Programátorská dokumentace

Cyklisté

Martin Halfar HAL0160

Tento dokument obsahuje stručný popis tříd, struktur a metod použitých při programování. Autorem všech uvedených metod je výše uvedený autor semestrální práce.

Obsah

1. Stručný popis programu

Tento program slouží k zobrazení a exportu statistik cyklistů.

Jako zdroj dat je program schopen využít jakýkoli textový soubor (nejlépe však formát .csv), který obsahuje záznamy tréninků na jednotlivých řádcích, kde je mezi jednotlivými daty oddělovací znak ‚;'. V případě, že vstupní soubor obsahuje neplatná data (jiné oddělovací znaky, vynechané hodnoty, neplatné hodnoty), tak jsou tyto záznamy ignorovány.

Základním výstupem programu je konzole, kdy je možné specifikovat výstupní soubor (.html), kam budou zpracované statistiky uloženy za použití příslušných HTML tagů – vytvoří se dvě tabulky se statistikami.

V programu je využita třída *HeapArrayInterface*, což je jednoduché rozhraní mezi uživatelem a dynamicky alokovaným polem. Toto rozhraní obsahuje sadu metod pro práci s obsahem interního pole a je ošetřeno proti zásahům mimo alokované prostory v paměťi. Jedná se o velmi zjednodušenou náhradu knihovny *vector*.

2. Struktury a třídy (včetně příslušných metod)

2.1 HeapArrayInterface

Tato třída slouží jako prostředník, mezi dynamicky alokovaným polem (dále jen ‚pole') libovolného typu (dále jen ‚T'), libovolné velikosti (dále jen ‚velikost') a programátorem. Třída obsahuje tři proměnné, a to:

a) Pointer na pole (T*)
b) Velikost pole (int)
c) Pseudo-iterátor (int)

2.1.1 Konstruktor (void)

Tento konstruktor vytvoří novou instanci třídy s velikostí 0. V tomto případě se nové pole nealokuje a pro rozšíření pole je nutné použít příkaz *expand()*.

2.1.2 Konstruktor (int)

Tento konstruktor vytvoří novou instanci třídy s nenulovou velikostí pole. První parametr specifikuje velikost pole. V případě, že je velikost menší než 1, tak je postup obdobný, jako u předchozího konstruktoru. V opačném případě se velikost nastaví na specifikovanou velikost a alokuje se dle ní pak pole.

2.1.3 [] overload (int)

Overload na operátor []. Pokud je index v rámci mezí pole, vrátí odkaz na prvek pole na indexu. V případě, že má pole velikost nula, rozšíří jej na velikost 1. Pokud není splněna první podmínka, vrací odkaz na první prvek v poli.

Funkce vrací odkaz na prvek v poli na indexu.

2.1.4 deallocate (void)

Tato funkce je privátní (lze ji volat pouze uvnitř třídy). Pokud je velikost pole větší než 0, tak uvolní paměť pole pomocí příkazu ‚delete[]'.

2.1.5 in_bounds (int)

Tato funkce je privátní (lze ji volat pouze uvnitř třídy). Vrací typ bool podle toho, zda-li je parametr v mezích pole.

2.1.6 resize (int, bool)

Tato funkce se stará o změnu velikosti pole. Je možné použít velikost vetší nebo rovné nule. Prvním parametrem je velikost, na kterou bude pole nastaveno. Druhým parametrem je možné specifikovat, zda-li chceme hodnoty v poli ponechat.

V případě, že je velikost 0, tak se pole dealokuje a velikost nastaví na 0. V opačném případě se alokuje nové pole a pokud chceme stávající hodnoty pole ponechat, pak jsou nakopírovány do nového pole. Dále se dealokuje pole staré a adresa nového pole se přesune do vnitřního pointeru. Pointer na pole nové se nastaví na NULL.

Funkce vrací typ bool, dle toho, zda-li byla změna velikosti provedena nebo ne.

2.1.7 purge (void)

Tato funkce se stará o dealokaci pole a nastavené velikosti na nulu. Je nutné tuto funkci zavolat vždy na konci práce s instancí třídy. Jinak zůstane pole stále alokované a nepřístupné.

2.1.8 set (int, T)

Tato funkce nastavuje prvek pole na určitém indexu na specifikovanou hodnotu. Prvním parametrem je index prvku v poli. Pokud je index v rozsahu pole, je hodnota druhého parametru vložena do pole na pozici indexu.

Funkce vrací typ bool, dle toho, zda-li byl index v mezích pole.

2.1.9 size (void)

Tato funkce vrací velikost pole.

2.1.10 iter_reset (void)

Tato funkce nastaví pseudo-iterátor na pozici 0.

2.1.11 iter_set (int)

Funkce nastavuje pozici pseudo-iterátoru na hodnotu parametru. V případě, že je nová hodnota v mezích pole, je hodnota nastavena.

Funkce vrací typ bool, dle toho, zda-li byla pozice v mezích pole.

