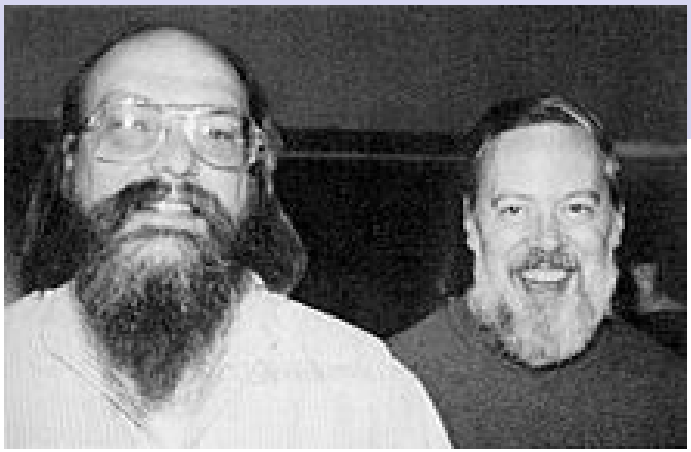
Język ANSI C

Pierwsze starcie.

Ostatnia aktualizacja: 2 listopada 2026

Znowu trochę historii



1972 Dennis Ritchie (Bell Labs w New Jersey), projekt języka C na bazie języka B

1973 UNIX, jądro w C, pierwszy przenośny system operacyjny

1978 D. Ritchie, Brian Kernighan, *„The C Programming Language"*

1983 Bjarne Stroustrup, Język C++

1989 Standard ANSI C, standardowe C, „pure C", C89, C90

1999 Standard C99

2011 Standard C11

2024 Standard C23

```c
#include <stdio.h>
#define PI 3.1415

int main()
{
    int i;
    float x;

    i = 10 * PI;
    x = 1.0;

    return 0;
}
```

Dyrektywy preprocesora

Funkcja główna

Deklaracje zmiennych

Instrukcje programu

Najkrótszy program w C

```
main(){}
```

Witaj świecie

```
#include<stdio.h>

int main()
{
   puts("Witaj swiecie!");
   return 0;
}
```

- mały język, ale duże możliwości; ważna jest rola bibliotek
- **pliki źródłowe** `*.c` zawierają instrukcje programu
- **pliki nagłówkowe** `*.h` zawierają deklaracje typów i funkcji, nie zawierają instrukcji
- biblioteki: zbiory typów danych, stałych i funkcji
- biblioteki standardowe, np.: `stdio.h`, `math.h`
- dostęp do pamięci i rejestrów
- zwięzły kod - ale nie wolno przesadzać
- C nie chroni programisty przed nim samym

