

# Základy programování v C

Jan Faigl

Katedra počítačů  
Fakulta elektrotechnická  
České vysoké učení technické v Praze

Přednáška 02

## B0B36PRP – Procedurální programování

Jan Faigl, 2018 B0B36PRP – Přednáška 02: Základy programování v C 1 / 39

Program v C Proměnné a jejich hodnoty Základní číselné typy Výrazy a operátory Formátovaný vstup a výstup

## Jazyk C

- Nízko-úrovňový programovací jazyk
- Systémový programovací jazyk (operační systém)  
*Jazyk pro vestavné (embedded) systémy — MCU, křížová (cross) kompilace*
- Téměř vše nechává na uživateli (programátorovi)  
*Inicializace proměnných, uvolňování dynamické paměti*
- Má blízko k využití hardwarových zdrojů výpočetního systému  
*Přímé volání služeb OS, přímý zápis do registrů a portů.*
- Klíčové pro správné fungování programu je zacházení s pamětí  
*Cílem kurzu PRP je naučit se základním principům, které lze následně generalizovat též pro jiné programovací jazyky. Pochopení těchto principů je klíčem k efektivnímu psaní efektivních programů.*

**Je výhodné mít překlad programu plně pod kontrolou.**

*Přestože to může z počátku vypadat složité, jsou základní principy relativně jednoduché. I proto je výhodné používat základní nástroje pro překlad programů a po jejich osvojení využít komplexnější vývojové prostředí.*

Jan Faigl, 2018 B0B36PRP – Přednáška 02: Základy programování v C 5 / 39

Program v C Proměnné a jejich hodnoty Základní číselné typy Výrazy a operátory Formátovaný vstup a výstup

## Struktura zdrojového souboru

- Komentovaný zdrojový soubor program.c
- ```
1 /* komentar zapisujeme do dvojice vyhrazenych znaku */
2 // Nebo v C99 jako jednoradkovy
3 #include <stdio.h> /* vloženi hlavičkoveho souboru
   standardní knihovny stdio.h */
4
5 int main(void) // zjednodusena hlavička funkce
6 { // hlavní funkce program main()
7   printf("I like B0B36PRP!\n"); /* volani funkce
   printf() z knihovny stdio.h pro tisk textoveho
   retezce na standardni vystup. Znak \n definuje novy
   radek (odradkovani). */
8
9   return 0; /* ukončení funkce a předání navratove
   hodnoty 0 operacnimu systemu */
10 }
```

Jan Faigl, 2018 B0B36PRP – Přednáška 02: Základy programování v C 8 / 39

## Přehled témat

- Část 1 – Základy programování v C
  - Program v C
  - Proměnné a jejich hodnoty
  - Základní číselné typy
  - Výrazy a operátory
  - Formátovaný vstup a výstup
- Část 2 – Zadání 1. domácího úkolu (HW01)

S. G. Kochan: kapitoly 2, 3

Jan Faigl, 2018 B0B36PRP – Přednáška 02: Základy programování v C 2 / 39

Program v C Proměnné a jejich hodnoty Základní číselné typy Výrazy a operátory Formátovaný vstup a výstup

## Zápis programu

- Zdrojový kód programu v jazyce C se zapisuje do textových souborů
  - Zdrojové soubory zpravidla pojmenované s koncovkou `.c`
  - Hlavičkové soubory s koncovkou `.h`
- Kompilací zdrojových souborů překladačem do binární podoby vznikají objektové soubory `.o`
- Z objektových souborů se sestavuje výsledný program
- Příklad zápisu jednoduchého programu:

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5   printf("I like B0B36PRP!\n");
6
7   return 0;
8 }
```

`lec02/program.c`

Jan Faigl, 2018 B0B36PRP – Přednáška 02: Základy programování v C 6 / 39

Program v C Proměnné a jejich hodnoty Základní číselné typy Výrazy a operátory Formátovaný vstup a výstup

## Zdrojové soubory

*Proč psát do dvou nebo více souborů?*

- Rozdělení na zdrojové a hlavičkové soubory umožňuje rozlišit **deklaraci** a **definici**, především však podporuje
  - Organizaci** zdrojových kódů v adresářové struktuře souborů
  - Modularitu**
    - Hlavičkový soubor obsahuje popis co modul nabízí, tj. popis (seznam) funkcí a jejich parametrů bez konkrétní implementace (deklarace funkcí)
  - Znovupoužitelnost**
    - Pro využití binární knihovny potřebuje znát její „rozhraní“, které je deklarované v hlavičkovém souboru

*Zatím nemusí být výhody zřejmé, ale budou. V úloze HW 10!*

Jan Faigl, 2018 B0B36PRP – Přednáška 02: Základy programování v C 9 / 39

Program v C Proměnné a jejich hodnoty Základní číselné typy Výrazy a operátory Formátovaný vstup a výstup

## Část I

### Část 1 – Základy programování v C

Jan Faigl, 2018 B0B36PRP – Přednáška 02: Základy programování v C 3 / 39

Program v C Proměnné a jejich hodnoty Základní číselné typy Výrazy a operátory Formátovaný vstup a výstup

## Překlad (kompilace) a spuštění programu

- Zdrojový soubor `program.c` přeložíme do spustitelné podoby kompilátorem např. `clang` nebo `gcc`  
`clang program.c`
- Vznikne soubor `a.out`, který můžeme spustit např.  
`./a.out`  
*Alternativně pouze jako `a.out` pokud je aktuální pracovní adresář nastaven v prohledávané cestě spustitelných souborů*
- Program po spuštění vypíše text uvedený jako argument `printf()`  
`./a.out`  
I like B0B36PRP!
- Pokud nechce psát `./a.out` ale raději jen `a.out` lze přidat aktuální pracovní adresář do cest(y) definované proměnnou prostředí `PATH`  
`export PATH="$PATH:pwd"`  
*Pracovních adresářů můžete mít více—používejte obezřetně.*
- Příkaz `pwd` vytiskne aktuální pracovní adresář, více viz `man pwd`

**Ano jde to, ale není dobrý nápad to používat!**

Jan Faigl, 2018 B0B36PRP – Přednáška 02: Základy programování v C 7 / 39

Program v C Proměnné a jejich hodnoty Základní číselné typy Výrazy a operátory Formátovaný vstup a výstup

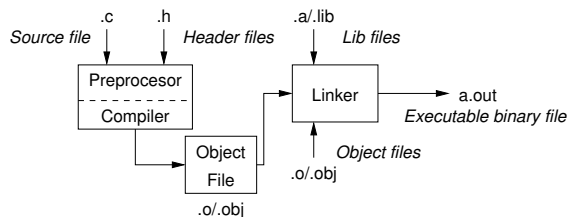
## Překlad a sestavení programu

- Uvedený příklad slučuje jednotlivé kroky překladu a sestavení programu do volání jediného příkazu (`clang` nebo `gcc`). Překlad se však skládá ze tří částí, které lze provést individuálně
  - Textové předzpracování **preprocesorem**, který má vlastní makro jazyk (příkazy uvozeny znakem `#`)  
*Všechny odkazované hlavičkové soubory se vloží do jediného zdrojového souboru*
  - Vlastní překlad zdrojového souboru do objektového souboru  
*Zpravidla jsou jména souborů zakončena příponou `.o`*  
`clang -c program.c -o program.o`  
*Příkaz kombinuje volání preprocesoru a kompilátoru.*
  - Spustitelný soubor se sestaví z příslušných dílčích objektových souborů a odkazovaných knihoven, tzv. „linkováním“ (**linker**), např.  
`clang program.o -o program`

Jan Faigl, 2018 B0B36PRP – Přednáška 02: Základy programování v C 10 / 39

## Schéma překladače a sestavení programu

- Vývoj programu se skládá z editace zdrojových souborů (.c a .h);  
*Lidsky čitelných*
- kompilace dílčích zdrojových souborů (.c) do objektových souborů (.o nebo .obj);  
*Strojově čitelných*
- linkování přeložených souborů do spustitelného programu;
- spuštění a ladění aplikace a opětovné editace zdrojových souborů.



## Části překladače a sestavení programu

- **preprocesor** – umožňuje definovat makra a tím přizpůsobit překlad aplikace kompilačnímu prostředí  
*Výstupem je textový („zdrojový“) soubor.*
- **compiler** – Překládá zdrojový (textový) soubor do strojově čitelné (a spustitelné) podoby  
*Nativní (strojový) kód platformy, bytecode, případně assembler*
- **linker** – sestavuje program z objektových souborů do podoby výsledné aplikace  
*Stále může odkazovat na knihovní funkce (dynamické knihovny linkované při spuštění programu), může též obsahovat volání OS (knihovny).*
- Dílčí části **preprocesor**, **compiler**, **linker** jsou zpravidla „jediný“ program, který se volá s příslušnými parametry

## Překladače jazyka C

- V rámci předmětu PRP budeme používat především překladače z rodin:
  - **gcc** – GNU Compiler Collection  
<https://gcc.gnu.org>
  - **clang** – C language family frontend for LLVM  
<http://clang.llvm.org>
- Pro win\* platformy pak odvozená prostředí **cygwin** <https://www.cygwin.com/> nebo **MinGW** <http://www.mingw.org/>
- Základní použití (přepínače a argumenty) je u obou překladačů stejné  
*clang je kompatibilní s gcc*
- Příklad použití
  - **compile:** `gcc -c program.c -o program.o`
  - **link:** `gcc program.o -o program`

## Příklad součtu dvou hodnot

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int sum; /* definice lokální proměnné typu int */
6
7     sum = 100 + 43; /* hodnota výrazu se uloží do sum */
8     printf("The sum of 100 and 43 is %i\n", sum);
9     /* %i formátovací příkaz pro tisk celého čísla */
10    return 0;
11 }
    
```

