Politechnika Wrocławska

Bardzo szybkie podsumowanie: wykład 1

wer. 10 z drobnymi modyfikacjami!

Wojciech Myszka

2023-03-05 21:05:09 +0100

HR EXCELLENCE IN RESEARCH

Uwagi

1. Obowiązuje cały materiał!
2. Tu tylko podsumowanie.

Politechnika Wrocławska

Informatyka I — ARM031007
Wprowadzenie do informatyki — MCM032102

wer. 15 z drobnymi modyfikacjami!

Wojciech Myszka

Katedra Mechaniki, Inżynierii Materiałowej i Biomedycznej

2022-03-03 20:21:16 +0100

HR EXCELLENCE IN RESEARCH

Politechnika Wrocławska

Program zajęć I

1. Wprowadzenie. Algorytm. Schematy blokowe. Idea programowania strukturalnego.
2. Struktura programów w C. Identyfikator, typy danych (typy fundamentalne: całkowite, rzeczywiste, znakowe, logiczny), deklaracja i inicjalizacja zmiennych, definiowanie stałych. Komunikacja poprzez konsolę. Operatory: arytmetyczne, logiczne, inkrementacji, dekrementacji, przypisania. Obliczanie wartości wyrażeń.
3. Struktury sterowania obliczeniami: rozgałęzienia i skoki, pętle pojedyncze i zagnieżdżone. Instrukcje proste i złożone; instrukcje warunkowe, wyrażenia warunkowe, instrukcje iteracyjne.
4. Preprocesor: dyrektywy, makrodefinicje.
5. Funkcje: budowa funkcji, argumenty funkcji, wynik wykonania funkcji, definicje i deklaracje globalne, argumenty funkcji main, rekurencja.

Politechnika Wrocławska

Program zajęć II

6. Tablice (tablice jedno i wielowymiarowe), łańcuchy znaków.
7. Wskaźniki. Pamięć dynamiczna.
8. **Kolokwium.**
9. Struktury danych, unie: deklaracja struktury, definiowanie zmiennej strukturalnej, tablice struktur, wskaźniki a struktury danych.
10. Operacje na plikach: otwieranie, zamykanie plików, czytanie i zapisywanie do plików.
11. Formatowanie w operacjach wejście/wyjście. Binarne wejście/wyjście.
12. Operacje na łańcuchach znaków.
13. Programowanie strukturalne w praktyce: podział programu na moduły, struktury danych, kompilacja.
14. Programy pomocnicze: diff, make, systemy rcs i cvs, debugger. Zarządzanie wersjami. Środowiska zintegrowane.
15. **Kolokwium zaliczeniowe.**

Po co uczyć się programowania? I

Pozwolę sobie przytoczyć tu 12 przewidywań na temat przyszłości programowania [31].

1. Procesory graficzne (GPU) będą naszymi następnymi procesorami.
2. Bardzo wiele przyszłego programowania dotyczyć będzie baz danych (*big data*).
3. JavaSript do wszystkiego.
4. Android na każdym urządzeniu.
5. Internet rzeczy — kolejne nowe platformy.
6. Open source na wiele sposobów.
7. Oparte na platformie WordPress aplikacje webowe.
8. Programy zostaną zastąpione przez „wtyczki” (plug-ins).
9. Niech żyją polecenia wydawane w terminalu!

Politechnika Wrocławska

Po co uczyć się programowania? II

10. Nie liczmy na dalsze upraszczanie językow programowania.
11. Programowaniem zajmować będą się programiści z krajów o najniższym koszcie pracy.
12. W dalszym ciagu szefostwo nie będzie rozumieć o co chodzi z tym programowaniem.

Politechnika Wrocławska

Slajdy

- Automatyka i Robotyka: https://kmim.wm.pwr.edu.pl/myszka/dydaktyka/informatyka-i/arm031007-w/
- Mechatronika: https://kmim.wm.pwr.edu.pl/myszka/dydaktyka/informatyka-i/mcm032101-w/

Bryk

Na podstawie slajdów powstał Bryk. Celowo nie nazywam go skryptem czy podręcznikiem. Można go znaleźć pod adresem: https://kmim.wm.pwr.edu.pl/myszka/wp-content/uploads/sites/2/2015/03/bryk.pdf. Odsyłacz jest na stronie wykładu.

