Wstęp do informatyki

Piotr Fulmański

Piotr Fulmański[1]

Wydział Matematyki i Informatyki,
Uniwersytet Łódzki
Banacha 22, 90-238, Łódź
Polska

e-mail 1: fulmanp@math.uni.lodz.pl

Data ostaniej modyfikacji: **9 stycznia 2010**

Spis treści

Rozdział 1

Wstęp

Computer Science is no more about computers than astronomy is about telescopes.

Edsger W. Dijkstra

The computer was born to solve problems that did not exist before.

Bill Gates

1.1 Czym jest informatyka?

Podobnie można zapytać o matematykę, filozofię czy biologię, a pytanie o to, czym jest dana dyscyplina zdaje się podstawowe w każdej dziedzinie nauki, jaką człowiek uprawia. Jednakże w przypadku informatyki odpowiedź nie jest tak oczywista. Z pozoru można sądzić, że informatyka jest nauką o komputerach. Stwierdzenie to zawiera w sobie tylko część prawdy. Owszem trudno sobie wyobrazić dzisiejszą informatykę bez komputera, ale czy to znaczy, że bez niego informatyka nie mogłaby istnieć jako dyscyplina wiedzy? Nic bardziej błędnego, komputer jest tylko narzędziem, dzięki któremu informatyka nabrała ogromnego znaczenia i stała się tak bardzo użyteczna. Informatyka to dziedzina, która zajmuje się przetwarzaniem informacji za pomocą pewnych schematów postępowania (zwanych algorytmami). Wynikiem takiego procesu jest znów informacja. Przy czym poprzez przetwarzanie nie należy wyłącznie rozumieć „zamianę" czy „przekształcenie" jednej

informacji w drugą. Do zadań informatyki należy również wyszukiwanie oraz udostępnianie czy prezentacja informacji. Wszystko to będziemy jednak traktowali jako szersze ujęcie „przetwarzania".

Można te rozważania w łatwy sposób zilustrować na przykładzie prostego programu – słownika polsko-angielskiego. Użytkownik wpisuje słowo po polsku (informacja wejściowa), prosi o wykonanie tłumaczenia (algorytm), w efekcie otrzymuje odpowiednik lub odpowiedniki w języku angielskim (informacja wyjściowa). Jeśli w powyższym przykładzie nie wspomnimy, że chodzi o program, tylko o zwykły drukowany słownik, czy jest to nadal informatyka? W pewien sposób tak, gdyż człowiek wykonujący czynność wyszukiwania odpowiednika słowa w języku obcym, świadomie i nieświadomie wykonuje pewien proces (algorytm) poszukiwania tego słowa (świadomie – gdyż bez udziału świadomości niczego nie znajdzie, nieświadomie – gdyż nie zdaje sobie sprawy, że wykonuje algorytm). A informatyka zajmuje się m.in. tworzeniem i badaniem takich algorytmów. Czy to oznacza, że jesteśmy komputerem? Oczywiście nie. Jednak przykład ten pokazuje, że komputer jest tylko narzędziem wspomagającym informatykę i informatyków. Ale nie oznacza to, że jest on niezbędny (choć są procesy, które bez komputera byłyby niewykonalne) czy też jedyny. Z pewnością trudno by było mówić o informatyce w dzisiejszej formie bez komputerów. Wynika to przede wszystkim z faktu, że informatycy do przetwarzania informacji używają coraz to bardziej skomplikowanych algorytmów, które by były wykonane w rozsądnym czasie, potrzebują coraz szybszych komputerów.

Kolejnym ważnym aspektem informatyki jest przechowywanie ogromnych ilości informacji, zwykle w celu późniejszego ich wyszukania czy też przeanalizowania[1]. I tu znów z pomocą przychodzą komputery i elektronika, która obecnie pozwala nam na magazynowanie coraz to większej ilości danych w coraz to mniejszych nośnikach informacji (CD, DVD *etc.*).

Niewątpliwie najbardziej intensywny rozwój informatyki przypadł na ostatnie kilkadziesiąt powojennych lat. Spowodowane to było przede wszystkim powstaniem i bardzo szybkim rozwojem elektroniki, w szczególności elektroniki cyfrowej. Rozwój technologii i w konsekwencji miniaturyzacja układów cyfrowych umożliwił powstanie komputerów osobistych posiadających dużą moc obliczeniową.

Jednak czynności związane z pobieraniem, przechowywaniem i przetwarzaniem informacji człowiek wykonywał od bardzo dawna. Możliwości me-

[1] Wygenerowanie raportu sprzedaży za ostatni miesiąc w programie magazynowo-sprzedażowym jest przykładem informacji, która została przetworzona (przeanalizowana).

chanizacji tych prac, a później ich automatyzacji pojawiały się stopniowo w miarę konstruowania i wytwarzania odpowiednich urządzeń, co było związane właśnie z rozwojem techniki i technologii. W tej sytuacji przestaje dziwić fakt, że najpowszechniej termin *informatyka* rozmiany jest jako *nauka o komputerach*. Bez nich, najprawdopodobniej, nie byłaby ona tym, czym jest obecnie. Takie postrzeganie zdaje się także potwierdzać ogólnie przyjęty fakt datowania początków informatyki traktowanej jako dziedzina naukowa na lata pięćdziesiąte ubiegłego wieku, kiedy to powstawały pierwsze komputery.

Odpowiedzi na postawione pytanie nie ułatwia też wcale obowiązująca terminologia. W języku polskim mamy termin *informatyka*, który najczęściej na język angielski tłumaczony jest jako „Computer science", co sugeruje bardzo bliskie powiązanie informatyki z komputerami. Istnieje jednak także drugi termin „Informatics" także tłumaczony na język polski jako *informatyka*.

Przyjrzyjmy się zatem im bliżej odwołując się do źródeł, jako że mowa (a szczególnie jej pisana postać) jest źródłem największych nieporozumień.

Definicja 1.1 (New Jersey Institute of Technology)**.** *Computer science is the study of information: its structure, its representation, and its utilization. This includes the theory, analysis, design, efficiency, implementation, and application of computer programs (software) and computer equipment (hardware) for developing computerized information processing systems in response to users' needs.*[2]

Definicja 1.2 (Computer Science Department, College of Saint Benedict, Saint John's University)**.** *Computer science is the study of computation. The computation we study is rarely arithmetic, though - often the computation is more symbolic. We might ask about how to compute a good move in chess. Or we might ask how to draw a picture of a three-dimensional scene. As computer scientists, we look for models of computation. And we ask what we can - or cannot - do with these models.*

Computer scientists learn to program computers, because a program is an excellent way of precisely describing a particular computational technique. We also study programming as a profession because it is an important job in today's society.

Thus, a computer science student can expect to study all of the following questions.

[2] http://www.njit.edu/v2/archivecatalog/undergraduate/91/19-und.html, New Jersey Institute of Technology, dostęp 2009-10-05.

- *How do you design an algorithm, (a step by step plan), to solve a problem?*
- *How do you write a computer program to implement your plan?*
- *How can you analyze a program's speed?*
- *How does a computer work to execute a program?*
- *What social responsibilities do programmers have?*
- *How can we develop reliable software systems?*
- *What are the mathematical properties of computation?*

You can expect a large emphasis on computer programming and on mathematics.

Equally important is what computer science is not. It does not emphasize the use of computers in a corporate environment. You can liken it to the difference between an aerospace engineer and an airplane pilot; computer science is more like aerospace engineering. Computer science students do not learn how to use spreadsheets, word processors and other application programs as part of their study of computer science, but may develop those skills on their own or through workshops offered on campus. When we study operating systems or networks, we emphasize the internals, not how they should be configured for use. If you want to study these topics, you want to look for an information technology or information systems program. [...]

We also are not computer engineering. Computer engineering emphasizes how computer hardware works. Computer science students learn about the fundamentals, but only as much as needed to understand how computer software works. [...][3]

Definicja 1.3 (Computer Science Department, University of Cambridge)**.** *There is unfortunately a lot of confusion over these terms in the general population and in schools, and people who should know better use them as if they were interchangeable. Computer Science is the study of information and computation. The other terms are more vocational and mostly describe training courses in how to use particular pieces of software. Think of it this way: someone with an ICT [Information Communications Tecnology] qualification will know how to use a program like Word. Someone with a*

[3] http://www.csbsju.edu/computerscience/curriculum/, Computer Science Department, College of Saint Benedict, Saint John's University, dostęp 2009-10-05.

Computer Science qualification will know how to create a program like Word, and will also know how to make it easier to use, how to make it work on a variety of machines, how to make it easy to add additional functionality, how to fix bugs in it, how to make it communicate with other pieces of hardware or software, how to market it and how to deal with any legal or copyright problems with it. They will understand the theoretical basis underlying the program. They will also know how to do a million other things besides. Not just now, but throughout their working career.[4]

Na podstawie przytoczonych materiałów źródłowych możemy teraz napisać, że informatyka (w sensie anglojęzycznego terminu *Computer Science*) jest studiowaniem teoretycznych podstaw informacji (powstawania, przepływu, składowania), obliczeń i praktycznych technik pozwalających na ich implementację i wykorzystanie w systemach komputerowych. Często opisywana jest jako studiowanie algorytmicznych procesów wytwarzających, opisujących, przekształcających i składujących informację. Powołując się na Petera Denninga ([2]), zasadnicze pytanie na jakie daje odpowiedź informatyka to „Co można efektywnie zautomatyzować".

Definicja 1.4 (School of Informatics, The University of Edinburgh)**.**
What is Informatics?

Informatics is the study of the structure, behaviour, and interactions of natural and engineered computational systems.

Informatics studies the representation, processing, and communication of information in natural and engineered systems. It has computational, cognitive and social aspects. The central notion is the transformation of information - whether by computation or communication, whether by organisms or artifacts.

Understanding informational phenomena - such as computation, cognition, and communication - enables technological advances. In turn, technological progress prompts scientific enquiry. The science of information and the engineering of information systems develop hand-in-hand. Informatics is the emerging discipline that combines the two.

In natural and artificial systems, information is carried at many levels, ranging, for example, from biological molecules and electronic devices through nervous systems and computers and on to societies and large-scale distributed systems. It is characteristic that information carried at higher levels is represented by informational processes at lower levels. Each of

[4] http://www.cl.cam.ac.uk/admissions/undergraduate/myths/#CSIT, Computer Science Department, University of Cambridge, dostęp 2009-09-05

these levels is the proper object of study for some discipline of science or engineering. Informatics aims to develop and apply firm theoretical and mathematical foundations for the features that are common to all computational systems.

The Scope of Informatics

In its attempts to account for phenomena, science progresses by defining, developing, criticising and refining new concepts. Informatics is developing its own fundamental concepts of communication, knowledge, data, interaction and information, and relating them to such phenomena as computation, thought, and language.

Informatics has many aspects, and encompasses a number of existing academic disciplines - Artificial Intelligence, Cognitive Science and Computer Science. Each takes part of Informatics as its natural domain: in broad terms, Cognitive Science concerns the study of natural systems; Computer Science concerns the analysis of computation, and design of computing systems; Artificial Intelligence plays a connecting role, designing systems which emulate those found in nature. Informatics also informs and is informed by other disciplines, such as Mathematics, Electronics, Biology, Linguistics and Psychology. Thus Informatics provides a link between disciplines with their own methodologies and perspectives, bringing together a common scientific paradigm, common engineering methods and a pervasive stimulus from technological development and practical application.

Three of the truly fundamental questions of Science are: "What is matter?", "What is life?ąnd "What is mind?". The physical and biological sciences concern the first two. The emerging science of Informatics contributes to our understanding of the latter two by providing a basis for the study of organisation and process in biological and cognitive systems. Progress can best be made by means of strong links with the existing disciplines devoted to particular aspects of these questions.

Computational Systems

Computational systems, whether natural or engineered, are distinguished by their great complexity, as regards both their internal structure and behaviour, and their rich interaction with the environment. Informatics seeks to understand and to construct (or reconstruct) such systems, using analytic, experimental and engineering methodologies. The mixture of observation, theory and practice will vary between natural and artificial systems.

In natural systems, the object is to understand the structure and behaviour of a given computational system. The theoretical concepts underlying natural systems ultimately are built on observation and are themselves used

to predict new observations. For engineered systems, the object is to build a system that performs a given informational function. The theoretical concepts underlying engineered systems are intended to secure their correct and efficient design and operation.

Informatics provides an enormous range of problems and opportunities. One challenge is to determine how far, and in what circumstances, theories of information processing in artificial devices can be applied to natural systems. A second challenge is to determine how far principles derived from natural systems are applicable to the development of new kinds of engineered systems. A third challenge is to explore the many ways in which artificial information systems can help to solve problems facing mankind and help to improve the quality of life for all living things. One can also consider systems of mixed character; a question of longer term interest may be to what extent it is helpful to maintain the distinction between natural and engineered systems. [5]

Definicja 1.5 (Department of Informatics, Donald Bren School of Information and Computer Science)**.**
What is informatics?

- *Informatics combines aspects of software engineering, human-computer interaction, and the study of organizations and information technology.*
- *In European universities, informatics is the term most often used for computer science.*
- *Computer science studies computers; informatics studies computers* and people.

Computing and information technology play an increasingly pervasive role in our daily lives. Informatics is based on recognizing that the design of this technology is not solely a technical matter, but must focus on the relationship between the technology and its use in real-world settings. That is, informatics designs solutions in context, and takes into account the social, cultural and organizational settings in which computing and information technology will be used.

What topics are part of informatics?

[5] http://www.inf.ed.ac.uk/about/vision.html, School of Informatics, The University of Edinburgh, dostęp 2009-10-05

These aspects of computer science form the core of informatics: software engineering, information retrieval and management, programming languages, human-computer interaction, computer-supported collaborative work, ubiquitous computing, privacy and security, and the effects of technology on society.

At its periphery, informatics touches upon many different disciplines, including management, digital arts, visualization, economics, social science, cognitive science, organizational computing, medical informatics, game technology, and many others.[6]

Informatyka (w sensie anglojęzycznego terminu *Informatics*) jest studiowaniem systemów pozyskujących, reprezentujących, przetwarzających i wytwarzających informację włączając w to wszystkie obliczeniowe, kognitywne[7] i społeczne aspekty. Zasadniczym przedmiotem zainteresowania jest przetwarzanie (przekształcanie) informacji czy to przez procesy obliczeniowe czy komunikacyjne, czy to przez organizmy żywe czy urządzenia. W tym sensie informatykę należy postrzegać jako dziedzinę znacznie szerszą niż informatyka w sensie Computer Science. Można powiedzieć, że informatyka (w sensie Informatics) ogólnie pojęty aspekt pozyskiwania, przetwarzania, składowania itd. informacji rozciąga zarówno nad maszynami (komputery) jak i istotami żywymi a ogólnie, wszystkim tym co ma jakikolwiek związek z informacją.

Materiał prezentowany w książce odpowiad temu co definiuje termin Computer Science. Spróbujmy zatem przybliżyć historię tak własnie rozumianej informatyki patrząc na minione wydarzenia z perspektywy urządzeń służących do automatyzacji pewnych zadań (w myśl definicji Petera Denninga) przy czym nie ukrywamy, że w przeważającej większości będą to urządzenia i maszyn służące do wkonywania obliczeń.

1.2 Historia informatyki

Ze względu na trudności ze zdefiniowaniem samego pojęcia *informatyka* o jakich wspomnieliśmy wcześniej, trudno też podać „jedynie słuszny" rys historyczny tej dziedziny nauki. Bedąc jednak konsekwentnym przyjętemu wcześniej założeniu, spojrzymy na historię informatyki przez pryzmat urzą-

[6] http://www.ics.uci.edu/informatics/qa/, Department of Informatics, Donald Bren School of Information and Computer Science, dostęp 2009-10-05

[7] Kognitywny (łac. *cognitio* poznanie) czyli związany z procesem poznawczym, odnoszący się do poznawania czegoś.

dzeń automatyzujących pewne zadania (głównie obliczenia) i ideii, które je miały usprawnić (lub w ogóle umożliwić).

1.2.1 Prehistoria

Sięgamy do tak odległych czasów aby pokazać, że potrzeba zliczania i zapisywania informacji o ilości towarzyszy ludzkości niemal od jej początków[8]. Zauważmy, że mówimy tutaj o zliczaniu i zapisywaniu ilości a nie o liczbach. Informacją jaką wtedy chciano zachować była ilość czegoś konkretnego a nie liczby wyrażające tą ilość. Problem liczb i ich zapisu poruszymy w rozdziale 2.

Za pierwsze „urządzenie" liczące (a raczej zliczające) uważa się kawałki kości z wykonaną odpowiednią ilością nacięć (ang. *tally stick*) (patrz też podrozdział 2.2.1). Najwcześniejszym tego typu przedmiotem znanym współczesnym archeologom datowanym na rok 35000 p.n.e jest pochodząca z Lebombo (Centralna Afryka Równikowa; Góry Lebombo) kość udowa pawian z 29 nacięciami.

Kolejne znalezisko tego typu, którym jest kość wilka z 57 nacięciami pogrupowanymi po 5, datowane jest na rok 30000 p.n.e. i pochodzi z miejscowości Dolni Vestonice leżącej na Morawach.

W roku 1960 belgijski geolog Jean de Heinzelin de Braucourt podczas prac w Kongu Belgijskim (obecnie Demokratyczna Republika Konga) na terenach Parku Narodowego Virunga w okolicach Ishango (północny kraniec Jeziora Edwarda) pośród pozostałości małej społeczności datowanych na rok 20000 p.n.e. znalazł kość strzałkową pawiana z licznymi nacięciami w kilku grupach i trzech kolumnach. Pierwotnie znalezisko uważano za przykład, potwierdzający wykorzystanie tego typu narzędzi do zliczania, ale obecnie niektórzy badacze sugerują, że informacje zapisane na kości są czymś więcej niż prostym przykładem zliczania i dowodzą znacznie większej świadomości matematycznej (mówi się np. o liczbach pierwszych) ([8], [9]).

1.2.2 Przed naszą erą

Pierwszym powszechnie używanym urządzeniem wspomagającym obliczenia był abakus. Na przestrzeni wieków występował on w różnych kultu-

[8]Przynajmniej tak odległych początków do jakich potrafimy dotrzeć i zweryfikować nasze przypuszczenia. Podane przykłady dotyczą okresu od 35000 lat p.n.e. a tak zwany człowiek neandertalski (neandertalczyk, Homo neanderthalensis) – wymarły gatunek hominida z rodzaju Homo, przez niektórych klasyfikowany jako podgatunek Homo sapiens, żył od ok. 400000 lat p.n.e. do ok. 24500 lat p.n.e.

rach pod różnymi postaciami i przetrwał aż do czasów nam współczesnych gdzie znany jest pod nazwą liczydło.

Pierwsze abakusy będące dziełem ludów sumeryjskich pochodzą mniej więcej z 3000 r. Były to gliniane tabliczki z wyżłobionymi rowkami, do których wkładano kamyki. Każdy rowek symbolizował kolejne rzędy systemu sześćdziesiętnego (ang. *sexagesimal*, *base-sixty*) a ilość kamieni w rowku odpowiadała wielokrotności danej potęgi. Wykonywanie działań za pomocą takiego abakusa polegało na odpowiednim przesuwaniu kamieni w rowkach i ewentualnym ich dokładaniu lub usuwaniu. Chcąc do 31 dodać 23 należało[9]:

1. W kolumnie jedności położyć 1 kamień.
2. W kolumnie dziesiątek położyć 3 kamienie.
3. Do kolumny jedności dołożyć 3 kamienie.
4. Do kolumny dziesiątek dołożyć 2 kamienie.
5. Policzyć kamyczki w kolumnie jedności i kolumnie dziesiątek.

Od razu zauważmy, że choć opisana procedura wydaje się być poprawna to rodzi pewnego rodzaju komplikacje. Jeśli bowiem chcielibyśmy do 17 dodać 7, to w kolumnie jedności mieć będziemy już 8 kamyków do których należałoby dodać jeszcze 7. Tego zrobić jednak nie można, gdyż każda kolumna może zawierać maksymalnie 9 kamyków. Aby rozwiązać ten problem należało w pamięci dodać 7 do 7 i zgodnie z otrzymaną w wyniku liczbą 14, z kolumny jedności odrzucić 3 kamyki pozostawiając 4, a do następnej kolumny dodać 1 kamień.

W drugim stuleciu przed naszą erą pojawiają się pierwsze wzmianki o chińskim abakusie nazywanym suanpan (counting tray). Suanpan był drewnianą ramką z rozciągniętymi w poprzek metalowymi drutami. Dodatkowo całość podzielona była wewnątrz listwą przebiegającą prostopadle do drutów w taki sposób, że każdy drut posiadał część krótszą (nazywaną niebem) i dłuższą (nazywaną ziemą). Na częsci dłuższej znajdowało się 5 koralików, na częsci krótszej 2. Kolejne potęgi używanego systemu liczbowego (był to system dziesiętny) reprezentowane były przez kolejne, począwszy do prawej, druty a położenie koralików decydowało o wielokrotności danej potęgi. Tylko koraliki stykające się z listwą wewnętrzną były brane pod uwagę. W tak skonstruowanym abakusie liczby reprezentowano w systemie bi-quinarnym (ang. *Bi-quinary coded decimal*) czyli mieszance systemu

[9]Posługujemy się tutaj systemem dziesiętnym.

dwójkowego (binary) i piątkowego (quinary). Zasada takiej reprezentacji jest prosta. Jeśli na danym drucie w jego górnej części żaden koralik nie dotyka listwy wewnętrznej, wówczas dolne koraliki reprezentują liczby od 0 do 5. Jeśli natomiast koralik w górnej części dotyka listwy wewnętrznej, wówczas dolne koraliki repreznetują liczby od 6 do 10. Dodatkowo przyjmowało się, że w dolnej częsci nigdy 5 koralików a w górnej 2 nie mogło dotykać listwy wewnętrznej, dzięki czemu reprezentacja była jednoznaczna. Choć taki system wydaje się nam dziś dziwnym to warto wiedzić, że był wykorzystywany w takich historycznych już komputerach jak IBM 650 (1953), UNIVAC 60 (1952) czy UNIVAC LARC (1960).

Przykład reprezentacji liczb w systemie bi-quinarnym w komputerze IBM 650

```
wartość dziesiętna
|
|  65 43210 bity
|  |      |
0  01 10000
1  01 01000
2  01 00100
3  01 00010
4  01 00001
5  10 10000
6  10 01000
7  10 00100
8  10 00010
9  10 00001
```

Wartość x liczby (cyfry) zapisanej w tym systemie znajdujemy według wzoru

$$x = 0 \cdot b_6 + 5 \cdot b_5 + 0 \cdot b_4 + 1 \cdot b_3 + 2 \cdot b_2 + 3 \cdot b_1 + 4 \cdot b_0,$$

gdzie b_i, $i = 0, \ldots, 6$ są bitami.

W starożytnym Rzymie wykorzystywano urządzenie będące czymś pomiędzy abakusem sumeryjskim a chińskim. Gliniane tabliczki zastąpiono bardziej trwałymi wykonanymi z brązu, ale każdy z rowków podzielono na dwie części. Dodatkowo przeznaczono dwa rowki na reprezentację części ułamkowych liczb.

Mniej więcej rok 500 to pierwsze znane przykłady użycia zera przez matematyków indyjskich. Warto wiedzieć, że zero w tamtych czasach traktowane było jak każda inna liczba – dopuszczalne było nawet dzielenie

przez nie. W tym samym okresie Panini, za pomocą 3959 reguł podał wysoce usystematyzowany, opis gramatyki Sanskrytu znany jako Ashtadhyayi („Osiem rozdziałów"). W swoim upisie używając m.in. **metareguł**, **transformacji** i **rekursji** spowodował, że gramatyka ta uważana jest za pierwszy **system formalny**. Występująca obecnie w powszechnym użyciu notacja BNF (Backus-Naur Form) opisu gramatyk bezkontekstowych wykorzystywana jako formalny sposób opisu gramatyki języków programowania, zbioru instrukcji czy protokołów jest do tego stopnia podobna do gramatyki Paniniego, że bywa nazywana też Panini-Backus form ([10]).

Notacja BNF jest zestawem **reguł produkcji** następującej postaci

```
<symbol nieterminalny> ::= <wyrazenie zawierajace symbole>
```

Znaczenie użytych symboli jest następujące

- `<` – lewy ogranicznik symbolu,
- `>` – prawy ogranicznik symbolu,
- `::=` – jest zdefiniowane jako,
- `wyrazenie zawierajace symbole` – składa się z **symboli nieterminalnych** i **symboli terminalnych** (czyli takich, które nigdy nie pojawiają się po lewej stronie reguł), rozdzielonych ewentualnie znakiem **alternatywy**: `|`.

Dla przykładu, używając notacji BNF, określimy liczby całkowite przy pomocy następujących reguł

- `<znak minus>::= -`
  Przykład wartości: -
- `<zero>::= 0`
  Przykład wartości: 0
- `<cyfra niezerowa>::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`
  Przykład wartości: 1, 2, 3
- `<ciąg cyfr> ::= <cyfra> | <cyfra><ciąg cyfr>`
  Przykład wartości: 0, 1, 01, 001, 23, 45, 99, 10023, 000001
- `<liczba calkowita dodatnia>::= <cyfra niezerowa> | <cyfra niezerowa><ciąg cyfr>`
  Przykład wartości: 1, 2, 34, 56, 406, 556066

- `<liczba calkowita ujemna>::= <znak minus><cyfra niezerowa> | <znak minus><cyfra niezerowa><ciąg cyfr>`
  Przykład wartości: -1, -2, -34, -56, -406, -556066

- `<liczba calkowita>::= <liczba calkowita ujemna> | <zero> | <liczba calkowita dodatnia>`
  Przykład wartości: -80001, -123, -3, 0, 1, 5, 41 73, 321001

Jeden z pierwszych znanych nam algorytmów, czyli precyzyjnie podany przepis na realizację konkretnego zadania, pochodzi mniej więcej z roku 400 i opisany został w nieśmiertelnym dziele Euklidesa „Elementy”[10] (księga VII dla liczb całkowitych i księga X dla odcinków, a mówiąc językiem dzisiejszym, dla liczb rzeczywistych). Mowa tutaj o algorytmie znajdowania największego wspólnego dzielnika. I choć algorytm nazywa się algorytmem Euklidesa to faktycznie wymyślił go Eudoksos z Knidos (IV wiek p.n.e.), a Euklides jedynie zawarł go w swoim dziele.

W indyjskich (dżinijskich) tekstach z okresu IV i III w p.n.e. pojawia się pojęcie nieskończoności i to nie tylko w sensie filozoficznym (bo w takim znane było jeszcze wcześniej), ale jako koncept matematyczny związany z liczbami.

That is whole, this is whole
From the whole, the whole arises
When the whole is taken from the whole
The whole still will remain

W trzecim stuleciu indyjski pisarz Pingala wykorzystał zaawansowane koncepcje matematyczne opisując wzorce prozodyczne (metryczne) a więc określające rytmiczną strukturę wersu. Wtedy też zanotowano pierwsze wykorzystanie binarnego systemu liczbowego.