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**

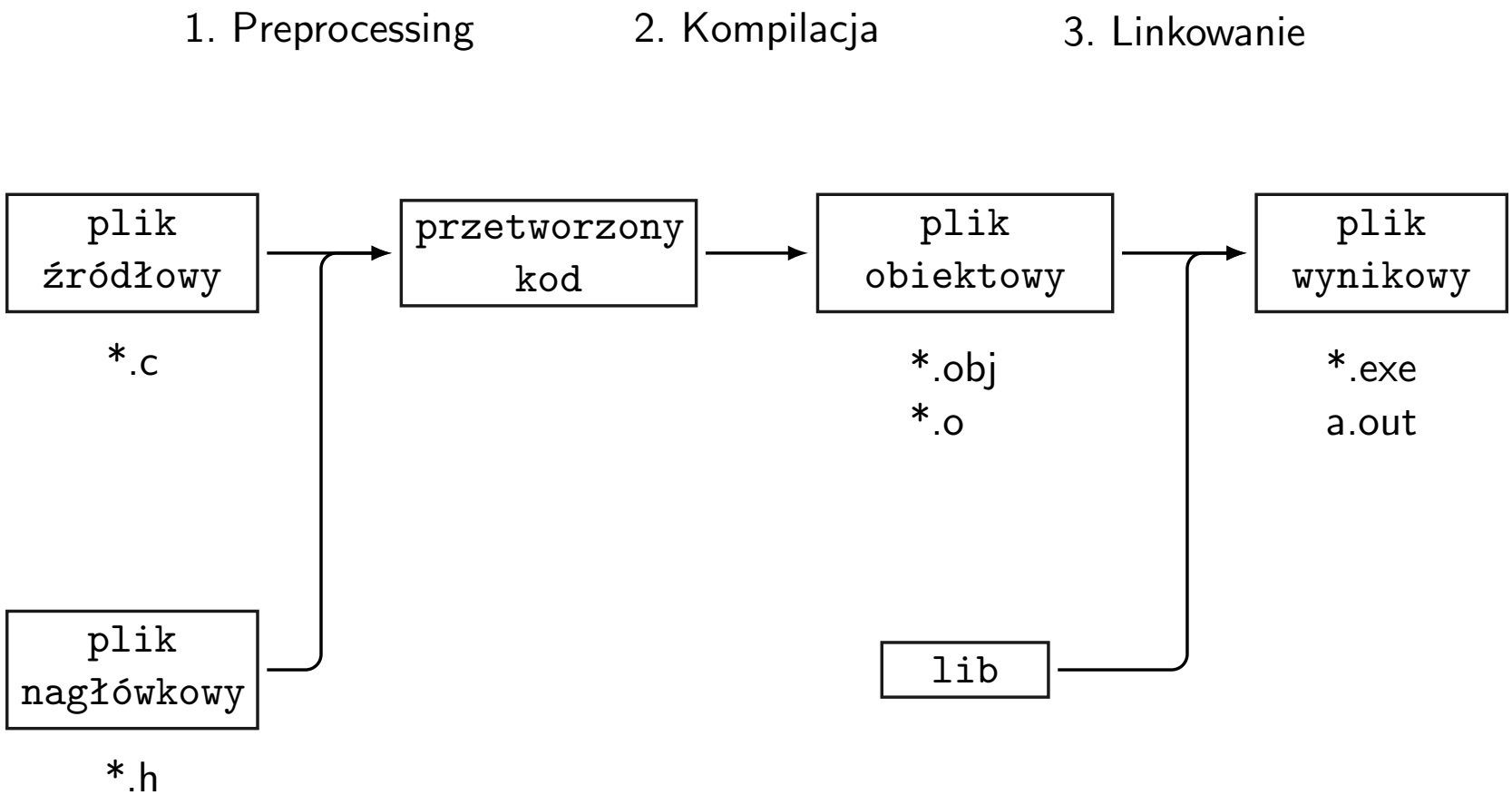
Dyrektywy to instrukcje zaczynające się od znaku #.
Nie są słowami języka C, wykonywane są podczas przetwarzanie wstępnego, przed właściwą kompilacją.

- `#include` *plik*
  dołączenie pliku nagłówkowego zawierającego definicje funkcji, typów i stałych

```
#include <stdio.h>
#include "../plik.h"
```

- #define *nazwa wartość*
  Definiuje makra, czyli reguły podstawiania tekstu podczas przetwarzania wstępnego.

```
#define PI 3.1415
#define TRUE 1
#define FALSE 0
#define real float
```


1. Preprocessing
2. Kompilacja
3. Linkowanie
plik źródłowy
*.c
przetworzony kod
plik obiektowy
*.obj
*.o
plik wynikowy
*.exe
a.out
plik nagłówkowy
*.h
lib

Preprocesor wykonuje instrukcje zaczynające się znakiem # (dyrektywy preprocesora). Przygotowuje pliki do kompilacji.

pliki źródłowe (`*.c, *.h`) ⇒ przetworzone pliki

```
#include<stdio.h>
#define PI 3.14
```

Kompilacja tłumaczy instrukcje C na kod maszynowy

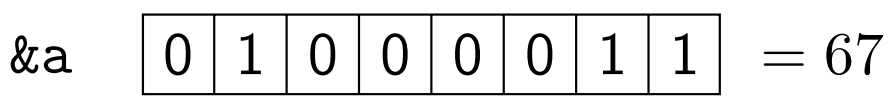
przetworzone pliki ⇒ pliki obiektowe (`*.obj, *.o`)

Konsolidacja (linkowanie) łączy pliki obiektowe w aplikację

pliki obiektowe + biblioteki ⇒ program (`*.exe , a.out`)

