## Podsumowanie

- Problemy optymalizacyjne to problemy, w których szukamy rozwiązania jak najlepszego, spełniającego określone kryterium.
- Podejście zachłanne w rozwiązywaniu problemu polega na podejmowaniu szeregu decyzji, z których każda wydaje się najkorzystniejsza w danym momencie, bez analizowania ich wpływu na optymalność całego rozwiązania. Stosowanie tej strategii prowadzi czasami do rozwiązania optymalnego, a czasami tylko do rozwiązania przybliżonego.
- Problem wydawania reszty polega na przedstawieniu danej kwoty reszty za pomocą jak najmniejszej liczby banknotów i monet. Aby rozwiązać ten problem metodą zachłanną, należy wybierać w danym kroku jak największy nominał nie większy od kwoty, która pozostała jeszcze do wydania.
- Aby zapobiec kumulowaniu się błędów przybliżeń w języku C++, należy unikać używania typów zmiennoprzecinkowych, np. **`float`** i **`double`**.

## Zadania

* **1** Napisz program, który liczbę wpisaną z klawiatury, mniejszą niż 256, zapisze jako sumę potęg liczby 2. Zdefiniuj w kodzie tablicę o wartościach 128, 64, 32, 16, 8, 4, 2, 1.

* **2** W pliku, który otrzymasz od nauczyciela (np. *mapy_konturowe.pdf*), znajdują się różne mapy. Korzystając z metody zachłannej, wyznacz minimalną liczbę kolorów, której trzeba użyć do pokolorowania każdej mapy tak, aby każde dwa sąsiednie obszary miały różne kolory.

* **3** Napisz program poszukujący rozwiązania problemu wydawania reszty, gdy masz ograniczoną liczbę każdego z nominałów. Zwróć uwagę, że nie każdą resztę uda się wydać (program powinien wtedy wypisać odpowiedni komunikat).

* **4** Zmodyfikuj kod źródłowy programu *Problem kinomana* tak, aby dane wczytywać z klawiatury.

** **5** Napisz program, który wczyta z klawiatury informacje o kolegach i koleżankach z klasy: imię, pseudonim oraz kolor oczu. Po wprowadzeniu danych program powinien uporządkować je według koloru oczu i wyświetlić informacje o osobach rozdzielone na grupy w zależności od koloru oczu. Grupa najliczniejsza powinna być wyświetlona na początku, a grupa najmniej liczna – na końcu. Zastosuj tablice równoległe.

*** **6** Na świecie rzadko stosuje się systemy monetarne, w których nie ma nominału 2. Przykładem jest szyling kenijski, którego monety mają wartości: 1, 5, 10 i 20 szylingów, a banknoty: 50, 100, 200, 500 i 1000 szylingów. Napisz program wydający resztę w tym systemie z użyciem podejścia zachłannego. Czy algorytm zachłanny jest w tym wypadku optymalny?



# 5. Rekurencja

W programowaniu rozwiązań problemów stosowaliśmy do tej pory iterację, czyli technikę polegającą na wielokrotnym powtarzaniu – w pętli – instrukcji zapisanych w tekście programu. Istnieje jednak inna technika programistyczna, która również wiąże się z powtarzaniem tego samego fragmentu kodu, choć instrukcja pętli jawnie w kodzie źródłowym nie występuje. Jest to rekurencja. W tym temacie dowiesz się, czym jest funkcja rekurencyjna, jakie ma zalety, a także dlaczego należy jej używać z rozwagą.

### Cele lekcji

- Dowiesz się, czym jest rekurencja, oraz zrozumiesz istotę rekurencyjnego rozwiązywania problemów.
- Poznasz przykład prostego fraktala w postaci drzewa binarnego.
- Nauczysz się zapisywać funkcje rekurencyjne w języku C++.
- Poznasz ograniczenia, jakie ma stosowanie techniki rekurencji.
- Nauczysz się na przykładach, jak rekurencję zastąpić iteracją i na odwrót.

