

ALEXANDER A. STEPANOV
DANIEL E. ROSE



OD
MATEMATYKI
DO
PROGRAMOWANIA
UOGÓLNIONEGO

Helion 

Tytuł oryginału: From Mathematics to Generic Programming

Tłumaczenie: Zdzisław Płoski

ISBN: 978-83-283-1028-5

Authorized translation from the English language edition, entitled: FROM MATHEMATICS TO GENERIC PROGRAMMING; ISBN 0321942043; by Alexander A. Stepanov; and by Daniel E. Rose; published by Pearson Education, Inc, publishing as Addison Wesley.

Copyright © 2015 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Polish language edition published by HELION S.A. Copyright © 2015.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/odmado>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Podziękowania	7
O autorach	9
Nota od autorów	11

1. O czym jest ta książka 13

1.1. Programowanie a matematyka	14
1.2. Perspektywa historyczna	15
1.3. Wymagania	15
1.4. Przewodnik	16

2. Pierwszy algorytm 19

2.1. Mnożenie po egipsku	20
2.2. Ulepszenie algorytmu	23
2.3. Przemyslenia związane z rozdziałem	26

3. Teoria liczb według starożytnych Greków 27

3.1. Geometryczne proporcje liczb całkowitych	27
3.2. Odsiewanie liczb pierwszych	30
3.3. Implementacja i optymalizacja kodu	33
3.4. Liczby doskonałe	38
3.5. Program pitagorejski	42
3.6. Fatalny błąd w tym programie	43
3.7. Przemyslenia związane z rozdziałem	47

4. Algorytm Euklidesa 49

4.1. Ateny i Aleksandria	49
4.2. Algorytm Euklidesa znajdowania największej wspólnej miary	51
4.3. Tysiąc lat bez matematyki	57
4.4. Dziwna historia zera	58

4.5. Algorytmy obliczania reszty i ilorazu	60
4.6. Współużytkowanie kodu	63
4.7. Uprawnoczenie tego algorytmu	65
4.8. Przemyslenia związane z rozdziałem	67
5. Pojawienie się nowoczesnej teorii liczb	69
5.1. Liczby pierwsze Mersenne'a i liczby pierwsze Fermata	69
5.2. Małe twierdzenie Fermata	74
5.3. Skracanie	78
5.4. Udowodnienie małego twierdzenia Fermata	82
5.5. Twierdzenie Eulera	84
5.6. Zastosowania arytmetyki modularnej	88
5.7. Przemyslenia związane z rozdziałem	89
6. Abstrakcja w matematyce	91
6.1. Grupy	91
6.2. Monoidy i półgrupy	95
6.3. Niektóre twierdzenia o grupach	98
6.4. Podgrupy i grupy cykliczne	101
6.5. Twierdzenie Lagrange'a	103
6.6. Teorie i modele	107
6.7. Przykłady teorii kategoriycznych i niekategoriycznych	110
6.8. Przemyslenia związane z rozdziałem	113
7. Wyprowadzenie algorytmu uogólnionego	115
7.1. Rozwikłanie wymagań dotyczących algorytmu	115
7.2. Wymagania dotyczące A	116
7.3. Wymagania dotyczące N	120
7.4. Nowe wymagania	122
7.5. Zamiana mnożenia na potęgowanie	123
7.6. Uogólnianie operacji	124
7.7. Obliczanie liczb Fibonacciego	127
7.8. Przemyslenia związane z rozdziałem	130
8. Więcej struktur algebraicznych	131
8.1. Wielomiany Stevina i NWD	131
8.2. Getynga i matematyka niemiecka	137
8.3. Noether i narodziny algebry abstrakcyjnej	142
8.4. Pierścienie	144
8.5. Mnożenie macierzy i półpierścienie	147
8.6. Zastosowanie: sieci społeczne i najkrótsze ścieżki	149
8.7. Dziedziny euklidesowe	151
8.8. Ciała i inne struktury algebraiczne	152
8.9. Przemyslenia związane z rozdziałem	154

9. Uporządkowanie wiedzy matematycznej	157
9.1. Dowody	157
9.2. Pierwsze twierdzenie	160
9.3. Euklides i metoda aksjomatyczna	163
9.4. Geometrie alternatywne wobec euklidesowej	165
9.5. Formalistyczne podejście Hilberta	168
9.6. Peano i jego aksjomaty	171
9.7. Budowanie arytmetyki	174
9.8. Przemyslenia związane z rozdziałem	177
10. Podstawowe koncepcje programowania	179
10.1. Arystoteles i abstrakcja	179
10.2. Wartości i typy	182
10.3. Koncepty	183
10.4. Iteratory	187
10.5. Kategorie, cechy i operacje iteratorowe	188
10.6. Przedziały	191
10.7. Wyszukiwanie liniowe	193
10.8. Wyszukiwanie binarne	194
10.9. Przemyslenia związane z rozdziałem	199
11. Algorytmy permutacyjne	201
11.1. Permutacje i transpozycje	201
11.2. Zamiana przedziałów	205
11.3. Rotacja	208
11.4. Zastosowanie cykli	211
11.5. Odwracanie	215
11.6. Złożoność przestrzenna	218
11.7. Algorytmy dostosowujące się do pamięci	219
11.8. Przemyslenia związane z rozdziałem	220
12. Rozszerzenia NWD	221
12.1. Ograniczenia sprzętowe i efektywniejsze algorytmy	221
12.2. Uogólnienie algorytmu Steina	224
12.3. Tożsamość Bézouta	227
12.4. Rozszerzony NWD	231
12.5. Zastosowania NWD	235
12.6. Przemyslenia związane z rozdziałem	236
13. Zastosowanie praktyczne	237
13.1. Kryptologia	237
13.2. Sprawdzanie pierwszośc	240
13.3. Test Millera-Rabina	243
13.4. Algorytm RSA — jak działa i dlaczego	245
13.5. Przemyslenia związane z rozdziałem	248

