

Programowanie imperatywne

dr inż. Piotr Szwed
Katedra Informatyki Stosowanej
C2, pok. 403

e-mail: pszwed@agh.edu.pl

<http://home.agh.edu.pl/~pszwed/>

Aktualizacja: 05.03.2021

2. Podstawy składni języka C

Składnia i semantyka

Wikipedia:

Język programowania składa się ze zbiorów reguł syntaktycznych oraz semantyki, które opisują, jak należy budować poprawne wyrażenia oraz jak komputer ma je rozumieć

- **Składnia** opisuje:
 - rodzaje dostępnych symboli
 - zasady, według których symbole mogą być łączone w większe struktury.
- **Semantyka** języka programowania definiuje precyzyjnie znaczenie poszczególnych symboli oraz ich funkcję w programie. Także określa typy danych i wykonywane na nich operacje.

Przykład 2

```
1 a=b=1;
2 n=100;
3 for i in range(1,n):
4     c=a+b
5     print(f"{i} {c:6}",end='\t')
6     if i%8==0:
7         print()
8     a=b
9     b=c
```

Ten sam algorytm w języku Python 3.x

- Nieco inna składnia
- Inna semantyka: liczby całkowite powiększają się w nieskończoność

1	2	2	3	3	5	4	8	5	13	6	21	7	34	8	55
9	89	10	144	11	233	12	377	13	610	14	987	15	1597	16	2584
17	4181	18	6765	19	10946	20	17711	21	28657	22	46368	23	75025	24	121393
25	196418	26	317811	27	514229	28	832040	29	1346269	30	2178309	31	3524578	32	5702887
33	9227465	34	14930352	35	24157817	36	39088169	37	63245986	38	102334155	39	165580141	40	267914296
41	433494437	42	701408733	43	1134903170	44	1836311903	45	2971215073	46	4807526976	47	7778742049	48	12586269025
49	20365011074	50	32951280099	51	53316291173	52	86267571272	53	139583862445	54	225851433717	55	365435296162	56	591286729879
57	956722026041	58	1548008755920	59	2504730781961	60	4052739537881	61	6557470319842	62	10610209857723	63	17167		
65	44945570212853	66	72723460248141	67	117669030460994	68	190392490709135	69	308061521170129	70	498454011879264				
73	2111485077978050	74	3416454622906707	75	5527939700884757	76	8944394323791464	77	14472334024676221	78	23416728348467685				
81	99194853094755497	82	160500643816367088	83	259695496911122585	84	420196140727489673	85	6798916376386122258	86	1100087778366101931				
89	4660046610375530309	90	7540113804746346429	91	12200160415121876738	92	19740274219868223167	93	31940434634990099905	94	51680708854858323072				
97	218922995834555169026	98	354224848179261915075	99	573147844013817084101										

Ostatnie wyrazy w C

97 -889489150

98 -980107325

99 -1869596475

Python

97 218922995834555169026

98 354224848179261915075

99 573147844013817084101

Składnia języka C

- Kompilator (preprocesor) analizuje plik zawierający źródło programu i wydziela z niego **symbole** (ang. *tokens*).
- Symbole mogą (ale nie muszą) być przedzielone **białymi znakami**.
- W wyniku translacji dokonywanych przez preprocesor **komentarz** jest zastępowany białymi znakami.

Białe znaki

- spacja
- znak tabulacji
- znak nowej linii
- znak powrotu karetki
- znak nowej strony

Komentarze

- Blokowe
 - `/*` początek komentarza
 - `*/` koniec komentarza
- Liniowe
 - `//` komentarz do końca wiersza

```
/* komentarz */
```

```
int main ( )  
{  
    /* komentarz  
    obejmujący  
    kilka wierszy */  
    printf("Hello world\n") ; // komentarz  
    return 0;  
}
```


Symbole

Rodzaje symboli

- słowa kluczowe (ang. *keywords*)
- identyfikatory (ang. *identifiers*)
- stałe (ang. *constants*)
- łańcuchy znakowe (ang. *string-literals*)
- operatory
- znaki interpunkcyjne: (ang. *punctators*)

