Programowanie. Od podstaw

Autor: Adrian Kingsley-Hughes, Kathie Kingsley-Hughes
Tłumaczenie: Radosław Meryk
ISBN: 83-246-0057-4
Tytuł oryginału: Beginning Programming
Format: B5, stron: 464
Przykłady na ftp: 449 kB

Mimo dynamicznego rozwoju informatyki wśród większości użytkowników komputerów nadal istnieje przekonanie, że programowanie jest zajęciem dla wybrańców posiadających ogromną i niemal tajemną wiedzę. Tymczasem pisanie programów to umiejętność, którą może opanować każdy. Oczywiście nauka programowania wymaga poznania wielu zagadnień teoretycznych i praktycznych, ale nie wymaga od osoby uczącej się żadnych niezwykłych umiejętności. Każdy może poznać zasady pisania programów, zarówno dla celów hobbystycznych, jak i zawodowych.

Książka „Programowanie. Od podstaw" to podręcznik programowania przeznaczony dla osób, które dopiero rozpoczynają swoją komputerową przygodę. Zawiera uniwersalne wiadomości przydatne każdemu programiście niezależnie od tego, co i w jakim języku będzie tworzyć. Czytając ją, poznasz wady i zalety różnych języków programowania, sposoby realizacji typowych zadań programistycznych i metody testowania aplikacji. Dowiesz się, jak komputery przechowują informacje, jakie systemy liczbowe wykorzystuje się w programowaniu i jakie narzędzia będą Ci potrzebne podczas pracy. Zdobędziesz solidne podstawy, które pozwolą Ci na dalsze rozwijanie swoich umiejętności.

- Sposób interpretacji kodu źródłowego przez komputer
- System binarny i szesnastkowy
- Warsztat pracy programisty
- Komentarze w kodach programów
- Definiowanie zmiennych
- Instrukcje warunkowe
- Testowanie i usuwanie błędów
- Projektowanie interfejsów użytkownika
- Operacje na plikach
- Wykorzystywanie rejestru Windows
- Zarządzanie wersjami kodu
- Kompilacja

Przekonaj się, że programowanie nie jest trudne

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

4

Od pojęć do kodu — język kodu

W tym rozdziale przeanalizujemy kilka elementów programowania, które pozwalają na przekształcenie pojęć programistycznych na kod. Idealnym językiem programowania byłby język naturalny, za pomocą którego można by wydawać komputerowi proste rozkazy. Na przykład:

```
Wydrukuj ten dokument.
Ten element wyświetl w kolorze zielonym.
Dodaj do siebie dwie ostatnie liczby.
```

Teoretycznie byłby to najłatwiejszy język, ponieważ jest najbardziej zbliżony do języka naturalnego. Byłby to również język najwyższego poziomu (im wyższy poziom, tym bliżej do języka naturalnego). Stworzenie takiego języka jest jednak nierealne ze względu na niejednoznaczność języka mówionego i pisanego oraz złożoność mechanizmu potrzebnego do jego interpretowania (takim mechanizmem jest ludzki mózg). Na przykład wszystkie trzy poniższe instrukcje mają identyczne znaczenie. Komputery nie znają sposobu, aby to stwierdzić:

```
Wydrukuj ten dokument.
Wydrukuj bieżący dokument.
Wyślij ten dokument na drukarkę.
```

Możliwości wyrażenia tej myśli jest znacznie więcej. Ludzie bez trudu to zrozumieją — komputery nie.

> Możliwość rozumienia skomplikowanych konstrukcji jest oczywiście celem, do którego dążą programiści pracujący nad językami programowania. Zwróćmy uwagę, że w filmach science fiction (*Star Trek*, *Star Wars* i wielu innych) ludzie komunikują się z komputerami za pomocą języka mówionego. Byłoby doskonale, gdyby to było możliwe w rzeczywistości.

Ponieważ do komunikacji z komputerami nie możemy wykorzystać języka naturalnego, musimy stosować języki specjalne — języki programowania. W tym rozdziale jeszcze nie będziemy się nimi zajmować; zamiast tego zajmiemy się pewnymi elementami, które tworzą takie języki. Rozpoczniemy od prostego schematu, który komputery wykorzystują do komunikacji — systemu dwójkowego.

System dwójkowy

System *dwójkowy* (nazywany też *binarnym*) jest systemem liczbowym. Różni się od systemu dziesiętnego, który używamy na co dzień tym, że zamiast dziesięciu cyfr (0,1, 2, 3, 4, 5, 6, 7, 8 i 9) wykorzystuje się w nim tylko dwie (0 i 1). Cyfry wykorzystywane w systemie dwójkowym określa się jako bity.

Dla wielu osób zrozumienie systemu dwójkowego stwarza problemy, ponieważ wykorzystywane w nim bity — 0 i 1 (cyfry, które wszyscy znamy) są wykorzystywane w niestandardowy sposób. Gdyby zamiast cyfr 0 i 1 wykorzystano inne symbole, być może zrozumienie systemu stwarzałoby mniej problemów.

Jak interpretować system dwójkowy?

Najprostszym sposobem przeanalizowania systemu dwójkowego jest porównanie go ze znanym systemem dziesiętnym.

Dziesiętnie	Dwójkowo
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011

Zwróćmy uwagę na to, w jaki sposób wykonywana jest inkrementacja. Ponieważ występują tylko dwa bity, wydaje się, że liczby wzrastają szybciej (z powodu większej liczby pozycji), ale nie należy się tym sugerować. Liczby dwójkowe czyta się od prawej do lewej. Im bliżej lewej strony znajduje się bit, tym większa jego waga.

Prostym sposobem nauczenia się interpretacji liczb dwójkowych jest wykorzystanie sposobu nauczania czytania liczb, jaki stosuje się w szkołach. Weźmy na przykład następującą liczbę:

```
1010
```

Gdyby była to liczba dziesiętna, przeczytalibyśmy ją w następujący sposób:

Tysiące	Setki	Dziesiątki	Jedności
1	0	1	0

Jeśli zsumujemy poszczególne pozycje, uzyskamy wartość liczby — tysiąc dziesięć.