Literatura I

- *A GNU Manual*, rozdział Nested Functions.
  Free Software Foundation, Inc., 2015.
- Programowanie w języku C, 2007.
  Wersja elektroniczna dostępna pod adresem:
  http://pl.wikibooks.org/wiki/Programowanie:C.
- David Griffiths, Dawn Griffiths.
  Head First C.
  Head First. O'Reilly, 2011.
  Są szanse, żeby ksiazka byłą dostępna pod tym adresem:
  http://proquestcombo.safaribooksonline.com/9781449335649.
  Można się o nią „dobijać" zgodnie z instrukcją na stronie biblioteki.

Politechnika Wrocławska

Literatura II

David Griffiths, Dawn Griffiths.
C. Rusz głową!
Helion, Gliwice, 2013.
Strona księgarni: http://helion.pl/ksiazki/c-rusz-glowa-david-griffiths-dawn-griffiths,cruszg.htm.

J. Grębosz.
Symfonia C++.
Kallimach, Kraków, 2000.

David Harel, Yishai Feldman.
Rzecz o istocie informatyki: algorytmika.
Klasyka informatyki. Wydawnictwa Naukowo-Techniczne, Warszawa, 2001, 2002, 2008.

Politechnika Wrocławska

Literatura III

George Heineman, Gary Pollice, Stanley Selkow.
Algorytmy. Almanach.
Helion, Gliwice, 2010.

Steve Holmes.
C programming.
http://www.imada.sdu.dk/~svalle/courses/dm14-2005/mirror/c/, 1995.

Ted Jensen.
A tutorial on pointers and arrays in C, Feb. 2000.
Dostępne jako
http://pweb.netcom.com/~tjensen/ptr/pointers.htm.

Literatura IV

- Rob Kendrick.
  Some dark corners of C, 2013.
- B. W. Kernighan, D. M. Ritchie.
  Język ANSI C.
  WNT, Warszawa, 2007.
- Ben Klemens.
  21st Century C.
  O'Reilly Media, 2012.
  http://shop.oreilly.com/product/0636920025108.do.

Politechnika Wrocławska

Literatura V


M. J. Kubiak.
Programuję w językach Turbo Pascal i C/C++: programowanie strukturalne z elementami programowania obiektowego.
Mikom, Warszawa, 2001.


Ciaran O'Riordan.
Learning GNU C.
Ciaran O'Riordan, 2002.


Nick Parlante.
Pointer basics, 1999.


Nick Parlante.
Pointers and memory, 2000.

Politechnika Wrocławska

Literatura VI


Nick Parlante.
Binary trees, 2001.


Nick Parlante.
The great tree-list recursion problem, 2001.


Nick Parlante.
Linked list problems, 2002.


Nick Parlante.
Essential C, 2003.


Nick Parlante, Julie Zelenski.
Unix programming tools, 2001.

Literatura VII


Richard M. Reese.
Understanding and Using C Pointers.
O'Reilly, 2013.


Richard M. Reese.
Wskaźniki w języku C: przewodnik.
Helion, Gliwice, 2014.
Dostęp po zalogowaniu w bazie NASBI.
http://biblioteka.pwr.wroc.pl/NASBI_Naukowa_Akademicka_Sieciowa_Biblioteka_Internetowa,161.dhtml.

Politechnika Wrocławska

Literatura VIII

Adam Sapek.
W głąb języka C.
Helion, Gliwice, lipiec 1993.
Wersja elektroniczna dostępna pod adresem:
`ftp://ftp.helion.pl/online/wglab/1-3.pdf`.

Herbert Schildt.
Leksykon C/C++.
Oficyna Wydawnicza LTP, Warszawa, 2002.