Słowa kluczowe - 1

Słowa kluczowe mają specjalne znaczenie dla kompilatora. Są to nazwy **instrukcji**, operatorów, modyfikatorów lub **typów danych**.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Słowa kluczowe - 2

Dodatkowo w C++ występują:

- bool true false
- try catch throw
- new delete
- template
- namespace, using
- static_cast, dynamic_cast
- inne zaznaczane w edytorze po ustawieniu opcji *syntax highlighting*

Słów kluczowych nie wolno używać jako identyfikatorów (nazw zmiennych, typów danych lub funkcji).

Identyfikatory

- Są to nazwy, które nadawane są zmiennym, stałym, nazwom funkcji oraz typom danych.
- Identyfikator jednoznacznie określa opisywany obiekt.
- Użycie tego samego identyfikatora do zadeklarowania lub zdefiniowania innego obiektu traktowane jest jako błąd.

Identyfikatory - składnia

identyfikator :

nondigit

identyfikator nondigit

identyfikator digit

nondigit : one of

_ 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

digit : one of

0 1 2 3 4 5 6 7 8 9

Przykłady: z y z abc _ Ala _x z1 x23

Nie jest identyfikatorem: 43A

Identyfikatory - przykłady

```
#include <stdio.h>

#define MAX_SIZE 23

int main()
{
    int a=7;
    double x=1.07;
    char c='A';
    printf("%d\n",a) ;
}
```

- `main`, `printf` – identyfikatory funkcji
- `a`, `x`, `c` – identyfikatory zmiennych
- `MAX_SIZE` – symbol preprocesora

Identyfikatory - style

- Przykłady stylów tworzenia identyfikatorów

`fileLength`

`FileLength`

`file_length`

`_file_length`

- Uwaga: należy unikać nadawania zmiennym i funkcjom nazw pisanych dużymi literami. Zwyczajowo są to nazwy stałych preprocesora.

```
#define ARRAY_LENGTH 234903
```

Identyfiaktory - standardy

- Kompilator języka C/C++ rozróżnia duże/małe litery.
- Minimalna liczba znaczących znaków jest w standardzie określona na 31, ale niektóre kompilatory dopuszczają znacznie więcej (Microsoft 247)
- Linker może nie odróżniać wielkości liter i ograniczać liczbę znaków znaczących w identyfikatorze (w standardzie ANSI 6 znaków)
- Linker może nie rozróżniać małych/dużych liter.

Stałe

Terminem stała określa się trzy typy symboli:

- Literały (ang. *literals*) – definiujące jawnie stałe:
100 12 0x7 2.17 „Ala ma kota”
- Stałe preprocesora (definiowane)
define SIZE 100
- Deklarowane stałe (niemodyfikowalne zmienne)
const int **size** = 100;

Stałe (literały)

- Literały są to wartości liczbowe, znaki lub łańcuchy znakowe. Są one używane do:
 - nadawania wartości zmiennym `x = 7;`
 - porównywania `if (x==12) {...}`
 - mogą być przekazywane jako argumenty przy wywołaniach funkcji `printf("Hello")`
 - nie mogą wystąpić po lewej stronie operatora przypisania `7 = x;`
- Literały w wyniku kompilacji umieszczane są w pamięci programu.
- Ta sama wartość stałej może być w kodzie źródłowym reprezentowana w różny sposób, np.: `32` i `' '` jest tą samą wartością.