W roku 1901 naszej ery we wraku obok greckiej wyspy Antykithiry (Antikythera), leżacej pomiędzy Kithirą i Kretą, odkryto datowany na lata 150-100 p.n.e. starożytny mechaniczny przyrząd przeznaczony do obliczania pozycji ciał niebieskich[11]. Do czasu XVIII-wiecznych zegarów nie jest

[10] Polskiego tłumczenia Elemntów Euklidesa dokonał w 1807 roku Jóżef Czech pt.: *Euklidesa początków jeometryi xiąg ośmioro* [wszystkich ksiąg jest 13] *, to jest sześć pierwszych, jedenasta i dwunasta z dodanemi przypisami i trygonometrią dla pożytku młodzi akademickiej tłumaczone i wydane*, natomiast teraz jest opracowywana nowa wersja elektroniczna dostepna na stronie `www.interklasa.pl/euklides`

[11] Urządzenie jest prezentowane w kolekcji Narodowego Muzeum Archeologicznego w Atenach.

znany żaden mechanizm o podobnym stopniu złożoności. Mechanizm ten można uważać za pierwszy analogowy komputer.

Na ostatnie stulecie datuje się pierwsze wykorzystanie liczb ujemnych, którego dokonali Chińczycy. Co ciekawe, zarówno liczby ujemne jak i koncepcja liczby zero były znane zapewne znacznie wcześniej. Już korzystając z abakusa obserwujemy, że czasem niektóre rowki (druty) są puste – nie zawierają żadnego kamienia czy koralika, a istotne jest na którym taka sytuacja ma miejsce. Najwyraźniej jednak pojęcia te nie były uważane wówczas za potrzebne. Na przykład system liczbowy stosowany w starożytnym Rzymie, tzw. addytywny system liczbowy, choć uciążliwy w zapisie to jednak doskonale obywał się bez pojęcia „zera".

1.3 Pierwsze tysiąclecie naszej ery

W II w. umiejscawia się wynalezienie logarytmu przez dżinijskich matematyków.

W roku 600 indyjski matematyk Brahmagupta zdefiniował pojęcia zera i liczb ujemnych a także opisał pozycyjny sposób zapisu liczb (ang. *place-value numeral system*). Jak zauważyliśmy zero pojawiło się już nacznie wcześniej, ale dopiero Brahmagupta nadał jemu indywidualny charakter wyróżniając tą liczbę spośród innych. Oto jak w rodziale ósmym swojego dzieła „Brahmasphutasiddhanta" Brahmagupta opisuje działania na liczbach ujemnych ([11])

- The sum of two positives is positive, of two negatives negative; of a positive and a negative [the sum] is their difference; if they are equal it is zero. The sum of a negative and zero is negative, [that] of a positive and zero positive, [and that] of two zeros zero.
- A negative minus zero is negative, a positive [minus zero] positive; zero [minus zero] is zero. When a positive is to be subtracted from a negative or a negative from a positive, then it is to be added.
- The product of a negative and a positive is negative, of two negatives positive, and of positives positive; the product of zero and a negative, of zero and a positive, or of two zeros is zero.
- A positive divided by a positive or a negative divided by a negative is positive; a zero divided by a zero is zero; a positive divided by a negative is negative; a negative divided by a positive is [also] negative.

- A negative or a positive divided by zero has that [zero] as its divisor, or zero divided by a negative or a positive [has that negative or positive as its divisor]. The square of a negative or of a positive is positive; [the square] of zero is zero. That of which [the square] is the square is [its] square-root.

Słowo **algorytm** pochodzi od nazwiska perskiego astronoma i matematyka żyjącego na przełomie VIII i IX w n.e. W 825 roku Muhammad ibn Musa al-Chorezmi (al-Khawarizmy) napisał traktat zatytułowany *O obliczeniach na liczbach indyjskich*[12], w którym podał wiele precyzyjnych opisów dotyczących różnych matematycznych reguł (np. dodawania czy mnożenia liczb dziesiętnych). W XII wieku dzieło to zostało przetłumaczone na łacinę jako *Algoritmi de numero Indorum*, co należało rozumieć następująco: *Algoritmi o liczbach Indyjskich*. Pojawiające się tutaj po raz pierwszy słowo *Algoritmi* było oczywiście inaczej zapisanym nazwiskiem matematyka. Większość ludzi rozumiała jednak tytuł bardziej jako *Algorytmy o liczbach Indyjskich* a stąd już blisko do *Algorytmy na liczbach indyjskich (arabskich)*. W ten oto sposób precyzyjnie opisaną metodę obliczeniową zaczęto nazywć algorytmem (łac. *algorismus*).

Z kolejnej (830 r.) książki al-Chorezmiego zatytułowanej „Al-Kitab al-mukhtasar fi hisab al-jabr wa-l-muqabala" (ang. *The Compendious Book on Calculation by Completion and Balancing*) pochodzi słowo „algebra" wywodzące się od nazwy jednej z podstawowych operacji wykorzystywanych w książce i nazywanej *al-jabr*.

Żyjący w latach 801–873 Abu Yusuf Ya'qub ibn Ishaq al-Kindi znany także jako Alkindus uważany jest za pioniera kryptografii i kryptoanalizy. Wprowadził metodę analizy częstotliwościowej (ang. *frequency analysis method*) pozwalającej w oparciu o statystyczny rozkład liter w tekście dokonać jego deszyfracji.

Alberuni (a właściwie Abu Rayhan Muhammad ibn Ahmad Biruni , 973-1048) to kolejny przykład wszechstronności i kunsztu islamskich naukowców. Zajmował się m.in. astronomią, matematyką, chemią, historią, geografią, geodezią, geologią, farmacją, psychologią, filozofią i teologią. A. I. Sabra, laureat Medalu Sartona przyznawanego przez History of Science Society, określił Alberuniego jako „jeden z największych umysłów naukowych w historii" ([13]). Dowodem na jego znaczący wkład w rozwój nauki jest także fakt nazwania jednego z kraterów Księżyca jako *Al-Biruni*. Pośród wielu jego odkryć i wynalazków znajdujemy także coś, co możemy nazwać

[12](ang. *On the Calculation with Hindu Numerals*)

analogowym komputerem, mianowicie planisferę i mechaniczny kalendarz księżycowo-słoneczny.

1.4 Drugie tysiąclecie naszej ery

Pierwsze 400 lat drugiego tysiąclecia to rozkwit nauki islamskiej. Odnajdujemy tutaj liczne nazwiska osób zajmujących się przede wszystkim astronomią (ale nie tylko), m.in.

- Abu Ishaq Ibrahim al-Zarqali (Arzachel, 1029–1087) wymyśla equatorium (przyrząd astronomiczny),
- Jabir ibn Aflah (Geber, 1100–1150), wymyśla torquetum (przyrząd astronomiczny),
- Abu al-'Iz Ibn Isma'il ibn al-Razaz al-Jazari (Al-Jazari, 1136-1206), który najlepiej znany jest ze swojego dzieła „Kitab fi ma'rifat al-hiyal al-handasiyya" (*Book of Knowledge of Ingenious Mechanical Devices*). w którym to opisał szczegółowo opisał 50 urządzeń mechanicznych.

W roku 1492 Leonardo da Vinci (1452–1519) sporządza szkice urządzenia składającego się z zachodzących na siebie kół zębatych. I choć konstrukcja nigdy nie powstała, uważa się, że mogł to być projekt mechanicznego kalkulatora pozwalającego na dodawanie i odejmowanie liczb. Da Vinci stworzył także plany mechanicznego człowieka, czyli pierwowzoru współczesnych robotów.

W 1588 Joost Buerghi (1552–1632) odkrywa logarytm naturalny a w 1614 logarytmy dziesiętne wprowadza Henry Briggs (1561 – 1630). W roku 1617 John Napier (1550–1617) publikuje *Rabdologiae* w którym opisuje urządzenie wspomagające proces mnożenia, dzielenia a także wyciągania pierwiastków kwadratowych za pomocą specjalnych pałeczek (tzw. pałeczki Napiera). Wzorując się na ideii logarytmu, specjalnie zaprojektowane pałeczki pozwalały sprowadzić np. mnożenie do serii dodawań a dzielenie do serii odejmowań. Pokażemy ideę działania pałeczek na dwóch prostych przykładach.

tutu przyklady

W tym samym czasie żył Wilhelm Schickard (1592–1635), który jest uznawany za twórcę pierwszej mechanicznej maszyny liczącej. W 1623 zbudował on maszynę (nazywaną zegarem liczącym) czterodziałaniową wykorzystującą pałeczki Napiera. Maszyna była zdolna do dodawania i odejmowania liczb 6-cio cyforwych sygnalizując przy tym błędy przepełnienia (czyli

błędy powstające gdy wynik nie daje się wyrazić za pomocą zaplanowanej ilości cyfr).

W 1642 r. francuski matematyk Blaise Pascal (1623–1662) buduje sumator arytmetyczny. Była to maszyna zawierająca kilka tarcz numerowych za pomocą których wprowadzało się liczby i kół zębatych (10 zębów z numerami od 0 do 9) odpowidzialnych za wykonanie obliczeń. Istota działania maszyny była niezwykle prosta i opierała się na automatcznym uwzględnianiu przeniesienia. Chcąc wykonać operacje ustawiało się koła na odpowiednią wartość początkową (na przykład koło dziesiątek na 1, jedności na 8), a następnie przestawiało się odpowiednie koła wymaganą liczbę razy (na przykład jedności o 5). W momencie, gdy jedno z kół obracało się z pozycji 9 na następną, jaką jest 0, powodowało ono również obrót kolejnego koła o jedną pozycję (czyli następowało przeniesienie). Pascal zaczął budować swoje maszyny, zwane pascalinami, z myślą o ojcu, który był poborcą podatkowym. Maszyny te, wyprodukowano ich ok. 50 sztuk, były głównie przeznaczone do obliczeń w różnych systemach monetarnych, kilka z nich przystosowanych było do obliczeń odległości i powierzchni i były przeznaczone dla geodetów.

W 1671 niemiecki matematyk Gottfried Wilhelm Leibniz (1646–1716) zbudował maszynę nazywaną *Stepped Reckoner*. Jej 16-cyfrowa wersja mogła

- dodać (odjąć) 8-cyfrową liczbę do (od) 16-cyfrowej;
- wykonać mnożenie dwóch 8-cyfrowych liczb;
- podzielić liczbę 16-cyfrową przez 8-cyfrową.

Dodawanie i odejmowanie odbywało się w jednym cyklu (tj. przy jednym obrocie korby). Mnożenie i dzielenie odbywało się „cyfra po cyfrze". Operacje te mogły odbywać się także na uprzednio otrzymanym wyniku przechowywanym w akumulatorze, dzięki czemu można było policzyć pierwiastek jako serię dzieleń i dodawań. Z punktu widzenia historii informatyki istotna jest informacja, iż Leibniz opisał system binarny, będący podstawą repreznetacji danych we współczesnych komputerach.

W 1773 Philipp Matthäus Hahn (1739 – 1790) skonstruował pierwszy mechaniczny kalkulator.

W 1786 Johann Helfrich von Müller (1746 – 1830) podał ideę maszyny różnicowej służącej do automatycznego wylicznia tabelaryzowanych wartości wielomianów. Zaproponowane przez niego rozwiązanie wykorzystał Charles Babbage (1791–1871) probując skonstruować swoje wersje ma-

szyny różnicowej. Projekt pierwszej maszyny zaproponowany został Królewskiemu Stowarzyszeniu Astronomicznemu (ang. *Royal Astronomical Society*) w roku 1822 a jej ulepszona wersja powstawała w latach 1847–1849. Niestety pomimo znacznych nakładów finansowych poniesionych na obie maszyny, żadna z nich nie została zbudowana. Spowodowane to było zapewne znacznym stopniem komplikacji[13] i trudnościami technologicznymi związanymi z wykonaniem precyzyjnych mechanizmów. Istotną różnicą pomiędzy wcześniej konstruowanymi maszynami a maszyną różnicową, było to, że po nastawieniu danych początkowych wszelkie dalsze obliczenia odbywały się automatycznie, bez udziału człowieka, za wyjątkiem samego faktu napędzania maszyny. Poniżej na prostym przykładzie wyjaśnimy, będącą podstawą działań maszyn, metodę różnicową.

tutu przyklad

Praktycznie już na samym początku prac nad maszyną różnicową, bo w 1834, Babbage rozpoczął projektowanie, rewolucyjnego jak na tamte czasy, urządzenia nazywanego **maszyną analityczną**. Maszyna ta miała składać się z następujacych bloków funkcjonalnych

- magazynu (odpowiednik pamięci w dzisiejszych komputerach), miał służyć do przechowywania danych oraz wyników z przeprowadzanych na nich operacji;
- młyna (jednostka licząca), odpowiednik dzisiejszej jednostki arytmetyczno-logicznej, miał wykonywać proste działania arytmetyczne;
- mechanizmu sterującego (jednostka sterująca), kierującego działaniem całego urządzenia i w założeniach programowalnego.

Już sam pomysł wydzielenia pewnych bloków funkcjonalnych zasługuje na uwagę i jednocześnie podziw. Pamiętając bowiem, że całe urządzenie było mechaniczne, trudno sobie wyobrazić realizację tak prostego dziś zadania jak przesłanie danych magistralą z jednego układu do drugiego. Istotnie rewolucyjną koncepcją Babbage'a była możliwość programowania maszyny. Nie była to więc maszyna przeznaczona do konkretnego celu, ale na tyle uniwersalna, że mogła liczyć dowolne rzeczy w dowolny sposób.

Wymieńmy najważniejsze parametry maszyny analitycznej

- Siłą napędową maszyny miał być silnik parowy.

[13] Pierwsza maszyna składać się miałą z 25000 częci, jej łączna waga to 13500kg, wysokość 8 stóp, tj. 2,4m. Udoskonalona wersja składać się miała z 4000 części, jej waga to 3000kg i 6 stóp wysokości.

- Maszyna miała mieć ok. 30 metrów długości i 10 metrów szerokości.
- Dane wejściowe (program oraz dane) wprowadzane miały być za pomocą **kart dziurkowanych**, powszechnie wykorzystywanych w tamtym czasie przez krosna mechaniczne[14]
- Do sygnalizowania i udostępniania danych wyjściowe maszyna pisiadała drukarkę i dzwonek. Dodatkowo wyniki mogły być przedstawione na kartach dziurkowanych w celu dalszego ich wykorzystania.
- Maszyna wykorzystywała stałoprzecinkową arytmetykę oraz system dziesiętny.
- Magazyn mógł pomieścić 1000 liczb 50-cio cyfrowych.
- Młyn mógł wykonać wszystkie cztery operacje arytmetyczne, porównanie oraz obliczyć pierwiastek kwadratowy.
- Dostępna była „instrukcja" skoku warunkowego.
- Operacja dodawania (odejmowania) trwać miała ok. 3 sekund natomiast mnożenie (dzielenie) zajmować miało 2-4 minut[15].

Biorąc pod uwagę, że znacznie mniej skomplikowana maszyna różnicowa była budowana przez ponad 25 lat pochłaniając ogromne środki[16] i nigdy nie została ukończona, nie dziw zbytnio fakt, że i maszyna analityczna nie doczekała się realizacji. Skoro jednak zbudowana, z zachowaniem XIX wiecznej tolerancji elementów, współcześnie maszyna różnicowa[17] po drobnych korektach działa poprawnie, więc można mieć nadzieję, że i projekt maszyny analitycznej jest poprawny. Idee Babbage'a znacznie wyprzedziły czas w którym żył, bo dopiero 100 lat później powrócono do zaproponowanej przez niego koncepcji uniwersalnej maszyny liczącej.

O maszynie Babbage'a, mimo iż nigdy nie zbudowanej, warto też pamiętać z innego powodu. To na nią bowiem powstały, w oparciu o dostępną „specyfikacje", pierwsze programy, kórych autorką jest Ada Augusta, hrabiny Lovelace, córka George'a Byrona. Tym samym stała się ona pierwszym

[14] Pomysł „programowania" za pomocą kart dziurkowanych wzoru tkanego przez krosna pochodzi od Joseph-Marie Jacquard'a i datowany jest na rok 1801.

[15] Jest to dosyć ciekawa „własność" wszystkich maszyn liczących: zwykle mnożenie zajmuje znacznie więcej (o rząd lub dwa) czasu niż dodawanie.

[16] Mówi się, że za poświęcone środki można było wybudować dwa okręty.

[17] Zbudowane w latch 1989–1991 dla London's Science Museum. W roku 2000 skompletowano także „drukarkę".

pierwszym programistą w historii a potwierdzeniem jej zasług jest nadanie jej imienia jednemu z najbardziej uniwersalnych i zaawansowanych obecnie języków programowania, jakim jest **Ada**.

W 1854 George Boole opublikował swoją najważniejszą pracę, *An Investiagtion into The Laws of Thought on Which Are Founded The Mathematical Theories of Logic and Probabilities* (Badanie praw myślenia, na którym oparte są matematyczne teorie logiki i prawdopodobieństwa), w której wykazał, jak prawa logiki podane przez Arystotelesa mogą stanowić przedmiot rachunków. Przedstawione idee stanowią podstawę działania współczesnych komputerów.

Kolejnego kroku w dziedzinie przetwarzania informacji dokonał pracownik Biura Statystycznego USA (U.S. Census Office) Herman Hollerith. Pracując jako statystyk zajmował się problemem zmniejszenia ilości czasu, potrzebnego do przetwarzania zebranych danych. Dla przykładu podajmy, że opracowanie wyników spisu powszechnego przeprowadzonego w USA w roku 1880 zajęło 8 lat ([6]), co mogło budzić pewne obawy przed zbliżającym się w 1890 kolejnym, „bogatszym w informacje" spisem. Najbardziej czasochłonną wówczas czynnością okazało się grupowanie i liczenie jednostek posiadających określoną cechę. Hollerith skonstruował maszynę, która automatycznie odczytywała, porządkowała i grupowała dane według określonych kryteriów. Do wprowadzania danych do maszyny użyto specjalnych kart perforowanych, na których występowanie danej cechy zaznaczano otworkiem w odpowiednim miejscu karty. Wydziurkowane karty umieszczano w matrycy, pod którą znajdowały się pojemniki z rtęcią. Do każdego pojemnika doprowadzano prąd elektryczny, następnie opuszczano na kratę płytę z kołeczkami. W miejscach, gdzie były otwory, następowało zamknięcie obwodu (kołeczki dotykały rtęci) i uruchomienie licznika. Dzięki temu urządzeniu prace obliczeniowe związane ze spisem powszechnym w roku 1890 udało się zakończyć w ciągu jednego roku.

Zadowolony z odniesionego sukcesu Hollerith założył w 1896 przedsiębiorstwo Tabulating Machine Company zajmujących się głównie produkcją maszyn podobnych do tej z 1890 roku. W 1911, cztery połączone firmy (w tym firma Hollerith'a) utworzyły firmę Computing Tabulating Recording Corporation (CTR), która w 1924 zmieniła nazwę na International Business Machines Corporation (IBM).

Popularność wykorzystywanych kart perforowanych doprowadziła do ich znormalizowania, co z kolei spowodowało, że stały się one przez wiele 10-leci uniwersalnym nośnikiem informacji i to zarówno w sensie danych do obliczeń, zapisywania programów, jak i ostatecznie pamiętania wyników

tych programów.

1.4.1 Wiek XX

Od okresu międzywojennego możemy już mówić o rozwoju technologii elektronicznej, dzięki wynalezieniu w 1906 przez Lee De Forest'a lampy elektronowej (triody). Lampy elektronowe w ogólności służą do wzmacniania, generacji, przekształcania itp. sygnałów elektrycznych. Trioda składa się z trzech elektrod – anody, katody i siatki. Umożliwia sterowanie przepływem elektronów z katody do anody przez zmianę napięcia na siatce a zatem umożliwia budowanie wzmacniaczy sygnałów elektrycznych.

Jedną z najbardziej znanych i zasłużonych postaci historii informatyki jest Alan Turing. W swoje pracy z 1936 roku „On Computable Numbers" podał on opis niezwykle prostej teoretycznej maszyny (nazywanej maszyną Turinga) zdolnej do wykonania dowolnych obliczeń matematcznych pod warunkiem, że dają się one przedstawić jako algorytm. I choć zbudowanie takiej maszyny mimo jej prostoty nie jest możliwe[18], to jest na dla nas istotna z tego powodu, że żaden istniejący komputer nie ma większej mocy obliczeniowej niż ta prosta maszyna. Eksperyment myślowy z maszyną Turinga pokazuje też bardzo wyraźnie, że obliczenia na współczesnych komputerach to nic innego jak elementarne manipulowanie symbolami. Z uwagi na wielkie znaczenie tej maszyny dla informatyki teoretycznej poświęcimy jej więcej miejsca w osobnym podrozdziale (patrz **??**).

II Wojna Światowa to okres intensywnych prac prowadzonych w Niemczech przez Konrada Zusego. I tak w 1938 ukończył on budowę komputera Z1, będącego pierwszym, co prawda mechanicznym, komputerem programowalnym. Zawierał on praktycznie wszystkie, wyraźnie odseparowane od siebie (w sensie pełnionej roli), współcześnie znane podukłady jak jednostka zmiennoprzecinkowa[19], jednostkę sterującą, pamięć czy urządzenia wejścia/wyjścia. Wykorzystywał system binarny (dane wprowadzano i wyniki otrzymywano w systemie dziesiętnym) i liczby zmiennoprzecinkowe a dane wprowadzane były za pomocą perforowanej taśmy filmowej 35mm. Otrzymany przez Zusego patent[20] wskazywał także na znajomość idei identycznego traktowania danych i kodu programu, tj. przechowywania ich w modyfikowalnej pamięci komputera, choć sam komputer Z1 kod programu pobierł tylko z taśmy a nie z pamięci. Używane było 9 rozkazów[21] o czasie

[18] Ze względu na założenie o niograniczonej pamięci.
[19] Brak było jednostki logicznej.
[20] Z23139/GMD Nr. 005/021
[21] Choć brak wśród nich było rozkazów instrukcji skoku warunkowego.

wykonania od 1 do 20 cykli co przy „zegarze" 1Hz dawało średnią prędkość dla dodawania 5 sekund a dla mnożenia 10. W pamięci mógł przechowywać 64 słowa o długości 22 bitów (176 bajtów). Waga całości to 1000kg.

Zmodyfikowana wersja komputera Z1 oznaczana jako Z2 zbudowana została w roku 1939. W tym przypadku jednostka arytmetyczno-logiczna składała się z przekaźników elektrycznych co wraz ze zwiększeniem częstotliwości pracy do 5Hz zaowocowało skróceniem czasu dodawania do 0.8 sekundy a mnożenia do 3 sekund. Zamiast 22 bitowej arytmetyki zmiennoprzecinkowej wykorzystywał 16 bitową arytmetykę stałoprzecinkową. Zapotrzebowanie na moc wynosiło 1000W.

W maju 1942 Zuse zaprezentował maszynę Z3 będącą wersją maszyny Z1 zbudowaną na przekaźnikach.

W 1945 Zuse opracowuje pierwszy język programowania wyższego poziomu – Plankalkül. Ponieważ historia maszyn Konrada Zusego jest znacznie obszerniejsza niż to co możemy napisać w tym skrótowym rysie historycznym, dlatego na zakończenie wspomnijmy jedynie, że zbudowany w 1950 Z4 wykorzystywany był w Instytucie Matematyki Stosowanej Konfederacyjnej Wyższej Szkoły Technicznej (ETH) w Zurychu przez 5 lat i był to jedyny działający wówczas komputer w Europie.

W 1937 r. rozpoczyna pracę zespół (Claire D. Lake, Francis E. Hamilton, Benjamin M. Durfeepod) konstruktorów złożony z pracowników firmy IBM, kierowany przez Howarda Aikena. Wynikiem ich prac było zbudowanie w roku 1944 największego w historii kalkulatora elektromechanicznego nazywanego IBM Automatic Sequence Controlled Calculator (ASCC) a inaczej Harvard Mark I. Programowanie tej maszyny polegało na odpowiednim łączeniu kabelkami gniazd w specjalnej tablicy sterującej. Dane wprowadzano za pomocą kart dziurkowanych, wyniki wyprowadzano na taśmę perforowaną lub drukowano za pomocą elektrycznych maszyn do pisania.

MARK I miał długość 16 metrów, wysokość 2,4 metra, głębokość 61cm. Składał się z 760 tys. części, w tym z 17 480 lamp elektronowych. Zawierał ponad 800 km przewodów z trzema milionami połączeń. Całkowita waga kształtowała się na poziomie 4500kg. Obsługę stanowiło 10 osób. Wykonywał 3,5 dodawania na sekundę oraz 1 dzielenie na 11 sekund. Częstotliwość pracy wynosiła 100 kHz. Szacowano, iż zastępuje on pracę 100 rachmistrzów wyposażonych w arytmometr mechaniczny. Najbardziej znaną programistką tej maszyny była Grace Hopper, znana m.in. z wprowadzenia do języka informatyków słowa *bug* (pluskwa, owad). Bardzo ciekawe informacje i zdjęcia maszyny przedstawione zostały w [7].

W 1942 r. zespół specjalistów pod kierunkiem Johna Mauchley'ego

i Johna Eckerta projektuje i buduje maszynę ENIAC (ang. Electronic Numerical Integrator And Computer). Jest to pierwsza maszyna, w której zastosowano wyłącznie elementy elektroniczne (lampy elektronowe), i jest uznawana powszechnie za pierwszy kalkulator elektroniczny. Programowanie ENIAC-a polegało na ręcznym ustawianiu przełączników oraz wymianie specjalnych tablic programowych. Długość komputera wynosiła 15 metrów, jego szerokość to 9 metrów, waga 30 ton, składał się z ok. 18 000 lamp elektronowych. Liczby były pamiętane w systemie dziesiętnym, był on w stanie wykonać 5000 dodawań na sekundę, oraz od 50 do 360 dzieleń na sekundę.

W 1945 r. do projektu EDVAC (ang. Electronic Discrete Variable Automatic Computer) przyłącza się John von Neumann (1903–1957), który w notatce zatytułowanej *First Draft of a Report on the EDVAC* zaproponował rozwiązania mające na celu zbudowanie komputera ogólnego przeznaczenia przechowującego program w pamięci w podobny sposób jak czynione jest to z danymi. W ten oto sposób zrodziła się[22] architektura według której są budowane komputery do dnia dzisiejszego a nazywana **von neumannowską**. Dzięki temu możliwe stało się odejście od sztywnych metod programowania sprzętowego (przełączanie kabelków czy zworek) i zastąpienie ich programowaniem wewnętrznym, poprzez umieszczenie w pamięci maszyny programu sterującego przetwarzaniem danych.