- Proměnná `sum` typu `int` reprezentuje celé číslo, jehož hodnota je uložena v paměti
- `sum` je námi zvolené symbolické jméno místa v paměti, kde je uložena celočíselná hodnota (typu `int`)

## Příklad součtu hodnot dvou proměnných

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int var1;
6     int var2 = 10; /* inicializace hodnoty proměnné */
7     int sum;
8
9     var1 = 13;
10    sum = var1 + var2;
11
12    printf("The sum of %i and %i is %i\n", var1, var2, sum);
13
14    return 0;
15 }
16
    
```

- Proměnné `var1`, `var2` a `sum` reprezentují tři různá místa v paměti (automaticky přidělené), ve kterých jsou uloženy tři celočíselné hodnoty

## Základní číselné typy

- Celočíselné typy – `int`, `long`, `short`, `char`  
*char – celé číslo v rozsahu jednoho bajtu nebo také znak*
  - Velikost paměti alokované příslušnou (celo)číselnou proměnnou se může lišit dle architektury počítače nebo překladače  
*Typ int má zpravidla velikost 4 bajty a to i na 64-bitových systémech*
  - Aktuální velikost paměťové reprezentace lze zjistit operátorem `sizeof()`, kde argumentem je jméno typu nebo proměnné.  
`int i;`  
`printf("%lu\n", sizeof(int));`  
`printf("ui size: %lu\n", sizeof(i));`  
*lec02/types.c*
- Neceločíselné typy – `float`, `double`  
*Jsou dané implementací, většinou dle standardu IEEE-754-1985*
  - `float` – 32-bit IEEE 754
  - `double` – 64-bit IEEE 754  
[http://www.tutorialspoint.com/cprogramming/c\\_data\\_types.htm](http://www.tutorialspoint.com/cprogramming/c_data_types.htm)

## Znaménkové a neznaménkové celočíselné typy

- Celočíselné typy kromě počtu bajtů rozlišujeme na
    - **signed** – **znaménkový** (základní)
    - **unsigned** – **neznaménkový**  
*Proměnná neznaménkového typu nemůže zobrazit záporné číslo*
  - Příklad (1 byte):  
`unsigned char: 0 až 255`  
`signed char: -128 až 127`
- ```

1 unsigned char uc = 127;
2 char su = 127;
3
4 printf("The value of uc=%i and su=%i\n", uc, su);
5 uc = uc + 2;
6 su = su + 2;
7 printf("The value of uc=%i and su=%i\n", uc, su);
    
```
- lec02/signed\_unsigned\_char.c*

## Znak – char

- Znak je typ `char`
- Znak reprezentuje celé číslo (byte)  
*Kódování znaků (grafických symbolů), např. ASCII – American Standard Code for Information Interchange.*
- Hodnotu znaku lze zapsat jako tzv. znakovou konstantu, např. `'a'`.  