Stałe zmiennoprzecinkowe 1

- Mogą zawierać część ułamkową (po kropce, opcjonalnie)
- Mogą zawierać część wykładniczą (po literze **e E**)
- Mogą zawierać przyrostek określający typ stałej
f F float
L l long double
- Przykłady:

15.75

1.575E1 /* = 15.75 */

1575e-2 /* = 15.75 */

2.5e-3 /* = 0.0025 */

25E-4 /* = 0.0025 */

1.3L /* long double */

100.F /* float */

Stałe zmiennoprzecinkowe 2

- Stałe zmiennoprzecinkowe są reprezentowane, jako wartości zmiennoprzecinkowe typu double (8 bajtów) lub float (4 bajty)
- Liczby zmiennoprzecinkowe można przedstawić, jako $s \cdot m \cdot 2^e$, gdzie s – znak, $m \in [1,2)$ – mantysa, e – część wykładnicza (cecha).

typ	znak	mantysa	cecha
double	1	52	11
float	1	23	8

- Reprezentacja zmiennoprzecinkowa może być niejednoznaczna (konieczna normalizacja).
- Reprezentowane są także wartości, które nie są liczbami (NaN) – np.: wynik dzielenia przez 0.

Zakresy wartości

```
#include <float.h>
```

```
int main() {  
    double tab[]={-2.1,-0.4,-5,-4.11,-34,-11.2};  
    double max = DBL_MIN;  
    for(int i=0;i<sizeof(tab)/sizeof(tab[0]);i++)  
        if(max<tab[i])  
            max=tab[i];  
    printf("Dlaczego min = %.1e a nie %.1f?\n",max,tab[1]);  
    printf("DBL_MIN = %e DBL_MAX = %e \n",DBL_MIN,DBL_MAX);  
}
```

Stałe całkowite

Stałe całkowite mogą być zapisane:

- dziesiętkowo (decymalnie): liczby o podstawie 10
- ósemkowo (oktalnie) : liczby o podstawie 8
- szesnastkowo (heksadecymalnie) : liczby o podstawie 16
- Mogą zawierać przyrostek określający typ stałej
 - u U** unsigned
 - l L** long

Stałe dziesiętne

Składnia

decimal-constant :

nonzero-digit

decimal-constant digit

nonzero-digit : one of

1 2 3 4 5 6 7 8 9

Przykłady

21

134

23679

Nie ma stałych będących liczbami ujemnymi, np.:

x = **-100**;

to to samo co

x = **-** /* jednoargumentowy operator - */ **100**;

Stałe ósemkowe

Składnia

octal-constant :

0

octal-constant octal-digit

octal-digit : one of

0 1 2 3 4 5 6 7

Przykłady

012 /* 1*8+2 */

0204 /* 2*64 + 0*8 + 4 */

07663 /* 7*8^3 + 6*8^2+6*8+3 */

- Czyli pisząc `int x=0;` używamy stałej ósemkowej.
- Poprawne jest `int x=00000;`
- Nie jest poprawne `x=08;`
- Wartość `x=0204;` jest dość nieoczekiwana (132)

Stałe szesnastkowe

Składnia

hexadecimal-constant:

0x hexadecimal-digit

0X hexadecimal-digit

hexadecimal-constant hexadecimal-digit

hexadecimal-digit : one of

0 1 2 3 4 5 6 7 8 9

a b c d e f A B C D E F

Przykłady

- `0xa` **lub** `0xA` `/* 10 */`
- `0x84` `/* 8*16 + 4 */`
- `0x7dB3` **lub** `0X7DB3`
`/* 7*16^3 + 13*16^2 + 11*16+3 */`

Stałe znakowe 1

- Stałe znakowe wprowadza się przez umieszczenie znaku wewnątrz pojedynczego cudzysłowu (' '), np.: ' ', 'a', '7'
- W ten sposób można wprowadzić jedynie znaki dostępne bezpośrednio w edytorze. Aby wprowadzić inne znaki, np.: przejścia do nowej linii, znak apostrofu stosuje się ciągi specjalne tzw. *sekwencje escape*.
- *Sekwencja escape* – ciąg znaków rozpoczynający się od znaku specjalnego \ definiujący pojedynczy znak.
- Podstawowe znaki *escape* to :

\n	\r	\t	\\	\'	\"
\?	\b	\f	\v		

Stałe znakowe 2

- Znaki można wprowadzać także liczbowo

\octal-number \ooo

\hexadecimal-number \xhhh

- Przykład

*'\n' '\012' '\xa' *

- Typy danych

Stałe znakowe są typu `int` (nie `char`).