Architektura von neumannowska wyróżniała następujące elementy składowe: pamięć złożoną z elementów przyjmujących stany 0 i 1, arytmometr wykonujący działania arytmetyczno-logiczne, jednostkę sterującą. Sterowanie odbywało się za pomocą programu, który był umieszczany w pamięci. Stanowiło to duży skok ideowy w stosunku do wcześniejszych koncepcji, w których program był zapisywany na kartach perforowanych i bezpośrednio z nich odczytywany oraz uruchamiany. W maszynie von neumannowskiej zarówno program, jak i dane, znajdowały się w pamięci fizycznej. Sam program mógł modyfikować zawartość tej pamięci, a co za tym idzie, mógł sam się modyfikować. Program składał się z ciągu instrukcji, które były pobierane i rozpoznawane przez jednostkę sterującą w takt zegara sterującego pracą całego komputera. Instrukcje te musiały odpowiadać poleceniom zakodowanym przez twórców układu elektronicznego. Taka idea powodowała, że nie było już różnicy pomiędzy danymi a rozkazami, wszystkie one były kodowane za pomocą systemu binarnego. Widać bardzo duże podobieństwo do budowy maszyny Turinga (zob. podrozdział **??**), co nie powinno dziwić,

[22]Pisząc „zrodziła" mamy na myśli raczej „ujrzała światło dzinne", gdyż koncepcja architektury tego typu znana była już wcześniej (patrz podrozdział **??**).

gdyż von Neumann znał doskonale wyniki Turinga.

Choć ocena przydatności komputerów i ich roli

I think there is a world market for maybe five computers.

Thomas Watson, chairman of IBM, 1943

There is no reason anyone would want a computer in their home.

Ken Olson, president, chairman and founder of DEC

nie do końca okazała się trafna to pewne optymistyczne prognozy

Computers in the future may weigh no more than 1.5 tons.

Popular Mechanics, forecasting the relentless march of science, 1949

spowodowały, że najpierw nieśmiało a potem coraz bardziej zdecydowanie komputery zaczęły wkradać się do naszego życia codzinnego.

Mówiąc o rozwoju komputerów trudno nie wspomnieć o początkach komputerów osobistych. Pierwszym komputerem tej klasy był Altair, wyprodukowany w 1975 r. przez firmę MITS. Wyposażony w 8-bitowy procesor Intel 8080 oraz 256 bajtów pamięci, pozbawiony klawiatury, monitora, napędu taśmowego, stał się niezwykle popularny wśród osób zajmujących się do tej pory elektroniką. Młody Bill Gates napisał dla niego język BASIC (ang. Beginner's All Purpose Symbolic Instruction Code). Mimo wielu ograniczeń Altair zyskał ogromną popularność, czego konsekwencją było powstawanie coraz większej liczby firm produkujących tego rodzaju „zabawki" – tak wówczas nazywano te urządzenia i tak o nich myślano. Praktycznie wszystkie one działały pod kontrolą systemu operacyjnego nazywanego CP/M (ang. Control Program/Monitor lub Control Program for Microcomputer) i wyprodukowanego przez małą kalifornijską firmę Digital Research. Na skutek dużego zainteresowania rynku urządzeniami podobnego typu, powstają produkowane przez IBM komputery PC (ang. Personal Computer), od których wzięła się nazwa „jakby" klasy – mówimy dziś „komputer PC" lub „komputer klasy PC". Należy tu jednak oddać sprawiedliwość i powiedzieć, że przed komputerem firmy IBM, powstał inny komputer o nazwie Apple II, który był pierwszy z całej gamy „jabłuszek" do dziś znanych i bardzo popularnych, choć głównie na rynku amerykańskim. Co ciekawe, komputery Apple, a później Macintosh, dużo wcześniej od komputerów PC posiadały graficzny interfejs użytkownika – od roku 1985. Inna też była architektura: IBM PC

były i do dziś są oparte na rodzinie procesorów firmy Intel[23], zaś Apple przez długie lata był związany z rodziną procesorów Motoroli i PowerPC.

Na koniec tego krótkiego przeglądu wydarzeń z historii informatyki przedstawiamy zestawienie wybranych dat z historii rozwoju komputerów obecnej generacji, opartych na krzemie.

1947 wynalezienie tranzystora – pierwszego i podstawowego składnika elektroniki cyfrowej i analogowej;

1958 wynalezienie układu scalonego – układu, który zawiera w sobie tranzystory zamknięte w jednej obudowie i realizujące pewne konkretne funkcje;

1964 komputer IBM S/360 – pierwszy superkomputer zwany do dziś Mainframe;

1964 graficzny interfejs użytkownika i mysz;

1971 Intel 4004 – zawierał 2,3 tys. tranzystorów, był taktowany zegarem 740 kHz, mógł zaadresować 1 kB pamięci dla danych oraz 4 kB pamięci programu;

1972 Intel 8008 – zawierał 3,5 tys. tranzystorów, mógł zaadresować do 16 kB RAM;

1974 Intel 8080 – zawierał 4,8 tys. tranzystorów, mógł zaadresować do 64 kB RAM, lista poleceń składała się z 75 rozkazów;

1975 Altair 8800 – pierwszy komputer domowy oparty na procesorze Intel 8080, posiadał 256 bajtów RAM;

1976 procesor Zilog Z80 – modyfikacja Intel 8080, lista poleceń zawierała 176 rozkazów, prędkość zegara wynosiła 4 MHz;

1976 procesor Intel 8086 i 8088;

1977 komputer Apple II;

1979 procesor Motorola 68000;

1981 komputer IBM PC – pierwszy komputer rozpoczynający całą rodzinę istniejących do dziś komputerów osobistych (ang. Personal Computer), był oparty na procesorze 8088, posiadał 64 kB RAM;

[23] Lub procesorów innych firm, ale kompatybilnych z procesorami Intel, np. AMD.

1982 procesor Intel 80286 – zawierał 134 tys. tranzystorów, mógł zaadresować do 16 MB RAM, był taktowany zegarem 6 MHz;

1983 komputer PC XT – oparty na procesorze Intel 8086;

1984 komputer PC AT – oparty na procesorze Intel 80286;

1985 procesor Intel 80386 – zawierał 275 tys. tranzystorów, był taktowany zegarem 16 MHz;

1989 procesor Intel 80486 – zawierał 1,18 mln tranzystorów, był taktowany zegarem 25 MHz;

1992 procesor Power PC – zawierał 2,8 mln tranzystorów, początkowo był taktowany zegarem 66 MHz;

1993 procesor Intel Pentium – zawierał 3,1 mln tranzystorów, początkowo był taktowany zegarem 60 MHz;

1993 procesor DEC Alpha – zawierał 1,7 mln tranzystorów, 300 MHz;

1995 procesor Intel Pentium Pro – 5,5 mln tranzystorów, był taktowany zegarem 200 MHz;

1996 procesor Intel Pentium MMX;

1997 procesor Intel Pentium II – zawierał 7,5 mln tranzystorów, początkowo był taktowany zegarem 300 MHz;

1999 procesor Intel Pentium III – zawierał 9,9 mln tranzystorów, początkowo był taktowany zegarem 600 MHz;

Tym oto sposobem proste urządzenie wspomagające obliczenia, na przestrzeni wieków, przekształciło się w urządzenie wspomagające przetwarzanie danych, a często też służące rozrywce.

Z pewnością dzisiejszy komputer jest bardziej złożony technologicznie od liczydła, lecz jego rola pozostała w dużej części niezmienna: jego zadaniem jest przetwarzanie danych, nawet jeśli tymi danymi będą kolejne plansze gry.

1.5 Zakres informatyki – teraźniejszość i przyszłość

A choć początki twoje były liche,
ostatki świetne będą.
Księga Hioba (przekład Cz. Miłosza)

Obecnie coraz trudniej wyobrazić sobie jakąkolwiek dziedzinę życia bez choćby najmniejszej obecności komputera, a w przyszłości sytuacja ta się jeszcze bardziej nasili. Wystarczy wymienić niektóre tylko dziedziny oraz zastosowanie tam informatyki:

- nauka
  - nauki ścisłe: narzędzie do wspomagania obliczeń, edytor prac naukowych, umożliwienie szybszego kontaktu ze światem,
  - nauki humanistyczne: inteligentne bazy danych, bazy wiedzy, edytory prac naukowych, kontakt ze światem;
- przemysł
  - elektronika: projektowanie układów, wspomaganie obliczeń, analiza poprawności zaprojektowanych obwodów,
  - chemia i farmacja: modelowanie procesów laboratoryjnych, wspomaganie obliczeń,
  - biologia, biochemia: bazy wiedzy, modelowanie cząsteczek, analiza DNA lub RNA,
  - aplikacje CAD/CAM: komputerowe wspomaganie projektowania i modelowania, wykorzystywane głównie przez inżynierów konstruktorów czy architektów, np. AutoCAD;
- zastosowania cywilne
  - modelowanie i przewidywanie zmian pogody czy prądów oceanicznych,
  - wspomaganie kierowania ruchem lotniczym, kolejowym czy morskim,
  - projektowanie dróg, mostów, tuneli, kolei żelaznych;
- zastosowania militarne

  - sterowanie systemem obrony,
  - zdalne naprowadzanie pocisków;

- biznes
  - aplikacje finansowe i ekonomiczne,
  - aplikacje biurowe,
  - systemy eksperckie i systemy wspomagania podejmowania decyzji.

Z pewnością przyszłość informatyki będzie związana z rozwojem technologii kolejnych generacji komputerów. Co więcej same komputery, jak i technologia ich wykonania, mogą się zmienić. Dość nadmienić, że już dziś prowadzone są próby konstruowania **komputerów biologicznych**, czyli takich, w których rolę tranzystora będzie pełniło białko.

Możemy stwierdzić jedynie, że charakter procesów przetwarzanych przez te komputery nie zmieni się, lecz sposób ich wykonania może ulec modyfikacji. Od wielu lat prowadzi się badania i wdraża przetwarzanie potokowe, rozproszone oraz równoległe. Wszystkie te pojęcia, za którymi stają technologie, są związane z wysiłkami człowieka, mającymi na celu przyśpieszenie przetwarzania informacji, a w efekcie uzyskanie wyniku w krótszym czasie.

Cała historia informatyki, a w szczególności jej niesłychanie dynamiczny rozwój przez ostatnie 40 lat, pozwala snuć wręcz bajeczne wizje dotyczące przyszłych zastosowań komputerów. Niemniej jednak należy pamiętać, że za tym wszystkim stoi człowiek i to od jego mądrości i dalekowzroczności będzie zależało, czy przyszłe rozwiązania będą służyły nam efektywnie, czy też staną się dla nas balastem i niechcianą koniecznością.

1.6 Kierunki współczesnej informatyki

Adde parvum parvo magnus
acervus erit.
(Jeśli będziesz dodawał małe do małego,
to zbierze się wielka sterta.)
Owidiusz

Obecnie informatyka jest na tyle szeroką dyscypliną naukową, że podobnie, jak w matematyce, fizyce czy chemii, nie ma już „specjalistów od

wszystkiego". Zatem pojawiają się specjalności, które skupiają się wokół jednego bądź kilku zagadnień. Z faktu ogromnej dynamiki rozwoju informatyki wynika często, że osoby uprawiające dany dział muszą poświęcać mu tyle czasu, że nie mają już możliwości śledzić tego, co dzieje się poza ich „poletkiem". Poniżej zamieszczamy jedną z możliwych klasyfikacji dziedzin informatyki wymienionych w kolejności alfabetycznej.

1.6.1 Teoria informacji

Teoria informacji jest jedną z niewielu dziedzin, dla których znamy dokładną datę narodzin. Jest rok 1948, kiedy to 32-letni matematyk Claude Shannon publikuje pracę w piśmie „Bell System Technical Journal". Ta właśnie praca uznawana jest za podstawową w tej dziedzinie. Jednak zanim opiszemy wyniki Shannona, spróbujemy ogólnie opowiedzieć, czym zajmuje się ta dziedzina.

Teoria informacji zajmuje się informacją, jej transmisją, jak również kodowaniem danych w celu pewniejszego lub szybszego przesłania jej od nadawcy do odbiorcy. Kodowania nie należy tu mylić z szyfrowaniem, gdyż nie jest tutaj celem ukrycie informacji, a wyłącznie zapewnienie, że dotrze ona do celu w całości lub jeśli nastąpi przekłamanie, to o tym się dowiemy. Najprostszym kodem, o jakim większość osób słyszała, jest kod Morse'a, który został skonstruowany, by za pomocą bardzo skromnego zbioru znaków (kropka, kreska) móc przekazać całe zdania dowolnego języka opartego na alfabecie łacińskim.

Należy podkreślić, że próba przekazania jakiejkolwiek informacji przez dowolny nośnik może się nie powieść lub informacja może zostać zniekształcona. Chyba każdy z nas pamięta dziecięcą zabawę w głuchy telefon, podobnie jest w przypadku rozmowy przez zwykły analogowy telefon – słychać trzaski i czasami słowa dochodzą zniekształcone. Shannon w swojej pracy podał warunki, jakie muszą być spełnione, by informacja dotarła bez zniekształceń, co więcej pokazał on drogę, w jaki sposób przekazać informację lepiej, bez zmiany parametrów kanału transmisyjnego.

Zarys teorii informacji znajduje się w rozdziale **??**.

1.6.2 Algorytmika

Algorytmika jest jedną z najstarszych dziedzin informatyki. Wynika to bezpośrednio z zakresu, jaki obejmuje, a jest nim tworzenie i badanie algorytmów. Poprzez badanie algorytmów rozumie się głównie zagadnienia związane z teorią obliczalności, dzięki której można stwierdzić, czy dany

algorytm da się w ogóle zrealizować w praktyce, a jeśli tak, to w jakim czasie[24]. Należy podkreślić, że jest to jeden z działów, który swoim zakresem przenika prawie wszystkie inne oraz niejednokrotnie wykracza poza samą informatykę, wnikając mocno w matematykę, technikę czy ekonomię. Patrząc na historię algorytmiki okazuje się, że najbardziej fundamentalne prace z teorii algorytmów pojawiły się w w latach trzydziestych. Można się temu dziwić, gdyż, jak wynika z rysu historii informatyki, wtedy nie istniał jeszcze żaden komputer elektronowy, a już na pewno elektroniczny. Jednak nie ma w tym nic niezwykłego – algorytmika w swoim aspekcie teoretycznym nie potrzebuje w ogóle komputerów lub potrzebuje ich tylko w minimalnym stopniu. To właśnie w latach trzydziestych Alan Turing formułuje założenia maszyny Turinga (zob. podrozdział **??**) oraz wykorzystuje ją do formułowania teorii obliczalności [**?**]. Stephen Kleene opisuje formalnie spójną klasę rekurencyjnie definiowalnych funkcji teorioliczbowych [**?**]. Z kolei Alonzo Church wymyśla inny sposób opisu algorytmów, tzw. rachunek lambda [**?**]. Nieco później, bo w roku 1961, Andriej Markow proponuje jeszcze inny sposób opisu algorytmów – oparty na łańcuchach [**?**].

Zarys algorytmiki znajduje się w rozdziale 3.

1.6.3 Bazy danych

W obecnym świecie żyjemy w zalewie informacji, którą należy przetworzyć i tym zajmują się wspomniane wcześniej algorytmy, ale również należy gdzieś tę informację magazynować. A jeśli magazynować, to w taki sposób, by łatwo i szybko można było do niej sięgnąć i odnaleźć potrzebne dane. Rolę takiego inteligentnego magazynu spełniają **bazy danych**. Historycznie bazy towarzyszyły informatykom od początku powstania komputerów, lecz różniła się ich forma i sposób przechowywania informacji. Można by nawet zaryzykować stwierdzenie, że już Herman Hollerith, opracowując nową metodę przeprowadzania spisu ludności, posługiwał się bazą danych głosów w postaci kart perforowanych i na ich podstawie opracowywał raporty. Pojawienie się komputerów, jakie znamy obecnie, dało też początek bazom danych z prawdziwego zdarzenia. Z początku były to bardzo prymitywne technologie, do których można zaliczyć bazy hierarchiczne i bazy sieciowe. Posiadały one wiele wad, jak np. nadmiarowość danych czy brak integralności. Pod koniec lat sześćdziesiątych matematyk Edgar Cood, pracujący w firmie IBM, stworzył model relacyjnej bazy danych. Idee tego modelu opi-

[24] Istnieje cała klasa algorytmów, które są poprawne i wiemy, że dają dobre rezultaty, ale np. po 3000 lat.

sał w pracy [?]. Obecnie przede wszystkim wykorzystuje się relacyjne bazy danych, lecz od ponad 15 lat bardzo dynamicznie rozwijają się obiektowe bazy danych, w których daje się, lepiej niż w relacyjnych, zamodelować pewne dane. Współcześnie można zaobserwować rozwój baz opartych na technologii XML.

W tym opracowaniu nie opisano tego zagadnienia szerzej.

1.6.4 Grafika komputerowa

Za datę powstania tego działu informatyki uznaje się lata pięćdziesiąte, kiedy to w Massachusetts Institute of Technology zbudowano komputer Whirlwind wyposażony w grafoskop. Przez ponad dwadzieścia lat dziedzina ta była dostępna wyłącznie nielicznym, głównie ze względu na bardzo wysokie koszty urządzeń. Dość przypomnieć, że wszystkie terminale komputerowe w tamtych czasach były alfanumeryczne, zatem mogły wyświetlać wyłącznie znaki. W latach sześćdziesiątych widać już pewne zainteresowanie przemysłu grafiką komputerową. General Motors wykorzystuje system DEC-1 produkcji IBM do projektowania samochodów. W ich ślady idą producenci samolotów oraz okrętów, a następnie praktycznie cały przemysł.

W dzisiejszych czasach trudno sobie wyobrazić nowoczesnego projektanta bez elektronicznej deski kreślarskiej i, co ciekawe, nie ma znaczenia, czy będzie to architekt, czy też projektant ubrań. Grafika komputerowa oddaje ogromne usługi prawie w każdej dziedzinie współczesnego życia. Idąc do architekta, można obejrzeć swój przyszły dom, wręcz po nim „spacerując", idąc do fryzjera można zweryfikować, czy fryzura nam proponowana jest dla nas odpowiednia. Nie można również zapomnieć o całym przemyśle rozrywkowym, czyli grach i wszelkiego rodzaju multimediach, które coraz chętniej sięgają po tzw. produkty interaktywne.

Wszystko to wymaga wsparcia od strony komputera, zarówno od strony sprzętowej, jak i programowej. Widać to wyraźnie w cenach elementów wchodzących w skład komputera – bardzo dobra karta graficzna potrafi kosztować tyle co reszta komputera, a nawet znacznie więcej. Aby wszystkie te programy działały bardzo wydajnie, wymagają udoskonalania algorytmów do generowania obrazu.

1.6.5 Programowanie

Programowanie jest jednym z działów informatyki, który jest właśnie z nią kojarzony, nawet przez osoby niedoświadczone. Do niedawna samo

stwierdzenie „jestem programistą" brzmiało prawie jak „jestem wszechmocny". Faktem jest, że języki programowania, będące jednym z elementów całości zagadnienia związanego z programowaniem, powstały zaraz na początku współczesnej historii informatyki. Choć, jak wiemy, możemy dopatrywać się ich korzeni jeszcze wcześniej – przy okazji maszyny analitycznej Babbage'a oraz hrabiny Ady. Niemniej jednak języki programowania w formie, w jakiej je znamy obecnie, a raczej „dziadkowie" dzisiejszych języków programowania, pojawiły się w momencie powstania idei maszyny von neumannowskiej, która pozwalała zapamiętywać dane i program jednocześnie w pamięci operacyjnej. Można zatem przyjąć za początek powstania współczesnych języków programowania lata pięćdziesiąte ubiegłego wieku. Bezspornym jest, że języki bardzo się zmieniły od tamtego czasu, choć ich cel pozostał taki sam. Jest nim dostarczenie wygodnej formy przekazu naszych myśli do komputera, w celu uzyskania pewnego konkretnego efektu. Oczywiście sposób zapisu, czyli wyrażania naszych myśli, musi być odpowiednio precyzyjny, ale ta tematyka jest szerzej omawiana w rozdziale **??**.

Programowanie to jednak nie tylko języki programowania, ale również wszelkiego rodzaju narzędzia wspomagające proces wytwarzania oprogramowania, tworzenia dokumentacji technicznej czy dokumentacji użytkownika. Co więcej cały proces tworzenia oprogramowania doczekał się wielu modeli i sposobów analizowania oraz porównywania, które z nich są lepsze i w jakich sytuacjach. Powstała oddzielna gałąź informatyki pod nazwą **inżynieria oprogramowania**, której zadaniem jest właśnie systematyzowanie i badanie procesów związanych z tworzeniem oprogramowania.

1.6.6 Systemy operacyjne

System operacyjny jest wyspecjalizowanym programem, który zapewnia sprawne funkcjonowanie systemu komputerowego. Jest to program ściśle związany z architekturą konkretnego komputera, w tym przede wszystkim z procesorem. Głównym zadaniem systemu operacyjnego jest dostarczanie podstawowych operacji dostępu do urządzeń i zasobów systemu komputerowego. Od jego jakości i wydajności w dużej mierze zależy wydajność całości systemu komputerowego. W dniu dzisiejszym istnieje cała gama systemów operacyjnych dla komputerów klasy PC opartych o procesory rodziny Intel[25]. Przykładowymi systemami dla takich komputerów są MS-

[25] Mówiąc „rodzina Intel" mamy na myśli procesory tej firmy oraz innych firm, kompatybilne (zgodne) z nimi na poziomie listy rozkazów. Przykładem może być rodzina procesorów firmy AMD.

DOS, rodzina systemów Windows, Linux, rodzina BSD, kilka komercyjnych systemów klasy UNIX. Co więcej systemy rodziny Unix, w tym Linux i BSD, są systemami posiadającymi swoje wersje na inne wersje procesorów, np. PowerPC. Istnieje również cała gama systemów operacyjnych specjalizowanych na konkretną maszynę i przeznaczonych do specjalnych celów. Przykładami mogą tu być OS/400 czy OS/390.

Trudno dzisiaj powiedzieć, który system był tym pierwszym. Wiemy na pewno, że w roku 1964 w firmie IBM zaczęto tworzyć złożony i wydajny system operacyjny z przeznaczeniem na komputery MainFrame – maszyny S/360 i ich późniejszych następców[26]. Były to ogromne komputery przeznaczone do ciągłej pracy i najwyższego obciążenia, projektowane z myślą o profesjonalnych systemach. Niemniej jednak IBM wcześniej również tworzył prostsze systemy operacyjne (zob. pkt **??**). Z kolei historia rodziny Unix sięga roku 1965, kiedy to trzy organizacje: Bell Telephone Laboratories, General Electric Company i Massachusetts Institute of Technology w ramach projektu MAC podjęły próbę stworzenia systemu operacyjnego nazwanego Multics. Zadaniem tego systemu było dostarczenie mechanizmów pozwalających na dostęp jednoczesny do danego komputera przez wielu użytkowników oraz wsparcie do obliczeń i współdzielenia danych pomiędzy jego użytkownikami. Niestety prace nad tym systemem nie zostały zakończony i ponieważ nic nie wskazywało, że kiedykolwiek to nastąpi, Bell Laboratories odłączyło się od tego projektu i ok. roku 1970 stworzyło pierwszą wersję systemu Unix, pracującego na maszynach PDP-11. Obecnie zarówno OS/390, jak i Unix istnieją jako dojrzałe i profesjonalne systemy operacyjne.

Niejako na drugim końcu stoją systemy, takie jak rodzina Windows i MacOS, które od samego początku były projektowane, jako systemy nastawione na łatwość obsługi i przeznaczone na popularne komputery. Co prawda istnieją obecnie wersje zarówno Windows jak i MacOS z przeznaczeniem na serwery, jednak nie to było ich pierwotnym przeznaczeniem.

Szerszy opis systemów operacyjnych znajduje się w rozdziale **??**.

1.6.7 Sieci komputerowe

Obecnie sieci komputerowe są jednym z najlepiej ugruntowanych działów informatyki. Ich korzenie sięgają 1957 r., kiedy to Departament Stanu USA powołał do życia Agencję Zaawansowanych Projektów Badawczych Departamentu Obrony USA (ang. Advanced Research Projects Agency (ARPA)). Jednym z jej celów było opracowanie takiej formy komunikacji komputerów,

[26] Obecnie rodzina tych maszyn nosi nazwę zSeries.

która byłaby wygodna, a jednocześnie odporna na ataki militarne. Pod koniec 1969 r. stworzono eksperymentalną sieć opartą na wymianie pakietów danych. Projekt ten pod nazwą ARPANET wykorzystywał protokół NCP (ang. Network Control Protocol). Umożliwiał on logowanie na zdalnym komputerze, drukowanie na zdalnej drukarce oraz przesyłanie plików. Na początku rozwój sieci nie był zbyt dynamiczny. Dla przykładu w 1973 r. ARPANET składała się z 23 komputerów. Dzisiejszą siłę sieć zawdzięcza globalizacji, która była możliwa dzięki zastosowaniu jej do celów cywilnych. Zaczęto dostrzegać, że można ją wykorzystywać do ułatwienia komunikacji na prawie wszystkich płaszczyznach. I tak w 1972 r. zostaje wysłana pierwsza wiadomość elektroniczna (e-mail), której autorem i pomysłodawcą idei był Roy Tomilnson. W 1974 r. zostaje opracowany stos protokołów TCP/IP, który został uznany, w tym samym roku, za oficjalny protokół w sieci i pozostaje nim do dzisiaj. Autorami tej idei byli Vinton Cerf oraz Robert Kahn. Mimo tych sukcesów wciąż są to technologie drogie i niedostępne dla przeciętnego użytkownika. W 1980 r. ARPANET liczyła 400 serwerów. Jednak wydarzenia nabrały tempa i już dziewięć lat później sieć przekroczyła 100 000 komputerów. Stało się to głównie za sprawą przyłączenia się do poparcia tej idei Narodowej Fundacji Nauki USA (ang. National Science Foundation), która stworzyła własną sieć NSFNET i dołączyła się do istniejących. Sieć ta stała się na tyle wydajna, że w roku 1990 zastąpiła całkowicie ARPANET i stała się oficjalnym szkieletem Internetu[27]. Z pewnością większość użytkowników kojarzy Internet z dwoma usługami: pocztą elektroniczną i stronami WWW. Jednak o ile poczta jest usługą wiekową (jak na historię sieci), o tyle WWW powstało dopiero w 1992 r., a technologię tę opracował Tim Berners-Lee.

Należy pamiętać, że sieci komputerowe to nie tylko Internet. W ich skład wchodzą również sieci lokalne lub osiedlowe. Ponadto dziedzina ta zajmuje się również badaniem wydajności sieci i opracowywaniem nowych algorytmów do przekazywania w niej informacji. W połączeniu z kryptografią dba o bezpieczeństwo przesyłanej informacji niezależnie od jej nośnika, a obecnie, poza tzw. przewodem, transmisja danych odbywa się za pomocą: podczerwieni, promieni laserowych czy fal radiowych.

Zarys zagadnień sieci komputerowych znajduje się w rozdziale **??**.

[27] Polska została przyłączona do tego szkieletu w 1991 r.

1.6.8 Kryptografia

Od dawna, to co cenne próbowano ukryć przed wzrokiem czy dotykiem osób niepowołanych. Ponad 2000 lat temu dostrzeżono konieczność ukrywania informacji, na ten okres datuje się pierwszy znany szyfr – **szyfr Cezara**. Obecnie trudno sobie wyobrazić transmisję jakichkolwiek ważniejszych danych w komputerach bez stosowania transmisji szyfrowanej przez **protokół SSL**. Większość programów pocztowych zawiera już wsparcie do szyfrowanego połączenia, a łącząc się z bankiem w celu sprawdzenia stanu konta, serwer banku wymusza na przeglądarce połączenie za pomocą **protokołu https**, który jest niczym innym, jak zwykłym protokołem http „przepuszczanym" przez SSL.

