

## Współpraca z plikami

Praktyczny program (szczególnie inżynierski) bardzo często musi współpracować z plikami. Czasem, przy obliczeniach trwających wiele godzin lub dni wręcz zależy nam na tym, by program działał samodzielnie bez potrzeby interakcji ze strony użytkownika. Wróćmy jednak do plików. Najczęściej chodzi o możliwość wczytania danych wejściowych z jednego (bądź wielu) plików, przeprowadzenie obliczeń wewnątrz programu i zapisanie wyników do innego pliku (bądź wielu plików). W języku C komunikacja z plikami prowadzona jest niemalże identycznie, jak czytanie danych z klawiatury i wydruk na ekran, co realizowaliśmy za pomocą znanych już funkcji `scanf` oraz `printf`. Musimy jednak najpierw określić, jaki plik chcemy utworzyć bądź otworzyć i w jakim celu go tworzymy/otwieramy. Ponadto, analogiem instrukcji `printf` do zapisu do plików jest instrukcja `fprintf`, zaś funkcja `scanf` jest zastąpiona przez funkcję `fscanf`. Przyjrzyjmy się przykładowemu kodowi źródłowemu.

```
FILE *f; // deklarujemy zmienną typu FILE o nazwie f
        // tak naprawdę to wskaźnik (o tym jednak w później)

errno_t err_f = fopen_s( &f, "plik.txt", "w"); // otwieramy plik o nazwie
        // z zamiarem zapisu ("w") i przypisujemy go do zmiennej o nazwie f
        // errno_t określa wynik otwarcia
        // kod err_f = 0 oznacza, że operacja przebiegła pomyślnie

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

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

Warto zwrócić uwagę na dwie kwestie. Po pierwsze, w funkcji `fprintf` (to samo dotyczy funkcji `fscanf` jako pierwszy argument trzeba podać *strumień* - de facto nazwę zmiennej typu `FILE`, z którym zachodzi komunikacja (zapis lub odczyt). Dlatego tu podajemy `f`, bo tak właśnie nazwaliśmy naszą zmienną. Ponadto, istotne jest, by sprawdzić, czy plik udało się otworzyć. W przeciwnym razie jakiegokolwiek operacje nie miałyby sensu lub skończyły się błędem naszego programu. Zmodyfikujmy więc nasze instrukcje. Teraz wyglądają tak:

```
FILE *f;
errno_t err_f = fopen_s( &f, "plik.txt", "w"); // otwieramy plik o nazwie
```

```
// z zamiarem zapisu ("w") i przypisujemy go do zmiennej o nazwie f
// errno_t określa wynik otwarcia
// kod err_f = 0 oznacza, że operacja przebiegła pomyślnie

if(err_f != 0)
{
    printf("Błąd otwarcia pliku %d \n", err_f); // możemy przeczytać kod błędny
    exit(-1); // zakończenie programu
}

// Tu wykonujemy operacje na pliku (w naszym przypadku zapis)
// Gdy plik już nie będzie więcej potrzebny w naszym programie,
// koniecznie go zamykamy!

fclose(f);
```

Pliki można otworzyć 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/czytany plik ma być obsługiwany w trybie tekstowym czy binarnym. Służą do tego odpowiednio sekwencje `t` i `b`. Przykładowe instrukcje zaprezentowano poniżej. Zauważmy też, że można otworzyć w tym samym czasie kilka plików.

```
int main()
{
    int a = 3;

    FILE *f, *g, *InnyPlik;
    errno_t err_f = fopen_s("plik1.txt", "wt"); // Zapis w trybie tekstowym
    errno_t err_h = fopen_s("plik2.dat", "wb"); // Zapis w trybie binarnym
    errno_t err_InnyPlik = fopen_s("Dane.txt", "r"); // Czytanie z pliku

    if (err_f != 0 || err_h != 0 || err_InnyPlik != 0)
    {
        printf("Nie udało się otworzyć chociaż jednego z plików\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_s(InnyPlik, "%d", &a); // Wczytujemy z pliku Dane.txt
// liczbe całkowita i przypisujemy jej wartosc do zmiennej a

