

Podstawy C

Stałe, zmienne i instrukcja przypisania

W tym rozdziale postaramy się przybliżyć Czytelnikowi zmienne używane w programowaniu. Podobnie jak w matematyce zmienna to symbol, który może przyjmować różne wartości. Zmienna jest związana ściśle ze swoim typem określającym, jakie wartości może ona przyjąć tzw. typem zmiennej. Określa on, jakie wartości mogą być przechowywane przez zmienną. Do tej pory poznaliśmy typ znakowy *char* przechowujący jeden znak w kodzie ASCII. Literatura podaje wiele typów zmiennych oraz efektywnych konstrukcji przypisywania wartości zmiennym [32, 89, 103-105]. Przedstawmy najpopularniejsze typy danych

char	Jeden bajt: znak w kodzie ASCII albo liczba od -128 do 127
int	4 bajty: liczba całkowita z przedziału od -2 147 483 648 (-2^{31}) do 2 147 483 647 ($2^{31} - 1$)
double	8 bajtów: liczba zmiennoprzecinkowa z przybliżoną dokładnością 15 miejsc mantysy z zakresu od 1.7E-308 do 1.7E+308

W zapisie liczb zmiennoprzecinkowych używamy litery małej e albo dużej E do określenia cechy np. 1.2-10 zapisujemy 1.2e-10 używając kropki do oddzielenia części całkowitej od ułamkowej. Przedstawmy teraz następujące zadanie

Przykład 1

Napisać program czytający dwie liczby rzeczywiste podane z klawiatury i wypisujący wynik na ekran konsoli.

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    double a,b;
```

```
    printf("Podaj a:");
```

```
    scanf("%lf",&a);
```

```
    printf( "Podaj b:" );
```

```
    scanf("%lf",&b);
```

```
    printf("%g%+g=%g\n",a,b,a+b);
```

```
    _getch();
```

```
    return 0;
```

```
}
```

1)

2)

3)

4)

5)

6)

1) deklaracja zmiennych rzeczywistych a i b .

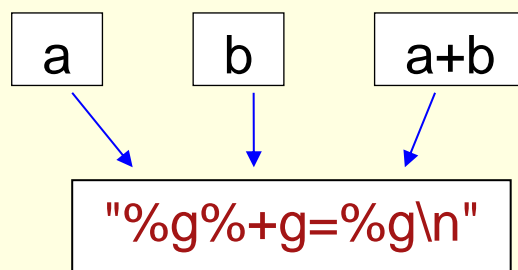
2) wypisanie napisu: Podaj a :

3) czytanie zmiennej a za pomocą funkcji `scanf()`. Funkcja ta ma dwa argumenty. Pierwszym argumentem jest pole tekstowe mówiące o sposobie przekształcenia strumienia wczytywanych znaków. Tutaj `%lf` oznacza przekształcenie wczytanych wartości do liczby typu *double*. Drugim argumentem jest wskaźnik, do której zmiennej ma nastąpić zapisanie wczytanej liczby. Tworzymy go za pomocą operatora `&`. Tak, więc `&a` oznacza, że należy wczytać liczbę do zmiennej a .

4) wypisanie napisu: Podaj b :

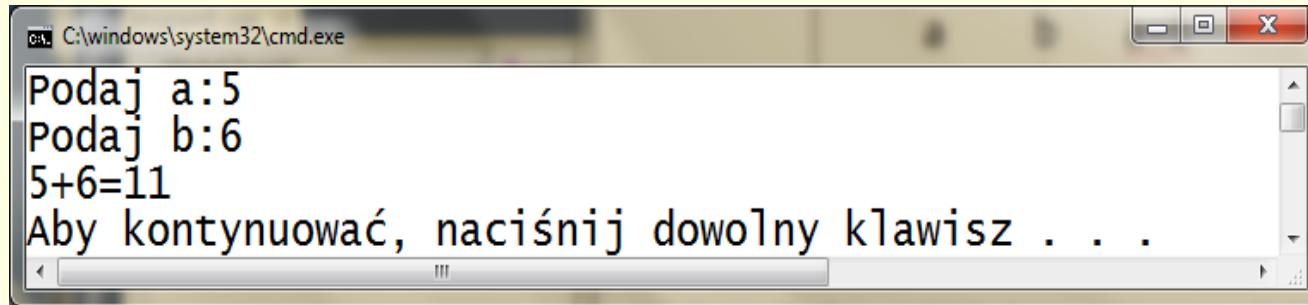
5) wczytanie zmiennej b za pomocą funkcji `scanf()`. Podobnie jak poprzednio istotny jest sposób przekształcenia strumienia dla liczby typu *double* `%lf` i czytanie pod adres `&b`.

6) wypisanie na ekran kolejno: przeczytanej liczby a ($\%g$), przeczytanej liczby b ze znakiem ($\%+g$), wypisanie znaku $=$ ($=$), wypisanie sumy liczb ($\%g$) i znaku nowej linii ($\backslash n$). Trzy argumenty a , b , $a+b$ funkcji `printf()` są odpowiednio przekształcane zgodnie z następującym formatem `"%g%+g=%g\backslash n"`, co pokazuje poniższa ilustracja.



Zobaczmy teraz jak będzie wyglądało okno konsoli w systemie Windows z programem wykonującym opisaną operację. Wpisując liczby np. 5 i 6 otrzymujemy wynik 11, co pokazuje ilustracja. Czytelnik zechce samodzielnie wykonać opisany przykład i sprawdzić jego działanie po skompilowaniu. Uwaga: W przypadku Visual Studio lepiej jest używać bezpiecznej funkcji `scanf_s()` zamiast `scanf()`.

Użycie Unixowej funkcji *scanf()* nie powoduje błędu programu, ale wykorzystany kompilator MVS daje ostrzeżenie, które nie ma większego znaczenia.