Kryptografia jest właśnie dziedziną wiedzy zajmującą się badaniem, tworzeniem i łamaniem szyfrów. Jej gwałtowny rozwój zaczął się od roku 1975 i do dzisiaj pozostaje jedną z najdynamiczniej rozwijających się dziedzin. Należy zaznaczyć, że jest ona przede wszystkim dziedziną matematyki. Ponieważ jednak tak powszechnie jest wykorzystywana w ochronie informacji, stąd też badaniem jej zajmują się również informatycy.

1.6.9 Sztuczna inteligencja

Pomimo, iż termin **sztuczna inteligencja** (ang. *artificial intelligence* (AI)) powstał stosunkowo niedawno, to próby stworzenia urządzeń inteligentnych sięgają dość daleko wstecz. Za pierwsze należy zapewne uznać wysiłki zmierzające w kierunku skonstruowania automatów potrafiących grać w szachy, które niestety ze względu na niedostatki w rozwoju ówczesnej techniki, najczęściej okazywały się mniej lub bardziej zręcznymi oszustwami.

Pierwsza połowa XX w. to okres upływający pod znakiem „robotyzacji" i pierwszych wizji maszyn wyglądających i zachowujących się jak ludzie. Pod koniec lat pięćdziesiątych stajemy się świadkami narodzin informatyki jako dyscypliny nauki, zajmującej się przetwarzaniem danych, wykorzystującej w tym celu nowe urządzenie nazwane komputerem. Wkrótce, na fali ogólnoświatowego zachwytu nad możliwościami, jakie otworzył on przed nami, powstają pierwsze proste programy zdolne do zadawania pytań i udzielania odpowiedzi na pytania zadane przez człowieka. Dziś jednak wcale nie jest łatwo odpowiedzieć na pytanie: czym jest inteligencja? W znacznej mierze jest to spowodowane brakiem zrozumienia natury ludzkiego mózgu.

Przez lata pojęcie sztucznej inteligencji ulegało przeobrażeniom, zależnie od bieżącego stanu rozwoju myśli ludzkiej – począwszy od automatu do gra-

nia, przez maszynę potrafiącą wykonywać pewne czynności za człowieka, aż po urządzenie zdolne samodzielnie decydować. Nie podejmujemy się w tym miejscu udzielić odpowiedzi na pytanie postawione w poprzednim akapicie. Zamiast tego powiemy, czego oczekuje się obecnie od sztucznej inteligencji. Otóż celem jest rozwój w kierunku stworzenia systemu, który potrafiłby zachowywać się jak istota inteligentna, tj. na podstawie zgromadzonej wiedzy i znanych przesłanek podejmowałby decyzje wykazujące znamiona racjonalności. Pragniemy podkreślić ogólność ostatniego stwierdzenia. Przez termin „system" można rozumieć program komputerowy, maszynę lub... cokolwiek innego. Również pojęcia „wiedza", „racjonalność" są tak samo mało precyzyjne, jak i sam termin „inteligencja".

Sztuczna inteligencja jako nauka ukierunkowana na stworzenie systemu naśladującego istoty rozumne z istoty rzeczy korzysta z opisu pewnych ich zachowań, budowy czy obserwowanych mechanizmów nimi rządzących. Nie dziwi zatem prowadzenie badań związanych ze sztucznymi sieciami neuronowymi, algorytmami genetycznymi, systemami działającymi w oparciu o logikę rozmytą, czy też badających zachowania populacji – algorytmy mrówkowe, algorytmy symulujące rój pszczół itp. Można powiedzieć, że zakres badań obejmuje wszystko „co się rusza". Tak jest w istocie, a wynika to z komplementarności tych poddziedzin sztucznej inteligencji. Na przykład sztuczne sieci neuronowe stosują stochastyczne algorytmy dopasowania modeli poprzez uczenie nadzorowane lub nie, opierające się na nieprecyzyjnych (obarczonych pewnym błędem, tzw. szumem) danych numerycznych. Złagodzenie wymogów precyzji w procesie tworzenia modeli, a co najważniejsze dla nas ludzi, możliwość opisu złożonych systemów za pomocą zmiennych pojmowanych intuicyjnie (np.: lodowato, zimno, w sam raz, ciepło, gorąco) to z kolei zaleta systemów przetwarzających informacje w oparciu o zbiory rozmyte i wnioskowanie przybliżone. Połączenie obu wspomnianych wyżej dziedzin pozwala na stworzenie systemu operującego nieprecyzyjnymi określeniami, zależnymi od zaistniałego kontekstu (logika rozmyta), oraz jednocześnie zdolnego do uczenia się (sztuczne sieci neuronowe). Doskonałym ich uzupełnieniem stają się algorytmy genetyczne poszukujące optymalnego rozwiązania (np. parametrów sieci neuronowej) na drodze kolejnych przybliżeń dobieranych poprzez wykorzystanie mechanizmów mutacji i krzyżowania chromosomów oraz oceny przez tzw. funkcję przystosowania.

Jak więc widać, sztuczna inteligencja to bardzo szeroka dziedzina wiedzy, łącząca w sobie matematykę, biologię, fizykę chemię, filozofię, lingwistykę..., o której precyzyjnie za wiele powiedzieć nie można. Pewne jedynie jest to, że na dzień dzisiejszy w związku z nią zadajemy więcej pytań niż

otrzymujemy odpowiedzi.

1.7 Zadania

1. Wymień kluczowe postacie z historii informatyki wraz z ich dokonaniami. Spróbuj ustawić je chronologicznie.

2. Wymień dwie dziedziny informatyki i krótko je scharakteryzuj.

3. Spróbuj wyjaśnić wpływ komputerów na dzisiejszą informatykę.

Rozdział 2

Systemy liczbowe

2.1 Liczby i ich systemy

Liczba jest pewnym abstrakcyjnym bytem wykorzystywanym do zliczania i mierzenia. Symbol lub słowo języka naturalnego wyrażające liczbę nazywamy **numerałem** lub **cyfrą**[1] (ang. *numeral*, *digit*) a w języku potocznym, po prostu **liczbą**. Numerały różnią się od liczb tak jak słowa różnią się od rzeczy, które określają. Symbole „11", „jedynaście" oraz „XI" są różnymi numerałami reprezentującymi tą samą liczbę.

W potocznym znaczeniu słowo *liczba* używane jest zarówno w pierwotnym znaczeniu abstrakcyjnego bytu wyrażającego ilość i wielkość jak i symbolu. Oto bowiem wyrażenia numeryczne (a więc złożone z cyfr) używane są jako pewnego rodzaju nazwy (np. numer telefonu), w celu uporządkowania (np. numer seryjny) czy też jako kod (np. ISBN).

Uwaga 2.1. *Określenie* liczba *bez żadnego przymiotnika jest nieścisłe, gdyż matematycy nie definiują* liczb, *lecz* liczby naturalne, liczby całkowite, *itp. Poszczególne rodzaje liczb są definiowane za pomocą aksjomatów lub konstruowane z bardziej podstawowych pojęć, takich jak zbiór, czy typy liczb prostsze od konstruowanego. Konstrukcję można też przeprowadzać od drugiej strony, to znaczy zacząć od zdefiniowania np. liczb rzeczywistych za pomocą aksjomatów, a następnie określić prostsze typy liczb jako podzbiory z tymi samymi działaniami.*

Mając zdefiniowane pojęcie cyfry (numerału) możemy teraz wprowadzić pojęcie systemu liczbowego.

[1] Choć termin *cyfra* zasadniczo zarezerwowany jest dla pojedyńczego symbolu to jednak np. język angielski zdaje się nie rozróżniać tych dwóch terminów.

Definicja 2.1 (System liczbowy)**. System liczbowy** *jest sposobem reprezentacji liczb przy użyciu cyfr (numerałów) w jednolity sposób. W zależności od kontekstu numerał „11" interpretować będziemy jako dwójkowe przedstawienie liczby trzy, dziesiętne przedstawienie liczby jedynaście lub być może jeszcze inna liczbę zapisaną w innym systemie.*

Porządane jest aby system liczbowy posiadał następujące cechy.

- Pozwalał reprezentować użyteczny zbiór liczb (np. wszystkie liczby, liczby całkowite itp.).
- Każdej liczbie przyporządkowywał jednoznaczną reprezentację lub przynajmniej w jakiś sposób ustandaryzowaną.

2.2 Rodzaje systemów liczbowych

2.2.1 Unarny system liczbowy

Najprostszym systemem liczbowym jest **unarny system liczbowy**, w którym każda liczba naturalna reprezentowana jest przy pomocy jednego znaku powielonego tyle razy ile wynosi liczba reprezentowana przez tworzony numerał. Jeśli wybranym symbolem będzie `|`, wówczas liczbę siedem zapiszemy jako siedmiokrotne powtórzenie tego znaku, czyli `|||||||`. Wbrew pozorom system ten wciąż funkcjonuje u ludów pierwotnych a i cywilizacje bardziej rozwinięte wykorzystują go do zapisu niewielkich liczb (patrz też 1.2.1).

Systemy tego typu nazywamy także **systemami addytywnymi**, bowiem wartość liczby otrzymujemy poprzez dodawanie kolejnych wartości wyrażanych przez symbole (w tym przypadku jeden symbol).

2.2.2 Skrócony zapis systemu unarnego

Dosyć „rozwlekły" system unarny można uczynić bardziej zwięzłym, stosując różne dodatkowe symbole na określenie pewnych wartości. Na przykład jeśli `|` oznacza „jeden", `-` oznacza „dziesięć" `=` oznacza „sto", wówczas liczbę 304 zwięźle zapisać możemy w następujący sposób: `===` `||||` a liczbę 123 jako: `=` `-` `-` `|||` bez potrzeby używania symbolu i pojęcia „zero". System liczbowy tego typu nazywać będziemy **systemem addytywnym** jako, że wartość liczby otrzymujemy przez dodanie wartości reprezentowanych przez kolejne znaki. Liczby rzymskie są właśnie liczbami takiego systemu.

Symbol	Wartość	
I	1	(*unus*)
V	5	(*quinque*)
X	10	(*decem*)
L	50	(*quinquaginta*)
C	100	(*centum*)
D	500	(*quingenti*)
M	1000	(*mille*)

Tablica 2.1. Symbole używane do zapisu liczb w rzymskim systemie liczbowym.

Trochę bardziej użytecznymi systemami są systemy używające pewnych symboli do określenia ilości powtórzeń symboli określających liczbę. Na przykład możemy używać pierwszych dziewięciu liter alfabetu na określenie ilości powtórzeń, przy czym `A` oznaczałoby „dwa powtórzenia", `B` „trzy powtórzenia" itd. Wówczas liczbę 304 mogli byśmy zapisać jako `C= D|`. Taki sposób wyrażania liczb obecny jest w większości nowożytnych języków europejskich, np. w angielskim: „three hundred [and] four" czy polskim „trzy-sta [i] cztery".

Jednym z dobrze znanych nam przykładów takiego systemu jest rzymski system liczbowy. Liczby wyrażano w nim stosując kilka podstawowych symboli (patrz tabela 2.1). W celu umożliwienia zwięzłego zapisu większych liczb wprowadzono dodatkowe symbole

- kreskę pionową – liczba umieszczona między kreskami była mnożona przez 100
$$|MD| = (1000 + 500) * 100 = 150000$$
- nadkreślenie – liczba nad którą występowała kreska była mnożona przez 1000
$$\overline{MD} = (1000 + 500) * 1000 = 1500000$$

W systemie tym przyjmujemy dodatkowo następujące założenia.

- Każdorazowe wystąpienie symbolu o mniejszej wartości przed symbolem o większej wartości oznacza odejmowanie (od większego mniejsze).
- Jako poprawne uważa się zapisy bardziej zwarte, np. IV zamiast IIII.

2.2.3 Dzisiejsze systemy liczbowe

Najpowszechniej dzisiaj używane systemy liczbowe mają indyjsko-arabski rodowód (patrz 1.3).

Aby zrozumieć jak działają współczesne systemy liczbowe, zastanówmy się co się dzieje gdy piszemy liczbę dzisiaj i co się działo gdy robili to starożytni Rzymianie.

```
Dziś       Rzymska
111=?!     III=?!
|||        |||
||1        ||1
||         ||
|10        |1
|          |
100        1
```

Rozważmy dwie liczby (numerały): 111 oraz III. W obu przypadkach korzystamy z tylko jednego symbolu: „1" w pierwszej liczbie (numerale) oraz „I" w drugiej. Ilość wystąpień każdego z nich jest identyczna. Zauważmy teraz, że w przypadku liczb rzymskich każde wystąpienie „I" ma takie samo znaczenie: oznacza wartość równą 1 (jeden). Inaczej sytuacja wygląda gdy spojrzymy na liczbę „współczesną" tj. 111. Kazde wystąpienie symbolu „1" oznacza coś innego: wartość równą 1 (jeden) albo 10 (dziesięć) albo 100 (sto). Zatem pierwszą naszą obserwacją jest:

Obserwacja 2.1. *We współczesnych systemach liczbowych pozycja cyfry w liczbie ma istotne znaczenie.*

```
Dziś       Rzymska
115=?!     IIV
|||        |||
||5        ||5
||         ||
|10        |-1
|          |
100        1
```

Rozważmy teraz dwie inne liczby: 115 oraz IIV[2]. Zauważmy, że symbol „I" w liczbie rzymskiej ma różne znaczenia – w szczególności środkowy

[2]Liczba ta nie jest do końca poprawna. Poprawna jednak nie jest też liczba IIII, którą można znaleźć na zegarach czy MDCCCCX zamiast MCMX (1910) widoczne na Łuku Admiralicji (Admiralty Arch) w Londynie.

symbol „I" ma inne znaczenie zarówno w kontekście tego, jak i poprzedniego przykładu. Znaczenie symbolu „1" w liczbie 115 pozostało takie samo jako w 111.

Obserwacja 2.2. *We współczesnych systemach liczbowych znaczenie cyfry nie zależy od kontekstu (innych symboli otaczających).*

Poszukując znaczenia dzisiejszych liczb, przyjrzyjmy się liczbie (numerałowi) 304

$$304 = 300 + 0 + 4 = \dots$$

który inaczej zapisujemy jako

$$\cdots = 10^2 \cdot 3 + 10^1 \cdot 0 + 10^0 \cdot 4.$$

Używamy trzech różnych cyfr (0, 3 i 4) oraz ich pozycji w liczbie celem określenia wykładnika dla potęgi liczby dziesieć – suma iloczynów cyfr i liczby dziesięć podniesionej do odpowiedniej potęgi daje nam poszukiwaną wartość. Zauważmy, że symbol „zero", który nie był potrzebny w systemach addytywnych, tutaj pełni bardzo istotną role pozwalając „pominąć" niektóre potęgi (rzędy wielkości). Ponieważ jako **podstawy systemu** używamy liczby 10 (to właśnie ona podnoszona jest do odpowiedniej potęgi), zatem ten system nazywany jest **systemem dziesiętnym** lub **systemem o podstawie dziesięć** (ang. *base-10 systems*, *decimal numeral system*). Zawdzięczamy go dorobkowi indyjskich matematyków.

Uogólniając, każda n-cyfrowa liczba dziesiętna przedstawiana jest w postaci

$$d_{n-1}...d_1d_0,$$

gdzie d_i, $i = n-1, \dots, 1, 0$ jest jedną z cyfr systemu dziesiętnego a więc należy do zbioru $\{0, 1, ..., 9\}$. Wartość v takiej liczby (numerału) obliczamy według poniższej formuły

$$v = d_{n-1} \cdot 10^{n-1} + \cdots + d_1 \cdot 10^1 + d_0 \cdot 10^0.$$

Obserwacja 2.3. *Arytmetyka jest znacznie łatwiejsza w systemach pozycyjnych niż addytywnych. Co więcej, addytywne systemy teoretycznie mogą potrzebować nieskończonej ilości symboli do wyrażenia co raz to większych liczb (np. co raz to większych potęg liczby 10). Wykorzystując natomiast pozycyjny system liczbowy, przy pomocy skończonego zbioru cyfr możemy wyrazić każdą liczbę.*

Przykład dziesiętnego systemu liczbowego pozwala nam w łatwy sposób podać ogólną definicję pozycyjnych systemów liczbowych o dowolnej podstawie.

Definicja 2.2. Pozycyjnym systemem liczbowym *(ang.* positional numeral system *lub* place-value numeral system*) nazywamy parę* (b, D)*, gdzie* b *jest lliczbą naturalną nazywaną podstawą systemu (ang.* base *lub* radix of that numeral system*),* D *jest skończonym zbiorem* b *symboli* $\{s_0, s_1, \ldots, s_{b-1}\}$*, nazywanych* **cyframi** *(ang.* digits*)*[3]*. System taki nazywamy* **systemem liczbowym o podstawie** b *(ang.* base-b system*).*

Jeśli $b = 10$ *to taki system będziemy nazywać także* **dziesiętnym***, jeśli* $b = 2$ – **dwójkowym***, jeśli* $b = 8$ – **ósemkowym***, itd.*

W takich systemach każda liczba jest jednoznacznie reprezentowana jako ciąg cyfr a jej wartość zależy zarówno od cyfr jak i pozycji na jakich one występują. Wartość v *ciągu*

$$d_k d_{k-1} \ldots d_1 d_0$$

obliczamy według poniższej formuły

$$v = d_k b^k + d_{k-1} b^{k-1} + \ldots + d_1 b^1 + d_0 b^0 \tag{2.1}$$

gdzie $d_0, \ldots, d_k \in D$*.*

Mówiąc inaczej, pozycyjny system liczbowy jest takim sposobem zapisu liczby, w którym każda kolejna pozycja na jakiej możemy zapisać cyfrę związana jest z porzednia (następną) wspólnym współczynnikiem (mnożnikiem) nazywanym podstawą systemu w ten sposób, że wartość pozycji następnej równa jest wartości pozycji poprzedniej pomnożonej przez ten współczynnik. Całkowita wartość liczby zapisanej w takim systemie równa jest sumie wartości każdej z jej pozycji pomnożonej przez cyfrę występującą na danej pozycji.

Używając jednocześnie kilku różnych pozycyjnych systemów liczbowych, zawsze musimy zaznaczyć w jakim z nich dana liczba jest zapisana. Przyjrzyjmy się przykładom:

- 11_{10} - liczba o dziesietnej wartości 11 zapisana z pozycyjnym systemi liczbowym o podstawie 10,

[3]Zazwyczaj zbiór D składa się z odpowiedniej liczby początkowych symboli tworzących ciąg $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ i jeśli zajdzie taka potrzeba to kolejnych liter alfabetu łacińskiego: $A, B, \ldots$, przyjumjąc zasadę, że A oznacza *dziesięć*, B – *jedynaście*, itd.

- 11_2 - liczba o dziesietnej wartości 3 zapisana z pozycyjnym systemi liczbowym o podstawie 2,
- 11_5 - liczba o dziesietnej wartości 6 zapisana z pozycyjnym systemi liczbowym o podstawie 5,
- 11_{25} - liczba o dziesietnej wartości 26 zapisana z pozycyjnym systemi liczbowym o podstawie 25.

2.2.4 System dwójkowy

Ponieważ współczesne komputery są konstruowane w oparciu o układy cyfrowe pracujące według reguł dwuelementowej algebry Boole'a (zob. podrozdział **??**), stąd w informatyce szczególną wagę przywiązuje się do systemu dwójkowego. Omówimy teraz dokładnie ten system, konwersję z systemu dziesiętnego na dwójkowy i odwrotnie oraz arytmetykę w systemie dwójkowym.

2.2.5 Conversion from base-2 (binary) into base-10 (decimal) system

Based on positional numeral system definition (definition 2.2) we conclude that the base of binary system is number 2, and set of digits D consist of two symbols, by definition denoted as 0 and 1.

Taking as an example number x as follow

$$x = 1011110110_{(2)} \tag{2.2}$$

and using (2.1), we obtain

$$\begin{aligned} v &= 1 \cdot 2^9 + 0 \cdot 2^8 + 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + \\ &+ 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0. \end{aligned}$$

Repleacing powers of 2 by coresponding values expressed in decimal system, we obtain

$$\begin{aligned} v &= 1 \cdot 512 + 0 \cdot 256 + 1 \cdot 128 + 1 \cdot 64 + 1 \cdot 32 + \\ &+ 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 758_{(10)}. \end{aligned}$$

It can be seen, that conversion from binary to decimal come down to use formula (2.1) and calculate value described by it. The only thing to remember is that all calculations have to be done in decimal system.

2.2.6 Conversion from base-10 (decimal) into base-2 (binary) system

While conversion from binary system to decimal system comes down to addition and multiplication, an opposite conversion require more attention. If some decimal number x is given, than conversion follows such an algorithm:

1. Begin.
2. Let $v = x$.
3. Divide v by 2 and denote it as r.
4. If the result r is an integer, write 0.
5. If the result r is not an integer, write 1.
6. Take as v integer part of result r.
7. If v is not equal to 0, go to step 3.
8. If v is equal to 0, go to step 9.
9. End.

In this way a sequence of 0's and 1's is obtained. This sequence write down from the right to the left (in order we obtain each digit) is a binary number we are looking for.

To illustrate this algorithm, consider following example. Say that we are looking for binary representation ofdecimal number 758. So the first step is to divide number 758 by 2

```
758 | 2 * 379 + 0 (v=379, r=0)
```

Now the number 379 is divided by 2

```
379 | 2 * 189 + 1 (v=189, r=0)
```

and by analogy we obtain Obtained in this way right column is a binary representation of decimal number 758. This column write down from the right to the left (in order we obtain each digit) is a binary number we are looking for.

$$1011110110_{(2)}.$$

```
189 | 2 * 94 + 1
 94 | 2 * 47 + 0
 47 | 2 * 23 + 1
 23 | 2 * 11 + 1
 11 | 2 * 5 + 1
  5 | 2 * 2 + 1
  2 | 2 * 1 + 0
  1 | 2 * 0 + 1
  0 |
```

As could be notice result is identical to (2.2) which is, as we checked above, binary representation of (decimal) value 758.

Below there are some more examples (example 2.1).

Przykład 2.1. Conversion from base-10 (decimal) into base-2 (binary) system.

```
158 | 2 * 79 + 0      108 | 2 * 54 + 0      59 | 2 * 29 + 1
 79 | 2 * 39 + 1       54 | 2 * 27 + 0      29 | 2 * 14 + 1
 39 | 2 * 19 + 1       27 | 2 * 13 + 1      14 | 2 * 7 + 0
 19 | 2 * 9 + 1        13 | 2 * 6 + 1        7 | 2 * 3 + 1
  9 | 2 * 4 + 1         6 | 2 * 3 + 0        3 | 2 * 1 + 1
  4 | 2 * 2 + 0         3 | 2 * 1 + 1        1 | 2 * 0 + 1
  2 | 2 * 1 + 0         1 | 2 * 0 + 1        0 | end
  1 | 2 * 0 + 1         0 | end
  0 | end
```

The results are:
$158_{(10)} \rightarrow 10011110_{(2)}$, $108_{(10)} \rightarrow 1101100_{(2)}$, $59_{(10)} \rightarrow 111011_{(2)}$

2.2.7 Binary arithmetic

Komputery pracujące w oparciu o układy cyfrowe dokonują większość obliczeń na liczbach w systemie dwójkowych. Pokażemy teraz sposób wykonywania podstawowych działań na liczbach zapisanych w ten sposób.

Dodawanie jest realizowane podobnie jak dla systemu dziesiętnego. Należy jedynie pamiętać, że

$$1_{(2)} + 1_{(2)} = 10_{(2)}.$$

Wynika to z faktu, iż w systemie dwójkowym nie ma cyfry reprezentującej liczbę 2, $1+1$ w tym systemie daje w wyniku 0 na pewnej pozycji, a jedność jest przenoszona na następną pozycję w liczbie. Jest to podoba sytuacja, jak w przypadku dodawania $1 + 9$ w systemie dziesiętnym: otrzymujemy w wyniku 0, a jedność jest przenoszona na następną pozycję. Przypatrzmy się następującym działaniom w przykładzie 2.2.

Przykład 2.2. Dodawanie w systemie dwójkowym.

```
   11
  00111000      01000001
+ 00010001    + 00010100
----------    ----------
  01001001      01010101
```

Odejmowanie, podobnie jak dodawanie, wykonywane jest według zasady identycznej, jak w systemie dziesiętnym, przy czym w przypadku odejmowania $0 - 1$ w systemie dwójkowym musimy dokonać zapożyczenia 1 na następnej pozycji liczby. Obliczenia zawiera przykład 2.3

Przykład 2.3. Odejmowanie w systemie dwójkowym.

```
  00111001      00101101
- 00001101    - 00010001
----------    ----------
  00101100      00011100
```

Mnożenie jest wykonywane analogicznie jak mnożenie w systemie dziesiętnym, ilustruje to przykład 2.4.

Przykład 2.4. Mnożenie w systemie dwójkowym.

```
       1111           10001
     *  101          *    11
-------------    ---------------
       1111           10001
      0000       +   10001
+    1111         ---------------
-------------          110011
    1001011
```

Dzielenie, podobnie jak mnożenie, wykonujemy tak samo jak w przypadku dzielenia w systemie dziesiętnym (przykład 2.5).

Przykład 2.5. Dzielenie w systemie dwójkowym.

```
    110                     1011
  -----                   -------
  10010:11=00000110       1111001:1011=1011
-  11                   - 1011
-------                 ---------
    11                     10000
-   11                   -  1011
-------                 ---------
     0                       1011
                         -   1011
                         ---------
                                0
```

2.2.8 Zapis liczby rzeczywistej w systemie dwójkowym

Na potrzeby tego podrozdziału przyjmujemy następujące założenie: mówiąc *liczba rzeczywista* mamy na myśli bezznakową liczbę rzeczywistą złożoną z części całkowitej i ułamkowej[4].

