Marek Gągolewski
Instytut Badań Systemowych PAN
Wydział Matematyki i Nauk Informacyjnych Politechniki Warszawskiej

Algorytmy i podstawy programowania

2. Podstawy organizacji i działania komputerów. Zmienne w języku C++ i ich typy. Operatory

Materiały dydaktyczne dla studentów matematyki
na Wydziale Matematyki i Nauk Informacyjnych Politechniki Warszawskiej
Ostatnia aktualizacja: 1 października 2016 r.

Spis treści

2.1. Zarys historii informatyki

Historia informatyki nieodłącznie związana jest z historią matematyki, a w szczególności zagadnieniem zapisu liczb i rachunkami. Najważniejszym powodem powstania komputerów było, jak łatwo się domyślić, *wspomaganie wykonywania żmudnych obliczeń*.

Oto wybór najistotniejszych wydarzeń z dziejów tej dziedziny, które pomogą nam rzucić światło na przedmiot naszych zainteresowań.

- Ok. 2400 r. p.n.e. w Babilonii używany jest już kamienny pierwowzór liczydła, na którym rysowano piaskiem (specjaliści umiejący korzystać z tego urządzenia nie byli liczni). Podobne przyrządy w Grecji i Rzymie pojawiły się dopiero ok. V-IV w. p.n.e. Tzw. *abaki*, żłobione w drewnie, pozwalały dokonywać obliczeń w systemie dziesiętnym.
- Ok. V w. p.n.e. indyjski uczony Pãnini sformalizował teoretyczne reguły gramatyki sanskrytu. Można ten fakt uważać za pierwsze badanie teoretyczne w dziedzinie lingwistyki.
- Pierwszy algorytm przypisywany jest Euklidesowi (ok. 400-300 r. p.n.e.). Ten starożytny uczony opisał operacje, których wykonanie krok po kroku pozwala wyznaczyć największy wspólny dzielnik dwóch liczb (por. zestaw zadań nr 1).
- Matematyk arabski Al Kwarizmi (IX w.) określa reguły podstawowych operacji arytmetycznych dla liczb dziesiętnych. Od jego nazwiska pochodzi ponoć pojęcie *algorytmu*.
- W epoce baroku dokonuje się „obliczeniowy przełom". W 1614 r. John Napier, szkocki teolog i matematyk, znalazł zastosowanie logarytmów do wykonywania szybkich operacji mnożenia (zastąpił je dodawaniem). W roku 1622 William Oughtred stworzył *suwak logarytmiczny*, który jeszcze bardziej ułatwił wykonywanie obliczeń.
- W. Schickard (1623 r.) oraz B. Pascal (1645 r.) tworzą pierwsze *mechaniczne sumatory*. Ciekawe, czy korzystał z nich Isaac Newton (1643–1727)?

Mamy bowiem $xy = e^{\log x + \log y}$.

Zadanie

Zastanów się przez chwilę, do czego może być przydatna umiejętność szybkiego wykonywania operacji arytmetycznych. W jakich dziedzinach życia czy nauki może się przydać?

- Jacques Jacquard (ok. 1801 r.) skonstruował krosno tkackie sterowane dziurkowanymi kartami. Było to pierwsze *programowalne* urządzenie w dziejach techniki. Podobny ideowo sposób działania miały *pianole* (poł. XIX w.), czyli automatycznie sterowane pianina.
- Ok. 1837–1839 r. Charles Babbage opisał wymyśloną przez siebie „maszynę analityczną" (parową!), programowalne urządzenie do wykonywania obliczeń. Była to jednak tylko koncepcja teoretyczna: nigdy nie udało się jej zbudować. Pomagająca mu Ada Lovelace może być uznana za *pierwszego programistę*.
- H. Hollerith konstruuje w 1890 r. maszynę zorientowaną na przetwarzanie dużej ilości danych, która wspomaga podsumowywanie wyników spisu powszechnego w USA.
- Niemiecki inżynier w 1918 r. patentuje maszynę szyfrującą Enigma. Na marginesie, jej kod łamie polski matematyk Marian Rejewski w 1932 r., co przyczyni się później do sukcesu aliantów w drugiej Wojnie Światowej.
- W 1936 r. Alan Turing i Alonzo Church definiują formalnie algorytm jako ciąg instrukcji matematycznych. Określają dzięki temu to, *co daje się policzyć*. Kleene stawia później tzw. *hipotezę Churcha-Turinga*.

Pierwszym programistą była kobieta!

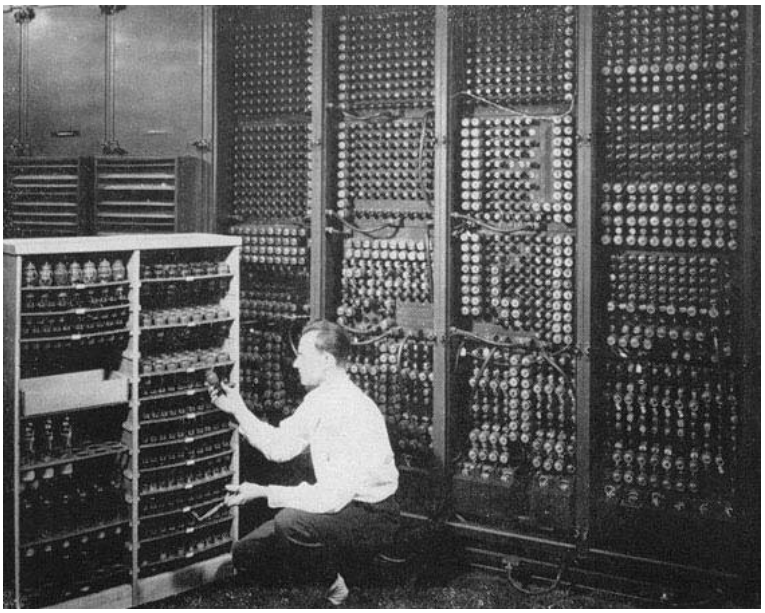
Informacja

Hipoteza Churcha-Turinga mówi o tym, że jeśli dla jakiegoś problemu istnieje efektywny algorytm korzystający z nieograniczonych zasobów, to da się go wykonać na pewnym prostym, abstrakcyjnym modelu komputera:

> *Każdy problem, który może być intuicyjnie uznany za obliczalny, jest rozwiązywalny przez maszynę Turinga.*

Tutaj jednak pojawia się problem: co może być uznane za „intuicyjnie obliczalne" — zagadnienie to do dziś nie jest rozstrzygnięte w sposób satysfakcjonujący.

- Claude E. Shannon w swojej pracy magisterskiej w 1937 r. opisuje możliwość użycia *elektronicznych przełączników* do wykonywania operacji logicznych (implementacja algebry Boole'a). Jego praca teoretyczna staje się podstawą konstrukcji wszystkich współczesnych komputerów elektronicznych.
- W latach 40' XX w. w Wielkiej Brytanii, USA i Niemczech *powstają pierwsze komputery elektroniczne*, np. Z1, Z3, Colossus, Mark I, ENIAC (zob. rys. 2.1), EDVAC, EDSAC. Pobierają bardzo duże ilości energii elektrycznej i potrzebują dużych przestrzeni. Np. ENIAC miał masę ponad 27 ton, zawierał około 18 000 lamp elektronowych i zajmował powierzchnię ok. 140 m^2. Były bardzo wolne (jak na dzisiejsze standardy), np. Z3 wykonywał jedno mnożenie w 3 sekundy. Pierwsze zastosowania były stricte militarne: łamanie szyfrów i obliczanie trajektorii lotów balistycznych.
- W 1947 r. Grace Hopper odkrywa pierwszego robaka (ang. *bug*) komputerowego w komputerze Harvard Mark II — dosłownie!
- W międzyczasie okazuje się, że komputery nie muszą być wykorzystywane tylko w celach przyspieszania obliczeń. Powstają *teoretyczne podstawy informatyki* (np. Turing, von Neumann). Zapoczątkowywane są nowe kierunki badawcze, m.in. sztuczna inteligencja (np. słynny test Turinga sprawdzający „inteligencję" maszyny).
- Pierwszy *kompilator* języka Fortran zostaje stworzony w 1957 r. Kompilator języka LISP powstaje rok później. Wcześniej programowano komputery w tzw. języku maszynowym, bardzo trudnym do zrozumienia dla człowieka.
- Informatyka staje się *dyscypliną akademicką* dopiero w latach sześćdziesiątych XX w. (wytyczne programowe ACM). Pierwszy wydział informatyki powstał na Uniwersytecie Purdue w 1962 r. Pierwszy doktorat w tej dziedzinie został obroniony już w 1965 r.
- W 1969 r. następuje pierwsze *połączenie sieciowe* pomiędzy komputerami w ramach projektu ARPAnet (prekursora Internetu). Początkowo ma ono mieć głównie zastosowanie (znowu!) wojskowe.
- Dalej rozwój następuje bardzo szybko: powstają bardzo ważne algorytmy (np. wyszukiwanie najkrótszych ścieżek w grafie Dijkstry, sortowanie szybkie Hoara itp.), teoria relacyjnych baz danych, okienkowe wielozadaniowe systemy operacyjne, nowe języki programowania, systemy rozproszone i wiele, wiele innych...
- Współcześnie komputer osobisty (o mocy kilka miliardów razy większej niż ENIAC) podłączony do ogólnoświatowej sieci Internet znajdziemy w prawie każdym domu, a elementy informatyki wykładane są już w szkołach podstawowych.