C. Sexton.
Język C to proste.
Wyd. RM, Warszawa, 2001.

Literatura IX

C. Sexton.
Programowanie w C++ — to proste.
RM, Warszawa, 2001.

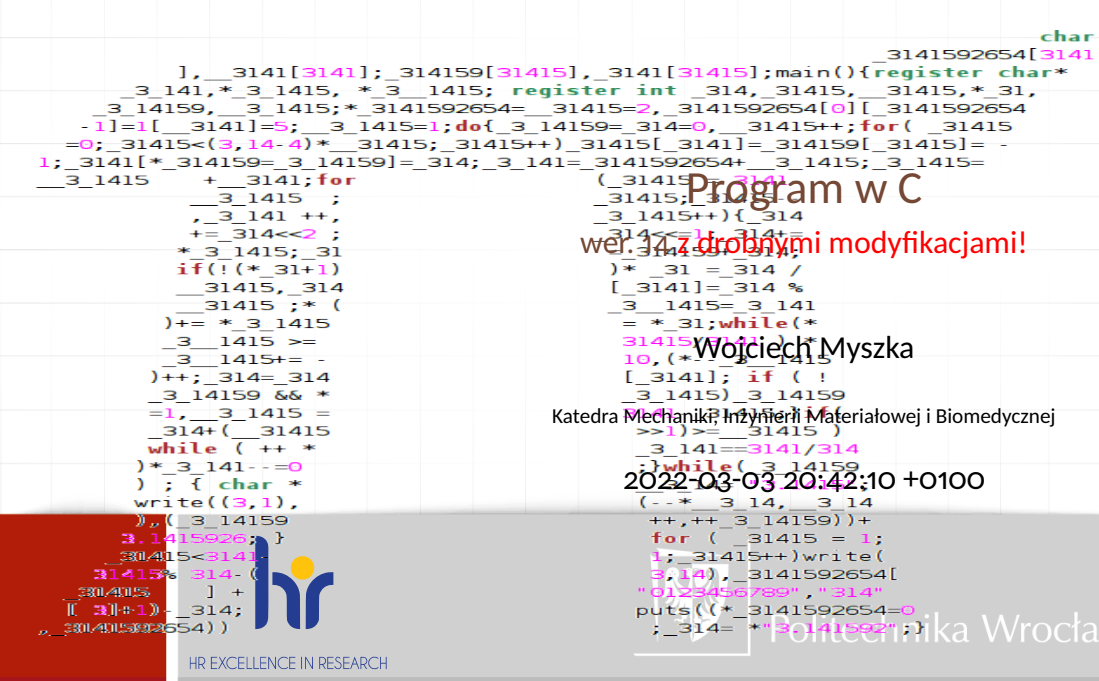
Piotr Stańczyk.
Algorytmika praktyczna: Nie tylko dla mistrzów.
Wydawnictwo Naukowe PWN, Warszawa, 2009.

K. Stec.
Wybrane elementy języka C.
Wyd. Pol. Śląskiej, Gliwice, 2001.

Clovis L Tondo, Scott E Gimpel, Danuta Kruszewska.
Język ANSI C: Ćwiczenia I Rozwiązania.
Wydawnictwa Naukowo-Techniczne, Warszawa, wydanie wyd. 2, 2004.

Literatura X

- Peter Wayner.
  12 predictions for the future of programming, Luty 2014.
- N. Wirth.
  Algorytmy + struktury danych = programy.
  WNT, Warszawa, 2001.
- Piotr Wróblewski.
  Algorytmy: struktury danych i techniki programowania.
  Helion, Gliwice, 2010.



Program w C

wer. 14 z drobnymi modyfikacjami!

Wojciech Myszka

Katedra Mechaniki Inżynierii Materiałowej i Biomedycznej

2022-03-03 20:42:10 +0100

A tak (ten sam) w Pascalu

```
program test;
{rozne deklaracje}
begin
        writeln('to jest program');
end.
```

Program składa się z:

1. Pewnej **struktury**;
2. **Poleceń** wykonywanych przez procesor;
3. Obiektów (zwanych **zmiennymi**) służących do przechowywania danych, wyników i wartości pośrednich uzyskanych podczas obliczeń;
4. **Stałych** używanych podczas obliczeń i do inicjowania wartości zmiennych;

Program

```c
/*
  Hello World in C, Ansi-style
*/
#include <stdio.h>
int main(void)
{
      int z;
      z = z + 1;
      puts("Hello World!"); // druk
}
```

Program

- komentarz

```
/*
  Hello World in C, Ansi-style
*/
#include <stdio.h>
int main(void)
{
      int z;
      z = z + 1;
      puts("Hello World!"); // druk
}
```

Program

- struktura

```c
/*
  Hello World in C, Ansi-style
*/
#include <stdio.h>
int main(void)
{
      int z;
      z = z + 1;
      puts("Hello World!"); // druk
}
```

Program

- struktura
  - deklaracje

```
/*
  Hello World in C, Ansi-style
*/
#include <stdio.h>
int main(void)
{
      int z;
      z = z + 1;
      puts("Hello␣World!"); // druk
}
```

Program

- polecenia

```c
/*
  Hello World in C, Ansi-style
*/
#include <stdio.h>
int main(void)
{
      int z;
      z = z + 1;
      puts("Hello World!"); // druk
}
```