```

1 char c = 'a';
2
3 printf("The value is %i or as char '%c'\n", c, c);
    
```

  
*lec02/char.c*  
`clang char.c && ./a.out`  
The value is 97 or as char 'a'
- Pro řízení výstupních zařízení jsou definovány řídicí znaky  
*Tzv. escape sequences*
  - `\t` – tabulátor (tabular), `\n` – nový řádek (newline),
  - `\a` – pípnutí (beep), `\b` – backspace, `\r` – carriage return,
  - `\f` – form feed, `\v` – vertical space

## Logický datový typ (Boolean) – `_Bool`

- Ve verzi **C99** je zaveden logický datový typ `_Bool`  
`_Bool logic_variable;`
- Jako hodnota `true` je libovolná hodnota typu `int` různá od 0
- Dále můžeme využít hlavičkového souboru `<stdbool.h>`, kde je definován typ `bool` a hodnoty `true` a `false`  
`#define false 0`  
`#define true 1`  
`#define bool _Bool`
- V původním (ANSI) C explicitní datový typ pro logickou hodnotu není definován.
  - Můžeme však použít podobnou definici jako v `<stdbool.h>`  
`#define FALSE 0`  
`#define TRUE 1`

## Rozsahy celočíselných typů

- Rozsahy celočíselných typů v C nejsou dány normou, ale implementací *Mohou se lišit implementací a prostředím 16 bitů vs 64 bitů*
- Norma garantuje, že pro rozsahy typů platí
  - $short \leq int \leq long$
  - $unsigned\ short \leq unsigned \leq unsigned\ long$
- Pokud chceme zajistit definovanou velikost můžeme použít definované typy například v hlavičkovém souboru `<stdint.h>`

```
int8_t      uint8_t
int16_t     uint16_t
int32_t     uint32_t

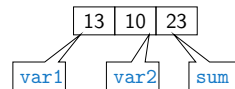
                lec02/inttypes.c
http://pubs.opengroup.org/onlinepubs/009695399/basedefs/stdint.h.html
```

IEEE Std 1003.1-2001

## Přirazení, proměnné a paměť – Vizualizace unsigned char

```
1 unsigned char var1;
2 unsigned char var2;
3 unsigned char sum;
4
5 var1 = 13;
6 var2 = 10;
7
8 sum = var1 + var2;
```

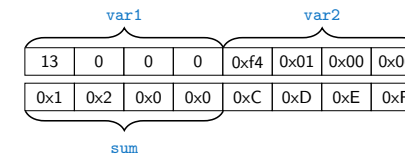
- Každá z proměnných alokuje právě 1 byte
- Obsah paměti není po alokaci definován *Undefined behavior*
- Jméno proměnné „odkazuje“ na paměťové místo
- Hodnota proměnné je obsah paměťového místa



## Přirazení, proměnné a paměť – Vizualizace int

```
1 int var1;
2 int var2;
3 int sum;
4
5 // 00 00 00 13
6 var1 = 13;
7
8 // x00 x00 x01 xF4
9 var2 = 500;
10
11 sum = var1 + var2;
```

- Proměnné typu `int` alokují 4 bajty *Zjistit velikost můžeme operátorem sizeof(int)*
- Obsah paměti není po alokaci definován



500 (dec) je 0x01F4 (hex)  
513 (dec) je 0x0201 (hex)

*V případě architektury Intel x86 a x86-64 jsou hodnoty uloženy v pořadí little-endian*

## Výrazy

- Výraz** předepisuje výpočet hodnoty určitého vstupu
- Struktura výrazu obsahuje **operandy**, **operátory** a **závorky**
- Výraz může obsahovat
  - literály
  - unární a binární operátory
  - proměnné
  - volání funkcí
  - konstanty
  - závorky
- Pořadí operací předepsaných výrazem je dáno **prioritou** a **asociativitou** operátorů.

### Příklad

```
10 + x * y // pořadí vyhodnocení 10 + (x * y)
10 + x + y // pořadí vyhodnocení (10 + x) + y