14. Wnioski 249**Lektury uzupełniające 251****A Notacja 257****B Typowe techniki dowodowe 261**

- B.1. Dowód nie wprost 261
- B.2. Dowód przez indukcję 262
- B.3. Zasada klatek w gotębniku 263

C C++ dla nieprogramujących w C++ 265

- C.1. Funkcje szablonowe 265
- C.2. Koncepty 266
- C.3. Składnia deklaracji i stałe z typami 267
- C.4. Obiekty funkcyjne 268
- C.5. Warunki początkowe i końcowe oraz asercje 269
- C.6. Algorytmy STL i struktury danych 269
- C.7. Iteratory i przedziały 271
- C.8. Zastosowanie synonimów i funkcji typów w C++11 272
- C.9. Listy inicjatorów w C++11 272
- C.10. Funkcje lambda w C++11 273
- C.11. Uwaga o podprogramach otwartych 273

Literatura 275

Skorowidz 279

Źródła materiału zdjęciowego 285



Pierwszy algorytm

*Mojesz szybko uczył się arytmetyki i geometrii.
[...] Wiedzę tę czerpał od Egipcjan,
którzy przedkładali naukę matematyki ponad wszystko.
Filon z Aleksandrii, Żywot Mojżesza*

Algorytm jest skończonym ciągiem kroków składających się na wykonanie zadania obliczeniowego. Algorytmy są tak blisko związane z programowaniem komputerów, że wielu ludzi znających to pojęcie przyjmuje zapewne, że idea algorytmu pochodzi z informatyki. Jednakże algorytmy istnieją od — dosłownie — tysięcy lat. W matematyce jest mnóstwo algorytmów, a niektórymi z nich posługujemy się na co dzień. Nawet sposób dodawania długich liczb, którego uczą się dzieci w szkole, jest algorytmem.

Mimo swej długiej historii pojęcie algorytmu nie istniało od zawsze; musiało zostać wynalezione. Choć nie potrafimy określić, kiedy powstały pierwsze algorytmy, wiemy, że niektóre były znane w Egipcie już przed co najmniej czterema tysiącami lat.

* * *

Cywilizacja starożytnych Egipcjan skupiała się wzdłuż Nilu, a ich rolnictwo zależało od użyźniających glebę wylewów tej rzeki. Rodziło to trudność tego rodzaju, że po każdej powodzi wszystkie oznakowania granic poszczególnych pól były zmywane przez wodę. Do mierzenia odległości Egipcjanie używali sznurów i opracowali procedury umożliwiające powracanie do stanu zapisanego w rejestrach nieruchomości i odtwarzanie ich granic. Za zadanie to odpowiadała wybrana grupa kapłanów biegłych w owych matematycznych metodach; zwano ich „rozciągaczami sznurów”. W późniejszych czasach Grecy zaczęli ich nazywać **geometrami**, czyli „mierniczymi Ziemi”.

Niestety, dysponujemy niewieloma zapisami dokumentującymi matematyczną wiedzę Egipcjan. Z tego okresu przetrwały tylko dwa źródła. Jednym z nich, na którym się skoncentrujemy, jest matematyczny papirus Rhinda, nazywany tak od nazwiska XIX-wiecznego

szkockiego kolekcjonera, który kupił go w Egipcie. Jest to dokument z około 1650 roku przed naszą erą napisany przez skrybę o imieniu Ahmes. Zawiera serię problemów arytmetycznych i geometrycznych oraz kilka tabel służących do obliczeń. W tym zwoju zapisano pierwszy odnotowany algorytm, metodę szybkiego mnożenia, a także drugi — do szybkiego dzielenia. Zaczniemy od przyjrzenia się algorytmowi szybkiego mnożenia, który (jak zobaczymy w dalszej części książki) aż do dzisiaj jest ważną metodą obliczeniową.

2.1. Mnożenie po egipsku

W egipskim systemie liczbowym, tak jak we wszystkich używanych przez starożytne cywilizacje, nie posługiwano się zapisem pozycyjnym i nie istniał sposób reprezentowania zera. Wskutek tego mnożenie było niezwykle trudne i tylko niewielu wyćwiczonych rachmistrzów potrafiło je wykonywać. (Wyobraźmy sobie mnożenie dużych liczb, gdy mamy do dyspozycji tylko coś w rodzaju rzymskich liczebników).

Jak definiujemy mnożenie? Nieformalnie jest to „dodawanie czegoś do siebie pewną liczbę razy”. Formalnie możemy zdefiniować mnożenie, rozbijając je na dwa przypadki: mnożenie przez 1 i mnożenie przez liczbę większą od 1.