```
C:\windows\system32\cmd.exe
Pодаj a:5
Pодаj b:6
5+6=11
Aby kontynuować, naciśnij dowolny klawisz . . .
```

The image shows a screenshot of a Windows command prompt window. The title bar indicates the path 'C:\windows\system32\cmd.exe'. The window contains the following text: 'Pодаj a:5', 'Pодаj b:6', '5+6=11', and 'Aby kontynuować, naciśnij dowolny klawisz . . .'. The text is displayed in a monospaced font. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

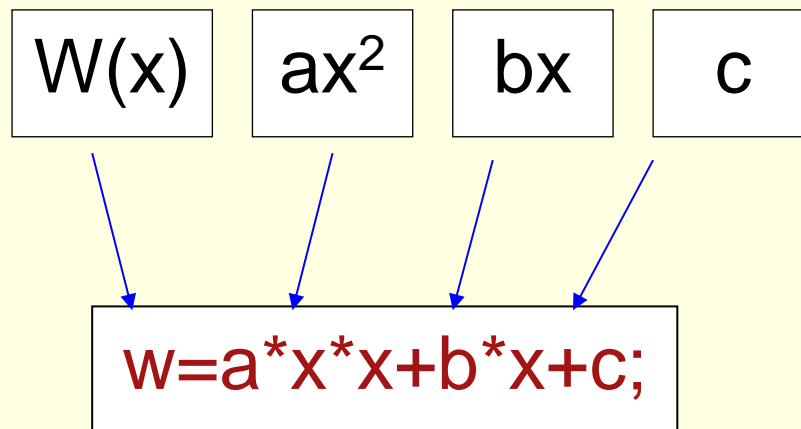
Operatory arytmetyczne

Każdy język programowania w pewien szczególny dla siebie sposób traktuje arytmetykę. W języku C/C++ programista pisząc program może użyć kilku operatorów arytmetycznych.

+	dodawanie
-	odejmowanie
*	mnożenie
/	dzielenie
%	modulo (reszta z dzielenia)

Operatorów arytmetycznych używamy w wielu wyrażeniach matematycznych. Przykładem opisującym omawiane zagadnienie niech będzie zadanie wyznaczenia wartości wielomianu . W zadaniu tym uznajemy, że mamy podane współczynniki a , b i c . Zatem problem polega jedynie na wyznaczeniu wartości $W(x)$. Zapiszmy najpierw omawiany wynik w postaci zrozumiałej dla komputera

Jak widać na ilustracji otrzymany wynik należy zapisać do nowej zmiennej, w tym przypadku w . Postawiony średnik na końcu wyrażenia informuje kompilator o zakończeniu poszczególnej instrukcji. Można zadać pytanie, dlaczego twórcy języka C w operacjach arytmetycznych użyli znaku równości zamiast np. strzałki (\leftarrow), tak jak to miało miejsc w innych językach programowania. W wyniku wykonanych badań częstości używania instrukcji podstawienia w stosunku do porównywania dwóch liczb, okazało się, że częściej używamy podstawienia niż porównania liczb ze sobą. Nie jest to jedyne odstępstwo od przyjętych powszechnie oznaczeń w matematyce. Tak naprawdę programista powinien pamiętać, że język programowania narzuca swoje własne oznaczenia. Szerzej zostało to opisane w literaturze [5, 13, 22-23, 25, 32].



Języki programowania przyjmują wiele zasad, jakie zostały wprowadzone w różnych działach matematyki. Przykładem mogą być zasady wykonywania obliczeń znane Czytelnikowi z algebry. Operatory mnożenia i dzielenia są wykonywane przed operatorami dodawania i odejmowania. Zatem jeżeli chcemy zmienić kolejność wykonywania operacji w programie musimy postawić odpowiednio nawiasy oznaczające . Nie jest tutaj celowe postawienie matematycznego znaku równości, gdyż doprowadziłibyśmy do kolizji oznaczeń, ponieważ dla kompilatora w takim przypadku nastąpiłoby podstawienie. Ostatni z operatorów modulo jest bardzo wygodny do stwierdzenia czy dana liczba jest podzielna przez drugą. Przyjmując np. $x\%2$ daje wynik 0, jeśli liczba x jest parzysta oraz 1 dla x nieparzystego. Zmiana charakteru wpisywanych wartości może znacząco wpływać na otrzymywane efekty, co pokazano w literaturze [23, 38, 40, 42, 49, 57].

Oznaczenia zmiennych w programie i kod ASCII

Zacznijmy od bardzo ważnej uwagi, która ułatwi Czytelnikowi programowanie i uczyni kody programów przejrzystszy.