Program

- identyfikatory (zmienne?)

```c
/*
  Hello World in C, Ansi-style
*/
#include <stdio.h>
int main(void)
{
      int z;
      z = z + 1;
      puts("Hello World!"); // druk
}
```

Program

- stałe

```
/*
  Hello World in C, Ansi-style
*/
#include <stdio.h>
int main(void)
{
      int z;
      z = z + 1;
      puts("Hello World!"); // druk
}
```

Program

- zmienne

```c
/*
  Hello World in C, Ansi-style
*/
#include <stdio.h>
int main(void)
{
      int z;
      z = z + 1;
      puts("Hello World!"); // druk
}
```

Program

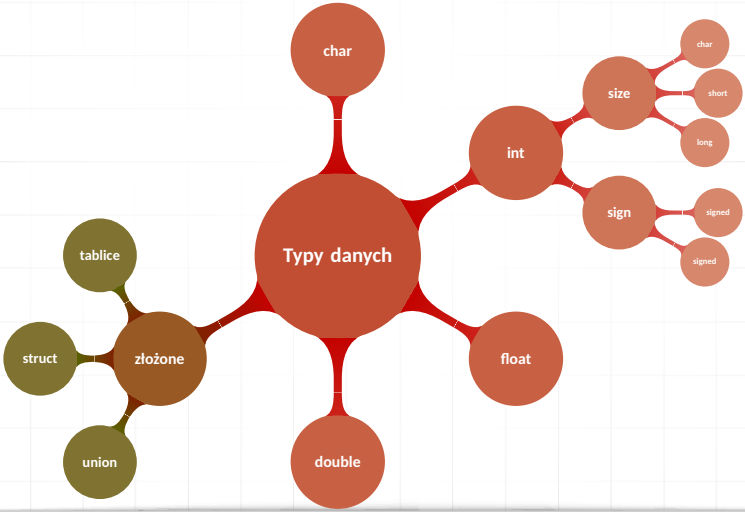
```c
/*
  Hello World in C, Ansi-style
*/
#include <stdio.h>
int main(void)
{
      int z;
      z = z + 1;
      puts("Hello World!"); // druk
}
```

- funkcje

Zmienne

1. Każdy program operuje na pewnych obiektach służących do przechowywania bieżących wartości.
2. Obiekty te nazywa się **zmiennymi**.
3. Każda zmienna musi mieć jakąś nazwę. **Pierwszym znakiem nazwy powinna być litera**, używać można również cyfr i znaków podkreślenia.
4. W odróżnieniu od różnych innych pojemników — te do przechowywania wartości rozróżniają typ przechowywanej wartości.
5. Jak zmienna przechowuje jedną wartość — nazywa się **zmienną prostą**, gdy potrafi przechować więcej wartości — **złożoną**.

Typy danych


Typy danych
char
int
size
char
short
long
sign
signed
signed
float
double
złożone
tablice
struct
union

Reprezentacja binarna: teksty

1. Każdy znak tekstu to jeden bajt.
2. Zmienna tych **char** zajmuje jeden bajt.
3. Reprezentacją binarną znaku ASCII jest liczba całkowita (bez znaku) o wartości równej kodowi ASCII tego znaku.

Reprezentacja binarna: liczby całkowite

1. Każdy liczba całkowita tekstu to jeden, dwa, cztery lub osiem bajtów…
2. Zmienna typu **int** zajmuje cztery bajty (czyli 32 bity).
3. Liczby całkowite (ze znakiem) reprezentowane są w zapisie uzupełnieniowym do dwu.
4. Najbardziej znaczący bit to bit znaku (1 oznacza liczbę ujemną, 0 dodatnią)
5. Liczby całkowite bez znaku reprezentowane są w kodzie binarnym.

Reprezentacja binarna: liczby zmiennoprzecinkowe

1. Każda liczba zmiennoprzecinkowa zajmuje 4, 8 lub 16 bajtów.
2. Dokładnie sposób zapisu liczb definiuje norma IEEE 754.
3. Stosuje się zapis wykładniczy w postaci $znak * mantysa * 2^{cecha}$.
4. Znak to jeden bit, cecha (wykładnik) to 8, 11 albo 15 bitów (binarna liczba całkowita ze znakiem); mantysa (ułamek binarny) — pozostałe bity.
5. Liczby zapisywane są w postaci znormalizowanej (przed kropką dziesiętną musi być 1 — cyfra różna od zera).
6. Specjalne „wartości” $+\infty$, $-\infty$ oraz NaN (Not a Number) mają również swoje reprezentacje binarne; takie wyniki mogą pojawić się w przypadku „dzielena przez zero”.

