

Programování (v C)

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 02

BAB36PRGA – Programování v C

Přehled témat

- Část 1 – Programování v C
 - Program v C
 - Funkce
 - Číselné typy
 - Literály
 - Výrazy a operátory
- Část 2 – Řídící struktury (úvod)
 - Řídící struktury
 - Složený příkaz
 - Větvení
 - Cykly
- Část 3 – Zadání 1. domácího úkolu (HW1)

S. G. Kochan: kapitoly 2, 3, 4

S. G. Kochan: kapitola 5 a část kapitoly 6

Část I

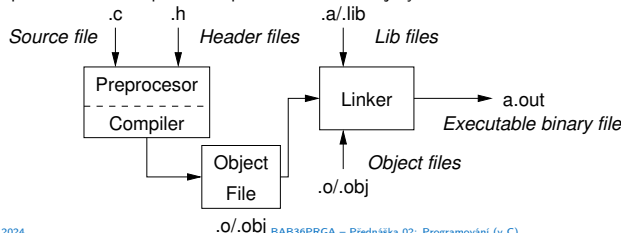
Část 1 – Programování v C

Zdrojové soubory programu v C

- Zdrojový soubor s koncovkou `.c`.
Zpravidla—základní rozlišení souborů, viz např. `.C`.
- Hlavičkový soubor s koncovkou `.h`.
Jména souborů volíme výstižně (krátké názvy) a zpravidla zapisujeme malými písmeny.
- Zdrojové soubory jsou překládány do binární podoby překladačem a vznikají objektové soubory (`.o`) nebo spustitelný program.
Objektový kód obsahuje relativní adresy proměnných a volání funkcí nebo pouze odkazy na jména funkcí, jejichž implementace ještě nemusejí být známy.
- Z objektových souborů (object files) se sestavuje výsledný program, ve kterém jsou již všechny funkce známy a relativní adresy se nahradí absolutními.
Program se zpravidla sestavuje z více objektových souborů umístěných například v knihovnách.

Schéma překladače a sestavení programu

- Vývoj programu se skládá z editace zdrojových souborů (`.c` a `.h`).
Lidsky čitelných zdrojově čitelných
 - Kompilace zdrojových souborů (`.c`) do objektových (`.o` nebo `.obj`).
 - Preprocesor – zpracování maker a přizpůsobení překladači kompilačnímu prostředí.
 - Linkování přeložených (objektových) souborů do spustitelného programu.
Také vytváření dynamicky linkovaných knihoven.
- Spouštění a ladění aplikace a opětovné editace zdrojových souborů.



Překladače jazyka C

- V PRGA používáme `gcc` a `clang` (C language family frontend for LLVM).
<https://gcc.gnu.org> <http://clang.llvm.org>
- Příklad použití
Základní použití (přepínače a argumenty) je u obou překladačů stejné.
 - compile: `gcc -c program.c -o program.o`
 - link: `gcc program.o -o program`
 - Sloučení překladače a sestavení v jediném příkazu `clang program.c -o program`
 - Linkování s vložením matematické knihovny `clang program.o -lm -o program`
Např. pokud použijeme funkci `sqrt` z knihovny `math.h`.
- Informace o souboru (file) a závislosti na dynamických knihovnách (ldd).

```
% clang var.c -o var % clang program.c -lm -o program
% file var % ldd program
var: ELF 64-bit LSB executable, x86-64, version 1 (
FreeBSD), dynamically linked, interpreter /
libexec/ld-elf.so.1, for FreeBSD 12.4
(1204500), FreeBSD-style, with debug_info, not
stripped % clang program.c -lm -static -o program
% ldd var % ldd program
ldd: program: not a dynamic ELF executable
var: libc.so.7 => /lib/libc.so.7 (0x2c41d000)
```

Zdrojové soubory

Proč psát program do více souborů?

- Zdrojové a hlavičkové soubory umožňují rozlišit deklaraci a definici podporující:
 - Znovupoužitelnost
 - K využití binární knihovny potřebuje znát „rozhraní“ funkcí (případně typů), které je deklarované v hlavičkovém souboru.
Např. funkce standardní knihovny `C`, `libc`.
 - Modularita
 - Hlavičkový soubor obsahuje popis co modul nabízí, tj. popis (seznam) funkcí a jejich parametrů (deklarace funkcí) bez konkrétní implementace.
Implementace funkce je definice funkce.
- Organizaci zdrojových kódů v adresářové struktuře souborů.
Deklarování, že funkce existují a jaké mají rozhraní (vstup a výstup) argumenty a návratový typ udávající velikost paměti pro předávaná data.

