Współpraca z plikami

Do tej pory posługiwaliśmy się danymi, które pochodziły ze standardowego wejścia (klawiatury) i były zapisywane do standardowego wyjścia (terminala). Wykorzystywanie takich danych jest jednak często niewygodne i nieefektywne (wyobraźmy sobie ilość pracy jaka by nas czekała przy podawaniu kilkudziesięciu parametrów za pomocą klawiatury, za każdym razem kiedy uruchamiamy program). W takich przypadkach posługujemy się plikami. Pliki pozwalają na wczytanie i zapisanie dużych ilości przygotowanych wcześniej danych.

W języku C, komunikacja z plikami przebiega niemalże identycznie, jak czytanie danych z klawiatury i drukowanie ich na ekranie. Odpowiednikiem funkcji `scanf()` jest `fscanf()` a funkcji `printf()` jest `fprintf()`. Różnica polega na tym, że w przypadku funkcji służących do obsługi plików, jako pierwszy argument musimy podać wskaźnik zawierający adres pliku.
Wskaźniki przechowujące adresy plików mają typ `FILE`, a sam adres można pobrać za pomocą funkcji `fopen()`.

Przyjrzyjmy się przykładowemu kodowi źródłowemu:

```
FILE *f;              // deklarujemy wskaznik do typu FILE, o nazwie f

f = fopen("dane.txt", "w");
                      // otwieramy plik o nazwie dane.txt z zamiarem
                      // zapisu ("w") a do zmiennej o nazwie f przypisuj

fprintf(f, "Zapisujemy wlasnie ten tekst do pliku\n");

fclose(f);            // zamykamy plik
```

W tym momencie należy zwrócić uwagę na pewną kwestię. Funkcja `fopen()` zgłasza żądanie otworzenia nowego pliku. Nie oznacza to jednak, że system operacyjny zawsze dysponuje odpowiednimi zasobami i takie żądanie zostanie spełnione. Jeśli system operacyjny nie przydzieli nam dostępu do pliku, do wskaźnika typu `FILE` zostanie przypisana wartość `NULL`. Dlatego należy koniecznie sprawdzić czy plik został poprawnie otworzony. Zmodyfikujmy odpowiednio nasze instrukcje.

Teraz wyglądają one tak:

```
FILE *f;
f = fopen("dane.txt", "w");

if(f == NULL) {
   printf("Blad otwarcia pliku!\n");  // drukujemy komunikat o bledzie
   exit(1);                           // konczymy prace programu (funkcja exi
                                      // znajduje sie w bibliotece stdlib.h)
}

// Tutaj wykonujemy operacje na pliku (w naszym przypadku zapis).
// Gdy plik nie bedzie juz wiecej potrzebny zamykamy go.

fclose(f);
```

Pliki można otwierać nie tylko w trybie zapisu *(ang. write)* `w` (który zawsze czyści plik i wypełnia go od nowa), ale również w trybie dopisywania do pliku *(ang. append)* `a` lub czytania z pliku *(ang. read)* `r`. Można również wybrać czy tworzony/odczytywany plik ma być obsługiwany w trybie tekstowym, czy binarnym. Aby móc obsługiwać plik binarny należy dodać literę `b` do trybu zapisu.

Przykładowe instrukcje zaprezentowano poniżej.

```
void main() {
  int a;
  FILE *f, *g, *innyPlik;

  f = fopen("plik1.txt", "w");       // Zapis w trybie tekstowym
  g = fopen("plik2.dat", "wb");      // Zapis w trybie binarnym
  innyPlik = fopen("Dane.txt", "r"); // Czytanie z pliku

  if (f == NULL || g == NULL || innyPlik == NULL) {
    printf("Nie udalo sie otworzyc co najmniej jednego pliku!\n");
    exit(1);
  }

  fprintf(f, "Zapisujemy wartosc a do plik1.txt, a = %d\n", a);
  fprintf(g, "Binarnie zapisujemy ten tekst do plik2.dat\n");

  fscanf(innyPlik, "%d", &a);        // wczytujemy liczbe calkowita z pliku Da
                                     // i zapisujemy jej wartosc do zmiennej a
```

```
  // Tu mozemy wykonac jeszcze inne operacje na otwartych plikach

  fclose(f);
  fclose(g);
  fclose(innyPlik);
}
```

W powyższym przykładzie, zaprezentowaliśmy użycie funkcji `fscanf()`, która działa analogicznie do funkcji `scanf()`.

Bufor

Zapisywanie danych do pliku jest „względnie" czasochłonną operacją. Jest to spowodowane różnicą pomiędzy czasem dostępu do pamięci RAM a dyskiem HDD. Aby nie zapisywać co chwilę drobnych ilości danych na dysk, system operacyjny gromadzi je w buforze. Gdy uzna, że jest on już pełny, przepisuje dane na dysk a sam bufor – czyści. Funkcja `fclose(plik)` zapisuje dane z bufora do pliku, a następnie zamyka plik. Chcąc wywołać jedynie opróżnienie bufora, należy użyć funkcji `fflush(plik)`.

