
# THE C PROGRAMMING LANGUAGE

Język C — próba podsumowania

wer. 10 z drobnymi modyfikacjami!

Wojciech Myszka

Katedra Mechaniki, Inżynierii Materiałowej i Biomedycznej

2023-06-12 08:59:30 +0200

# Ogólne

1. Bardzo już stary — powstał w latach 1969–1973.
2. Język „wysokiego poziomu”.
3. Zaliczany do grupy języków „proceduralnych” a czasami „imperatywnych strukturalnych języków programowania”.
4. Bardzo popularny: ciągle opierają się na nim wszystkie dystrybucje uniksopodobne (w tym i Linux).
5. Microsoft preferuje język C++.

Politechnika Wrocławska

# Słowa kluczowe

1. Słowa kluczowe są **zastrzeżone**.
2. Jest ich stosunkowo niewiele:

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

3. Tymi czerwonymi się nie zajmowaliśmy!
4. Komentarz
   /∗ *dowolny napis* ∗/
   czasami też
   // *napis*

Politechnika Wrocławska

# Podstawowe typy danych

| | | |
|---|---|---|
| **char** | 1 bajt | |
| **unsigned char** | 1 bajt | |
| **signed char** | 1 bajt | |
| **int** | 2 lub 4 bajty | |
| **unsigned int** | 2 lub 4 bajty | |
| **short int** | 2 bajty | |
| **unsigned short int** | 2 bajty | |
| **long int** | 4 bajty | |
| **unsigned long int** | 4 bajty | |
| **long long int** | 8 bajtów | tylko w nowych wersjach |
| **unsigned long long int** | 8 bajtów | tylko w nowych wersjach |
| **float** | 4 bajty | |
| **double** | 8 bajtów | |
| **long double** | 8, 10 lub 12, a nawet 16 bajtów | |
| **void** | | |

# Operacje

Wszystkie operacje (operatory) dostępne w C można podzielić na następujące grupy:

1. arytmetyczne
2. przypisania
3. logiczne
4. bitowe (działające na bitach)
5. pozostałe

Politechnika Wrocławska

# Operatory arytmetyczne

- dodawanie $+$
- odejmowanie $-$
- mnożenie $*$
- dzielenie $/$
- reszta z dzielenie % (modulo) (Tylko dla liczb typu całkowitego!)
- zwiększenie $++$
- zmniejszenie $--$

# Operatory przypisania

1. Oprócz najzwyklejszego operatora przypisania ($=$) używanego w kontekście:

$$a = b$$

co czytamy *zmiennej a przypisz wartośc zmiennej b*, czyli od prawej do lewej!

2. Występują operatory „złożone”
$+=$ $-=$ $*=$ $/=$ $\%=$ $<<=$ $>>=$ $\&=$ $\hat{}=$ $|=$
stosowane w następujący sposób ($\odot$ oznacza jeden z symboli $+, -, *, / \ldots$)

$$a \odot= b$$

co czyta się

$$a = a \odot b$$

# Operatory logiczne

1. == równy
2. ! = nie równy
3. > większy
4. < mniejszy
5. >= większy lub równy
6. <= mniejszy lub równy
7. && logiczne I (AND)
8. || logiczne LUB (OR)
9. ! logiczne NIE (NOT)

# Operatory logiczne

## Uwagi

1. W języku C **nie ma** typu logicznego!
2. Zatem operator może i jest typu **logicznego**, ale zwraca wartości **arytmetyczne**!
3. Z definicji numeryczną wartością wyrażenia logicznego lub relacyjnego jest **1** jeżeli jest ono prawdziwe lub **0** jeżeli nie jest prawdziwe.
4. W operatorach logicznych **każda wartość różna od zera** traktowana jest jako prawda; zero to fałsz.
5. Operatory logiczne mają priorytet niższy od operatorów arytmetycznych; dzięki temu wyrażenie `i < lim-1` jest rozumiane właściwie jako `i < (lim-1)`.

# Operatory logiczne

## && i ||

1. Wyrażenia połączone tymi operatorami oblicza się od strony lewej do prawej.
2. Koniec obliczania następuje natychmiast po określeniu wyniku jako „prawda” lub „fałsz”.
3. Wiele programów korzysta z tego faktu. (Hakerstwo!).
4. Priorytet operatora && jest wyższy od priorytetu operatora ||.
5. Priorytety obu operatorów są niższe od priorytetów operatorów relacji i porównania.