W systemie dwójkowym znaczenie poszczególnych pozycji jest nieco inne. Nie są to kolejne potęgi liczby 10, ale liczby 2 (tzn. wartość kolejnych pozycji podwaja się).

Ósemki	Czwórki	Dwójki	Jedności
1	0	1	0

A zatem liczba 1010 ma następującą wartość:

1	×	8	=	8
0	×	4	=	0
1	×	2	=	2
0	×	1	=	0
			Razem dziesiętnie	10

Aby uzyskać dziesiętną wartość dwójkowej liczby 1010, wystarczy zsumować liczby w prawej kolumnie. Uzyskaliśmy wynik dziesięć.

A zatem liczba 1010 to jedna ósemka i jedna dwójka, czyli 10.

Odczytanie większej liczby dwójkowej wymaga zastosowania tabelki o większej liczbie kolumn. Weźmy następującą liczbę dwójkową:

```
10010111
```

Najpierw utworzymy tabelkę (pamiętając, że idąc od prawej do lewej, wartość kolejnej kolumny to podwojona wartość kolumny poprzedniej):

Sto dwudziestki ósemki	Sześćdziesiątki czwórki	Trzydziestki dwójki	Szesnastki	Ósemki	Czwórki	Dwójki	Jedności
1	0	0	1	0	1	1	1

Aby obliczyć wartość liczby w systemie dziesiętnym, utworzymy tabelkę tak, jak poprzednio, i umieścimy poszczególne bity w lewej kolumnie:

1	×	128	=	128
0	×	64	=	0
0	×	32	=	0
1	×	16	=	16
0	×	8	=	0
1	×	4	=	4
1	×	2	=	2
1	×	1	=	1
			Razem dziesiętnie	151

A zatem w systemie dziesiętnym liczba dwójkowa 10010111 to 151.

Duże liczby

Kiedy opowiadamy dzieciom o liczbach, pierwszą rzeczą, którą robią, jest zapisanie możliwie największej liczby. W przypadku 8 bitów największa możliwa liczba ma wartość 11111111. Oto sposób obliczenia jej wartości:

1	×	128	=	128
1	×	64	=	64
1	×	32	=	32
1	×	16	=	16
1	×	8	=	8
1	×	4	=	4
1	×	2	=	2
1	×	1	=	1
			Razem dziesiętnie	**255**

Wystarczy dodać więcej bitów, a liczba będzie większa!

```
1111111111111111
```

1	×	32768	=	32768
1	×	16384	=	16384
1	×	8192	=	8192
1	×	4096	=	4096
1	×	2048	=	2048
1	×	1024	=	1024
1	×	512	=	512

1	×	256	=	256
1	×	128	=	128
1	×	64	=	64
1	×	32	=	32
1	×	16	=	16
1	×	8	=	8
1	×	4	=	4
1	×	2	=	2
1	×	1	=	1
			Razem dziesiętnie	**65535**

Grupowanie bitów

Dla wygody grupom bitów nadano nazwy, dzięki którym łatwiej je zapamiętać i posługiwać się nimi. Przyjrzyjmy się im pokrótce.

Bit

Bit to pojedyncza cyfra dwójkowa. Bitem jest:

```
0
```

Podobnie jak:

```
1
```

Maksymalna wartość, jaką można przedstawić za pomocą jednego bitu, to oczywiście 1.

Półbajt

Grupę czterech bitów nazywa się *półbajtem* (ang. *nibble*). Oto przykładowy półbajt:

```
1010
```

Maksymalna wartość, jaką można przedstawić za pomocą półbajtu, to 15:

```
1111
```

1	×	8	=	8
1	×	4	=	4
1	×	2	=	2
1	×	1	=	1
			Razem dziesiętnie	**15**

Bajt

Grupa ośmiu bitów to *bajt*.

```
10101010
```

Maksymalna wartość, jaką można przedstawić za pomocą bajtu, to 255:

```
11111111
```

1	×	128	=	128
1	×	64	=	64
1	×	32	=	32
1	×	16	=	16
1	×	8	=	8
1	×	4	=	4
1	×	2	=	2
1	×	1	=	1
			Razem dziesiętnie	**255**

Półsłowo

Grupę 16 bitów określa się jako *półsłowo*.

```
1010101010101010
```

Maksymalna wartość, jaką można przedstawić za pomocą półsłowa, to 65535:

```
1111111111111111
```

1	×	32768	=	32768
1	×	16384	=	16384
1	×	8192	=	8192
1	×	4096	=	4096
1	×	2048	=	2048
1	×	1024	=	1024
1	×	512	=	512
1	×	256	=	256
1	×	128	=	128
1	×	64	=	64

1	×	32	=	32
1	×	16	=	16
1	×	8	=	8
1	×	4	=	4
1	×	2	=	2
1	×	1	=	1
			Razem dziesiętnie	**65535**

Słowo

Grupę 32 bitów nazywamy *słowem*. Oto przykład słowa:

```
10101010101010101010101010101010
```

Maksymalna wartość, jaką można przedstawić za pomocą słowa, to 4294967295:

```
11111111111111111111111111111111
```

1	×	2147483648	=	2147483648
1	×	1073741824	=	1073741824
1	×	536870912	=	536870912
1	×	268435456	=	268435456
1	×	134217728	=	134217728
1	×	67108864	=	67108864
1	×	33554432	=	33554432
1	×	16777216	=	16777216
1	×	8388608	=	8388608
1	×	4194304	=	4194304
1	×	2097152	=	2097152
1	×	1048576	=	1048576
1	×	524288	=	524288
1	×	262144	=	262144
1	×	131072	=	131072
1	×	65536	=	65536
1	×	32768	=	32768
1	×	16384	=	16384
1	×	8192	=	8192

1	×	4096	=	4096
1	×	2048	=	2048
1	×	1024	=	1024
1	×	512	=	512
1	×	256	=	256
1	×	128	=	128
1	×	64	=	64
1	×	32	=	32
1	×	16	=	16
1	×	8	=	8
1	×	4	=	4
1	×	2	=	2
1	×	1	=	1
			Razem dziesiętnie	**4294967295**