*Pro jednoduché programy a domácí úkoly nedává moc smysl.
Vyplní se především v HW7, HW8 a HW9, případně HW6 (Matice)!*

Příklad kompilace programu z více souborů

```
1 #ifndef __COMPUTE_H__
2 #define __COMPUTE_H__
3
4 // deklarace funkce (hlavička/prototyp)
5 int compute(int a);
6
7 #endif
8
9 #include "compute.h"
10
11 int compute(int a) // definice funkce
12 {
13     int b = 10 + a; // tělo funkce
14     return b; // návratová hodnota funkce
15 }
16
17 #include "compute.h" // vložení deklarace funkce
18 { // hlavní funkce
19     int v = 10; // definice proměnné
20     int r = compute(v); // volání funkce
21     return 0; // ukončení hlavní funkce
22 }
23
24 clang -c compute.c
25 clang -c main-compute.c
26 clang main-compute.o compute.o -o compute
27 ./compute
```

- Výsledný spustitelný soubor linkujeme s `main-compute.o` a `compute.o`, musí obsah právě jednu funkci `main()`.
- Linkování spustitelné aplikace pouze s `main-compute.o` skončí chybou.

```
% gcc main-compute.o -o compute
/usr/local/bin/ld: main-compute.o: in function 'main':
main-compute.c:(.text+0x21): undefined reference to 'compute'
collect2: error: ld returned 1 exit status
```

Spustitelný program – `main()`

- Spustitelný program musí obsahovat jedinou definici funkce `main()`, která má základní tvary předání argumentů programu.

```
int main(int argc, char *argv) int main(int argc, char **argv, char **envp)
{ } { ... }
{ ... }
```
- Při spuštění programu předává OS programu počet argumentů (`argc`) a argumenty (`argv`), jako pole textových řetězců.
První argument je jméno programu.
- Návratová hodnota je předána OS, kde je možné ji dále použít.
 - Návratová hodnota programu je v proměnné `$?` . `sh`, `bash`, `zsh`
 - Příklad spuštění s různým počtem argumentů.

```
1 int v;
2 {
3     int v;
4     v = 10;
5     v = v + 1;
6     return argc;
7 }
```


`1 ./var a; echo $?
2`

Program v C Funkce Číselné typy Literály Výrazy a operátory

Příklad kompilace a spuštění programu

- Sestavení programu `clang var.c` automaticky dojde ke kompilaci a linkování programu do spustitelného souboru `a.out`.
Výchozí jméno programu.
- Výstupní (output) soubor specifikuje `clang var.c -o var` a spustíme, např. `./var`.
- Kompilaci a spuštění můžeme spojit do dvojice příkazů `clang var.c -o var; ./var`.
- Spuštění můžeme podmínit úspěšnou kompilací programu `clang var.c -o var && ./var`.
Návratová hodnota programu — 0 (EXIT_SUCCESS) znamená OK, chyb může být více.
Logický operátor && závisí na příkazovém interpretu, např. sh, bash, zsh.
- Příznakem `-E` můžeme při „kompilaci“ vyvolat pouze preprocesor: `gcc -E var.c`.

```

1 #1 "var.c"
2 #1 "<built-in>"
3 #1 "<command-line>"
4 #1 "var.c"
5 int main(int argc, char **argv) {
6     int v;
7     v = 10;
8     v = v + 1;
9     return argc;
10 }

```

lec02/var.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 11 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Příkaz return

- Příkaz ukončení funkce `return vyraz;`;
- `return` lze použít pouze v těle funkce.
- `return` ukončí funkci, vrátí návratovou hodnotu funkce určenou hodnotou `vyraz` a předá řízení volající funkci.
- `return` lze použít v těle funkce vícekrát.
Kódovací konvence může doporučovat nejvýše jeden výskyt return ve funkci.
- U funkce s prázdným návratovým typem, např. `void fce()`, nahrazuje uzavírací závorka těla funkce příkaz `return;`.

```

void fce(int a)
{
    ...
}

```

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 15 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

min_max() – příklad volání

- Vytvoříme vstupní soubor s pěti náhodnými čísly od 1 do 99: `shuf -i 1-99 -n 5`.
- Standardní výstup programu `shuf` přeměrujeme do souboru `in.txt`.
- Standardní vstup našeho programu `minmax` přeměrujeme ze souboru `in.txt`.
- Vytiskneme návratovou hodnotu volání programu.

```

1 % clang min_max.c -o minmax
2 % shuf -i 1-99 -n 5 > in.txt
3 % ./minmax <in.txt
4 Read 5 numbers, min: 1, max: 9
5 % echo $?
6 0

```

lec02/min_max.c

- Vytvoříme alternativní (chybný) vstup, nebo zadáme ručně.

```

1 % echo "a" >in2.txt
2 % lec02 cat in.txt >>in2.txt
3 % ./minmax <in2.txt
4 ERROR: No input given!
5 % echo $?
6 1

```

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 18 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Funkce

- Funkce tvoří základní stavební blok **modulárního** jazyka C.
Modulární program je složen z více modulů/zdrojových souborů.
- Každý spustitelný program v C obsahuje *alespoň* jednu funkci a to funkci `main()`.
Běh programu začíná funkcí `main()`.
- Deklarace** se skládá z hlavičky funkce.

```

typ_návratové_hodnoty jméno_funkce(seznam parametrů);
C používá prototyp (hlavičku) funkce k deklaraci informací nutných pro překlád tak, aby
mohlo být přeloženo správné volání funkce i v případě, že definice je umístěna dále v kódu.