Nazwy zmiennych

1. Każdy identyfikator musi być unikatowy!
2. Wielkie i małe litery są ważne.
3. Nazwy identyfikatorów powinny zaczynać się od litery i składać z liter i cyfr.
4. Znak _ (podkreślenie) traktowany jest jak litera i może służyć do tworzenia bardziej czytelnych nazw.
5. Znaku _ nie powinno się stosować na początku nazwy — bardzo wiele nazw systemowych zaczyna się od niego.
6. Nie należy używać polskich liter w nazwach.
7. Tradycyjnie zmienne pisze się małymi literami, a różnego rodzaju stałe — wielkimi.
8. Słowa kluczowe (**int**, **void**, **if**, **else**,...) są zarezerwowane i nie mogą być używane jako nazwy.

Słowa kluczowe języka C

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

Deklaracje zmiennych I

Język C zna następujące typy zmiennych:

1. **char** znakowe (elementarną jednostką informacji jest jeden znak).
2. **int** całkowite.
3. **float** zmiennoprzecinkowe („rzeczywiste”).
4. **double** zmiennoprzecinkowe, podwójnej precyzji.
5. **void** pusty typ; nie można zadeklarować zmiennej takiego typu, ale może być on wykorzystany do zwrócenia uwagi, że funkcja nie zwraca wartości lub, że nie przyjmuje parametrów.

Przykład deklaracji:
typ nazwa;

Deklaracje zmiennych II

```
int a;
char b;
float c;
double d;
```

Wyprowadzanie wartości zmiennych

printf("Jakiś tekst %*cośtam* tekst", *zmienna*);

Teraz będzie nudno…

…ale bez tego się nie da.

Omówię wszystkie typy

1. wskazując jakie dane przechowują
2. opisując jak wyglądają stałe omawianego typu
3. mówiąc jakie operacje można na nich wykonywać
4. w jaki sposób wyprowadzamy wartości tych typów

char I

1. Zajmuje jeden bajt w pamięci komputera.
2. Typ służący do przechowywania znaków (ewentualnie liczb z zakresu $0 \ldots 255$ albo $-128 \ldots 127$).
3. Każdy znak przechowywany jest jako odpowiadający mu kod ASCII.
4. Znaki zapisujemy w pojedynczych cudzysłowach (w odróżnieniu od tekstów — gdzie cudzysłowy są podwójne).
5. `'a' ; 'A' ; '7' ; '!' ; '$'`
6. Znaki „specjalne” (trudne do uzyskania z klawiatury!):
   - `'\a'` — alarm (sygnał akustyczny terminala),
   - `'\b'` — backspace (usuwa poprzedzający znak),
   - `'\f'` — wysuniecie strony (np. w drukarce),
   - `'\r'` — powrót kursora (karetki) do początku wiersza,
   - `'\n'` — znak nowego wiersza,

char II

- `'\"'` — cudzysłów,
- `'\''` — apostrof,
- `'\\'` — ukośnik wsteczny (backslash),
- `'\t'` — tabulacja pozioma,
- `'\v'` — tabulacja pionowa,
- `'\?'` — znak zapytania (pytajnik),
- `'\ooo'` — liczba zapisana w systemie oktalnym (ósemkowym), gdzie 'ooo' należy zastąpić trzycyfrową liczbą w tym systemie,
- `'\xhh'` — liczba zapisana w systemie heksadecymalnym (szesnastkowym), gdzie 'hh' należy zastąpić dwucyfrową liczbą w tym systemie,
- `'\unnnn'` — uniwersalna nazwa znaku, gdzie 'nnnn' należy zastąpić czterocyfrowym identyfikatorem znaku w systemie szesnastkowym. 'nnnn' odpowiada dłuższej formie w postaci 'oooonnnn',
- `'\unnnnnnnn'` — uniwersalna nazwa znaku, gdzie 'nnnnnnnn' należy zastąpić ośmiocyfrowym identyfikatorem znaku w systemie szesnastkowym.

char III

7. Napisy przechowujemy w tablicach typu znakowego! — Zmienne złożone.

Działania na zmiennych znakowych

- Takie jak na zmiennych całkowitych!
- Działania wykonywane są na wartościach ASCII odpowiadających literom.