- Podczas kompilacji stałym znakowym przypisywane są wartości zgodne z tabelą ASCII, czyli instrukcje

`x = 'A';`

`x = 65;`

są identycznie tłumaczone.

Stable znakowe 3

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Stałe znakowe 4

Przykład - wartość cyfry liczby szesnastkowej
(0-9, A-F lub a-f)

```
#include <stdio.h>

int hexDigitValue(int digit) {
    if(digit>='0' && digit<='9') return digit-'0';
    if(digit>='A' && digit<='F') return digit-'A'+10;
    if(digit>='a' && digit<='f') return digit-'a'+10;
    return -1;
}

int main(int argc, char** argv) {
    int c='b'; // wypróbuj różne wartości
    printf("%d", hexDigitValue(c));
    return 0;
}
```

Łańcuchy znakowe 1

- Łańcuchy znakowe (ang. string literals) to ciągi znaków o skończonej długości n .
W praktyce $n = 500-2048$ (ANSI – 509)
- Specyfikuje się je przez umieszczenie tekstów w znakach cudzysłowu (" ")
- Wewnątrz łańcucha mogą pojawić się dowolne znaki lub ciągi specjalne (*escape sequence*).

Łańcuchy znakowe 2

Typy danych

- Łańcuchy znakowe są typu `char[]` (tablica znaków). Oznacza to, że po translacji stałe łańcuchowe są reprezentowane w wygenerowanym kodzie maszynowym (w sekcji danych) jako ciągła sekwencja znaków
- Sekwencja ta jest zakończona znakiem specjalnym `'\0'` oznaczającym koniec łańcucha. Znak ten jest dodawany automatycznie podczas kompilacji.

"Kraków"



Łańcuchy znakowe 3

W 1999 roku wprowadzono nowy, rozszerzający format stałych łańcuchowych. Typem danych jest `wchar_t` (zwykle dwa bajty)

`L"Kraków"`

K	\0	r	\0	a	\0	k	\0	\xf3	\0	w	\0	\0	\0
---	----	---	----	---	----	---	----	------	----	---	----	----	----

Łańcuchy znakowe 3

Łączenie (konkatenacja) łańcuchów znakowych

- Długość stałych tekstowych może przekraczać maksymalną liczbę znaków, którą można wprowadzić w jednym wierszu za pomocą typowego edytora (255 znaków).
- Preprocesor języka C/C++ może sklejać ze sobą stałe. Maksymalna długość sklejonego tekstu nie może przekraczać limitów narzuconych przez kompilator (509-2048 znaków).

-

Łańcuchy znakowe 4

```
#include <stdio.h>

int main()
{
    char*string1="To jest" " podzielony tekst \n";
    char*string2= "To je\
st tekst w kilku wierszach\n";
    printf(string1);
    printf(string2);
    return 0;
}
```

```
>To jest podzielony tekst
To jest tekst w kilku wierszach
>
```

Czy tekst jest identyfikatorem?

```
int is_identifier(const char tab[]){
    //negacja (tab[0]!='_' || (tab[0]>='a'&&tab[0]<='z') || (tab[0]>='A'&& tab[0]<='Z'))
    if( tab[0]!='_' &&
        !(tab[0]>='a'&&tab[0]<='z') &&
        !(tab[0]>='A'&& tab[0]<='Z'))return 0;
    for(int i=1;tab[i];i++){
        if( tab[i]!='_' &&
            !(tab[i]>='a'&&tab[i]<='z') &&
            !(tab[i]>='A' && tab[i]<='Z') &&
            !(tab[i]>='0' && tab[i]<='9'))return 0;
    }
    return 1;
}

int main(){
    char*tab[]={ "ala_ma_kota", "_", "_1", "123", "a<b" };
    for(int i=0;i<sizeof(tab)/sizeof(tab[0]);i++) {
        printf("%s %sjest identyfikatorem\n", tab[i],
            is_identifier(tab[i]) ? "" : "nie ");
    }
}
```

```
ala_ma_kota jest identyfikatorem
_ jest identyfikatorem
_1 jest identyfikatorem
123 nie jest identyfikatorem
a<b nie jest identyfikatorem
```

Inna wersja

```
#include <ctype.h>

int is_identifier(const char tab[]){
    if( tab[0]!='_' && !isalpha(tab[0]) )
        return 0;
    for(int i=1;tab[i];i++){
        if( tab[i]!='_' && !isalnum(tab[i]) )
            return 0;
    }
    return 1;
}
```

Znaki interpunkcyjne i znaki specjalne

Znaki te służą do nadania struktury programom, mają zastosowanie przy definiowaniu typów oraz umożliwiają sterowanie procesem kompilacji.

