Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Maraton

autor	Dorian Barański
prowadzący	dr inż. Adam Gudyś
rok akademicki	2019/2020
kierunek	informatyka
rodzaj studiów	SSI
semestr	5
termin laboratorium / ćwiczeń	środa, 13:45 – 15:15
grupa	6
sekcja	11
termin oddania sprawozdania	2020-01-27
data oddania sprawozdania	2020-01-22

1. Treść zadania

Napisać program, który wczytuje z plików informacje o maratonach. Maratończycy biorą udział w różnych zawodach. Po zakończeniu tworzona jest lista rankingowa w następującej postaci: w pierwszej linii jest nazwa maratonu, w drugiej data (w formacie: rrrr-mm-dd), w kolejnych są wyniki zawodników: kolejność na mecie, nazwisko, nr zawodnika w zawodach, czas (w formacie: gg:mm:ss).

Przykładowy plik z danymi:
Maraton bostonski
2012-09-04
2, Jaworek, 1432, 04:34:12
1, Bukowy, 434, 03:54:45
3, Krol, 243, 04:37:32

Plików z wynikami zawodów może być dowolna liczba. Po wykonaniu programu uzyskujemy pliki wynikowe zawierające wyniki zawodników. Każdy plik zawiera wyniki tylko jednego maratończyka. Nazwa pliku jest tożsama z nazwiskiem maratończyka. W pliku tym podane jest nazwisko i wyniki, które uzyskał w zawodach. Wyniki te są przedstawione w następujący sposób: <data><nazwa maratonu><czas>. Wyniki są posortowane wg daty.

Przykładowy plik dla Jaworka:
Jaworek
2011-05-03 Maraton Swiateczny 04:01:43
2011-09-14 Bieg Rzeznika 04:13:32
2012-05-03 Do przodu! 04:02:43

Program uruchamiany jest z linii poleceń z wykorzystaniem następującego przełącznika: -i pliki wejściowe z protokołami zawodów

2. Analiza zadania

Zagadnienie przedstawia problem wczytywania danych z jednego lub więcej plików w odpowiedni sposób oraz zapisywania danych do nowo utworzonych plików.

2.1 Struktury danych

W programie wykorzystano listę jednokierunkową przechowującą informacje o maratonach. Elementy listy posiadają wskaźniki na następnika oraz na drzewo binarne zawodników, do którego trafiają zawodnicy biorący udział w danym maratonie. Mogą oni mieć od 0 do 2 potomków, przy czym na lewo znajdują się elementy nie większe od węzła rodzicielskiego, po prawej stronie natomiast są większe wartości. W projekcie jest również zdefiniowana pomocnicza lista jednokierunkowa przechowująca nazwiska zawodników.

2.2 Algorytmy

Program został napisany z wykorzystaniem funkcji rekurencyjnych, które bardzo ułatwiają poruszanie się po drzewie binarnym. W tym przypadku jest to znacząca optymalizacja, gdyż znalezienie danego zawodnika zajmie o wiele mniej czasu. Utworzenie drzewa i jego przejście jest wykonywane w średnim czasie O(n log n). W przypadku pesymistycznym (gdy drzewo zdegeneruje do listy) mamy O(n2).

3. Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń bądź za pomocą kompilatora. Należy przekazać do programu nazwy plików wejściowych z maratonami po odpowiednim przełączniku –i.

