



Politechnika
Wroclawska

Język C w pigułce

Wojciech Myszka

Katedra Mechaniki, Inżynierii Materiałowej i Biomedycznej

2 marca 2021





Język C

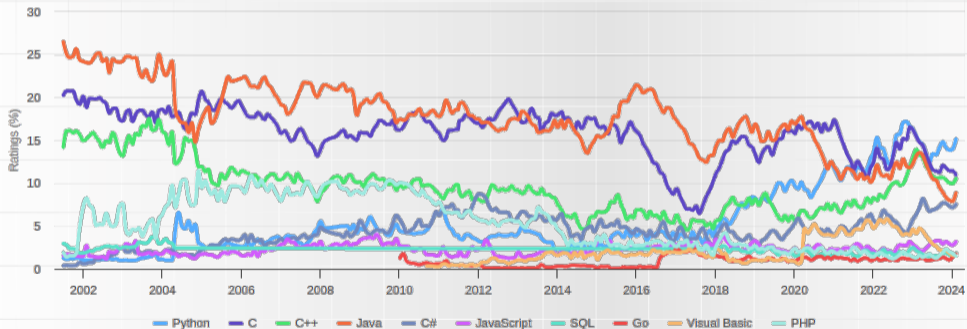
Czemu?

- ▶ nie wiem
- ▶ jest bardzo stary
- ▶ ale bardzo prosty
- ▶ ciągle używany

Tiobe index

TIOBE Programming Community Index

Source: www.tiobe.com



Tworzenie programu

1. Zadanie (problem)
2. Pomysł na rozwiązanie
3. Algorytm
4. Program
5. Kompilacja

Czyli konwersja programu w formacie czytelny dla człowieka do formatu zrozumiałego dla komputera

6. Uruchomienie i testowanie

Język C I

1. Jak każdy formalny język programowania ma bardzo ściśle zasady, które **muszą** być przestrzegane przez programistę.
2. Program w C składa się z jednego lub więcej „modułów” (zwanymi funkcjami)
3. Zawsze musi wystąpić moduł (funkcja) `main`

```
int main()  
{  
  ...  
}
```

między nawiasami klamrowymi wpisujemy kod programu

4. Ostatnim poleceniem powinno być polecenie `return 0`

5. Program może korzystać z różnych „rozszerzeń”
 - ▶ dostarczanych przez programistę
 - ▶ dostępnych **standardowo** w postaci „bibliotek”
6. W przypadku korzystania z zewnętrznych bibliotek trzeba dostarczyć kompilatorowi podstawowe informacje na temat biblioteki i dostępnych funkcjonalności.
7. Służy do tego polecenie `#include <...>`

Biblioteka wejścia/wyjścia

1. Jeżeli chcemy aby program wyprowadzał jakieś informacje (na przykład wyniki obliczeń)...
2. ... albo oczekuje jakichś informacji...
3. ... należy skorzystać z poleceń wejścia/wyjścia
4. Aby z nich skorzystać program powinien zawierać na samym początku linię

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




Podsumowanie

Podsumujmy jak powinien wyglądać minimalny kod programu

Minimalny kod programu

```
#include <stdio.h>
int main()
{
    return 0;
}
```

Komentarze

1. W treści programu można umieszczać teksty mające sens jedynie dla programisty (lub „czytelnika” kodu)
2. Komentarz jednolinijkowy:

```
// To jest tekst komentarza
```

3. Komentarz wielolinijkowy

```
/*  
Tu kilka linijek  
komentarza  
ograniczonych nawiasem  
zamykającym  
*/
```



Zmienne I

1. Każda zmienna przed pierwszym użyciem powinna być „zadeklarowana” (czyli określamy jej **nazwę** oraz **typ**). Dodatkowo, przed pierwszym użyciem zmienna powinna mieć jakąś wartość.
2. **Nazwa**
 - 2.1 Nazwa musi zaczynać się od litery
 - 2.2 „wielkie” i „małe” litery są różne!
 - 2.3 Znak podkreślenia (_) zaliczany jest do liter
 - 2.4 Oprócz liter w nazwie mogą pojawiać się cyfry
 - 2.5 **Nie może być odstępów**
3. Wartości zmiennych kodowane są **binarnie**. Aby poprawnie za/roz-kodować wartości binarne trzeba dodatkowo wiedzieć:
 - ▶ ile bitów zostało użytych (8/16/32/64/...)
 - ▶ jakiego typu są dane (inaczej koduje się litery, wartości całkowite i wartości niecałkowite)

