

Programovací jazyk C: Základy

Programování F1400 + F1400a
doc. RNDr. Petr Mikulík, Ph.D.

podzimní semestr 2020

```
/*  
    Nejjednodussi program v Cecku  
*/  
  
#include <stdio.h>  
  
int main()  
{  
    printf("Hello world!\n");  
    return 0;  
}
```

```
#include <stdio.h>  
  
int main()  
{  
    int x, y1, y2;  
    for (x=-20; x<=20; x++) {  
        y1 = 2*x;  
        y2 = 3 + x*x;  
        printf("%i\t%i\t%i\n", x, y1, y2);  
    }  
    return 0;  
}
```

- **Jazyk C** – historie: 1972 pro naprogramování Unixu, do té doby se operační systémy psaly v assembleru
- Vývoj jazyka – K&R C, pak některá drobná vylepšení, ISO normy: ANSI C a C99
- Současnost: **Podobná syntaxe (zápis příkazů) se používá i v dalších programovacích jazycích:** C++, Java; awk, javascript, ...
- **Programování:**
 - Napsání musí předcházet analýza problému (např. matematická)
 - Vývojové diagramy či jiné čmárání po papíru
 - Program = Algoritmus + Datové struktury + Syntaxe konkrétního jazyka
 - Napsání programu v daném programovacím jazyce
 - 90 % času padne na odladění a vylepšování programu
 - A když se program osvědčil, je třeba ho změnit. . .

- **Nejjednodušší program v C:** na obrazovku vypíše Ahoj, světe!

```
1 /* Toto je muj nejjednodussi program v Cecku. */  
2 #include <stdio.h>  
3 int main()  
4 {  
5     printf("Hello world!\n");  
6     return 0;  
7 }
```

- Srovnajte s gnuplotem: stačilo napsat
gnuplot> print "Hello, world!"

Struktura programu v C: #include, //, /* */

- Popis jednotlivých částí programu:

```
1 // Toto je muj prvni program.    ... jednoradkova poznamka
2 /* Muj program bude neco        ... viceradkova poznamka
3    vypisovat na obrazovku.
4 */
5                                ... prazdny radek nic neznamenava
6 #include <stdio.h>              ... pouzij tyto knihovny
7 int main()                       ... vstupni bod do programu
8 {                                ... zacatek tela funkce main()
9     printf("Hello world!\n");    ... prikaz, kterym neco vypiseme
10    return 0;                    ... vratime systemu 0 pri uspechu
11 }
```

- Na začátku programu uvedeme **seznam knihoven**, které budeme používat. Obvykle se jedná o knihovny pro vstup/výstup, standardní (běžné) funkce, matematické funkce, práce s řetězci atd.
- Do **poznámek** píšeme komentáře k programu, např. na začátku programu datum, autora a požadovanou činnost programu, dále pak třeba popis jednotlivých algoritmů či význam datových struktur (proměnných).

Struktura programu v C: `int main()`, `{}`, `;`;

- Popis jednotlivých částí programu:

```
1 #include <stdio.h>           ... použij tyto knihovny
2 int main()                   ... vstupní bod do programu
3 {                             ... začátek těla funkce main()
4     printf("Hello world!\n"); ... příkaz, kterým něco vypíšeme
5     return 0;                ... vrátíme systému 0 při úspěchu
6 }
```

- Text `int main()` označuje **vstup do programu** a říká, že při ukončení programu vrátíme operačnímu systému nějaké celé číslo (`int` = integer).
- V případě úspěchu **vrátíme systému** nulu příkazem `return 0;`, jinak nějaké nenulové číslo, např. `return 1;`.
- Ve **složených závorkách** je nějaký blok kódu (viz též složené závorky v $\text{T}_{\text{E}}\text{X}$ u).
- **Středníky** oddělují jednotlivé příkazy a jsou velmi důležité! Za některá klíčová slova totiž nepatří (např. cykly).

