Politechnika Śląska w Gliwicach

Wydział Informatyki ,Elektroniki i Informatyki

Podstawy Programowania Komputerów

Temat Projektu

„Maraton”

Autor:	Andrzej Szuwald
Prowadzący:	dr.inż Agnieszka Danek
Rok Akademicki	2020/2021
Kierunek	Teleinformatyka
Rodzaj studiów	Stacjonarne
Semestr	1
Termin laboratorium	Wtorek 8:00-10:15, Czwartek 8:00-10:15
Grupa	1
Sekcja	3
Termin oddania sprawozdania	13.12.2020
Data oddania sprawozdania	13.12.2020

1. Treść zadania

Maratończycy biorą udział w różnych zawodach. Po zakończeniu tworzona jest lista rankingowa w następującej postaci: w pierwszej linii jest nazwa maratoru, 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 ma postać:

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 Jawroka:

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 utworzenia plików z wynikami zawodników z różnych maratonów w plikach z ich nazwiskiem biorąc dane z różnych plików wejściowych z danymi z danych maratonów.

2.1 Struktury danych

W programie wykorzystano wektory do przechowywania wartości. Wektory to dynamicznie rozszerzalne tablice, które okazały się w rozwiązywaniu problemów bardzo użytne z powodu ich specyfiki oraz łatwości dodawania, bądź usuwania danych. Były lepszym rozwiązaniem niż listy, które były pierwotnie przewidziane przeze mnie na ten program, ponieważ łatwiej jest trzymać dane w wektorach oraz łatwiej jest zarządzać wektorami.

2.2 Algorytmy

Na początku program sprawdza czy użytkownik wprowadził poprawnie dane, jeśli tak to później program wczytuje nazwy plików do wektora pliki(„std::vector<std::string> pliki;”) sprawdzając czy dany plik istnieje. Później tworzy nowy wektor zawodnicy („std::vector<zawodnik> zawodnicy;”) do którego zostaną umieszczone później nazwiska zawodników oraz ich wyniki używając struktury zawodnik zawierającej (std::string nazwisko; oraz std::vector<wynik>wyniki(struct wynik posiada: (string czas; string maraton; string data;)). Przed jednak umieszczeniem wyników program sprawdza za pomocą („int indeksZawodnika = szukajZawodnika(nazwiskoZawodnika, zawodnicy);”), czy dany zawodnik już został dodany do programu, jeśli tak to doda wyniki, a jeśli nie to doda nowego zawodnika. Gdy program doda wszystkie wyniki i zawodników to program wpierw sortuje wyniki zawodników za pomocą („zawodnicy[i].wyniki = sortujWyniki(zawodnicy[i].wyniki);”), a później tworzy pliki z nazwiskiem zawodnika i posegregowane zapisuje do pliku z nazwiskiem zawodnika w pętli, gdy program utworzy wszystkie pliki z zawodnikami i z ich wynikami to zamyka się.

3.Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programy nazwy plików wejściowych

Program –i plik.txt plik2.txt plik3.txt

Pliki są plikami tekstowymi z dwolonym rozszerzeniem. Błędne uruchomienie programu spowoduje wykonanie się funkcji help() i wyświetlenie komunikatu:

Wpisales cos nie poprawnie
Prosze wprowadzic w ten sposob:(-i nazwapliku nazwapliku nazwapliku)
Dobry Przyklad:(-i plik.txt tekst.txt tekst2.txt)

Natomiast w przypadku, gdy któryś plik zostanie nieznaleziony to program poinformuje użytkownika o tym, który plik nie został znaleziony za pomocą:

---

cout << "Pliku:[" << nazwapliku << "] Nie udalo sie znalesc" << endl; w funkcji int czyplikistnieje(string nazwapliku)

4.Specyfikacja wewnętrzna

4.1Typy

W programie występują następujące typy:

```
struct wynik {
   string czas;
   string maraton;
   string data;
};
```

Typ ten to struktura, która będzie zawierać dane danego wyniku.

```
struct zawodnik {
   std::string nazwisko;
   std::vector<wynik> wyniki;
};
```

To struktura dla każdego zawodnika zawierająca jego wyniki z każdego maratonu w którym brał udział.

4.2 Ogólna struktura programu

W funkcji głównej znajduje się sprawdzenie czy program został wywołany w prawidłowy sposób za pomocą:

if(argc>=3) oraz if (przelacznik.compare(argv[1]) == 0)

Gdy program został wywołany w nieprawidłowy sposób to urochomi się funkcja help() I wypisze na ekran stosowny komunikat i program się zakończy.