## 5.1. Czym jest rekurencja?

**Rekurencja** lub **rekursja** (ang. *recursion*) w programowaniu oznacza odwołanie się w wybranym kroku algorytmu rozwiązywania problemu do tego samego problemu, ale dla mniejszego rozmiaru danych. Algorytm, w którym stosuje się takie podejście, nazywamy **algorytmem rekurencyjnym**.

Rekurencja (rekursja)

Algorytm rekurencyjny

Podejście rekurencyjne w rozwiązywaniu problemu algorytmicznego jest procesem dwufazowym. Pierwsza faza to redukcja rozmiaru problemu, druga to łączenie częściowych wyników pochodzących z rozwiązania tych mniejszych problemów.

Jeśli danego problemu nie potrafimy rozwiązać, gdyż $n$-elementowy zbiór danych jest zbyt złożony, to umiejętnie redukujemy go do podobnego problemu, ale dla zbioru o np. $n - 1$ elementach. Taki problem może być łatwiej rozwiązać. Jeśli ten mniejszy problem wciąż jest zbyt złożony, to możemy dalej redukować jego złożoność, aż dojdziemy do takiego problemu, który potrafimy rozwiązać od razu. Później, w fazie zbierania i łączenia częściowych wyników, uzyskujemy rozwiązanie całego problemu.

Aby lepiej zrozumieć rekurencję, przyjrzyjmy się różnym jej przykładom.

**Warto wiedzieć**

Efekt podobny do rekurencyjnego można uzyskać, wykorzystując lustra i aparat fotograficzny w sposób pokazany na zdjęciu. Można też stanąć plecami do dużego lustra i zrobić zdjęcie przednim aparatem telefonu (tzw. selfie).

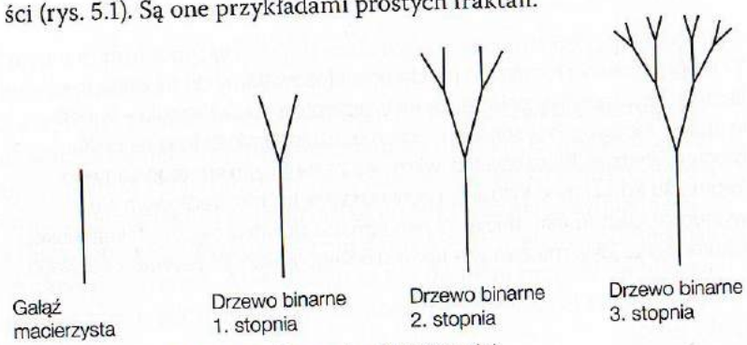
### Rekurencyjna budowa drzewa binarnego

**Fraktal** to figura geometryczna, której podstawową cechą jest samopodobieństwo (podobieństwo wewnętrzne, powtarzalność). Oznacza to, że we fraktalu można wyodrębnić fragmenty podobne do całej figury.

Fraktal

Drzewo binarne

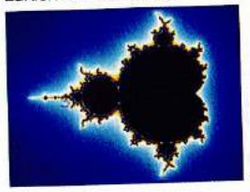
Oto kilka przykładów **drzew binarnych** o różnym stopniu złożoności (rys. 5.1). Są one przykładami prostych fraktali.



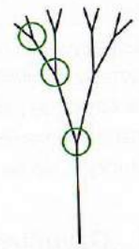

Rys. 5.1. Drzewa binarne o różnym stopniu złożoności

**Warto wiedzieć**

Pojęcie fraktala wprowadził w latach 70. XX w. Benoît B. Mandelbrot, urodzony w Warszawie w rodzinie litewskich Żydów, którzy po I wojnie światowej zamieszkali w Polsce. Jednym z najbardziej znanych fraktali jest zbiór Mandelbrota, nazywany również żukiem Mandelbrota.