* má vyšší prioritu než +
+ je asociativní zleva
```

## Základní rozdělení operátorů

- Operátory jsou vyhrazené znaky (nebo posloupnost znaků) pro zápis výrazů
- Můžeme rozlišit čtyři základní typy binárních operátorů
  - Aritmetické** operátory – sčítání, odčítání, násobení, dělení
  - Relační** operátory – porovnání hodnot (menší, větší, ...)
  - Logické** operátory – logický součet a součin
  - Operátor přiřazení** - na levé straně operátoru = je proměnná
- Unární operátory
  - indikující kladnou/zápornou hodnotu: `+` a `-` *operátor – modifikuje znaménko výrazu za ním*
  - modifikující proměnnou: `++` a `--`
  - logický operátor doplněk: `!`
  - operátor přetypování: (jméno typu)
- Ternární operátor – podmíněně přiřazení hodnoty

## Proměnné, operátor přiřazení a příkaz přiřazení

- Proměnné definujeme uvedením typu a jména proměnné
  - Jména proměnných volíme malá písmena
  - Víceslovná jména zapisujeme s podtržítkem `_` *Nebo volíme CamelCase*
- Proměnné definujeme na samostatném řádku
 

```
int n;
int number_of_items;
```
- Přiřazení je nastavení hodnoty proměnné, tj. uložení definované hodnoty na místo v paměti, kterou proměnná reprezentuje
- Tvar **přiřazovacího operátoru**

$$\langle \text{proměnná} \rangle = \langle \text{výraz} \rangle$$
*Výraz je literál, proměnná, volání funkce, ...*
- Příkaz přiřazení** se skládá z operátoru přiřazení `=`;
  - Levá strana přiřazení musí být **l-value** – **location-value**, **left-value** *Tj. musí reprezentovat paměťové místo pro uložení výsledku.*
- Přiřazení je výraz a můžeme jej použít všude, kde je dovolen výraz příslušného typu

## Základní aritmetické výrazy

- Pro operandy číselných typů `int` a `double` jsou definovány operátory *Ale také pro char, short, float*
  - unární operátor změna znaménka `-`
  - binární sčítání `+` a odčítání `-`
  - binární násobení `*` a dělení `/`
- Pro operandy celočíselných typů pak dále
  - binární zbytek po dělení `%`
- Pro oba operandy stejného typu je výsledek aritmetické operace stejného typu
- V případě kombinace typů `int` a `double`, se `int` převede na `double` a výsledek je hodnota typu `double`. *Implicitní typová konverze*
- Dělení operandů typu `int` je celá část podílu *Např. 7/3 je 2 a -7/3 je -2*
- Pro zbytek po dělení platí  $x \% y = x - (x/y) * y$  *Např. 7 % 3 je 1 -7 % 3 je -1 7 % -3 je 1 -7 % -3 je -1*
  - Pro záporné operandy je v C99 výsledek celočíselného dělení blíže 0, platí (a/b)\*b + a%b = a. Pro starší verze C závisí výsledek na překladáči.*
  - Další aritmetické operátory přístě.*

## Příklad – Aritmetické operátory 1/2

```
1 int a = 10;
2 int b = 3;
3 int c = 4;
4 int d = 5;
5 int result;
6
7 result = a - b; // rozdíl
8 printf("a - b = %i\n", result);
9
10 result = a * b; // násobení
11 printf("a * b = %i\n", result);
12
13 result = a / b; // celočíselné dělení
14 printf("a / b = %i\n", result);
15
16 result = a + b * c; // prioritá operátoru
17 printf("a + b * c = %i\n", result);
18
19 printf("a * b + c * d = %i\n", a * b + c * d); // -> 50
20 printf("(a * b) + (c * d) = %i\n", (a * b) + (c * d)); // -> 50
21 printf("a * (b + c) * d = %i\n", a * (b + c) * d); // -> 350
```

lec02/arithmetic\_operators.c

## Příklad – Aritmetické operátory 2/2

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int x1 = 1;
6     double y1 = 2.2357;
7     float x2 = 2.5343f;
8     double y2 = 2;
9
10    printf("P1 = (%i, %f)\n", x1, y1);
11    printf("P1 = (%i, %i)\n", x1, (int)y1);
12    printf("P1 = (%f, %f)\n", (double)x1, (double)y1); // operator
13    printf("P1 = (%.3f, %.3f)\n", (double)x1, (double)y1); // pretypování (double)
14
15    printf("P2 = (%f, %f)\n", x2, y2);
16
17    double dx = (x1 - x2); // implicitní konverze na float, resp.
18    double dy = (y1 - y2);
19
20    printf("P1 - P2)=(%.3f, %.3f)\n", dx, dy);
21    printf("|P1 - P2|^2=%.2f\n", dx * dx + dy * dy);
22    return 0;
23 }
```