- **zmienne** są określone przez typ i unikatową nazwę
- każda zmienna zajmuje pamięć: 1B, 2B, 4B, 8B, ...
- **adres** zmiennej określa jej położenie w pamięci
- binarna reprezentacja zmiennych
- **skończoność** - liczby są reprezentowane poprawnie w pewnym przedziale
- **rozdzielczość** - liczby rzeczywiste są reprezentowane z pewną dokładnością

```
char a='C';
```



typ	reprezentacja	zakres wartości
`char`	znaki (*character*), kod ASCII	$[-128, 127]$
`int`	liczby całkowite (*integer*)	$[-2^{31}, 2^{31} - 1]$
`float`	liczby rzeczywiste (*floating point*) reprezentacja zmiennoprzecinkowa najmniejsza dodatnia wartość: 1.17×10^{-38}	$[-3.4 \times 10^{38},$ $+3.4 \times 10^{38}]$
logiczny	brak typu logicznego wartość całkowita 0 to fałsz a wartość różna od 0 to prawda	
`void`	typ pusty, brak typu	

typ `bool` wprowadzony w standardzie C23

Deklaracja

występuje na początku bloku instrukcji

typ identyfikator;
typ identyfikator1, identyfikator2, identyfikator3;

powiązanie zmiennej, stałej lub funkcji danego typu z identyfikatorem (unikatową nazwą)

Przykład:

```
int i;
float x, y;
char c, d, e;
```

- dozwolone znaki: litery `a-z` i `A-Z`, cyfry `0-9` oraz podkreślnik `_`
- cyfra nie może być pierwszym znakiem
- małe i duże litery są rozróżniane: `a` $\neq$ `A`
- zarezerwowane słowa nie mogą wystąpić w roli identyfikatorów:
  - słowa kluczowe języka (np. `while`, `int`, `return`)
  - identyfikatory zadeklarowane wcześniej (zmienne, funkcje)

Przykład:

```
int a,b,c;
float srednica_kola;
char PewienZnak;
float x1, x2, x3;
int _wazna_zmienna;
```

```
int 4abc;
float średnica;
char pewien znak;
int wazna-zmienna;
float return;
```

Źle!

Instrukcja prosta

wyrażenie ;

```
a = x + 1;
```

Instrukcja prosta jest zawsze zakończona średnikiem.

Instrukcja złożona (blok instrukcji)

```
{
    instrukcja 1
    instrukcja 2
    instrukcja 3
}
```

```
{
    float x, y;
    x = 1.0;    y = 2.4;
    x = x + y;
}
```

Instrukcje sterujące

`if` `else` `do` `while` `for` `goto`

```
int x = 1;

int main()
{
    int x = 2;

    {
        int x = 3;


    }

}
```

zmienna **globalna**,
dostępna w zakresie całego pliku

zmienna **lokalna**,
dostępna w zakresie funkcji `main`,
przysłania zmienną globalną `x`

zmienna lokalna
dostępna w zakresie bloku instrukcji {}
przysłania zmienną `x` z funkcji `main`

Zmienne deklaruje się na samym początku bloku (czyli przed pierwszą instrukcją).

Od C99 można deklarować w dowolnym miejscu przed użyciem zmiennej

Operatory arytmetyczne

* / % + -

Operatory relacji

< > <= >= == !=

Operatory logiczne

! && ||

Operator przypisania

=

```
zmienna = wyrażenie;
```

```
int i, j, k;
float x = 1.5;
float y = 1.5e-5;
char znak = 'A';

i = i + 3;
3 = x;
x + y = x - y;
znak = 65;
```

deklaracje zmiennych

inicjalizacja

brak inicjalizacji

zła kolejność

błąd

Zmienne lokalne nie są domyślnie inicjowane. Jeśli nie przypiszesz zmiennej wartości, będzie ona miała nieokreśloną wartość. Dlatego warto zawsze inicjować zmienne.

	przypisanie	porównanie
C	`a = 3`	`a == 3`
Pascal	`a := 3`	`a = 3`
pseudo-kod	$a \leftarrow 3$	$a = 3$

*	mnożenie	`x * y`	$x \cdot y$
/	dzielenie	`x / y`	$\frac{x}{y}$
+	dodawanie	`x + y`	$x + y$
-	odejmowanie	`x - y`	$x - y$
%	reszta z dzielenia (modulo)	`x % y`	$x \mod y$