Drzewo 1. stopnia składa się z dwóch gałęzi odchodzących od gałęzi macierzystej. Gałęzie te mają długość równą 0,6 długości gałęzi macierzystej i są odchylone od kierunku przez nią wyznaczonego o ten sam kąt – 15 stopni w prawo i w lewo. Gałęzie końcowe drzewa 2. stopnia mają te same parametry w odniesieniu do gałęzi drzewa 1. stopnia. Analogicznie są zbudowane drzewa kolejnych stopni.



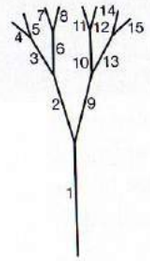
Rys. 5.2. Części samopodobne drzewa binarnego

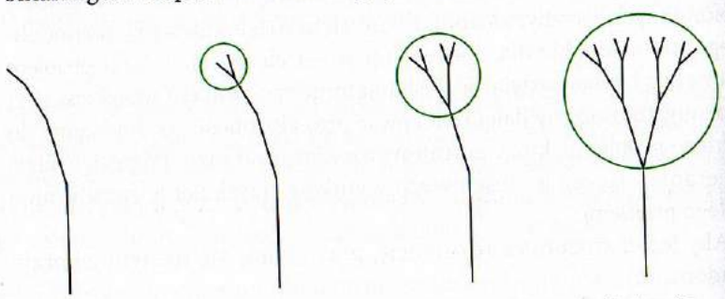
Rysunek 5.2 przedstawia drzewo binarne z zaznaczonymi miejscami rozwidlenia, które wskazują na cechę samopodobieństwa: fragmenty z wyższych części drzewa są podobne do całości. Drzewo 3. stopnia składa się z gałęzi macierzystej oraz dwóch drzew 2. stopnia odpowiednio pomniejszonych i obróconych.

Programy rysujące fraktale najczęściej wykorzystują funkcje rekurencyjne. Rysunek 5.3 przedstawia wybrane etapy rysowania drzewa binarnego 3. stopnia realizowanego przez funkcję rekurencyjną.

**Warto wiedzieć**

Odcinki tworzące drzewo binarne 3. stopnia są rysowane w następującej kolejności:



Rys. 5.3. Proces rekurencyjnego rozrastania się drzewa binarnego 3. stopnia

Najpierw rysowana jest gałąź macierzysta. Następnie powstaje drzewo 2. stopnia (w odpowiedniej skali) odchylone w lewo, a dopiero po jego narysowaniu – drzewo 2. stopnia odchylone w prawo.





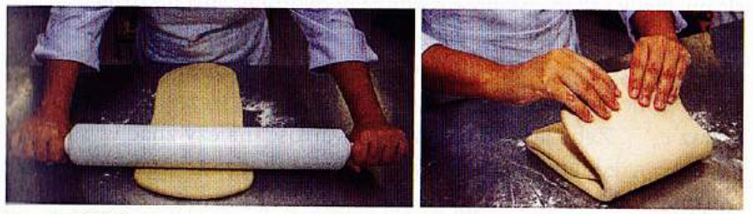
Aby narysować każde z drzew 2. stopnia, należy w podobny sposób narysować drzewa 1. stopnia (rozpoczynając od lewej strony).

Drzewo binarne rozrasta się w sposób rekurencyjny: z każdego rozwidlenia wyrastają dokładnie dwa drzewa niższego stopnia.

### Rekurencyjny przepis na ciasto francuskie

Ciasto francuskie charakteryzuje się dużą ilością zawartego w nim powietrza. Zawdzięczamy to warstwowej strukturze uzyskiwanej przez wielokrotne składanie ciasta (rys. 5.4).

Aby przygotować ciasto francuskie, należy na pewnym etapie wielokrotnie wykonywać takie same polecenia: ciasto rozwałkuj na długość równą 3-krotności jego szerokości. Następnie złóż je na 2 razy, aby uzyskać 3 warstwy i schłodź. Powtórz te czynności jeszcze 5 razy.