Politechnika Wrocławska

# Operatory inne

1. `sizeof()` wielkość obiektu/typu danych
2. & Adres (operator jednoargumentowy)
3. $*$ Wskaźnik; operator „wyłuskania” (jednoargumentowy)
4. ? Wyrazenie warunkowe
5. : Wyrazenie warunkowe
6. , Operator serii

# Zmienne I

1. Deklaracja zmiennej — bardzo prosta:

```
typ nazwa;
```

2. Nazwa musi zaczynać się od litery, może zawierać również cyfry i znak podkreślenia (który traktowany jest jak litera).
3. Typy pochodne:
   3.1 typ wyliczeniowy:

```
enum nazwa { jeden, dwa };
```

```
enum miesiac {
    STY =  1, LUT =  2, MAR =  3, KWI =  4,
    MAJ =  5, CZE =  6, LIP =  7, SIE =  8,
    WRZ =  9, PAZ = 10, LIS = 11, GRU = 12
}
```

Politechnika Wrocławska

# Zmienne II

3.2 struktury:

```
struct nazwa {
    typ1 nazwa1;
    typ2 nazwa2;
};
```

- Pola bitowe

```
typ [identyfikator] : dlugosc;
```

3.3 Unie

```
union nazwa {
    typ1 nazwa1;
    typ2 nazwa2;
};
```

3.4 Tablice

```
typ nazwa[liczba];
```

# Zmienne III

## 3.5 Wskaźniki

```
typ *nazwa;
typ **nazwa;
typ_zwracany (*nazwa_wsk_do_funkcji)\
                        (typ nazwa_param1,\
                         typ nazwa_param2,...);
```

# Zmienne i typy — cd

- **sizeof**(<*zmienna*>)
- **sizeof**(<*typ*>)
- **sizeof**(<*stała*>)

# Zmienne i typy — cd

## Logiczne

1. Typ logiczny nie istnieje (w podstawowej wersji języka C — ANSI)!
2. Istnieją operatory, które normalnie powinny dawać wynik logiczny: `&&` i `||`
3. Istnieją polecenia, które gdzie indziej korzystają ze zmiennych i wyrażeń typu logicznego.
4. Prawda (true)
   - na „wejściu” (argument!) każda wartość różna od zera
   - na „wyjściu” (wynik) jeden (zazwyczaj – nie zawsze!)
5. Fałsz (false)
   - na wejściu i wyjściu — zero

Politechnika Wrocławska

# Zmienne i typy – różne takie

1. Podstawowy typ zmiennoprzecinkowy: **double**
2. To jest stała typu **double**: 3.141592365
3. To jest stała typu **float**: 3.141592365F
4. To jest stała typu **int**: 1234
5. To jest stałą typu **unsigned**: 1234U
6. To jest stała typu **long**: 1234L
7. To jest stała tycpu **char**: "Ala ma kota”
8. To jest stałą typu **wchar_t**: L"Ala ma małpę"

Politechnika Wrocławska

# Reprezentacja binarna I

1. Każdy zmienna ma:
   - nazwę,
   - typ,
   - wartość;
2. Typ zmiennej definiuje sposób w jaki przechowywane są w pamięci jej wartości.
3. **Wszystkie** wartości przechowywane są w postaci binarnej.
4. Nazywa się to reprezentacją binarną.

# Reprezentacja binarna II

## char

Wartości **char** przechowywane są jako jednobajtowa liczba binarna o wartości równej kodowi ASCII znaku.
Zmienne typu **char** mogą być używane do przechowtywania niewielkich liczb całkowitych.

# Reprezentacja binarna III

**w_char**

Wartości **char** przechowywane są jako czterobajtowa liczba binarna o wartości równej kodowi UNICODE znaku.

# Reprezentacja binarna IV

## int

Wartości **int** przechowywane są jako liczba binarna używając systemu kodowania U2.

- **short** — 2 bajty
- **int** — 4 bajty
- **long** — 8 bajtów

# Reprezentacja binarna V

## unsigned int

Wartości **unsigned int** przechowywane są jako liczba binarna.
Ponieważ „zwykły” typ **int** używa kodowania U2, operacje arytmetyczne w przypadu obu typów wykonywane są w identyczny sposób; różnica pojawia się podczas intrpretacji rezultatu (konwersji do postaci dziesiętnej).

# Reprezentacja binarna VI

## float, double

Wartości **int** przechowywane są jako liczba binarna używając systemu kodowania opisanego w normie IEEE 754.