2.1.12 iter_current (ITER_OPERATION)

Funkce vrací odkaz na položku pole, kde je jako index použita hodnota pseudo-iterátoru. Parametr slouží ke specifikaci operace, co provést s pseudo-iterátorem po vrácení odkazu. Je možné provést 3 možné operace:

a) *INCREASE* - Zvýšit hodnotu pseudo-iterátoru o 1
b) *IGNORE* - Nedělat nic
c) *DECREASE* - Snížit hodnotu pseudo-iterátoru o 1

V případě, že hodnota pseudo-iterátoru přesáhne meze pole, tak se jeho hodnota nastaví na příslušnou mez pole (dle toho, jestli jde o snížení či zvýšení hodnoty).

Funkce vrací odkaz na položku pole.

2.1.13 iter_current (void)

Alternativa funkce *iter_current()* s operací *IGNORE*.

2.1.14 iter_at (void)

Funkce vrací hodnotu pseudo-iterátoru.

2.1.15 enum ITER_OPERATION

Tento enumerátor se využívá společně s funkcí *iter_current()* a určuje operaci, která bude s pseudo-iterátorem provedena. Obsahuje hodnoty:

a) *DECREASE* -1
b) *IGNORE* 0
c) *INCREASE* 1

2.2 Training

Tato stuktura slouží k uchování jednoho tréninku – ujeté vzdálenosti a stráveného času. Obsahuje tyto proměnné:

a) Vzálenost (double)
b) Čas (double)

2.2.1 Konstruktor (void)

Tento konstruktor vytváří novou instanci struktury s nulovými hodnotami.

2.2.2 Konstruktor (double, double)

Tento konstruktor vytváří novou instanci struktury a nastavuje hodnoty podle parametrů. Prvním parametrem je specifikována ujetá vzdálenost a druhý slouží k určení stráveného času tréninkem.

2.3 Cyclist

Tato struktura slouží k uchování informací o cyklistovi a jeho trénincích. Ve struktuře je uloženo jméno cyklisty (v proměnné typu string) a *HeapArrayInterface* s tréninky. Kromě těchto dvou hlavních součástí, struktura také obsahuje čtyři pomocné proměnné, pro zvýšení výkonu při výpočtů průměrných a celkových vzdáleností a časů.

Pro minimalizaci výpočtů, nutných k určení celkových a průměrných hodnot, struktura uchovává velikost *HeapArrayInterface* tréninků posledního výpočtu celkové hodnoty (zvlášť pro celkovou dobu a celkovou vzdálenost). Tím pádem je možné využít již vypočtené hodnoty, pokud je výpočty opakují se stejným obsahem *HeapArrayInterface* tréninků.

Obsahuje tyto proměnné:

a) Poslední kontrola vzdálenosti (int)
b) Poslední kontrola času (int)
c) Celková vzdálenost (double)
d) Celkový čas (double)
e) Jméno cyklisty (string)
f) Tréninky cyklisty (*HeapArrayInterface*<*Training*>)

2.3.1 Konstruktor (void)

Tento konstruktor vytváří novou instanci struktury. Nastaví pouze pomocné proměnné – jméno a další je nutné nastavit ručně.

2.3.2 Konstruktor (string)

Tento konstruktor vytváří novou instanci struktury, nastaví jméno cyklisty na hodnotu parametru a nastaví také pomocné proměnné na defaultní hodnoty.

2.3.3 preset (void)

Tato privátní funkce nastaví všechny pomocné proměnné na jejich defaultní hodnoty.

2.3.4 getTotalDistance (void)

Tato funkce vrací celkovou ujetou vzdálenost cyklisty během tréninku. Nejdříve funkce zkontroluje, zda-li už výpočet nebyl proveden. V případě že ano, vrátí stávající

uložený výsledek. Pokud ne, nebo bude *HeapArrayInterface* tréninků mít jinou velikost než při minulém výpočtu, tak funkce projde všechny cyklistovy tréninky a vypočte celkovou vzdálenost. Tu pak uloží do pomocné proměnné spolu s velikostí *HeapArrayInterface* tréninků a vrátí vypočtenou hodnotu.

2.3.5 getTotalDuration (void)

Tato funkce vrací celkovou dobu strávenou tréninky. Funkce je identická s funkcí *getTotalDistance()* s tím rozdílem, že počítá čas.

2.3.6 getAverageDistance (void)

Tato funkce vrací průměrnou vzdálenost, kterou cyklista ujel během svých tréninků. Díky pomocným proměnným je možné výpočet provést pomocí již vypočtených hodnot. Tato funkce volá funkci *getTotalDistance()* a dělí ji velikostí *HeapArrayInterface* tréninků, která byla zaznamenána při posledním výpočtu celkové vzdálenosti. V případě, že cyklista nemá žádné tréninky v záznamu, je vrácena nula.

2.3.7 getAverageDuration (void)

Tato funkce vrací průměrnou dobu strávenou tréninky. Funkce je identická s funkcí *getAverageDistance()* s tím rozdílem, že počítá čas.

3. Použité metody (funkce)

3.1 sortQuick

Tato funkce slouží k třídění typu ‚quick sort'. Tento způsob pracuje na principu ‚pivot pointu' a rozdělování hodnot na menší skupiny. Jedná se o klasický, známý příklad řešení tohoto typu třídění.