Nazwy zmiennych powinny być skojarzone z obiektami, które reprezentują. Nie powinny być jednak za długie, aby nie utrudniały rozpoznawania i stosowania.

W różnych językach programowania stosuje się właściwe im zasady. Na przykład w języku FORTRAN przyjmowano domyślnie, że zmienne *i*, *j*, *k*, *l*, *m*, *n* są typu całkowitego, natomiast pozostałe zmienne typu rzeczywistego. Ponieważ języki programowania jak i cała wiedza z dziedziny IT podlega standaryzacji, również zmienne mają określone im zasady i schematy. Deklaracja zmiennych w klasycznym języku ANSI C (ang. American National Standards Institute) musiała się odbyć na początku programu. W późniejszych wersjach możliwa jest deklaracja wewnątrz programu.

W języku C jest możliwe przypisywanie wartości poprzez operatory jedno i wieloargumentowe. Dla arytmetycznych operatorów dwuargumentowych omówionych wcześniej +, -, *, /, % możemy stosować operator przypisania w następujący sposób:

`zmienna (operator) = wyrażenie;`
jest równoważne
`zmienna = zmienna (operator) wyrażenie;`

np.: `i=i+3`; możemy zapisać `i+=3`; jednocześnie instrukcja `x*= a-3`; odpowiada instrukcji `x=x*(a-3)`;

Instrukcje zwiększania i zmniejszania wartości zmiennych oznaczamy ++ lub --. Operator zwiększania ++ dodaje jeden, natomiast operator zmniejszania -- odejmuje 1. Mogą one mieć postać przedrostkową (przed zmienną: ++*i*) oraz przyrostkową (po zmiennej: *i*++).

W przypadku inkrementacji albo dekrementacji przedrostkowej zwiększana albo zmniejszana jest wartość zmiennej przed użyciem jej w wyrażeniu. Natomiast inkrementacja przyrostkowa powoduje zmianę wartości zmiennej dopiero po wykonaniu zadanych operacji.

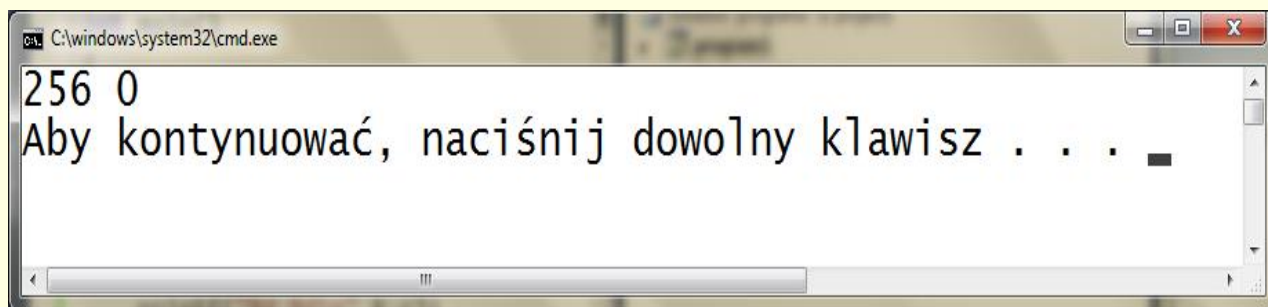
W wyrażeniu mogą występować różne typy zmiennych *char*, *int* czy *double*. W takim wypadku wszystkie zmienne i stałe zamieniane są do typu o najwyższej precyzji, a następnie wykonywane jest wyrażenie. Należy pamiętać, że typ *char* jest traktowany jak liczba ze znakiem i można na nim dokonywać operacji arytmetycznych. Czytelnik musi jednak pamiętać o możliwości uzyskania nadmiaru w danej zmiennej, ponieważ zakres takiej liczby jest możliwy od -128 do 127. Po wykonaniu operacji wynik jest rzutowany do typu zmiennej wynikowej przez obcięcie części ułamkowej albo do trzech najbardziej znaczących bajtów w przypadku rzutowania *int* na *char*, co pokazuje kolejny przykład.

Przykład 2

Napisać program drukujący liczbę typu *double* jako typ *int* oraz *char*.

<pre>#include<stdio.h> int main() { double a=256.88; int t; char c; t=a; c=a; printf("%d %d\n",t,c); _getch(); return 0; }</pre>	<pre>// deklaracja biblioteki // deklaracje zmiennych // operacje podstawienia // wydrukowanie zmiennych</pre>
--	---

Zobaczmy, jaki będzie efekt kompilacji przedstawionego kodu. Okno konsoli realizującej przedstawiony program zostało pokazane na ilustracji. Widzimy, że program pokazuje dwie wartości całkowite 256 oraz 0 ponieważ właśnie takiego typu jest zmienna, w której są przechowywane wartości do wydrukowania ich na ekranie.