Rys. 5.4. Wałkowanie i składanie ciasta francuskiego

**Warto wiedzieć**

Gdyby fragment przepisu na ciasto francuskie dotyczący wałkowania, składania i chłodzenia ciasta zapisać nieco bardziej formalnie, mógłby on brzmieć następująco:

```
Rozwałkuj-Złóż-Schłodź(n):
  jeśli n = 1, to zakończ
  rozwałkuj ciasto
  złóż je na 3 warstwy
  schłodź przez 1 godzinę
  Rozwałkuj-Złóż-Schłodź(n-1)
```

Zauważ, że tak sformułowany opis odwołuje się do samego siebie w słowach „Powtórz te czynności". Jest to cecha każdego algorytmu rekurencyjnego.

**Ćwiczenie 1**

Tabliczka czekolady składa się z 16 kawałków (4 rzędy po 4 kawałki). Zapisz rekurencyjny algorytm łamania czekolady na 16 pojedynczych kostek.

**A to ciekawe**

## Efekt Droste – rekurencja w projektowaniu

Około 1900 r. na puszce kakao produkowanego przez holenderską firmę cukierniczą Droste pojawił się wizerunek pielęgniarki, która trzymała w ręku tacę z filiżanką oraz puszką kakao Droste. Na tej puszce widać pielęgniarkę trzymającą w ręku tacę z filiżanką oraz puszką kakao Droste itd. Ten sposób prezentacji grafiki został pod koniec lat 70. XX w. nazwany efektem Droste. Jest to rodzaj rekurencyjnego obrazu. Również dziś można zaobserwować ten zabieg m.in. na puszce polskiego mleka zagęszczonego.

## 5.2. Rekurencyjna definicja ciągu

**Warto wiedzieć**

Ciąg $(c_n)$ jest ciągiem geometrycznym. Do obliczania jego wyrazów można więc użyć wzoru: $c_n = 3^{n-1}$

Wróćmy raz jeszcze do przykładu ciasta francuskiego. Ciasto powstałe jako efekt pierwszego wałkowania możemy nazwać ciastem początkowym o numerze 1. Z niego powstaje 3-warstwowe ciasto o numerze 2, a z niego 9-warstwowe ciasto o numerze 3 itd.

Zauważmy, że $i$-te ciasto powstaje z ciasta o numerze $i - 1$, które po złożeniu na trzy części zostaje rozwałkowane. Ciąg opisujący liczbę warstw ciasta to: 1, 3, 9, 27, ... Z matematycznego punktu widzenia można

Ciąg zdefiniowany rekurencyjnie

mówić o ciągu $c_n$ **zdefiniowanym rekurencyjnie** w następujący sposób:

$$c_n = \begin{cases} 1 & \text{dla } n = 1 \\ 3 \cdot c_{n-1} & \text{dla } n > 1 \end{cases}$$

Warunek początkowy

Pierwszą część definicji nazwiemy **warunkiem początkowym**. Określa on, kiedy zakończyć proces odwołań rekurencyjnych z drugiej części definicji. Zgodnie z drugą częścią definicji kolejny wyraz ciągu wyznacza się, korzystając z wartości wyrazu poprzedniego.

Aby obliczyć wartość $c_n$ dla danego $n$, należy wcześniej wyznaczyć wyraz $c_{n-1}$. Żeby jednak obliczyć tę wartość, należy znać wyraz $c_{n-2}$ itd. Proces redukowania wartości indeksu $n$ skończy się, gdy $n$ przyjmie wartość 1 (warunek początkowy).

