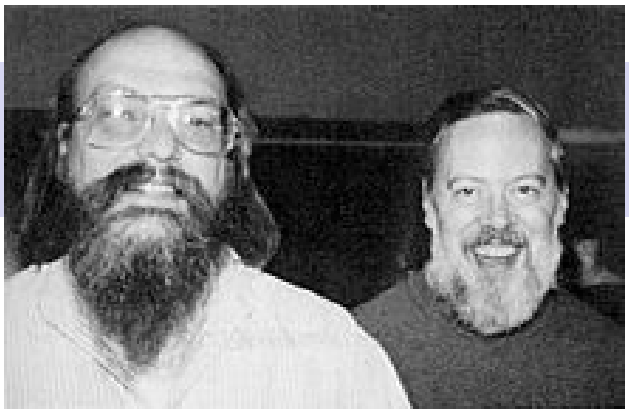
# Język ANSI C

Pierwsze starcie.



# Znowu trochę historii

1972 Dennis Ritchie (Bell Labs.,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

# Struktura programu w C

```c
#include <stdio.h>          /* Dyrektywy preprocesora */
#define PI 3.1415

int main()                  /* Funkcja glowna */
{
    int i;                  /* Deklaracje */
    float x;

    i = 10 * PI;            /* Instrukcje */
    x = 1.0;

    return 0;
}
```

# Najkrótszy program

## Najkrótszy program w C

```
main(){}
```

## Witaj świecie

```
#include<stdio.h>

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

# Język ANSI C

- mały język ale duże możliwości, ważna rola bibliotek
- pliki źródłowe `*.c`
- 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

# Słowa kluczowe

| | | | |
|---|---|---|---|
| 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 preprocesora

Instrukcje zaczynające się od znaku #. Nie są słowami języka C i wykonywane są przed właściwą kompilacją.

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

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

- `define`
  pozwala zdefiniować stałe lub przezwać słowa kluczowe

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

# Ogólnie o procesie kompilacji

- **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`

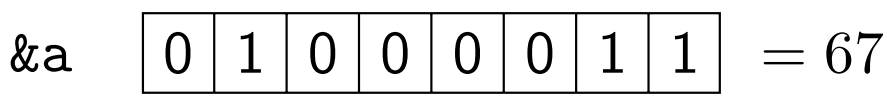
- **kompilacja** tłumaczy instrukcje C na kod maszynowy
  przetworzone pliki ⇒ pliki obiektowe (`*.obj, *.o`)

- **konsolidacja (linkowanie)** łączy pliki obiektowe w aplikację
  pliki obiektowe + biblioteki ⇒ program (`*.exe , a.out`)

# Zmienne w komputerze

- **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 reprezentowane poprawnie w pewnym przedziale
- rozdzielczość - liczby rzeczywiste reprezentowane z pewną dokładnością

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



# Podstawowe typy danych

| typ | reprezentacja |
|---|---|
| `char` | znak (*character*), kod ASCII |
| | rozmiar 1 bajt |
| | zakres wartości $[-127, 128]$ |
| `int` | liczby całkowite (*integer*) |
| | rozmiar 4 bajty |
| | zakres $[-2^{31}, 2^{31}]$ |
| `float` | liczby rzeczywiste, zmiennoprzecinkowa (*floating point*) |
| | rozmiar 4B |
| | zakres $[-3.4 \times 10^{38}, 3.4 \times 10^{38},]$ |
| | najmniejsza dodatnia wartość $1.17 \times 10^{-38}$ |
| logiczny | brak typu logicznego |
| | wartość całkowita 0 to fałsz a wartość różna od 0 to prawda |
| `void` | typ pusty, brak typu |

## Deklaracja

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

## Identyfikatory

- dozwolone znaki: litery `a-z`, `A-Z`, cyfry `0-9`, podkreślnik `_`
- cyfra nie może być pierwszym znakiem
- małe i duże litery są rozróżniane: a $\neq$ `A`
- zarezerwowane słowa: zadeklarowane wcześniej identyfikatory, słowa kluczowe

Przykłady deklaracji zmiennych:

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