```
C:\windows\system32\cmd.exe
256 0
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Przykład 3

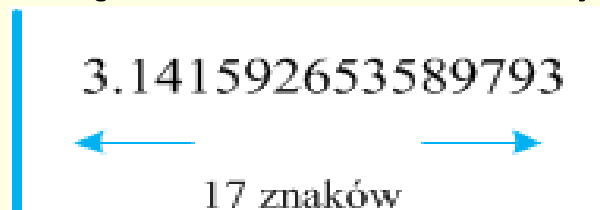
Napisać program obliczający π z dokładnością odpowiadającą reprezentacji liczby typu double.

W języku C/C++ argumentami wszystkich funkcji matematycznych $\sin(x)$, $\cos(x)$, $\tan(x)$ są kąty wyrażone w radianach. Ponieważ często używamy w rachunkach stopni, to do przeliczania ich na radiany potrzebna jest nam przybliżona wartość π . Możemy tutaj skorzystać z własności funkcji tangens, która dla 45 stopni przyjmuje wartość jeden. Obliczenie funkcji odwrotnej do funkcji $\tan(x)$ z wartości jeden da nam wartość 4π . Nasz program będzie wyglądał następująco:

<pre>#include<stdio.h> #include<math.h> int main() { double pi=4*atan(1.0); printf("%17.16g\n",pi); _getch(); return 0; }</pre>	<pre>\\ deklaracja bibliotek \\ deklaracja zmiennej \\ wydruk zmiennej</pre>
--	---

Uwaga: Aby móc dołączyć bibliotekę Matematyczną w Linux w przypadku kompilatora *gcc* wymagana jest kompilacja z linkowaniem biblioteki za pomocą polecenia: *gcc nazwa_pliku.c -lm*. W przypadku kompilatora *g++* oraz kompilatora Visual Studio wystarczy jedynie sama deklaracja *#include<math.h>*.

Zobaczmy jak wygląda deklaracja zmiennej π i wstawienie wartości początkowej π z dokładnością liczb typu double. Wydruk otrzymanego przybliżenia liczby π na siedemnastu znakach z dokładnością do szesnastu cyfr pokazuje poniższa ilustracja.



3.141592653589793

17 znaków

The diagram shows the value of pi, 3.141592653589793, enclosed in a light blue box. Below the box, two blue arrows point outwards from the center, with the text '17 znaków' (17 characters) written below them, indicating the precision of the value.

Jeżeli kompilujemy program pod systemem Linux domyślną nazwą programu jest *a.out*, jednocześnie program jest umieszczany w katalogu domyślnym, czyli tam gdzie użytkownik wywołał edytor. W przypadku, gdy chcemy dokonać kompilacji i otrzymać program wynikowy o zadanej nazwie używamy następującego polecenia:

```
gcc -o nazwa_wynikowa nazwa.c -lm
```

Należy tutaj zwrócić uwagę na kilka bardzo istotnych własności. Nazwa wynikowa programu nie musi zawierać rozszerzenia i nie może być zgodna z nazwą pliku źródłowego. Nazwanie pliku źródłowego i wynikowego tak samo doprowadzi do utraty pliku źródłowego naszego programu i nie otrzymalibyśmy programu wykonywalnego. Natomiast w systemie Windows otrzymamy program wykonywalny o rozszerzeniu *exe* i nazwą zgodną z nazwą programu źródłowego.

Kod ASCII

Aby zrozumieć operacje, jakie mają miejsce podczas interpretacji kodu programu należy zapoznać się z kodem ASCII (ang. American Standard Code for Information Interchange). Nazwę tę nosi 7 bitowy kod, który przyporządkowuje liczby z zakresu od 0 do 127 literom alfabetu angielskiego, cyfrom, znakom interpunkcyjnym i wszystkim innym symbolom oraz poleceniom podstawowym. Zapis informacji w kodzie ASCII został przedstawiony w poniższych tabelach. Spróbujemy odszyfrować zapis podany w tabelach ASCII. Czytelnik zechce sprawdzić, że np. małej literze 'g' odpowiada w zapisie binarnym ciąg 0110 0111, co w zapisie dziesiętnym oznacza numer porządkowy 103. Jednocześnie np. znakowi ';' binarnie odpowiada ciąg 0011 1011, co rozumiemy jako numer porządkowy 59. W ten sposób można zakodować wszystkie podstawowe znaki i operacje, które jest w stanie obsługiwać komputer.