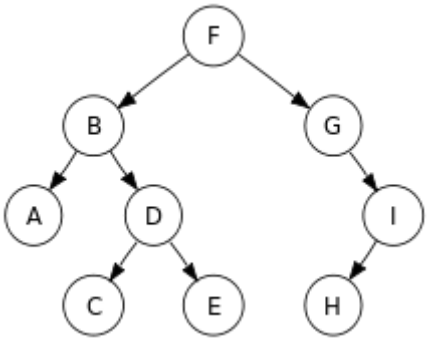
Przykładowe wywołanie programu:

maraton.exe –i maraton1.txt –i maraton2.txt –i maraton3.txt

Pliki są plikami tekstowymi. Uruchomienie programu bez żadnego parametru lub z niepoprawnymi parametrami powoduje wyświetlenie komunikatu:

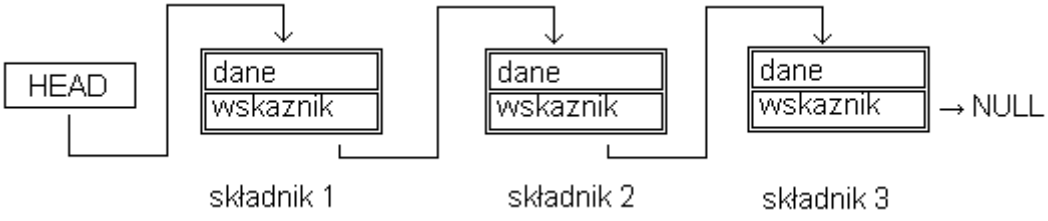
„nie podano protokołów”

„wprowadzono niepoprawne parametry”



Rys.1 Drzewo Binarne

Rysunek 1 przedstawia przykładowe drzewo binarne. W naszym przypadku litery reprezentują elementy przechowujące informacje o zawodnikach.



Rys.2 Lista jednokierunkowa

Rysunek 2 przedstawia listę jednokierunkową. Każdy element przechowuje informacje o konkretnym maratonie oraz dodatkowo wskaźnik na drzewo z rysunku1.

4. Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym. W programie rozdzielono interfejs (komunikację z użytkownikiem) od logiki aplikacji (zapisywania danych do plików wyjściowych).

4.1 Ogólna struktura programu

W funkcji głównej programu są tworzone dynamicznie listy maratonów i nazwisk. Następnie są wywoływane funkcje:

```
wczytaj_arg(argc, argv, mGlowa); //wczytuje dane tworząc strukturę listy drzew

zapisz(mGlowa, nGlowa); //zapisuje zawodników

usun_liste_i_drzewo(mGlowa); //zwalnia pamięć

delete mGlowa; //usuwamy wskaźnik na listę maratonów
```

4.2 Typy strukturalne

```
struct zawodnik {
	int msce_na_mecie=0;
	string nazwisko;
	int nr_zawodnika=0;
	string czas;
	zawodnik * lewy = nullptr;
	zawodnik * prawy = nullptr;
};
```

Ma postać drzewa binarnego, przechowuje informacje o zawodnikach. Każdy zawodnik posiada wskaźnik na prawy i lewy element.

```
struct maraton{
	string nazwa_maratonu;
	string data;
	zawodnik* zawodnik_wsk = nullptr;
	maraton* nast = nullptr;
};
```

Ma postać listy jednokierunkowej, przechowuje informacje o maratonie. Każdy element posiada wskaźnik na następny oraz na strukturę zawodnik.

```
struct nazwiska {
	string nazwisko;
	nazwiska* nast=nullptr;
};
```

Ma postać listy jednokierunkowej, przechowuje nazwiska. Każdy element posiada wskaźnik na następnika.

4.3 Szczegółowy opis implementacji funkcji

```
bool wczytaj(const string& nazwaPliku, maraton*& glowa);
```

Wczytuje dane z plików wejściowych do struktur.

```
void polacz_maratony(vector<string> maratony, int ilosc, maraton*& glowa);
```

Wywołuje metodę wczytaj dla plików, których nazwy są zawarte w vectorze maratony. Wywołuje sie do momentu wczytania wszystkich plików.

```
void zapisz_zawodnika(zawodnik* glowa, maraton* mGlowa);
```

Zapisuje zawodnika do pliku w momencie gdy jego nazwisko wystąpiło po raz pierwszy.

```
void dodaj_wynik_zawodnikowi(zawodnik* glowa, maraton* mGlowa);
```

