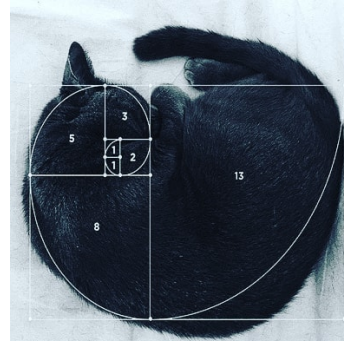


## Kot Leonarda



```
#ciąg_Fibonacci'ego
#tablica
#rekurencja
#pętla_for
```

Pod koniec XII w. w Pizie w rodzinie dyplomaty o nazwisku Bonacci, urodził się chłopiec o imieniu Leonardo. Z racji obowiązków służbowych ojca podróżował z nim po wielu krajach Afryki Północnej i Europy. Nie zaniedbywał wszakże nauki, pobierając lekcje od najlepszych nauczycieli. W tamtych czasach nauka arabska i hinduska (zwłaszcza matematyka) były absolutnie na topie, więc młody Leonardo mógł poznać ich najważniejsze tajniki. Po powrocie do rodzinnej Pizy sam zaczął publikować rozprawy matematyczne, w tym sławne *Liber abaci* (wymawiaj: *liber abaci*, z twardym **c**), w którym opisał między innymi ciąg liczb naturalnych nazwany później jego imieniem.\* Współczesna matematyka, a zwłaszcza arytmetyka, wiele zawdzięcza jego dziełom.

Ciąg Fibonacciego jest zdefiniowany w bardzo prosty sposób.† Oznaczmy jego kolejne wyrazy przez  $F_0, F_1, F_2, \dots$  i podamy taki oto przepis na ich obliczanie:

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2}, \quad \text{dla } n \geq 2. \end{aligned}$$

*Notabene*, taki wzór jest przykładem *metody rekurencyjnej*, czyli odwołującej się samej do siebie – temu zagadnieniu poświęćmy oddzielny podrozdział.

Kolejne wyrazy ciągu przedstawiają się następująco:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \dots$$

Spróbujemy napisać program, który oblicza i wypisuje pewną ilość wyrazów tego ciągu. Zrobimy to w dwóch etapach: najpierw wyliczymy wyrazy ciągu i zapiszemy je w tablicy, a następnie wypiszemy na ekranie.

Załóżmy, że chcemy wyliczyć liczby od  $F_0$  do  $F_{20}$  – oznacza to, że potrzebujemy miejsca na 21 liczb, czyli tablicy liczb całkowitych o takim właśnie rozmiarze (nazwiemy ją  $F[]$ , jak we wzorze).

\*Fibonacci (wymawiaj: *fibonaczcz*) – od łacińskiego *Filius Bonacci* czyli *syn Bonacciego*.

†Rzeczy fundamentalne często wykazują się bardzo prostą konstrukcją. Ciąg ten pojawia się w niezliczonych zagadnieniach matematycznych i informatycznych, ale także znajdziemy go w botanice i zoologii (choćby w budowie szyszek, słoneczników czy muszli ślimaka) oraz w sztuce (złoty podział, *Człowiek witraż* Leonarda da Vinci).

Zatem w naszym programie powinna znaleźć się taka oto deklaracja:

```
const int MAX{ 20 };
int F[MAX + 1];
```

Tablica (ang. **array**, wymawiaj: *arej*, z akcentem na *e*) jest podstawową strukturą danych dostępną w każdym języku programowania. Komórki tablicy o rozmiarze  $d$  ponumerowane są od 0 do  $d - 1$ , jak w znanym nam już ciągu znaków (**string**). Jednak dla tablic jako takich nie ma funkcji `length()` (czy `size()`), zwracającej rozmiar tablicy.

Rozmiar tablicy jest wielkością stałą (stąd `const`, wymawiaj: *konst*), jednak zdefiniowanie i dalsze konsekwentne używanie stałej `MAX` jest zalecanym zwyczajem, gdyż kompilator wykryje na przykład próbę jej zmiany poniżej w kodzie, a gdyby nam przyszła ochota wyliczyć inną ilość tych liczb, wtedy łatwo o korektę. Tak samo nazewnictwo: nazwy stałych piszemy wielkimi literami.

Początkowe dwie wartości w tablicy wpisujemy ręcznie:

```
F[0] = 0;
F[1] = 1;
```

Każdą następną wartość będziemy już wyliczać jako sumę dwóch poprzednich, czyli:

```
F[n] = F[n - 1] + F[n - 2];
```

Tę instrukcję umieścimy w pętli i jest to okazja, aby poznać nowy rodzaj pętli, niezwykle często używany przez programistów: pętlę `for` (z ang. *dla*). Jej składnia jest następująca:

```
for(instrukcja1; warunek; instrukcja2)
{
    treść_pętli
}
```

Co to jest *treść\_pętli*, to chyba jasne. Otaczające ją nawiasy klamrowe można pominąć, jeśli w treści pętli mamy pojedynczą instrukcję.

W każdej pętli występuje warunek kontynuacji – tutaj *warunek*, czyli zdanie logiczne – póki prawdziwe, póty pętla się kręci. W naszym przypadku warunek powinien mieć postać:

```
n <= MAX
```

bo w takim zakresie obliczamy wyrazy ciągu.

Czas na część sterującą pętli: *instrukcja1* jest wykonywana przed rozpoczęciem wykonywania całej pętli, zatem na ogół zawiera ona deklarację i inicjalizację potrzebnych zmiennych. W tym przykładzie mogłaby wyglądać tak:

```
int n{ 2 }
```

Z kolei *instrukcja2* jest wykonywana po każdym obiegu pętli, czyli po każdorazowym wykonaniu jej treści. Narzuca się, aby zawrzeć w niej powiększanie (lub pomniejszanie) istotnych zmiennych o 1, choć oczywiście można tu wstawić cokolwiek. W naszym przypadku użyjemy wyrażenia `n++`.

Cała pętla powinna wyglądać tak:

```
for(int n{ 2 }; n <= MAX; n++)
    F[n] = F[n - 1] + F[n - 2];
```

Po wygenerowaniu wszystkich (zamierzonych) liczb Fibonacciego powinniśmy je wypisać na ekranie. Użyjemy do tego analogicznej pętli `for`, ale z wartością początkową  $n = 0$ .

Możemy więc już przedstawić pełny program dla tego problemu:

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    const int MAX{ 20 };
    int F[MAX + 1];
    F[0] = 0;
    F[1] = 1;
    for(int n{ 2 }; n <= MAX; n++)
        F[n] = F[n - 1] + F[n - 2];
    for(int n{ 0 }; n <= MAX; n++)
        cout << n << ' ' << F[n] << '\n';
    return 0;
}
```

Po uruchomieniu programu na ekranie zobaczymy takie liczby:

```
0 0
1 1
2 1
3 2
4 3
5 5
6 8
7 13
8 21
9 34
10 55
11 89
12 144
13 233
14 377
15 610
```

16 987  
17 1597  
18 2584  
19 4181  
20 6765