Zadanie 2 przecięcia zer.cpp

Aleksander Kłak, prowadzący Łukasz Janiec

13 listopada 2020

Spis treści

1 Przygotowanie do zajęć

Przygotowaniem do zajęć był algorytm zapisany w kodzie programu oraz wykonanie trzech programów, które zawierały podstawowe instrukcje sterujące. Na GitLab umieściłem zarówno algorytm jak i wszystkie programy. Wszystkie pliki znajdują się w repozytorium aklak_podstawy_programowania/Z2/przygotowanie.

2 Kompilacja oraz działanie programu

Docelowy program, tetno.cpp kompilowałem w *Visual Studio Code*. Program opiera się na funkcji *main()*, ale przydatna jest też stworzona przeze mnie funkcja *signum()*. Ta druga zwraca jedynkę dla wszystkich liczb dodatnich, ujemną jedynkę dla wszystkich liczb ujemnych oraz zero dla wszystkich czytanych zer. W funkcji main zadeklarowałem wskaźnik na otwierany plik, tablicę służącą do wyseparowania czasu, który następnie będzie zignorowany, potrzebne indeksy oraz wartości czytane z pliku. Deklaruję też użycie value_prev, aby móc sprawdzić czy między dwoma kolejnymi wartościami zachodzi przejście przez zero.

Następnie program czyta plik za pomocą funkcji *fopen()*. Jest też możliwość wybrania pliku tekstowego, z którego program będzie pobierał dane. 1.txt oraz 2.txt to dane profesora Muszyńskiego. Plik z moimi danymi to 5.txt. Liczby z 5.txt zostały wygenerowane całkowicie losowo, dlatego poprawny wynik dla tego zestawu danych jest bardzo ciekawy, o czym opowiem później. Kolejną częścią programu jest pętla, która wykonuje się aż do końca pliku. Wskazany plik jest czytany linijka po linijce, gdzie za każdym razem *sample_index* zwiększa się o jeden. Pierwszy *if* (28 linijka kodu) grupuje sto próbek w jedną paczkę, w której liczone jest, ile razy nastąpiło przecięcie zera. Następnie policzone tętno sprawdzane jest z wartościami - minimalną i maksymalną, tak, że jeśli *hr* nie mieści się w tych granicach, to program wypisuje tekst o niepoprawności wartości *hr*. W następnej części, program obsługuje przypadek specjalny **99**, który kończy program oraz używam nakreślonej przeze mnie funkcji *signum()*. Pod koniec kodu, plik jest zamykany, a program kończy się.

3 Kod programu

Listing 1: C++

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int signum(double x)
{
    if (x>0) return 1;
    else if (x==0) return 0;
    else return -1;
}

int main ()
{
    FILE * data_file;
    char cline[50];
    int sample_index = 0,polarity_change_index = 0;
    double value,value_prev,hr;

    data_file = fopen ("1.txt", "r");
    while ( ! feof (data_file) )
    {
        fscanf (data_file,"%s %lf\n",cline,&value);

        sample_index++;

        if (sample_index % 100 == 0)
        {
            hr = polarity_change_index * 6;
            if ((hr >= 50) && (hr <= 80)) cout << "hr=" << hr << "\n";
            else cout << "wartosc hr niepoprawna \n";
            polarity_change_index = 0;
            sample_index = 0;
        }
        if(value != 99)
        {
            if ((sample_index > 1) && (signum(value) * signum(value_prev) < 0))
            polarity_change_index ++;

            if (signum(value) != 0) value_prev = value;
        }
    }
    fclose (data_file);
    return 0;
}
```

4 Testy oraz dane wejściowe

Program wykonuje poprawne obliczenia dla każdego z plików z danymi wejściowymi. Program odczytuje oba formaty daty poprawnie, dzięki czemu może z obu typów wyselekcjonować pożądaną wartość. Wyniki programu przedstawiam w Tabeli 1: Testy.

Tabela 1: Testy

hr z pliku 1.txt	hr z pliku 2.txt	5.txt	liczba przejść przez zero w 5.txt	niepoprawne tętno w 5.txt
66	66	niepoprawne	46	276
niepoprawne	niepoprawne	niepoprawne	47	282
54	54	niepoprawne	52	312
66	66	niepoprawne	54	324
66	66	niepoprawne	39	234
66	66	niepoprawne	45	270
66	66	niepoprawne	46	276
78	78	niepoprawne	42	252
niepoprawne	niepoprawne	niepoprawne	41	246
66	66	niepoprawne	46	276

Dla plików 1.txt oraz 2.txt wyniki prezentują się prawdopodobnie oraz są takie same. Ich podobieństwo świadczy o tym, że program sobie dobrze radzi z każdym z formatów czasu. W moim pliku, 5.txt, został zastosowany format czasu z 1.txt, jednak żadna z wartości nie jest poprawna. Jest to zasługa ograniczeń tętna z 31 linijki kodu. Mój plik z tysiącem losowych liczb od -10 do 10 zawiera w każdej setce około 50 dodatnich i około 50 ujemnych liczb, dodatkowo są one w losowych miejscach, co tylko zwiększa liczbę przejść przez zero.

5 Wnioski

Starałem się by układ graficzny programu był jak najczytelniejszy. Moim zamysłem było również tworzenie takich nazw zmiennych, żeby osobie czytającej kod ułatwić zrozumienie ich przeznaczenia. Komentarze do programu zawarte są w wersji kodu udostępnionej na GitLabie. Głównym wnioskiem z zadania przecięć zer jest zdecydowanie umiejętność określenia jakie zastosowanie ma mieć program i co będzie w nim pryncypialne jak i umieć określić to co w treści zadania tylko zaciemnia obraz finalnego kodu. W tym przypadku jest to format czasu, który nie jest wykorzystywany później w programie, należy jedynie umieć go oddzielić od sedna, czyli wartości.