- Reszta z dzielenia (modulo, %) jest określona tylko dla argumentów całkowitych.
- Operator dzielenia / dla argumentów całkowitych wykonuje **dzielenie całkowite**.

```
1 / 2 → 0
1 / 2.0 → 0.5
1.0 / 2 → 0.5
```

Priorytety operatorów

`*` `/` `%`	mają wyższy priorytet
`+` `-`	mają niższy priorytet

Dla operatorów arytmetycznych o takim samym priorytecie obliczenia są wykonywane od lewej do prawej.

$$z = \frac{x}{2y}$$

```
z = x / 2 * y;      /* zle */
z = x / (2 * y);
```

$$z = \frac{x - y}{x + y}$$

```
z = x - y / x + y;      /* zle */
z = (x - y) / (x + y);
```

W razie wątpliwości użyj nawiasów okrągłych.

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

`stdio.h`: **St**an**d**ard **i**nput/**o**utput; biblioteka obsługująca komunikację ze standardowym wejściem i wyjściem aplikacji.

`printf()`

formatowane wyjście (wyświetlanie w terminalu)

```
printf("format", arg1, arg2, ... )
```

`scanf()`

formatowane wejście (odczyt z klawiatury)

```
scanf("format", adres1, adres2, ... )
```

Format

specyfikator	znaczenie	przykład	wynik
%f	zmiennoprzecinkowa	`printf("%f",3.14);`	3.140000
%.2f	2 miejsca po przecinku	`printf("%.2f",3.14);`	3.14
%d	dziesiętna	`printf("%d",75);`	75
%c	znak	`printf("%c",75);`	K
%x	szesnastkowy	`printf("%x",75);`	4b

Przykład z większą liczbą argumentów

```
int a=2;
int b=3;
printf("Liczba %d plus %d wynosi %d\n", a, b , a + b);
```

Wynik:

```
Liczba 2 plus 3 wynosi 5
```

Symbole specjalne

symbol	znaczenie	przykład	wynik
\n	nowa linia	printf("n\nn")	n n
\t	tabulator	printf("t\tt")	t t
\"	"	printf("raz \"dwa\"")	raz "dwa"
\\	ukośnik \	printf("C:\\Users\\")	C:\Users\
%%	%	printf("%d%%", 5)	5%
\?	?	printf("Jeszcze raz\?")	Jeszcze raz?

Przykład

```
printf("P\tj\tz%cr\te\ta\no\ts\tb\ng\tt\ta\nr\t ", 10);
printf("\tw\n%x\t \tn\nm\t \te\no\nw\na\nn\ni\ne\n", 10);
```

☞ printf–specjalne.c

- Specyfikator formatu powinien pasować do typu zmiennej.

```
int a;
scanf("%f", &a);
```

problem !!!

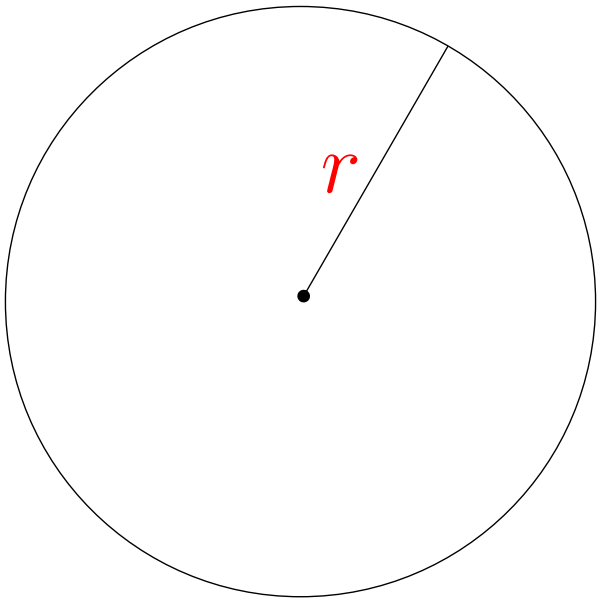
- Drugim i kolejnymi argumentami funkcji `scanf` są adresy zmiennych, np. `&a`.
- Formaty `%f` i `%d` pomijają początkowe białe znaki aż do napotkania wartości liczbowej.
- Znak niepasujący do formatu może przerwać wczytywanie i pozostać w strumieniu wejściowym.

```
char a;
float x,y;
scanf("%f%f", &x, &y);
scanf("%c", &a);
```

wczyta 2 liczby oddzielone białymi znakami

czyta pojedynczy znak

Problem: wyznacz pole i obwód koła o promieniu r.