Podwójne słowo

Grupę 64 bitów nazywa się *podwójnym słowem*. Oto przykład:

```
1010101010101010101010101010101010101010101010101010101010101010
```

Maksymalna wartość, jaką można przedstawić za pomocą podwójnego słowa, wyrażona w postaci dziesiętnej, to 18446744073709551615:

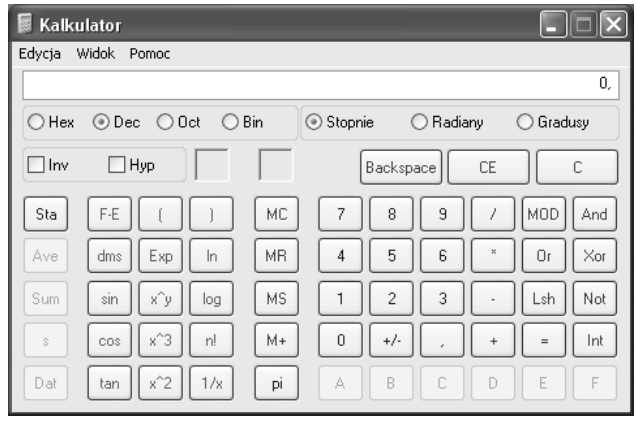
```
1111111111111111111111111111111111111111111111111111111111111111
```

Arytmetyka dwójkowa

Obliczenia z liczbami dwójkowymi wykonuje się w sposób podobny do operacji na liczbach dziesiętnych. Jednak dla potrzeb zrozumienia materiału zawartego w tej książce nauka teorii arytmetyki binarnej nie jest konieczna. Zamiast tego zaprezentuję o wiele łatwiejszy sposób wykonywania obliczeń z liczbami dwójkowymi — wykorzystanie narzędzia o rozbudowanych możliwościach, które ma do tego dodatkową zaletę — jest darmowe. Owo narzędzie pokazano na rysunku 4.1.

Tak, to windowsowy Kalkulator! Dla użytkowników systemu Windows jest to doskonała aplikacja służąca do tego celu. Użytkownicy innych systemów operacyjnych również powinni poszukać kalkulatora z możliwością wykonywania operacji na liczbach dwójkowych.

Rysunek 4.1.



Wykorzystanie Kalkulatora Windows

Oto szybki przewodnik wykorzystania Kalkulatora Windows do operacji na liczbach dwójkowych. Jeśli nie korzystamy z Kalkulatora Windows zbyt często, prawdopodobnie po uruchomieniu przyjmie on postać taką, jak pokazano na rysunku 4.1, a nie taką, jak na rysunku 4.2.

Rysunek 4.2.

Na rysunku pokazano windowsowy Kalkulator w trybie standardowym. Aby wykonywać operacje na liczbach dwójkowych, trzeba przełączyć go do trybu naukowego. W tym celu należy kliknąć menu *Widok*, a następnie pozycję *Naukowy*, tak jak pokazano na rysunku 4.3.

Rysunek 4.3.

Wyświetli się Kalkulator w trybie naukowym, w którym dostępnych jest wiele nowych przycisków i funkcji.

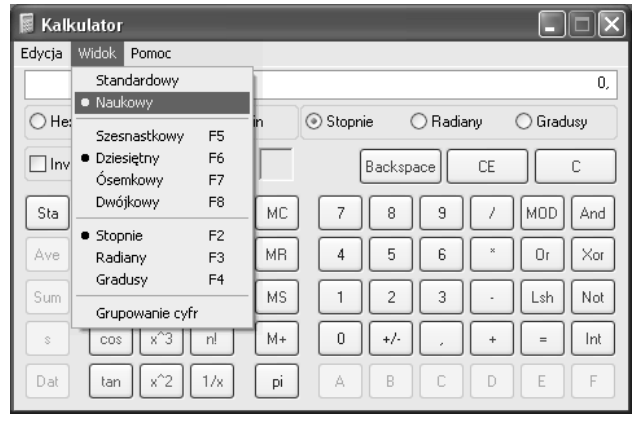
> Jeśli z naszego komputera PC korzystają inni użytkownicy, którym potrzebny jest Kalkulator do wykonywania prostych działań, pamiętajmy o ponownym przełączeniu go do trybu standardowego po wykonaniu działań! To tak na wszelki wypadek, aby ich nie przestraszyć.

W trybie naukowym są dostępne elementy sterujące do zmiany formatu liczb. Liczby dziesiętne reprezentuje napis *Dec*, natomiast dwójkowe — *Bin* (rysunek 4.4).

Rysunek 4.4.

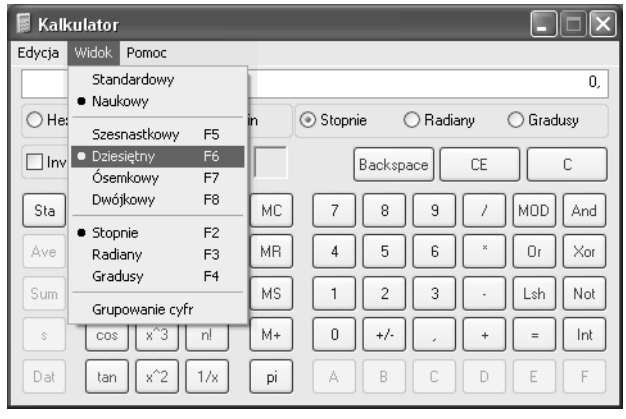
Format liczb można także zmienić za pomocą menu. Służy do tego menu *Widok*, tak jak pokazano na rysunku 4.5.

Rysunek 4.5.