Dopisuje dane do pliku konkretnego zawodnika gdy wystąpił w więcej niż jednym maratonie.

```
zapisz_lub_dodaj(zawodnik* glowa, maraton* mGlowa, nazwiska*& nGlowa);
```

Sprawdza czy nazwisko danego zawodnika juz wystąpiło, jeżeli tak to wywołuje funkcje dodaj_wynik_zawodnikowi, jeżeli nie to wywołuje zapisz_zawodnika

```
zapisz_rekurencja(zawodnik* glowa, maraton* mGlowa, nazwiska*& nGlowa);
```

Przechodzi po wszystkich elementach drzewa zawodników i w zależności od spełnionych warunków wywołuje funkcje zapisujące

```
zapisz(maraton* mGlowa, nazwiska* nGlowa);
```

Jeżeli lista maratonów nie jest pusta wówczas funkcja przechodzi po wszystkich elementach listy i drzewa i wykonuje operacje zapisujące dane zawodników w plikach.

```
usun_liste_i_drzewo(maraton*& glowa);
```

Zwalnia pamięć ze struktur listy maratonów i drzewa zawodników

```
wczytaj_arg(int argc, char* argv[], maraton*& glowa);
```

Odczytuje parametry programu, jeżeli są poprawnie wprowadzone wówczas dodaje nazwy plików wejściowych do wektora maratony oraz wywołuje funkcje scalającą wszystkie podane maratony.

```
maraton_program(int argc, char* argv[]);
```

Główna funkcja programu, wywołuje metody realizujące program w odpowiedniej kolejności oraz tworzy dynamicznie nowe elementy strukturalne.

```
wstaw_do_drzewa(zawodnik*& glowa, zawodnik*& zawodnikN);
```

Wstawia przekazany w argumencie element na odpowiednie miejsce w drzewie

```
usun_drzewo(zawodnik*& glowa);
```

Funkcja usuwa rekurencyjnie wszystkie elementy drzewa zawodników

```
zapisz_zawodnikow_rekurencja(zawodnik* glowa, maraton* pGlowa);
```

Rekurencyjnie przechodzi po elementach drzewa zawodników i dla każdego z nich wywołuje funkcje zapisz_zawodnika

```
dodaj_do_listy_nazwisk(nazwiska*& glowa, const string& nazwisko);
```

Obsługuje jednokierunkowa listę nazwisk, dodaje do niej nowe elementy

```
znajdz(nazwiska* glowa, const string& nazwisko);
```

Szuka w liście nazwisko wystąpienia przekazanego przez argument nazwiska

```
usun_liste_nazwisk(nazwiska*& glowa);
```

Zwalnia pamięć ze struktury listy jednokierunkowej

5. Testowanie

Program został przetestowany na różnego rodzaju plikach wejściowych. W wyniku testów pojawiły się pewne błędy, które w większości zostały poprawione. Były one głównie związane z danymi w drzewach binarnych, poruszaniu się po nich oraz znajdowaniu konkretnych elementów. Pojawiły się wycieki pamięci. Początkowo uważałem, że były one związane ze zwalnianiem pamięć po drzewie, lecz finalnie okazało się, że powodem jest wczytywanie danych do drzewa.

6. Wnioski

Sam program jest dość prostym w implementacji projektem, ponieważ operujemy jedynie na zapisywaniu części z wcześniej wczytanych danych. Jednak jeżeli stawiamy na optymalizację działania, wówczas konieczne jest skorzystanie ze struktury drzewa binarnego co wymaga znajomości jego budowy oraz sposobu poruszania się. Pomimo wielu problemów, które napotkałem po drodze, wszystkie udało się naprawić. Program działa poprawnie i nie posiada wycieków pamięci.

Literatura

- Jerzy Grębosz „Symfonia C++ Standard. Programowanie w języku C++”
- http://eduinf.waw.pl