Najpierw spróbujmy zrozumieć dziesiętną reprezentację liczby rzeczywistej. Dziesiętna reprezentacja liczby rzeczywistej r jest wyrażeniem postaci

$$r = I, d_{-1}d_{-2}d_{-3}\ldots$$

gdzie I stanowi **część całkowitą** liczby r wyrażoną w postaci dziesiętnej, natomiast a_{-1}, a_{-2}, a_{-3},... są cyframi tworzącymi **część ułamkową** liczby r. Obie części rozdziela separator, tj. znak przecinka (,). Wartość v takiego wyrażenia obliczamy według wzoru

$$v = v_I + d_{-1}10^{-1} + d_{-2}10^{-2} + d_{-3}10^{-3} + \ldots,$$

gdzie v_I jest wartością części całkowitej I. Ponieważ I wyrazić możemy jako

$$I = \cdots + d_3 10^3 + d_2 10^2 + d_1 10^1 + d_0 10^0$$

[4]Podobnie jak dla liczb całkowitych, nie należy postrzegać prezentowanego zapisu liczby rzeczywistej jako tego, który jest stosowany w komputerze w sposób bezpośredni.

więc ostatecznie otrzymujemy

$$\begin{aligned} v &= \cdots + d_3 10^3 + d_2 10^2 + d_1 10^1 + d_0 10^0 \\ &\quad + d_{-1} 10^{-1} + d_{-2} 10^{-2} + d_{-3} 10^{-3} + \ldots \end{aligned}$$

co w zwięzłej postaci zapisujemy jako

$$v = \sum_{i=-\infty}^{\infty} d_i \cdot 10^i$$

Obserwacja 2.4. *Zapis część ułamkowej jest podobny do zapisu części całkowitej, ale używamy ujemnych wykładników zamiast dodatnich.*

Jeżeli teraz nasze rozważania przeniesiemy do systemu dwójkowego, to wartość części ułamkowej będzie wyliczana na podobnej zasadzie (zmieni się jedynie podstawa), zatem:

$$v = c_{-1} 2^{-1} + c_{-2} 2^{-2} + \cdots + c_{-n} 2^{-n}$$

a wartość całej liczby to

$$v = \sum_{i=-\infty}^{\infty} d_i \cdot 2^i$$

Oczywiście w tym wypadku cyfry są elementami zbioru $\{0, 1\}$.

Obserwacja 2.5. *Binarne liczby rzeczywiste tworzy się i interpretuje analogicznie do dziesiętnych liczb rzeczywistych, ale zamiast podstawy 10 używa się jako podstawy liczby 2.*

Stosując powyższy wzór jesteśmy w stanie bezpośrednio zamienić ułamek zapisany w systemie dwójkowym na system dziesiętny. Rozważmy poniższy przykład:

Przykład 2.6. Zamiana ułamka dwójkowego na dziesiętny.

$$0,11_{(2)} = 2^{-1} \cdot 1 + 2^{-2} \cdot 1 = 0,5 + 0,25 = 0,75_{(10)}$$

Do wyjaśnienia pozostaje sposób zamiany w drugą stronę, czyli jak zamieniać ułamek dziesiętny na dwójkowy. Ograniczymy się w tym miejscu wyłącznie do ułamków dodatnich. Schemat postępowania jest podobny do przedstawionych wcześniej. Mamy następujący algorytm (ułamek zapisany dziesiętnie oznaczmy przez x):

1. Start.
2. Niech $w = x$.
3. Mnożymy w przez 2.
4. Jeśli wynikiem operacji mnożenia jest liczba większa od jedności, zapisujemy na boku 1.
5. Jeśli wynikiem operacji mnożenia jest liczba mniejsza od jedności, zapisujemy na boku 0.
6. Ułamkową część wyniku – po odrzuceniu ewentualnej części całkowitej – zapisujemy jako w.
7. Jeśli w jest różne od 0, przechodzimy z powrotem do kroku 2.
8. Jeśli w jest równe 0, kończymy procedurę.
9. Koniec.

Następnie otrzymane zera i jedynki zapisywane w kolejności otrzymywania od lewej do prawej stanowią ułamek zapisany w systemie dwójkowym. Zilustrujmy powyższy algorytm przykładem.

Przykład 2.7. Zamiana ułamka dziesiętnego na ułamek dwójkowy.

```
0,75 => ?
                  liczba dwójkowa
                  |
0,75 | 2 * 0,75 = 1,5 <- część ułamkową przepisujemy
0,5  | 2 * 0,5  = 1,0
0,0  | koniec
```

Jak widać, w przykładzie 2.7, otrzymaliśmy wartość $0,11$, która odpowiada tej z przykładu 2.6, zatem jednocześnie dokonaliśmy sprawdzenia.

Przedstawmy jeszcze jeden przykład dla liczby $0,40625_{(10)}$, otrzymujemy:

0,40625	2 * 0,40625 = 0,8125
0,8125	2 * 0,8125 = 1,625
0,625	2 * 0,625 = 1,25
0,25	2 * 0,25 = 0,5
0,5	2 * 0,5 = 1,0
0,0	koniec

Zatem $0,40625_{(10)} = 0,01101_{(2)}$, sprawdźmy:

$$2^{-1} \cdot 0 + 2^{-2} \cdot 1 + 2^{-3} \cdot 1 + 2^{-4} \cdot 0 + 2^{-5} \cdot 1 = 0,25 + 0,125 + 0,03125 = 0,40625.$$

Przejdźmy teraz do zamiany liczb rzeczywistych większych od jednego. W takim przypadku stosujemy dla części całkowitej poznany wcześniej proces, a dla części ułamkowej omówiony powyżej.

Przykład 2.8. Zamiana liczby rzeczywistej dziesiętnej na postać dwójkową.

```
9,25 => ?

 9 | 2 * 4 + 1      0,25 | 2 * 0,25 = 0,5
 4 | 2 * 2 + 0       0,5 | 2 * 0,5 = 1,0
 2 | 2 * 1 + 0       0,0 | koniec
 1 | 2 * 0 + 1
 0 | koniec
```

Zatem otrzymujemy ostatecznie $9,25_{(10)} = 1001,01_{(2)}$.

Na koniec zauważmy, że nie każdą liczbę, którą da się zapisać w postaci ułamka dziesiętnego, da się dobrze przedstawić w postaci ułamka dwójkowego, co ilustruje przykład 2.9.

Przykład 2.9. Zamiana ułamka dziesiętnego na ułamek dwójkowy okresowy.

```
0,3 | 2 * 0,3 = 0,6
0,6 | 2 * 0,6 = 1,2
0,2 | 2 * 0,2 = 0,4
0,4 | 2 * 0,4 = 0,8
0,8 | 2 * 0,8 = 1,6
0,6 | ...
```

Zatem, jak widać, ułamek może mieć „dobrą” postać dziesiętną liczby 0,3, zaś „złą” – dwójkową 0,01001... W powyższym przykładzie ułamek dwójkowy jest okresowy, zatem w którymś momencie musimy zaprzestać procedury, gdyż inaczej prowadzilibyśmy ją w nieskończoność. Jeśli jednak zaprzestaniemy, to będzie to niedokładna reprezentacja wyjściowego ułamka. Dla przykładu ograniczmy się do pięciu cyfr po przecinku, otrzymamy wtedy:

$$0,01001_{(2)} = 0,25 + 0,03125 = 0,28125_{(10)}.$$

Jak widać błąd takiego zaokrąglenie wcale nie jest mały, jednak pamiętajmy, że nie jest to „wina” systemu dwójkowego, lecz samego procesu dzielenia

w ogóle. Zauważmy, że rozwinięcie dziesiętne ułamka zwykłego 1/3 również będzie przybliżone, jeśli chcemy je przechować na skończonym nośniku danych.

2.2.9 Kod szesnastkowy

Wadą systemu dwójkowego jest „rozwlekłość" zapisywanych w ten sposób liczb. Dla człowieka bywa trudne zapamiętanie ciągu zer i jedynek. Stąd też często w praktyce informatyka spotykamy się z zapisem liczby w **systemie szesnastkowym** lub **kodzie szesnastkowym**. Z definicji pozycyjnego systemu liczbowego (zob. podrozdział **??**) wynika, że w systemie szesnastkowym musimy mieć szesnaście cyfr, ale jak sobie poradzić w sytuacji, gdy mamy do dyspozycji cyfry arabskie $(0, 1, \ldots, 9)$. Problem ten rozwiązano poprzez nazwanie kolejnych cyfr tego systemu, poczynając od dziesięciu, kolejnymi literami alfabetu, zatem $A_{(16)} = 10_{(10)}, B_{(16)} = 11_{(10)}$ itd., dopuszcza się również stosowanie małych liter zamiast dużych.

Można oczywiście zapytać, czemu nie stosujemy w informatyce systemu osiemnastkowego, przyjrzyjmy się poniższej tabelce, a wszystko się wyjaśni.

Zapis dwójkowy	Zapis szesnastkowy	Zapis dziesiętny
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	*A*	10
1011	*B*	11
1100	*C*	12
1101	*D*	13
1110	*E*	14
1111	*F*	15

Widać z powyższego zestawienia, że jedna cyfra kodu szesnastkowego odpowiada dokładnie czterocyfrowej liczbie systemu dwójkowego. Ponieważ

bajt informacji (zob. dodatek **??**) składa się z 8 bitów, zatem każdy bajt danych daje się w sposób jednoznaczny przedstawić za pomocą dwu cyfr kodu szesnastkowego, co upraszcza kwestię konwersji – nie wymaga ona w praktyce obliczeń, a jedynie podstawienia. Oto przykład

$$00011101_{(2)} = 1D_{(16)},\ 1011111_{(2)} = BF_{(16)},\ 11100011_{(2)} = E3_{(16)}.$$

2.2.10 Inne pozycyjne systemy liczbowe

Punkt ten należy traktować jako informację poszerzającą wiedzę ogólną, gdyż inne systemy liczbowe niż dwójkowy i szesnastkowy praktycznie nie są wykorzystywane w informatyce lub ich obecność jest marginalna. Dla przykładu niekiedy można spotkać zapis liczby w systemie ósemkowym.

Korzystając z definicji pozycyjnego systemu liczbowego, zapiszemy tą samą liczbę w systemach pozycyjnych o różnych podstawach. Przykład 2.10 zawiera ilustrację tego procesu.

Przykład 2.10. Liczba 758 zapisana w pozycyjnych systemach liczbowych o różnych podstawach.

$$\begin{aligned} 23312_{(4)} &= 2 \cdot 4^4 + 3 \cdot 4^3 + 3 \cdot 4^2 + 1 \cdot 4^1 + 2 \cdot 4^0 = 758_{(10)} \\ 1366_{(8)} &= 1 \cdot 8^3 + 3 \cdot 8^2 + 6 \cdot 8^1 + 6 \cdot 8^0 = 758_{(10)} \\ 2F6_{(16)} &= 2 \cdot 16^2 + 15 \cdot 16^1 + 6 \cdot 16^0 = 758_{(10)} \end{aligned}$$

Jedną z możliwych metod przejścia od zapisu w systemie o podstawie p_1 do systemu o podstawie p_2 jest zamiana zapisu liczby na system dziesiętny, a dopiero potem na system o podstawie p_2. Metoda ta wydaje się naturalna, gdyż ludzie najsprawniej operują systemem dziesiętnym. Ilustruje to poniższy przykład

Przykład 2.11. Zamiana liczby z systemu 25 na system 11 poprzez system dziesiątkowy.

Wpierw zamieniamy liczbę z systemu o podstawie 25 na liczbę w systemie dziesiątkowym

$$2K_{(25)} = 2 \cdot 25^1 + K \cdot 25^0 = 2 \cdot 25 + 20 \cdot 1 = 70_{(10)}$$

następnie z systemu dziesiątkowego na jedenastkowy

```
70 | 11 * 6 + 4
 6 | 11 * 0 + 6
```

Ostatecznie więc otrzymujemy

$$2K_{(25)} = 64_{(11)}$$

Zauważmy jednocześnie, że w powyższym przykładzie, przy zapisie liczby w systemie dwudziestopiątkowym, przyjęliśmy podobną notację cyfr, jak w systemie szesnastkowym, zatem wykorzystaliśmy kolejne litery alfabetu.

W szczególnych przypadkach można proces ten jednak znacznie uprościć. Przyjrzyjmy się przykładowym liczbom w zapisie dwójkowym, czwórkowym i dziesiętnym.

Zapis dwójkowy	Zapis czwórkowy	Zapis dziesiętny
00	0	0
01	1	1
10	2	2
11	3	3

$$11011001_{(2)} \to 3121_{(4)},$$
$$3321_{(4)} \to 11111001_{(2)}.$$

Zauważmy, że zamieniając liczbę dwójkową na czwórkową, grupujemy bity po dwa, a następnie te dwójki zapisujemy za pomocą odpowiednich liczb czwórkowych (podobnie czyniliśmy to w zamianie z systemu dwójkowego na szesnastkowy). Zapisując liczbę czwórkową jako dwójkową, każdą jej cyfrę przedstawiamy za pomocą odpowiedniego dwuwyrazowego ciągu zero-jedynkowego.

```
11 01 10 01        3  3  2  1
|  |  |  |         |  |  |  |
3  1  2  1         11 11 10 01
```

Teraz zapiszmy kolejne liczby w systemach: dwójkowym, ósemkowym i dziesiętnym.

Zapis dwójkowy	Zapis ósemkowy	Zapis dziesiętny
000	0	0
001	1	1
010	2	2
011	3	3
100	4	4
101	5	5
110	6	6
111	7	7

W tym przypadku, aby dokonać przeliczenia, możemy grupować cyfry kodu dwójkowego po 3.

```
011 011 001
 |   |   |
 3   3   1
```

Otrzymujemy zatem

$$11011001_{(2)} \to 331_{(8)},$$
$$331_{(8)} \to 11011001_{(2)}.$$

Do konwersji z systemu dziesiętnego na inny pozycyjny system liczbowy, stosujemy analogiczny sposób, jak opisany w pkt **??**, z tą różnicą, że tym razem resztą z dzielenia mogą być nie tylko cyfry zero lub jeden (przykład 2.12).

Przykład 2.12. Przykład ilustrujący zamianę liczby 758 zapisanej dziesiętnie na system czwórkowy, ósemkowy i szesnastkowy.

758	4 * 189 + 2	758	8 * 94 + 6	758	16 * 47 + 6
189	4 * 47 + 1	94	8 * 11 + 6	47	16 * 2 + 15
47	4 * 11 + 3	11	8 * 1 + 3	2	16 * 0 + 2
11	4 * 2 + 3	1	8 * 0 + 1		
2	4 * 0 + 2				

Zatem otrzymujemy:
$758_{(10)} \to 23312_{(4)}$, $758_{(10)} \to 1366_{(8)}$, $758_{(10)} \to 2F6_{(16)}$

Nic nie stoi na przeszkodzie, aby stosować systemy liczbowe o podstawach różnych od potęgi 2. Niech np. liczba $2K$ będzie liczbą zapisaną w systemie o podstawie 25. Zapiszemy ją teraz jako liczbę systemu o podstawie 11. Najbardziej naturalną drogą, zasugerowaną na początku tego podrozdziału jest konwersja: $(25) \to (10) \to (11)$,

$$2K_{(25)} = 2 \cdot 25^1 + K \cdot 25^0 = 2 \cdot 25 + 20 \cdot 1 = 70_{(10)}.$$

70	11 * 6 + 4
6	11 * 0 + 6

Ostatecznie otrzymujemy, że $2K_{(25)} = 64_{(11)}$.

Innym podejściem jest pominięcie pośredniego kroku w postaci przeliczania na system dziesiętny i operowanie od razu wyłącznie na systemie

źródłowym i docelowym. W trakcie konwersji z systemu o mniejszej podstawie do większej nie nastręcza to żadnych trudności. W tym wypadku ma zastosowanie wzór (**??**), jednak należy pamiętać, że wszelkie obliczenia muszą być prowadzone w arytmetyce docelowego systemu liczbowego.

Rozważmy poniższy przykład, niech będzie dana liczba $1111101_{(2)}$, jak widać zapisana w systemie dwójkowym. Załóżmy teraz, że chcemy dokonać konwersji na system szesnastkowy. Napiszmy wzór przedstawiający wartość liczby

$$w = 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0,$$

teraz wszystkie obliczenia po prawej stronie równości należy prowadzić w arytmetyce szesnastkowej. Zatem

$$w = 40 + 20 + 10 + 8 + 4 + 0 + 1 = 70 + D = 7D_{(16)},$$

co jak widać zgadza się ze zwykłą regułą prezentowaną w pkt 2.2.9.

Niestety sytuacja komplikuje się przy dokonywaniu odwrotnej konwersji, wtedy bowiem dzielimy liczbę w systemie źródłowym przez podstawę systemu docelowego, oraz musimy pamiętać o resztach, których liczba, jaką się rozważa, jest zależna od systemu docelowego. Weźmy liczbę $96_{(16)}$ i zamieńmy ją na system trójkowy. W tym celu przeprowadzamy następującą serię dzieleń

```
 96 | 3 * 32 + 0
 32 | 3 * 10 + 2
 10 | 3 * 05 + 1
  5 | 3 * 01 + 2
  1 | 3 * 00 + 1
```

Zatem $96_{(16)} = 12120_{(3)}$. Wykonajmy sprawdzenie powyższej konwersji, zamieniając postacie w obu systemach na system dziesiętny

$$96_{(16)} = 9 \cdot 16^1 + 6 \cdot 16^0 = 144 + 6 = 150_{(10)},$$

$$12120_{(3)} = 1 \cdot 3^4 + 2 \cdot 3^3 + 1 \cdot 3^2 + 2 \cdot 3^1 + 0 = 81 + 54 + 9 + 6 + 0 = 150_{(10)}.$$

2.3 Kod BCD

Do tej pory zapoznaliśmy się z systemem dwójkowym, szesnastkowy oraz innymi. Jak można było zaobserwować, istotnych różnic pomiędzy nimi nie ma i w zasadzie to jesteśmy w stanie operować dowolnym z nich, a nawet kilkoma, dokonując odpowiednich konwersji przy przejściu od jednego do drugiego. Mimo to cały czas odczuwa się jak gdyby wrodzoną skłonność do operowania systemem dziesiętnym, przejawiającą się chociażby w tym, że w konwersji najczęściej stosujemy schemat **podstawa_źródłowa → podstawa_10 → podstawa_docelowa**. Cały kłopot z systemami liczbowymi o podstawach innych niż 10 polega na tym, że ciąg cyfr jest dla nas abstrakcyjnym napisem, który nabiera znaczenia dopiero wtedy, gdy przekształcimy go do postaci dziesiętnej[5].

Zilustrujemy problemy, o których mówimy za pomocą prostego testu. Kto bez zastanowienia odpowie, ile wynosi 1/3 z liczby $111100_{(2)}$? Jak więc widać, systemem dwójkowym operujemy z konieczności, a najchętniej (najsprawniej) używamy dziesiętnego. Problemem jest konwersja z jednego na drugi. Chcąc ułatwić to zadanie wprowadzono system kodowania BCD (ang. *binary-coded decimal*). W tym systemie liczby dziesiętne kodujemy za pomocą ciągu bitów przypisując każdej cyfrze dziesiętnej odpowiadający jej ciąg 4 bitów (patrz tab. 2.2). Ponieważ mamy jedynie 10 możliwości, więc wystarczy zapamiętać tylko tyle, aby móc sprawnie i szybko przechodzić z systemu dziesiętnego do BCD i odwrotnie. Zobaczmy, jak odbywa się ten proces. Zapiszmy liczbę dziesiętną 120 najpierw w systemie dwójkowym, a następnie w formacie BCD. Zatem $120_{(10)} = 1111000_{(2)}$.

[5] Oczywiście wynika to z przyzwyczajenia i wychowania w określonej kulturze. Kiedyś dla przykładu liczono tuzinami czy kopami.

Cyfra dziesiętna	„Cyfra” BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
–	1010
–	1011
–	1100
–	1101
–	1110
–	1111

Tablica 2.2. Cyfry dziesiętne i odpowiadające im „cyfry” BCD

Przykład 2.13. Zamiana liczby dziesiętnej na jej postać w kodzie BCD.

$1_{(10)} = 0001_{(\mathrm{BCD})}$
$2_{(10)} = 0010_{(\mathrm{BCD})}$
$0_{(10)} = 0000_{(\mathrm{BCD})}$

Zatem: $120_{(10)} = 00010010000_{(\mathrm{BCD})}$

Łatwo można poczynić następujące spostrzeżenia:

- Pomimo, iż kod BCD do zapisu używa cyfr dwójkowych, to otrzymany ciąg różny jest od odpowiadającej liczby dwójkowej.
- Konwersja $10 \rightarrow \mathtt{BCD}$ lub $\mathtt{BCD} \rightarrow 10$ odbywa się na podobnych zasadach, jak konwersja $2 \rightarrow 16$ lub $16 \rightarrow 2$ – grupujemy bity po 4.
- Kod BCD wymaga więcej pamięci niż „tradycyjny” zapis dwójkowy.
- Konwersja jest naturalna i odbywa się prawie natychmiast.

W kodzie BCD oczywiście można wykonywać działania arytmetyczne. Ilustruje to przykład 2.14, w którym pokazano dodawanie w kodzie BCD i właśnie do dodawania ograniczymy się przy omawianiu działań.

Przykład 2.14. Dodawanie bez przeniesienia w kodzie BCD.

```
142 (10) = 000101000010 (BCD)
311 (10) = 001100010001 (BCD) +
--------------------------------
           010001010011
           |   |   |
           |   |   0011 (BCD) = 3 (10)
           |   |
           |   0101 (BCD) = 5 (10)
           |
           0100 (BCD) = 4 (10)
```

Niestety nie zawsze wszystko układa się tak dobrze... Zauważmy, że możemy w wyniku dodawania uzyskać na pewnej pozycji liczbę większą od 9, co spowoduje konieczność dokonania przeniesienia (przykład 2.15).

Przykład 2.15. Ilustracja wystąpienia przeniesienia w dodawaniu w kodzie BCD.

```
 9 (10) = 1001 (BCD)
 6 (10) = 0110 (BCD) +
----------------------
15 (10) = 1111 (BCD) ????
```

Otrzymany w przykładzie 2.15 wynik nie jest poprawną „cyfrą" kodu BCD! Pojawia się konieczność korekcji otrzymanych wyników. Korekcja polega na dodaniu liczby dziesiętnej 6 (`0110 (BCD)`) do niepoprawnej grupy 4 bitów; ma to miejsce, gdy „cyfra" BCD jest większa od `1001` lub gdy nastąpi przeniesienie z jednej czwórki bitów do następnej, ilustruje to przykład 2.16.

Przykład 2.16. Dodawanie w kodzie BCD z uwzględnieniem przeniesienia.

```
 60 (10) = 01100000 (BCD)
 55 (10) = 01010101 (BCD) +
----------------------------
115 (10) = 10110101
  korekcja 0110         +
     --------------------
          100010101
       000100010101
       |   |   |
       |   |   0101 (BCD) = 5 (10)
       |   |
       |   0001 (BCD) = 1 (10)
       |
       0001 (BCD) = 1 (10)
```

```
 99 (10) =  10011001 (BCD)
 99 (10) =  10011001 (BCD) +
----------------------------
189 (10) = 100110010
  korekcja  01100110 +
  ---------------------
           110011000
        000110011000
        |   |   |
        |   |   1000 (BCD) = 8 (10)
        |   |
        |   1001 (BCD) = 9 (10)
        |
        0001 (BCD) = 1 (10)
```

```
 99 (10) = 10011001 (BCD)
 11 (10) = 00010001 (BCD) +
---------------------------
110 (10) = 10101010
  korekcja 01100110 +
 ---------------------
          100010000
       000100010000
       |   |   |
       |   |   0000 (BCD) = 0 (10)
       |   |
       |   0001 (BCD) = 1 (10)
       |
       0001 (BCD) = 1 (10)
```

2.4 Zadania

1. Wykonaj uproszczenie następujących wyrażeń algebraicznych, a następnie sprawdź prawdziwość metodą zero-jedynkową:
   a) $xz + xy + y\overline{z}$,
   b) $xyz + x\overline{y} + y\overline{z}$,
   c) $xyz + x\overline{y}z + xy\overline{z}$,
   d) $xy + x\overline{y} + \overline{x}y$,
   e) $xz + \overline{z}x + yz + y\overline{z} + \overline{x}$.

2. Dokonaj konwersji zapisu następujących liczb z systemu o podstawie 10 na liczby w systemach o podstawach 2, 8 i 16:
   a) 172, b) 517, c) 778, d) 13, e) 76,
   f) 107, g) 300, h) 201, i) 472, j) 802.

3. Dokonaj konwersji zapisu następujących liczb z systemu o podstawie 2 na liczby w systemach o podstawach 8, 16 i 10:
   a) 11100, b) 1011110001,
   c) 10001010001, d) 100100100,
   e) 1110011100, f) 101110001,
   g) 1001001100, h) 101010000,
   i) 10100001, j) 1000101.

4. Dokonaj konwersji zapisu następujących liczb z systemu o podstawie 16 na liczby w systemach o podstawach 2, 8 i 10:
   a) $F2$, b) $11F$, c) AAA, d) 100, e) $1AB$,
   f) 123, g) FF, h) $F0$, i) BAB, j) 307.

5. Dokonaj konwersji zapisu następujących liczb z systemu o podstawie 8 na liczby w systemach o podstawach 2, 16 i 10:
   a) 123, b) 457, c) 177, d) 65, e) 22,
   f) 10, g) 27, h) 55, i) 222, j) 512.

6. Dokonaj konwersji zapisu liczby z systemu o podstawie 7 na system o podstawie 5:
   a) 565, b) 100, c) 62, d) 12, e) 166,
   f) 306, g) 255, h) 32, i) 836, j) 56.

7. Dokonaj konwersji zapisu liczby z systemu o podstawie 5 na system o podstawie 11:
   a) 1234, b) 4222, c) 2131, d) 1421, e) 3123,
   f) 1121, g) 2041, h) 4131, i) 3211, j) 3114.

8. Dokonaj konwersji zapisu liczby z systemu o podstawie 13 na system o podstawie 9:
   a) $C99$, b) $2A5$, c) 91, d) 65, e) $3BC$,
   f) 910, g) $B7$, h) 18, i) 301, j) $40C$.

Rozdział 3

Algorytmy i struktury danych

Programy stanowią w końcu
skonkretyzowane sformułowania
abstrakcyjnych algorytmów na
podstawie określonej reprezentacji
i struktur danych.

Niklaus Wirth

3.1 Pojęcie algorytmu

Słowo **algorytm**, jak to wyjaśniono w pkt **??**, pochodzi od nazwiska perskiego astronoma i matematyka żyjącego na przełomie VIII i IX w n.e. W 825 roku Muhammad ibn Musa al-Chorezmi (al-Khawarizmy) napisał traktat, w którym podał wiele precyzyjnych opisów dotyczących różnych matematycznych reguł (np. dodawania czy mnożenia liczb dziesiętnych). W XII wieku dzieło to zostało przetłumaczone na łacinę jako *Algoritmi de numero Indorum*, co należało rozumieć następująco: *Algoritmi o liczbach Indyjskich*. Pojawiające się tutaj po raz pierwszy słowo *Algoritmi* było oczywiście inaczej zapisanym nazwiskiem matematyka. Większość ludzi rozumiała jednak tytuł bardziej jako *Algorytmy o liczbach Indyjskich* a stąd już blisko do *Algorytmy na liczbach arabskich*. W ten oto sposób precyzyjnie opisaną metodę obliczeniową zaczęto nazywć algorytmem (łac. *algorismus*).

Nie ma jednej uniwersalnej definicji algorytmu. W potocznym tego słowa znaczeniu, algorytm oznacza sposób postępowania, przepis na coś, schemat

działania. Przykładem może być tu wspominany algorytm Euklidesa znajdowania największego wspólnego dzielnika liczb (zob. pkt **??**) czy wyznaczanie podziału kąta na dwie równe części za pomocą cyrkla i linijki, ale równie dobrze algorytmem możemy nazwać procedurę wymiany uszkodzonego elementu w komputerze.