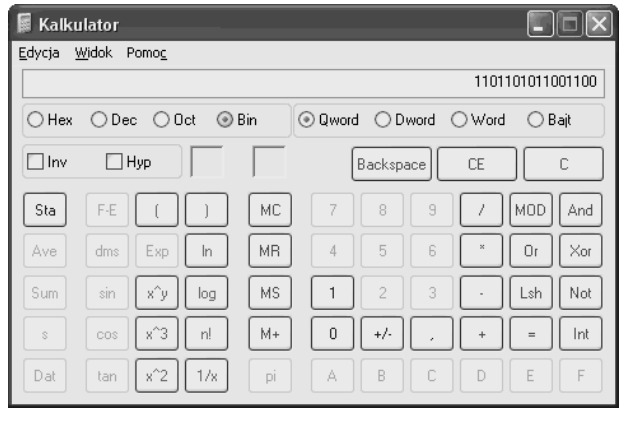
Ustawiamy Kalkulator w tryb liczb dziesiętnych i wpisujemy liczbę. Aby obejrzeć tę liczbę w postaci dwójkowej, wystarczy przełączyć Kalkulator z trybu *Dec* (rysunek 4.6)

Rysunek 4.6.



na *Bin*, co pokazano na rysunku 4.7

Rysunek 4.7.



…i liczba dziesiętna wyświetla się w formacie dwójkowym.

Każdą operację arytmetyczną — dodawanie, odejmowanie, mnożenie i dzielenie — wykonuje się za pomocą Kalkulatora tak jak zwykle — na liczbach dziesiętnych. Aby zobaczyć wynik w formacie dwójkowym, wystarczy odpowiednio przełączyć widok. Nie potrzeba używać ołówka i papieru, by robić to ręcznie.

Warto zwrócić uwagę, że opcje Kalkulatora obejmują także inne systemy liczbowe. *Hex* oznacza format szesnastkowy, natomiast *Oct* — ósemkowy. Omówienie formatu ósemkowego wykracza poza zakres tej książki i, ściśle rzecz biorąc, nie ma wielkiego znaczenia, natomiast format szesnastkowy będzie opisany w następnym podrozdziale tego rozdziału.

Dlaczego system dwójkowy?

Często można usłyszeć pytanie o to, dlaczego trzeba używać systemu dwójkowego? Dlaczego po prostu nie korzystać z systemu dziesiętnego?

Odpowiedź jest prosta: system dwójkowy umożliwia reprezentowanie dwóch stanów — włączenia i wyłączenia. Oznacza to, że w systemie komputerowym (lub dowolnym innym systemie elektronicznym) informacje można reprezentować za pomocą przełączników. Zmiana stanu tych przełączników z włączenia na wyłączenie i z powrotem umożliwia wykonywanie działań na liczbach dwójkowych. Jest to podstawa działania procesorów i pamięci (zarówno pamięci RAM, jak i twardych dysków).

System dwójkowy można także przedstawić za pomocą impulsów elektrycznych przesyłanych w przewodach. Jest to podstawa działania sieci i internetu.

System dwójkowy może sprawiać wrażenie rozwlekłego i kłopotliwego, ale w rzeczywistości tworzy on system nerwowy wszystkich urządzeń związanych z przesyłaniem, przetwarzaniem i zapisywaniem danych. System dwójkowy jest wszędzie — płynie w obwodach, przewodach i w eterze.

System szesnastkowy

System *szesnastkowy* (nazywany także *heksadecymalnym*) jest systemem liczbowym podobnym do dwójkowego i dziesiętnego, ale jest nieco bardziej skomplikowany.

Dlaczego bardziej skomplikowany? Otóż w systemie dwójkowym są dwie cyfry, w dziesiętnym — 10, natomiast w szesnastkowym jest ich 16:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E i F.

> Czasami można spotkać liczby szesnastkowe, w których zamiast liter ABCDEF stosowane są odpowiednio znaki ~, !, @, #, $ i %.

Interpretacja liczb szesnastkowych

Przyjrzyjmy się systemowi szesnastkowemu i porównajmy go z dziesiętnym. W ten sposób zapoznamy się z nim bliżej.

Dziesiętnie	Szesnastkowo
0	0
1	1
2	2
3	3
4	4
5	5
6	6

7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
...	...
30	1E
31	1F
32	20
33	21

Zwróćmy uwagę na to, w jaki sposób wykonywana jest inkrementacja. Ponieważ jest 16 cyfr, wydaje się, że liczby wzrastają wolniej niż w systemie dziesiętnym. Podobnie jak w systemie dwójkowym, liczby szesnastkowe czyta się od prawej do lewej. Im bliżej lewej strony znajduje się bit, tym większa jego waga.

> Słowo *heksadecymalny* (ang. *hexadecimal*) jest trochę dziwne, ponieważ przedrostek *hexa* wywodzi się od greckiego słowa *hexifor* — 6, natomiast słowo *decimal* pochodzi z łaciny i oznacza liczbę 10. Starszy i bardziej dokładny jest pochodzący z łaciny termin *szesnastkowy* (ang. *sexidecimal*). Terminu zaprzestano używać z powodu nieprzyzwoitych skojarzeń (nie mówiąc już o tym, że używano go także do określenia systemu sześćdziesiątkowego).

Liczby szesnastkowe poprzedza się przedrostkiem *0x* lub przyrostkiem *h*. Na przykład:

```
0x3F7D
```

Dla potrzeb objaśnienia zasad dotyczących systemu szesnastkowego pominiemy przyrostki i przedrostki, które tylko dodatkowo komplikują zapis i są mylące dla początkujących.

Aby przekształcić liczbę szesnastkową na postać dwójkową, wystarczy przekształcić każdą cyfrę szesnastkową na 4-bitową liczbę dwójkową. Tak więc przytoczoną poprzednio liczbę 3F7D można przekształcić na liczbę dwójkową w następujący sposób:

3	F	7	D
0011	1111	0111	1101

Przekształcenie dowolnego ciągu reprezentującego liczbę dwójkową na postać liczby szesnastkowej również nie powinno sprawiać problemu. Weźmy następującą liczbę:

```
101111
```

Wystarczy dodać kilka zer z lewej strony liczby, tak aby liczba jej bitów dzieliła się na 4 bez reszty:

```
00101111
```

Następnie podzielić liczbę na grupy 4-bitowe (półbajty):

```
0010 1111
```

Ostatnią czynnością jest zamiana każdej grupy na liczbę szesnastkową:

```
2 F
```

A zatem uzyskaliśmy następującą liczbę szesnastkową:

```
2F
```

Zrobione!