Należą do nich:

[] () { } * , : = ; ... #

```
#include<stdio.h>
void main()
{
    int d[7];    // definicja tablicy
    int i=4;    // definicja z inicjalizacją
    if(i>0){    // instrukcja warunkowa
        printf("i>0");
    }
    return 0;
}
```

Operatory

Operatory służą do definiowania wyrażeń. Wyrażenia składają się z argumentów połączonych operatorami.

Mogą one:

- służyć do obliczania wartości;
- identyfikować obiekt lub funkcję;
- modyfikować argumenty.

```
int a=3,b=5;
int d[7];
printf("%d",a+b);
printf("%d",sizeof(d));
a++;
printf(a<=b?"a<=b":"a>b");
d[0]=a+++b; // (a++) + b
```

Struktura kodu 1

- Omówione zostały podstawowe symbole, z których składa się program w języku C: słowa kluczowe, identyfikatory, stałe, łańcuchy znakowe, operatory, znaki interpunkcyjne.
- W kodzie źródłowym programu te symbole powinny wystąpić w określonej kolejności zdefiniowanej przez:
 - składnię języka
 - zwyczaje (konwencje programistyczne)
- Składnia języka ściśle określa wzajemne położenie symboli, w tym znaków interpunkcyjnych: nawiasów, średników itd...
- Język C ogranicza także kolejność wystąpienia pewnych konstrukcji językowych

Struktura kodu 2

```
double mod(double x, double y)
{
    double z;

    y = sqrt(x * x + y * y);
    return z;
}
```

```
double mod(double x, double y)
{double z; y = sqrt(x*x +
y*y);return z;}
```

- Pary nawiasów (), { }, [] muszą do siebie pasować;
- Każda deklaracja jest zakończona średnikiem.
- Każda instrukcja poza instrukcją blokową jest zakończona średnikiem.
- Białe znaki obok znaków interpunkcyjnych i operatorów są pomijalne.

Przykład – czego brakuje?

```
int funkcja
int x
{
    printf("%d",x);
}
```

```
int funkcja (
int x)
{
    printf("%d",x);
}
```

Możliwe na poziomie pliku

```
int funkcja;
int x()
{
    printf("%d",x);
}
```

Możliwe na poziomie pliku

```
int funkcja;
int x;
{
    printf("%d",x);
}
```

Możliwe wewnątrz funkcji

- Znaki interpunkcyjne pomagają zorientować się w intencjach programisty.
- Definicje preprocesora na ogół nie podlegają analizie składni

```
#define BAD_CONSTANT {if>A break B ))while() return}
```

Typowa struktura kodu źródłowego

```
#include <stdio.h>
#include <stdlib.h>
#define LEN 100
```

```
double a;
int x;
```

```
void funkcja()
{
}

double potega (double x)
{
    return x*x;
}

int main(int argc, char** argv)
{
    return 0;
}
```

Dyrektywy

Deklaracje zmiennych
globalnych

Definicje funkcji

Struktura kodu wewnątrz funkcji

```
int main(int argc, char** argv)
{
    int i=7;
    int k=0;

    if(k<i)k=i;
    printf("wiekszy: %d", k);
    return 0;
}
```

Deklaracje zmiennych
lokalnych (wewnątrz funkcji)

Instrukcje

Konwencje programistyczne 1

Zazwyczaj kod modułu to następujące po sobie dyrektywy, deklaracje zmiennych globalnych i definicje funkcji (co nie znaczy, że umieszczone w innym porządku nie zostaną skompilowane).