```

- Definice funkce obsahuje **hlavičku funkce a její tělo**, syntax:

```

typ_návratové_hodnoty jméno_funkce(seznam parametrů)
{
    //tělo funkce
}

```

Definice funkce bez předchozí deklarace je zároveň deklarací funkce.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 13 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Argumenty funkce

- Argumenty funkce se předávají hodnotou.

```

1 int main(void)
2 {
3     int v1 = 10;
4     int v2 = 20;
5
6     printf("v1: %i v2: %i\n", v1, v2);
7     swap0(v1, v2);
8     printf("v1: %i v2: %i\n", v1, v2);
9     swap(&v1, &v2); //předání pamětového místa
10    printf("v1: %i v2: %i\n", v1, v2);
11    return 0;
12 }

```

```

14 void swap0(int a, int b)
15 {
16     int t = a; // dočasná proměnná
17     a = b;
18     b = t;
19 }
21 void swap(int *a, int *b)
22 {
23     int t = *a; // dočasná proměnná
24     *a = *b;
25     *b = t;
26 }

```

lec02/swap.c

- Proto předáváme adresu pamětového místa (pointer/ukazatel) – `&v1` a `&v2`.

```

% clang swap.c -o swap && ./swap
v1: 10 v2: 20
v1: 10 v2: 20
v1: 20 v2: 10

```

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 16 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

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 pamětové 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));`
- Neceločíselné typy – `float, double`
Konkretní reprezentace je dána 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

lec02/types.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 20 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Vlastnosti funkcí

- C nepovoluje funkce vnořené do jiných funkcí.
- Jména funkcí se mohou exportovat do ostatních modulů.
Modul je samostatně překládaný soubor.
- Funkce jsou implicitně deklarovány jako **extern**, tj. viditelné.
- Specifikátorem `static` před jménem funkce omezíme viditelnost jména funkce pouze pro daný modul (tj. konkrétní jméno souboru `.c`).
Lokální funkce modulu.
- Formální parametry funkce jsou **lokální proměnné**, které jsou inicializovány skutečnými parametry při volání funkce.
Parametry se do funkce předávají hodnotou (Call by Value).
- C **dovoluje rekurzi** – lokální proměnné jsou pro každé jednotlivé volání zakládány znovu na zásobník.
Kód funkce v C je reentrantní ve smyslu volání funkce ze sebe sama.
- Funkce nemusí mít žádné vstupní parametry, zapisujeme klíčovým slovem `void`.
`fce(void)`
- Funkce nemusí vracet funkční hodnotu – návratový typ je `void`.
`void fce(void)`

lec02/function.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 14 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Argumenty funkce a návratová hodnota

- K „vrácení“ více hodnot, můžeme využít předání pamětových míst.
Podobně jako scanf().
- Příklad načtení celých čísel typu `int` a určení minimální a maximální hodnoty.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <limits.h>
4
5 void min_max(int v, int *min, int *
max);
6
7 int main(void)
8 {
9     int ret = EXIT_SUCCESS;
10    int min = INT_MAX; // limits
11    int max = INT_MIN; // limits
12    int c = 0;
13    int v;
14
15    while (scanf("%i", &v) == 1) {
16        min_max(v, &min, &max);
17        c = c + 1;
18    }
19
20    if (c > 0) {
21        printf("Read %d numbers, min: %
d, max: %d\n", c, min, max);
22    } else {
23        fprintf(stderr, "ERROR: No
input given!\n");
24        ret = EXIT_FAILURE;
25    }
26    return ret;
27 }
28 void min_max(int v, int *min, int *
max)
29 {
30    if (v < *min) *min = v;
31    if (v > *max) *max = v;
32 }

```

lec02/min_max.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 17 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

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

`$ clang signed_unsigned.c && ./a.out`
The value of uc=127 and su=127
The value of uc=129 and su=-127

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 21 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

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';
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.
 - `\t` – tabulátor (tabular), `\n` – nový řádek (newline),
 - `\a` – pípnutí (beep), `\b` – backspace, `\r` – carriage return,
 - `\f` – form feed, `\v` – vertical space

Tzv. *escape sequences*

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 22 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

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

#define FALSE 0
#define TRUE 1
```
- 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
```

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 23 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

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 ≤ int ≤ long`
 - `unsigned short ≤ unsigned ≤ unsigned long`
- Pokud chceme zajistit definovanou velikost můžeme použít definované typy například z hlavičkového souboru `<stdint.h>`.