4. Typ

4.1 litera

4.2 tekst

4.3 wartość całkowita

4.4 wartość niecałkowita

Typ całkowity

1. `int` — 32 bity
2. `short int` — 16 bitów
3. `long int` — 64 bity

do tego można dodać `unsigned` żeby zrezygnować z wartości ujemnych

Użycie

```
int a, b, c;  
long int d;  
short int e;
```

Stałe

```
10, -23 // Stałe typu int  
10U, 123U // Stałe typu unsigned int  
10L, -23L // Stałe typu long int
```



Typ niecałkowity

1. float — 32 bity (raczej nie używamy)
2. double — 64 bity (typ podstawowy)
3. long double — 80/128 bitów

Użycie

```
float a;  
double x, y, z;  
long double v;
```

Stałe

```
2., 3.5, -5.e6 // stałe typu double  
-3.2F, 2e3f // stałe typu float  
-11.345L, 34e2001 // stałe typu long double
```



Nadawanie wartości zmiennej

Podczas deklaracji

```
int a = 7, b = 5;  
double x = 11., v = 1e3;
```

Jako wynik operacji (arytmetycznej)

```
a = 3;  
b = a * a;  
x = 11.5;  
v = 0;  
x = x / v;
```

Operatory arytmetyczne

Ogólne zasady

1. Wszystkie operatory są dwuargumentowe choć
2. Wyjątkiem jest jednoargumentowy operator negacji (-)
3. Najlepiej jeżeli po obu stronach operatora są stałe lub zmienne tego samego typu
4. W przeciwnym razie nastąpić może ciąg automatycznych konwersji

Dotyczy to również operatora podstawienia (=)!

```
double x;
```

```
x = 2; // stała jest typu int, ale wynik będzie OK
```

```
x = 2/5; // obie stałe po prawej stronie są typu int  
// wynikiem dzielenia 5/2 będzie zero (typu int)
```


Operatory I

Arytmetyczne

operator	działanie
+	dodawanie
-	odejmowanie
*	mnożenie
/	dzielenie
%	reszta z dzielenia (tylko int)
jednoargumentowy -	zmiana znaku
jednoargumentowy +	
++	zwiększenie
--	zmniejszenie

Operatory II

Nie ma operatora potęgowania!

Podstawienia

operator	działanie
----------	-----------

=	podstawienie $1 = p$
---	----------------------

$\odot =$	$a \odot = b$ jest równoważne $a = a \odot b$ (\odot to $+$, $-$, $*$, $/$, ...)
-----------	---

Porównania

operator	działanie
==	czy równe
>	czy większe
<	czy mniejsze
<=	mniejsze równe
>=	większe równe
!=	nierówne

Logiczne

operator	działanie
----------	-----------

&&	AND
----	-----

	OR
--	----

!	NOT
---	-----

Operatory V

W języku C nie ma typu logicznego!

Bitowe

operator	działanie
&	bitowe ¹ I (AND)
	bitowe LUB (OR)
^	bitowa różnica symetryczna (XOR)
>>	przesuń bitowo w prawo
<<	przesuń bitowo w lewo
~	negacja bitowa (zera na jedyнки, jedyнки na zera)

Operatory VI

Operacje na bitach można wykonywać na zmiennych i stałych typu całkowitego!

Inne

operator	działanie
()	operator grupowania
,	operator serii
[]	element tablicy
.	
->	
jednoargumentowy &	
jednoargumentowy *	
?:	wyrażenie warunkowe

Operatory VII

`sizeof()` rozmiar

¹To znaczy wykonywane bit po bicie na wszystkich bitach.

Rzutowanie

1. W sytuacji gdy po obu stronach operatora są zmienne/stałe różnych typów uruchamiany jest proces „rzutowania” (*casting*).
2. W większości przypadków efekty automatycznego rzutowania są zgodne z oczekiwaniami.
3. Czasami nie.
4. Można wymusić zmianę typu

```
int a, b;
```

```
double x;
```

```
...
```

```
x = a / b; // automatyczne rzutowanie podczas podstawienia
```

```
x = (double) a / (double) b; // rzutowanie argumentów operacji
```




To teraz może jakiś program?