Mnożenie przez 1 definiujemy w ten sposób:

$$1a = a. \quad (2.1)$$

Teraz mamy do czynienia z przypadkiem, w którym chcemy obliczyć iloczyn liczniejszy o jeden od już obliczonego. Niektórzy czytelnicy mogą dopatrzeć się w tym postępowania indukcyjnego; później skorzystamy z tej metody bardziej formalnie.

$$(n + 1)a = na + a. \quad (2.2)$$

Jednym ze sposobów mnożenia a przez n jest n -krotne dodanie a do siebie. Jednak w wypadku dużych liczb może to być skrajnie żmudne, gdyż wymaga $n - 1$ dodawań. W C++ taki algorytm wygląda jak niżej¹:

```
int mnozenie0(int n, int a) {
    if (n == 1) return a;
    return mnozenie0(n - 1, a) + a;
}
```

Te dwa wiersze kodu odpowiadają równaniom 2.1 i 2.2. Zarówno a , jak i n muszą być dodatnie, tak jak to było za czasów starożytnych Egipcjan.

Algorytm opisany przez Ahmesa, wśród starożytnych Greków znany jako „mnożenie egipskie”, a przez wielu współczesnych autorów określane mianem „algorytmu rosyjskich chłopów”², zasada się na następującym spostrzeżeniu:

¹ Dla wygody czytania stosujemy w przykładzie wersję kodu z polskimi nazwami w pełnym alfabecie, odmienianymi w tekście; po usunięciu z liter znaków diakrytycznych kod można ćwiczyć na komputerze — *przypr. tłum.*

$$\begin{aligned} 4a &= ((a+a)+a)+a \\ &= (a+a) + (a+a). \end{aligned}$$

To udoskonalenie opiera się na prawie łączności dodawania:

$$a + (b + c) = (a + b) + c.$$

Umożliwia ono jednokrotne obliczenie $a + a$ i zmniejszenie liczby dodawań.

Pomysł polega na połowieniu n i podwajaniu a , co prowadzi do konstruowania sumy wielokrotności potęg 2. W tamtych czasach w opisach algorytmów nie posługiwano się zmiennymi w rodzaju a i n ; autor podałby przykład i skwitowałby go tak: „Tedy postępuj tak samo z innymi liczbami”. Ahmes nie był wyjątkiem. Przedstawił algorytm, pokazując następującą tabelę mnożenia $n = 41$ przez $a = 59$:

1	✓	59
2		118
4		236
8	✓	472
16		944
32	✓	1888

Każda pozycja po lewej jest potęgą 2. Każda pozycja po prawej jest wynikiem podwojenia poprzedniej pozycji (ponieważ dodawanie czegoś do siebie jest względnie łatwe). Zaznaczone wartości odpowiadają bitom 1 w dwójkowej reprezentacji liczby 41. Tablica w gruncie rzeczy oznajmia, że

$$41 \times 59 = (1 \times 59) + (8 \times 59) + (32 \times 59),$$

gdzie każdy z iloczynów po prawej stronie może być obliczony przez podwojenie 59 właściwą liczbę razy.

W algorytmie trzeba sprawdzać, czy n jest parzyste, czy nieparzyste, możemy więc domniemywać, że Egipcjanom znane było to rozróżnienie, choć nie mamy na to bezpośredniego dowodu. Natomiast wiedzieli o tym z pewnością starożytni Grecy, którzy twierdzili, że nauczyli się matematyki od Egipcjan. Oto jak zdefiniowali³ oni liczby parzyste i nieparzyste, jeśli wyrazić to w notacji współczesnej⁴: $n = \frac{n}{2} + \frac{n}{2} \Rightarrow$ parzyste(n),

² Wielu informatyków zaczerpnęło tę nazwę ze *Sztuki programowania* Knutha, gdzie jest podane, że podróżujący po XIX-wiecznej Rosji spotykali chłopów używających tego algorytmu. Jednakże pierwsza wzmianka o tym pochodzi z wydanej w 1911 roku książki Sir Thomasa Heatha, który tak zauważa: „Powiedziano mi, że jest dziś w użyciu sposób (niektórzy mówią, że w Rosji, lecz nie zdołałem tego sprawdzić) [...]”.

³ Ta definicja występuje w pracy z I wieku zatytułowanej *Wstęp do arytmetyki*, księga I, rozdział VII, autorstwa Nikomachosa z Gerazy. Piszze on: „Parzyste jest to, co można podzielić na dwie równe części bez jedności pośrodku, a nieparzyste jest to, czego nie da się podzielić na dwie równe części z powodu wyżej wspomnianego wtrącenia jedności”.

⁴ Symbol strzałki „ \Rightarrow ” czytamy jako „implikuje”. Zob. dodatek A, który zawiera zestawienie notacji matematycznej stosowanej w książce.

$$n = \frac{n-1}{2} + \frac{n-1}{2} + 1 \Rightarrow \text{nieparzyste}(n).$$

My oprzemy się również na tym wymaganiu:

$$\text{nieparzyste}(n) \Rightarrow \text{połowa}(n) = \text{połowa}(n-1)^5.$$

Oto jak możemy wyrazić algorytm mnożenia po egipsku w języku C++:

```
int mnozenie1(int n, int a) {
    if (n == 1) return a;
    int wynik = mnozenie1(połowa(n), a + a);
    if (nieparzyste(n)) wynik = wynik + a;
    return wynik;
}
```

Funkcję `nieparzyste(x)` możemy łatwo zaimplementować, badając najmniej znaczący bit `x`, a `połowa(x)` przez przesunięcie o jeden bit w prawo:

```
bool nieparzyste(int n) { return n & 0x1; }
int połowa(int n) { return n >> 1; }
```

Ile dodawań będzie wykonanych w mnożeniu? W każdym wywołaniu funkcji musimy wykonać dodawanie wskazywane przez `+` w `a + a`. Ponieważ w trakcie rekursji połowimy wartość, wywołamy funkcję $\log n$ razy⁵. Czasami będziemy też musieli wykonać inne dodawanie, wskazane przez `+` w `wynik + a`. Tak więc łączna liczba dodawań wyniesie

$$\#_+(n) = \lfloor \log n \rfloor + (v(n) - 1),$$

gdzie $v(n)$ oznacza liczbę jedynek w dwójkowej reprezentacji n (*licznik populacji*, inaczej *licznik pop*). Zredukowaliśmy zatem algorytm [o złożoności] $O(n)$ do $O(\log n)$.