Rys. 2.1. Komputer ENIAC

Zadanie

Zastanów się, jak wyglądał świat 30–50 lat temu. A jak 100 lat temu. W jaki sposób ludzie komunikowali się ze sobą. W jaki sposób spędzali czas wolny.

Tak przyzwyczailiśmy się do wszechobecnych komputerów, że wiele osób nie potrafi sobie wyobrazić bez nich życia.

Wielu najznakomitszych informatyków XX wieku było z wykształcenia matematykami. Informatyka teoretyczna była początkowo poddziałem matematyki stosowanej. Z czasem dopiero stała się niezależną dyscypliną akademicką. Nie oznacza to oczywiście, że obszary badań obu dziedzin są rozłączne.

Zapamiętaj

Wielu matematyków zajmuje się informatyką, a wielu informatyków — matematyką! Bardzo płodnym z punktu widzenia nauki jest obszar na styku obydwu dziedzin. Jeśli planujesz w przyszłości karierę naukowca czy analityka danych może to będzie i Twoja droga. Jeśli nie, tym bardziej dostrzeżesz za kilka lat, że bez komputera po prostu nie będziesz mógł (mogła) wykonywać swojej pracy.

2.1.1. Główne kierunki badań w informatyce współczesnej

Można pokusić się o następującą klasyfikację głównych kierunków badań w informatyce[1]:

1. Matematyczne podstawy informatyki
   (a) Kryptografia (kodowanie informacji niejawnej)

[1] Por. *www.newworldencyclopedia.org/entry/Computer_science*.

   (b) Teoria grafów (np. algorytmy znajdywania najkrótszych ścieżek, algorytmy kolorowania grafów)
   (c) Logika (w tym logika wielowartościowa, logika rozmyta)
   (d) Teoria typów (analiza formalna typów danych, m.in. badane są efekty związane z bezpieczeństwem programów)

2. Teoria obliczeń

   (a) Teoria automatów (analiza abstrakcyjnych maszyn i problemów, które można z ich pomocą rozwiązać)
   (b) Teoria obliczalności (co jest obliczalne?)
   (c) Teoria złożoności obliczeniowej (które problemy są „łatwo" rozwiązywalne?)

3. Algorytmy i struktury danych

   (a) Analiza algorytmów (ze względu na czas działania i potrzebne zasoby)
   (b) Projektowanie algorytmów
   (c) Struktury danych (organizacja informacji)
   (d) Algorytmy genetyczne (znajdywanie rozwiązań przybliżonych problemów optymalizacyjnych)

4. Języki programowania i kompilatory

   (a) Kompilatory (budowa i optymalizacja programów przetwarzających kod w danym języku na kod maszynowy)
   (b) Języki programowania (formalne paradygmaty języków, własności języków)

5. Bazy danych

   (a) Teoria baz danych (m.in. systemy relacyjne, obiektowe)
   (b) Data mining (wydobywanie wiedzy z baz danych)

6. Systemy równoległe i rozproszone

   (a) Systemy równoległe (bada wykonywanie jednoczesnych obliczeń przez wiele procesorów)
   (b) Systemy rozproszone (rozwiązywanie tego samego problemu z użyciem wielu komputerów połączonych w sieć)
   (c) Sieci komputerowe (algorytmy i protokoły zapewniające niezawodny przesył danych pomiędzy komputerami)

7. Architektura komputerów

   (a) Architektura komputerów (projektowanie, organizacja komputerów, także z punktu widzenia elektroniki)
   (b) Systemy operacyjne (systemy zarządzające zasobami komputera i pozwalające uruchamiać inne programy)

8. Inżynieria oprogramowania

   (a) Metody formalne (np. automatyczne testowanie programów)
   (b) Inżynieria (zasady tworzenia dobrych programów, zasady organizacji procesu analizy, projektowania, implementacji i testowania programów)

9. Sztuczna inteligencja

   (a) Sztuczna inteligencja (systemy, które wydają się cechować inteligencją)
   (b) Automatyczne wnioskowanie (imitacja zdolności samodzielnego rozumowania)
   (c) Robotyka (projektowanie i konstrukcja robotów i algorytmów nimi sterujących)

   (d) Uczenie się maszynowe (tworzenie zestawów reguł w oparciu o informacje wejściowe)

10. Grafika komputerowa

   (a) Podstawy grafiki komputerowej (algorytmy generowania i filtrowania obrazów)
   (b) Przetwarzanie obrazów (wydobywanie informacji z obrazów)
   (c) Interakcja człowiek-komputer (zagadnienia tzw. interfejsów służących do komunikacji z komputerem)

Do tej listy można też dopisać wiele dziedzin z tzw. pogranicza matematyki i informatyki, np. bioinformatykę, statystykę obliczeniową, analizę danych, uczenie się maszynowe, zbiory rozmyte itd.

Zapamiętaj

Zauważ, że w trakcie studiów mniej lub bardziej dokładnie zapoznasz się z zagadnieniami z p. 1–5. Pamiętaj o tym.

Na koniec tego paragrafu warto zwrócić uwagę (choć zdania na ten temat są podzielone), że to, co w języku polskim określamy mianem *informatyki* czy też *nauk informacyjnych*, jest przez wielu uznawane za pojęcie szersze od angielskiego *computer science*, czyli nauk o komputerach. Informatyka jest więc ogólnym przedmiotem dociekań dotyczących przetwarzania informacji, w tym również przy użyciu urządzeń elektronicznych.

Informacja

Informatyka jest nauką o komputerach w takim stopniu jak astronomia — nauką o teleskopach. (Dijkstra)

2.2. Organizacja współczesnych komputerów

Jak zdążyliśmy nadmienić, podstawy teoretyczne działania współczesnych komputerów elektronicznych zawarte zostały w pracy C. Shannona (1937 r.). Komputery, na na najniższym możliwym poziomie abstrakcji, mogą być pojmowane jako zestaw odpowiednio sterowanych *przełączników*, tzw. *bitów* (ang. *binary digits*).

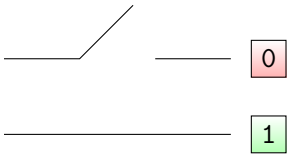
Zapamiętaj

Każdy przełącznik może znajdować się w jednym z dwóch stanów (rys. 2.2):

- prąd nie płynie (co oznaczamy jako `0`),
- prąd płynie (co oznaczamy przez `1`).

Co ważne, informacje, które przetwarza komputer (por. rozdz. 1 — określiliśmy je jako ciągi liczb lub symboli) zawsze reprezentowane są przez ciągi zer i jedynek. Tylko tyle

i aż tyle. W niniejszym rozdziale dowiemy się m.in. w jaki sposób za ich pomocą możemy zapisywać liczby całkowite, rzeczywiste, a kiedy indziej — znaki drukowane (np. litery alfabetu, tzw. kod ASCII).

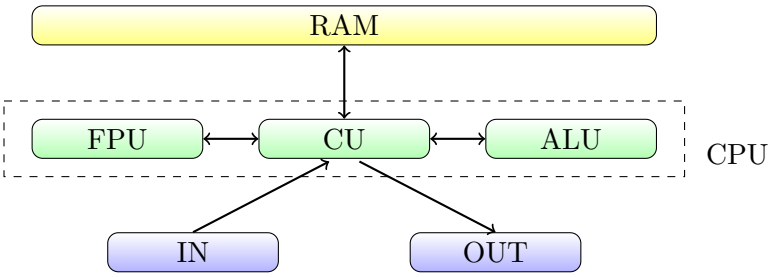


Rys. 2.2. Dwa stany przełączników

Na współczesny komputer osobisty (PC, od ang. *personal computer*) składają się następujące elementy:

1. jednostka obliczeniowa (CPU, od ang. *central processing unit*):
   - jednostki arytmetyczno-logiczne (*ALU*, ang. *arithmetic and logical units*),
   - jednostka do obliczeń zmiennopozycyjnych (*FPU*, ang. *floating point unit*),
   - układy sterujące (*CU*, ang. *control units*),
   - rejestry (akumulatory), pełniące funkcję pamięci podręcznej;
2. pamięć *RAM* (od ang. *random access memory*) — zawiera dane i program,
3. urządzenia wejściowe (ang. *input devices*),
4. urządzenia wyjściowe (ang. *output devices*).

Jest to tzw. *architektura von Neumanna* (por. rys. 2.3). Najważniejszą jej cechą jest to, że w pamięci operacyjnej znajdują się zarówno instrukcje programów, jak i dane (wszystko zapisane za pomocą tylko zer i jedynek). To od kontekstu zależy, jak powinny być one interpretowane przez komputer.



Rys. 2.3. Architektura współczesnych komputerów osobistych

2.3. Zmienne w języku C++ i ich typy

2.3.1. Pojęcie zmiennej

Czytając pseudokody algorytmów z pierwszego zestawu zadań, kilka razy napotykaliśmy instrukcję podobną do następującej.

```
niech x ∈ ℝ;
```

Tym samym mieliśmy na myśli: „niech *od tej pory* `x` będzie *zmienną* ze zbioru liczb rzeczywistych". Formalnie, dokonaliśmy tym samym *deklaracji* zmiennej (ang. *variable declaration*) typu rzeczywistego. Dzięki temu mogliśmy użyć `x` np. do obliczenia wartości pewnego podwyrażenia w skomplikowanym wzorze tak, by rozwiązanie uczynić czytelniejszym.