Struktura programu v C: funkce printf()

- Popis jednotlivých částí programu:

```
1 #include <stdio.h>           ... použij tuto knihovnu
2
3 int main()
4 {
5     printf("Ahoj ");          ... tisk, ale neodradkujeme
6     printf("svete!\n");       ... druhé slovo s odradkovanim.
7     printf("Hello world!\n"); ... celý radek najednou
8     printf("1\t2\t3\t4\n");   ... čísla do sloupce
9     printf("1.0\t2.34\t3.1415\t4.2\n");
10    return 0;
}
```

- Funkce printf() tiskne text (a nejen text...). Kvůli ní je třeba na začátku dokumentu začlenit knihovnu stdio (standard input/output).
- Text k tisku je v uvozovkách. Za zpětným lomítkem jsou tzv. řídicí sekvence, např.
 - \n: newline, značí nový řádek
 - \t: tabulátor, odskok na 8. pozici (tabulky)

- **Zdrojový kód** (zdroják) napíšeme v **textovém editoru** (nebo textovém editoru uvnitř nějakého GUI). Vyrobíme soubor např. `ahoj.c`.
- **Překladač**: zdrojový kód (zdroják) → spustitelná binárka. Použití:

```
gcc ahoj.c -o ahoj      nebo      gcc ahoj.c -o ahoj.exe
clang ahoj.c -o ahoj
cc ahoj.c -o ahoj
```

Vznikne nám spustitelný program `ahoj` (unixové systémy) nebo `ahoj.exe` (VMS, DOS, OS/2, MSW).

- Z příkazové řádky, jsme-li v tom správném pracovním aktuálním adresáři, **spustíme program** na unixu příkazem

```
./ahoj
```

a na ostatních systémech

```
ahoj.exe      nebo      ahoj
```

Samozřejmě ho můžeme pustit i odkliknutím jeho ikony z nějakého souborového manažeru.

- Svobodné vs komerční
 - Tradiční dodavatelé unixových systémů (DEC, HP, ...): překladač cc
 - GNU Compiler Collection: GNU C, gcc
 - MacOSX, nyní i další: LLVM, clang
 - „Novější komerční“: Intel, Borland, Watcom, Microsoft
- Distribuce překladačů obsahuje další potřebné programky (např. kolekci GNU binutils)
- GUI: opravdové GUI vs menítka v textových editorech
 - Geany, Anjuta, Kdevelop4 vs gedit, (g)vim, emacs, ...
 - Dev-C++ vs PSpad, ...

Musíte (měli byste...) vědět či alespoň tušit, co udělá GUI po stlačení ikonky, a **co/kde/proč/jak** nastavit.

- Program pro **výpis hodnot celočíselných výrazů**:

```
1 #include <stdio.h>
2 int main()
3 {
4     printf("Soucet %i a %i je %i\n", 1, 2, 1+2);
5     printf("Soucet %d a %d je %d\n", 1, 2, 1+2);
6     return 0;
7 }
```

- Řetězec v printf() obsahuje formátovací parametry %i nebo %d pro zobrazení hodnot celočíselných výrazů. Obě možnosti jsou synonyma (integer nebo decimal integer).
- Srovnejte s gnuplotem: stačilo napsat
gnuplot> print 1, 2, 1+2

Počítání s celočíselnými proměnnými

- Program pro základní práci s **celočíselnými proměnnými**:

```
1 #include <stdio.h>
2 int main()
3 {
4     int a, b, c; // deklarace promennych, ktere budeme pouzivat
5     a = 7; b = -22; // mezery kolem = pro nasi prehlednost
6     c = a*b-(b+1)*2;
7     printf("Vyras z %i a %i je %i\n", a, b, c);
8     a++; b--; c *= 2; // inkrement, dekrement, samonasobeni
9     printf("vyssi %i nizsi %i dvojnásobek %i\n", a, b, c);
10    return 0;
11 }
```

- Nejdříve se proměnné **deklarují**, zde zavádíme tři proměnné typu integer. Teprve pak je můžeme používat, tj. přiřazovat jim hodnoty. Lze též provést první přiřazení již v deklaraci, např.

```
1 int a=7, b=-22;
```