Po sprawdzeniu czy program został wywołany w prawidłowy sposób główna funkcja tworzy wektor pliki do którego wpisywane są nazwy plików, a jeśli się nie udaje otworzyć pliku co oznaczy, że nie istnieje to :

```
int czyplikistnieje(string nazwapliku)
```

Wypisuje, że nie udało się znaleźć danego pliku.
Później za pomocą funkcji:

```
std::vector<zawodnik> czytajWyniki(std::vector<std::string> pliki)
```

dodaje zawodników do wektora zawodnicy oraz wczytuje ich wyniki do niego. W funkcji znajduje się funkcja szukajzawodnika, która sprawdza czy dany zawodnik już się pojawił.

wczytuje odpowiednie dane do wektora zawodnicy:

```
std::vector<zawodnik> zawodnicy = czytajWyniki(pliki);
```

Gdy wszystkie dane zostały wczytane to w pętli wyniki są najpierw sortowane za pomocą funkcji

```
zawodnicy[i].wyniki=sortujWyniki(zawodnicy[i].wyniki);
```

a później są tworzone po kolei pliki z nazwiskami zawodników do których są wpisywane ich wyniki w różnych maratonach za pomocą funkcji

```
zapiszZawodnikaDoPliku(zawodnicy[i]);
```

Gdy wszystkie pliki zostały utworzone i wszystkie dane zostały wpisane to program wychodzi.

4.3 Szczegółowy opis implementacji funkcji

W funkcji głównej znajduje się sprawdzenie czy program został wywołany w prawidłowy sposób za pomocą:
if(argc>=3) oraz if (przelacznik.compare(argv[1]) == 0)
Gdy program został wywołany w nieprawidłowy sposób to urochomi się funkcja help() wypisze na ekran stosowny komunikat i program się zakończy.

Po sprawdzeniu czy program został wywołany w prawidłowy sposób główna funkcja tworzy wektor pliki

```
std::vector<std::string> pliki;
```

Później za pomocą pętli I funkcji

```
int czyplikistnieje(string nazwapliku)
{
   std::ifstream plik;
   plik.open(nazwapliku);
   if (plik.good()==true)
   {
      plik.close();
      return 1;

   }
   if (plik.good() == false)
   {
      cout << "Pliku:[" << nazwapliku << "] Nie udalo sie znalezc" << endl;
      plik.close();
      return 0;
   }

}
```

sprawdzane jest czy plik został poprawnie otwarty, czyli czy istnieje, jeśli plik istnieje to jest dodawany do wektora pliki.

```
for (int i = 2; i < argc; i++)
{
   int m=czyplikistnieje(argv[i]);
   if (m == 1)
   {
    pliki.push_back(argv[i]);
   }

}
```

Później tworzony jest wektor zawodnicy w którym przechowywane będą nazwiska zawodników i wyniki poszczególnych zawodników

```
std::vector<zawodnik> zawodnicy = czytajWyniki(pliki);
```

Struktura zawodnik(użyto wektora do przechowywania wyników danego zawodnika):

```
struct zawodnik {
   std::string nazwisko;
   std::vector<wynik> wyniki;
};
```

Struktura wynik:

```
struct wynik {
   string czas;
   string maraton;
   string data;
};
```

W funkcji

```
std::vector<zawodnik> czytajWyniki(std::vector<std::string> pliki)
```

otwierane są po kolei pliki w pętli

```
for (int i = 0; i < pliki.size(); i++)
```

po otwarciu pliku następuje pobranie nazwy,daty maratonu, a później pozycji,nazwiska,numeru i czasu danego zawodnika w danym maratonie.

```
std::vector<zawodnik> czytajWyniki(std::vector<std::string> pliki) {

   std::vector<zawodnik> zawodnicy;
   for (int i = 0; i < pliki.size(); i++)
   {
      std::ifstream plik;
      plik.open(pliki[i]);

      if (plik) {
         std::string nazwaMaratonu, dataMaratonu;

         std::getline(plik, nazwaMaratonu);
         std::getline(plik, dataMaratonu);

         std::string pozycjaNaMecie;
         std::string nazwiskoZawodnika;
         std::string numerZawodnika;
         std::string czas;

         while (!plik.eof())
         {
            std::getline(plik, pozycjaNaMecie, ',');
            std::getline(plik, nazwiskoZawodnika, ',');
            std::getline(plik, numerZawodnika, ',');
            std::getline(plik, czas, '\n');
```