Zapamiętaj

Zmienna służy do przechowywania w pamięci RAM komputera (lub, w przypadku pseudokodu, czymś, co pamięć komputera reprezentuje) dowolnych wartości z pewnego ustalonego zbioru (dziedziny), zwanego *typem* zmiennej.

2.3.2. Typy liczbowe

Jako że każda zmienna musi mieć określoną dziedzinę, omówmy wpierw typy zmiennych liczbowych dostępne w języku C++. Będą to:

1. typy całkowite, reprezentujące odpowiednie podzbiory $\mathbb{Z}$,
2. typy zmiennoprzecinkowe, reprezentujące pewne podzbiory $\mathbb{R}$ oraz
3. typ logiczny.

2.3.2.1. Typy całkowite

Do *typów całkowitych* zaliczamy:

Nazwa typu	Zakres	Liczba bitów[1]
`char`	$\{-128, -127, \ldots, 127\}$	8
`short`	$\{-32\,768, -32\,767, \ldots, 32\,767\}$	16
`int`	$\{-2\,147\,483\,648, \ldots, 2\,147\,483\,647\}$	32
`long long`	$\{\sim -9 \times 10^{18}, \ldots, \sim 9 \times 10^{18}\}$	64

Do reprezentacji liczb całkowitych będziemy korzystać z typu `int`. W 32-bitowych systemach operacyjnych typ ten pozwala zapisywać liczby rzędu $\pm$ 2 miliardów. W systemach 64-bitowych (zależy to ponadto od wersji kompilatora) typ ten być może pozwoli na reprezentację liczb z zakresu $\pm$ 9 trylionów.

Ciekawostka

Typ `int` reprezentuje najczęściej co najmniej 32-bitową (64-bitową w niektórych systemach operacyjnych i wersjach kompilatora) liczbę całkowitą w systemie U2 (ze znakiem). Czytelników zainteresowanych reprezentacją liczb odsyłamy do rozdz. 2.5.

Zapamiętaj

Zauważmy, że typy zmiennych nie reprezentują całego zbioru liczb całkowitych $\mathbb{Z}$, a jedynie ich podzbiór. Ich zakres jest jednak na tyle duży, że wystarcza do większości zastosowań praktycznych.

[1] Podana liczba bitów dotyczy *Microsoft Visual C++* działającym w środowisku 32 bitowym, zob. rozdz. 2.5.4.

Ciekawostka

Do każdego z wyżej wymienionych typów możemy zastosować modyfikator `unsigned` — otrzymujemy liczbę nieujemną (bez znaku), np.

$$\texttt{unsigned int} = \{0, 1, \ldots, 4{,}294{,}967{,}295\} \subset \mathbb{N}_0.$$

W codziennej praktyce jednak nie używa się go zbyt często.

Informacja

Stałe całkowite, np. `0`, `-5`, `15210` są reprezentantami typu `int` danymi w postaci dziesiętnej, czyli takiej, do której jesteśmy przyzwyczajeni.

Oprócz tego można używać stałych w innych systemach liczbowych (zob. rozdz. 2.5) np. postaci szesnastkowej, poprzedzając liczby przedrostkiem `0x`, np. `0x53a353` czy też `0xabcdef`, oraz ósemkowej, korzystając z przedrostka `0`, np. `0777`. Nie ma, niestety, możliwości definiowania bezpośrednio stałych w postaci dwójkowej.

Zapamiętaj

`010` oraz `10` oznaczają dwie różne liczby! Pierwsza z nich to liczba równa *osiem* (notacja ósemkowa), a druga — *dziesięć* (notacja dziesiętna).

2.3.2.2. Typy zmiennoprzecinkowe

Do *typów zmiennoprzecinkowych* zaliczamy:

Nazwa typu	Zakres (przybliżony)	Liczba bitów
`float`	$\subseteq [-10^{38}, 10^{38}]$	32
`double`	$\subseteq [-10^{308}, 10^{308}]$	64

Do reprezentacji liczb rzeczywistych będziemy używać najczęściej typu `double`.

Informacja

Stałe zmiennoprzecinkowe wprowadzamy podając zawsze część dziesiętną i część ułamkową rozdzieloną kropką (nawet gdy część ułamkowa jest równa 0), np. `3.14159`, `1.0` albo `-0.000001`. Domyślnie stałe te należą do typu `double`.

Dodatkowo, liczby takie można wprowadzać w formie *notacji naukowej*, używając do tego celu separatora `e`, który oznacza „razy dziesięć do potęgi”. I tak liczba `-2.32e-4` jest stałą o wartości $-2{,}32 \times 10^{-4}$.

Dokładne omówienie zagadnienia reprezentacji i arytmetyki liczb zmiennoprzecinkowych wykracza poza ramy naszego wykładu. Więcej szczegółów, w tym rachunek błędów arytmetyki tych liczb, przedstawiony będzie na wykładzie z metod numerycznych.

Zainteresowanym osobom można polecić następujący angielskojęzyczny artykuł (o bardzo sugestywnym tytule): D. Goldberg, What Every Computer Scientist Should Know About Floating-Point Arithmetic, *ACM Computing Surveys* 21(1), 1991, 5–48; zob. *www.ibspan.waw.pl/~gagolews/Teaching/aipp/priv/Czytelnia/Goldberg1991.pdf*.

Zapamiętaj

Warto mieć na uwadze zawsze problemy związane z użyciem liczb zmiennoprzecinkowych.

1. Nie da się reprezentować całego zbioru liczb rzeczywistych, a tylko jego podzbiór (właściwie jest to tylko podzbiór liczb wymiernych!). Wszystkie liczby są zaokrąglane do najbliższej reprezentowalnej.
2. Arytmetyka zmiennoprzecinkowa nie spełnia w pewnych przypadkach własności łączności i rozdzielności.

Ich konsekwencje będziemy badać wielokrotnie podczas zajęć laboratoryjnych.

Informacja

W standardzie IEEE-754, który jest stosowany w komputerach typu PC, określone zostały wartości specjalne, które mogą być przyjmowane przez zmienne typu zmiennoprzecinkowego:

- ∞ (`Inf`, ang. *infinity*),
- $-\infty$ (`-Inf`),
- nie-liczba (`NaN`, ang. *not a number*).

Np. $1/0$ = `Inf`, $0/0$ = `NaN`.

2.3.2.3. Typ logiczny

Dostępny jest także typ `bool`, służący do reprezentowania *zbioru wartości logicznych*.

Informacja

Zmienna typu logicznego może znajdywać się w dwóch stanach, określonych przez *stałe logiczne* `true` (prawda) oraz `false` (fałsz).

2.3.3. Identyfikatory

Mamy dwie możliwości użycia zmiennej: możemy *przypisać* jej pewną wartość (jednocześnie „kasując” wartość poprzednią) bądź przechowywaną weń wartość *odczytać*. Każda

zmienna musi mieć swój *identyfikator*, to jest jednoznaczną nazwę używaną po to, by można się było do niej odwołać. Identyfikatory mogą składać się z następujących znaków:
`a b c d e f g h i j k l m n o p q r s t u v w x y z`
`A B C D E F G H I J K L M N O P Q R S T U V W X Y Z _`
oraz z poniższych, pod warunkiem, że nie są one pierwszym znakiem identyfikatora:
`0 1 2 3 4 5 6 7 8 9`

Wobec tego przykładami poprawnych identyfikatorów są: `i`, `suma`, `wyrazenieZeWzoru3`, `_2m1a` oraz `zmienna_pomocnicza1`. Zgodnie jednak z podaną regułą np. `3523aaa`, `ala ma kota` oraz `:-)` nie mogą być identyfikatorami.

Identyfikatorami nie mogą być też *zarezerwowane słowa kluczowe języka C++* (np. **`int`**, **`if`**). Z tego powodu wyróżniamy je w niniejszym skrypcie pogrubioną czcionką.

Zapamiętaj

W języku C++ wielkość liter ma znaczenie!

Dla przykładu, `T` oraz `t` to identyfikatory dwóch różnych zmiennych. Dalej, np. identyfikator `Int` (poprawny!) nie jest tożsamy z **`int`** (słowo kluczowe).

Informacja

Dobrą praktyką jest nadawanie zmiennym takich nazw, by „same się objaśniały". Dzięki temu czytając kod programu będzie nam łatwiej dowiedzieć się, do czego dana zmienna jest używana.

Zmienne `poleKwadratu`, `delta`, `wspolczynnikX` w pewnych kontekstach posiadają wyżej wymienioną cechę. Często używamy też identyfikatorów `i`, `j`, `k` jako liczników w pętlach. Z drugiej strony, np. zmienne `dupa` albo `ratunku` niezbyt jasno mówią czytelnikowi, do czego mogą służyć.

2.3.4. Deklaracja zmiennych

Dzięki zmiennym pisane przez nas programy (ale także np. twierdzenia matematyczne) nie opisują jednego, szczególnego przypadku pewnego interesującego nas problemu (np. rozwiązania konkretnego układu dwóch równań liniowych), lecz wszystkie możliwe w danym kontekście (np. rozwiązania dowolnego układu dwóch równań liniowych). Taki mechanizm daje więc nam możliwość *uogólniania* (abstrakcji) algorytmów.