Pierwsze opisy, które później nazwano algorytmami, dotyczyły rozwiązań zadań matematycznych. Już w starożytnym Egipcie i Grecji stworzono wiele metod, które pozwalały rozwiązywać pewne zadania w sposób algorytmiczny. Intuicyjnie algorytm kojarzy się z metodą rozwiązywania zadania, z przepisem postępowania czy ze schematem działania. Należy jednak podkreślić, że nie każda metoda czy schemat są algorytmami. Algorytm – w matematyce oraz informatyce to skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego zadania. Powinien on spełniać następujące wymagania:

1. Musi posiadać określony stan początkowy, czyli operację, od której zaczyna się jego realizacja.
2. Liczba operacji potrzebnych do zakończenia pracy musi być skończona – **warunek dyskretności**.
3. Musi dać się zastosować do rozwiązywania całej klasy zagadnień, a nie jednego konkretnego zadania – **warunek uniwersalności**.
4. Interpretacja poszczególnych etapów wykonania musi być jednoznaczna – **warunek jednoznaczności**.
5. Cel musi być osiągnięty w akceptowalnym czasie – **warunek efektywności**[1]. Co oznacza *akceptowalny* to zależy oczywiście od postawionego zadania.
6. Musi posiadać wyróżniony koniec.

W przeszłości matematycy czasami próbowali rozwiązywać różne problemy w sposób algorytmiczny, ale na ogół nie przywiązywano większej wagi do precyzji opisu kolejnych kroków procedury. Wraz z rozwojem matematyki pojawiły się problemy, co do których nie było pewności, czy można je rozwiązać w sposób algorytmiczny, czy też nie. Dopiero wówczas zaczęto zastanawiać się nad tym, jak precyzyjnie sformułować pojęcie algorytmu. Aby wykazać, że dany problem można rozwiązać w sposób algorytmiczny wystarczy podać konkretny algorytm rozwiązania tego problemu. Jeśli jednak

[1] Warunek ten nie jest konieczny dla poprawności działania algorytmu, ale z drugiej strony nikt nie będzie używał poprawnego algorytmu, na którego wynik będzie musiał czekać „kilkaset" lat.

chcemy udowodnić, że danego problemu nie da się rozwiązać algorytmicznie potrzeba dowieść, że nie istnieje algorytm rozwiązania danego problemu. A to powoduje, że potrzebna jest ścisła definicja algorytmu. W latach dwudziestych XX w. zagadnienie ścisłego, w sensie matematycznym, określenia pojęcia algorytmu było jednym z głównych problemów matematyki. Definicje te zostały opracowane m. in. przez Alana Turinga, Emila Posta i Alonzo Churcha. Na podstawie tych definicji wykazano, że w matematyce istnieje wiele problemów, których nie można rozwiązać w sposób algorytmiczny.

Zainteresowanie algorytmami wzrastało wraz z rozwojem informatyki. W latach pięćdziesiątych pojawił się nowy wykonawca algorytmu – komputer. Oznaczało to z jednej strony potencjalny wzrost prędkości przetwarzania danych, a co za tym idzie coraz więcej algorytmów stawało się rozwiązywalnymi w skończonym czasie. Lecz z drugiej strony wymagało to zarówno dopracowania formalizmu definicji algorytmów, jak i powstania języków programowania, za pomocą których te algorytmy miały być zapisywane i wpisywane do komputera.

Miejsce algorytmu w procesie tworzenia programu (implementowania algorytmu) wyróżniono na poniższej liście (proszę porównać ją z pierwszymi akapitami kolejnego rozdziału).

1. Problem.
2. Komputer (czas, wewnętrzna reprezentacja danych, oprogramowanie).
3. Język (dostępne konstrukcje i typy danych).
4. **Algorytm.**
5. Program.

Jak to zostało napisane w rozdziale **??**, komputer dysponuje pewnym określonym zestawem operacji. Aby więc przekazać mu do wykonania jakieś zadanie, należy stworzyć algorytm, a następnie zapisać go za pomocą takich instrukcji, które dany komputer potrafi wykonać; instrukcje te muszą być przekazane w formie zrozumiałej dla niego. Wyrażony w ten sposób algorytm nazywa się **programem**[2]. Początkowo jedynym językiem komunikowania się człowieka z komputerem był język wewnętrzny maszyny, a

[2]Oczywiście w skład dużego programu może wchodzić wiele pomniejszych algorytmów, jednak można je wszystkie postrzegać jako jeden złożony algorytm realizujący pewne złożone zadanie.

następnie asembler (zob. pkt **??**). Jednak trudności, jakie temu towarzyszyły, doprowadziły szybko do powstania wielu nowych, niezależnych od konkretnego komputera, języków programowania (zob. rozdział **??**).

Wraz z rozwojem informatyki algorytm stał się pojęciem podstawowym. Jest on, z jednej strony, narzędziem pracy informatyka, służącym do rozwiązywania za pomocą komputera różnego rodzaju problemów, a z drugiej zaś stanowi przedmiot badań. Interesujące jest nie tylko, jak skonstruować algorytm rozwiązujący dane zagadnienie, ale także czy jest on poprawny oraz realizowalny w skończonym czasie, tzn. czy daje prawidłowe rezultaty i czy czas jego wykonania jest „akceptowalny". Problematyka związana z projektowaniem, oceną jakości i poprawności algorytmów nosi nazwę **analizy algorytmów**.

3.2 Struktury danych

Dzisiejsze komputery w dużym stopniu służą do przechowywania i wyszukiwania informacji, których jest coraz więcej. Ponadto oczekujemy również możliwości porządkowania tej informacji i jej hierarchizacji. Wszystko po to, by móc znaleźć potrzebne dane szybciej i dokładniej[3]. Wszelka informacja przechowywana w komputerach jest tylko pewnym wycinkiem świata rzeczywistego. Można powiedzieć, iż stanowi ona jedynie pewien abstrakcyjny jego model. Co więcej model ten albo, inaczej mówiąc, wycinek rzeczywistości może być inny, w zależności od tego, dla jakich potrzeb będzie on konstruowany. Dla przykładu jeśli potrzebujemy stworzyć program do ewidencji pacjentów w szpitalu (abstrakcyjny model pacjenta), będziemy potrzebowali innych danych o fizycznie istniejącym pacjencie, niż w programie do ewidencji dłużników, choć w obu przypadkach możemy mówić o tej samej osobie. Zatem abstrakcja w tym miejscu oznacza ignorowanie tych cech danego rzeczywistego obiektu, które dla rozwiązania danego problemu nie są potrzebne.

W przypadku przystępowania do rozwiązania problemu za pomocą komputera należy najpierw zastanowić się, jakie dane będziemy przetwarzać, a następnie, w jaki sposób będą one w komputerze reprezentowane – czyli, jakie **struktury danych** będą je przechowywały. Często przy wyborze sposobu reprezentacji bierze się pod uwagę pewne ograniczenia maszyny, wynikające chociażby z architektury dzisiejszych komputerów. Dla przykładu

[3]Najprostszym przykładem może być poszukiwanie informacji za pomocą wyszukiwarek internetowych: zwykle znajdujemy wiele stron zawierających dane słowo. Jednak odszukanie naprawdę przydatnej z naszego punktu widzenia informacji spoczywa na nas.

wiemy, że liczby zmiennoprzecinkowe są reprezentowane z pewną dokładnością. Zatem wybór reprezentacji często nie jest łatwy, co więcej bywa często tak, że daną informację można reprezentować na kilka sposobów i każdy z nich ma pewne wady i zalety. Prowadząc dalej to rozumowanie dochodzimy do wniosku, że wybór struktury danych determinuje wybór algorytmów na nich operujących. To w konsekwencji prowadzi do określonej wydajności i sprawności całego programu.

Podsumowując powyższe:

- Informacja przechowywan w komputerze i przez niego przetwarzana stanowi pewien niewielki wycinek rzeczywistości zawierający dane niezbędne do rozwiązania postawionego problemu.
- Musimy się zastanowić jakie informacje są nam niezbędne, jakie mogą pomóc a jakie są całkiem niepotrzebne.
- Musimy się zastanowić jak będziemy reprezentować wybrane przez nas informacje.

3.2.1 Pojęcie typu danych

Jeśli przypomnimy sobie pewne elementarne pojęcia z matematyki, to wśród nich znajdziemy takie elementy, jak liczby rzeczywiste czy liczby całkowite. Dalej mówimy o zmiennych całkowitych lub zmiennych rzeczywistych, czyli, elementach zbioru liczb całkowitych i rzeczywistych. W przypadku algorytmów przetwarzających dane i dalej języków programowania, w których te algorytmy są zapisywane, wprowadza się również pojęcie zmiennej i jej typu, czyli zbioru dopuszczalnych dla niej wartości[4].

Zwykle wyróżnia się **typy proste** nazywane też **typami wbudowanymi** oraz **typy złożone**. Do typów prostych zalicza się z reguły: **typy liczbowe**, takie jak całkowity czy zmiennoprzecinkowy; **typ znakowy** – dla zmiennych zapamiętujących znaki alfanumeryczne; **typ logiczny**[5] – dla zmiennych logicznych; i wiele innych, które są już zależne od konkretnego języka programowania.

[4]Istnieją języki programowania, w których nie ma konieczności informowania, jakiego typu jest dana zmienna, jednak nie będziemy ich tu szczegółowo omawiać. Według naszej opinii dezorientują one początkujących informatyków, a co najgorsze, uczą złych nawyków. Poza tym języki beztypowe wymagają pewnego doświadczenia, ze względu na łatwość popełniania w nich trudnych do znalezienie błędów.

[5]Zwany również typem boolowskim.

Z kolei typy złożone są konglomeratami zmiennych prostych lub innych typów złożonych[6], i tu zasadniczo wyróżnia się **tablicę**, **słownik**, **zbiór**, **rekord**, **klasa**, **plik**, **kolejkę**, **stos**, **drzewo**.

Omówimy teraz kolejno wymienione typy złożone, przy czym skupimy się wyłącznie na najważniejszych ich cechach, bez wchodzenia w niuanse implementacyjne czy też warianty tych typów. Szersze omówienie, zarówno struktur danych, jak i algorytmów, można znaleźć w wielu książkach, które wskazujemy w bibliografii.

3.2.2 Tablica

Tablica (ang. *array*) jest jedną z najpowszechniej stosowanych struktur danych, ponadto występuje na poziomie implementacyjnym w prawie wszystkich językach programowania. Jest to struktura danych pozwalająca zapamiętywać wyłącznie elementy tego samego typu[7]. Wyobraźmy sobie pojemnik na płyty CD, nie mamy możliwości włożenia do niego np. kasety VHS. Kolejną cechą tablic jest **dostęp swobodny** (ang. *random access*) do jej elementów – oznacza to, że dostęp do dowolnego elementu tablicy odbywa się w taki sam sposób oraz że można do nich się odwoływać w dowolnej kolejności. Tu znów posłużmy się analogią ze stojakiem na płyty CD: po każdą z płyt można sięgnąć w ten sam sposób – swobodnie. W przypadku tablic do odwołania się (sięgnięcia) do konkretnego jej elementu stosuje się **indeksowanie**, tzn. podajemy, oprócz nazwy tablicy, numer elementu, do którego chcemy się odwołać. Dla przykładu niech `T` będzie nazwą zmiennej tablicowej utworzonej w następujący sposób:

```
T: Array of Integer;
```

Tak zdefiniowana tablica `T` pozwala na przechowywanie liczb całkowitych. Odwołanie do elementu o indeksie 1 i zapisanie do niego wartości 3 będzie miało postać:

```
T[1] := 3
```

W prezentowanym przykładzie zastosowano znak :=, oznaczający przypisanie wartości, dla odróżnienia od znaku =, który będzie oznaczał porównanie dwu wartości.

[6] Stąd też wydaje się być uzasadnione używanie w stosunku do nich terminu *struktura danych*.

[7] Istnieją języki programowania umożliwiające przechowywanie w jednej tablicy obieków różnych typów, jednak dla jasności wywodów będziemy ich unikać, podobnie jak będziemy unikać języków beztypowych.

Zauważmy tutaj, iż w niektórych implementacjach należy podać rozmiar tablicy. Wówczas można w niej przechować jedynie określoną ilość elementów. Definicja pięcioelementowej tablicy obiektów będących liczbami całkowitymi może mieć postać:

```
T: Array[5] of Integer;
```

Powyższy zapis jest sprawą umowną, która zależy od przyjętej symboliki lub konkretnego języka programowania. Nie jest na przykład oczywiste od jakiej liczby rozpoczyna się indeksowanie tablicy; zwykle jest to 0 lub 1. Niektóre języki pozwalają na określenie zakresu indeksów, np. w następujący sposób:

```
T1: Array[2..6] of Real;
T2: Array[-6..-2] of Real;
```

W powyższym przykładzie utworzyliśmy dwie zmienne tablicowe (będące tablicą), pozwalające na przechowywanie pięciu liczb rzeczywistych, przy czym dla pierwszej znich indeksy zmieniają się od 2 do 6, dla drugiej zaś od -6 do -2.

Nic też nie stoi na przeszkodzie, by mówić o tablicach wielowymiarowych, w których każdy z indeksów może „przebiegać" inny przedział wartości. Intuicyjnie tablica dwuwymiarowa może być utożsamiana z występującą w matematyce macierzą. Jest to właśnie tablica dwuwymiarowa. Jako przykład tablicy trójwymiarowej można podać choćby kostkę Rubika, natomiast jeśli chodzi o większą liczbę wymiarów, to wyobraźnia już zawodzi[8]. Niemniej jednak można podać prosty przykład uzasadniający stosowanie tego typu konstrukcji. Rozważmy wielopiętrowy budynek – jego adres jest pierwszym wymiarem, następnie piętro – drugi wymiar, korytarz – trzeci, numer pokoju na korytarzu – czwarty. Oczywiście można ten przykład rozwinąć bez problemu na 5 czy 6 wymiarów, dołączając miasto, kraj itp. Tablicę indeksowaną podwójnie możemy zdefiniować np. w następujący sposób:

```
T: Array[2..6][3..7] of Integer;
```

Teraz podamy kilka przykładów zaczerpniętych z różnych języków programowania.

```
Ada:
-- definicja typu tablicowego
```

[8]Przynajmniej piszących te słowa autorów.

```
type TableType is array(1 .. 100) of Integer;
-- definicja zmiennej określonego typu tablicowego
MyTable : TableType;

Visual Basic:
Dim a(1 to 5,1 to 5) As Double
Dim MyIntArray(10) As Integer
Dim MySingleArray(3 to 5) As Single

C:
char my_string[40];
int my_array[] = {1,23,17,4,-5,100};

Java:
int [] counts;
counts = new int[5];

PHP:
$pierwszy_kwartal  = array(1 => 'Styczeń', 'Luty', 'Marzec');

Python:
mylist = ["List item 1", 2, 3.14]
```

3.2.3 Słownik

Spróbujmy najpierw zrozumieć czym jest słownik; potem podamy jak można go zapisać. Wyobraźmy więc sobie podełko. Pudełko to przeznaczonej jest do przechowywania obiektów różnego typu. Wszystkie obiekty w pudełku nie są poukładane jak w przypadku tablicy ale bezładnie porozrzucane. Dodatkowo jednak do każdego obiektu przywiązany jest sznurek zakończony karteczką. Na karteczce zaś jest tekst jednoznacznie identyfikujący obiekt na drugim końcu sznurka. Tekst ten nazwijmy **kluczem**. Wszystkie karteczki są w jakiś sposób poukładane, np. alfabetycznie, od największej do najmniejszej itd. – zależy to od naszych preferencji. Jeśli teraz zależy nam na konkretnym obiekcie z pudełka to nie szukamy go w nim, ale ciągniemy za sznureczek z odpowiednią karteczką.

Tak więc słownik może przechowywać obiekty będące parą **klucz-wartość**. Klucz jest jak gdyby identyfikatorem lub krótkim hasłem wskazującym na właściwy obiekt, czyli wartość. Obiekty w słowniku nie posiadają żadnej

kolejności. Ewentualna kolejność zależy bowiem od tego w jakiej kolejności będziemy odwoływać się do poszczególnych kluczy.

Poniżej przedstawiamy operacje na słowniku wykorzystujące składnię języka Python.

```
d = {"key1":"val1", "key2":"val2"}
x = d["key2"]
d["key3"] = 122
d[42] = "val4"
```

3.2.4 Rekord

Jak można było zauważyć pewną wadą tablicy jest konieczność przechowywania elementów tego samego typu[9]. W pewnych sytuacjach jest to niedogodne, ale dla nich został zaprojektowany **rekord** (ang. *record*), który jest najbardziej ogólną metodą na stworzenie struktury danych. W rekordzie istnieje możliwość zapamiętania dowolnej, ale określonej liczby danych dowolnych typów. Elementy występujące w rekordzie nazywamy jego **polami**. Jako przykład takich konstrukcji może służyć rekord danych osobowych, gdzie będzie zapamiętane imię, nazwisko, wiek itp. Dopuszcza się również, by elementami (składowymi) rekordu były inne typy złożone, czyli np. tablice, rekordy, zbiory itd.

Głównym przeznaczeniem rekordu jest powiązanie różnych informacji w jedną logiczną całość, tak by móc się do niej odwoływać wspólnie. Przykładowo, jeśli znajdziemy poszukiwaną osobę z imienia i nazwiska, z pewnością będą nas interesowały dodatkowe informacje z nią związane, jak np. adres zamieszkania[10].

Typ rekordowy będziemy definiowali w następujący sposób:

```
type osoba is record
  imie: napis;
  nazwisko: napis;
  wiek: Integer;
end
```

Powyższy przykład definiuje rekord składający się z imienia, nazwiska,

[9]Co prawda konkretne realizacje tablic w istniejących językach programowania dopuszczają wkładanie do tablicy elementów różnych typów, ale tak naprawdę są to nowe konstrukcje struktur złożonych, tzw. tablice z wariantami.

[10]Faktycznie adres zamieszkania także może być rekordem.

wieku[11]. W przeciwieństwie do tablicy w rekordzie nie ma mowy o 5., czy 2., elemencie, mówimy wyłącznie o polu o określonej nazwie. Zatem odwołanie od konkretnego pola odbywa się poprzez określenie, o jaki rekord nam chodzi i o jakie jego pole, zwykle te nazwy oddzielone są kropką[12]. Stosując definicję rekordu „osoba" z powyższego przykładu, odwołania do jego pól będą miały postać:

```
o: osoba;
wypisz(o.imie);
o.wiek := 12;
```

Powyższy zapis oznacza definicję zmiennej o nazwie „o", która jest typu „osoba", a następnie odwołanie się do imienia z tego rekordu i wypisanie go, po czym wpisanie wartości do pola „wiek" w tym rekordzie.

Wprowadza się również pojęcie rekordu z wariantami. Jest to taka konstrukcja, która może mieć pewne pola alternatywne, np. dla osoby można zapamiętać wiek, lub datę urodzenia. Jednak nie będziemy tej wersji omawiali szerzej – zainteresowany czytelnik może znaleźć opis tej struktury danych np. w książce [**?**].

3.2.5 Klasa

Wiemy już, że do opisu bardziej złożonych obiektów posiadających różnorodne cechy możemy wykorzystać rekordy. Często jednak jest tak, że specyficzne cechy obiektu wymagają specyficznych działań na nim wykonywanych. Co więcej, często działania te są słuszne tylko dla tego obiektu. Zatem dobrze by było aby móc zapisać zarówno cechy obiektu (jak to ma miejsce właśnie w przypadku rekordu) jak i funkcje, które obiekt ten może wykonywać. Do tego właśnie celu można użyć **klasę**.

I tak na przykład możemy użyć tej struktury jeśli chcemy opisać samochód wraz z jego cechami, takimi jak marka, nazwa, ilość miejsc, pojemność i moc silnika oraz pewnymi działaniami jakie chcemy na nim wykonać: uruchomić silnik, jechać do przodu, itd.

```
class Samochod:
  marka: String;
  pojemnosc: Float;
```

[11] W definicji użyto typu „napis", który wcześniej nie był zdefiniowany, ale przyjmujemy jego intuicyjne znaczenie.

[12] „Zwykle", ponieważ zależy to od przyjętej notacji, kropka jest stosowana w większości języków programowania.

```
  kolor: Color;

  Jedz(kierunek);
  StanLicznika();
  Stop();
  UruchomSilnik();
end
```

Wykorzystać tą strukturę możemy natomiast w następujący sposób:

```
x,i: Float;
i:=0;
myCar = new Samochod();
myCar.marka="Nissan";
myCar.pojemnosc=1.8;
myCar.UruchomSilnik();
x=myCar.StanLicznika();
myCar.Jedz("przod");
while(i<10)
begin
i=myCar.StanLicznika()-x;
end
myCar.Stop();
```

Podobnie jak to miało miejsce w przypadku rekordu, zarówno do własności obiektu jak i jego funkcji odwołujemy się przy pomocy operatora kropki. Zapis

```
myCar = new Samochod();
```

można utożsamiać z zapisem typu

```
nazwaZmiennej: Typ;
```

W przypadku klas mamy dodatkowo do czynienia z operatorem `new`. Nie będziemy omawiać jego w tym momencie przyjmując jego wystąpienie jako element składni (tak samo jak np. kropki stawiane na zakończenie zdania).

Do ważnych pojęć związanych z klasą należą

- **Obiekt** (ang. *object*) Obiekt tym różni się od klasy czym różni się np. słowo *samochód* od konkretnego samochodu stojącej na ulicy. Mówiąc *samochód* mamy na myśli pewien abstrakcyjny byt nazywany *samochodem* posiadający pewien specyficzny i charakterystyczny zestaw

cech. Samochód na ulicy to konkretny przykład (wcielenie, realizacja) tego czym samochód powinien być. Tak więc klasa, to w pewnym sensie pieczątka, natomiast obiekt (będące tzw. instancjami klasy) to konkretne odbicia tej pieczątki.

- **Dziedziczenie** (ang. *inheritance*) Umożliwia definiowanie i tworzenie specjalizowanych obiektów na podstawie obiektów bardziej ogólnych. Istotną zaletą jest to, że dla obiektu specjalizowanego nie trzeba definiować całej funkcjonalności, lecz tylko taką jej część, której nie ma obiekt ogólniejszy.

  Bez trudu możemy wyobracić sobie pewną hierarchie zależności związaną z pojazdami.

  Ogólnie rzecz biorąc, każdy `pojazd`, bez względu na przeznaczenie, budowe itp. posiada pewną `prędkość` i `położenie` w przestrzeni. Oprócz tego można za jego pomocą wykonać czynności: `poruszaj się` – czyli wprawienie jego w ruch, oraz`stój` – czyli zatrzymanie. Możemy to zapisać w następujący sposób:

  ```
  pojazd:
    własności (dane): prędkość, położenie
    działania: Stój, PoruszajSię
  ```

  2.

  pojazdCzterokołowy: taki sam jak typ pojazd oraz dodatkowo własności: ilośćDobrychKół działania: SkręćWLewo, SkręćWPrawo

  3.

  wodnosamolot: taki sam jak typ pojazd oraz dodatkowo własności: szybkośćTonięcia działania: Startuj, Ląduj

  4.

  pojazdKosmiczny: własności: zapasPowietrza

  5.

  pojzadMarsjański: taki sam jak typ pojazdCzterokołowy oraz taki sam jak typ pojazdKosmiczny oraz własności: stanNaładowaniaAkumulatorów działania: RozłóżBaterieSłoneczne, ŁadujAkumulatory

- **Abstrakcja** (ang. *abstraction*) Różne traktowanie tego samego obiektu Samochód–Nissan–środek transportu.

- **Enkapsulacja** (ang. *encapsulation*) Czyli ukrywanie implementacji, hermetyzacja. Zapewnia, że obiekt nie może zmieniać stanu wewnętrznego innych obiektów w nieoczekiwany sposób. Tylko wewnętrzne metody obiektu są uprawnione do zmiany jego stanu. Każdy typ obiektu prezentuje innym obiektom swój interfejs, który określa dopuszczalne metody współpracy. Pewne języki osłabiają to założenie, dopuszczając pewien poziom bezpośredniego (kontrolowanego) dostępu do „wnętrznośc" obiektu. Ograniczają w ten sposób poziom abstrakcji.
- **Polimorfizm** (ang. *polymorphism*) Polimorfizm pozwala traktować klasy pochodne w taki sam sposób jak klasy rodzicielskie.

3.2.6 Plik

Omawiane do tej pory struktury danych charakteryzowały się tym, że była znana ich wielkość[13]. Zatem stosunkowo łatwo było je przechowywać w pamięci. Niestety istnieje całą rzesza dynamicznych struktur danych, takich jak kolejki, drzewa, grafy itp., które nie posiadają tej cechy. Przyjrzyjmy się jednej z takich struktur danych, a mianowicie **plikowi**. Plik reprezentuje strukturę danych, która odpowiada koncepcji ciągu. Ciąg jest sekwencyjną strukturą danych, gdzie elementy są ustawione jeden po drugim. Elementami ciągu mogą być rekordy, tablice i inne typy złożone, przy czym ich liczba jest nieograniczona, zatem zawsze można dopisać następny element do ciągu[14]. Podstawowe operacje na ciągach to: pobranie elementu pierwszego, dodanie elementu na końcu, połączenie dwu ciągów w jeden ciąg. Fizycznie ciąg jest reprezentowany poprzez **plik sekwencyjny**, gdyż to pozwala odwzorować charakter ciągu i operacji na nim wykonywanych.

Definicja typu plikowego ma postać:

```
T:file;
```

Dla pliku zdefiniowane są podstawowe operacje, takie jak: **tworzenie pliku**, **otwarcie pliku**, **dopisanie elementu na końcu pliku**, **odczytanie następnego elementu pliku**.

3.2.7 Kolejka

Kolejka jest pierwszą z całego szeregu dynamicznych struktur danych, które pokrótce omówimy. Termin *dynamiczny* oznacza w tym kontekście

[13] Oczywiście, o ile znana była wielkość ich elementów.
[14] O ile oczywiście wystarczy nośnika danych.

tyle, co *o nieokreślonym (dowolnym) rozmiarze*. Nie będziemy wnikać jak to się dzieje, że struktura ta ma nieokreślny rozmiar a tym samym może przechowywać, teoretycznie, nieskończoną ilość elementów. Zapamiętajmy jednak, że jest to jedna z jej istotnych cech. Elementy w kolejce zwykle mogą być jednego typu. Każdy element ma ściśle wyznaczone miejsce względem pozostałych elementów. Elementy te tworzą sekwencję przypominającą rzeczywistą kolejkę, np. ludzi stojących po bilety lub samochodów przejeżdżających przez bardzo wąski tunel. Tak więc przede wszystkim sekwencja ta ma zawsze dwa końce: **początek** i **koniec** nazywane także **głową** i **ogonem**. Nowy element dołączany jest do tej sekwencji zawsze na końcu (stając się ogonem) zaś pierwszy element obsłużony, to pierwszy element tej sekwencji (czyli jego głowa). Elementy obsłużone usuwane są z kolejki. Zwykle na kolejce możemy wykonać dwie elementarne operacje: dodać coś do kolejki (zawsze na koniec) lub coś z niej usunąć (zawsze z początku).