Przykład

```
char a = 'B';
char b = a / 2;
char c = 'Z'/2;
char d = a + 1;
```

Drukowanie zmiennej znakowej

1. putchar(zmienna_znakowa);
2. albo
   printf ("Wartość␣zmiennej␣znakowej␣wynosi␣=␣ %c\n",␣zmienna_znakowa);

Przykład

```
#include <stdio.h>
int main ()
{
    char a = 'B';
    char b = a / 2;
    putchar(a);  // albo printf('%c', a);
    putchar('\n');
    putchar(b);  // albo printf('%c', b);
    putchar('\n');  // albo printf('\n', a);
    return 0;
}
```

int

1. Typ służy do przechowywania liczb całkowitych
2. Stałe całkowite:
   - Liczby można zapisywać w układzie dziesiętnym: `1; 123; 48; -579` itd.,
   - Liczby można zapisywać w układzie ósemkowym (zaczynając od zera): `01; 0123; 047; 0579` — w ostatnim przypadku będzie błąd!
   - Liczby można zapisywać w układzie szesnastkowym (zaczynając od `0x`): `0xffff, 0xabcdef, 0Xabcd, 0xABCD`.

Działania na liczbach całkowitych

1. Dodawanie (nic ciekawego): +,
2. Odejmowanie (nic ciekawego): −,
3. Mnożenie (nic ciekawego): *,
4. Dzielenie (uwaga na kłopoty): /,
   - wynik dzielenia dwu liczb typu całkowitego jest zawsze liczbą całkowitą!
   - 5 / 2 w wyniku daje 2;
   - 2 / 3 * 3 daje w wyniko o (zero!);
5. Reszta z dzielenia (modulo): ,
6. Działania na bitach:
   - iloczyn bitowy: & (&),
   - suma bitowa: |,
   - różnica symetryczna: ^,
   - negacja: ~.

Dzielenie liczb całkowitych

I jeszcze raz: Są dwie operacje: / \%.

W przypadku argumentów całkowitych:

- Dzielenie wykonywane jest jako dzielenie całkowitoliczbowe!
- $3/5$ daje w wyniku 0
- $3/5 * 5$ daje w wyniku 0 (bo operacje wykonywane są od lewej do prawej z powodu jednakowych priorytetów mnożenia i dzielenia).
- \% oznacza resztę z dzielenia całkowitoliczbowego

$$7 \,\%\, 3 = 1$$

bo

$$7 = 2 * 3 + 1$$

oraz

$$3 \,\%\, 7 = 3$$

bo

$$3 = 0 * 7 + 3$$

Jeszcze o dzieleniu liczb całkowitych

Jeszcze parę uwag o dzieleniu całkowitoliczbowym. Popatrzmy na taki program:

```
#include <stdio.h>
int main ( )
{
 for ( int i = b; i >= o; i-- )
   {
    c = a / i;
    if ( c != poprzednie )
      {
       poprzednie = c;
       printf ( "%d/%d=%d\n", a, i, c );
      }
   }
 return o;
}
```

Wyniki programu

```
50/100=0
50/50=1
50/25=2
50/16=3
50/12=4
50/10=5
50/8=6
50/7=7
50/6=8
50/5=10
50/4=12
50/3=16
50/2=25
50/1=50
Floating point exception (core dumped)
```

Operatory bitowe

1. & I (AND),
2. | LUB (OR),
3. ^ różnica symetryczna (exclusive OR),
4. << przesuń w lewo,
5. >> przesuń w prawo,
6. ~ dopełnienie do jednego (przełącza wartość bitu: z 0 na 1, a z 1 na 0).
   Operator jednoargumentowy!

Argumenty **całkowitoliczbowe** traktowane cą jako ciągi bitów, a operacje wykonywane są na poszczególnych bitach.

Wyprowadzenie wartości całkowitej

1. Format dziesiętny: %d (%l — long, %h — short, %u — unsigned);
2. Format ósemkowy: %o;
3. Format szesnastkowy: %x;

`float`

1. Typ służy do przechowywania danych zmiennoprzecinkowych (zmiennopozycyjnych) czyli, tak zwanych rzeczywistych.
2. Liczby zapisujemy z **kropką** dziesiętną: `1.5f; 23.387f; 3.f`
3. Albo używając zapisu „naukowego”: `3e6f; 2.34E8f; 3.14e-5F`
4. Zwracam uwagę na literę f (F) na końcu! Bez niej stałe zmiennoprzecinkowe są traktowane jako typu **double**.
5. Zapis liczb zgodny z normą IEEE 754 na 32 bitach:
   - 7,22 cyfr dziesiętnych (23 bity),
   - zakres $10^{38,23}$.