Oto prosty przykład. Ciąg twierdzeń:
Twierdzenie 1. *Pole koła o promieniu* 1 *wynosi* π.
Twierdzenie 2. *Pole koła o promieniu* 2 *wynosi* 4π.
Twierdzenie 3. ...
przez każdego matematyka (a nawet przyszłego matematyka) byłby uznany, łagodnie rzecz ujmując, za nieudany żart. Jednakże uogólnienie powyższych wyników przez odwołanie się do pewnej, wcześniej zadeklarowanej zmiennej sprawia, że wynik staje się interesujący.
Twierdzenie 4. *Niech* $r \in \mathbb{R}_+$. *Pole koła o promieniu* r *wynosi* πr^2.

Zauważmy, że w matematyce zdanie deklarujące zmienną „niech $r \in \mathbb{R}_+$" nie musi się pojawiać w takiej właśnie formie. Większość naukowców podchodzi do tego, rzecz jasna, w sposób elastyczny. Mimo to, pisząc np. „pole koła o promieniu $r > 0$ wynosi πr^2" bądź

nawet „pole koła o promieniu r wynosi πr^2”, domyślamy się, że chodzi właśnie o powyższą konstrukcję (czyli założenie, że r jest liczbą rzeczywistą dodatnią). Wiemy już jednak z pierwszego wykładu, że komputery nie są tak domyślne jak my.

Zapamiętaj

W języku C++ wszystkie zmienne muszą zostać *zadeklarowane przed* ich *pierwszym użyciem*.

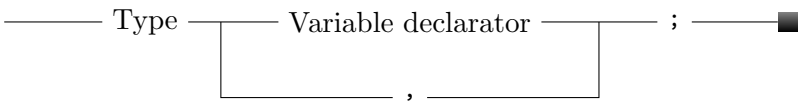
Aby zadeklarować zmienną, należy napisać w kodzie programu odpowiednią instrukcję o ściśle określonej składni.

Informacja

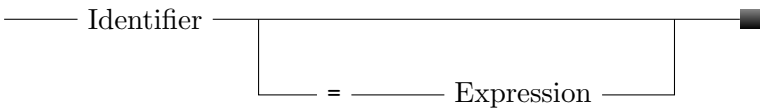
Instrukcja (ang. *statement*) jest podstawowym składnikiem pisanego programu. Dzięki niej możemy nakazać komputerowi wykonać pewną ustaloną czynność (być może bardzo złożoną), której znaczenie jest ściśle określone przez reguły semantyczne języka programowania.

Odpowiednikiem instrukcji w języku polskim jest w pewnym sensie zdanie rozkazujące. Na przykład „wynieś śmieci!”, „zacznij się wreszcie uczyć AiPP!” są dobrymi instrukcjami. Z drugiej strony zdanie „Jasiu, źle mi, zrób coś!” instrukcją nie jest, ponieważ nie ma ściśle określonego znaczenia (semantyki).

Składnia *instrukcji deklarującej zmienną lokalną* w języku C++ (ang. *local variable declaration statement*), czyli np. takiej, którą możemy umieścić w funkcji `main()`, jest następująca:



gdzie Type to typ zmiennej (np. `int` bądź `double`), a Variable declarator to tzw. deklarator zmiennej. Jest on postaci:



Tutaj Identifier oznacza identyfikator zmiennej, a Expression jest pewnym wyrażeniem. Więcej o wyrażeniach dowiemy się za chwilę. Na razie wystarczy nam wiedzieć, że wyrażeniem może być po prostu stała całkowita bądź zmiennoprzecinkowa albo też pewna operacja arytmetyczna.

Zapamiętaj

Prawie wszystkie instrukcje w języku C++ muszą być zakończone *średnikiem*!

Oto kilka przykładów instrukcji deklarujących zmienne:

```
int x;              // niech x będzie zmienną całkowitą
bool p = false;     /* niech p będzie wartością logiczną
                       ustawioną (na początku) na PRAWDA */
int y=10;
double a, b;        // niech a i b będą zmiennymi typu double
int u = y+10, v = -y;
```

Informacja

Tekst drukowany kursywą i szarą czcionką to *komentarze*. Komentarze są albo zawarte między znakami /* i */ (i wtedy mogą zajmować wiele wierszy kodu), albo następują po znakach // (i wtedy sięgają do końca aktualnego wiersza).

Treść komentarzy jest ignorowana przez kompilator i nie ma wpływu na wykonanie programu. Jednakże opisywanie tworzonego kodu jest bardzo dobrym nawykiem, bo sprawia, że jest on bardziej zrozumiały dla jego czytelnika.

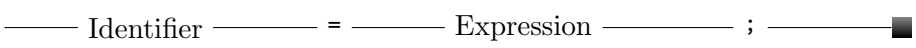
2.3.5. Operator przypisania

Deklaracja zmiennej powoduje przydzielenie jej pewnego miejsca w pamięci RAM komputera (poznamy to zagadnienie bardzo dokładnie na wykładzie o wskaźnikach).

Informacja

Zmienna nie jest inicjowana automatycznie. Dopóki nie przypiszemy jej jawnie jakiejś wartości, będzie przechowywać „śmieci”. Próba odczytania wartości niezainicjowanej zmiennej może (lecz nie musi, jest to zależne od kompilatora) skutkować nagłym zakończeniem uruchomionego programu.

Operator przypisania (ang. *assignment operator*), służy do nadania zmiennej określonej wartości. Składnię instrukcji, która go wykorzystuje, obrazuje poniższy diagram:



Co ważne, najpierw obliczana jest wartość wyrażenia Expression. Dopiero po tym wynik wpisywany jest do zmiennej o nazwie Identifier.

Prześledźmy poniższy kod.

```
int i, j;  // deklaracja dwóch zmiennych (niezainicjowanych)
i = 4;     // przypisanie i wartości 4, czytaj: i staje się 4
j = i;     /* przypisanie j wartości takiej,
              jaką aktualnie przechowuje zmienna i */
cout << j; // wypisze na ekran "4"
```

Informacja

Obiekt o nazwie `cout` umożliwia wypisywanie wartości zmiennych różnych typów na ekran monitora (a dokładnie, na tzw. standardowe wyjście). Wartość wyrażenia (powyżej: wartość przechowywaną w zmiennej `j`) „wysyła" się na ekran za pomocą operatora <<.

Inny sposób przypisywania wartości zmiennym, dzięki któremu wartości nie muszą być z góry ustalane podczas pisania programu, zapewnia obiekt `cin` i operator >>. Korzystając z nich możemy poprosić użytkownika już podczas *działania* napisanego przez nas programu o podanie za pomocą klawiatury konkretnych wartości, które mają być przypisane określonym zmiennym.

Przykład:

```
int x;
cout << "Podaj wartość x: ";
cin >> x; /* wczytanie wartości x z tzw. standardowego wejścia
             (domyślnie jest to klawiatura komputera) */
cout << "Teraz x=" << x << endl;
```

Ciekawostka

Przy okazji omawiania operatora przypisania, warto wspomnieć o możliwości definiowania *stałych symbolicznych*, według składni:

— `const` — Type — Identifier — = — Expr. — ; —■

Przy tworzeniu stałej symbolicznej należy od razu przypisać jej wartość. Cechą zasadniczą stałych jest to, że po ich zainicjowaniu nie można wcale zmieniać ich wartości. Ich stosowanie może zmniejszać prawdopodobieństwo omyłek programisty.

Przykłady:

```
const int dlugoscCiagu = 10;
const double pi = 3.14159;
const double G = 6.67428e-11; /* to samo, co 6,67428 · 10^-11. */
pi = 4.3623; // błąd - pi jest "tylko do odczytu"
```

2.3.6. Rzutowanie (konwersja) typów. Hierarchia typów

Zastanówmy się, co się stanie, jeśli zechcemy przypisać zmiennej jednego typu wartość innego typu.

```
double x; // deklaracja zmiennej niezainicjowanej typu double
int i = 4, j; /* deklaracja dwóch zmiennych typu int,
                 zmienna i zostanie zainicjowana wartością 4,
                 zmienna j jest niezainicjowana */

x = i;       /* inny typ! niejawna konwersja na 4.0,
                tzw. promocja - OK */

x = 3.14159;

```

```
11 j = x;       // błąd! (typ docelowy jest "mniej pojemny")
12 j = (int)x; // jawna konwersja (tzw. rzutowanie) — teraz OK
13
14 cout << j << ", " << x; // wypisze na ekran "3, 3.14159"
```

Przypisanie wartości typu `double` do zmiennej typu `int` zakończy się błędem kompilacji. Jest to spowodowane tym, że w wyniku działania takiej instrukcji mogłaby wystąpić utrata informacji. W języku C++ obowiązuje następująca *hierarchia typów*:

$$\overbrace{\texttt{bool}}^{\text{typ logiczny}} \subset \overbrace{\texttt{char} \subset \texttt{short} \subset \texttt{int} \subset \texttt{long long}}^{\text{typy całkowite}} \Subset \overbrace{\texttt{float} \subset \texttt{double}}^{\text{typy zmiennoprzecinkowe}} .$$

Promocja typu, czyli konwersja na typ silniejszy, bardziej „pojemny”, odbywa się automatycznie. Dlatego w powyższym kodzie instrukcja `x=i;` wykona się poprawnie.