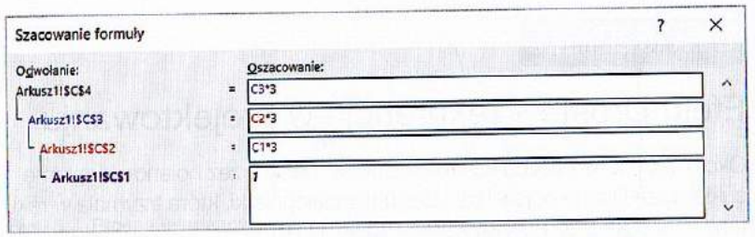
Wyrazy ciągów zdefiniowanych rekurencyjnie można łatwo obliczyć w arkuszu kalkulacyjnym. Dla powyższego ciągu wystarczy wpisać w komórce C1 wartość wyrazu początkowego, czyli 1, a w komórce C2 – formułę opisującą zależność rekurencyjną: =3∗C1, i skopiować formułę w dół. Wartość w komórce C4 (rys. 5.5) jest obliczana błyskawicznie. Arkusz wyznacza ją jednak krok po kroku, ponieważ formuła w komórce C4 odwołuje się do formuły z komórki C3, która odwołuje się do formuły z komórki C2 itd. Łańcuch kolejnych wywołań rekurencyjnych (rys. 5.6) kończy się w komórce C1, zawierającej stałą wartość. Dalej następuje faza zbierania wyników, która kończy się wstawieniem do komórki C4 wyniku 27.

| | C |
|---|---|
| 1 | 1 |
| 2 | 3 |
| 3 | 9 |
| 4 | =3*C3 |

Rys. 5.5. Ciąg w arkuszu kalkulacyjnym

**Dobra rada**

Jeśli chcesz prześledzić krok po kroku obliczenia wykonywane w formułach arkusza kalkulacyjnego, skorzystaj z narzędzia **Szacuj formułę**, dostępnego na karcie **Formuły**.




Rys. 5.6. Podgląd wywołań rekurencyjnych w narzędziu **Szacuj formułę**

Napiszemy teraz programy obliczające wybrany wyraz ciągu rekurencyjnego. Sformułujmy specyfikację problemu.





**Specyfikacja**

**Dane:** liczba naturalna $n$.

**Wynik:** $n$-ty wyraz ciągu danego wzorem:

$$c_n = \begin{cases} 1 & \text{dla } n = 1 \\ 3 \cdot c_{n-1} & \text{dla } n > 1 \end{cases}$$

### Program *Ciąg rekurencyjnie*

Zapiszmy funkcję o nazwie CiagRek, która będzie wyznaczała $n$-ty wyraz ciągu w sposób rekurencyjny. Oto jej kod źródłowy:

Kod źródłowy funkcji CiagRek w programie *Ciąg rekurencyjnie*

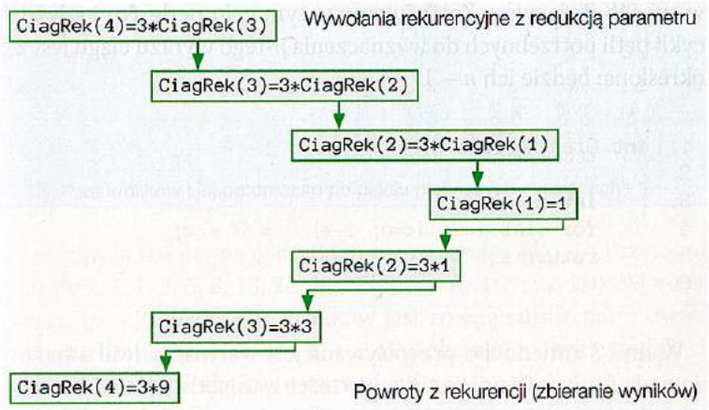
```
1. int CiagRek(int n)
2. {
3.     if (n == 1) return 1;
4.     return 3 * CiagRek(n-1);
5. }
```

Budowa wnętrza **funkcji rekurencyjnej** odzwierciedla dwa możliwe przypadki. Pierwszy to warunek początkowy (linia 3), drugi to kod **wywołania rekurencyjnego**, czyli wywołania funkcji przez samą siebie, ale dla innej wartości parametru (linia 4).