Opisane zachowanie ma istotne konsekwencje:

- Jeżeli przed wykonaniem komendy `fclose(plik)` nastąpi błąd programu, może się zdarzyć, że żadne dane nie zostaną zapisane.
- Wyjmując pendrive z komputera bez skorzystania z opcji „bezpieczne usuwanie sprzętu" narażamy się na błędne zapisanie pliku. Część danych może być jeszcze w buforze, mimo że okienko kopiowania zostało już formalnie zamknięte.

Uwaga: wszystkie funkcje związane z obsługą plików znajdują się w bibliotece `stdio.h`. W związku z tym, do pliku programu należy dołączyć instrukcję preprocesora załączającą tę bibliotekę: `#include <stdio.h>`

Ćwiczenia

W praktyce inżynierskiej, pliki często zawierają dane pochodzące z eksperymentu lub symulacji. Plik przebieg.txt zawiera fragment przebiegu czasowego dla wartości trzech składowych prędkości (u, v, w) pochodzących z symulacji przepływu powietrza przez turbinę wiatrową. Wartości tych składowych zostały zmierzone w punkcie znajdującym się tuż za turbiną. Otwórz i obejrzy plik, aby przekonać się, w jaki sposób ułożone są dane (każda z kolumn odpowiada jednej ze składowych prędkości (u, v, w) a kolejne wiersze odpowiadają kolejnym krokom czasowym).

Napisz program, który:

- Otworzy plik `przebieg.txt`.
- Wczyta dane z pliku do trzech zadeklarowanych statycznie tablic `u`, `v` oraz `w`. Każda z tablic powinna mieć wymiar 2000. Odczytywanie danych z pliku zrealizuj za pomocą pętli `for`.
- Po wczytaniu wszystkich wartości do tablic, obliczy średnią dla każdej ze składowych. Średnia dana jest wzorem:

$$\mu = \frac{\sum_{i=1}^{n} u_i}{n}$$

- Obliczy odchylenie standardowe dla każdej ze składowych. Odchylenie standardowe dane jest wzorem:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (u_i - \mu)^2}{n-1}}$$

- Zapisze do innego pliku raport z obliczeń, w którym poda wszystkie obliczone wielkości.
- Wydrukuje na ekran to samo co w powyższym punkcie.
- Za pomocą biblioteki `winbgi2.h` narysuje wykres każdego z wczytanych przebiegów i zaznaczy na nim wartości μ, $\mu+\sigma$ oraz $\mu-\sigma$. Oceń krytycznie wyniki uzyskane swoim programem na podstawie obserwacji wykresu. Czy średnie i odchylenia standardowe mają wiarygodne wartości?

Dla sprawdzenia wczytaj też plik `przebieg.txt` do arkusza kalkulacyjnego i utwórz wykres obrazujący te przebiegi.

Wskazówka: Zauważ, że każda suma daje się policzyć z wykorzystaniem pętli `for` w następujący sposób:

```
double suma = 0;
int i;

for(i = 0; i < n; i++) {
   suma += a[i];
}
```

Wykorzystanie funkcji do poprawienia przejrzystości programu

Zmodyfikuj swój kod tak, aby odpowiednie bloki instrukcji były realizowane w funkcjach `srednia()` i `odchStd()`. Funkcje te powinny mieć poniższe nagłówki:

```
double srednia(double tablica[], int n);
double odchStd(double tablica[], int n, double wartoscSrednia);
```

Następnie zmodyfikuj kod funkcji `main()` tak, aby część dotycząca obliczeń dała się zwięźle zapisać w poniższej postaci:

```
void main() {
   // deklaracje i kod wczytujacy dane

   um = srednia(u, n);
   vm = srednia(v, n);
   wm = srednia(w, n);

   u_std = odchStd(u, n, um);
   v_std = odchStd(v, n, vm);
   w_std = odchStd(w, n, wm);

   // dalsza czesc programu zajmujaca sie raportowaniem wynikow
}
```

* Dla dociekliwych

Obliczenia na komputerze prowadzone są ze skończoną dokładnością. Zmodyfikuj swój kod tak, aby bieżąca wartość średniej była liczona „w locie” – w trakcie czytania danych z pliku (będzie to średnia wartość przeczytanych dotąd elementów). Wystarczy, że zrobisz to dla jednej składowej prędkości (np. u). Możesz średnią obliczaną podczas czytania danych drukować na bieżąco na ekran. Na końcu, porównaj wartość średniej uzyskanej w ten sposób z wartością policzoną w poprzednim poleceniu. Pseudokod algorytmu znajdziesz poniżej. Zapisz go w sposób zrozumiały dla komputera, w języku C.

```
biezaca_srednia = 0

petla po i od 1 do n (czytajaca dane)
{
   przeczytajNowyElementZPlikuIWpiszGoDoTablicy

   biezaca_srednia = (biezaca_srednia * (i - 1) + u[i]) / i
}

// Po zakonczeniu petli, biezaca_srednia to srednia z calego zbioru
```

Zastanów się, dlaczego taki algorytm liczenia średniej w sensie matematycznym prowadzi do tak samo zdefiniowanej średniej. Jeśli trudno Ci go zrozumieć, wymyśl sobie zbiór 4-elementowy i zastosuj do niego powyższy algorytm krok po kroku na kartce.

Pytanie: Czy obie średnie (policzone na komputerze dwoma sposobami) mają tę samą wartość? Czy coś się zmieni, gdy uśrednisz inną składową prędkości?