя и обект от тип “*PlaneDatabase”.* Той съдържа статични функции, които приемат входните данни, необходими за изпълнението на командите.

- **static void create(PlaneDatabase& db)** - приема от стандартният вход *“Id”, “name”, “type”* и *“flights”.* Подава получените данни на функцията *“createPlane”.*

- **static void deletePlane(PlaneDatabase& db)** - приема от стандартният вход *“Id”.* Подава получените данни на функцията *“deletePlane”.*

- **static void update(PlaneDatabase& db)** - приема от стандартният вход *“Id”, “attribute”* и *“value”.* Подава получените данни на функцията *“updatePlane”.*

- **static void show(PlaneDatabase& db)** - приема от стандартният вход *“offset”* и *“limit”.* Подава получените данни на функцията *“show”.*

- **static void optimize(PlaneDatabase& db)** - пуска функцията за оптимизация.

- **static void search(PlaneDatabase& db)** - приема от стандартният вход *“Id”.* Отпечатва на стандартният изход данните от записа с идентификатор *“Id”,* ако такъв съществува.

- **static void save(PlaneDatabase& db, std::string fileName)** - вика функцията за запазване на базата данни във файл.

2. Четене на входните данни

Потребителят въвежда команди. При невалидна команда се извежда съобщение с възможните команди.

Командите които могат да бъдат въведени са:

- **create *Id Plane Type Flights*** - създава запис за самолет с въведените от потребителя данни за самолет;

- **delete *Id*** - изтрива запис за самолет с въведения от потребителя Id;

- **update *Id attribute value*** - обновява записаната информация за въведения от потребителя атрибут *attribute* (възможната стойност е едно от следните: *Id* или *Plane* или *Type* или *Flights*) за запис за самолет с въведения от потребителя *Id*;

- **show *offset limit*** - отпечатва на стандартния изход *limit* на брой пълни записи за самолети започвайки от записа на разстояние *offset* от първия запис;

- **optimize** - оптимизира алгоритъма за търсене;

- **search *Id*** - отпечатва на стандартния изход пълния запис за самолетът с въведения от потребителя *Id*

- **exit** запазва промените в текстов файл *“Planes.db”* и приключва изпълнението на програмата.

Поради потенциално големият обем данни, програмата общува 2 пъти с файла в постоянната памет - веднъж при стартиране и веднъж при приключване. Всяко нежелано поведение е нормализирано, за да не се налага предсрочно прекратяване на програмата.

3. Схема на проекта