```
int8_t      uint8_t
int16_t     uint16_t
int32_t     uint32_t
```

`IEEE Std 1003.1-2001`
`lec02/inttypes.c`
<http://pubs.opengroup.org/onlinepubs/009695399/basedefs/stdint.h.html>

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 24 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Přirazení, proměnné a paměť – Vizualizace `int`

```
1 int var1;
2 int var2;
3 int sum;

5 // 00 00 00 13
6 var1 = 13;

8 // x00 x00 x01 xF4
9 var2 = 500;

11 sum = var1 + var2;
```

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

var1				var2			
13	0	0	0	0xf4	0x01	0x00	0x00
0x1	0x2	0x0	0x0	0xc	0xd	0xe	0xf
sum							

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.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 25 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Literály

- Jazyk C má 6 typů literálů (konstantních hodnot).
 - Celočíselné;
 - Racionální;
 - Znakové;
 - Řetězcové;
 - Výčtové – pojmenovaná celá čísla typu `int`;
- Symbolické – `#define NUMBER 10`.

Enum
Preprocesor

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 27 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Literály racionálních čísel

- Formát zápisu racionálních literálů:
 - S řádovou tečkou – `13.1`;
 - Mantisa a exponent – `31.4e-3` nebo `31.4E-3`.
- Typ racionálního literálu:
 - `double` – pokud není explicitně určen;
 - `float` – přípona `F` nebo `f`;
 - `long double` – přípona `L` nebo `l`.

`float f = 10f;`
`long double ld = 10l;`

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 28 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Znakové literály

- Formát – jeden (případně více) znaků v jednoduchých apostrofech `'A'`, `'B'` nebo `'\n'`.
- Hodnota – jednoznakový literál má hodnotu odpovídající kódu znaku `'0' ~ 48`, `'A' ~ 65`.
 - Hodnota znaků mimo ASCII (větší než 127) závisí na překladači.
- Typ znakové konstanty.
 - Znaková konstanta je typu `int`. Automatická konverze kódu ASCII znaku na typ `char`.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 29 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Řetězcové literály

- Formát – posloupnost znaků a řídicích znaků (escape sequences) uzavřená v uvozovkách.
 - Řetězcové konstanty oddělené oddělovači (white spaces) se sloučí do jediné, např. `"Řetězcová konstanta" " s koncem řádku\n"` se sloučí do `"Řetězcová konstanta s koncem řádku\n"`.
- Typ
 - Řetězcová konstanta je uložena v poli typu `char` a zakončená znakem `'\0'`. Např. řetězcová konstanta `"word"` je uložena jako posloupnost znaků/bajtů (pole).


```
'w' 'o' 'r' 'd' '\0'
```

Pole tak musí být vždy o 1 položku delší!
Více o textových řetězcích na 4. přednášce a cvičení.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 30 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Konstanty výčtového typu

- Formát
 - Implicitní hodnoty konstanty výčtového typu začínají od 0 a každý další prvek má hodnotu o jedničku vyšší.
 - Hodnoty můžeme explicitně předepsat.


```
enum {
    SPADES,
    CLUBS,
    HEARTS,
    DIAMONDS
};

enum {
    SPADES = 10,
    CLUBS, /* the value is 11 */
    HEARTS = 15,
    DIAMONDS = 13
};
```

Hodnoty výčtu zpravidla píšeme velkými písmeny.
- Typ – výčtová konstanta je typu `int`.
 - Hodnotu konstanty můžeme použít pro iteraci v cyklu.


```
enum { SPADES = 0, CLUBS, HEARTS, DIAMONDS, NUM_COLORS };
for (int i = SPADES; i < NUM_COLORS; ++i) {
    ...
}
```

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 31 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Symbolické konstanty – #define

- Formát – konstanta je založena příkazem preprocesoru **#define**.
 - Je to makro příkaz bez parametru.
 - Každý **#define** musí být na samostatném řádku.


```
#define SCORE 1
```

Zpravidla píšeme velkými písmeny.

- Symbolické konstanty mohou vyjadřovat konstantní výraz.


```
#define MAX_1 ((10*6) - 3)
```
- Symbolické konstanty mohou být vnořené.


```
#define MAX_2 (MAX_1 + 1)
```
- Preprocesor provede textovou náhradu definované konstanty za její hodnotu.**

```
#define MAX_2 (MAX_1 + 1)
```

*Je-li hodnota výraz, jsou kulaté závorky nutné pro správné vyhodnocení výrazu, např. pro 5*MAX_1 s vnějšími závorkami je 5*(10*6) - 3=285 vs 5*(10*6) - 3=297.*

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 32 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Proměnné s konstantní hodnotou – modifikátor (const)

- Uvedením klíčového slova **const** můžeme označit proměnnou jako konstantní.

Překladač kontroluje přiřazení a nedovolí hodnotu proměnné nastavit znovu.
- Pro definici konstant můžeme použít konstantní proměnné, symbolické konstanty (preprocesor) a v případě celočíselných hodnot (**int**) také **enum**.
- Proměnné s konstantní hodnotou mají typ a paměť