Potrzebować jeszcze będziemy... I

1. Sposobu na wyświetlanie informacji z wewnątrz programu.

Służy do tego funkcja `printf()`

```
printf("Ala ma kota"); // produkuje napis "Ala ma kota"
```

```
double x = 4.5;
```

```
printf("x = %f", x); // wyprodukuje napis x = wstawiając z
```

```
// %f wartość zmiennej x
```

2. Instrukcji warunkowej, która w zależności od wartości zmiennej będzie rozgałęziała program...

Jest nią instrukcja `if`



Potrzebować jeszcze będziemy... II

```
if (x == 0)
{
    printf("x jest zero");
}
else if (x < 0)
{
    printf("x jest ujemne");
}
else
{
    printf("x jest dodatnie");
}
```

Jej idea wydaje się prosta. Te nawiasy klamrowe służą do wskazania miejsca na wpisywanie kodu wykonywanego po spełnieniu warunku.

Potrzebować jeszcze będziemy... III

3. Funkcji do obliczania pierwiastka kwadratowego. Nazywa się ona `sqrt()` i wymaga biblioteki funkcji matematycznych

```
#include <math.h>
```



Równanie kwadratowe I

1. Dane są trzy liczby a , b , c będące współczynnikami wielomianu

$$ax^2 + bx + c = 0$$

2. W ogólnym przypadku te trzy liczby/zmienne powinny być typu `double`

```
double a, b, c;
```

```
a = 1;
```

```
b = 2;
```

```
c = 1;
```

potrzebować jeszcze będziemy zmiennych

```
double x, x1, x2;
```



Równanie kwadratowe II

3. Znamy metodę rozwiązywania
trzeba zacząć od wyliczenia „ Δ ”

$$\Delta = b^2 - 4ac$$

```
double Delta;  
Delta = b * b - 4 * a * c;
```

4. W zależności od wartości Δ podejmujemy odpowiednie działania.
Użyjemy instrukcji warunkowej `if`



Równanie kwadratowe III

```
if (Delta == 0)
{
    x = -b/(2 * a);
    printf("Jest jeden pierwiastek x = %f", x);
}
else if (Delta < 0)
{
    printf("Nie ma pierwiastków rzeczywistych");
}
else
{
    x1 = (-b - sqrt(Delta))/(2 * a);
    x2 = (-b + sqrt(Delta))/(2 * a);
    printf("Sa dwa pierwiastki rzeczywiste: x1 = %f i
```



Równanie kwadratowe IV

```
}      x2 = %f", x1, x2);
```




Pętle

Pętle

- ▶ Komputery najlepiej się sprawdzają wszędzie tam, gdzie trzeba wielokrotnie wykonywać nużące czynności.
- ▶ Stąd każdy język programowania ma **kilka** konstrukcji ułatwiających powtarzanie czynności: instrukcje do konstruowania pętli.

Tablice

- ▶ Najprostsza instrukcja pętlowa stosowana głównie do przetwarzania tablic.
- ▶ Tablica to zmienna złożona mogąca przechowywać wiele wartości tego samego typu.
- ▶ Dostęp do każdego elementu tablicy odbywa się przez podanie indeksu.

indeks	0	1	2	3	4	5	6	7	8	9
wartość	33	43	-11	7	1	0	22	1023	9	11

- ▶ Tablica ma nazwę. Niech to będzie T . $T[2]$ to element tablicy T o indeksie 2.
- ▶ Jeżeli zmienna (typu `int`) i ma wartość 5, to dostęp do i -tego elementu tablicy można zapisać jako $T[i]$.
- ▶ Gdy i będzie się zmieniało w zakresie od 0 do 9 — uzyskamy dostęp do każdego z elementów tablicy

for l

- ▶ Polecenie `for` najłatwiej wykorzystać do przeglądania elementów tablicy:
- ▶ Wygląda ona tak

```
for(początek ; koniec ; przyrost)  
{  
  
}
```

- ▶ *początek*: wartość początkowa indeksu
- ▶ *koniec*: warunek kontynuacji pętli
- ▶ *przyrost*: z jakim krokiem zmienia się indeks
- ▶ w nawiasach klamrowych podaje się kod, który będzie wykonywany w każdym obiegu pętli



for II

- ▶ Żeby nadać wartość zero wszystkim elementom tablicy T użyjemy takiego kodu:

```
for(int i = 0; i < 10; i = i + 1)
{
    T[i] = 0;
}
```

