Programovací jazyk C: Podmínky, větvení, if

Programování F1400 + F1400a

doc. RNDr. Petr Mikulík, Ph.D.

podzimní semestr 2020

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{
    double x = -5.0;
    if (x <= 0) {
        printf("Chci kladne!\n");
        return 1; // exit(1);
    }
    printf("ln(%g)=%g\n",x,log(x));
    return 0;
}
```

```
int a, b;
a = 8 / 2;
b = 2 * 2;
if (a == b) printf("Jsou stejna\n");

double x, y;
x = 1.000000001;
y = 2.000000003 / 2;
if (fabs(x-y) <= fabs(x+y) * 5e-16)
    printf("Asi jsou stejna\n");
```

Podmínky a cykly

- **Podmínky (logické výrazy)** jsou nutné pro větvení programu a pro vykonávání cyklů. Podmínky jsou logické výrazy nabývající hodnoty, které můžeme nazývat pravda či nepravda, 0 nebo 1, false nebo true.
- Např. pravdivým výrazem je $(0 < 1)$, nepravdivým výrazem je $(1 + 2 > 5)$.
- Booleova algebra (boolean algebra) $\rightarrow$ *booleovské* = *logické výrazy*.
- **Podmínky jsme již použili v cyklech** `for` **a** `while`:

```
for ( ...; podmínka; ... ) { příkazy }
while (podmínka) { příkazy }
```

- **Dalším cyklem je** `do`:

```
do {
    prikaz1;
    prikaz2;
    ...
} while (podmínka);
```

- Cyklus `do { ... } while (...)` **proběhne vždy alespoň 1**$\times$.
  Cyklus `while (...) { ... }` **nemusí proběhnout ani jednou**.
- Jiné jazyky pracují **s negací podmínky** – Octave/Matlab:
  `do { ... } until (...)` či Pascal: `repeat { ... } until (...)`

Použití cyklu do ... while

- Příklad: **Program pro nalezení nejmenšího reálného čísla** $x > 0$ **v oboru double** (srovnejte s vaším výsledkem při ručním hledání tohoto čísla v gnuplotu a s programem s cyklem `while() {...}` minule).

  Algoritmus: začnu s hodnotou např. 1.0 a budu ji zmenšovat, dokud se z ní výpočtem (počítač je stroj s omezenou přesností) nestane nula:

```
1    double x, x_new;  // pamatuj si posledni dve vypoctene hodnoty
2    x_new = 1.0;
3    do {                    // opakuj
4        x = x_new;             // zkopiruj predchozi hodnotu do x
5        x_new *= 0.5;          // nova hodnota x_new = x_new/2
6    } while (x_new > 0);   // dokud je odhad nenulovy
7    printf("Nejmensi kladne cislo double je %g\n", x);
```

- Pozn.: totéž pomocí cyklu `while () {...}`:

```
1    double x, x_new = 1.0;
2    while (x_new > 0.0) {
3        x = x_new;           // zkopiruj predchozi hodnotu do x
4        x_new *= 0.5;        // nova hodnota x_new = x_new/2
5    }
6    printf("Nejmensi kladne cislo typu double je %.8g\n", x);
```

Podmínky

- Podmínky pro **menší** a **menší nebo rovno**: `(a < b)` `(a <= b)`
  Podmínky pro **větší** a **větší nebo rovno**: `(a > b)` `(a >= b)`
  Podmínky pro **rovnost** a **nerovnost**: `(a == b)` `(a != b)`
- Jazyk C, **logická vs číselná hodnota**:
  - 0 ... nula znamená nepravda/false,
  - $\neq 0$ ... nenulové číslo znamená pravda/true.

  Tedy (0) nebo $(123 - 123)$ je nepravda, (3.14) nebo $(1 + 2 * 3)$ je pravda.
- **Řetězení podmínek**:
  - logické **nebo** (OR): `0 || 1` je 1; `(1 || 2)`, `(a > b) || (c > b)`
  - logické **a** (AND): `(0 && 1)` je 0; `(0 && 2)`, `(a > b) && (c > b)`
  - logické **exkluzivní nebo** (XOR): `(0 ^ 0)` je 0, `(0 ^ 1)` je 1, ...
- **Negace** výrazu:
  - `!(a == b)`, `!(a>b && c>d)`, `!(a>b || c>d)`
  - `!0`, `!1`, `!3.1415`
- **Priorita operátorů**: V C si výrazy `(a>b && c>d || f>g)` a `((a>b) && (c>d)) || (f>g)` odpovídají.

Větvení if (...) ... else ...

- **Větvení programu**: jestliže něco platí, pak dělej tohle, jinak něco jiného. Část „jinak“ je nepovinná. Příkazy lze psát na jeden i více řádků:

```
1     if (podminka) prikaz;

2     if (podminka) prikaz1; else prikaz2;

3     if (podminka) prikaz1;
4       else prikaz2;

5     if (podminka)
6         prikaz1;
7     else
8         prikaz2;
9
10    if (podminka) {
11        prikaz1a;
12        prikaz1b;
13        ...
14    } else {
15        prikaz2a;
16        prikaz2b;
17        ...
18    }
```

```
POZOR NA FATÁLNÍ CHYBU:

   if (podminka) ;
  prikazVzdy;

PŘÍKAZ prikazVzdy SE PROVEDE VŽDY!
```

Příklad na if: vyhodnocení odmocniny

- Příklad použití `if` při výpočtu odmocniny:

```
1     double x;

2     // x = +81.0;
3     x = -81.0;

4     printf("Odmocnina z %g v oboru realnych cisel ", x);
5     if (x >= 0)
6         printf("je %g\n",  sqrt(x));
7     else
8         printf("neni definovana\n");

9     printf("Odmocnina z realneho %g v komplexnim oboru je ", x);
10    if (x >= 0)
11        printf("+- %g\n",  sqrt(x));
12    else
13        printf("+- %gi\n", sqrt(-x));
```

Příklad na if: okamžité ukončení běhu programu

- Výpočet logaritmu:

```
1  #include <stdio.h>
2  #include <stdlib.h>     ... deklarace funkce exit()
3  #include <math.h>

4  int main()
5  {
6      double x;
7      x = -5;

8      if (x <= 0) {
9          printf("Chybny vstup do programu! Chci pouze kladne cislo!\n");
10 BUD:
11         return 1;
12 ANEBO:
13         exit(1); // okamzite ukonceni programu s nenulovou navratovou hodnotou
14     }

15     printf("Prirozeny logaritmus z cisla %g je %g\n", x, log(x));
16     return 0; // bezne ukonceni programu s navratovou hodnotou 0
17 }
```

Příklad na if: porovnávání reálných čísel

- **Porovnávání celých čísel** je jednoznačné:

```
1    int x, y;
2    x = 8 / 2;
3    y = 2 * 2;
4    if (x == y) printf("Vysledky se rovnaji\n");
```

- **Porovnávání reálných čísel** je problém kvůli zaokrouhlovacím chybám ve výpočtu (srovnejte měření fyzikálních veličin dvěma různými postupy – intervaly spolehlivosti).
  Např. `cos(asin(1.0))` dá místo nuly číslo `6.1232339957367e-17`, `cos(3*pi/2)` dá `-1.8369701987210e-16`.
  Je třeba srovnat, zdali relativní rozdíl dvou čísel je dostatečně malý, tedy např. srovnatelný se **strojovým epsilon**:

```
1    double x, y;
2    x = 1.000000001;
3    y = 2.000000003 / 2;
4    if (fabs(x - y) <= fabs(x + y) * 1e-15)
5        printf("Cisla %.17g a %.17g jsou skoro stejna\n", x, y);
```

Iterační výpočty: princip

- Testování výsledku výpočtu **při iteračních výpočtech**: nejdříve něco spočtu, a poté danou podmínkou otestuji, zda (to stále ještě) musím spočítat znovu (tedy iterovat, cyklit).

  Obvykle cyklus ukončíme poté, až chyba výsledku dosáhne určité hodnoty při zadané (požadované) relativní přesnosti. Potřebujeme tedy nějaký vhodný postup (algoritmus), který **konverguje**. (Další znalosti získáte později např. v numerické matematice.)

  Např. najdi x pro zadanou hodnotu (číslo) $y = f(x)$, neznám-li k zadané funkci $f(x)$ její inverzní funkci $x = F(y)$.
  Ekvivalentní vyjádření problému lze zapsat takto: $y - f(x) = 0$.
  Řešení hledáme na zadaném intervalu např. půlením intervalu.
- **Výpočty řadou**: spočítáme „mnoho“ prvků řady, anebo počítáme řadu tak dlouho, až se dva po sobě jdoucí členy už příliš neliší – dosáhneme požadované přesnosti.
  Znalost matematiky je žádoucí – součet harmonické řady...

Výpočty funkcí a konstant

- **Iterační výpočty**: Mám postup, jak **hádat a zpřesnovat** výsledek. Např. výpočet odmocniny reálného čísla $\sqrt{x}$, hledání minima funkce, hledání kořene polynomu.
- **Výpočty řadou**. Mám analytický vzorec jako **nekonečnou řadu**. Např. **vzorce** pro konstanty:

$$\begin{aligned} \pi &= \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \dots \\ e &= 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \dots \\ 2 &= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots \end{aligned}$$

Nebo např. použitím **Taylorova rozvoje** pro funkce e^x, $\sin(x)$, $\cos(x)$, $\sinh(x)$, ...

Program na výpočet odmocniny

- **Zadání**: Mám zadané x a chci spočítat $\sqrt{x}$.
- **Newtonův algoritmus**
  - Mám-li odhad odmocniny $a \approx \sqrt{x}$, pak zkusím co nejpřesněji spočítat chybu e tohoto odhadu, $a + e = \sqrt{x}$:

$$x = \lim_{e \to 0} (a + e)^2 = \lim_{e \to 0} (a^2 + 2ae + e^2) \approx \lim_{e \to 0} (a^2 + 2ae)$$

$$\longrightarrow e \approx \frac{x - a^2}{2a} = \frac{1}{2} \left(\frac{x}{a} - a \right)$$

  - Tipnu nějaké počáteční a, k němu spočtu přibližnou chybu $e(a, x)$, takže pak by $a + e(a, x)$ mělo být lepším odhadem $\sqrt{x}$ než původní a. Do a dosadím $a + e$, spočítám novou chybu odhadu e, což by mělo vést k nové přesnější hodnotě $a + e$. Tuto opět dosadím do a, spočítám novou chybu odhadu e, ...
  - Algoritmus v počítači: bud provedu předem dané dostatečně velké množství iterací nebo iteruji (hádám) tak dlouho, až bude výsledek dostatečně přesný, tj. už se nebude měnit v rámci dané relativní přesnosti.

Program na výpočet odmocniny

- **Implementace Newtonova algoritmu**
  Nejjednodušší verze, která vždy provede 100 iterací:

```
1     double x, a, e;  // zadane x, aktualni vysledek, odhad
2     int n;           // pocitadlo iteraci

3     x = 81.0;        // jake cislo odmocnit

4     a = x * 0.5;     // pocatecni odhad, napr. x/2
5     n = 0;           // pocatecni odhad je nultou iteraci

6     while (n < 100) {
7         n++;                  // zvetsi pocitadlo iteraci o jednicku
8         e = (x/a - a) * 0.5;  // odhad chyby
9         a += e;               // novy odhad odmocniny
10        printf("%2i. x=%g   a=%-.14f   e=%-.14f\n", n, x, a, e);
11    }

12    printf("Odmocnina z %.16g je %.16g\n", x, a);
13    printf("Relativni chyba vypoctu je %g\n", fabs(e)/x);
```