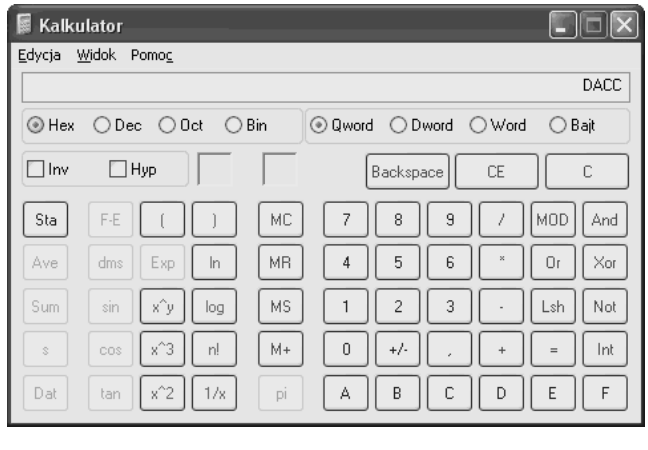
Każdy półbajt można przedstawić w postaci jednej cyfry szesnastkowej.

> Z podanych powyżej powodów bajt zazwyczaj jest przedstawiany jako dwa półbajty.

System szesnastkowy a Kalkulator w systemie Windows

Za pomocą Kalkulatora Windows można wykonywać operacje na liczbach szesnastkowych, podobnie jak na dwójkowych. Format wyświetlanych liczb można zmieniać za pomocą przycisków interfejsu Kalkulatora lub za pośrednictwem systemu menu. Liczby można przekształcać z systemu szesnastkowego na dziesiętny oraz z dziesiętnego na szesnastkowy w podobny sposób, jak w przypadku liczb dwójkowych. Pokazano to na rysunku 4.8.

Rysunek 4.8.



Reprezentacja znaków

Do tej pory omówiliśmy kilka sposobów reprezentowania liczb, ale jak powszechnie wiadomo, użytkowanie komputerów to nie tylko przetwarzanie liczb. Są jeszcze dane tekstowe, symbole i inne znaki. W jaki sposób przejść od systemu dwójkowego (stanowiącego podstawę działania wszystkich komputerów) do znaków wpisywanych z klawiatury lub wyświetlanych na ekranie?

Aby umożliwić reprezentowanie znaków w komputerach, opracowano system *ASCII* (wymawia się *aski*). Skrót ASCII pochodzi od *American Standard Code for Information Interchange* — amerykański standard kodu wymiany informacji. Ponieważ komputery potrafią przetwarzać tylko liczby, kod ASCII opisuje przekształcenie znaków, takich jak *a* lub ~, albo określonych działań (jak np. wysuw arkusza papieru) na liczby.

Kod ASCII to standardowy sposób przekształcenia wielkich i małych liter alfabetu angielskiego, a także cyfr i znaków specjalnych. Przekształcenie to wykonywane jest z wykorzystaniem systemu dwójkowego — do reprezentacji wymienionych znaków potrzeba 7 bitów. Jak łatwo policzyć, za pomocą 7 bitów można przedstawić 128 znaków.

DZIESIĘTNIE	ÓSEMKOWO	SZESNASTKOWO	DWÓJKOWO		WARTOŚĆ
000	000	000	00000000	NUL	(Znak Null.)
001	001	001	00000001	SOH	(Początek nagłówka)
002	002	002	00000010	STX	(Początek tekstu)
003	003	003	00000011	ETX	(Koniec tekstu)
004	004	004	00000100	EOT	(Koniec transmisji)
005	005	005	00000101	ENQ	(Zapytanie)
006	006	006	00000110	ACK	(Potwierdzenie)
007	007	007	00000111	BEL	(Sygnał dźwiękowy)
008	010	008	00001000	BS	(Powrót o 1 pozycję)
009	011	009	00001001	HT	(Tabulacja pozioma)
010	012	00A	00001010	LF	(Wysuw wiersza)
011	013	00B	00001011	VT	(Tabulacja pionowa)
012	014	00C	00001100	FF	(Wysuw arkusza)
013	015	00D	00001101	CR	(Powrót karetki)
014	016	00E	00001110	SO	(Zmiana kodu)
015	017	00F	00001111	SI	(Powrotna zmiana kodu)
016	020	010	00010000	DLE	(Przekodowanie znaków sterujących)
017	021	011	00010001	DC1	(XON)(Sterowanie urządzeniem 1)
018	022	012	00010010	DC2	(Sterowanie urządzeniem 2)
019	023	013	00010011	DC3	(XOFF)(Sterowanie urządzeniem 3)
020	024	014	00010100	DC4	(Sterowanie urządzeniem 4)
021	025	015	00010101	NAK	(Potwierdzenie negatywne)
022	026	016	00010110	SYN	(Tryb synchroniczny)
023	027	017	00010111	ETB	(Koniec bloku transmisji)
024	030	018	00011000	CAN	(Anulowanie)
025	031	019	00011001	EM	(Koniec nośnika)
026	032	01A	00011010	SUB	(Zastąpienie)