Z drugiej strony, rzutowanie do słabszego typu musi być zawsze jawne. Należy *explicité* oznajmić kompilatorowi, że postępujemy świadomie. Służy do tego tzw. *operator rzutowania*. Taki operator umieszczamy przed rzutowanym wyrażeniem. Oprócz zastosowanej w powyższym przykładzie składni `(typ)wyrażenie`, dostępne są dodatkowo dwie równoważne: `typ(wyrażenie)` oraz `static_cast<typ>(wyrażenie)`.

Zapamiętaj

W przypadku rzutowania zmiennej typu zmiennoprzecinkowego do całkowitoliczbowego następuje *obcięcie części ułamkowej*, np. `(int)3.14` da w wyniku `3`, a `(int)-3.14` zaś — wartość `-3`.

Z kolei dla konwersji do typu `bool`, wartość równa `0` da zawsze wynik `false`, a różna od `0` będzie przekształcona na `true`.

Przy konwersji typu `bool` do całkowitoliczbowego wartość `true` zostaje zamieniona zawsze na `1`, a `false` na `0`.

2.4. Operatory

2.4.1. Operatory arytmetyczne

Operatory arytmetyczne mogą byś stosowane (poza wyszczególnionymi wyjątkami) na wartościach typu całkowitego i zmiennoprzecinkowego. Operatory binarne (czyli dwuargumentowe) prezentujemy w poniższej tabeli.

Operator	Znaczenie
`+`	dodawanie
–	odejmowanie
`*`	mnożenie
`/`	dzielenie rzeczywiste bądź całkowite
`%`	reszta z dzielenia (tylko liczby całkowite)

Informacja

Zauważmy, że w języku C++ nie ma określonego operatora potęgowania.

Zastosowanie operatorów do argumentów dwóch różnych typów powoduje konwersję operandu o typie słabszym do typu silniejszego.

Przykłady:

```
cout << 2+2 << endl;
cout << 3.0/2.0 << endl;
cout << 3/2 << endl;
cout << 7%4 << endl;
cout << 1/0.0 << endl; // uzgodnienie typów: to samo, co 1.0/0.0
```

Wynikiem wykonania powyższych instrukcji będzie:

```
4
1.5
1    (dlaczego?)
3
Inf
```

Oprócz operatorów arytmetycznych binarnych mamy też dostęp do kilku *operatorów unarnych*, czyli jednoargumentowych.

Operator	Znaczenie
+	unarny plus (nic nie robi)
−	unarny minus (zmiana znaku na przeciwny)
++	inkrementacja (przedrostkowy i przyrostkowy)
−−	dekrementacja (przedrostkowy i przyrostkowy)

Przykład:

```
int i = -4; // zmiana znaku stałej 4
i = -i;     // zmiana znaku zmiennej i
cout << i;  // wynik: 4
```

Operator *inkrementacji* (++) służy do zwiększania wartości zmiennej o 1, a operator *dekrementacji* (−−) zmniejszania o 1. Stosuje się je najczęściej na *zmiennych* typu całkowitego. Operatory te występują w dwóch wariantach: przedrostkowym (ang. *prefix*) oraz przyrostkowym (ang. *suffix*).

W przypadku operatora inkrementacji/dekrementacji *przedrostkowego*, wartością całego wyrażenia jest wartość zmiennej po wykonaniu operacji (operator ten działa tak, jakby najpierw aktualizował stan zmiennej, a potem zwracał swoją wartość w wyniku). Z kolei dla operatora *przyrostkowego* wartością wyrażenia jest stan zmiennej sprzed zmiany (najpierw zwraca wartość zmiennej, potem aktualizuje jej stan).

Zadanie

Przeanalizuj następujący przykładu.

```
int i = 5, j = 9;

--j;      // wynik: j==8, to samo to j = j-1;
j--;      //        j==7, to samo to j = j-1;

j = ++i; // wynik: i==6, j==6 (przedrostkowy)   (*)
j = i++; //        j==6, i==7 (przyrostkowy)    (*)

j = ++100; // błąd, 100 jest stałą
```

Podsumowując, `j = ++i;` można zapisać jako dwie instrukcje: `i = i+1; j = i;` Z drugiej strony, `j = i++;` jest równoważne operacjom: `j = i; i = i+1;`

Niewątpliwą zaletą tych dwóch operatorów jest możliwość napisania zwięzłego fragmentu kodu. Niestety, jak widać, odbywa się to kosztem czytelności. Nie polecamy zatem stosować ich w przypadkach takich jak te oznaczone gwiazdką w powyższym przykładzie.

2.4.2. Operatory relacyjne

Operatory relacyjne służą do porównywania dwóch wartości. Są to więc operatory binarne. Wynikiem porównania jest zawsze wartość typu `bool`.

Operator	Znaczenie
`==`	czy równe?
`!=`	czy różne?
`<`	czy mniejsze?
`<=`	czy nie większe?
`>`	czy większe?
`>=`	czy nie mniejsze?

Zapamiętaj

Często popełnianym błędem jest omyłkowe użycie operatora przypisania (=) w miejscu, w którym powinien wystąpić operator porównania (==).

Przykłady:

```
bool w1 = 1==1; // true
int  w2 = 2>=3; // 0, czyli (int)false
bool w3 = (0.0 == (((1e34 + 1e-34) - 1e34) - 1e-34)); // false!
   // Uwaga na porównywanie liczb zmiennoprzecinkowych - błędy!
```

2.4.3. Operatory logiczne

Operatory logiczne służą do przeprowadzania działań na wyrażeniach typu `bool` (lub takich, które są sprowadzalne do `bool`). W wyniku ich działania otrzymujemy wartość logiczną.

Operator	Znaczenie
`!`	negacja (unarny)
`||`	alternatywa (lub)
`&&`	koniunkcja (i)

Mamy, co następuje:

!	
false	true
true	false

||	false	true
false	false	true
true	true	true

&&	false	true
false	false	false
true	false	true

Kilka przykładów:

```
bool w1 = (!true);       // false
bool w2 = (1<2 || 0<1); // true -> (true || false)
bool w3 = (1.0 && 0.0); // false -> (true && false)
```

Ciekawostka

Okazuje się, że komputery są czasem leniwe. W przypadku wyrażeń składających się z wielu podwyrażeń logicznych, obliczane jest tylko to, co jest potrzebne do ustalenia wyniku. I tak w przypadku koniunkcji, jeśli pierwszy argument ma wartość `false`, to wyznaczanie wartości drugiego jest niepotrzebne. Podobnie jest dla alternatywy i pierwszego argumentu o wartości `true`. Na przykład:

```
int a = 0;
bool p = (true || (++a==0)); // leniwy
cout << a;                   // a==0
bool q = (true && (++a==0)); // tutaj już musi policzyć
cout << a;                   // a==1
```

Zwróćmy jednak uwagę, że czytelność takiego kodu pozostawia wiele do życzenia. Mimo to niektórzy programiści lubią stosować różne „sztuczki języka C++" w swoich programach. Dopóki dobrze nie opanujemy języka, starajmy się powstrzymać od tego typu skrótowców.

2.4.4. Operatory bitowe ⋆

Operatory bitowe działają na poszczególnych bitach liczb całkowitych (por. rozdz. 2.5). Zwracany wynik jest też liczbą całkowitą.

Operator	Znaczenie
`~`	bitowa negacja (unarny)
`|`	bitowa alternatywa
`&`	bitowa koniunkcja
`^`	bitowa alternatywa wyłączająca (albo, ang. *exclusive-or*)
`<< k`	przesunięcie w lewo o `k` bitów (ang. *shift-left*)
`>> k`	przesunięcie w prawo o `k` bitów (ang. *shift-right*)

Operator bitowej negacji zamienia każdy bit liczby na przeciwny. Operatory bitowej alternatywy, koniunkcji i alternatywy wyłączającej zestawiają bity na odpowiadających sobie pozycjach dwóch operandów według następujących reguł.

~	
0	1
1	0

|	0	1
0	0	1
1	1	1

&	0	1
0	0	0
1	0	1

^	0	1
0	0	1
1	1	0

Np. `0xb6 ^ 0x5f == 0xe9`, gdyż

$$\begin{array}{rcccccccc} & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hat{} & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ \hline & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}$$

Zapamiętaj

Nie należy mylić operatorów `~` (który w notacji matematycznej oznacza czasem logiczną negację) i `!` (który oznacza logiczną negację w C++).

Operatory przesunięcia zmieniają pozycje bitów w liczbie. Bity, które nie „mieszczą się" w danym typie są tracone. W przypadku operatora `<<`, na zwalnianych pozycjach

wstawiane są zera. Operator >> wstawia zaś na zwalnianych pozycjach bit znaku (por. rozdz. 2.5.4). Dla przykładu, dokonajmy przesunięcia bitów liczby 8 bitowej o 3 pozycje w lewo.

$$\underset{\texttt{0xab}}{10101011} \quad \texttt{<< 3} \quad == \quad \overbrace{101\,01011\mathbf{000}}^{\text{wynik}} \underset{\texttt{0x58}}{}$$

Oto kolejne przykłady:

```
int  p1 = 1 | 2;     // 3,  bo  00000001 | 00000010 -> 00000011
int  p2 = ~0xf0;     // 15, bo ~11110000           -> 00001111
int  p3 = 0xb ^ 5;   // 14, bo  00001011 ^ 000000101-> 00001110
int  p4 = 5 << 2;    // 20, bo  00000101 << 2      -> 00010100
int  p5 = 7 >> 1;    // 3,  bo  00000111 >> 1      -> 00000011
int  p6 = -128 >> 4; // -8, bo  10000000 >> 4      -> 11111000
```

Ciekawostka

Operacja n<<k równoważna jest (o ile nie nastąpi przepełnienie) $n \times 2^k$, a n>>k jest równoważna całkowitoliczbowej operacji $n \times 2^{-k}$ (k krotne dzielenie całkowite przez 2).