```
const float pi = 3.14159265;
```
- na rozdíl od symbolické konstanty


```
#define PI 3.14159265
```
- reprezentující literál.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 33 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Výrazy

- Výraz** předepisuje výpočet hodnoty určitého vstupu.
- Struktura výrazu obsahuje **operandy**, **operátory** a **závorky**.
- Výraz může obsahovat
 - literály,
 - proměnné,
 - konstanty,
 - unární a binární operátory,
 - volání funkcí,
 - 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*

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 35 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

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 -;
 - modifikující proměnou ++ a --;
 - logický operátor doplněk !;
 - operátor přetypování (jméno typu).

Unární operátor minus – modifikuje znaménko výrazu za ním.
- Ternární operátor** – podmíněný výsledek výrazu ze dvou výrazů.


```
výraz ? hodnota1 : hodnota2
```

Hodnota výrazu ternárního operátoru je buď první nebo třetí operand v závislosti na logické hodnotě prvního operandu.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 36 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Proměnné, operátor přiřazení a příkaz přiřazení

- Proměnné definujeme uvedením typu a jména proměnné.
 - Jména proměnných volíme malá písmena.
 - Víceřádková jména zapisujeme s podtržítkem _.
 - Proměnné definujeme na samostatném řádku.


```
int n;
int number_of_items;
```
 - Proměnné reprezentují data, proto volíme podstatná jména.**
- Přiřazení je nastavení hodnoty proměnné, tj. uložení definované hodnoty na místo v paměti, kterou proměnná reprezentuje.
- Tvar **přiřazovacího operátoru**.


```
<proměnná> = <výraz>
```

Výraz je literál, proměnná, volání funkce, ...
- Příkaz přiřazení** se skládá z operátoru přiřazení = a ;.
 - 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.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 37 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Základní aritmetické výrazy

- Pro operandy (ne)celočíslných typů **int**, **char**, **short** a **double** a **float** jsou definovány operátory:
 - 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 blíž 0, platí (a/b)*b + a%b = a.
Pro starší verze C závisí výsledek na překladači.*

Další operátory přístě.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 38 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Příklad – Aritmetické operátory 1/2

```
1 int a = 10;
2 int b = 3;
3 int c = 4;
4 int d = 5;
5 int result;

7 result = a - b; // rozdíl
8 printf("a - b = %i\n", result);

10 result = a * b; // nasobeni
11 printf("a * b = %i\n", result);      lec02/arithmetic_operators.c

13 result = a / b; // celociselné deleni
14 printf("a / b = %i\n", result);

16 result = a + b * c; // priorita operatoru
17 printf("a + b * c = %i\n", result);

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
```

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 39 / 60

Program v C Funkce Číselné typy Literály Výrazy a operátory

Příklad – Aritmetické operátory 2/2

```
1 #include <stdio.h>

3 int main(void)
4 {
5     int x1 = 1;
6     double y1 = 2.2357;
7     float x2 = 2.5343f;
8     double y2 = 2;

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     // přetypování (double)      lec02/points.c
14     printf("P1 = (%.3f, %.3f)\n", (double)x1, (double)y1);

15     printf("P2 = (%f, %f)\n", x2, y2);

17     double dx = (x1 - x2); // implicitní konverze na float, resp. double
18     double dy = (y1 - y2);

20     printf("(P1 - P2)=(%.3f, %.3f)\n", dx, dy);
21     printf("|P1 - P2|^2=%.2f\n", dx * dx + dy * dy);
22     return 0;
23 }
```

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 40 / 60

Řídicí struktury Složený příkaz Větvení Cykly

Část II

Část 2 – Řídicí struktury

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 41 / 60

Řídicí struktury

Řídicí struktura je programová konstrukce, která se skládá z dílčích příkazů a předepisuje pro ně způsob provedení.

Trí základní druhy řídicích struktur:

- Posloupnost** – předepisuje **postupné provedení** dílčích příkazů;
- Větvení** – předepisuje provedení dílčích příkazů v závislosti na **splnění určité podmínky**;
- Cyklus** – předepisuje **opakované provedení** dílčích příkazů v závislosti na splnění určité podmínky.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 43 / 60

Typy řídicích struktur

- Sekvence**
- Podmínka If**
- Podmínka If**
- Větvení switch**
- Cyklus for a while**
- Cyklus do**

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 44 / 60

Složený příkaz a blok

- Řídicí struktury mají obvykle formu strukturovaných příkazů.
 - Složený příkaz** – posloupnost příkazů.
 - Blok** – posloupnost definic proměnných a příkazů.