Czy to jest algorytm optymalny? Nie zawsze. Gdybyśmy na przykład chcieli pomnożyć przez 15, poprzedni wzór dałby taki wynik:

$$\#_+(15) = 3 + 4 - 1 = 6.$$

A przecież przez 15 moglibyśmy pomnożyć z użyciem tylko pięciu dodawań, korzystając z następującej procedury:

```
int mnozenie_przez_15(int a) {
    int b = (a + a) + a;           // b == 3*a
    int c = b + b;                 // c == 6*a
    return (c + c) + b;           // 12*a + 3*a
}
```

Taki ciąg dodawań jest nazywany **łańcuchem dodawań**. Odkryliśmy tu optymalny łańcuch dodawań dla 15. Niemniej algorytm Ahmesa jest w większości sytuacji wystarczająco dobry.

⁵ Połowa w sensie „przepołowienia” liczby naturalnej, z odrzuceniem reszty 1 — *przyp. tłum.*

⁶ Zapis „log” oznacza w całej książce logarytm przy podstawie 2, chyba że określono inaczej.

Ćwiczenie 2.1. Znajdź optymalne łańcuchy dodawań dla $n < 100$.

W którymś momencie czytelnik mógł zauważyć, że niektóre z tych obliczeń dałoby się wykonać jeszcze szybciej, gdybyśmy w sytuacji gdy pierwszy argument jest większy niż drugi, odwrócili najpierw ich kolejność (na przykład znacznie łatwiej byłoby obliczyć 3×15 niż 15×3). To prawda, Egipcjanie też o tym wiedzieli. Nie będziemy jednak dodawać tu tej optymalizacji, ponieważ — jak zobaczymy w rozdziale 7 — zechcemy na koniec uogólnić nasz algorytm na przypadki, w których argumenty są różnych typów i kolejność argumentów jest istotna.

2.2. Ulepszenie algorytmu

Nasza funkcja `mnozenie1` wypada dobrze, jeżeli bierzemy pod uwagę liczbę dodawań, jednak wykonuje ona również $\lfloor \log n \rfloor$ rekurencyjnych wywołań. Ponieważ wywołania funkcji są kosztowne, chcielibyśmy przekształcić program tak, aby uniknąć tego wydatku.

Zasada, z której tu skorzystamy, brzmi następująco. *Często łatwiej jest wykonać więcej pracy niż mniej.* W szczególności zamierzamy obliczać

$$r + na,$$

gdzie r jest bieżącym wynikiem, który gromadzi częściowe iloczyny na . Inaczej mówiąc, zamierzamy wykonać **mnożenie akumulacyjne** zamiast samego mnożenia. Ta zasada okazuje się słuszna nie tylko w programowaniu, lecz również w projektowaniu sprzętu i w matematyce, gdzie często łatwiej jest udowodnić wynik ogólny niż konkretny.

Oto jak wygląda nasza funkcja `mnozenie_akumulacyjne`:

```
int mnozenie_akumulacyjne0(int r, int n, int a) {
    if (n == 1) return r + a;
    if (nieparzyste(n)) {
        return mnozenie_akumulacyjne0(r + a, połowa(n), a + a);
    } else {
        return mnozenie_akumulacyjne0(r, połowa(n), a + a);
    }
}
```

Zachowuje ona niezmiennik $r + na = r_0 + n_0 a_0$, gdzie r_0 , n_0 i a_0 są początkowymi wartościami tych zmiennych.

Możemy to jeszcze ulepszyć, upraszczając rekursję. Zauważmy, że dwa rekurencyjne wywołania różnią się tylko pierwszym argumentem. Zamiast dwóch rekurencyjnych wywołań — dla przypadku parzystego i nieparzystego — po prostu zmodyfikujemy wartość r , zanim wykonamy nawrót, w taki oto sposób:

```
int mnozenie_akumulacyjne1(int r, int n, int a) {
    if (n == 1) return r + a;
    if (nieparzyste(n)) r = r + a;
    return mnozenie_akumulacyjne1(r, połowa(n), a + a);
}
```

Obecnie nasza funkcja jest **rekursją ogonową** (ang. *tail-recursive*), tzn. wywołanie rekurencyjne występuje w niej tylko w wartości zwracanej. Wkrótce zrobimy z tego użytek.

Zauważmy dwie rzeczy:

- n rzadko jest równe 1;
- jeśli n jest parzyste, nie ma sensu sprawdzać, czy jest równe 1.

Możemy zatem dwukrotnie zmniejszyć liczbę porównań z 1, sprawdzając najpierw nieparzystość:

```
int mnozenie_akumulacyjne2(int r, int n, int a) {
    if (nieparzyste(n)) {
        r = r + a;
        if (n == 1) return r;
    }
    return mnozenie_akumulacyjne2(r, połowa(n), a + a);
}
```

Niektórzy programiści sądzą, że w procesie optymalizacji wykonywanej przez kompilator tego rodzaju przekształcenia zostaną zrobione za nas, jednak nieczęsto jest to zgodne z prawdą; kompilatory nie zamieniają jednego algorytmu na inny.