Dwa programy

```c
#include <stdio.h>
int main()
{
    float x = 10.3f;
    printf("%30.20g\n", x);
    return ( 0 );
}
```

```
10.300000190734863281
```

Dwa programy

```c
#include <stdio.h>
int main()
{
    float x = 10.3f;
    printf("%30.20g\n", x);
    return ( 0 );
}
```

```
10.300000190734863281
```

```c
#include <stdio.h>
int main()
{
    double x = 10.3;
    printf("%30.20g\n", x);
    return ( 0 );
}
```

```
10.300000000000000711
```

Dwa programy

```c
#include <stdio.h>
int main()
{
    float x = 10.3f;
    printf("%30.20g\n", x);
    return ( 0 );
}
```

```
10.300000190734863281
```

```c
#include <stdio.h>
int main()
{
    double x = 10.3;
    printf("%30.20g\n", x);
    return ( 0 );
}
```

```
10.300000000000000711
```

O liczbach typu **float** zapomnijmy!

double

1. Typ służy do przechowywania danych zmiennoprzecinkowych podwójnej precyzji. Liczba zajmuje podwójną ilość miejsca ale też jest dokładniejsza (więcej cyfr po przecinku).
2. Zapis liczb zgodny z normą IEEE 754 na 64 bitach:
   - 15,95 cyfr dziesiętnych (53 bity),
   - zakres $10^{307,95}$.
3. Oprócz typu **double** jest jeszcze **long double** (16 bajtów, 128 bitów). Stałe muszą być wyposażone w przyrostek L.

long double

```
#include <stdio.h>
int main()
{
    long double x = 10.3L;
    printf("%35.30Lg\n", x);
    return ( 0 );
}
```

```
10.300000000000000000001734723476
```

Operacje zmiennoprzecinkowe

1. dodawanie,
2. odejmowanie,
3. mnożenie,
4. dzielenie.

Wyprowadzanie liczb zmiennoprzecinkowych

1. format stałoprzecinkowy:
   %f (albo %*szerokosc_pola*.*liczba_cyfr_po_przecinku***f**) na przykład %10.3f,
2. format zmiennoprzecinkowy: %e,
3. format „mieszany”: %g.

Wyprowadzanie liczb zmiennoprzecinkowych

```c
#include <stdio.h>
int main()
{
    double y, x = 1.23456789;
    y = x;
    int i = 0;
    while ( i < 10 )
    {
        printf("%15g\t  %15g\n", x, y);
        x = x * 10;
        y = y / 10;
        i++;
    }
    return 0;
}
```

Modyfikatory (kwalifikatory) typów

- **short** — krótki
- **long** — długi

oraz

- **signed** — ze znakiem
- **unsigned** — bez znaku

Zatem **int**, **short int**, **long int**, **unsigned int**, unsignet **short int** i **unsigned long int** to zazwyczaj zupełnie różne typy danych (przechowują liczby z różnych zakresów.

Jak się w tym wszystkim połapać?

1. Zapewne nie warto…
2. Ale trzeba wiedzieć, że taki problem jest!
3. Można napisać malutki programik:

```
#include <stdio.h>

int main()
{
printf("sizeof(char␣␣)␣=␣%d\n", sizeof(char  ));
printf("sizeof(short␣)␣=␣%d\n", sizeof(short ));
printf("sizeof(int␣␣␣)␣=␣%d\n", sizeof(int   ));
printf("sizeof(long␣␣)␣=␣%d\n", sizeof(long  ));
printf("sizeof(float␣)␣=␣%d\n", sizeof(float ));
printf("sizeof(double)␣=␣%d\n", sizeof(double));
printf("sizeof(long␣double)␣=␣%d\n", sizeof(long double));
return 0;
}
```

Jak się w tym wszystkim połapać?