Kolejkę tego typu nazywa się **kolejką FIFO** (ang. *First In First Out*). Występują także jej modyfikacje, np. **kolejka priorytetowa** gdzie kolejność elementów w sekwencji zależy od liczby określającej ich ważność (priorytet) czy **kolejka cykliczna** gdzie element pierwszy ma swojego poprzednika (jest nim ogon kolejki), a element ostatni ma swojego następcę (jest nim głowa kolejki). Uogólnienie pojęcia kolejki nazywa się **listą**.

3.2.8 Stos

Stos to kolejna dynamiczna struktura danych. Podobna do kolejki, ale w odróżnieniu od niej dodawanie elementów i ich usuwanie zawsze dotyczy tego samego końca. Stąd obsługiwanie elementów stosu dobywa się na zasadzie **LIFO** (ang. *Las In First Out*). Elementy stosu można wyobrazić sobie jako samochody wjeżdżające do wąskiego i ślepego tunel. Ten który wjedzie jako ostatni będzie miał najbliżej do wyjści.

3.2.9 Kolejka a stos – przykładowe zastosowanie

Pokażemy teraz przykład w którym zmiana tylko i wyłącznie struktury danych, przy pozostawieniu niezmienionego algorytmu, powoduje całkiem odmienne zachowanie.

1. Przyjmij jako wykorzystywaną strukturę kolejkę.
2. Dodaj do kolejki wskazany element.
3. Dopóki struktura nie jest pusta, wykonuj

(a) Pobierz element ze struktury.

(b) Zaznacz pobrany element jako `odwiedzony`.

(c) Znajdź najbliższych sąsiadów pobranego elementu.

(d) Dodaj do struktury tych sąsiadów, którzy nie zostali odwiedzeni i którzy już tam się nie znajdują.

3.2.10 Drzewo

Ponownie dynamiczna struktura danych i ponownie jej nazwa odzwierciedla jej budowę. Tak jak w prawdziwym drzewie tak i tutaj mamy korzeń, węzły, gałęzie i liście. Jedyne co jest inne to to, że drzewo to rośnie do dołu.

Najważniejszym węzłem drzewa jest jego **korzeń**. Od niego odchodzą (czyli z nim są połączone) inne węzły – nazwijmy je węzłami pierwszego poziomu. Od węzłów pierwszego poziomu odchodzą kolejne węzły tworząc węzły poziomu drugiego. I tak dalej. Węzły od których nie odchodzą żadne węzły nazywane są **liśćmi**. Ścieżka prowadząca od korzenia do liścia nazywana jest **gałęzią**. Czasami mówiąc o drzewie stosujemy terminologię pokrewieństwa rodzinnego. Oto węzeł poziomu n nazywamy **rodzicem** odchodzącego od niego węzła poziomu $n+1$ a węzeł poziomu $n+1$ odchodzace od węza poziomu n nazywa się jego **dzieckiem**.

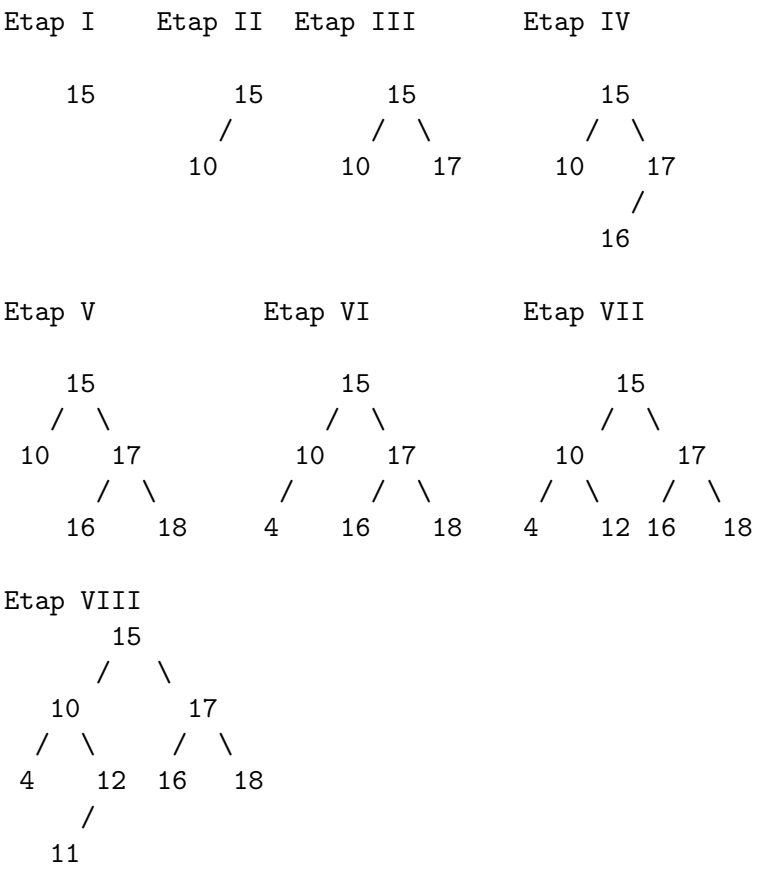
Chcąc być bardziej precyzyjnym, można powiedzieć, że drzewo jest grafem spójnym pozbawionym cykli.

Od definicji konkretnego drzewa zależy m.in. czy elementy przechowujemy w każdym węźle czy tylko w liściach, ile dzieci może mieć każdy węzeł oraz zasady według których wstawiamy elementy do tej struktury danych. Jako przykład wykorzystania drzewa pokażemy jak można przy jego pomocy łatwo sortować liczby.

Przyjmujemy następujące założenia:

- w każdym węźle przechowujemy jedną wartość;
- każdy rodzic może mieć co najwyżej dwójkę dzieci;
- po lewej stronie każdego węzła przechowujemy wartości mniejsze zaś po prawej większe od wartości w węźle;
- elementy drzewa wypisujemy w kolejności: najpierw wszystkie elementy z lewej strony danego węzła, potem element z węzła a na końcu elementy z prawej strony węzła.

Niech ciąg jaki mamy posortować tworzą liczby: 15,10,17,16,18,4,12,11.



Teraz pozostaje nam tylko wypisać elementy według przyjętej zasady.

3.3 Metody opisu algorytmów

Jak wspomniano wcześniej, algorytm jest pewnego rodzaju sformalizowanym zapisem pewnej procedury. W celu uniknięcia nieporozumień co do znaczenia pewnych słów, które mogłyby spowodować złą interpretację algorytmu, wprowadza się umowne sposoby zapisu algorytmów. I tak zasadniczo wyróżnia się zapis za pomocą: **języka naturalnego**, **schematu blokowego**, **metajęzyka programowania**. Zanim omówimy pokrótce każdy z tych sposobów, wymieńmy najważniejsze ich cechy.

- język naturalny
  - teoretycznie łatwy do napisania (wypunktowanie czynności)

  - często mała precyzja[15]
  - kłopoty implementacyjne

- schemat blokowy
  - duża przejrzystość i czytelność
  - odzwierciedla strukturę algorytmu wyraźnie zaznaczając występowanie rozgałęzień (punktów decyzyjnych)
  - kłopoty implementacyjne

- pseudojęzyk
  - ułatwia implementację
  - mniejsza przejrzystość

3.3.1 Język naturalny

Zapis algorytmu w postaci języka naturalnego jest niczym innym jak usystematyzowaniem i wypunktowaniem czynności jakie należy wykonać. Przyjrzyjmy się dobrze znanemu przykładowi — algorytmowi Euklidesa.

1. Weźmy dwie liczby całkowite dodatnie: a i b.

2. Jeśli $b = 0$ idź do **3.**, w przeciwnym razie wykonaj:

  2.1. Jeśli $a > b$ to $a := a - b$.

  2.2. W przeciwnym razie $b := b - a$.

  2.3. Przejdź do **2.**

3. a jest szukanym największym dzielnikiem.

4. Koniec

[15] Będąca wynikiem niejednoznaczności języka naturalnego i brakiem doświadczenia — ten sposób zapisu wybierają najczęściej osoby nie mające na co dzień do czynienia z algorytmami.

Rys. 3.1. Przykładowe elementy graficzne stosowane w schematach blokowych

3.3.2 Schemat blokowy

Schematy blokowe są uniwersalną i dającą odpowiedni stopień ogólności formą zapisu algorytmu. Cechuje je przy tym duża przejrzystość i czytelność. Schematy te odzwierciedlają dobrze strukturę algorytmu, w tym takie elementy, jak **rozgałęzienia** czy **punkty decyzyjne**. Punkt decyzyjny to miejsce, w którym podejmowana jest decyzja, którą z gałęzi (dróg) algorytmu pójść. Decyzja ta jest podejmowana na podstawie pewnego warunku lub ich zespołu. Zatem punkt decyzyjny jest miejscem rozgałęzienia algorytmu. Rysunek 3.1 zawiera podstawowe elementy schematu blokowego, których znaczenie jest następujące:

stan – określa zwykle moment startu i końca;

zapis/odczyt – wskazuje miejsca, w których odbywa się zapis danych (bądź ich odczyt) na nośniki informacji;

instrukcje – blok instrukcji, które mają być wykonane;

decyzja – wyliczenie warunku logicznego znajdującego się wewnątrz symbolu i podjęcie na jego podstawie decyzji;

łącznik – połączenie z inną częścią schematu blokowego, np. gdy nie mieści się on na jednej stronie;

dokument – jakaś forma dokumentu (zwykle „papierowa"), która jest generowana przez system (np. zestawienie, raport, lista, podsumowanie).

Schemat blokowy, zwany również **siecią działań**, tworzony jest według pewnych reguł:

1) składa się on z bloków połączonych zorientowanymi liniami;
2) bloki obrazują ciąg operacji;
3) zawsze wykonywane są albo wszystkie instrukcje w bloku, albo żadna;
4) dalsze operacje nie zależą od poprzednich wariantów, chyba że zależności te zostały przekazane za pomocą danych;
5) kolejność wykonania operacji jest ściśle określona przez zorientowane linie łączące poszczególne bloki;
6) do każdego bloku może prowadzić co najwyżej jedna linia;
7) linie mogą się łączyć, a punkty takiego połączenia określane są jako **punkty zbiegu**.

Na rys. 3.2 znajduje się fragment algorytmu do obliczania pierwiastków trójmianu kwadratowego. Łączniki A i B informują, że dalsze części algorytmu znajdują się na osobnym schemacie lub schematach. Wtedy tamte schematy muszą zaczynać się od odpowiedniego łącznika.

3.3.3 Schemat zapisu algorytmu za pomocą pseudojęzyka

Inną metodą przedstawienia algorytmu jest użycie zapisu w pewnym **pseudojęzyku programowania**. Zaletą tego podejścia jest bardzo łatwa późniejsza implementacja za pomocą już konkretnie wybranego, istniejącego języka programowania[16]. Wadę stanowi natomiast mniejsza przejrzystość. Dla osób nieprzywykłych do analizowania kodu źródłowego programów dużo czytelniejsze jest przedstawienie w postaci schematu blokowego. Poniżej przedstawiamy typowe operacje użyte do opisu algorytmu za pomocą omawianej metody, przy czym dla porównania i gdzie jest to sensowne przedstawiamy również ich postać blokową.

Opiszemy teraz podstawowe instrukcje służące do opisu algorytmów prezentując również ich graficzne notacje.

instrukcja podstawienia

```
x:=y;
wiek:=12.6;
imie:="Piotr";
```

W wyniku wykonania powyższych instrukcji zmienna `x` przyjmie wartość zmiennej `y`, zmienn `wiek` przyjmie wartość 12.6 a zmienna `imie` przyjmie wartość będącą napisem „Piotr”.

[16]Dzieje się tak, ponieważ większość konstrukcji podstawowych, takich jak warunek czy pętla, ma podobną postać w różnych językach.

Rys. 3.2. Schemat blokowy – fragment algorytmu znajdowania pierwiastków trójmianu kwadratowego

blok instrukcji

```
begin
  instrukcje zawarte
  w bloku
end
```

Blok jest sposobem na grupowanie instrukcji. W pewnym sensie odpowiada jemu blok instrukcji ze schematu blokowego, choć tutaj znaczenie bloku jest znacznie szersze. W jego wnętrzu może wystąpić bowiem dowolna instrukcja (np. instrukcja warunkowa). Konsekwencją stosowania bloku jest tak zwany **zasięg nazw**. Otórz zmienna utworzona w bloku istnieje tylko do końca tego bloku. Oznacza to, że w poniższym przykładzie zmienna `x` może być używana w całym bloku pierwszym (także wewnątrz bloku drugiego) natomiast zmienna `y` widoczna (dostępna) jest tylko w bloku drugim.

```
#blok pierwszy
begin
  x:=1;
  #blok drugi
  begin
    y:=2;
    Wypisz(x);
    Wypisz(y);
  end
  #nie można napisać:
  #Wypisz(y);
  Wypisz(x);
begin
```

W powyższym kodzie możemy także zauważyć występowanie symbolu `#` który oznacza komentarz a więc wszystko to co w linii jego wystąpienia znajduje się za nim jest pomijane podczas wykonywania kodu programu.

instrukcja warunkowa

```
if (WARUNEK) then
begin
  instrukcje realizowane, gdy warunek jest spełniony
  (gdy warunek jest prawdziwy)
end
else
begin
  instrukcje realizowane, gdy warunek jest niespełniony
  (gdy warunek jest fałszywy)
end
```

Podstawowa instrukcja warunkowa: wpierw wyliczany jest warunek w nawiasie[17], następnie, jeśli jest on prawdziwy, wykonywane są instrukcje w pierwszym bloku, w przeciwnym razie w drugim bloku, po słowie `else`. Często instrukcja warunkowa używana jest w uproszczonej wersji, bez bloku „w przeciwnym razie". Niektóre języki wielokrotnie zagnieżdżone warunki typu

```
if (WARUNEK) then
```

[17] „Wyliczany" oznacza w tym miejscu tyle, co „wyznaczana jest" jego wartość logiczna.

```
begin
  blok 1
end
else
begin
  if (WARUNEK) then
  begin
    blok 2
  end
  else
  begin
    if (WARUNEK) then
    begin
      blok 3
    end
    else
    begin
      blok 4
    end
  end
end
```

pozwalają zapisać jako

```
if (WARUNEK) then
begin
  blok 1
end
else  if (WARUNEK) then
begin
  blok 2
end
else if (WARUNEK) then
begin
  blok 3
end
else
begin
  blok 4
end
```

Ilustracja graficzna instrukcji warunkowej znajduje się na rys. 3.3. Przykłady warunków:

```
x=7
x>12
x>12 and y<3
x=5 and (y=1 or z=2)
```

Rys. 3.3. Instrukcja warunkowa `if` zapisana w postaci schematu blokowego: a) warunek `if (W) instrukcje`, b) `if (W) instrukcje1 else instrukcje2`

instrukcje pętli do–while i while

```
do
begin
  ... ciąg instrukcji powtarzanych
  dopóki warunek jest spełniony ...
end
while (warunek);

while (warunek)
begin
  ... ciąg instrukcji powtarzanych
  dopóki warunek jest spełniony ...
end
```

Są to dwie pętle, umożliwiające wykonywanie cykliczne instrukcji, które znajdują się w ich wnętrzu. Przy czym instrukcje te są wykonywane tak długo, jak długo `warunek` ma wartość logiczną `prawda`. Nie można więc okeślić ile razy pętla się wykona, ale za to wiemy jaki warunek powoduje, że się ona zakończy. Różnica pomiędzy tymi dwoma pętlami sprowadza się do kolejności wykonania instrukcji wewnętrznych i sprawdzenia warunku. Pierwsza z nich najpierw wykonuje instrukcje, a następnie sprawdza warunek, druga zaś odwrotnie. Ilustracja graficzna omawianych pętli znajduje się na rys. 3.4.

Rys. 3.4. Pętla `while` i `do-while` zapisana w postaci schematu blokowego

instrukcja pętli for

```
for i:=1 to 10 step 1 do
begin
  ... instrukcje powtarzane 10 razy ...
end
```

Pętla ta wykonuje cyklicznie instrukcje zawarte pomiędzy słowani `begin` i `end`. W przykładzie wykonanie zaczyna się od przypisania zmiennej `i` wartości `1` – początek (P), po czym następuje sprawdzenie, czy `i` nie przekracza wartości granicznej `10` – warunek końca (K). Jeśli warunek ten jest fałszywy, wykonane zostają instrukcje, następnie zostaje zmieniona wartość zmiennej `i` o wartość występującą po słowie `step` (S) i na koniec następuje przejście do sprawdzania warunku (K).

A zatem, w odróżnieniu od na przykład pętli *while*, w przypadku pętli *for* zawsze wiemy ile razy się ona wykona. Można również pomijać jawne wskazanie wartości kroku. W takim przypadku przyjmuje się, że jest on równy `1`. Ilustracja graficzna z odpowiednim oznaczeniem literowym znajduje się na rys. 3.5.

Rys. 3.5. Pętla `for` zapisana w postaci schematu blokowego

funkcje **Funkcją** nazwiemy zbiór instrukcji, który posiada nazwę, wtedy nazwa ta stanowi jednocześnie nazwę tej funkcji. Do funkcji można przekazać dane poprzez jej argumenty, można też żądać od niej zwrotu policzonych wartości[18]. Funkcję postrzegać możemy jako zamknięte pudełko, które po dostarczeniu wymaganych argumentów może wykonać dla nas jakieś ściśle określone zadanie. Często traktowana jest jako zamknięta i niezależna całość, do której jeśli ktoś nie chce nie musi zaglądać i o której nie musi wiedzieć jak działa. Wywołanie funkcji polega na podaniu jej nazwy wraz z wymaganymi dla jej działania argumentami umieszczonymi w nawiasie:

```
NazwaFunkcji(argumenty);
x:=Funkcja(arg1,arg2,arg3);
```

[18]To, jakie argumenty funkcja przyjmuje i jaką wartość zwraca, określa jej twórca.

Definicja funkcji ma natomiast postać:

```
function NazwaFunkcji(argumenty)
begin
  instrukcje
  return zwracanaWartosc;
end
```

Występująca pomiędzy słowami `begin` oraz `end` (a więc w tak zwanym ciele funkcji) instrukcja `return` ma specjalne znaczenie. Po jej napotkaniu kończy się natychmiast działanie funkcji i zwracana jest ewentualnie[19] wartość wymieniona za słowem `return`.

instrukcje nieformalne

Zdarza się również, że w schematach blokowych wstawia się nieformalne instrukcje wyrażone w języku naturalnym. Można spotkać taki oto zapis:

```
if (plik=OTWARTY) then
begin
  Zapisanie danych do pliku
end
else
begin
  Komunikat o błędzie
end
```

gdzie oczywiście komunikaty „`Zapisanie danych do pliku`" i „`Komunikat o błędzie`" są tylko informacjami o tym, co w tym miejscu ma wystąpić, a czego z jakiś przyczyn nie chcemy w danej chwili wyspecyfikowywać. Często są to po prostu odwołania do innych procedur, które są dobrze znane i opisane gdzie indziej.

Dla porównania pokażemy teraz sposób zapisu algorytmu Euklidesa przy użyciu pseudokodu.

```
function Euklides(a,b)
begin
   while(b!=0)
   begin
      if(a>b) then
```

[19] Ewentualnie, gdyż można pominąc wyrażenie `zwracanaWartość` — w takim przypadku funkcja nic nie zwraca.

```
      begin
         a:=a-b;
      end
      else
      begin
        b:=b-a;
      end
   end
   return a;
end
```

W przedstawionym kodzie proszę zwrócić uwagę na stosowane wcięcia. Na ogół nie mają one charakteru formalnego i mogą być stosowane dowolnie, ale zwykle ułatwiają czytanie pseudokodu. Stąd dobrą praktyką programistyczną jest wypracownie sobie pewnego sposobu stosowania wcięć i konsekwentnego ich stosowania. Dal porównania pokazujemy też kod zapisany bez użycia wcięc — funkcjonalnie wciąż jest to en sam program choć o znacznie zmniejszonej czytelności.

```
SPOSÓB 1                        SPOSÓB 2
=============================================================
function Euklides(a,b)          function Euklides(a,b)
begin                           begin while(b!=0) begin
while(b!=0)                     if(a>b) then begin a:=a-b; end
begin                           else begin b:=b-a; end end
if(a>b) then                    return a; end
begin
a:=a-b;
end
else
begin
b:=b-a;
end
end
return a;
end
```

3.4 Podstawowe algorytmy

Pokażemy teraz kilka algorytmów zapisanych bądź w pseudojęzyku, bądź gdzie to będzie zbyt skomplikowane, ograniczymy się jedynie do nieformalnego zarysowania ich idei. Algorytmy te reprezentują pewne charakterystyczne typy występujące w algorytmice. Z konieczności ograniczymy się do kilku podstawowych grup.

3.4.1 Algorytmy obliczeniowe

Jest to cała gama algorytmów służących do wyliczania pewnych wartości bądź rozwiązywania problemów numerycznych. Przykładami takich algorytmów mogą być NWD, NWW czy obliczanie pierwiastków trójmianu kwadratowego. W przykładzie 3.1 użyto procedury `wypisz`, której działanie, polegające na wyświetleniu podanego w cudzysłowie napisu, przyjmujemy intuicyjnie.

Przykład 3.1. Algorytm obliczania rzeczywistych pierwiastków równania kwadratowego.

```
if (A=0) then
begin
  wypisz("To nie jest równanie kwadratowe");
end
else
begin
  D:=B^2-4*A*C;
  if (D<0) then
  begin
    wypisz("Nie ma rzeczywistych rozwiązań");
  end
  else
  begin
    if (D>0) then
    begin
      R1:=(-B-Sqrt(D))/(2*A);
      R2:=(-B+Sqrt(D))/(2*A);
    end
    else
```

```
    begin
      R1:=-B/(2*A);
      R2:=R1;
    end
  end
  wypisz("Pierwiastkami są:", R1, R2);
end
```

3.4.2 Algorytmy sortowania

Bardzo często czynnością wykonywaną przez każdego z nas, często w sposób nieświadomy, jest procedura **sortowania**. Mówiąc inaczej jest to algorytm porządkowania czy też układania pewnego zbioru elementów według zadanego klucza. Bardzo istotne jest tu wskazanie klucza, który będzie wyróżnikiem, na którym będzie dokonywana operacja porównania. Zauważmy, że jeśli mamy zbiór osób, i teraz ustawimy je rosnąco względem wzrostu, to nie jest to równoznaczne z porządkiem względem wagi, a już na pewno nie względem długości włosów.

Jako przykład prostego algorytmu sortowania opiszemy **sortowanie przez wstawianie**, zwane również sortowaniem przez proste wstawianie. Spójrzmy wpierw na rys. 3.6 i znajdujące się na nim, w linii zatytułowanej START, liczby całkowite w losowej kolejności.

Rys. 3.6. Proces sortowania przez wstawianie

Naszym zadaniem jest poukładanie tych ośmiu liczb, tj, 23, 54, 13, 21,

63, 15, 10 i 61 rosnąco. W tym celu bierzemy pod uwagę wpierw element drugi (54)i zapamiętujemy go w zmiennej pomocniczej (krok 1). Dla tak wyróżnionego elementu sprawdzamy, czy nie można go wstawić gdzieś przed nim, tj. w tym konkretnym przypadku, czy nie jest on mniejszy od pierwszego elementu (23) (krok 2). Zauważamy, że nie jest, zatem pozostawiamy go na miejscu i przechodzimy do następnego – trzeciego elementu (krok 3). Dla niego sprawdzamy wpierw, czy nie możemy go wstawić na drugie miejsce. Możemy, ale najpierw musimy zrobić na niego miejsce, czyli przepisać element drugi na pozycję trzecią (krok 4). Teoretycznie można by teraz wpisać na pozycję drugą ów wyróżniony element. Sprawdzamy jednak dalej, czy przypadkiem wyróżniony element nie jest jeszcze mniejszy od elementu pierwszego. Ponieważ okazuje się mniejszy, zatem przepisujemy element pierwszy na pozycję drugą (zwalniajac tym samym pozycję pierwszą) (krpk 5). Sprawdzanie takie wykonujemy tak długo, aż natrafimy na element mniejszy od elementu wyróżnionego lub rozpatrzymy wszystkie poprzedzające go elementy. W takim przypadku element wyróżniony wstawiamy na ostatnie zwolnione miejsce (krok 6). Całą procedurę powtarzamy kolejno dla pozostałych elementów. Przebieg powyższego procesu przedstawia przykład 3.2. Dla ułatwienia przyjmujemy, że zmienne przechowywane są w tablicy, a rolę zmiennej tymczasowej pełni `a[0]`

Przykład 3.2. Przebieg procesu sortowania

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
	--	23	54	13	21	63	15	10	61
krok 1	54	23	54	13	21	63	15	10	61
krok 2	54	23	54	13	21	63	15	10	61
krok 3	13	23	54	13	21	63	15	10	61
krok 4	13	23	54	54	21	63	15	10	61
krok 5	13	23	23	54	21	63	15	10	61
krok 6	13	13	23	54	21	63	15	10	61
	...								

Spróbujmy zapisać opisany algorytm za pomocą pseudojęzyka. Przy pierwszej próbie zapisania algorytm ten będzie miał następującą postać:

```
for i:=2 to n do
begin
  x:=a[i];
  "wstaw x w odpowiednie miejsce w ciągu a[1]...a[i-1]";
end
```

Teraz sprecyzujemy pseudokod procesu wyszukiwania miejsca wstawienia elementu `x` oraz uzupełnimy powyższy algorytm o ten fragment. W efekcie otrzymamy prostą implementację algorytmu sortowania przez wstawianie zaprezentowaną w przykładzie 3.3. Zauważmy jednocześnie, że algorytm ten działa niejako jak procedura układania książek na półce, wyjmujemy rozważaną książkę, następnie przesuwamy w prawo książki od niej większe, a następnie wsadzamy ją w miejsce zwolnione i dla niej przeznaczone. Teraz, by można było tę książkę wyjąć, należy mieć wolną rękę, która ją przytrzyma. Podobnie w algorytmie: musimy mieć gdzie przechować liczbę, by móc przesunąć na jej miejsce inną. Stąd w przykładzie 3.3 występuje przypisanie `a[0]:=a[i]`, które właśnie zapamiętuje w zerowej komórce tablicy, traktowanej jako zmienna pomocnicza, element, który w danej chwili rozważamy.

Przykład 3.3. Algorytm sortowania przez wstawianie.

```
for i:=2 to n do
begin
  a[0]:=a[i];
  j:=i-1;
  while (a[0] < a[j])
  begin
    a[j+1]:=a[j];
    j:=j-1;
  end
  a[j+1]:=a[0];
end
```

3.4.3 Algorytmy wyszukujące

Drugim, często spotykanym zagadnieniem jest poszukiwanie informacji o zadanym kluczu, czyli mamy np. zbiór osób, które posiadają jakąś cechę (klucz), niech to będzie wzrost. Następnie szukamy osoby o wzroście 178 cm. Możemy spotkać się z dwoma przypadkami. Pierwszy to taki, gdy osoby te są w dowolnym porządku, czyli niepoukładane, oraz drugi, gdy mamy grupę osób ustawioną względem klucza „wzrost" – jak na wojskowej paradzie. Jeśli będziemy rozpatrywali pierwszy przypadek, to nie pozostaje nam nic innego, jak brać po jednym z elementów i sprawdzać, czy pasuje do naszego wzorca. Taki sposób nosi też nazwę **wyszukiwania liniowego**. Jednak w drugim

Rys. 3.7. Proces przeszukiwania połówkowego

przypadku możemy poradzić sobie lepiej, tzn. efektywnej, jeśli zastosujemy **wyszukiwanie połówkowe** lub **binarne**.