```

1 {
2     //blok je vymezen složenými závorkami
3     int steps = 10;
4
5     printf("No. of steps %i\n", steps);
6 }
7
8 steps += 1; //nelze - mimo rozsah platnosti bloku
9         Definice - alokace paměti podle konkrétního typu proměnné. Rozsah platnosti proměnné je lokální v rámci bloku.

```

- Budeme používat složené příkazy:
 - složený příkaz nebo blok pro posloupnost;
 - příkaz **if** nebo **switch** pro větvení;
 - příkaz **while**, **do** nebo **for** pro cyklus.

Podmíněné opakování bloku nebo složeného příkazu.

- Funkce je pojmenovaný blok příkazů**, který můžeme **znovupoužít**.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 46 / 60

Větvení if

- Příkaz **if** umožňuje větvení programu na základě podmínky.
- Má dva základní tvary.
 - if (podmínka) příkaz₁**
 - if (podmínka) příkaz₁ else příkaz₂**
- podmínka** je logický výraz, jehož hodnota je logického (celočíslného) typu.

Tj. *false (hodnota 0) nebo true (hodnota různá od 0).*
- příkaz** je příkaz, složený příkaz nebo blok.

Příkaz je zakončen středníkem ;

- Ukázka zápisu zjištění menší hodnoty z x a y .

Varianta zápisu 1	Varianta zápisu 2	Varianta zápisu 3
<pre>int min = y; if (x < y) min = x;</pre>	<pre>int min = y; if (x < y) min = x;</pre>	<pre>int min = y; if (x < y) { min = x; }</pre>

Která varianta splňuje kódovací konvenci a proč?

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 48 / 60

Příklad větvení if

Příklad: Jestliže $x < y$ vyměňte hodnoty těchto proměnných. Nechť proměnné x a y jsou definovány a jsou typu `int`.

Varianta 1	Varianta 2	Varianta 3	Varianta 4
<pre>if (x < y) tmp = x; x = y; y = tmp;</pre>	<pre>if (x < y) int tmp = x; x = y; y = tmp;</pre>	<pre>int tmp; if (x < y) tmp = x; x = y; y = tmp;</pre>	<pre>if (x < y) { int tmp = x; x = y; y = tmp; }</pre>

- Která varianta je správně a proč?

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 49 / 60

Příklad větvení if-then-else

Příklad: Do proměnné `min` uložte menší z čísel x a y a do `max` uložte větší z čísel. Nechť proměnné x , y , `min` a `max` jsou definovány a jsou typu `int`.

Varianta 1	Varianta 2
<pre>if (x < y) min = x; max = y; else min = y; max = x;</pre>	<pre>if (x < y) { min = x; max = y; } else { min = y; max = x; }</pre>

- Která varianta odpovídá našemu zadání?

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 50 / 60

Cyklus while ()

- Příkaz **while** má tvar `while (vyraz) prikaz;`
- Příkaz cyklu **while** probíhá:
 - Vyhodnotí se výraz `vyraz`;
 - Pokud `vyraz != 0`, provede se příkaz `prikaz`, jinak cyklus končí;
 - Opakování vyhodnocení výrazu `vyraz`.
- Řídicí cyklus se vyhodnocuje na začátku cyklu, cyklus se nemusí provést ani jednou.
- Řídicí výraz `vyraz` se musí aktualizovat v těle cyklu, jinak je cyklus nekonečný.

Příklad zápisu

```
int i = 0;
while (i < 5) {
    ...
    i += 1;
}
```

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 52 / 60

Příklad cyklu while

- Základní příkaz cyklu **while** má tvar `while (podmínka) prikaz.`

Příklad

```
int x = 10;
int y = 3;
int q = x;

while (q >= y) {
    q = q - y;
}
```

- Jaká je hodnota proměnné q po skončení cyklu?

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 53 / 60

Cyklus do...while ()

- Příkaz **do...while** () má tvar `do prikaz while (vyraz);`
- Příkaz cyklu **do...while** () probíhá:
 - Provede se příkaz `prikaz`;
 - Vyhodnotí se výraz `vyraz`;
 - Pokud `vyraz != 0`, cyklus se opakuje provedením příkazu `prikaz`, jinak cyklus končí.
- Řídicí cyklus se vyhodnocuje na konci cyklu, tělo cyklu se vždy provede nejméně jednou.
- Řídicí výraz `vyraz` se musí aktualizovat v těle cyklu, jinak je cyklus nekonečný.

Příklad zápisu

```
int i = -1;
do {
    ...
    i += 1;
} while (i < 5);
```

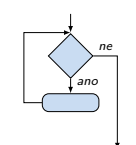
Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 54 / 60

Rídící struktury Složený příkaz Větvení Cyklus

Cyklus for

- Základní příkaz cyklu **for** má tvar **for** (*inicializace; podmínka; změna*) příkaz.
- Odpovídá cyklu while v následujícím tvaru.


```
inicializace;
while (podmínka) {
    příkaz;
    změna;
}
```



Příklad

```
for (int i = 0; i < 10; ++i) {
    printf("i: %i\n", i);
}
```

- Změnu řídicí proměnné lze zkráceně zapsat operátorem inkrementace **++** nebo dekrementace **--**.
- Alternativně lze též použít zkrácený zápis přiřazení, např. **+=**.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 55 / 60

Rídící struktury Složený příkaz Větvení Cyklus

Cyklus for – příklady

- Jak se změní výstup když použijeme místo prefixového zápisu **++i** postfixový zápis **i++**.