lec02/points.c

## Standardní vstup a výstup

- Spuštěný program v prostředí operačního systému má přiřazený znakově orientovaný standardní vstup (`stdin`) a výstup (`stdout`)  
*Výjimkou jsou zpravidla programy pro MCU bez OS.*
- Program může prostřednictvím `stdout` a `stdin` komunikovat s uživatelem
- Základní funkce pro znakový výstup je `putchar()` a pro vstup `getchar()` definované ve standardní knihovně `<stdio.h>`.
- Pro načítání číselných hodnot lze využít funkci `scanf()`
- Formátovaný výstup je možné tisknout funkcí `printf()`, např. číselné hodnoty

*Jedná se o knihovní funkce, ze standardní knihovny. Jména funkcí nejsou klíčová slova jazyka C.*

## Formátovaný výstup – printf()

- Číselné hodnoty lze tisknout (vypsat) na standardní výstup prostřednictvím funkce `printf()`  
*man printf, resp. man 3 printf*
- Argumentem funkce je textový řídící řetězec formátování výstupu
- Řídící řetězec formátu je uvozen znakem `'%'`
- Znakové posloupnosti (nezačínající `%`) se vypíší tak jak jsou uvedeny
- Základní řídící řetězec pro výpis hodnot jednotlivých typů
 

<code>char</code>	<code>%c</code>
<code>_Bool</code>	<code>%i, %u</code>
<code>int</code>	<code>%i, %x, %o</code>
<code>float</code>	<code>%f, %e, %g, %a</code>
<code>double</code>	<code>%f, %e, %g, %a</code>
- Dále je možné specifikovat počet vypsaných míst, zarovnání vlevo (vpravo), atd.

*Více na cvičení a v domácích úkolech.*

## Formátovaný vstup – scanf()

- Číselné hodnoty ze standardního vstupu lze načíst funkcí `scanf()`  
*man scanf, resp. man 3 scanf*
- Argumentem je textový řídící řetězec  
*Syntax podobný příkazu printf()*
- Je nutné předat paměťové místo pro uložení hodnoty
- Příklad načtení hodnoty celého čísla a hodnoty typu `double`

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int i;
6     double d;
7
8     printf("Enter int value: ");
9     scanf("%i", &i); /* operator & vraci adresu
10                      promenne i */
11
12     printf("Enter a double value: ");
13     scanf("%lf", &d);
14     printf("You entered %02i and %0.1f\n", i, d);
15
16     return 0;
17 }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

## Část II

## Část 2 – Zadání 1. domácího úkolu (HW01)

## Zadání 1. domácího úkolu HW01

## Téma: Načítání vstupu, výpočet a výstup

Povinné zadání: **1b**; Volitelné zadání: **není**; Bonusové zadání: **není**

- **Motivace:** Získat představu o interakci uživatele s programem
- **Cíl:** Osvojit si načítání vstupu, formátovaného výstupu a základní posloupnosti příkazů
- **Zadání:** <https://cw.fel.cvut.cz/wiki/courses/b0b36prp/hw/hw01>
  - Načítání celých čísel ze standardního vstupu  
(čísla v rozsahu [-10 000; 10 000])
  - Výpis čísel v dekadické a šestnáctkové soustavě
  - Provedení základní aritmetických operací s načtenými čísly
  - Výpočet podílu a průměrné hodnoty čísel
  - Dodržení správného formátování výstupu  
Použijte **hex** zobrazení výstupu – `hexdump -C`
- **Termín odevzdání:** **20.10.2018, 23:59:59 PDT**

*PDT – Pacific Daylight Time*

Diskutovaná témata

## Shrnutí přednášky

## Diskutovaná témata

- Základy programování v C
  - Program, zdrojové soubory a kompilace programu
  - Struktura zdrojového souboru a zápis programu
  - Proměnné, základní číselné typy
  - Proměnné, přiřazení a paměť
  - Základní výrazy
  - Standardní vstup a výstup programu
  - Formátovaný vstup a výstup
- **Příště:** Zápis programu v C a základní řídicí struktury