```
int 0abc;
float średnica;
char pewien znak;
int wazna-zmienna;
```

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`

# Operatory

## Operatory arytmetyczne

`*` `/` `%` `+` `-`

## Operatory relacji

`<` `>` `<=` `>=` `==` `!=`

## Operatory logiczne

`!` `&&` `||`

## Operator przypisania

`=`

# Operacja przypisania wartości

```
zmienna = wyrażenie;
```

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

i = i + 3;                          /* brak inicjalizacji */
3 = x;                              /* zle */
x + y = x - y;                      /* zle */
znak = 65;
```

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

# Operatory arytmetyczne

| | | | |
|---|---|---|---|
| * | 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` | $x \mod 2$ |

- Reszta z dzielenia (modulo, %) określona jest tylko dla argumentów całkowitych
- Operator dzielenia / dla argumentów całkowitych realizuje **dzielenie bez reszty!**

`1/2` $\rightarrow$ `0`
`1/2.0` $\rightarrow$ `0.5`

## Priorytety operatorów

| | | | |
|---|---|---|---|
| * | / | % | wyższy priorytet |
| + | - | | niższy priorytet |

Dla operatorów 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.

## Funkcja printf()

formatowane wyjście (wyświetlanie w terminalu)

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

## Funkcja 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", x, y , x + y);
```

Wynik:

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

## Symbole specjalne

| symbol | znaczenie | przykład | wynik |
|---|---|---|---|
| \n | nowa linia | printf("n\nn") | n<br>n |
| \t | tabulator | printf("t\nt") | 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\nr\te\ta\no\ts\tb\ng\tt\ta\nr\t \t");
printf("w\na\t \tn\nm\t \te\no\nw\na\nn\ni\ne\n");
```

☞ printf–specjalne.c

# SCANF - KILKA WAŻNYCH UWAG

- specyfikator formatu powinien pasować do typu zmiennej

```
int a;
scanf("%f",&a);                    /* problem !!! */
```

- drugim argumentem funkcji `scanf` jest adres zmiennej (pamiętaj o &)
- formaty %f i %d pomijają poprzedzające liczbę białe znaki
- znak niepasujący do formatu przerywa wczytywanie i pozostaje w strumieniu wejściowym

```
char a;
float x,y;
scanf("%f%f",&x,&y);          /* wczytanie 2 liczb */
scanf("%c",&a);               /* czyta jeden znak */
```

# Przykład: pole i obwód koła

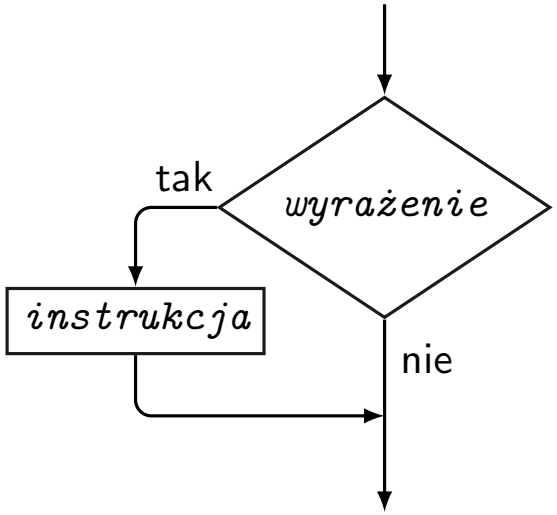
```c
#include <stdio.h>
#define PI 3.14159

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

    return 0;
}
```