W naszym laboratorium wyniki tego programu są następujące:

```
sizeof(char   ) = 1
sizeof(short  ) = 2
sizeof(int    ) = 4
sizeof(long   ) = 8
sizeof(float  ) = 4
sizeof(double) = 8
sizeof(long double) = 16
```

Operatory arytmetyczne

- dodawanie: $+$
- odejmowanie: $-$
- mnożenie: $*$
- dzielenie: $/$
- reszta z dzielenie (modulo): $\%$ (tylko liczby całkowite!)
- zwiększenie $++$
- zmniejszenie $--$

Operatory arytmetyczne (cd) I

Zwiększenie

1. Występuje w dwu wariantach:
   - przedrostkowym $++x$
   - przyrostkowym $x++$
2. Na pierwszy rzut oka nie ma różnicy. Zarówno $++x$ jak i $x++$ oznacza tyle co $x = x + 1$.
3. W programie może wystąpić tak:

```
int x;
x++;
++x;
```

4. W takim kontekście, po wykonaniu operacji

Operatory arytmetyczne (cd) II

Zwiększenie

```
int x, z;
z = 6;
x = 2;
z = z/++x;
```

z przyjmie wartość **2**, a x przyjmie wartość 3. Najpierw zwiększane jest x, a później wykonywane dzielenie.

5. A w takim kontekście, po wykonaniu operacji

```
int x, z;
z = 6;
x = 2;
z = z/x++;
```

Operatory arytmetyczne (cd) III

Zwiększenie

z przyjmie wartość **3**, a x przyjmie wartość 3. Najpierw wykonywane jest dzielenie, a później zwiększane x.

Operatory arytmetyczne (cd)

Zmniejszenie

1. Występuje w dwu wariantach:
   - przedrostkowym $--x$
   - przyrostkowym $x--$
2. Na pierwszy rzut oka nie ma różnicy. Zarówno $--x$ jak i $x--$ oznacza tyle co $x = x - 1$.
3. $--x$ — najpierw zmniejszane jest x i taka wartość bierze udział w kolejnej operacji (jeżeli występuje).
4. $x--$ — aktualna wartość x bierze udział operacji (jeżeli taka występuje), a później zmniejszane jest x.

Operatory przypisania

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

$$a = b$$

   co czytamy *zmiennej a przypisz wartośc zmiennej b*

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)

Operatory logiczne

Uwagi

1. W języku C **nie ma** typu logicznego!
2. Z definicji numeryczną wartością wyrażenia logicznego lub relacyjnego jest **1** jeżeli jest ono prawdziwe lub **0** jeżeli nie jest prawdziwe.
3. W operatorach logicznych **każda wartość różna od zera** traktowana jest jako prawda.
4. 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.

Operatory logiczne

&& — przykład

```
i < lim-1 &&
(c = getchar()) != '\n' &&
c != EOF
```

1. Najpierw sprawdzamy czy `i < lim-1`.
2. Jeżeli nie — nie zostaną wykonane kolejne operacje.
3. Następnie czytany jest znak (`c = getchar()`)...
4. ...a w kolejnym kroku sprawdza się czy nie jest to znak nowej linii `'\n'` (Hakerstwo!)
5. W ostatnim kroku sprawdza się czy wcześniej wczytany znak nie jest znakiem końca pliku (EOF)

Operatory inne

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

Kolejność operacji

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

Type casting I

Zasady

1. Automatycznie dokonywane są jedynie konwersje nie powodujące straty informacji (np. zamiana liczby całkowitej do float w wyrażeniu f + i).
2. Pewne operacje są niedozwolone (np. indeksowanie tablicy wskaźnikiem typu float) i żadna konwersja nie będzie wykonana.
3. Operacje powodujące utratę informacji (zmiana typu zmiennoprzecinkowego do całkowitego) mogą powodować ostrzeżenie, ale nie są zabronione.
4. Niejawne przekształcenia arytmetyczne działają na ogół zgodnie z oczekiwaniami. Gdy nie korzystamy z typu „unsigned” to w wyrażeniach arytmetycznych:
   - jeżeli jeden z argumentów jest long double, pozostały jest zamieniany do long double,
   - w przeciwnym przypadku jeżeli jeden z argumentów jest double drugi zostanie przekształcony do tego typu,

Type casting II

Zasady

- w przeciwnym razie jeżeli jeden jest typu float drugi zostanie przekształcony do tego typu.
- w przeciwnym przypadku wszystkie argumenty char i short zostaną przekształcone do typu int
- następnie jeżeli którykolwiek z argumentów ma kwalifikator long drugi zostanie przekształcony do tego typu.

5. Zwracam uwagę, że typ float nie jest automatycznie zmieniany do double
6. W operacji podstawienia typ prawej strony jest przekształcany do typu lewej (co może powodować utratę informacji).
7. Dłuższe (bitowo) liczby przekształcane są do krótszych przez odrzucenie bardziej znaczących bitów.
8. W każdym wyrażeniu można wskazać sposób przekształcenia używając operatora cast (rzut): (nazwa-typu) wyrażenie.