$$P_\circ = \pi r^2 \qquad O_\circ = 2\pi r$$

Przykład: pole i obwód koła

```c
#include <stdio.h>
#define PI 3.14159

int main()
{
    float r, pole, obw;

    printf("Podaj promien kola\nr= ");
    scanf("%f",  &r);

    pole = PI * r * r;
    obw  = 2 * PI * r;

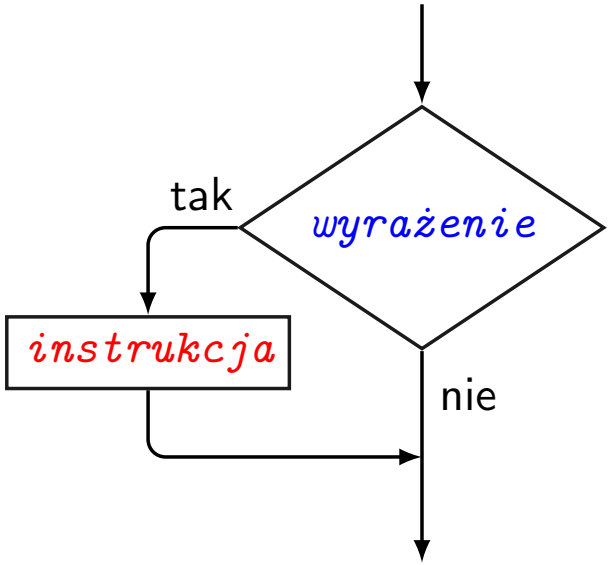
    printf("Pole kola o promieniu %f wynosi %.2f\n", r, pole);
    printf("Obwod kola o promieniu %f wynosi %.2f\n", r, obw);

    return 0;
}
```

☞ kolo.c

Składnia instrukcji `if`

```
if ( wyrażenie )
    instrukcja
```



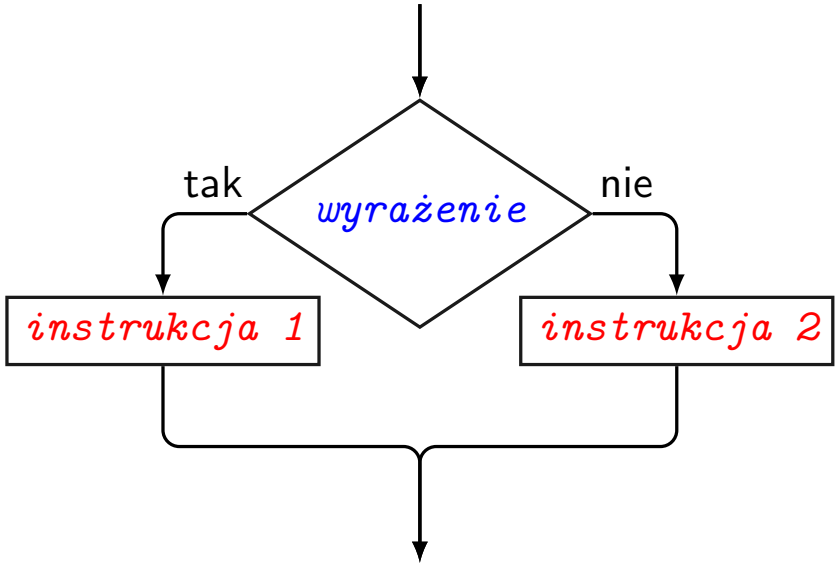
Przykład:

```
if( x > 0 ) printf("liczba dodatnia");

if ( x % 2 == 0 )
{
    printf("liczba parzysta");
    x = x - 1;
}
```

Składnia instrukcji `if else`

```
if ( wyrażenie )
    instrukcja 1
else
    instrukcja 2
```