Informacja

Zauważmy, że operatory << i >> mają inne znaczenie w przypadku użycia ich na obiektach, odpowiednio, cout i cin. Jest to praktyczny przykład zastosowania tzw. *przeciążania operatorów*, czyli różnicowania ich znaczenia w zależności od kontekstu. Więcej na ten temat dowiemy się podczas wykładu z *Programowania obiektowego* na II semestrze.

2.4.5. Operatory łączone

Dla skrócenia zapisu zostały też wprowadzone tzw. *operatory łączone*, które w zwięzły sposób łączą operacje arytmetyczne i przypisanie:

$$\texttt{+=},\ \texttt{-=},\ \texttt{*=},\ \texttt{/=},\ \texttt{\%=}.$$

Np. x += 10; znaczy to samo, co x = x + 10;

Ciekawostka

W analogiczny sposób można też łączyć operatory bitowe.

2.4.6. Priorytety operatorów

Zaobserwowaliśmy, że w jednym wyrażeniu można używać wielu operatorów. Kolejność wykonywania działań jest jednak ściśle określona. Domyślnie wyrażenia obliczane są według priorytetów, zestawionych w kolejności od największego do najmniejszego w tab. 2.1.

Dla przykładu, zapis `1+2*0` test tożsamy z `1+(2*0)`, ponieważ operacja mnożenia ma wyższy priorytet niż dodawanie. Co ważne, w przypadku operatorów o tym samym priorytecie, operacje wykonywane są w kolejności od lewej do prawej, np. `4*3/2` oznacza to samo, co `(4*3)/2`.

Zapamiętaj

W przypadku jakiejkolwiek wątpliwości co do priorytetu stosowanych operacji, określoną kolejność działań możemy zawsze wymusić za pomocą nawiasów okrągłych (...).

Tab. 2.1. Priorytety operatorów

13	`++`, `--` (przyrostkowe)
12	`++`, `--` (przedrostkowe), `+`, `-` (unarne), `!`, `~` (⋆)
11	`*`, `/`, `%`
10	`+`, `-`
9	`<<`, `>>` (⋆)
8	`<`, `<=`, `>`, `>=`
7	`==`, `!=`
6	`&` (⋆)
5	`^` (⋆)
4	`|` (⋆)
3	`&&`
2	`||`
1	`=`, `+=`, `-=`, `*=`, `/=`, `%=`

Przykłady:

```
int p1  = 2+4*3/2;    // p1 = (2+((4*3)/2));
p1 += 2>3 == !true;   /* p1 += ((2>3) == (!true));
                          czyli to samo, co p1++;
                      doskonały przykład na to, jak nie pisać! */
cout << p1;           // 9
```

2.5. Reprezentacja liczb całkowitych ⋆

Powiedzieliśmy wcześniej, że wszelkie informacje w komputerze reprezentowane są przez ciągi bitów. Zainteresowany Czytelnik z pewnością będzie chciał się dowiedzieć, w jaki sposób zapisywane są *liczby całkowite*.

Zacznijmy od liczb nieujemnych.

Informacja

Systemy liczbowe można podzielić na pozycyjne i addytywne. Oto przykłady każdego z nich.

Pozycyjne systemy liczbowe:

1. system dziesiętny (decymalny, indyjsko-arabski) — o podstawie 10,
2. system dwójkowy (binarny) — o podstawie 2,
3. system szesnastkowy (heksadecymalny) — o podstawie 16,
4. ...

Addytywne systemy liczbowe:

1. system rzymski,
2. system sześćdziesiątkowy (Mezopotamia).

Tutaj szczególnie będą interesować nas systemy pozycyjne, w tym używany na co dzień system dziesiętny.

2.5.1. System dziesiętny ⋆

System dziesiętny (decymalny, ang. *decimal*), przyjęty w Europie zachodniej w XVI w., to pozycyjny system liczbowy o podstawie 10. Do zapisu liczb używa się w nim następujących symboli (cyfr): $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$.

Jako przykład rozważmy liczbę 1945_{10} (przyrostek $_{10}$ wskazuje, że chodzi nam o liczbę w systemie dziesiętnym). Jak w każdym systemie pozycyjnym istotne jest tu, na której pozycji stoi każda z cyfr.

	1	9	4	5	$_{10}$
$=$	**1**	0	0	0	
$+$		**9**	0	0	
$+$			**4**	0	
$+$				**5**	

Równoważne:

	1	9	4	5	$_{10}$
$=$	**1**				$\times 1000$
$+$		**9**			$\times 100$
$+$			**4**		$\times 10$
$+$				**5**	$\times 1$

Numerację pozycji cyfr zaczynamy od 0. Dalej oznaczamy je w kierunku od prawej do lewej, czyli od najmniej do najbardziej znaczącej.

	1	9	4	5	$_{10}$
$=$	**1**				$\times 10^3$
$+$	$\vdots$	**9**			$\times 10^2$
$+$	$\vdots$	$\vdots$	**4**		$\times 10^1$
$+$	$\vdots$	$\vdots$	$\vdots$	**5**	$\times 10^0$
	3	2	1	0	

Ogólnie rzecz biorąc, mając daną liczbę n cyfrową, zapisaną w postaci *ciągu* cyfr dziesiętnych $b_{n-1}b_{n-2}\dots b_1b_0$, gdzie $b_i \in \{0, 1, \dots, 9\}$ dla $i = 0, 1, \dots, n-1$, jej wartość można określić za pomocą wzoru

$$\sum_{i=0}^{n-1} b_i \times 10^i.$$

Liczbę 10 występującą we wzorze nazywamy *podstawą systemu*.

Jak łatwo się domyślić, można określić systemy pozycyjne o innych podstawach. Systemy o podstawie 2 i 16 są najbardziej przydatne z punktu widzenia informatyki.

2.5.2. System dwójkowy ⋆

Dwójkowy pozycyjny system liczbowy (binarny, ang. *binary*) to system pozycyjny o podstawie 2. Używanymi tu symbolami (cyframi) są tylko 0 oraz 1. Zauważmy, że cyfry te mają właśnie „przełącznikową" interpretację: prąd nie płynie/płynie czy też przełącznik wyłączony/włączony. Zatem n cyfrowa liczba w tym systemie może opisywać stan n przełączników naraz.

Jako przykład rozpatrzmy liczbę 101001_2.

	1	0	1	0	0	1	$_2$	
$=$	**1**							$\times 2^5$
$+$		**0**						$\times 2^4$
$+$			**1**					$\times 2^3$
$+$				**0**				$\times 2^2$
$+$					**0**			$\times 2^1$
$+$						**1**		$\times 2^0$

	1	0	1	0	0	1	$_2$	
$=$	**1**							$\times 32$
$+$		**0**						$\times 16$
$+$			**1**					$\times 8$
$+$				**0**				$\times 4$
$+$					**0**			$\times 2$
$+$						**1**		$\times 1$

Zatem $101001_2 = 32_{10} + 8_{10} + 1_{10} = 41_{10}$. Jest to przykład *konwersji* (zamiany) podstawy liczby dwójkowej na dziesiętną.

Konwersja liczb do systemu dziesiętnego jest stosunkowo prosta. Przyjrzyjmy się jak przekształcić liczbę dziesiętną na dwójkową. Na listingu 2.1 podajemy algorytm, który może być wykorzystany w tym celu.

Listing 2.1. Konwersja liczby dziesiętnej na dwójkową

```
// Wejscie: liczba k ∈ ℕ.
// Wyjście: postać tej liczby w systemie dwójkowym (kolejne cyfry
    będą wypisywane na ekran w kolejności od lewej do prawej).
niech i – największa liczba naturalna taka, że k ⩾ 2^i;
dopóki (i ⩾ 0)
{
    jeśli (k ⩾ 2^i)
    {
        cout << 1;
        k = k – 2^i;
    }
    w_przeciwnym_przypadku
        cout << 0;

    i = i – 1;
}
```

Tab. 2.2. Przykład: konwersja liczby 1945_{10} na dwójkową

	k	2^i	i	
	1945	~~2048~~		
	1945	1024	10	1
1945-1024 =	921	512	9	1
921-512 =	409	256	8	1
409-256 =	153	128	7	1
153-128 =	25	~~64~~	6	0
	25	~~32~~	5	0
	25	16	4	1
25-16=	9	8	3	1
9-8=	1	~~4~~	2	0
	1	~~2~~	1	0
	1	1	0	1
1-1=	0			↑

Dla przykładu, rozważmy liczbę 1945_{10}. Tablica 2.2 opisuje kolejno wszystkie kroki potrzebne do uzyskania wyniku w postaci dwójkowej. Możemy z niej odczytać, iż $1945_{10} = 11110011001_2$.