027	033	01B	00011011	ESC	(Wyjście)
028	034	01C	00011100	FS	(Separator pliku)
029	035	01D	00011101	GS	(Separator grupy)
030	036	01E	00011110	RS	(Żądanie wysłania separatora rekordu)
031	037	01F	00011111	US	(Separator jednostki)
032	040	020	00100000	SP	(Spacja)
033	041	021	00100001	!	(wykrzyknik)
034	042	022	00100010	"	(cudzysłów)
035	043	023	00100011	#	(hash)
036	044	024	00100100	$	(dolar)
037	045	025	00100101	%	(procent)
038	046	026	00100110	&	(ampersand)
039	047	027	00100111	'	(apostrof)
040	050	028	00101000	(	(lewy nawias - otwierający)
041	051	029	00101001	)	(prawy nawias - zamykający)
042	052	02A	00101010	*	(gwiazdka)
043	053	02B	00101011	+	(plus)
044	054	02C	00101100	,	(przecinek)
045	055	02D	00101101	-	(minus lub myślnik)
046	056	02E	00101110	.	(kropka)
047	057	02F	00101111	/	(ukośnik)
048	060	030	00110000	0	
049	061	031	00110001	1	
050	062	032	00110010	2	
051	063	033	00110011	3	
052	064	034	00110100	4	
053	065	035	00110101	5	
054	066	036	00110110	6	
055	067	037	00110111	7	
056	070	038	00111000	8	
057	071	039	00111001	9	
058	072	03A	00111010	:	(dwukropek)
059	073	03B	00111011	;	(średnik)
060	074	03C	00111100	<	(mniejszy niż)
061	075	03D	00111101	=	(znak równości)
062	076	03E	00111110	>	(większy niż)
063	077	03P	00111111	?	(znak zapytania)
064	100	040	01000000	@	(symbol AT)
065	101	041	01000001	A	
066	102	042	01000010	B	
067	103	043	01000011	C	
068	104	044	01000100	D	
069	105	045	01000101	E	
070	106	046	01000110	F	
071	107	047	01000111	G	
072	110	048	01001000	H	
073	111	049	01001001	I	

074	112	04A	01001010	J	
075	113	04B	01001011	K	
076	114	04C	01001100	L	
077	115	04D	01001101	M	
078	116	04E	01001110	N	
079	117	04F	01001111	O	
080	120	050	01010000	P	
081	121	051	01010001	Q	
082	122	052	01010010	R	
083	123	053	01010011	S	
084	124	054	01010100	T	
085	125	055	01010101	U	
086	126	056	01010110	V	
087	127	057	01010111	W	
088	130	058	01011000	X	
089	131	059	01011001	Y	
090	132	05A	01011010	Z	
091	133	05B	01011011	[	(lewy nawias kwadratowy)
092	134	05C	01011100	\	(lewy ukośnik)
093	135	05D	01011101	]	(prawy nawias kwadratowy)
094	136	05E	01011110	^	(daszek)
095	137	05F	01011111	_	(podkreślenie)
096	140	060	01100000	`	
097	141	061	01100001	a	
098	142	062	01100010	b	
099	143	063	01100011	c	
100	144	064	01100100	d	
101	145	065	01100101	e	
102	146	066	01100110	f	
103	147	067	01100111	g	
104	150	068	01101000	h	
105	151	069	01101001	i	
106	152	06A	01101010	j	
107	153	068	01101011	k	
108	154	06C	01101100	l	
109	155	06D	01101101	m	
110	156	06E	01101110	n	
111	157	06F	01101111	o	
112	160	070	01110000	p	
113	161	071	01110001	q	
114	162	072	01110010	r	
115	163	073	01110011	s	
116	164	074	01110100	t	
117	165	075	01110101	u	
118	166	076	01110110	v	
119	167	077	01110111	w	
120	170	078	01111000	x	
121	171	079	01111001	y	

122	172	07A	01111010	z	
123	173	07B	01111011	{	(lewy nawias klamrowy)
124	174	07C	01111100	|	(pionowa kreska)
125	175	07D	01111101	}	(prawy nawias klamrowy)
126	176	07E	01111110	~	(tylda)
127	177	07F	01111111	DEL	(usuń)

Zwróćmy uwagę, że do zaprezentowania powyższych znaków wykorzystano 7-bitowe liczby dwójkowe. Jak wspomniałem wcześniej, w ten sposób można przedstawić 127 znaków. Zbiór reprezentowanych znaków tworzy tzw. *zestaw znaków*.

> Dlaczego 7 bitów? Pierwsze 127 kodów zarezerwowano dla najważniejszych znaków, aby zmniejszyć ilość miejsca potrzebnego do przechowywania danych. W wielu aplikacjach wykorzystywanych w początkach komputerów ósme zero było pomijane.

Aby można było przedstawić więcej znaków, takich jak znaki narodowe lub symbole matematyczne, wprowadzono ósmy bit. Dzięki temu można przedstawić 256 znaków. Istnieje wiele różnych zestawów znaków. Poniżej zaprezentowano tzw. *rozszerzony zestaw znaków*.

WARTOŚĆ	DZIESIĘTNIE	SZESNASTKOWO
€	128	80
--	129	81
‚	130	82
--	131	83
„	132	84
…	133	85
†	134	86
‡	135	87
--	136	88
‰	137	89
Š	138	8a
<	139	8b
Ś	140	8c
Ť	141	8d
Ž	142	8e
Ź	143	8f
--	144	90
‘	145	91
’	146	92
“	147	93
”	148	94
•	149	95
-	150	96
–	151	97
--	152	98

™	153	99
š	154	9a
›	155	9b
ś	156	9c
ť	157	9d
ž	158	9e
ź	159	9f
	160	a0
ˇ	161	a1
˘	162	a2
Ł	163	a3
¤	164	a4
Ą	165	a5
¦	166	a6
§	167	a7
¨	168	a8
©	169	a9
Ş	170	aa
«	171	ab
¬	172	ac
--	173	ad
®	174	ae
Ż	175	af
°	176	b0
±	177	b1
˛	178	b2
ł	179	b3
´	180	b4
µ	181	b5
¶	182	b6
·	183	b7
¸	184	b8
ą	185	b9
ş	186	ba
»	187	bb
Ľ	188	bc
˝	189	bd
ľ	190	be
ż	191	bf
Ŕ	192	c0
Á	193	c1