3.2 sortSwap

Tato funkce je komplementární k funkci *sortQuick().* Zamění objekty (proměnné / struktury) na specifikovaných indexech třídy *HeapArrayInterface*.

3.3 outputHtml

Tato fukce se stará o výpis obsahu *HeapArrayInterface* cyklistů do specifikovaného souboru. K zápisu se používají příznaky ‚out' a ‚trunc'. Do objektu fstream jsou pak postupně posílány HTML tagy a samotné hodnoty. Ve výsledku jsou pak vytvořeny dvě tabulky – jedna obsahující cyklisty a jejich základní statistiku, a druhá s podrobnými daty o jednotlivých trénincích cyklistů.

Funkce vrací typ bool v závislosti na tom, zda-li byl soubor úspěšně otevřen.

3.4 readFile

Tato funkce se stará o načtení všech hodnot z textového souboru (.csv) do HeapArrayInterface cyklistů (dále jen pole cyklistů). Funkce využívá funkcionalitu pseudo-iterátoru třídy *HeapArrayInterface*. Nejdříve se nastaví velikost pole cyklistů na počet daný funkcí *sumOfCyclists()*. K tomu se navíc vytvoří *HeapArrayInterface* string (dále jen pomocného pole string) o velikosti 3 jako mezipaměť při čtení řádku. Funkce otevře soubor s příznakem ‚in' a čte postupně všechny záznamy / řádky.

Funkce načte řádek do pomocné proměnné. Dále pak resetuje pseudo-iterátor pomocného pole string. Dále pak rozdělí řádek (pomocí znaku ‚;') na jednotlivá slova a ty pomocí iter_current() uloží do pomocného pole string. Pak provede převod string na double (vzdálenost a čas tréninku) a uloží je do příslušných pomocných proměnných. Dále zkontroluje, zda-li není v datech chyba. Pokud ano, tak vypíše oznámení o chybě a pokračuje dalším řádkem. Pokud ne, pokračuje průchodem pole cyklistů a kontroluje, zda-li se takový cyklista v poli nachází. Pokud ano, tak přidá do jeho tréninků právě načtený záznam a nastaví flag ‚exists', aby nedošlo k vytvoření nového cyklisty. V případě, že cyklista s tímto jménem neexistuje, tak vytvoří v poli cyklistů na pozici pseudo-iterátoru nového cyklistu a rozšíří jeho *HeapArrayInterface* tréninků na velikost danou funkcí *sumOfTrainings()*. Pak zapíše do pozice pseudo-iterátoru nový trénink a pseudo-iterátor pole cyklistů posune na další pozici. Na konci souboru čtený soubor uzavře a vymaže pomocné pole stringů.

V dalším kroku pak projde všechny cyklisty a v případě, že nemá žádné platné tréninky, tak ho vymaže z pole cyklistů. Pokud má více než 0 tréninků tak projde pole tréninků cyklisty a pokud narazí na nulovou vzdálenost nebo čas tak trénink vymaže.

Funkce vrací typ bool v závislosti na tom, zda-li byl soubor úspěšně otevřen.

3.5 sumOfCyclists

Tato funkce vrací počet unikátních cyklistů v textovém souboru (.csv). Funkce otevře soubor s příznakem ‚in' a přečte postupně všechny jeho řádky. Všechna již přečtená jména uchovává v proměnné typu string. Při přečtení jména porovná jméno s již přečtenými jmény. V případě, že se zde jméno nenachází, tak ho do přečtených jmen přidá a přičte k výstupní proměnné jedničku. V opačném případě řádek ignoruje a pokračuje dále.

Funkce vrací typ unsigned int, podle počtu unikátních cyklistů.

3.6 sumOfTrainings

Tato funkce vrací počet tréninků v textovém souboru (.csv), které patří určenému cyklistovi. Funkce otevře soubor s příznakem ‚in' a přečte postupně všechny řádky. Jméno každého záznamu se pak porovná se jménem cyklisty a v případě shody se přičte k výstupní proměnné jednička. V opačném případě záznam ignoruje a pokračuje dále.

Funkce vrací typ unsigned int, podle počtu tréninků cyklisty.

4. Main

Toto je speciální funkce a volá se automaticky po inicializaci statických objektů po spuštění programu. V této funkci se nejdříve vytvoří nová instance třídy *HeapArrayInterface* typu *Cyclist* (dále jen databáze), do které budou ukládány všechny načtené hodnoty ze souboru (.csv).

Nejdříve program od uživatele načte cestu k souboru a zpracuje ho. V případě, že je soubor nenalezen, tak se program s chybou ukončí. V opačném případě se databáze setřídí pomocí funkce *quickSort().* Dále program načte od uživatele cestu k souboru pro výpis. Pokud uživatel nezadá nic, výpis nebude proveden. Pokud však uživatel zadá cestu k souboru, ale výpis nebude úspěšně proveden, zobrazí se chybové hlášení.

Dále program provede výpis statistiky cyklistů.

Následuje vymazání databáze a program se ukončí po stisku klávesy enter.