- Srovnajte s gnuplotem: deklarace typu proměnné není třeba, stačí napsat:
gnuplot> a=1; b=3;
gnuplot> print "Vysledek je ", a

Tabulka hodnot funkce – použití cyklu for

- Počítač má hodně počítat, tak třeba **tabulku hodnot** nějaké funkce.
- Popis programu: pro x od -20 do $+20$ s krokem $\Delta x = 1$ spočítej hodnotu $y_1 = 2x$, $y_2 = 3 + x^2$ a na obrazovku vypiš tabulku se třemi sloupci čísel x , y_1 , y_2 .
- Program tedy musí obsahovat deklaraci proměnných a nějaký cyklus, ve kterém se hodnoty počítají a vypisují na obrazovku.
- Program bude vypadat například takto:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int x, y1, y2; // souradnice a pocitane funkcní hodnoty
6     for (x=-20; x<=20; x++) {
7         y1 = 2*x; // prima umernost
8         y2 = 3 + x*x; // kvadraticka funkce
9         printf("%i\t%i\t%i\n", x, y1, y2);
10    }
11    return 0;
12 }
```

Základní cyklus: možnosti zápisu cyklu for

- **Cyklus for – syntaxe v Céčku a jemu podobných jazycích:**

```
1   for (nastaveni; podminka; prikaz na konci cyklu) {  
2       prikaz1;  
3       prikaz2;  
4       ...  
5   }
```

- **Cyklus for pro cyklení jediného příkazu (složené závorky nejsou nutné):**

```
1   for (nastaveni; podminka; prikaz na konci cyklu)  
2       JedinyPrikaz;
```

- **Cyklus for s vynechávkami:**

```
1   for ( ; podminka; )           for ( ; x<20; )  
2       JedinyPrikaz;             x += 2;
```

- **Cyklus for, který nikdy neproběhne (nesplnitelná podmínka):**

```
1   for ( ; 1 < 0; ) cokoliv;
```

- **Cyklus for nikdy nekončící (stále platná podmínka):**

```
1   for ( ; 0 < 1; ) cokoliv;
```

- **Cyklus for nic nedělající (extra středník – zjevně chyba):**

```
1   for (x=80; x <= 20; x++);  
2       printf("x je %i, trojnásobek je %i\n", x, 3*x);
```

Více cyklů v sobě

- **Cykly lze nořit do sebe.** Ukázka kódu počítajícího tabulku dvou rozměrné funkce $z(x, y) = x^2 - y^2$ (tj. povrch) pro zadaný rozsah hodnot souřadnic x (od -20 do $+20$) a y (od -10 do $+10$):

```
1 #include <stdio.h>
2 int main()
3 {
4     int x, y, z;
5     for (x=-20; x<=20; x++) {
6         for (y=-10; y<=10; y++) {
7             z = x*x - y*y;
8             printf("%i\t%i\t%i\n", x, y, z);
9         }
10    }
11    return 0;
12 }
```

- Abychom se v našem kódu vyznali, je dobré jednotlivé úrovně cyklů řádně odsazovat a obestoupit složenými závorkami.

Poznámka: Jazyk Python nepoužívá složené závorky, jako blok kódu se bere blok se stejným odsazením.

Více cyklů v sobě ... pokračování

- ... a pokud chceme, aby každé dvě po sobě jdoucí isolinie (bloky dat pro konstantní x) byly na výstupu **odděleny prázdným řádkem** (pro ozřejmění struktury dat, např. pro gnuplot), pak do předešlého kódu stačí připsat kód pro vytištění prázdného řádku po každé isolinii:

```
1 #include <stdio.h>
2 int main()
3 {
4     int x, y, z;
5     for (x=-20; x<=20; x++) {
6         for (y=-10; y<=10; y++) {
7             z = x*x - y*y;
8             printf("%i\t%i\t%i\n", x, y, z);
9         }
10        printf("\n"); // přidaný příkaz pro prázdný řádek
11    }
12    return 0;
13 }
```