- Wszystkie elementy są opcjonalne
- Dyrektywy (`#include`) następują zazwyczaj przed deklaracjami, zmiennych bo w plikach nagłówkowych mogą się znajdować definicje typów oraz tzw. prototypy funkcji
- Zmienne globalne deklaruje się przed funkcjami, bo zapewne w środku będą występowały odwołania do nich.

Konwencje programistyczne 2

- Formatując kod programu stosujemy wcięcia poprawiające czytelność.
- Zazwyczaj zagłębiamy się o jeden poziom w prawo po każdym otwierającym nawiasie klamrowym { oraz wracamy o jeden poziom w lewo przed każdym nawiasie zamykającym }.

```
#include <limits.h>
int foo(int n)
{
    → int i, k=1;

    for (i=n; i>1; i--) {
        → if ((double)k*i>INT_MAX) {
            → return -1;
            ←
        }
        ← k=k*i;
    }
    ← return k;
}
```

Ta funkcja oblicza silnię $n!$

Konwencje programistyczne 3

- http://en.wikipedia.org/wiki/Indent_style

K & R

```
int main(int argc, char *argv[])
{
    ...
    while (x == y) {
        something();
        somethingelse();

        if (some_error) {
            /* the curly braces around this code
               block could be omitted */
            do_correct();
        } else
            continue_as_usual();
    }

    finalthing();
    ...
}
```

Konwencje programistyczne 3

GNU

```
static char *
concat (char *s1, char *s2)
{
    while (x == y)
        {
            something ();
            somethingelse ();
        }
    finalthing ();
}
```

Horstmann

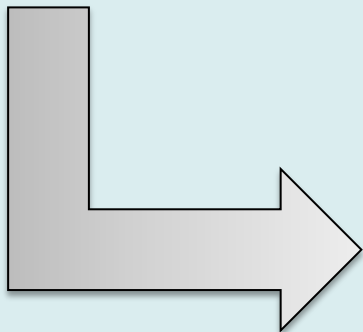
```
while (x == y)
{    something();
    somethingelse();
    //...
    if (x < 0)
    {    printf("Negative");
        negative(x);
    }
    else
    {    printf("Non-negative");
        nonnegative(x);
    }
}
finalthing();
```

Konwencje programistyczne 4

- C/C++ Beautifier – narzędzie do zmiany formatowania (często konfigurowalne)
- Wbudowane w popularne IDE

```
int main(int argc, char** argv) {int i =
7;int k = 0;
if(k < i){k = i; printf("wiekszy: %d", k);
}else
printf("mniejszy: %d", k);
return 0;}
```

```
int main(int argc, char** argv) {
int i = 7;
int k = 0;
if (k < i) {
k = i;
printf("wiekszy: %d", k);
} else printf("mniejszy: %d", k);
return 0;
}
```



Netbeans: Alt-Shift-F
Clion: Ctrl+Alt+L

Co należy zapamiętać?

- Rodzaje symboli języka
- Składnia identyfikatorów
- Postaci literałów
 - całkowitoliczbowych,
 - zmiennoprzecinkowych,
 - znakowych
 - łańcuchowych
- Ogólna struktura kodu
- Zwyczaje dotyczące formatowania

Różne linki

- <http://www.ericgiguere.com/articles/ansi-c-summary.html>
- <http://stackoverflow.com/questions/11965402/where-the-c-literal-constant-storage-in-memory>
- <http://msdn.microsoft.com/en-us/library/fw5abdx6.aspx>
- <http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html>

Dla dociekliwych

Co robi ten program?

```
#include <wchar.h>

int main(int argc, char** argv) {

    const wchar_t*p=L"Kraków";
    size_t i,j;

    for(j=0;j<wcslen(p);j++){
        unsigned char*a=(unsigned char*)(p+j);

        for(i=0;i<sizeof(wchar_t);i++){
            printf("%02x",*(a+i));
        }
        printf("|");
    }
    return 0;
}
```

4b00|7200|6100|6b00|f300|7700|