Â	194	c2
Ă	195	c3
Ä	196	c4
Ĺ	197	c5
Ć	198	c6
Ç	199	c7
Č	200	c8
É	201	c9
Ę	202	ca
Ë	203	cb
Ě	204	cc
Í	205	cd
Î	206	ce
Ď	207	cf
Đ	208	d0
Ń	209	d1
Ň	210	d2
Ó	211	d3
Ô	212	d4
Ő	213	d5
Ö	214	d6
×	215	d7
Ř	216	d8
Ů	217	d9
Ú	218	da
Ű	219	db
Ü	220	dc
Ý	221	dd
Ţ	222	de
ß	223	df
ŕ	224	e0
á	225	e1
â	226	e2
ă	227	e3
ä	228	e4
ĺ	229	e5
ć	230	e6
ç	231	e7
č	232	e8
é	233	e9
ę	234	ea
ë	235	eb

ě	236	ec
í	237	ed
î	238	ee
ď	239	ef
đ	240	f0
ń	241	f1
ň	242	f2
ó	243	f3
ô	244	f4
ő	245	f5
ö	246	f6
÷	247	f7
ř	248	f8
ů	249	f9
ú	250	fa
ű	251	fb
ü	252	fc
ý	253	fd
ţ	254	fe
˙	255	ff

Symbol -- oznacza, że dla określonego kodu nie zdefiniowano znaku.

Dzięki wykorzystaniu kodu ASCII można przekształcić dane tekstowe na ciągi zer i jedynek oraz w drugą stronę — z postaci ciągów zer i jedynek na postać tekstową. Przekształcenie to nie powoduje utraty danych i jest jednoznaczne.

Weźmy na przykład następujący ciąg znaków:

```
Programowanie jest fajne!
```

Dzięki wykorzystaniu tablicy ASCII można przekształcić te znaki na bajty zapisane w formacie dwójkowym. Oto, co uzyskamy:

```
01010000011100100110111101100111011100100110000101101101011011110111011011001
01101110011010010110010100100000011010100110010101110011011101000100000011001100110000
01 01101010011011100110010100100001
```

Znak	Kod dwójkowy
P	01010000
r	01110010
o	01101111
g	01100111
r	01110010
a	01100001

Znak	Kod dwójkowy
m	01101101
o	01101111
w	01110111
a	01100001
n	01101110
i	01101001
e	01100101
	00100000
j	01101010
e	01100101
s	01110011
t	01110100
	00100000
f	01100110
a	01100001
j	01101010
n	01101110
e	01100101
!	00100001

Przekształcenie ciągu zer i jedynek na postać tekstową nie jest trudne. Aby to zrobić ręcznie, najpierw trzeba podzielić go na bajty. Przetwarzanie długich ciągów zer i jedynek jest uciążliwe — bardzo łatwo o popełnienie błędu.

```
01010000 01110010 01101111 01100111 01110010 01100001 01101101 01101111 01110111
01100001 01101110 01101001 01100101 00100000 01101010 01100101 01110011 01110100
00100000 01100110 01100001 01101010 01101110 01100101 00100001
```

Po podzieleniu ciągu zer i jedynek na bajty wystarczy skorzystać z tablicy ASCII i odczytać odpowiednie znaki:

```
   P       r       o       g       r       a       m       o       w
01010000 01110010 01101111 01100111 01110010 01100001 01101101 01101111 01110111
   a       n       i       e               j       e       s       t
01100001 01101110 01101001 01100101 00100000 01101010 01100101 01110011 01110100
           f       a       j       n       e       !
00100000 01100110 01100001 01101010 01101110 01100101 00100001
```

Przekształcenie nie powoduje utraty danych.

W nowoczesnym programowaniu programista niezbyt często jest zmuszony do przekształcania znaków na postać dwójkową. W większości przypadków operacje te są wykonywane automatycznie. Czasami jednak zdarzają się sytuacje, w których znajomość relacji pomiędzy liczbami dwójkowymi i znakami jest przydatna (między innymi pokazuje, w jaki sposób dane są reprezentowane w pamięci i na twardym dysku — rysunek 4.9).

Rysunek 4.9.

Warto zapamiętać, że znaki wyświetlane na ekranie nie mają nic wspólnego z kodem dwójkowym. Liczby dwójkowe (a także szesnastkowe i ósemkowe) są przekształcane na znaki, a dopiero potem wyświetlane przez aplikację lub system operacyjny.

Przeanalizowaliśmy różne formaty liczb oraz opisaliśmy sposób wykorzystania liczb do reprezentowania znaków wyświetlanych na ekranie, przetwarzanych bądź zapisywanych w pamięci zewnętrznej. W następnym podrozdziale zajmiemy się operatorami arytmetycznymi i logicznymi.

Operatory

Programowanie w dużej części polega na przetwarzaniu danych zarówno tekstowych, jak i liczbowych. Operatory umożliwiają wykonywanie operacji na danych oraz porównywanie informacji.

Rozpoczniemy od opisania podstawowych operatorów. Dzięki temu zapoznamy się z ich przeznaczeniem i działaniem. Należy pamiętać, że nie dla wszystkich operatorów w różnych językach programowania są stosowane takie same symbole (jest to jedna z najczęstszych przyczyn frustracji wśród osób, które przystępują do nauki drugiego języka programowania). Na razie zapamiętajmy więc przeznaczenie podstawowych operatorów. To z pewnością przyda się na dalszych etapach nauki.

Operatory można podzielić na pięć kategorii. Każda z nich zostanie opisana w kolejnych punktach:

- arytmetyczne,
- przypisania,
- porównań,
- logiczne,
- tekstowe.

Operatory arytmetyczne

Są to najprostsze operatory, które zna większość osób uczących się programowania, ponieważ używamy ich na co dzień do wykonywania działań arytmetycznych.

Siedem podstawowych operatorów arytmetycznych przedstawiono w poniższej tabeli:

Operator	Opis	Przykład	Wynik
+	Dodawanie	If x = 6 x + 4	10
-	Odejmowanie	If x = 6 x - 2	4
*	Mnożenie	If x = 6 x * 2	12
/	Dzielenie	If x = 6 x / 3	2
++	Inkrementacja (dodanie jedynki)	If x = 6 x++	7
--	Dekrementacja (odjęcie jedynki)	If x = 6 x--	5
%	Modulo (reszta z dzielenia)	7%2 10%4 12%6	1 2 0

Operatory przypisania

Operatory przypisania służą do przypisywania wartości. Operatory te wykorzystamy w szerszym zakresie w dalszej części tego rozdziału, kiedy zajmiemy się zastosowaniem zmiennych (x z przykładów w poprzednim punkcie jest przykładem zmiennej, do której przypisano wartość 6).

