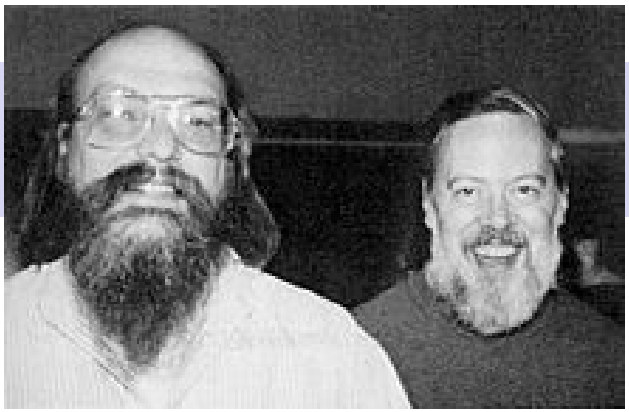
Język ANSI C

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
- modularność: programowanie proceduralne
- dostęp do pamięci i rejestrów
- zwięzły kod - ale nie wolno przesadzać
- C nie chroni programisty przed nim samym

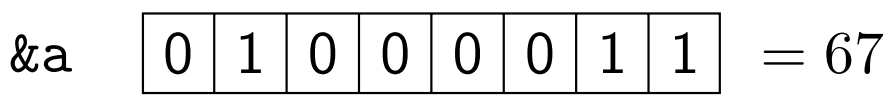
Ogólnie o procesie kompilacji

1. **preprocesor**
   pliki źródłowe (`*.c, *.h`) $\Rightarrow$ przetworzone pliki
   dyrektywy preprocesora, np.:
   ```
   #include<stdio.h>
   #define PI 3.14
   ```
2. **kompilacja**
   przetworzone pliki $\Rightarrow$ pliki obiektowe (`*.obj, *.o`)
3. **konsolidacja (linkowanie)**
   pliki obiektowe + biblioteki $\Rightarrow$ program (`*.exe , a.out`)

Binarna reprezentacja zmiennych

- zmienna zajmuje pamięć: 1B, 2B, 4B, 8B, ...
- adres zmiennej w pamięci:
- operacje na bitach
- typy całkowite ze znakiem i bez znaku, skończony zakres
- typy zmiennoprzecinkowe: rozdzielczość i skończoność

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



Podstawowe typy danych

- `float` - zmiennoprzecinkowa (*floating point*),
  4 bajty
  zakres $[-3.4 \times 10^{38}, 3.4 \times 10^{38},]$,
  wartość minimalna większa od zera 1.17×10^{-38}
- `int` - liczby całkowite (*integer*)
  4 bajty
  zakres $[-2^{31}, 2^{31}]$
- `char` - znak (*character*)
  1 bajt
  zakres $[-127, 128]$, kod ASCII
- brak typu logicznego
  0 to fałsz, nie 0 to prawda
- `void` - typ pusty, brak typu
- `unsigned int` - całkowita bez znaku

Deklaracja

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

Identyfikatory

- litery `a-z`, `A-Z`, cyfry `0-9` (ale nie pierwsza), podkreślnik `_`
- małe i duże litery rozróżniane: `zmienna`, `Zmienna`, `ZMIENNA`

Przykłady deklaracji zmiennych:

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

Niepoprawne nazwy: `2abc`, `promień`, `wazna zmienna`, `pewna-zmienna`

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**

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`

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

*	/	%	wyższy priorytet
+	-		niższy priorytet

Reszta z dzielenia (modulo, %) tylko dla zmiennych całkowitych
Dzielnie bez reszty (/) dla zmiennych całkowitych

Kolejność obliczeń

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

```
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);
```

Obliczenia od lewej do prawej, priorytet * / % przed + -

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

```
#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, ... )
```

Specyfikacja formatu

Format

`%f`	zmiennoprzecinkowa	`printf("%f",3.14);`	`3.140000`
`%.2f`	2 miejsca po przecinku	`printf("%.2f",3.14);`	`3.14`
`%e`	notacja naukowa	`printf("%e",0.01);`	`1.000000e-02`
`%d`	dziesiętne	`printf("%d",75);`	`75`
`%c`	znak	`printf("%c",75);`	`K`
`%x`	szesnastkowo	`printf("%x",75);`	`4b`
`%s`	łańcuch znakowy	`printf("%s","napis");`	`napis`

Symbole specjalne

`\n`	nowa linia	`\\`	\ *backslash*
`\t`	tabulator	`\' \" \?`	' " ?
`\b`	backspace	`\r`	powrót karetki
`%%`	%		

SCANF - KILKA WAŻNYCH UWAG

- dopasowanie formatu do typu zmiennej

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

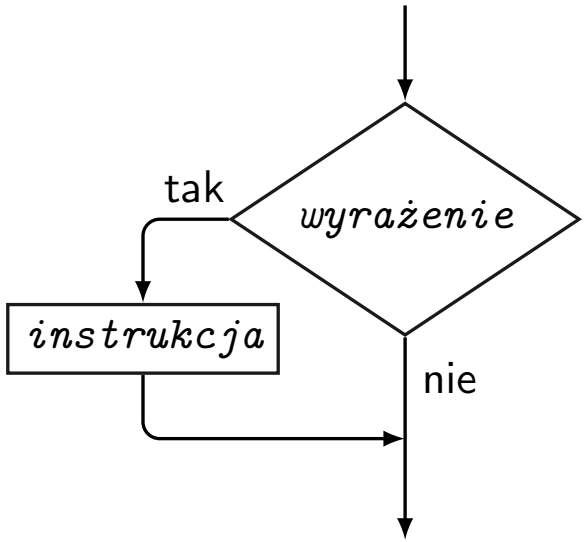
- argumentem jest adres zmiennej (pamiętaj o &)
- formaty liczbowe %f i %d zjadają poprzedzające białe znaki
- spacja w formacie pomija wszystkie białe znaki
- znak niepasujący do formatu przerywa wczytywanie i pozostaje w strumieniu wejściowym