Później sprawdzam czy dany zawodnik już się pojawił za pomocą funkcji:

```
int szukajZawodnika(std::string nazwisko, std::vector<zawodnik>& zawodnicy){
   for (int i = 0; i < zawodnicy.size(); i++)
   {
      if (zawodnicy[i].nazwisko.compare(nazwisko) == 0)
      {
         return i;
      }
   }

   return -1;
}
```

Jeśli dany zawodnik jeszcze się nie pojawił to jest dodawany i dodaje mu się jego wynik, jeśli natomiast już się pojawił to dodaje się jego kolejny wynik.

```
        int indeksZawodnika = szukajZawodnika(nazwiskoZawodnika, zawodnicy);
        if (indeksZawodnika == -1)
            {

               zawodnik nowyZawodnik;
               nowyZawodnik.nazwisko = nazwiskoZawodnika;

               zawodnicy.push_back(nowyZawodnik);
               indeksZawodnika = zawodnicy.size() - 1;
            }

            wynik wynik;
            wynik.maraton = nazwaMaratonu;
            wynik.data = dataMaratonu;
            wynik.czas = czas;

            zawodnicy[indeksZawodnika].wyniki.push_back(wynik);
         }
      }

      plik.close();
   }

   return zawodnicy;
}
```

Gdy wszyscy zawodnicy wraz z ich wynikami zostaną dodani już do wektora zawodnicy to za pomocą pętli sortuje się wyniki i zapisuje do plików za pomocą dwóch funkcji

```
        for (int i = 0; i < zawodnicy.size(); i++)
        {
            zawodnicy[i].wyniki = sortujWyniki(zawodnicy[i].wyniki);
            zapiszZawodnikaDoPliku(zawodnicy[i]);
        }
```

Funkcji sortującej wyniki:

```
std::vector<wynik> sortujWyniki(std::vector<wynik> wyniki)
{
   for (int i = 0; i < wyniki.size(); i++)
   {
      for (int j = 0; j < wyniki.size(); j++)
      {
         if (i != j)
         {
            wynik temp = wyniki[i];
            if (temp.data.compare(wyniki[j].data) < 0)
            {
               wyniki[i] = wyniki[j];
               wyniki[j] = temp;
            }
         }
      }
   }

   return wyniki;
}
```

Funkcja ta sortuje od najstarszej daty do najnowszej po czym zwraca posortowane wyniki. Później wywoływana jest funkcja, która tworzy pliki z nazwiskami zawodników oraz ich wynikami:

```
void zapiszZawodnikaDoPliku(zawodnik zawodnik)
{
   std::ofstream plik;

   plik.open(zawodnik.nazwisko);
   if (plik)
   {
      plik << zawodnik.nazwisko << std::endl << std::endl;
      for (int i = 0; i < zawodnik.wyniki.size(); i++)
      {
         wynik temp = zawodnik.wyniki[i];
         plik << temp.data << " " << temp.maraton << " " << temp.czas << std::endl;
      }
   }
plik.close();
}
```

W funkcji tej najpierw tworzymy plik, a później po kolei wypisujemy daty oraz nazwy danego maratonu oraz czas zawodnika w nim biorąc dane z wektora zawodnicy.
Po wykonaniu tych wszystkich funkcji program się zakończy.

5.Testowanie

Program został przetestowany na różnych plikach.W przypadku, gdy plik jest pusty to nie powoduje błędu w programie. W przypadku, gdy plik nie istnieje, bądź nie udało się go otworzyć to użytkownika program informuje, że pliku o danej nazwie nie znaleziono. Gdy użytkownik źle wpisze parametry program się nie wykonuje, natomiast wypisuje na ekran przykład poprawnego wpisania.

6.Wnioski

Program służący do tworzenia plików z danymi zawodników jest programem nieco wymagającym, choć nie najtrudniejszym. Jednym z problemów okazało się wczytywanie danych pod nazwą i datą maratonu, ponieważ w przypadku, gdy zostały wczytywane getlinem do przecinka to program pobierał dane jeszcze z następnej linii. Początkowo próbowałem zrobić to z pomocą zmiennej int numerlinii, która określała pozycję na której znajdował się program i według niej pobierać dane w pętli , ale zrezygnowałem z tego na rzecz pętli while(!plik.eof()) w której po kolei wczytywałem getline odpowiednie dane. Kolejnym problemem było wybranie odpowiednich stuktur po wielu próbach napisania kodu i usuwania go doszedłem do wniosku, że wektory były najprzyjemniejszą strukturą do pracy nad tym programem. Listy sprawiły nieco kłopotów przy pracy wraz z nimi przez co zrezygnowałem z list w tym programie na rzecz wektorów.

Literatura