Cały zapis kodu ASCII jest podzielony na mniejsze podzbiory. Czytelnik może sprawdzić, że litery, cyfry czy inne znaki drukowane tworzące zbiór znaków ASCII znajdują się w przedziale porządkowym o kodach 32-126 gdzie mieści się 95 znaków. Pozostałe 33 kody od 0 do 31 oraz 127 nazywamy tzw. kodami sterującymi służącymi do sterowania urządzeniami zdolnymi do komunikowania się jak drukarka itp. Czytelnik zapewne zauważył, że w przedstawionym kodzie ASCII nie znajdują się znaki należące do poszczególnych języków. Jest jednak możliwość odwzorowania tych znaków w odpowiednich rozszerzeniach. Możliwość wprowadzenia nowych znaków wiąże się z budową kodu ASCII. Jest on 7 bitowy, natomiast komputery operują przeważnie na 8 i więcej bajtach. Zatem daje to możliwość powiększenia kodu. Obecnie można się spotkać z różnymi rozszerzeniami pracującymi na ósmym bicie. Są to np. norma ISO 8859, rozszerzenia firm IBM czy Microsoft lub inne tabele kodów potocznie nazywane stronami kodowymi różnych systemów operacyjnych. Obecnie jednak największe znaczenie zyskuje stworzone rozszerzenie UTF-8 i jego pochodne, które tak często stosowane są w różnego rodzaju edytorach tekstowych [22, 32, 40, 42, 75, 88]. Należy jednak pamiętać, iż w tym przypadku dodatkowe znaki kodowane są na 2 i więcej bajtach.

Tabela poleceń podstawowych odwzorowanych w kodzie ASCII.

Bin	Dec	Hex	Znak	Skrót
0000 0000	0	00	Null	NUL
0000 0001	1	01	Start Of Heading	SOH
0000 0010	2	02	Start of Text	STX
0000 0011	3	03	End of Text	ETX
0000 0100	4	04	End of Transmission	EOT
0000 0101	5	05	Enquiry	ENQ
0000 0110	6	06	Acknowledge	ACK
0000 0111	7	07	Bell	BEL
0000 1000	8	08	Backspace	BS
0000 1001	9	09	Horizontal Tab	HT
0000 1010	10	0A	Line Feed	LF
0000 1011	11	0B	Vertical Tab	VT
0000 1100	12	0C	Form Feed	FF
0000 1101	13	0D	Carriage Return	CR
0000 1110	14	0E	Shift Out	SO
0000 1111	15	0F	Shift In	SI
0001 0000	16	10	Data Link Escape	DLE

Tabela poleceń podstawowych odwzorowanych w kodzie ASCII.

Bin	Dec	Hex	Znak	Skrót
0001 0001	17	11	Device Control 1 (XON)	DC1
0001 0010	18	12	Device Control 2	DC2
0001 0011	19	13	Device Control 3 (XOFF)	DC3
0001 0100	20	14	Device Control 4	DC4
0001 0101	21	15	Negative Acknowledge	NAK
0001 0110	22	16	Synchronous Idle	SYN
0001 0111	23	17	End of Transmission Block	ETB
0001 1000	24	18	Cancel	CAN
0001 1001	25	19	End of Medium	EM
0001 1010	26	1A	Substitute	SUB
0001 1011	27	1B	Escape	ESC
0001 1100	28	1C	File Separator	FS
0001 1101	29	1D	Group Separator	GS
0001 1110	30	1E	Record Separator	RS
0001 1111	31	1F	Unit Separator	US

Tabela cyfr i znaków specjalnych odwzorowanych w kodzie ASCII.

Bin	Dec	Hex	Znak
0010 0000	32	20	Spacja
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	\$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/

Tabela cyfr i znaków specjalnych odwzorowanych w kodzie ASCII.

Bin	Dec	Hex	Znak
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

Tabela wielkich liter alfabetu angielskiego odwzorowanych w kodzie ASCII.