Wyjaśnimy ten sposób realizacji procedury szukania na przykładzie. Spójrzmy na rys. 3.7, gdzie w pierwszym wierszu są przedstawione liczby całkowite posortowane od najmniejszej do największej. Będziemy poszukiwali wartości `15`. Jeśli spróbujemy jej szukać tak, jak w zbiorze nieuporządkowanym i zaczniemy naszą procedurę od pierwszego elementu, to znajdziemy tę wartość w czwartym kroku.

Możemy jednak poszukiwać inaczej. Wpierw wybieramy element środkowy w tym ciągu[20]. Porównujemy wartość tego elementu z wartością szukaną. Jeśli jest to ta wartość to zwracamy numer tego elementu. W przeciwnym razie wiemy, że poszukiwany element jest bądź na prawo, bądź na lewo od środka[21]. I dalej rozważamy tylko ten fragment ciągu, w którym potencjalnie znajduje się nasz element[22]. Cały proces powtarzamy dla okrojonego zbioru. Jeśli znajdziemy element, zwracamy jego pozycję. Jeśli dojdziemy do zbioru pustego, zwracamy informację, że nie znaleźliśmy tego elementu. Proces ten jest właśnie zilustrowany na rys. 3.7.

Zauważmy, że element poszukiwany (`15`) został znaleziony w trzech krokach, zatem o jeden krok szybciej niż w przypadku przeszukiwania liniowego. Możemy oczywiście uznać, że to niewielka oszczędność czasu. Jednak zauważmy, że poszukiwanie jakiegokolwiek elementu w tym zbiorze będzie zajmowało co najwyżej cztery kroki w przypadku wyszukiwania połówkowego, zaś aż 9 w najgorszym przypadku wyszukiwania liniowego. Oczywiście, gdy mamy „pecha", poszukujemy akurat elementu występującego na

[20] Jeżeli liczba elementów jest parzysta, to umawiamy się, że jest to jeden z elementów leżących przy środku. Ważne jest tylko, by zawsze to robić konsekwentnie.

[21] Zauważmy, że to jest miejsce, w którym wykorzystujemy wiedzę o uporządkowaniu zbioru. Gdyby te elementy były ułożone przypadkowo, nie można byłoby wysunąć takiego wniosku.

[22] Potencjalnie, gdyż zaczynając poszukiwania nie wiemy, czy on w ogóle jest w tym zbiorze.

końcu, ale często to właśnie te najgorsze przypadki bierze się pod uwagę przy analizowaniu sprawności (wydajności) algorytmów. Drugą miarą, jaką się rozważa, jest średnia liczba kroków, jaką trzeba wykonać w przypadku jednego bądź drugiego algorytmu.

3.5 Rekurencja a iteracja

W tym punkcie zasygnalizujemy dwa podstawowe podejścia do realizacji prostych algorytmów: **iteracyjne** i **rekurencyjne**. Zacznijmy od wyjaśnienia pojęcia rekurencji.

Obiekt nazwiemy **rekurencyjnym**, jeśli częściowo składa się z siebie samego lub jego definicja odwołuje się do niego samego. Innym słowem, które jest odpowiednikiem **rekurencji** jest **rekursja**.

Jeden z łatwiejszych do zaobserwowania przykładów rekurencji wziętych z życia polega na ustawieniu naprzeciw siebie dwóch lusterek. Powinniśmy zobaczyć lusterko, a w nim lusterko, a w nim lusterko, a w nim ...

Siła rekursji wyraża się w możliwości definiowania nieskończonego zbioru obiektów za pomocą skończonego wyrażenia. Dokładnie na tej samej zasadzie, nieskończoną liczbę obliczeń można zapisać za pomocą skończonego programu rekurencyjnego. Narzędziem umożliwiającym tworzenie programów rekurencyjnych jest, wspomniana przy okazji omawiania pseudokodu, funkcja (można też mówić o procedurze czy podprogramie). Jeśli jakaś funkcja zawiera odwołanie do siebie samej, to nazwiemy ją funkcją rekurencyjną.

Podobnie jak pętle, które pozwalają nam iterować (czyli powtarzać wielokrotnie zestaw instrukcji), funkcje rekurencyjne dopuszczają możliwość wykonywania powtórzeń, z tym wszakże wyjątkiem, że ich ilość będzie nieskończona[23]. Wiąże się z tym konieczność rozwiązania problemu stopu. Zasadniczym wymaganiem w takiej sytuacji jest uzależnienie rekurencyjnego wywołania funkcji od warunku, który w pewnym momencie przestaje być spełniony, co w konsekwencji powoduje zatrzymanie procesu rekurencji. W celu wykazania, że proces wywołań rekurencyjnych się skończy, należy pokazać, że osiągnięty będzie warunek stopu.

Algorytmy rekurencyjne są szczególnie odpowiednie wtedy, gdy rozważany problem lub przetwarzane dane są zdefiniowane w sposób rekurencyjny. Dla przykładu rozważmy dwa sposoby realizacji funkcji obliczającej silnię. Pierwszy z nich, pokazany w przykładzie 3.4, realizuje silnię za pomocą pętli, zatem w sposób iteracyjny.

[23] Oczywiście ta nieskończona liczba operacji jest wyłącznie teoretyczna, w praktyce przecież realizacja wykonania funkcji odbywa się na komputerze, który nie jest wieczny.

Przykład 3.4. Funkcja obliczająca silnię – podejście iteracyjne.

```
function SilniaI(n)
begin
  i:=0;
  s:=1;
  while (i<n) do
  begin
    i:=i+1;
    s:=s*i
  end
  return s;
end
```

Drugi sposób realizacji silni jest pokazany w przykładzie 3.5 i jest to podejście rekurencyjne.

Przykład 3.5. Funkcja obliczająca silnię – podejście rekurencyjne.

```
function SilniaR(n)
begin
  if (n=0) then
  begin
    return 1;
  end

  return n*SilniaR(n-1);
end
```

Przykład 3.6. Przebieg wywołań rekurencyjnych przy obliczaniu wartości $5!$**.**

```
s_r(5)
.   |
.   5*s_r(4)
.   .     |
.   .     4*s_r(3)
.   .     .     |
.   .     .     3*s_r(2)
.   .     .     .     |
```

```
.   .     .     .     2*s_r(1)
.   .     .     .     .     |
.   .     .     .     .     1*s_r(0)
.   .     .     .     .     .     |
.   .     .     .     .     . <---1
.   .     .     .     .     . |
.   .     .     .     . <---1*1
.   .     .     .     . |
.   .     .     . <---2*1
.   .     .     . |
.   .     . <---3*2
.   .     . |
.   . <---4*6
.   . |
<---5*24
|
120
```

Jednak nie zawsze funkcję, która jest dana wzorem rekurencyjnym powinno się realizować według takiego algorytmu. W istocie istnieją bardziej skomplikowane schematy rekurencyjne, które mogą i powinny być przekształcane do postaci iteracyjnych. Dobry przykład stanowi tutaj zagadnienie obliczania liczb ciągu Fibonacciego. Ciąg Fibonacciego, dla $n > 1$, zdefiniowany jest następująco

$$fib_n := fib_{n-1} + fib_{n-2},$$

natomiast wyrazy 1. i 0. przyjmują wartość 1.

Rekurencyjna implementacja funkcji obliczającej liczby ciągu Fibonacciego jest zaprezentowana w przykładzie 3.7.

Przykład 3.7. Funkcja obliczająca n-tą liczbę ciągu Fibonacciego – podejście rekurencyjne.

```
function FibR(n)
begin
  if (n=0 or n=1) then
  begin
    return 1;
  end
```

```
  return FibR(n-1)+FibR(n-2);
end
```

Podejście rekurencyjne nie sprawdza się w tym przypadku; każde wywołanie powoduje dwa dalsze wywołania, tj. całkowita liczba wywołań rośnie wykładniczo (patrz przykład 3.8, tabela 3.1 oraz rysunek 3.8), co szybko prowadzi do wyczerpania stosu i jednocześnie działa wolno, gdyż funkcja dla tych samych danych jest wyliczana kilkakrotnie[24].

Przykład 3.8. Przebieg wywołań rekurencyjnych przy obliczaniu szóstego wyrazu ciągu Fibonacciego.

```
FibR(5)
     |
     +--FibR(4)
     |       |
     |       +--FibR(3)
     |       |       |
     |       |       +--FibR(2)
     |       |       |       |
     |       |       |       +--FibR(1)
     |       |       |       +--FibR(0)
     |       |       |
     |       |       +--Fib(1)
     |       |
     |       +--FibR(2)
     |               |
     |               +--FibR(1)
     |               +--FibR(0)
     +--FibR(3)
             |
             +--FibR(2)
             |       |
             |       +--FibR(1)
             |       +--FibR(0)
             |
             +--FibR(1)
```

Wyraz	Wywołań	Wyraz	Wywołań	Wyraz	Wywołań
0	1	14	1219	28	514229
1	1	15	1973	29	832040
2	3	16	3193	30	1346269
3	5	17	5167	31	2178309
4	9	18	8361	32	3524578
5	15	19	13529	33	5702887
6	25	20	21891	34	9227465
7	41	21	17711	35	14930352
8	67	22	28657	36	24157817
9	109	23	46368	37	39088169
10	177	24	75025	38	63245986
11	287	25	121393	39	102334155
12	465	26	196418	40	165580141
13	753	27	317811		

Tablica 3.1. Ilość wywołań funkcji podczas obliczania wyrazów ciągu Fibonaciego od 0 do 40.

Rys. 3.8. Zależność ilości wywołań c od czasu t.

W tym przypadku lepiej użyć funkcji iteracyjnej, jak to zaprezentowano w przykładzie 3.9.

Przykład 3.9. Funkcja obliczająca n-tą liczbę ciągu Fibonacciego – podejście iteracyjne.

```
function FibI(n)
begin
  i:=1; // licznik pętli
  tmp :=0 // zmienna tymczasowa (pomocnicza)
  x:=1; // wyraz n-1
  y:=1; // wyraz n-2

  while (i<n)
```

[24] Nawet kilkadziesiąt bądź kilkaset razy.

```
  begin
    tmp:=y; // zapamiętaj wyraz n-2
    y:=y+x; // przesuń wyraz n-2 na kolejną wartość ciągu
    x:=tmp; // przesuń wyraz n-1 na kolejną wartość ciągu
            // czyli na wartość wyrazu n-1 przed jego
            // przesunięciem

    i:=i+1; // aktualizacja licznika
  end

  return x;
end
```

W celu lepszego uzmysłowienia różnic pomiędzy powyższymi funkcjami dokonajmy porównania czasu wykonania odpowiadających im programów. Poniższa tabela zawiera czasy dla algorytmu rekurencyjnego. Wersja iteracyjna oblicza 1000-ny wyraz w czasie niemierzalnym przy wykorzystaniu stopera. Test przeprowadzono na komputerze 486 DX 4 120 MHz, w środowisku MS-DOS[25].

Wyraz	Czas (s)
30	2
31	2,56
32	3,54
33	5,10
34	7,11
35	12,67
36	18,84
37	29,40
38	48,12
39	75,82
40	>90,0

3.6 Analiza złożoności

Często ten sam problem można rozwiązać za pomocą różnych algorytmów, co było pokazane choćby w podrozdziale 3.5. W takim razie należy

[25] Test przeprowadzono specjalnie na starym komputerze, by wyraźniej było widać różnice.

się zastanowić, jakie są różnice pomiędzy tymi algorytmami oraz czy różnice te mają wpływ na wybór konkretnego algorytmu do implementacji. Przy porównywaniu algorytmów zwykle bierze się pod uwagę ich **efektywność** (szybkość działania), zapotrzebowanie na zasoby pamięciowe systemu, wreszcie ich czytelność. Tak naprawdę czytelność jest rzeczą względną i zależy od długości praktyki osoby czytającej, stąd też rzadko jest brana pod uwagę przy porównywaniach. Istotna z praktycznego punktu widzenia różnica wynika z prędkości działania algorytmu i jego zapotrzebowania na pamięć. Przy czym od razu należy zaznaczyć, że różnice między algorytmami są najczęściej bez znaczenia przy przetwarzaniu małej liczby danych, ale rosną razem z nią i to liniowo, logarytmicznie, z kwadratem, wykładniczo itd.

W celu porównania algorytmów pod względem ich prędkości działania, wprowadzono pojęcie **złożoności obliczeniowej**. Dysponując takim pojęciem, posiadamy pewną miarę, która pozwala porównać różne algorytmy, ocenić efekt poprawienia istniejącego algorytmu, czy wreszcie oszacować czas realizacji danego zadania.

Złożoność obliczeniowa określa, ilu zasobów wymaga użycie danego algorytmu lub jak jest ono kosztowne. Koszt ten można oczywiście mierzyć na wiele różnych sposobów zależnych od postawionego zadania. Stąd czas wyrażany jest za pomocą umownych jednostek.

Rozważmy następujący przykład. Mamy funkcję f zmiennej całkowitej n daną następującym wzorem:

$$f(n) = n^2 + 100n + \log_{10} n + 1000.$$

Dla małych wartości n ostatni czynnik jest największy. Jednak wraz ze wzrostem n maleje jego znaczenie na rzecz pozostałych. Ostatecznie, gdy n przekroczy wartość 100 000, jego znaczenie w ogólnym wyniku jest marginalne. Również pozostałe składniki, oprócz n^2, mają coraz mniejszy wkład. To właśnie głównie od pierwszego składnika zależy wartość funkcji i dla dużych n można przyjąć, że funkcja ta jest postaci

$$f(n) = cn^2,$$

gdzie c jest pewną stałą.

W wyniku takiej obserwacji, jak powyższa, wprowadzono pewne notacje mające na celu określenie złożoności obliczeniowej danego algorytmu. Notacji tych jest kilka. Przedstawimy najprostszą z nich – notację „wielkie O”.

Dla danej pary funkcji f i g o dodatnich wartościach rozważmy następujące definicje.

Definicja 3.1. *Notacja „wielkie* O*". Powiemy, że funkcja* $f(n)$ *jest rzędu* $\mathrm{O}(g(n))$*, jeśli istnieją dodatnie liczby* c *i* N *takie, że* $0 \leq f(n) \leq cg(n)$ *dla wszystkich* $n \geq N$*.*

Na przykład wyrażenie

$$n^2 + 100n + \log_{10} n + 1000$$

można zapisać jako

$$n^2 + 100n + \mathrm{O}(\log_{10} n),$$

co oznacza, że część „obcięta" jest zawsze mniejsza niż pewna ustalona stała pomnożona przez $\log_{10} n$.

Własności notacji „wielkie O".

Własność 1. Jeśli $f(n)$ jest $\mathrm{O}(g(n))$ i $g(n)$ jest $\mathrm{O}(h(n))$, to $f(n)$ jest $\mathrm{O}(h(n))$. Inaczej: $\mathrm{O}(\mathrm{O}(g(n)))$ jest $\mathrm{O}(g(n))$.

Własność 2. Jeśli $f(n)$ jest $\mathrm{O}(h(n))$ i $g(n)$ jest $\mathrm{O}(h(n))$, to $f(n) + g(n)$ jest $\mathrm{O}(h(n))$.

3.7 Zadania

1. Napisz algorytm do obliczania x^y, gdzie x, y są liczbami naturalnymi, przy czym dopuszczamy, że y może być 0. Napisz wersję iteracyjną i rekurencyjną.

2. Dla iteracyjnej wersji funkcji określonej w poprzednim zadaniu narysuj schemat blokowy.

3. Mając poniższy schemat blokowy zamień go na zapis w pseudojęzyku programowania, zastanów się co robi ten algorytm (rys. 3.9).

4. Napisz funkcję rekurencyjną, która realizuje algorytm wyszukiwania połówkowego opisany w pkt 3.4.3.

Rys. 3.9. Schemat blokowy pewnego algorytmu

Rozdział 4

Reprezentacja danych w komputerach

4.1 Znaki alfanumeryczne

4.1.1 Przykład – kodowanie FOO

Przyjmujemy następujący sposób kodowania znaków alfanumerycznych, nazywany kodowaniem FOO[1] Znak „spacji" posiada kod 42.

Dodatkowo wprowadzamy następujące sekwencje sterujące:

- `ESC1` (kod 43) służącą do zamiany litery małej występującej zaraz za sekwencją na dużą.
- `ESC2` (kod 44) służącą do uzyskania znaków diakrytycznych. Sekwencja

[1]Termin *foo* jest jednym z elementów kultury hackerskiej, tak jak *kludge* czy *cruft*. Zwykle oznacza nieznaną wartość (zmienną). Więcej informacji znajdziesz w [3], [5] i ogólnie [12].

a	b	c	d	e	f	g	h	i	j	k	l	m	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13
o	p	q	r	s	t	u	v	w	x	y	z	0	1
14	15	16	17	18	19	20	21	22	23	24	25	26	27
2	3	4	5	6	7	8	9	,	.	(	)	-	’
28	29	30	31	32	33	34	35	36	37	38	39	40	41

Tablica 4.1. Tablica kodów dla kodowania FOO.

  - `ESC2 , litera` dodaje „ogonek” do litery,
  - `ESC2 .  litera` dodaje „kropkę” do litery,
  - `ESC2 - litera` dodaje „przekreślenie” do litery,
  - `ESC2 ’ litera` dodaje „kreskę nad” do litery.

- `NL` (kod 45) powodujący przejście do nowego wiersza.

Jak widać w kodzie WDI liczba znaków jest mniejsza niż 64 ale większ niż 32. Stąd wniosek, że musimy używać co najmniej 6 bitów do zapisania kodów.

Spróbujmy zakodować następujące zdanie:

Miała (kiedyś) Zośka 371 kotów a teraz ma 1 szczura - „Mańka”.

Zapiszemy je przy pomocy sekwencji kodów wskazanej w tabeli 4.1.1. Otrzymujemy tym samym ciąg liczb dziesiętnych przedstawiony w tabeli 4.3. Zamieniając teraz każdy kod (każdą liczbę dziesiętną) na ciąg 6-bitowy (bo przyjęliśmy ustalenie, że do zapisu wykorzystujemy 6 bitów) otrzymujemy ciąg zero-jedynkowy przedstawiony[2] w tabeli 4.4. Następnie ciąg ten dzielimy na podciągi 8-bitowe, co zostało przedstawione w tabeli 4.5. Dlaczego tak? Otóż najmniejszą jednostką zapisywania danych na dysk (czy też przechowywania ich w pamięci) jest bajt. Nawet gdy chcemy zapisać tylko jeden bit, to i tak musimy zapisać cały bajt. Skoro więc zawsze zapisujemy cały bajt, to staramy się wykorzystać wszystkie jego bity. Oczywiście możemy zapisywać dane 6-bitowe, ale wówczas każda taka dana zajmuje i tak 8 bitów, co oznacza zmarnowanie 25% przestrzeni dyskowej (pamięci).

W wyniku dokonanego podziału na samym końcu otrzymaliśmy sekwencję `100101`, czyli ciąg zero-jedynkowy krótszy niż 8 bitów. Zwykle w takiej sytuacji dopełnia się otrzymany ciąg tak aby otrzymać 8 bitów. Dopełniając go, np. zerami otrzymujemy ciąg `10010100`. W tym przypadku taki sposób dopełnienia nie będzie problemem, gdyż długość dopełnienia jest mniejsza niż 6 bitów a w naszym kodowaniu poprawny znak musi zajmować dokładnie 6 bitów. Stąd wniosek, że jeśli przy odkodowywaniu otrzymamy ciąg 2 bitowy, to nie jest on poprawnym znakiem i należy go najzwyczajniej zignorować. Może się jednak zdarzyć, że dopełnienie będzie miało długość 6 bitów. I jak w takim przypadku odróżnić dopełnienie od normalnego znaku? Na ogół właściwą część wiadomości kończy się specjalnie do tego celu zarezerwowaną sekwencją znaków, która ma długość 6 bitów, ale nie jest

[2] Dla czytelności co 6 bitów zrobiono przerwę.

M	i	a	ł	a		(	k	i	e	d
43,12	8	0	44,40,11	0	42	38	10	8	4	3
y	ś	)		Z	o	ś	k	a		3
24	44,41,18	39	42	43,25	14	44,41,18	10	0	42	29
7	1		k	o	t	ó	w		a	
33	27	42	10	14	19	44,41,14	22	42	0	42
t	e	r	a	z		m	a		1	
19	4	17	0	25	42	12	0	42	27	42
s	z	c	z	u	r	a		-		,
18	25	2	25	20	17	0	42	40	42	36
,	M	a	ń	k	a	’	’	.		
36	43,12	0	44,41,13	10	0	41	41	37		

Tablica 4.2. Przyporządkowanie kodów do znaków zdania *Miała (kiedyś) Zośka 371 kotów a teraz ma 1 szczura - „Mańka”.* w kodowaniu FOO.

poprawnym znakiem. Dopiero za tą sekwencją umieszcza się dopełnienia. Trzymając się tej zasady, przyjmijmy, że sekwencja kończąca wiadomość to `111111`. Wówczas zakończenie naszego tekstu będzie wyglądało następująco.

- Najpierw będzie ostatnie 6 bitów wiadomości: `100101`.
- Potem sekwencja `111111`.
- Dopełnienie celem uzyskania ilości bitów równej wielokrotności 8. Łącznie dwie pierwsze sekwencje mają długość 12, więc jako dopełnienia należy użyć 4 bitów – niech będą to bity zerowe.

Tak więc więc ostatni wiersz tabeli 4.5 przyjmie postać jak w tabeli 4.6. Na zakończenie wystarczy zapisać uzyskane w ten sposób bajty (czy to na dysk, czy też do pamięci). Bajty te reprezentują ciąg liczb dziesiętnych przedstawionych w tabeli 4.7. Przyglądając się plikowi zawierającemu te bajty, zobaczymy znaki ASCII przyporządkowane właśnie tym kodom.

43,	12,	8,	0,	44,	40,	11,	0,	42,	38,	10,	8,	4,	3,
24,	44,	41,	18,	39,	42,	43,	25,	14,	44,	41,	18,	10,	0,
42,	29,	33,	27,	42,	10,	14,	19,	44,	41,	14,	22,	42,	0,
42,	19,	4,	17,	0,	25,	42,	12,	0,	42,	27,	42,	18,	25,
2,	25,	20,	17,	0,	42,	40,	42,	36,	36,	43,	12,	0,	44,
41,	13,	10,	0,	41,	41,	37							

Tablica 4.3. Ciąg kodów dziesiętnych przyporządkowanych do znaków zdania *Miała (kiedyś) Zośka 371 kotów a teraz ma 1 szczura - „Mańka”.* w kodowaniu FOO.

101011	001100	001000	000000	101100	101000	001011	000000
101010	100110	001010	001000	000100	000011	011000	101100
101001	010010	100111	101010	101011	011001	001110	101100
101001	010010	001010	000000	101010	011101	100001	011011
101010	001010	001110	010011	101100	101001	001110	010110
101010	000000	101010	010011	000100	010001	000000	011001
101010	001100	000000	101010	011011	101010	010010	011001
000010	011001	010100	010001	000000	101010	101000	101010
100100	100101	101011	001100	000000	101100	101001	001101
001010	000000	101001	101001	100101			

Tablica 4.4. Ciąg zero-jedynkowy odpowiadający ciągowi kodów dziesiętnych z tabeli 4.3.

10101100	11000010	00000000	10110010	10000010	11000000
10101010	01100010	10001000	00010000	00110110	00101100
10100101	00101001	11101010	10101101	10010011	10101100
10100101	00100010	10000000	10101001	11011000	01011011
10101000	10100011	10010011	10110010	10010011	10010110
10101000	00001010	10010011	00010001	00010000	00011001
10101000	11000000	00101010	01101110	10100100	10011001
00001001	10010101	00010001	00000010	10101010	00101010
10010010	01011010	11001100	00000010	11001010	01001101
00101000	00001010	01101001	100101		

Tablica 4.5. Ciąg zero-jedynkowy z tabeli 4.4 podzielony na podciągi 8-bitowe.

00101000	00001010	01101001	10010111	11110000

Tablica 4.6. Ostatni wiersz tabeli 4.5 po zakończeniu danych sekwencją 111111 i dopełnieniu sekwencją 0000.

43,	12,	8,	0,	44,	40,	11,	0,	42,	38,	10,	8,	4,	3,
24,	44,	41,	18,	39,	42,	43,	25,	14,	44,	41,	18,	10,	0,
42,	29,	33,	27,	42,	10,	14,	19,	44,	41,	14,	22,	42,	0,
42,	19,	4,	17,	0,	25,	42,	12,	0,	42,	27,	42,	18,	25,
2,	25,	20,	17,	0,	42,	40,	42,	36,	36,	43,	12,	0,	44,
41,	13,	10,	0,	41,	41,	37							

Tablica 4.7. Liczby dziesiętne odpowiadające finalnej postaci zakodowanej informacji (tabela chwilowo jest błędna – czeka na poprawne uzupełnienie).

Bibliografia

[1] Al-Kindi, Cryptgraphy, Codebreaking and Ciphers, http://www.muslimheritage.com/topics/default.cfm?ArticleID=372, dostęp 2009-10-07.

[2] Denning, P.J., *Computer Science: The Discipline*, Encyclopedia of Computer Science, 2000, http://web.archive.org/web/20060525195404/http://www.idi.ntnu.no/emner/dif8916/denning.pdf

[3] Eastlake D. III et al., *Etymology of „Foo"*, Internet Engineering Task Force (2001), http://www.ietf.org/rfc/rfc3092.txt, dostęp 2009-12-18

[4] http://history-computer.com

[5] Fulmański P., *foo, kludge i cruft.*

[6] Hollerith's Electric Sorting and Tabulating Machine, ca. 1895 from the American Memory archives of the Library of Congress, http://memory.loc.gov/cgi-bin/query/r?ammem/mcc:@field(DOCID+@lit(mcc/023)), dostęp 2009-10-08

[7] IBM's ASCC (a.k.a. The Harvard Mark I), IBM http://www-03.ibm.com/ibm/history/exhibits/markI/markI_intro.html

[8] http://www.simonsingh.net/The_Ishango_Bone.html, dostęp 2009-10-07.

[9] http://planetmath.org/encyclopedia/IshangoBone.html, dostęp 2009-10-07.

[10] Knuth, Donald E., *Backus Normal Form vs. Backus Naur Form*, Communications of the ACM 7 (12), (1964): 735–736.

[11] Plofker, Kim, *Mathematics in India. The Mathematics of Egypt, Mesopotamia, China, India, and Islam: A Sourcebook*, Princeton University Press, (2007), ISBN 9780691114859.

[12] Raymond Eric S., *The New Hacker's Dictionary*, (1996).

[13] A. I. Sabra, *Ibn al-Haytham*, Harvard Magazine, September-October 2003, s. 54-55.