Funkcja rekurencyjna

Wywołanie rekurencyjne

Na rysunku 5.7 przedstawiono fazy działania funkcji CiagRek: wywołania rekurencyjne dla coraz mniejszych wartości parametru $n$ oraz powroty z rekurencji (zbieranie wyników).




Rys. 5.7. Schemat wywołań rekurencyjnych funkcji CiagRek i powrotów z rekurencji

W pierwszej części schematu strzałki wskazują kolejne wywołania rekurencyjne funkcji CiagRek dla coraz mniejszych parametrów. Druga część schematu to faza zbierania wyników – strzałki pokazują, które wartości są przekazywane dalej.

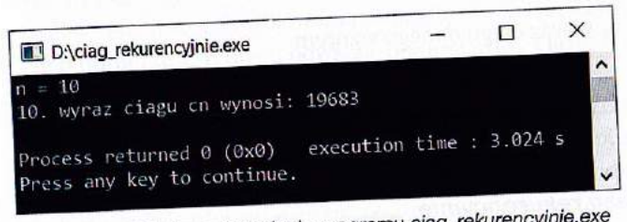
**Dobra rada**

Nazwa „rekurencja" pochodzi od łacińskiego słowa *recurrere*, czyli „przybiec z powrotem". Jeśli chcesz zrozumieć daną funkcję rekurencyjną, przeanalizuj dokładnie właśnie powroty z rekurencji.

**Warto wiedzieć**

Kody źródłowe funkcji, które są zapisem algorytmów rekurencyjnych, same mają strukturę podobną do rekurencyjnej definicji ciągu. Składają się z dwóch części: pierwsza określa warunek początkowy, a druga opisuje, jak przypadek dla parametru $n$ zredukować do prostszego (np. $n - 1$).

Rysunek 5.8 przedstawia przykładowe wywołanie programu *Ciąg rekurencyjnie*.




**Rys. 5.8.** Przykładowe wywołanie programu *ciag_rekurencyjnie.exe*

### Ćwiczenie 2

**a.** W pliku otrzymanym od nauczyciela (np. *wyraz_ciagu_rek.cpp*) znajduje się fragment kodu źródłowego zawierający funkcję `CiagRek`. Dopisz kod funkcji głównej, skompiluj kod programu i sprawdź jego działanie.

**b.** Napisz program, który obliczy wartość $n$-tego wyrazu ciągu $a_n$ danego wzorem:

$$a_n = \begin{cases} 6 & \text{dla } n = 1 \\ 2 \cdot a_{n-1} & \text{dla } n > 1 \end{cases}$$

Sprawdź działanie programu i podaj wynik dla $n = 12$

## Program *Ciąg iteracyjnie*

Pętla `for`, s. 247

Wyraz ciągu $c_n$ zdefiniowanego rekurencyjnie można również obliczyć w sposób iteracyjny. Zastosujemy w tym celu pętlę `for`, gdyż liczba cykli pętli potrzebnych do wyznaczenia $n$-tego wyrazu ciągu jest z góry określona: będzie ich $n - 1$.

Kod źródłowy funkcji `CiagIter` w programie *Ciąg iteracyjnie*

```
1. int CiagIter(int n)
2. {
3.     int c = 1;
4.     for (int i=2; i<=n; i++) c = 3 * c;
5.     return c;
6. }
```

**Dobra rada**

Instrukcję przypisania `c = 3 * c;` możesz zapisać krócej jako `c *= 3;`.

W linii 3 zmiennej `c` przypisywana jest wartość 1. Jeśli użytkownik wywoła funkcję `CiagIter` dla wartości 1, to pętla z linii 4 nie wykona się ani razu, ponieważ nie będzie spełniony warunek `i<=n`. W pozostałych przypadkach pętla wykona się $n - 1$ razy.

### Ćwiczenie 3

Zapisz kod źródłowy programu *Ciąg iteracyjnie*. Skompiluj kod i uruchom program dla różnych wartości.