Bin	Dec	Hex	Znak
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O

Tabela wielkich liter alfabetu angielskiego odwzorowanych w kodzie ASCII.

Bin	Dec	Hex	Znak
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D]
0101 1110	94	5E	^
0101 1111	95	5F	_

Tabela małych liter alfabetu angielskiego odwzorowanych w kodzie ASCII.

Bin	Dec	Hex	Znak	Skrót
0110 0000	96	60	`	
0110 0001	97	61	a	
0110 0010	98	62	b	
0110 0011	99	63	c	
0110 0100	100	64	d	
0110 0101	101	65	e	
0110 0110	102	66	f	
0110 0111	103	67	g	
0110 1000	104	68	h	
0110 1001	105	69	i	
0110 1010	106	6A	j	
0110 1011	107	6B	k	
0110 1100	108	6C	l	
0110 1101	109	6D	m	
0110 1110	110	6E	n	
0110 1111	111	6F	o	

Tabela małych liter alfabetu angielskiego odwzorowanych w kodzie ASCII.

Bin	Dec	Hex	Znak	Skrót
0111 0000	112	70	p	
0111 0001	113	71	q	
0111 0010	114	72	r	
0111 0011	115	73	s	
0111 0100	116	74	t	
0111 0101	117	75	u	
0111 0110	118	76	v	
0111 0111	119	77	w	
0111 1000	120	78	x	
0111 1001	121	79	y	
0111 1010	122	7A	z	
0111 1011	123	7B	{	
0111 1100	124	7C		
0111 1101	125	7D	}	
0111 1110	126	7E	~	
0111 1111	127	7F	Delete	DEL