To, co zrobiliśmy do tej pory, jest całkiem niezłe, lecz dążymy do całkowitego wyeliminowania rekursji, aby uniknąć nakładów związanych z wywoływaniem funkcji. Jest to łatwiejsze do osiągnięcia, jeśli rekursja w funkcji jest ściśle ogonowa.

Definicja 2.1. Procedura rekurencyjna z **rekursją ściśle ogonową** to taka, w której wszystkie ogonowe wywołania rekurencyjne są wykonane z parametrami formalnymi procedury w roli odpowiednich argumentów.

To również możemy osiągnąć w prosty sposób, przypisując pożądane wartości zmiennym, które będziemy przekazywać, zanim spowodujemy rekursję:

```
int mnozenie_akumulacyjne3(int r, int n, int a){
    if (nieparzyste(n)) {
        r = r + a;
        if (n == 1) return r;
    }
    n = połowa(n);
    a = a + a;
    return mnozenie_akumulacyjne3(r, n, a);
}
```

Teraz już łatwo to zamienić na program iteracyjny przez zastąpienie rekursji ogonowej konstrukcją `while(true)`:

```
int mnozenie_akumulacyjne4(int r, int n, int a) {
    while (true) {
        if (nieparzyste(n)) {
            r = r + a;
            if (n == 1) return r;
        }
        n = połowa(n);
    }
}
```

```

    a = a + a;
}
}

```

Korzystając z tak zoptymalizowanej funkcji mnożenia akumulacyjnego, możemy napisać nową wersję mnożenia. Ta nowa wersja będzie wywoływać naszą pomocniczą funkcję `mnozenie_akumulacyjne4`:

```

int mnozenie2(int n, int a) {
    if (n == 1) return a;
    return mnozenie_akumulacyjne4(a, n - 1, a);
}

```

Zauważmy, że pomijamy jedną iterację mnożenia `akumulacyjnego4`, wywołując je z wynikiem już ustawionym na a .

Wszystko to wygląda bardzo ładnie z wyjątkiem sytuacji, gdy n jest potęgą 2. Pierwsze, co wykonujemy, to odjęcie 1, co oznacza, że `mnozenie_akumulacyjne4` będzie wywoływane z liczbą, która ma w rozwinięciu dwójkowym same jedynki — jest to najgorszy przypadek w naszym algorytmie. Unikniemy tego, wykonując zawczasu nieco pracy, gdy n jest parzystą, połowiąc ją (i podwajając a) dopóty, dopóki nie stanie się nieparzystą:

```

int mnozenie3(int n, int a) {
    while (!nieparzyste(n)) {
        a = a + a;
        n = połowa(n);
    }
    if (n == 1) return a;
    return mnozenie_akumulacyjne4(a, n - 1, a);
}

```

Teraz jednak zauważamy, że zmuszamy funkcję `mnozenie_akumulacyjne4` do wykonania jednego niepotrzebnego sprawdzenia nieparzystości n , ponieważ wywołujemy ją z liczbą parzystą. Wykonamy więc na argumentach jedno połowienie i podwojenie przed jej wywołaniem, dochodząc w ten sposób do wersji ostatecznej:

```

int mnozenie4(int n, int a) {
    while (!nieparzyste(n)) {
        a = a + a;
        n = połowa(n);
    }
    if (n == 1) return a;
    // parzyste(n-1) ⇒ n-1 ≠ 1
    return mnozenie_akumulacyjne4(a, połowa(n - 1), a + a);
}

```

Przepisywanie kodu

Jak można było zobaczyć na przykładzie wykonanych przez nas przekształceń, przepisywanie kodu jest ważne. Nikt nie pisze dobrego kodu za pierwszym razem. Znalezienie najefektywniejszego lub ogólnego sposobu wykonania czegoś wymaga wielu iteracji. Żaden programista nie powinien mieć jednoprzebiegowej mentalności.

W czasie tego postępowania w którymś momencie mogła Ci przyjść do głowy taka myśl: „Jedna operacja więcej nie zrobi dużej różnicy”. Może się jednak okazać, że Twój kod będzie wykorzystywany ponownie wiele razy przez wiele lat. (Rzeczywistość dowodzi, że prowicki w kodzie potrafią egzystować całymi latami). Ponadto tę taniutką operację, której w tej chwili zaoszczędzasz, w przyszłej wersji kodu ktoś mógłby zastąpić bardzo kosztowną.

Inną korzyścią z dążenia do efektywności jest to, że takie postępowanie wymusza na Tobie głębsze zrozumienie problemu. Z kolei owo głębsze zrozumienie prowadzi do efektywniejszej implementacji — działa tu efekt spirali.

2.3. Przemyślenia związane z rozdziałem

Uczący się elementarnej algebry mają upraszczać wyrażenia tak długo, jak długo się da. W naszych kolejnych realizacjach algorytmu mnożenia po egipsku przeszliśmy podobny proces, przeformułowując kod w celu uczynienia go przejrzystym i sprawniejszym. Każdy, kto programuje, musi nabrać nawyku podejmowania prób przekształcania kodu w dążeniu do otrzymania ostatecznej jego postaci.

Zobaczyliśmy, jak powstawała matematyka w starożytnym Egipcie i jak dzięki niej zyskał pierwszy znany algorytm. Nieco dalej w książce powrócimy do niego, aby go rozszerzyć. Na razie jednak przeskoczmy o tysiąc lat do przodu, żeby przyjrzeć się odkryciom matematycznym z czasów starożytnej Grecji.