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

Instrukcja warunkowa if else

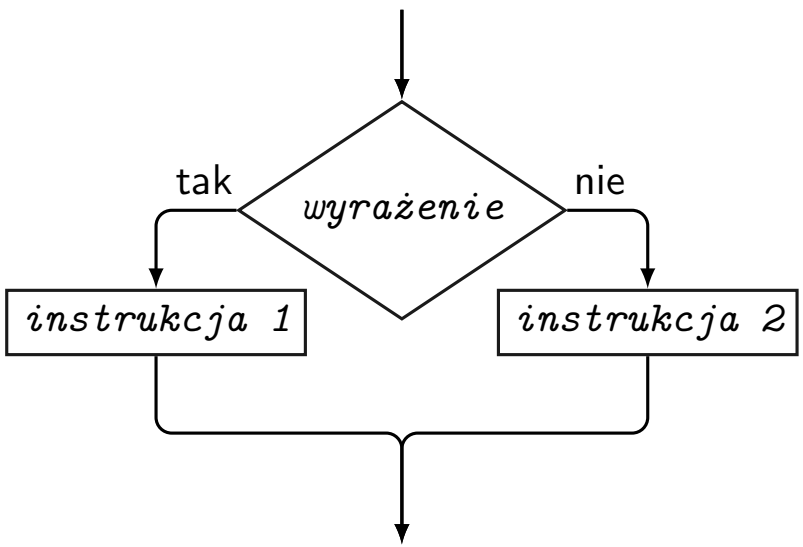
Warunek if (jeżeli)

```
if ( wyrażenie )
    instrukcja
```



Warunek if else

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



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

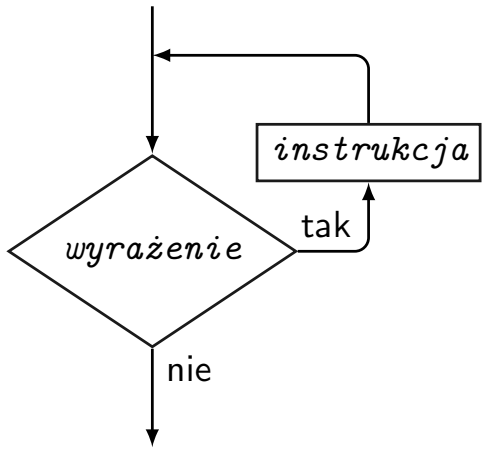
1: **jeżeli** $a \neq 0$ **wykonaj**
2: $\quad x_0 \leftarrow -\frac{b}{a}$
3: $\quad$ **wypisz:** x_0
4: **w przeciwnym wypadku**
5: $\quad$ **wypisz:** `Brak rozwiązania`

```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

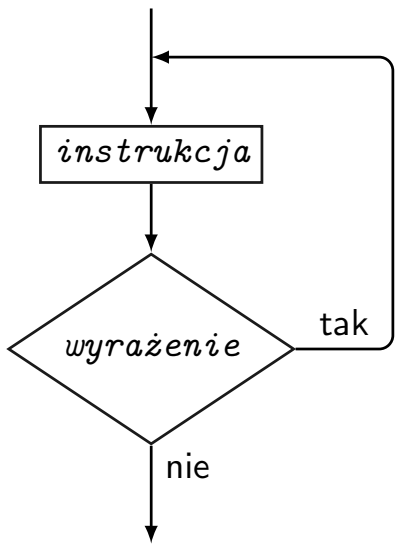
Pętla while (dopóki)

```
while ( wyrażenie )
    instrukcja
```



Pętla do while

```
do
    instrukcja
while ( wyrażenie );
```



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

Przykład: Algortym Herona

Metoda babilońska

Problem: wyznaczenie wartości pierwiastka $\sqrt{a}$

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \xrightarrow{n\to\infty} \sqrt{a}$$

Algorytm 3 Algorytm Herona

Dane wejściowe: liczba $a \geqslant 0$, wartość początkowa $x_0 > 0$, dokładność obliczeń $\epsilon > 0$

Wynik: wartość przybliżona $x \approx \sqrt{a}$ z dokładnością ϵ

1: $x \leftarrow x_0$
2: **wykonuj**
3: $\quad x_0 \leftarrow x$
4: $\quad x \leftarrow \frac{1}{2}\left(x_0 + \frac{a}{x_0}\right)$
5: **dopóki** $|x - x_0| > \epsilon$
6: **wypisz** x

```c
#include<stdio.h>

int main()
{
   const float eps=1e-4;
   float x,a,x0;

   printf("a = "); scanf("%f",&a);

   if( a < 0 ) printf("Zle dane: a < 0\n");
   else
   {
      x0 = 1;
      x = x0;
      do
      {
         x0 = x;
         x = (x0 + a/x0)/2;
      }while(x - x0 > eps || x - x0 < -eps);

      printf("pierwiastek z %f wynosi %f  (eps= %f)\n",a,x,eps);
   }
   return 0;
}
```

☞ heron.c

Bądź kompilatorem

Wypisywanie podzielników liczby całkowitej

```c
#include<studio.h>;

char main()
{
    int n

    printf("Podaj liczbe calkowita wieksza od zera: ");
    scanf("%f",n);

    if(n <= 0)
        if ( n=0 ) printf("To jest zero!\n");
    else
    {
        printf("Dzielniki liczby %d:\n ",n);
        int i;
        while( i<n );
        {
            if( n % i )  printf("%c/n",i);
            i = i + 1;
        }
    }
    return 0;
}
```

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

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`