```
for (int i = 0; i < 10; i++) {
    printf("i: %i\n", i);
}
```
- V cyklu můžeme také řídicí proměnou dekrementovat.


```
for (int i = 10; i >= 0; --i) {
    printf("i: %i\n", i);
}
```
- Kolik řádků vypíše program?


```
for (int i = 10; i > 0; --i) {
    printf("i: %i\n", i);
}
```
- Řídicí proměnná může být také neceločíslného typu, např. **double**.


```
#include <math.h>

for (double d = 0.5; d < M_PI; d += 0.1) {
    printf("d: %f\n", d);
}
```

M_PI je symbolická konstanta definovaná v math.h.

Kolik program vypíše řádků?

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 56 / 60

Část III

Část 3 – Zadání 1. domácího úkolu (HW1)

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 57 / 60

Zadání 1. domácího úkolu HW1

Téma: Načítání vstupu

Povinné zadání: **3b**; Volitelné zadání: **není**; Bonusové zadání: **není**

- Motivace:** „Automatizovat“ a zobecnit výpočet pro „libovolně“ dlouhý vstup.
- Cíl:** Osvojit si využití cyklů jako základní programové konstrukce pro hromadné zpracování dat.
- Zadání:** <https://cw.fel.cvut.cz/wiki/courses/bab36prga/hw/hw1>
 - Zpracování **libovolně dlouhých** posloupností celých čísel.
 - Výpis načtených čísel.
 - Výpis statistiky vstupních čísel.
 - Počet načtených čísel; Počet kladných a záporných čísel a jejich procentuální zastoupení na vstupu.
 - Četnost výskytu sudých a lichých čísel a jejich procentuální zastoupení na vstupu.
 - Průměrná, maximální a minimální hodnota načtených čísel.
- Termín odevzdání:** **16.03.2024, 23:59:59 PDT.**

PDT – Pacific Daylight Time

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 58 / 60

Diskutovaná témata

Shrnutí přednášky

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 59 / 60

Diskutovaná témata

- Programování v C
 - Zápis programu v C
 - Program, zdrojové soubory a kompilace programu
 - Literály a konstantní hodnoty
 - Proměnné, základní číselné typy
 - Proměnné, přiřazení a paměť
 - Základní výrazy
 - Řídicí struktury
- Příště: Dokončení řídicích struktur, výrazy.**

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 60 / 60

Část V

Appendix

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 61 / 60

Kódovací příklad – Zadání

- Implementujte program, který vytiskne vzor o sedmi řádcích.


```
*****
** ** ** **
*** ** **
*****
** ** **
** ** **
*****
```
- Výchozí šířka *n* je 27 znaků nebo je načtena jako první argument programu (je-li zadán).
- Šířka *n* musí být liché číslo, jinak program vrátí **100**.
- Platí $11 \leq n \leq 67$, jinak program vrátí **101**.
- Při úspěchu program vytiskne sedm řádků a vrátí **0 (EXIT_SUCCESS)**.
- Snažte se maximálně vyhnout použití "magic numbers" v programu.**

- Argument programu `argv[1]` převed' te na číslo `atoi()`, jeli zadán.
- Dekomponujte program jako tisk 7× řádků.
- Implementujte „infrastrukturu“ programu.
- Následně řešte logiku jednotlivých řádků řízených **vhodně navrženým výrazem**.

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 62 / 60

Příklad kódování – Strategie implementace 1/4

- Definujeme návratové (chybové) hodnoty (0, 100, 101) využitím `enum`, aby byl „kód čistý“.
- Definujeme platný rozsah (11, 67), (`#define`).
- Zajistíme přístup k argumentům programu pouze tehdy, pokud jsou zadány.
- Kontrolujeme, že počet řádků *n* je platná hodnota, jinak program vrací chybu.
- Provádíme libovolnou operaci pouze v případě, že argumenty (hodnoty) jsou platné.
- Tisk 7-mi řádků rozdělíme do dvou `for` smyček, mezi smyčkami bude tisk plného * řádku.
- Implementujeme samostatnou funkci tisk vřozu řádku.