2.5.3. System szesnastkowy ⋆

System szesnastkowy (heksadecymalny, ang. *hexadecimal*) jest systemem pozycyjnym o podstawie 16. Zgodnie z przyjętą konwencją, używanymi symbolami (cyframi) są: $0, 1, \ldots, 9, A, B, C, D, E$ oraz F. Jak widzimy, wykorzystujemy tutaj (z konieczności) także kolejne litery alfabetu, które mają swoją liczbową (a właściwie cyfrową) interpretację.

Po co taki system, skoro komputer właściwie opiera się na liczbach w systemie binarnym? Jak można zauważyć, liczba w postaci dwójkowej prezentuje się na kartce lub na ekranie dosyć... okazale. Korzystając z faktu, że $16 = 2^4$, możemy użyć jednego symbolu w zapisie szesnastkowym zamiast czterech w systemie dwójkowym. Fakt ten sprawia również, że konwersja z systemu binarnego na heksadecymalny i odwrotnie jest bardzo prosta.

Tablica 2.3 przedstawia cyfry systemu szesnastkowego i ich wartości w systemie dwójkowym oraz dziesiętnym.

Tab. 2.3. Cyfry systemu szesnastkowego i ich wartości w systemie dwójkowym oraz dziesiętnym

BIN	DEC	HEX
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

BIN	DEC	HEX
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Zatem na przykład $1011110_2 = 5E_{16}$, gdyż grupując kolejne cyfry dwójkowe czwórkami,

od prawej do lewej, otrzymujemy

$$\overbrace{0101}^{5}\overbrace{1110}^{\mathrm{E}}{}_2.$$

Podobnie postępujemy dokonując konwersji w przeciwną stronę, np. $2\mathrm{D}_{16} = 101101_2$, bowiem

$$\overbrace{2}^{0010}\overbrace{\mathrm{D}}^{1101}{}_{16}.$$

Jak widzimy, dla uproszczenia zapisu początkowe zera możemy pomijać bez straty znaczenia (wyjątek w rozdz. 2.5.4!).

Konwersji z i do postaci dziesiętnej możemy dokonywać za pośrednictwem opanowanej już przez nas postaci binarnej.

2.5.4. System U2 reprezentacji liczb ze znakiem ⋆

Interesującym jest pytanie, w jaki sposób można reprezentować w komputerze liczby ze znakiem, np. -1001_2? Jak widzimy, znak jest nowym (trzecim) symbolem, a elektroniczny przełącznik może znajdować się tylko w dwóch stanach.

Spośród kilku możliwości rozwiązania tego problemu, obecnie w większości komputerów osobistych używany jest tzw. *kod uzupełnień do dwóch*, czyli *kod U2*. Niewątpliwą jego dodatkową zaletą jest to, iż pozwala na bardzo łatwą realizację operacji dodawania i odejmowania (zob. ćwiczenia).

Zapamiętaj

W notacji U2 najstarszy (czyli stojący po lewej stronie) bit determinuje znak liczby. I tak:

- najstarszy bit równy 0 oznacza liczbę nieujemną,
- najstarszy bit równy 1 oznacza liczbę ujemną.

Jeśli dana jest n cyfrowa liczba w systemie U2 postaci $b_{n-1}b_{n-2}\ldots b_1b_0$, gdzie $b_i \in \{0,1\}$ dla $i = 0, 1, \ldots, n-1$, jej wartość wyznaczamy następująco:

$$-b_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} b_i \times 2^i.$$

Zatem dla liczby n bitowej najstarszy bit mnożymy nie przez 2^{n-1}, lecz przez -2^{n-1}. Dla przykładu, $0110_{U2} = 110_2$ oraz

$$1011_{U2} = -\mathbf{1} \times 2^3 + \mathbf{0} \times 2^2 + \mathbf{1} \times 2^1 + \mathbf{1} \times 2^0 = -5_{10} = -101_2.$$

Co ważne, najstarszego bitu równego 0 nie można dowolnie pomijać w zapisie! Mamy bowiem $0b_{n-2}\ldots b_{0\mathrm{U}2} \neq b_{n-2}\ldots b_{0\mathrm{U}2}$, gdy $b_{n-2} = 1$. Jednakże,

$$0b_{n-2}\ldots b_{0\mathrm{U}2} = 00\ldots 0b_{n-2}\ldots b_{0\mathrm{U}2}.$$

Z drugiej strony, można udowodnić (zob. ćwiczenia), że również

$$1b_{n-2}\ldots b_{0\mathrm{U}2} = 11\ldots 1b_{n-2}\ldots b_{0\mathrm{U}2}.$$

Informacja

Łatwo pokazać, że użycie k bitów pozwala na zapis liczb $-2^{k-1}, \ldots, 2^{k-1} - 1$, np. liczby 8 bitowe w systemie U2 mogą mieć wartości od -128 do 127.

2.6. Ćwiczenia

Zadanie 2.1. Jaką wartość mają następujące liczby dane w notacji naukowej: a) `4.21e6`, b) `1.95323e2`, c) `2.314e-4`, d) `4.235532e-2`?

Zadanie 2.2. Wyznacz wartość następujących wyrażeń. Ponadto, określ jakiego są one typu (`int` czy `double`).

1. `10.0+15.0/2+4.3`,
2. `10.0+15/2+4.3`,
3. `3.0*4/6+6`,
4. `20.0-2/6+3`,
5. `10+17*3+4`,
6. `10+17/3.0+4`,
7. `3*4%6+6`,
8. `3.0*4%6+6`,
9. `10+17%3+4`.

Zadanie 2.3. Dane są trzy zmienne zadeklarowane i zainicjowane w następujący sposób.

```
double a = 10.6, b = 13.9, c = -3.42;
```

Oblicz wartość poniższych wyrażeń.

1. `int(a)`,
2. `int(c)`,
3. `int(a+b)`,
4. `int(a)+b+c`,
5. `int(a+b)*c`,
6. `double(int(a))/c`.

Zadanie 2.4. Korzystając z przekształceń logicznych (np. praw De Morgana, praw rozdzielności) uprość następujące wyrażenia.

1. `!(!p)`,
2. `!p && !q` ,
3. `!(!p || !q || !r)`,
4. `!(b>a && b<c)`,
5. `!(a>=b && b>=c && a >=c)`,
6. `(a>b && a<c) || (a<c && a>d)`,
7. `p || !p` .

Zakładamy, że `a,b,c,d` są typu `double`, a `p,q,r` typu `bool`.

Zadanie 2.5. [MD] Napisz w języku C++ funkcję `abs()`, która obliczy wartość bezwzględną danej liczby rzeczywistej.

Zadanie 2.6. [MD] Napisz (w pseudokodzie, w którym korzysta się z poznanych już operatorów) funkcję `pierwiastek()`, która będzie wyznaczać pierwiastek kwadratowy danej liczby rzeczywistej dodatniej x według podanego algorytmu. (Uwaga: zakładamy, że `epsilon` jest ustaloną, małą wartością, a nie parametrem funkcji).

```
przyblizenie=1
dopóki różnica dwóch ostatnich przybliżeń jest większa niż epsilon
    przyblizenie = srednia z przyblizenie oraz x/przyblizenie
zwróć przybliżenie
```

Prześledź działanie algorytmu np. dla `x=9` oraz `epsilon=1`.

Zadanie 2.7. [MD] Wyznacz typ i wartość następujących wyrażeń:

1. `1+0.5`,
2. `1.0==1`,
3. `5/2 - 2.5`,
4. `pierwiastek(4)==2`,
5. `pierwiastek(9)-int(3.5)`,
6. `6/2 + 7/2`,
7. `abs(pierwiastek(9)-3)<1`.

Reprezentacje liczb całkowitych ⋆

⋆ **Zadanie 2.8.** Przedstaw następujące liczby całkowite nieujemne w postaci binarnej, dziesiętnej i szesnastkowej: $10_{10}, 18_{10}, 18_{16}, 101_2, 101_{10}, 101_{16}, \mathrm{ABCDEF}_{16}, 64135312_{10}, 110101111001_2, \mathrm{FFFFF0C2}_{16}$.

⋆ **Zadanie 2.9.** Dana jest n cyfrowa liczba w systemie U2 zapisana jako ciąg cyfr $b_{n-1}b_{n-2}\ldots\ldots b_1b_0$, gdzie $b_i \in \{0,1\}$ dla $i = 0, 1, \ldots, n-1$. Pokaż, że powielenie pierwszej cyfry dowolną liczbę razy nie zmienia wartości danej liczby, tzn. $b_{n-1}b_{n-2}\ldots b_1b_0 = b_{n-1}b_{n-1}\ldots\ldots b_{n-1}b_{n-2}\ldots b_1b_0$.

⋆ **Zadanie 2.10.** Przedstaw następujące liczby dane w notacji U2 jako liczby dziesiętne: 10111100, 00111001, 1000000111001011.

⋆ **Zadanie 2.11.** Przedstaw następujące liczby dziesiętne w notacji U2 (do zapisu użyj 8, 16 lub 32 bitów): $-12, 54, -128, -129, 53263, -32000, -56321, -3263411$.

⋆ **Zadanie 2.12.** Dana jest wartość $x \in \mathbb{N}$. Napisz wzór matematyczny, który opisze najmniejszą liczbę cyfr potrzebnych do zapisania wartości x w systemie a) binarnym, b) dziesiętnym, c) szesnastkowym.