Jest sześć operatorów przypisania:

Operator	Przykład	Równoważne wyrażeniu
=	x = 6 x = y	x = 6 x = y
+=	x += y	x=x+y
-=	x -= y	x=x-y
*=	x *= y	x=x*y
/=	x /= y	x=x/y
%=	x %= y	x=x%y

> W powyższych przykładach x i y to zmienne. Zmiennym należy nadawać unikatowe nazwy (chyba że korzysta się z nich wielokrotnie). Dla potrzeb omówienia tego materiału wystarczą nam zmienne x i y!

Operatory porównań

Operatory porównań wykorzystujemy w dwóch celach — jak się przekonamy w rozdziale 7., „Struktury kodowania", są one bardzo przydatne.

Jest sześć operatorów porównań. Wynikiem działania każdego z nich jest wartość `true` lub `false` w zależności od tego, czy wynik porównywania jest prawdziwy, czy fałszywy.

Operator	Opis	Przykład
`==`	Równy	`5 == 8` (zwraca `false`) `8 == 8` (zwraca `true`) `"kot" == "pies"` (zwraca `false`) `"kot" == "kot"` (zwraca `true`)
`!=`	Różny	`5 != 8` (zwraca `true`) `8 != 8` (zwraca `false`)
`>`	Większy niż	`5 > 8` (zwraca `false`) `8 > 3` (zwraca `true`)
`<`	Mniejszy niż	`5 < 8` (zwraca `true`) `8 < 3` (zwraca `false`)
`>=`	Większy lub równy	`5 >= 8` (zwraca `false`) `8 >= 3` (zwraca `true`) `8>=8` (zwraca `true`)
`<=`	Mniejszy lub równy	`5 <= 8` (zwraca `true`) `8 <= 3` (zwraca `false`) `3 <= 3` (zwraca `true`)

Operatory logiczne

Operatory logiczne umożliwiają stosowanie wyrażeń logicznych w tworzonym kodzie. Można je łączyć z operatorami zaprezentowanymi wcześniej.

Są trzy operatory logiczne.

Operator	Opis	Przykład

`&&`	And	`x = 7`
		`y = 2`
		`(x < 12 && y > 1)`
		zwraca `true`
		Powyższe wyrażenie można odczytać w następujący sposób: „`x` jest mniejsze od 12 i jednocześnie `y` większe niż 1".
		`x = 7`
		`y=2`
		`(x < 12 && y < 1)`
		zwraca `false`

Operator	Opis	Przykład

`||`	Or	`x=6`
		`y=1`
		`(x==6 || y==5)`
		zwraca `true`
		Powyższe wyrażenie można odczytać w następujący sposób: „`x` jest równe 6 lub `y` jest równe 5”.
		`x=6`
		`y=1`
		`(x==5 || y==5)`
		zwraca `false`
		`x=6`
		`y=1`
		`(x==6 || y==1)`
		zwraca `true`

`!`	Not	`x=6`
		`y=3`
		`!(x==y)`
		zwraca `true`
		Powyższe wyrażenie można odczytać w następujący sposób: „`x` nie jest równe `y`”.
		`x=3+3`
		`y=4+2`
		`!(x==y)`
		zwraca `false`

Operatory znakowe

Operatory znakowe służą do wykonywania operacji na ciągach znaków. Zazwyczaj za pomocą ciągów znaków zapisujemy fragmenty tekstu. Oto dwa przykładowe ciągi znaków:

```
string1 = "Witaj, "
string2 = "świecie!"
```

Za pomocą operatorów znakowych można łączyć ze sobą ciągi znaków (taka operacja nazywa się *konkatenacją*).

```
string3 = string1 + string2
```

Po wykonaniu tej instrukcji w zmiennej `string3` będzie zapisany ciąg `Witaj, świecie!`.

Jednym z elementów sprawiających trudności osobom początkującym w wykonywaniu operacji na ciągach znaków są spacje. Dokładniej mówiąc, problemy stwarza odpowiednie stosowanie spacji pomiędzy ciągami znaków, tak aby ostatni wyraz był napisany poprawnie.

Na przykład gdyby w poprzednim przykładzie zmienne miały wartości:

```
string1 = "Witaj,"
string2 = "świecie!"
```

to po wykonaniu konkatenacji:

```
string3 = string1 + string2
```

zmienna `string3` zawierała by wartość `Witaj,świecie!`. Jednym z rozwiązań tego problemu może być wprowadzenie spacji podczas wykonywania konkatenacji:

```
string3 = string1 + " " + string2
```

Po wykonaniu takiej operacji zmienna `string3` będzie miała prawidłową wartość `Witaj, świecie!`.

Operacjom na ciągach znaków przyjrzymy się bardziej szczegółowo w kolejnych rozdziałach. Teraz jednak, kiedy znamy podstawy, o wiele łatwiej będzie nam zrozumieć szczegóły i będą one miały więcej sensu.

Podsumowanie

W tym rozdziale prześledziliśmy sposoby przechodzenia od pojęć do kodu. Przeanalizowaliśmy kilka systemów liczbowych, z którymi można się spotkać, a także sposób wykorzystania systemu dwójkowego, szesnastkowego i ósemkowego do reprezentowania znaków w kodzie ASCII. Pokrótce powiedzieliśmy także, dlaczego system dwójkowy jest ważny w technice komputerowej i stanowi podstawę niemal wszystkich działań wykonywanych w elektronice cyfrowej.

Omówiliśmy także operatory, które umożliwiają wykonywanie operacji na prostych danych (na tym właśnie polega przetwarzanie danych).

W następnym rozdziale omówimy narzędzia potrzebne do programowania.