- **float** — 4 bajty
- **double** — 8 bajtów

Politechnika Wrocławska

# Konwersje typów (rzutowanie)

1. Niejawne (czyli niezadeklarowane):
   - Typ „mniejszy" jest **promowany** do „większego" w wyrażeniach dwuargumentowych (gdy argumenty mają różny typ!).
   - Typem wyniku jest typ „większy" (ale obliczenia wykonywane są — jeżeli nie ma istotnej potrzeby — bez dokonywania konwersji).
   - Kłopoty dla typów unsigned (omijam!).
   - Obiekt znakowy zmienia się w liczbę całkowitą (co ze znakiem??).
   - Dłuższe liczby całkowite są przekształcane do krótszych przed odcięcie „wystających" **znaczących** bitów.
2. Jawne (zadeklarowane):
   - wygląda tak (*nazwa typu*)*wyrażenie*
   - Nazywa się to rzutem (ang: *cast*)

# Instrukcje sterujące I

1. Instrukcja **if**

```
if (warunek1) {
    instrukcje;
}
else if(warunek2){
    instrukcje;
}
else {
    instrukcje;
}
```

# Instrukcje sterujące II

Instrukcja if to podstawowa instrukcja warunkowa w C — gdy warunek1 jest spełniony (**zwraca wartość niezerową**), wykonany zostanie kod zawarty w bloku ograniczonym klamrami. Instrukcje **else if** i **else** są opcjonalne, sprawdzane są wyłącznie, gdy podstawowy warunek nie jest spełniony. Wykorzystywana jest idea „drogi na skróty". Gdy wyrażenie logiczne wygląda tak:

```
A && B && C && D
```

lub tak

```
A || B || C || D
```

obliczenia prowadzone są tak długo, żeby móc jednoznacznie określić wynik.

# Instrukcje sterujące III

2. Pętla **while**

```
while (wyrazenie) {
 instrukcje;
}
```

Pętla while — instrukcja wykonuje kod zawarty w bloku ograniczonym klamrami tak długo, dopóki jej warunek jest spełniony (ma wartość różną od zera). Instrukcja sprawdza warunek przed wykonaniem ciała pętli. Pętla **while** może wykonywać się nieskończoną ilość razy, gdy wyrażenie nigdy nie przyjmie wartości 0, może także nie wykonać się nigdy, gdy wartość przed pierwszym przebiegiem będzie zerowa.

# Instrukcje sterujące IV

3. Pętla do...while

```
do {
instrukcje;
}
while(warunek);
```

Pętla **do**...**while** jest podobna do pętli **while** z tą różnicą, że warunek sprawdzany jest po każdym wykonaniu pętli, a więc instrukcje w pętli zawsze wykonają się co najmniej raz.

# Instrukcje sterujące V

4. Pętla for

```
for (wyr1; wyr2; wyr3) {
  instrukcje;
}
```

Pętla **for** jest rozwinięciem pętli **while** o instrukcję wykonywaną przed pierwszym obiegiem oraz dodatkową instrukcję wykonywaną po każdym przebiegu — najczęściej służącą jako licznik obiegów. Często zmienną liczącą kolejne wykonania ciała pętli nazywa się iteratorem.
Powyższa instrukcja jest równoważna rozwinięciu:

```
wyr1;
while (wyr2){
   instrukcja
   wyr3;
}
```

# Instrukcje sterujące VI

5. Instrukcja switch

```
switch (wyrazenie) {
      case wartosc1 :
         instrukcje;
         [break;]
      case wartosc2 :
         instrukcje;
         [break;]
      default :
         instrukcje;
         [break;]
}
```

# Instrukcje sterujące VII

Instrukcją decyzyjną **switch** zastąpić można wielokrotne wywoływanie instrukcji warunkowej **if** np. dla różnych wartości tej samej zmiennej — przykładowo, gdy zmienna może przyjąć 10 różnych wartości, a dla każdej z nich należy podjąć inne działanie.
Należy pamiętać o **break**.

Politechnika Wrocławska

# Funkcje

1. Definicja

```
[klasa_pamięci] [typ] nazwa([lista_argumentów])
{
  instrukcje;
  [return wartość;]
}
```

   Klasa pamięci, określenie zwracanego typu oraz lista argumentów są opcjonalne. Jeżeli nie podano typu, domyślnie jest to typ liczbowy int, a instrukcję return kończącą funkcję i zwracającą wartość do funkcji nadrzędnej można pominąć. Listę argumentów tworzą wszystkie zmienne (zarówno przekazywane przez wartość jak i wskaźniki) wraz z określeniem ich typu. Dozwolona jest rekurencja, nie ma natomiast możliwości przeciążania funkcji.
2. Funkcja musi być zdefiniowana przed pierwszym jej użyciem (prototyp!).

