

Základy programování v C

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 02

B0B36PRP – Procedurální programování

Část I

Část 1 – Základy programování v C

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

S. G. Kochan: kapitoly 2, 3

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

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 složitě, jsou základní principy relativně jednoduché. I proto je výhodné používat základní nástroje pro překlad programů a po jejich osvojení využít komplexnější vývojové prostředí.

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 BOB36PRP!\n");
6
7     return 0;
8 }

```

lec02/program.c

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> /* vlozeni hlavickoveho souboru
4     standardni knihovny stdio.h */
5
6 int main(void) // zjednodusena deklarace
7 { // hlavni funkce program main()
8     printf("I like BOB36PRP!\n"); /* volani funkce
9     printf() z knihovny stdio.h pro tisk textoveho
10    retezce na standardni vystup. Znak \n definuje novy
11    radek (odradkovani). */
12
13    return 0; /* ukonceni funkce a predani navratove
14    hodnoty 0 operacnimu systemu */
15 }

```

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 BOB36PRP!
```
- 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!

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
 - Popis (seznam) funkcí a jejich parametrů bez konkrétní implementace
 - Znovopoužitelnost**
 - 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!

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

Části překladu 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

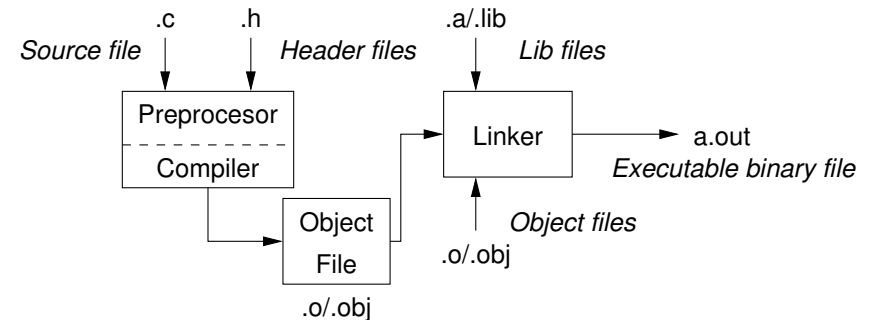
- linker** – sestavuje program z objektových souborů do podoby výsledné aplikace

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

Schéma překladu 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;
- spouštění a ladění aplikace a opětovné editace zdrojových souborů.



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/) <https://www.cygwin.com/> nebo [MinGW](http://www.mingw.org/) <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`)

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

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
11    sum = var1 + var2;
12
13    printf("The sum of %i and %i is %i\n", var1, var2, sum);
14
15    return 0;
16 }

```

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

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

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 v hlavičkovém souboru `<stdint.h>`

IEEE Std 1003.1-2001

```
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>

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
```

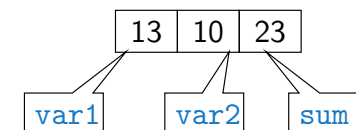
- V původním (ANSI) C explicitní datový typ pro logickou hodnotu není definován.
 - Můžeme však použít podobnou definici jako v `<stdbool.h>`

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

Přřazení, 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

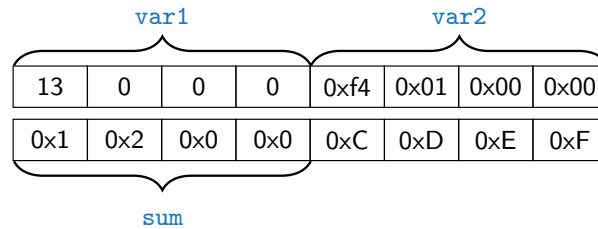


Přiřazení, proměnné a paměť – Vizualizace int

```

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***

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 proměnou: `++` a `--`
 - logický operátor doplněk: `!`
 - operátor přetypování: `(jméno typu)`
- Ternární operátor – podmíněné přiřazení hodnoty

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*

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**
`<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

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 překladači.*

Další aritmetické operátory příště.

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);
14
15     printf("P2 = (%f, %f)\n", x2, y2);
16
17     double dx = (x1 - x2); // implicitni konverze na float, resp.
18     double dy = (y1 - y2);
19
20     printf("P1 - P2)=(%.3f, %0.3f)\n", dx, dy);
21     printf("|P1 - P2|^2=%.2f\n", dx * dx + dy * dy);
22     return 0;
23 }

```

`lec02/points.c`

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; // nasobeni
11 printf("a * b = %i\n", result);
12
13 result = a / b; // celociselne deleni
14 printf("a / b = %i\n", result);
15
16 result = a + b * c; // prioritá operatoru
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/arithmetric_operators.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 řídicí řetězec formátování výstupu
- Řídicí řetězec formátu je uvozen znakem `'%'`
- Znakové posloupnosti (nezačínající `%`) se vypíší tak jak jsou uvedeny
- Základní řídicí řetězce 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 řídicí 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                                     lec02/scanf.c

```

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:** 21.10.2017, 23:59:59 PDT

PDT – Pacific Daylight Time

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**