- ▶ Oczywiście polecenia można użyć w dowolny inny sposób pamiętając, że
 1. najpierw zostanie (jednokrotnie) wykonane polecenie *początek*
 2. następnie sprawdzany jest czy spełniony jest *warunek*; jeżeli tak — przechodzimy do kroku 3; w przeciwnym razie pętla kończy pracę
 3. następnie wykonane zostaną wykonane wszystkie polecenia w nawiasach klamrowych,
 4. po ich wykonaniu wykonane zostanie polecenie *przyrost*



for III

5. przechodzimy do kroku 2

while I

1. Pewnym uogólnieniem polecenia for jest pętla while

2. Ogólna jej konstrukcja wygląda tak

```
while (warunek)
```

```
{
```

```
}
```

3. Najpierw sprawdza się czy spełniony jest *warunek*

4. Jeżeli tak wykonywany jest kod zawarty w nawiasach klamrowych; jeżeli nie — pętla kończy pracę

5. Każdorazowo po wykonaniu kodu w nawiasach sprawdzane jest spełnienie warunku — warunkuje on powtarzanie pętli.



while II

Przykład 1

```
double x =1.;  
while (x > 0.)  
{  
    x = x / 2.;  
}
```


while III

Uwaga

Jeżeli α , β , γ i δ są pewnymi poleceniami to polecenie

```
for( $\alpha$ ;  $\beta$ ;  $\gamma$ )
```

```
{
```

```
     $\delta$ 
```

```
}
```

jest równoważne

```
 $\alpha$ ;
```

```
while( $\beta$ )
```

```
{
```

```
     $\delta$ ;
```

```
     $\gamma$ ;
```

```
}
```



while IV

Przykład 2

```
for(double x = 1.; x > 0.; x /= 2.);
```

realizuje tę samą funkcję co Przykład 1

do

- ▶ Ostatnim rodzajem pętli jest pętla do.
- ▶ Jest bardzo podobna do pętli while, tyle że warunek jest sprawdzany po wykonaniu „tego co w nawiasach”

```
do{
```

```
} while (warunek)
```

- ▶ Oznacza to, że zawartość nawiasów zostanie wykonana **co najmniej raz**
- ▶ Niektóre języki programowania nie mają tej instrukcji pętlowej

Przykład

```
int x = 0;  
do  
{  
    x = x + 1;  
} while (x > 0);  
printf("x = %d", x);
```



break

- ▶ Polecenie `break` nie służy do tworzenia pętli
- ▶ Jest ono wykorzystywane do **natychmiastowego** przerwania wykonywania pętli
- ▶ Zazwyczaj jego użycie uwarunkowane jest spełnieniem jakiegoś warunku

```
for (i = 1; i < N; i = i + 1)
{
    if (B[i] == 0)
        break;
    C[i] = A[i] / B[i];
}
```

Polecenie nie może być używane poza pętlą^{^^}



continue

- ▶ Podobnie jak `break`, `continue` służy do zarządzania obliczeniami wewnątrz pętli.
- ▶ Użycie go powoduje natychmiastowe przejście na „koniec tego co w nawiasach”

```
for (i = 1; i < N; i = i + 1)
{
    if (C[i] < 0)
        continue;
    C[i] = . . .
}
```

Polecenie nie może być używane poza pętlą!

goto I

- ▶ W słownych opisach wielu algorytmów pojawia się konstrukcja typu: *jeżeli spełniony jest warunek, przejdź do punktu 3* (trójka wybrana jako jakiś przykład)
- ▶ Język C wyposażony jest w polecenie realizujące operację „przejdź do”; jest to polecenie `goto`
- ▶ Może być ono użyte do realizowania pętli (wraz z poleceniem `if`)



goto II

Przykład

Zwarty przykład

```
for(double x = 1.; x > 0.; x /= 2.);
```

może być zrealizowany tak:

```
double x = 1;
```

```
poczatek:                               // etykieta
```

```
if (x <= 0) goto koniec;
```

```
x = x / 2.;
```

```
goto poczatek;
```

```
koniec:                                  // etykieta
```