Przykład:

```
if( x % 2 ) printf("liczba nieparzysta");
else printf("liczba parzysta");

if ( x > 0 )
{
    printf("liczba dodatnia");
    x = x - 1;
}
else x = 0;
```

operator	znaczenie	przykład	mat.
<	mniejszy niż	`x < y`	$x < y$
>	większy niż	`x > y`	$x > y$
<=	mniejszy lub równy	`x <= y`	$x \leqslant y$
>=	większy lub równy	`x >= y`	$x \geqslant y$
==	równy	`x == y`	$x = y$
!=	różny	`x != y`	$x \neq y$

operator	znaczenie	przykład	mat.
!	negacja (NOT)	`!x`	$\neg x$
&&	koniunkcja (AND)	`x>1 && y<2`	$x > 1 \wedge y < 2$
||	alternatywa (OR)	`x<1 || y>2`	$x < 1 \vee y > 2$

x	!x
0	1
1	0

x	y	x && y
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x || y
0	0	0
0	1	1
1	0	1
1	1	1

```
if ( x > 0)
   if ( x < 10 )
      printf("liczba wieksza od 0 i mniejsza niz 10");

if ( x > 0 && x < 10 )
   printf("liczba wieksza od 0 i mniejsza niz 10");

if (!(x > 0)) printf("liczba ujemna lub zero");

if ( !x > 0 ) printf("\?");

if ( x+1 > y-1 ) printf("\?");

if ( x  || ! y && z) printf("\?");
```

Kolejność operatorów: !, arytmetyczne, relacje, &&, ||
W razie wątpliwości użyj nawiasów okrągłych.

Przykład: Równanie z jedną niewiadomą

Problem: znajdź miejsce zerowe funkcji liniowej

$$f(x) = ax + b$$

Algorithm Równanie z jedną niewiadomą

Dane wejściowe: współczynniki $a, b \in \mathbb{R}$

Wynik: miejsce zerowe $x_0 \in \mathbb{R}$ lub informacja o braku rozwiązania

1: **jeżeli** $a \neq 0$ **wykonaj**
2: $\quad x_0 \leftarrow -\frac{b}{a}$
3: $\quad$ **wypisz:** x_0
4: **w przeciwnym wypadku**
5: $\quad$ **wypisz:** `Brak rozwiązania`

```c
#include <stdio.h>

int main()
{
    float a, b, x0;

    printf("Podaj wspolczynniki rownania\n");
    printf("a = "); scanf("%f", &a);
    printf("b = "); scanf("%f", &b);

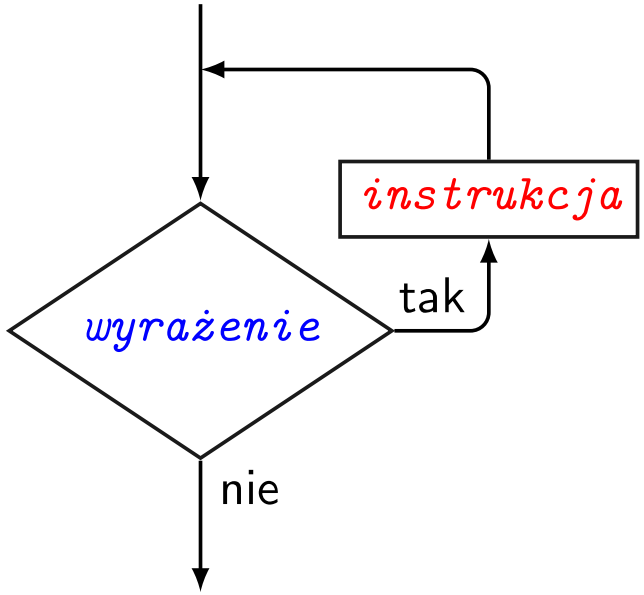
    if( a != 0.0 )
    {
        x0 = -b / a;
        printf("x0 = %.4f\n", x0);
    }
    else printf("Brak rozwiazan\n");

    return 0;
}
```

☞ linia.c

Składnia pętli `while` (dopóki)

```
while ( wyrażenie )
    instrukcja
```



Przykład

```
int n = 10;
while( n > 0 )
{
    printf("%d\n", n);
    n = n - 1;
}
```