// Tu mozemy wykonac jeszcze inne operacje na otwartych plikach

fclose(f);
fclose(g);
fclose(InnyPlik);

exit(0);
}
```

W powyższym przykładzie zaprezentowaliśmy jednocześnie użycie funkcji `fscanf`, która działa analogicznie do dobrze już znanej 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 zbiera je w buforze, a gdy uzna że jest on już odpowiednio pełny to czyści go przepisywając dane na dysk. Funkcja `fclose(plik)` zapisuje dane z bufora do pliku, a następnie zamyka plik. Chcąc wywołać jedynie opróżnienie bufora należy użyć funkcji `fflush(plik)`.

Process ten ma istotne konsekwencje:

- jeżeli nastąpi "crash" programu przed wykonaniem komendy `fclose()`
- wyjmując pendrive z komputera bez korzystania z opcji "bezpieczne

## Uwaga

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

## Ćwiczenia

W praktyce inżynierskiej pliki często zawierają dane pochodzące z eksperymentu lub symulacji. Plik `przebieg.txt` (do [ściągnięcia tu](#)) zawiera fragment przebiegu czasowego wartości trzech składowych prędkości ( $u$ ,  $v$ ,  $w$ ) pochodzących z symulacji przepływu powietrza przez dużą turbinę wiatrową. Chwilowe wartości tych składowych zostały zebrane z punktu znajdującego się tuż za turbiną. Napisz program, który: - Otworzy plik. - Wczyta dane z pliku do trzech tablic  $u$ ,  $v$ ,  $w$  zadeklarowanych statycznie (każda o rozmiarze 2000 - za tydzień będzie o lepszej metodzie deklaracji dużych tablic). Czytanie zrealizuj z użyciem pętli `for`. Obejrzyj plik, aby przyjrzeć się, w jaki sposób ułożone są dane (każda z kolumn odpowiada jednej ze składowych prędkości ( $u$ ,  $v$ ,  $w$ ); kolejne wiersze odpowiadają kolejnym krokom czasowym). - Po wczytaniu wszystkich wartości do tablic obliczy średnią każdej ze składowych. Średnia wyrażona jest wzorem:

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

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

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

- Zapisze do innego pliku raport z obliczeń, w którym poda wszystkie obliczone wielkości oraz wydrukuje to samo na ekran. - Wczytaj też plik `przebieg.txt` do arkusza kalkulacyjnego i utwórz wykres obrazujący te przebiegi. Oceń krytycznie wyniki uzyskane swoim programem na podstawie obserwacji wykresu. Czy średnie i odchylenia standardowe mają wiarygodne wartości?

## Wskazówka

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

```
double suma = 0;

for(int i = 0; i<n; i++)
{
```

```
suma += a[i];  
}
```

To tylko wskazówka. Oczywiście musisz zmodyfikować powyższy kod tak, aby liczył sumy z powyższych wzorów.

## Ważne: Dalej o funkcjach

Wiemy, że funkcje mogą przyjmować argumenty. Dowiedzieliśmy się też, że funkcje mogą zwracać wartości. Zmodyfikuj swój kod tak, aby odpowiednie bloki instrukcji były realizowane w funkcjach `Srednia` i `OdchylenieStandardowe`. Powinny mieć takie nagłówki:

```
double Srednia(double *tablica, int n);  
double OdchStd(double *tablica, double WartoscSrednia, int n);
```

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

```
void main()  
{  
    (...) // Deklaracje i kod wczytujący dane  
  
    um = Srednia(u, n);  
    vm = Srednia(v, n);  
    wm = Srednia(w, n);  
  
    u_std = OdchStd(u, um, n);  
    v_std = OdchStd(v, vm, n);  
    w_std = OdchStd(w, wm, n);  
  
    (...) // 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 (naturalnie będzie to średnia wartość przeczytanych dotąd elementów). Wystarczy, że zrobisz to dla jednej składowej prędkości (np.  $u$ ). Możesz tę średnią też na bieżąco podczas czytania danych drukować na ekran. Na końcu porównaj wartość średniej uzyskanej w ten sposób z wartością policzoną *a posteriori* 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 czteroelementowy i wykonaj go krok po kroku na kartce.

## Pytanie

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