Skorowidz

A

- abakus, 58
- abstrakcja, 14, 91
- AKS, 244
- aksjomat, 113, 154, 164
 - indukcji, 172
 - Peana, 171, 176
 - Playfaira, 165
- algebra
 - abstrakcyjna, 14, 91, 142
 - nieprzemienne, 145
 - przemienne, 145
 - Racaha, 221
- algebraiczne liczby całkowite, 142
- algorytm, 19
 - Ahmesa, 22
 - Euklidesa, 49, 54
 - Fletchera i Silvera, 212
 - Griesa-Millsa, 210
 - mnożenia, 22, 63
 - mnożenia egipskiego, 115
 - NWD, 65, 141
 - obliczania ilorazu, 60
 - obliczania reszty, 60, 65
 - odwracania, 219
 - rozszerzony NWD, 234
 - RSA, 240, 245
 - Steina, 224
- algorytmy
 - dostosowywalne do pamięci, 219
 - permutacyjne, 201
 - rotacji, 236

- STL, 269
 - uogólnione, 115
- Arystoteles, 179
- arytmetyka
 - liczb wymiernych, 235
 - modularna, 78, 88
- asercja, 269
- atrybuty typu, 185
- automorfizm, 109

B

- Bachet de Méziriac, 227
- biblioteka STL, 13
- boczne wejście, 239
- bryły platońskie, 49

C

- C++
 - algorytmy STL, 269
 - funkcje szablonowe, 265
 - iteratory, 271
 - koncepty, 266
 - obiekty funkcyjne, 268
 - składnia deklaracji, 267
 - stałe, 267
 - struktury danych, 269
 - zastosowanie synonimów, 272
- C++11
 - funkcje lambda, 273
 - funkcje typów, 272
 - listy inicjatorów, 272

całkowanie symboliczne, 236
 cechy iteratorów, 190, 271
 charakterystyka ciała, 153
 ciało, 152, 155
 ciało reszt pierwszych, 153
 ciąg
 Fibonacciego, 128
 rekurencji liniowej, 130
 Claude Gaspar Bachet de Méziriac, 227
 cykl, 211
 cykl trywialny, 204
 części zdalne, 183

D

dana, 182
 Dirichlet Gustav Lejeun, 141
 dodawanie, 174
 domknięcie przechodnie, 149
 dostęp do cechy iteratora, 272
 dowód, 157
 małego twierdzenia Fermata, 82
 nie wprost, 261
 niekonstruktywny, 231
 orzeczenia, 160
 przez indukcję, 262
 przez zaprzeczenie, 44
 dyrektywa inline, 273
 dziedzina
 całkowitości, 147, 155
 definicji, 117
 euklidesowa, 151, 155
 dzielenie
 egipskie, 63
 wielomianu, 136
 dzielnik
 właściwy, 41
 zera, 146

E

efektywność algorytmów, 221
 element
 główny, 229
 neutralny, 92
 odwracalny, 146
 Euklides, 52
 Euler Leonhard, 75

F

Fermat, 71, 73
 funkcja
 get_temporary_buffer, 220
 iloraz_reszta, 64
 mnożenia, 121
 mnożenia dla monoidów, 122
 negate, 126
 NWM, 55, 60
 odwracanie, 216, 217
 odwracanie_multiplikatywne, 127
 przesiej, 36
 rotowanie, 216
 tocjentu Eulera, 86
 znajdź, 250
 funkcje
 jednokierunkowe, 239
 jednokierunkowe z bocznym wejściem, 239
 lambda, 273
 multiplikatywne, 40
 regularne, 186
 szablonowe, 265
 typu, 185, 272
 funktor, 268

G

Galois Évariste, 94
 Gauss Carl Friedrich, 138
 GCD, 234
 generator grupy, 102, 111
 geometria hiperboliczna, 166
 główna dziedzina ideału, 230
 gnomon, 29
 graf, 150
 graf zaprzyjaźnień, 149
 grupa, 91, 154
 abelowa, 92, 154
 addytywna, 92, 119
 cykliczna, 101, 102
 Kleina, 112

H

Hilbert David, 169
 historia zera, 58

I

ideał, 228
 główny, 229
 kombinacji liniowych, 228
 identyczność warstw, 103
 iloczyn
 macierzy i wektora, 147
 skalarny wektorów, 147
 implementacja
 kodu, 33
 teorii, 108
 implikacja przeciwna, 259
 indukcja, 99, 262
 iteratory, 186
 dwukierunkowe, 188, 271
 o dostępie swobodnym, 188, 271
 postępujące, 188, 271
 powiązane, 189
 segmentowane, 189
 wejściowe, 188, 271
 wyjściowe, 188

J

jedynka, 146
 język
 APL, 127
 C++, 265
 C++11, 265

K

kategorie iteratorów, 188
 klucze, 238
 koncepcje programowania, 179
 koncepty, 33, 183, 266
 konstrukcja kopii, 117
 konstruktor kopiujący, 186
 krok, 32
 krok indukcyjny, 99, 175, 262
 kryptoanaliza, 237
 kryptografia, 237
 kryptologia, 237
 kryptosystem, 238
 kryptosystem z kluczem publicznym, 237, 239
 kwadrywium, 28