⋆⋆ **Zadanie 2.13.** Rozważmy operacje dodawania i odejmowania liczb nieujemnych w reprezentacji binarnej. Oblicz wartość następujących wyrażeń korzystając z metody analogicznej do sposobu „szkolnego” (dodawania i odejmowanie słupkami). Pamiętaj jednak, że np. $1_2 + 1_2 = 10_2$ (tzw. przeniesienie).

1. $1000_2 + 111_2$,
2. $1000_2 + 1111_2$,
3. $1110110_2 + 11100111_2$,
4. $11111_2 + 11111111_2$,
5. $1111_2 - 0001_2$,
6. $1000_2 - 0001_2$,
7. $10101010_2 - 11101_2$,
8. $11001101_2 - 10010111_2$.

⋆⋆ **Zadanie 2.14.** Okazuje się, że liczby w systemie U2 można dodawać i odejmować tą samą metodą, co liczby nieujemne w reprezentacji binarnej. Oblicz wartość następujących wyrażeń i sprawdź otrzymane wyniki, przekształcając je do postaci dziesiętnej. Uwaga: operacji dokonuj na dwóch liczbach n bitowych, a wynik podaj również jako n bitowy.

1. $1000_{\mathrm{U2}} + 0111_{\mathrm{U2}}$,
2. $10110110_{\mathrm{U2}} + 11100111_{\mathrm{U2}}$,
3. $10101010_{\mathrm{U2}} - 00011101_{\mathrm{U2}}$,
4. $11001101_{\mathrm{U2}} - 10010111_{\mathrm{U2}}$.
5. $10001101_{\mathrm{U2}} - 01010111_{\mathrm{U2}}$.

⋆⋆ **Zadanie 2.15.** Dana jest liczba w postaci U2. Pokaż, że aby uzyskać liczbę do niej przeciwną, należy odwrócić wartości jej bitów (dokonać ich negacji, tzn. zamienić zera na jedynki i odwrotnie) i dodać do wyniku wartość 1. Ile wynoszą wartości $0101_{\mathrm{U2}}, 1001_{\mathrm{U2}}$ i 0111_{U2} po dokonaniu tych operacji? Sprawdź uzyskane rezultaty za pomocą konwersji tych liczb do systemu dziesiętnego.

⋆⋆ **Zadanie 2.16.** [SK] Dana jest liczba w postaci binarnej. W jaki sposób za pomocą jednej operacji logicznej, jednej bitowej oraz jednej arytmetycznej sprawdzić, czy dana liczba jest potęgą liczby 2.

⋆ **Zadanie 2.17.** [MD] Napisz algorytm, który znajdzie binarną reprezentację dodatniej liczby naturalnej. Wskazówka: jako wynik wypisuj na ekran kolejne cyfry dwójkowe, od najmłodszej do najstarszej.

⋆ **Zadanie 2.18.** [MD] Dany jest ciąg n liczb o elementach ze zbioru $\{0,1\}$, który reprezentuje pewną liczbę w postaci U2. Wartości te będą wczytywane z klawiatury, w kolejności tym razem od najstarszego bitu do najmłodszego. Napisz algorytm, który wypisze wartość tej liczby w systemie dziesiętnym.

⋆ **Zadanie 2.19.** Jaki wynik dadzą poniższe operacje bitowe dla zmiennych typu `short`?

1. `0x0FCD | 0xFFFF`,
2. `364 & 0x323`,
3. `~163`,
4. `0xFC93 ^ 0x201D`,
5. `14 << 4`,
6. `0xf5a3 >> 8`.

2.7. Wskazówki i odpowiedzi do ćwiczeń

Odpowiedź do zadania 2.2.

1. `10.0+15.0/2+4.3 == 21.8` (`double`).
2. `10.0+15/2+4.3 == 21.3` (`double`).
3. `3.0*4/6+6 == 8.0` (`double`).
4. `20.0−2/6+3 == 23.0` (`double`).
5. `10+17*3+4 == 65` (`int`).
6. `10+17/3.0+4` $\simeq$ `19.6667` (`double`).
7. `3*4%6+6 == 6` (`int`).
8. `3.0*4%6+6` — błąd kompilacji.
9. `10+17%3+4 == 16` (`int`).

□

Odpowiedź do zadania 2.8.

1. $10_{10} = 1010_2 = A_{16}$.
2. $18_{10} = 10010_2 = 12_{16}$.
3. $18_{16} = 11000_2 = 24_{10}$.
4. $101_2 = 5_{10} = 5_{16}$.
5. $101_{10} = 1100101_2 = 65_{16}$.
6. $101_{16} = 100000001_2 = 257_{10}$.
7. $\mathrm{ABCDEF}_{16} = 101010111100110111101111_2 = 11259375_{10}$.
8. $64135312_{10} = 11111001001010000010010000_2 = 3D2A090_{16}$.
9. $110101111001_2 = D79_{16} = 3449_{10}$.
10. $\mathrm{FFFFF0C2}_{16} = 11111111111111111111000011000010_2 = 4294963394_{10}$.

□

Odpowiedź do zadania 2.10.

1. $10111100_{\mathrm{U2}} = -68$.
2. $00111001_{\mathrm{U2}} = 57_{10}$.
3. $1000000111001011_{\mathrm{U2}} = -32309$.

□

Odpowiedź do zadania 2.11.

1. $-12_{10} = 11110100_{\mathrm{U2}}$.
2. $54_{10} = 00110110_{\mathrm{U2}}$.
3. $-128_{10} = 10000000_{\mathrm{U2}}$.
4. $-129_{10} = 1111111101111111_{\mathrm{U2}}$.
5. $53263_{10} = 00000000000000001101000000001111_{\mathrm{U2}}$.
6. $-32000_{10} = 11111111111111111000001100000000_{\mathrm{U2}}$.
7. $-56321_{10} = 11111111111111110010001111111111_{\mathrm{U2}}$.
8. $-3263411_{10} = 11111111100111000110100010011101_{\mathrm{U2}}$.

□

Odpowiedź do zadania 2.13.

1. $1000_2 + 111_2 = 1111_2$.
2. $1000_2 + 1111_2 = 10111_2$.
3. $1110110_2 + 11100111_2 = 101011101_2$, gdyż

	1	1	1			1	1		
			1	1	1	0	1	1	0
+		1	1	1	0	0	1	1	1
=	1	0	1	0	1	1	1	0	1

4. $11111_2 + 11111111_2 = 100011110_2$.
5. $1111_2 - 0001_2 = 1110_2$.
6. $1000_2 - 0001_2 = 111_2$.
7. $10101010_2 - 11101_2 = 10001101_2$, gdyż

				-1	-1	-1		-1	
	1	0	1	0	1	0	1	0	
-					1	1	1	0	1
=	1	0	0	0	1	1	0	1	

8. $11001101_2 - 10010111_2 = 110110_2$, gdyż

		-1	-1		-1	-1			
	1	1	0	0	1	1	0	1	
-	1	0	0	1	0	1	1	1	
=	0	0	1	1	0	1	1	0	

□

Odpowiedź do zadania 2.14.

1. $1000_{U2} + 0111_{U2} = 1111_{U2} = -1_{10} = -8_{10} + 7_{10}$.
2. $10110110_{U2}+11100111_{U2} = 110011101_{U2} = 10011101_{U2} = -99_{10} = -74_{10}+(-25)_{10}$.
3. $10101010_{U2} - 00011101_{U2} = 10001101_{U2} = -115_{10} = -86_{10} - 29_{10}$.
4. $11001101_{U2} - 10010111_{U2} = 00110110_{U2} = 54_{10} = -51_{10} - (-105_{10})$.
5. $10001101_{U2} - 01010111_{U2} = 00110110_{U2} = 54_{10} \neq -115_{10} - 87_{10} = -202_{10}$! Wystąpiło tzw. *przepełnienie*, czyli przekroczenie zakresu wartości dla liczby 8-bitowej (ale mamy $2^8 - 202_{10} = 54_{10}$). Można zapisać operandy na większej liczbie bitów, np. szesnastu:

$$1111111110001101_{U2} - 0000000001010111_{U2} = 1111111100110110_{U2} = -202_{10}.$$

□

Wskazówka do zadania 2.16.

```
niech x ∈ ℕ;
zwróć (x & (x − 1)) == 0 jako wynik;
```

Dlaczego tak jest? Wykaż. □

Odpowiedź do zadania 2.18.

```
// Wejście: n > 0 oraz x[0], x[1], ..., x[n − 1] ∈ {0, 1}
// Wyjście: wartość x w systemie dziesiętnym
niech ret, pot ∈ ℕ;

ret = 0;
pot = 1;
dla (i=1,2,...,n−1)
{
  ret = ret + pot*x[n−i];
  pot = pot * 2;
}
ret = ret − pot * x[0];
zwróć ret jako wynik;
```

□

Odpowiedź do zadania 2.19.

1. `0x0FCD | 0xFFFF == −1 (0xFFFF)`.
2. `364 & 0x323 == 288 (0x0120)`.
3. `~163 == −164 (0xFF5C)` (dlaczego -164?).
4. `0xFC93 ^ 0x201D == −9074 (0xDC8E)`.
5. `14 << 4 == 224 (0x00E0)`.
6. `0xf5a3 >> 8 == 0xFFF5` (bit znaku == 1).

□