Pętla nieskończona

```
while(1) printf("C");
```

Problem: wyznaczenie wartości silni $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$

Algorithm Silnia

Dane wejściowe: liczba całkowita $n \geqslant 0$

Wynik: wartość $x = n!$

1: $i \leftarrow 2$

2: $x \leftarrow 1$

3: **dopóki** $i \leqslant n$ **wykonuj**

4: $\quad x \leftarrow x \cdot i$

5: $\quad i \leftarrow i + 1$

6: **wypisz** x

```c
#include<stdio.h>

int main()
{
    int x, i, n;

    printf("n = ");
    scanf("%d", &n);

    if(n<0) printf("Zle dane: n<0\n");
    else
    {
        x=1;
        i=2;
        while( i <= n )
        {
            x = x * i;
            i = i + 1;
        }
        printf("%d! = %d\n", n, x);
    }
    return 0;
}
```

☞ silnia.c

Tablica

przechowuje elementy tego samego typu, elementy identyfikowane liczbami (indeksem).

Przykład

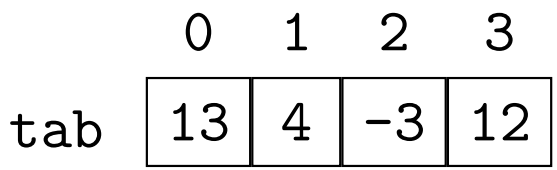
```
int tab[4];

tab[0] = 13;
tab[1] = 4;
tab[2] = -3;
tab[3] = tab[0] - 1;
tab[4] = -1;
```

Deklaracja tablicy 4 elementowej

Tablice w C są indeksowane od 0

Źle! Poza zakresem.



Przykład: odwracanie kolejności

```c
#include<stdio.h>

int main()
{
    int tab[100];
    int i = 0;
    int a = -1;

    printf("Podaj sekwencje liczb calkowitych.\n");
    printf("Aby zakonczyc podaj 0.\n");

    while( a != 0 && i < 100)
    {
        scanf("%d", &a);
        tab[i] = a;
        i = i + 1;
    }

    printf("Podales %d liczb.\n", i);

    while(i > 0)
    {
        i = i - 1;
        printf("%d\n", tab[i]);
    }

    return 0;
}
```

```
1  #include<studio.h>;
2
3  char main()
4  {
5     int n
6
7     printf("Podaj liczbe calkowita wieksza od zera: ");
8     scanf("%f",n);
9
10    if(n <= 0)
11       if ( n=0 ) printf("To jest zero!\n");
12    else
13    {
14       printf("Dzielniki liczby %d:\n ",n);
15       int i;
16       while( i<n );
17       {
18          if( n % i )  printf("%c/n",i);
19          i = i + 1;
20       }
21    }
22    return 0;
23 }
```

```c
#include<stdio.h>                                   /* studio.h, srednik */

int main()                                                    /* int */
{
   int n,i=1;                  /* srednik, deklaracja, inicjalizacja */

   printf("Podaj liczbe calkowita  wieksza od zera : ");
   scanf("%d",&n);                                  /* format, adres */

   if(n <= 0)
   {                                                        /* nawiasy */
      if ( n==0 ) printf("To jest zero!\n");                    /* == */
   }
   else
   {
      printf("Dzielniki liczby %d:\n",n);
      while( i<=n )                      /* srednik, p. nieskonczona */
      {
         if( n % i == 0 )  printf("%d\n",i);          /* %d, \n, == */
         i = i + 1;
      }
   }
   return 0;
}
```

☞ liczba–ok.c

```
1 #include<stdio.h>
2 int nwd(int a,int b){int c;
3 while(b!=0){c=a%b;a=b;b=c;}
4 return a;} int main(){int a,
5 b;printf("Podaj dwie li"
6 "czby calkowite: ");scanf("%d %d",
7 &a,&b);printf("NWD(%d,%d) = %d\n",
8 a,b,nwd(a,b));return 0;}
```

☞ nwd–balagan.c

- Czytelność przede wszystkim
- ☞ The International Obfuscated C Code Contest
- narzędzia dbające o czytelność