☞ kolo.c

# Instrukcja warunkowa if else

## Warunek if (jeżeli)

```
if ( wyrażenie )
    instrukcja
```



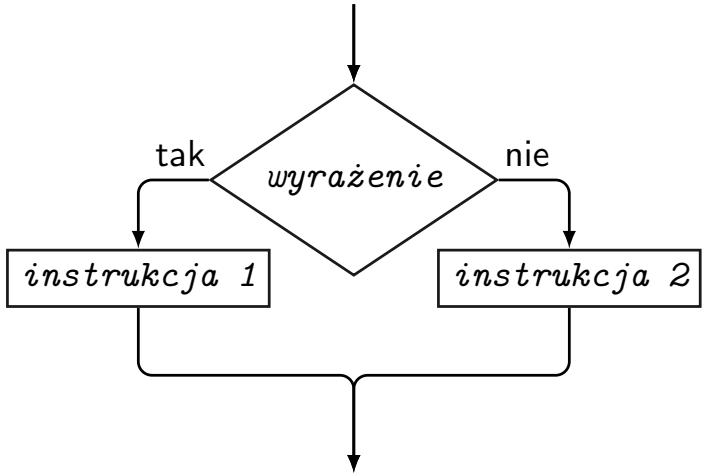

## Przykład:

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

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

# Instrukcja warunkowa if else

## Warunek 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;
```

# Operatory relacji

| 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` | $x \neq y$ |

# Operatory logiczne

| 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 |

# Przykłady

```
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, relacji, &&, ||
W razie wątpliwości użyj nawiasów okrągłych.

# Przykład: Równanie z jedną niewiadomą

Problem: znajdź miejsce zerowe funkcji liniowej

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

**Algorytm 1** 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

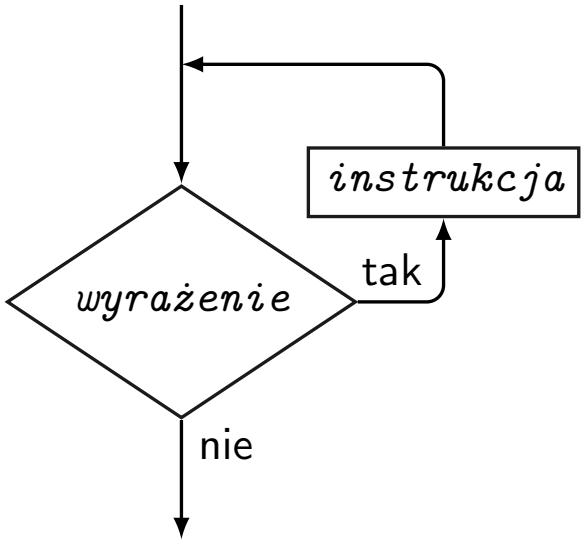
1: **jeżeli** $a \neq 0$ **wykonaj**
2: $\quad x_0 \leftarrow -\frac{b}{a}$
3: $\quad$ **wypisz:** $x_0$
4: **w przeciwnym wypadku**
5: $\quad$ **wypisz:** `Brak rozwiązania`

```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

## Pętla while (dopóki)

```
while ( wyrażenie )
    instrukcja
```




## Przykład

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

## Pętla nieskonczona

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

# Przykład: wyznaczanie silni $n!$

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

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

☞ silnia.c

# Bądź kompilatorem

## Wypisywanie podzielników liczby całkowitej

```
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 }
```

# Bądź kompilatorem

```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

# Graficzna postać kodu

```
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(){
5 int a,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

```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;

    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;
}
```

# Wcięcia

## Graficzna postać kodu

- Wewnętrzne bloki instrukcji wcięte względem zewnętrznych
- Instrukcje w jednym bloku zaczynają się w tej samej kolumnie
- Nie przesadzaj z długością linii (max. 78 znaków)
- Oddzielaj deklaracje zmiennych od instrukcji lub grupy spójnych instrukcji pustymi liniami
- Długie ciągi instrukcji warto rozbić na kilka linii i otoczyć nawiasami

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

- Python - wcięcia elementem składni języka

# Komentarze w C

- Komentarz blokowy umieszcza się pomiędzy /* a */. Zawieraja informacje dla czytających kod (nie są kompilowane).

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

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

- Nie komentuj oczywistych rzeczy, raczej opis sensu operacji

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

- Komentarz liniowy // niedostępny w C89 !

## 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`

Uwaga: nie wszystkie kompilatory wspierają pełny standard C99 dlatego ANSI C daje największą szansę na przenośność.

- każdą zmienną trzeba zadeklarować (określić typ i nazwę)
- przejrzystość: czytelne nazwy zmiennych i wcięcia
- nie zapomnij o średniku na końcu instrukcji
- operator przypisania `a=b` a operator porównania `a==b`
- najpierw deklaracja potem użycie
- inicjuj zmienne

# Literatura dodatkowa



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/`