L

Lagrange Joseph-Louis, 105
 lemat
 o ideale kombinacji liniowych, 228
 o ideałach w dziedzinach euklidesowych, 229
 o nieodwracalności, 83
 o odwracalności, 230
 o pełnym pokryciu warstwami, 103
 o permutacji reszt, 77
 o reszcie rekurencyjnej, 55
 o rozłączności warstw, 103
 o rozmiarze warstw, 103
 o transpozycji, 203
 o wyszukiwaniu binarnym, 197
 prawo samoskracania, 80
 prawo skracania, 80
 Leonardo z Pizy, 59, 127
 liczba
 dodawań, 23
 rekurencyjnych wywołań, 23
 liczby
 Carmichaela, 242
 całkowite Eisensteina, 142
 całkowite Gaussa, 140, 142
 doskonałe, 38, 70
 Fibonacciego, 127
 figuratywne, 42
 naturalne, 171
 nieparzyste, 21
 niewymierne, 46
 parzyste, 21
 pierwsze, 30
 pierwsze Fermata, 69, 74
 pierwsze Mersenne'a, 69, 70
 porządkowe pozaskończone, 174
 prostokątne, 29
 rzeczywiste, 153
 trójkątne, 29
 względnie pierwsze, 40, 83
 zaprzyjaźnione, 69
 zespolone, 139, 153
 złożone, 85
 licznik rund, 207
 listy inicjatorów, 272

L

łańcuch dodawań, 22
 łączność
 dodawania, 158, 175
 mnożenia, 158
 Łobaczewski Mikołaj Iwanowicz, 166

M

małe twierdzenie Fermata, 74, 89, 106
 Mersenne Marie, 70
 metoda aksjomatyczna, 163
 miara wspólna, 53
 mnożenie, 20, 123, 174
 akumulacyjne, 23
 egipskie, 115
 macierzy, 147, 148
 modularne, 88
 model, 108
 model izomorficzny, 109
 modulo, 80
 moduł, 153, 155
 monoid, 95, 122, 154
 monoid multiplikatywny, 124
 multiplikatywna odwrotność modulo, 80

N

największa wspólna miara, 42
 największy wspólny dzielnik, *Patrz* NWD
 następnik, 171, 187
 nieprzemienna grupa addytywna, 119
 Noether Emmy, 142
 norma, 152
 euklidesowa, 152
 zespolona, 152
 notacja, 257
 NWD, 65, 136, 151, 221
 NWD rozszerzony, 234

O

obiekt, 183
 obiekt funkcyjny, 126, 212, 268
 obliczanie
 ilorazu, 60
 liczb Fibonacciego, 127, 129
 NWD, 51, 67, 136
 reszty, 60

odcedzanie liczb pierwszych, 30
 odwracalność, 215, 230
 odwracalność następnika, 174
 odwrotność, 99
 iloczynu, 99
 potęgi, 99
 odwrócenie małego twierdzenia Fermata, 84
 operacja, 124
 operacja odwracania, 92
 optymalizacja kodu, 33
 otwieranie kodu, 274

P

palindromowe liczby pierwsze, 37
 Peano Giuseppe, 172
 permutacja, 202
 PID, principal ideal domain, 230
 pierścień, 144, 155
 pierścień unitarny, 145
 Pitagoras, 27
 Platon, 50
 podejście Hilberta, 168
 podgrupa, 101, 102
 Poincaré Henri, 231
 postulat równoległości, 164
 potęga odwrotności, 99
 potęgowanie, 123
 pozaskończone liczby porządkowe, 174
 półgrupa, 96, 118, 154
 półpierścień, 148, 150, 154
 boolowski, 150
 tropikalny, 150
 półregularność, 186
 prawo
 doskonalenia interfejsu, 217
 kompletności, 207
 łączności dodawania, 21
 przydatnych zwrotów, 64, 205
 samoskracania, 80
 separowania typów, 206
 skracania, 80
 trychotomii, 43
 predykat, 184
 problemy Hilberta, 170
 programowanie uogólnione, 13, 130
 proporcje liczb całkowitych, 27
 prymarność, 240

przedział
 danych, 271
 ograniczony, 191
 wyliczany, 191
 przemienność
 dodawania, 157, 176
 mnożenia, 158
 potęgowania, 97
 przesiewanie, 34
 przestrzeń
 Hilberta, 170
 polilogarytmiczna, 218
 wektorowa, 153, 155
 przypadek bazowy, 262
 punkt podziału, 196

R

redukcja, 127
 redukcja mocy, 35
 rekurencja liniowa, 130
 rekursja
 ogonowa, 24
 ściśle ogonowa, 24
 w dół, 61
 w górę, 56
 relacje między strukturami, 113
 reszta rekurencyjna, 55
 rotacja, 208
 rozkład
 na czynniki, 141
 na czynniki pierwsze, 41
 rozłączność warstw, 103
 rozmiar
 kroku, 32
 warstw, 103
 rozszerzanie ciała, 153
 rozszerzony
 algorytm Euklidesa, 234
 NWD, 221, 231
 równoważność, 118
 RSA, 240, 245
 rząd, 100
 dowolnego elementu, 106
 grupy, 100
 podgrupy, 104

S

schemat Hornera, 134
 sito Eratostenesa, 31
 skracanie, 78
 sprawdzanie prymarności, 240
 Stevin Simon, 132
 STL, Standard Template Library, 13, 110
 stopień wielomianu, 135
 struktura
 ciało, 155
 ciało proste, 155
 dziedzina całkowitości, 155
 dziedzina euklidesowa, 155
 grupa, 113, 154
 grupa abelowa, 113, 154
 grupa cykliczna, 114
 moduł, 155
 monoid, 113, 154
 pierścień, 155
 półgrupa, 113, 154
 półpierścień, 154
 przestrzeń wektorowa, 155
 struktury
 algebraiczne, 91, 131
 danych, 269
 strumień wejściowy, 188
 suma alikwotowa, 40
 symbol strzałki, 21
 szyfr Cezara, 238