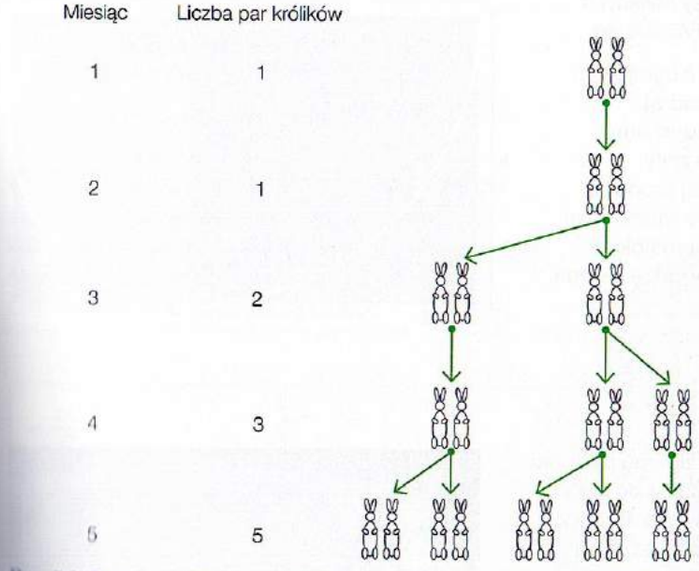
## 5.3. Ciąg Fibonacciego

W 1202 r. Leonardo z Pizy, zwany Fibonaccim, przedstawił zadanie problemowe, które opisywało wzrost populacji królików. Pewien gospodarz kupił parę królików. Ile par królików w sumie będzie miał po 12 miesiącach, jeśli każda para rodzi co miesiąc po jednej parze, która staje się zdolna do rozmnażania po miesiącu życia?

Przeanalizujmy, jak rośnie liczba par królików w hodowli. W pierwszym miesiącu mamy parę młodych królików, która na początku drugiego miesiąca staje się płodna. Na początku trzeciego miesiąca tej parze urodzi się para młodych. Razem będą więc 2 pary królików. W czwartym miesiącu ze starszej pary zrodzi się kolejna para – mamy więc 2 + 1 = 3 pary. Ile par królików będzie na początku kolejnego, tj. piątego miesiąca? Wyjaśnia to rysunek 5.9.

**Warto wiedzieć**

Leonardo z Pizy (1175–1250) był włoskim matematykiem. W młodości towarzyszył ojcu, który był kupcem, w podróżach handlowych i uczył się matematyki u arabskich nauczycieli. W 1202 r. opublikował swoje pierwsze dzieło matematyczne *Liber abaci* (*Księga rachunków*). Zawierało ono prawdopodobnie całą ówczesną wiedzę o arytmetyce. Na stronie tytułowej Leonardo podpisał się *filius Bonacci* (syn Bonacciego). W XIX w. historycy nadali mu przydomek Fibonacci.




**Rys. 5.9.** Para królików i jej potomstwo po pięciu pierwszych miesiącach

Oznaczmy liczbę par na początku $n$-tego miesiąca jako $f_n$. Otrzymamy ciąg liczb: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... (patrz rys. 5.9). W $n$-tym miesiącu ($n > 2$) liczba par królików jest równa sumie par z dwóch poprzednich miesięcy: $n - 2$ i $n - 1$. Współcześnie zwykle dodaje się na początku ciągu dodatkowy wyraz $f_0 = 0$.

Otrzymany ciąg nazywamy **ciągiem Fibonacciego**. Początkowe dwa wyrazy tego ciągu to 0 i 1, a każdy następny jest sumą poprzednich dwóch. Ciąg Fibonacciego ma następującą definicję rekurencyjną:

Ciąg Fibonacciego

$$f_n = \begin{cases} 0 & \text{dla } n = 0 \\ 1 & \text{dla } n = 1 \\ f_{n-1} + f_{n-2} & \text{dla } n > 1 \end{cases}$$