  `$ astyle nwd-balagan.c`

```c
#include<stdio.h>
int nwd(int a,int b)
{
int c;
while (b!=0)
{
c=a%b;
a=b;
b=c;
}
return a;
}
int main()
{
int a,b;
printf("Podaj dwie liczby calkowite: ");
scanf("%d %d",&a,&b);
printf("NWD(%d,%d) = %d\n",a,b,nwd(a,b));
return 0;
}
```

☞ **nwd–balagan2.c**

Styl Allmana (BSD)

```
int main()
{
    int i, n;

    scanf("%d", &n);
    i=0;
    while (i < n)
    {
        if( i % 2 )
        {
            printf("%d\n",i);
        }
        i = i + 1;
    }
    return 0;
}
```

Styl K&R (GNU)

```
int main()
{
    int i, n = 100;

    scanf("%d", &n);
    i=0;
    while (i < n){
        if( i % 2 ){
            printf("%d\n",i);
        }
        i = i + 1;
    }
    return 0;
}
```

- Wewnętrzne bloki instrukcji wcięte względem zewnętrznych
- Instrukcje w jednym bloku zaczynają się w tej samej kolumnie
- Ogranicz długość linii do 80–120 znaków (szerokość ekranu)
- Rozbijaj długie instrukcje na kilka linii, zachowując czytelność:

```
while ((a < b && wplata(x) != -1) ||
       (c > d && sprawdz(x) == 1)) {
    if (w != NULL) {
        a = a - 1 + sin(PI * 2);
    }
}
```

- Oddzielaj deklaracje zmiennych, funkcje i logiczne bloki kodu pustymi liniami.
- Stosuj odstępy (spacje) między operatorami i po przecinkach.
- Wcięcia w kodzie C poprawiają czytelność ale ich brak nie generuje błedu kompilacji. Istnieją jezyki (Python), w których wcięcia są elementem składni.

Komentarze pozwalają umieścić dodatkowe informacje dla czytających kod (nie są kompilowane)

```
/* komentarz blokowy */
// komentarz liniowy wprowadzony w C99
```

```c
int main()
{
    /* Wszystko co tutaj jest napisane
       jest komentarzem i nie zostanie skompilowane */

    int x;    /* Bardzo wazna zmienna */
    int y;    // Komentarz do konca linii
}
```

Pisz **krótkie, klarowne komentarze** wyjaśniające „dlaczego”, a nie „co” (kod powinien być samodokumentujący się).
Nie komentuj rzeczy oczywistych.

```c
i = i + 1;    /* Zwiekszenie licznika o 1 */
i = i + k;    /* Ustawienie indeksu na ostatni element */
```

Standard C99

- funkcje `inline`
- deklaracje zmiennych w dowolnym miejscu w programie
- typ logiczny (`bool`), `long long int`
- tablice o zmiennej liczbie elementów
- komentarze w stylu C++ `// to jest komentarz`
- biblioteki, np.: `complex.h`, `stdbool.h`

Podsumowanie

- każdą zmienną trzeba zadeklarować (określić typ i nazwę)
- nie zapomnij o średniku na końcu instrukcji
- operator przypisania `a=b`, operator porównania `a==b`
- najpierw deklaracja zmiennej, potem jej użycie (ANSI C)
- inicjuj zmienne wartościami początkowymi
- czytelność kodu jest bardzo ważna:
  nieczytelny kod najprawdopodobniej jest błędny; ważne są zrozumiałe nazwy zmiennych, wcięcia i komentarze
- spójny styl w całym projekcie

- Co wypisze na wyjściu poniższa instrukcja?

```
while (0) printf("C");
```

- Znak nowej linii '\n' jest kodowany za pomocą wartości dziesiętnej 10. Co wypisze na wyjściu poniższy program?

```
char x = '\n';
printf("%x%c%d", x+1, x, x/3 );
```

- Jaki typ służy do reprezentowania liczb rzeczywistych?
- Jaki indeks posiada pierwszy element tablicy w C?


David Griffiths, Dawn Griffiths *„Rusz głową! C.”*, Helion, Gliwice, 2013.


„Kurs programowania w C”, WikiBooks, `http://pl.wikibooks.org/wiki/C`


„C Programming Tutorial”, Tutorials Point, `http://www.tutorialspoint.com/cprogramming/`