Ś

świadek, 242

T

tablica, 119
 Tales z Miletu, 161
 techniki dowodowe, 261
 tekst
 jawny, 238
 zaszyfrowany, 238
 teoria, 107
 kategoriowa, 110
 liczb, 14, 27, 69
 niekategoriowa, 110
 uniwersalna, 110
 test Millera-Rabina, 243

tocejnt, 85
 tożsamość Bézouta, 227, 230
 transpozycja, 203
 twierdzenie, 160

- Cayleya, 202
- Euklidesa, 39, 76
- Eulera, 84, 86, 107
- Lagrange'a, 103
- o grupach, 100
- o liczbie podstawień, 204
- o wartości pośredniej, 133
- Talesa, 162
- Wilsona, 81

 typ

- obiektu, 183
- regularny, 118
- szablonowy, 116
- wartości, 183
- zakresu, 190

U

uogólnienie

- operacji, 124
- algorytmu Steina, 224

W

warstwa, 103
 wartość, 182
 warunki początkowe, 269
 wektor, 119
 wielkie twierdzenie Fermata, 73
 wielkości niewspółmierne, 53
 wielomian, 136

- Stevina, 131
- unormowany, 142

 wnioskowanie podług równości, 117
 wspólna miara odcinków, 42
 współużytkowanie kodu, 63

wydobywanie, 187
 wymagania

- dotyczące A, 116
- dotyczące algorytmu, 115
- dotyczące N, 120
- semantyczne, 117

 wyrażenie lambda, 198
 wyszukiwanie

- binarne, 194, 197
- liniowe, 193

 wzajemne odwrotności, 79
 wzór na

- różnicę potęg, 39
- sumę dzielników, 41
- sumę nieparzystych potęg, 39

Z

zamiana

- mnożenia na potęgowanie, 123
- przedziałów, 205

 zasada

- dobrego uporządkowania, 43
- dołączania – wykluczania, 87
- kłatek w gołębniku, 100, 263

 zastosowania

- arytmetyki modularnej, 88
- cykli, 211
- NWD, 235
- synonimów, 272

 zbiór liczb względnie pierwszych, 85
 zdanie warunkowe, 259
 zero, 58
 złożoność

- algorytmu, 22
- przestrzenna, 218

 znajdowanie

- największego wspólnego dzielnika, 51
- ścieżek, 149

 znakowanie sita, 33

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄZKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

PRZEKONAJ SIĘ, JAKIE TAJEMNICE KRYJE ŚWIAT MATEMATYKI!

Program to nic innego jak ciąg poleceń realizujących zadany algorytm. A gdy mówimy o algorytmach, jesteśmy tylko o krok od matematyki! To wyjątkowo interesująca dziedzina, którą w praktyce powinien znać każdy programista. Jeżeli chcesz zrozumieć uogólnione zasady programowania oraz podstawy matematycznych abstrakcji, na których się ono opiera, to trzymasz w rękach odpowiednią publikację.

Na kolejnych stronach znajdziesz interesujące informacje na temat pierwszych algorytmów, historii zera oraz nowoczesnych teorii liczb. Po zdobyciu podstawowych wiadomości oraz poznaniu zarysu dziejów matematyki przejdiesz do zaznajamiania się z abstrakcjami, takimi jak grupy, monoidy, półgrupy. Następnie opanujesz m.in. takie zagadnienia jak wyprowadzanie algorytmu uogólnionego, struktury algebraiczne oraz sposoby organizacji wiedzy matematycznej. Sprawdzisz też, jak wyglądają najważniejsze koncepcje programowania, co to są algorytmy permutacyjne i czym zajmuje się kryptologia. Ta książka pochłonie Cię na wiele godzin!

DZIĘKI TEJ KSIĄŻCE:

- poznasz najciekawsze historie z dziedziny matematyki
- przekonasz się, jaki wpływ ma świat matematyki na świat informatyki
- zrozumiesz ciekawe teorie związane z liczbami
- zapoznasz się z podstawowymi koncepcjami programowania
- zdobędziesz solidną wiedzę matematyczną

Alexander A. Stepanov — jest autorem licznych prac o podstawach programowania. W swojej karierze programował systemy operacyjne, narzędzia, kompilatory oraz dodatkowe biblioteki. Jest laureatem nagrody Excellence in Programming, przyznawanej przez miesięcznik „Dr. Dobbs Journal”, i autorem projektu standardowej biblioteki szablonów (STL) w języku C++.

Daniel E. Rose — zajmował kierownicze stanowiska w firmach: Apple, AltaVista, Xigo, Yahoo! i A9.com. W swoich badaniach skupia się na wszystkich aspektach związanych z wyszukiwaniem danych. Na Uniwersytecie Kalifornijskim w San Diego zrobił doktorat z kognitywistyki.

 Addison-Wesley

Helion 

36271 numer katalogowy

kolegiarnia internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Informatyka w najlepszym wydaniu

Sprawdź najnowsze promocje:

● <http://helion.pl/promocje>

Książki najchętniej czytane:

● <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

● <http://helion.pl/newsoci>

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

sięgnij po **WIECEJ**



KOD KORZYSCI

ISBN 978-83-283-1028-5



9 788328 310285

cena: 49,00 zł