Politechnika Wrocławska

# Obfuscated C code

## Co to jest?

```
_(__,___,____){___/__<=1?_(__,___+1,____):!(___%__)?_(__,___+1,0):___%__==___/
__&&!____?(printf("%d\t",___/__),_(__,___+1,0)):___%__>1&&___%__<___/__?_(__,1+
___,____+!(___/__%(___%__))):___<__*__?_(__,___+1,____):0;}main(){_(100,0,0);}
```

Politechnika Wrocławska

# Kolejność (priorytet) operatorów I

1. () [] -> .
2. ! ~ ++ -- + - * & **sizeof**
3. * / %
4. + -
5. << >>
6. < <= >= >
7. == !=
8. &
9. ^
10. |
11. &&
12. ||
13. ?:
14. = += -= *= /= %= &= ^= |= <<= >>=
15. ,

# Kolejność operatorów

## Operatory „przynależności”

1. Operator funkcji () (w05)
2. Tablica [] (w06)
3. Wskaźnik do struktury $->$ (w07, w08)
4. Element struktury . (w08)

# Kolejność operatorów

## Operatory jednoargumentowe (unarne)

1. ! Logiczne NIE
2. $\sim$ Uzupełnienie „do jednego” (bitowa zamiana 0 na 1 a 1 na 0)
3. $++$ Zwiększ (Uwaga: przyrostek i przedrostek!)
4. $--$ Zmniejsz (Uwaga: przyrostek i przedrostek!)
5. $+$ Po prostu: $x = +2$
6. $-$ Zmiana znaku liczby $x = -2$
7. $*$ Wskaźnik do zmiennej
8. & Pobranie adresu
9. **sizeof** Operator zwracający ilość miejsca (w bajtach) zajmowaną przez zmienną albo typ danych

# Kolejność operatorów

## Operatory dwuargumentowe (binarne)

1. $*$ Mnożenie (Co znaczy $2*+3$)
2. $/$ Dzielenie (Co znaczy $2/-3$)
3. % Modulo

Uwaga: Dzielenie nie wyprowadza poza typ! (w wyniku dzielenia dwu liczb całkowitych (**int**) zawsze dostaniemy wartość całkowitą).

# Kolejność operatorów

## Operatory dwuargumentowe (binarne)

1. $+$ Suma ($3{+}{+}3$ — ŹLE!! $3{+}\ {+}3$)
2. $-$ Różnica ($3{+}{-}3$)

# Kolejność operatorów

## Operatory bitowe

Po lewej stronie operatora co przesuwamy, po prawej o ile

1. $<<$ Bitowe przesunięcie w lewo
2. $>>$ Bitowe przesunięcie w prawo

Przesunięcie w lewo o jeden równoważne jest pomnożeniu przez dwa.
Przesunięcie w prawo o jeden równoważne jest podzieleniu przez dwa.
Uwaga: najlepiej działa z liczbami typu **unsigned int**.
$-3 >> 1$ w wyniku daje $-2$
$+3 >> 1$ w wyniku daje 1

# Kolejność operatorów

## Operatory relacji

1. <
2. <=
3. >=
4. >

# Kolejność operatorów

## Operatory relacji

1. == (Równe)
2. != (Różne)

**Do przemyślenia**

Jaki będzie wynik 3>5==5>7

# Kolejność operatorów

## Operatory bitowe

Operatory mają różne priorytety, wymienian w kolejności od najważnieszego do najmniej waznego

1. & (1 & 2 daje 0 bo 01 & 10)
2. $\hat{}$ (1 $\hat{}$ 2 daje 3 bo 01 $\hat{}$ 10)
3. | (1 | 2 daje 3 bo 01 | 10)

# Kolejność operatorów

## Operatory logiczne

Operatory mają różne priorytety, wymienione w kolejności od najważnieszego do najmniej waznego

1. && Iloczyn logiczny 4 && $-4$ daje 1
2. || Suma logiczna $-4$ || 0 daje 1

# Kolejność operatorów

## if then else

1. ?:

```
if ( x == 1 )
    y = 10;
  else
    y = 20;
```

jest równoważne

```
y = (x == 1) ? 10 : 20;
```

```
if ( x == 1 )
   puts("take car");
 else
   puts("take bike");
```

jest równoważne

```
(x == 1) ? puts("take car") :\
           puts("take bike");
```

albo

```
puts( (x == 1) ? "take car" :\
                 "take bike");
```

Politechnika Wrocławska

# Kolejność operatorów

## Operatory podstawienia

1. = operatr przypisania (podstawienia) a = b
2. += to znaczy a = a + b czyli a += b
3. −=
4. *=
5. /=
6. %=
7. &=
8. ^=
9. |=
10. <<=
11. >>=

# Kolejność operatorów

## Separator

1. , (przecinek) — oddziela parametry funkcji albo dane

# Wskaźnik do wskaźnika albo podwójny wskaźnik

1. Używany, na przykład wtedy, gdy chcemy aby funkcja zwracała jako jeden z parametrów wskaźnik
2. Przykład

```
#include <stdlib.h> /* malloc */

void Func(char **DoublePtr);

main()
{
    char *Ptr;
    Func(&Ptr);
}

void Func(char **DoublePtr)
{
    *DoublePtr = malloc(50);
}
```