```
#include <stdio.h> //for putchar()
#include <stdlib.h> //for atoi()

enum {
    ERROR_OK = 0,
    ERROR_INPUT = 100,
    ERROR_RANGE = 101
};

#define MIN_VALUE 11
#define MAX_VALUE 67

#define LINES 3

// Print line of the with n using character
in c and space; with k continuous
characters c followed by space.
void print(char c, int n, int k);
```

Jan Faigl, 2024 BAB36PRGA – Přednáška 02: Programování (v C) 63 / 60

Příklad kódování – Strategie implementace 2/4

- Definujeme návratové (chybové) hodnoty (0, 100, 101) využitím `enum`, aby byl „kód čistý“.
- Definujeme platný rozsah (11, 67), (`#define`).
- Zajistíme přístup k argumentům programu pouze tehdy, pokud jsou zadány.
- Kontrolujeme, že počet řádků n je platná hodnota, jinak program vrací chybu.
- Provádíme libovolnou operaci pouze v případě, že argumenty (hodnoty) jsou platné.
- Tisk 7-mi řádků rozdělíme do dvou `for` smyček, mezi smyčkami bude tisk plného `*` řádku.
- Implementujeme samostatnou funkci tisk vzoru řádku.

```
...
int main(int argc, char *argv[])
{
    int ret = ERROR_OK;
    int n = argc > 1 ? atoi(argv[1]) : 27; //
    convert argv[1] or use default value

    ret = n % 2 == 0 ? ERROR_INPUT : ret; //
    ensure n is odd number
    if (!ret &&
        (n < MIN_VALUE || n > MAX_VALUE)) {
        ret = ERROR_RANGE; //ensure n is in the
        closed interval [MIN_VALUE, MAX_VALUE]
    }
    ...
    return ret;
}
```

Příklad kódování – Strategie implementace 3/4

- Definujeme návratové (chybové) hodnoty (0, 100, 101) využitím `enum`, aby byl „kód čistý“.
- Definujeme platný rozsah (11, 67), (`#define`).
- Zajistíme přístup k argumentům programu pouze tehdy, pokud jsou zadány.
- Kontrolujeme, že počet řádků n je platná hodnota, jinak program vrací chybu.
- Provádíme libovolnou operaci pouze v případě, že argumenty (hodnoty) jsou platné.
- Tisk 7-mi řádků rozdělíme do dvou `for` smyček, mezi smyčkami bude tisk plného `*` řádku.
- Implementujeme samostatnou funkci tisk vzoru řádku.

```
// print a line with n characters with the
// pattern: k-times c, then space.
// the line ends by new line character '\n'.
void print(char c, int n, int k);

int main(int argc, char *argv[])
{
    ...
    if (!ret) { // only if ret == ERROR_OK
        for (int l = 1; l <= LINES; ++l) {
            print('* ', n, l); // print l x '* '
        }
        print('* ', n, n); // print n x '* '
        for (int l = LINES; l > 0; --l) {
            print('* ', n, l); // print l x '* '
        }
    }
    return ret;
}
```

Příklad kódování – Strategie implementace 4/4

- Definujeme návratové (chybové) hodnoty (0, 100, 101) využitím `enum`, aby byl „kód čistý“.
- Definujeme platný rozsah (11, 67), (`#define`).
- Zajistíme přístup k argumentům programu pouze tehdy, pokud jsou zadány.
- Kontrolujeme, že počet řádků n je platná hodnota, jinak program vrací chybu.
- Provádíme libovolnou operaci pouze v případě, že argumenty (hodnoty) jsou platné.
- Tisk 7-mi řádků rozdělíme do dvou `for` smyček, mezi smyčkami bude tisk plného `*` řádku.
- Implementujeme samostatnou funkci tisk vzoru řádku.

```
void print(char c, int n, int k)
{
    for (int i = 0; i < n; ++i) {
        putchar( (i+1) % (k+1) ? c : ' ');
    }
    putchar('\n');
}

■ Řádek se skládá z  $n$  znaků, takže je třeba vypsat  $n$  znaků.
■ Za každým  $k$ -tým znakem  $c$  je mezerka.
■ Násobek  $k$  lze zjistit ze zbytku po celočíselném dělení, operátor %.
■ Ošetříme, že  $i$  začíná od 0.
■ Mezera je každý  $(k+1)$ -tý znak.
```

Příklad kódování – Strategie implementace 4(b)/4

- Definujeme návratové (chybové) hodnoty (0, 100, 101) využitím `enum`, aby byl „kód čistý“.
- Definujeme platný rozsah (11, 67), (`#define`).
- Zajistíme přístup k argumentům programu pouze tehdy, pokud jsou zadány.
- Kontrolujeme, že počet řádků n je platná hodnota, jinak program vrací chybu.
- Provádíme libovolnou operaci pouze v případě, že argumenty (hodnoty) jsou platné.
- Tisk 7-mi řádků rozdělíme do dvou `for` smyček, mezi smyčkami bude tisk plného `*` řádku.
- Implementujeme samostatnou funkci tisk vzoru řádku.

```
void print(char c, int n, int k)
{
    int i, j;
    for (i = j = 0; i < n; ++i, ++j) {
        if (j == k) {
            putchar(' ');
            j = 0;
        } else {
            putchar(c);
        }
    }
    putchar('\n');
}
```

- Použijeme extra proměnnou `j` pro tisk mezery, jako každý k -tý vytištěný znak.
- Využijeme operátor čárky k inkrementaci `j` v rámci smyčky `for`.