Schematy blokowe

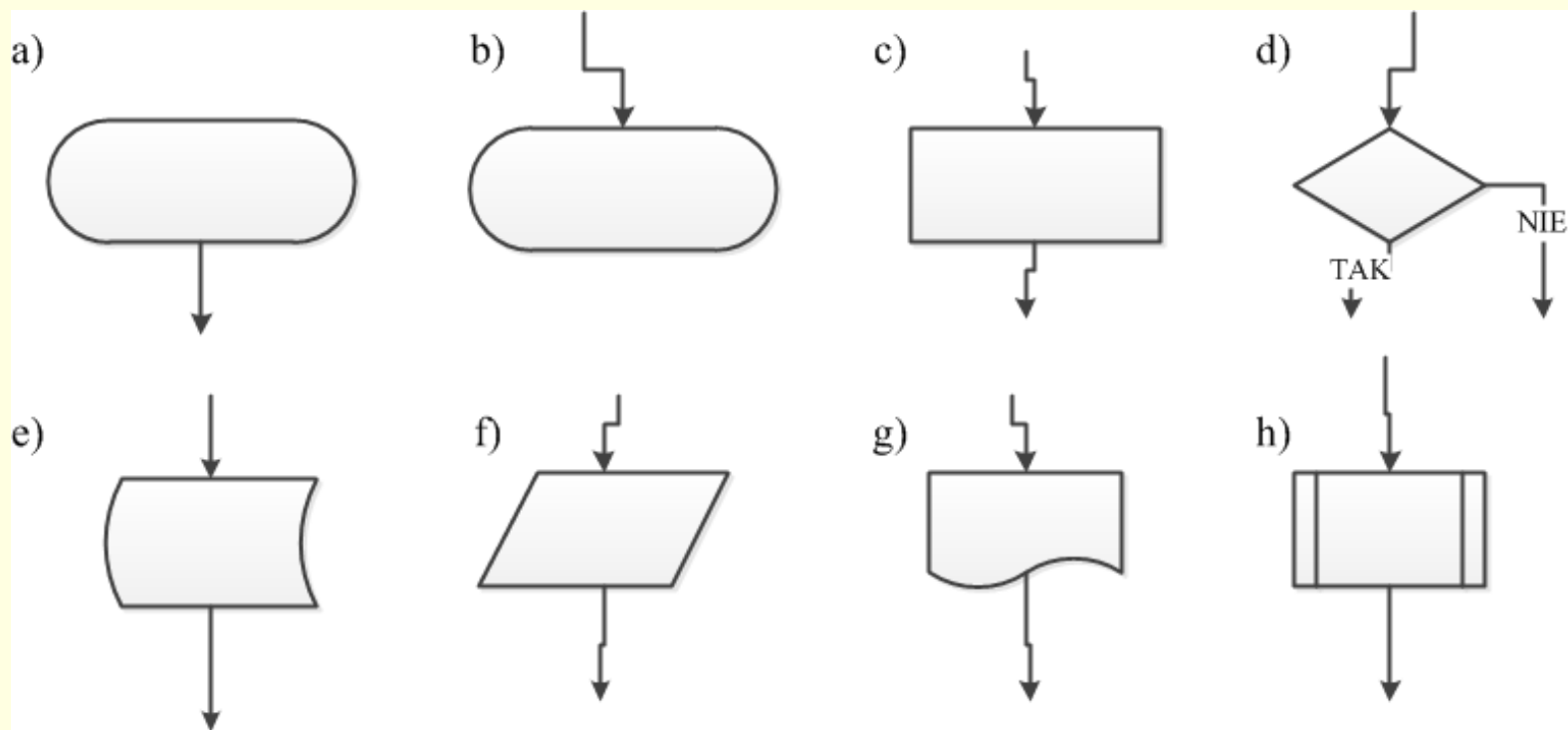
Schemat blokowy (ang. block diagram, flowchart) jest to proste i efektywne narzędzie programisty, którego zadaniem jest zaprezentowanie kolejnych etapów i czynności, jakie należy wykonać w celu rozwiązania zadanego problemu. Zadanie prezentacji graficznej algorytmu polega na opisie w postaci odpowiednich figur geometrycznych czynności przyporządkowanych kolejnym etapom w trakcie realizacji procedury. Figury te są połączone odpowiednimi liniami zgodnie z kolejnością wykonywania procedur. Schematy blokowe dzięki swej budowie ułatwiają programowanie, ponieważ pozwalają na prostą zamianę bloków na instrukcje kodu – programu komputerowego. Postaramy się przedstawić Czytelnikowi najważniejsze zasady i elementy budowania schematów blokowych, które w szerszym opisie można znaleźć w literaturze [52, 58]. Wśród nich wyróżnia się najważniejsze elementy:

Strzałka – wskazuje jednoznacznie powiązania pomiędzy procedurami i kierunek wykonywania operacji,

Operand – prostokąt, w którym zamieszczane są wszystkie operacje. Jednakże nie wpisujemy w takim elemencie graficznym instrukcji wyboru,

Predykat – romb, element graficzny, do którego wpisujemy wyłącznie instrukcje wyboru,

Etykieta – owal służący do oznaczania początku lub końca sekwencji schematu.



Umowne kształty elementów schematu blokowego przedstawia powyższy rysunek.

- a) Blok graniczny oznaczający początek działania np. blok startu programu,
- b) Blok graniczny oznaczający koniec operacji, przerwanie lub wstrzymanie wykonywania działania programu,
- c) Blok obliczeniowy oznaczający wykonanie operacji zmieniającej wartości, postać lub miejsce zapisu danych, np. $k := c + d$,
- d) Blok decyzyjny lub warunkowy przedstawiający wybór jednego z dwóch wariantów w programie na podstawie wpisanego warunku, np. $a > b$,
- e) Blok oznaczający wczytanie danych zewnętrznych, na których pracować będzie program,
- f) Blok oznaczający pobranie danych do wykonywanej procedury i przyporządkowania ich zmiennym,
- g) Blok odpowiedzi programu, którą komputer wydrukuje na zadeklarowane wyjście standardowe,
- h) Blok podprocesu oznaczający wywołanie w programie procedury pomocniczej.