Programovací jazyk C: Reálná čísla, cyklus while

Programování F1400 + F1400a

doc. RNDr. Petr Mikulík, Ph.D.

podzimní semestr 2020

```
#include <stdio.h>
#include <math.h>
int main()
{
    double x, y1, y2;
    for (x=-3; x<=1.8; x+=0.1) {
        y1 = -2.3 + exp(0.2*x);
        y2 = 7.6 * sin(M_PI*x/180);
        printf("%g\t%f\t%e\n", x, y1, y2);
    }
    return 0;
}
```

```
#include <stdio.h>
int main ( )
{
    double x_pred, x = 1.0;
    while (x > 0.0) {
        x_pred = x;
        x = x*0.5;
    }
    x = x_pred;
    printf("Vyslo: %.8g\n", x);
    return 0;
}
```

Počítání s reálnými proměnnými – deklarace typů

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

```
1  #include <stdio.h>                    // použité knihovny
2  int main()                            // začátek programu
3  {
4     double a, b, c;                    // deklarace reálných proměnných
5     a = 7.21;                          // výpočty
6     b = -1.23e5;
7     c = a * 17.23e-12 -(b+1)/0.45;
8     printf("Vyraz z %g a %g je %g\n", a, b, c);  // tisk výsledků
9     return 0;                          // návrat z programu
10 }
```

- Reálné proměnné **v jednoduché přesnosti** (single precision) se deklarují typem **float** a mají formátovací znak %g.
- Reálné proměnné **v dvojnásobné přesnosti** (double precision) se deklarují typem **double** a mají formátovací znak s prefixem l: %lg, ale lze používat i %g.
- Reálné proměnné **ve čtyřnásobné přesnosti** se deklarují typem **long double** a mají formátovací znak s prefixem L: %Lg.

Formátování výstupu (printf) reálných proměnných

- Předpokládejme reálnou proměnnou typu **double**.
  Pak formátovací zkratky určují **formát výstupu**:
  - %e a %E vynutí exponencielní formát, např. 1.23e-2 nebo 1.234E2,
  - %f zobrazí desetinný rozvoj, např. 0.0123 nebo 123.4,
  - %g pro optimální kombinaci %e a %f.

  Lze navolit **šířku čísla** (počet rezervovaných pozic na řádku) a **počet platných cifer** anebo **desetinných míst**:

```
1    printf("%15.8g nebo %15.8f nebo %15.8e\n", a, a, a);
```

- Další značky ve formátovací syntaxi

```
1    printf("% 15.8g nebo %-15.8g nebo %- 15.8g\n", a, a, a);
```

  znamenají vynechání mezery místo znaménka + u nezáporného čísla, zarovnání vlevo, a obojí.

- Poznámka: pro **celá čísla** to funguje podobně. Např. pro `int k`:

```
1    printf("%7i nebo % 7i nebo %- 7i\n", k, k, k);
```

Výpočet funkčních hodnot – tabulka hodnot funkcí

- Nechme počítač počítat např. **tabulku hodnot** nějakých funkcí:

  $y_1(x) = -2.3 + 0.7\,\mathrm{e}^{0.02x}$

  $y_2(x) = 8.2\sin(x[\mathrm{rad}]) = 8.2\sin(\frac{\pi}{180}x[^\circ])$

- Požadavky na program: pro x od ≈ -360 do ≈ 180 s krokem 0.1 spočítej (tabeluj, interpoluj) průběh dvou funkcí a na obrazovku vypiš tabulku se třemi sloupci.

```
1  #include <stdio.h>
2  #include <math.h>        // knihovna matematickych funkci

3  int main()
4  {
5      double x, y1, y2; // souradnice a pocitane funkcni hodnoty
6      for (x=-360.0; x<=180.01; x+=0.1) {    // cyklus přes x
7          y1 = -2.3 + 0.7 * exp(0.02*x);     // prvni funkce y1(x)
8          y2 = 8.2 * sin(M_PI * x / 180);    // druha funkce y2(x)
9          printf("%g\t%g\t%g\n", x, y1, y2); // tisk výsledků
10     }
11     return 0;
12 }
```

Překlad programu – knihovny a volby

- Překlad programu vyžaduje **přilinkovat knihovnu matematických funkcí** parametrem `-lm` (library of math):

  ```
  gcc tab.c -o tab -lm
  ```

  Pokud zapomenete na přepínač `-lm`, tak bude překladač křičet něco jako `undefined reference to (sin, exp, ...)`
- Po odladění programu chceme, aby výpočet proběhl co **nejrychleji**. Proto přidáme optimalizační parametr `-O2` (v odůvodněných případech i vyšší), kdy překladač provede optimalizaci výsledného binárního kódu.
- Parametr `-s` **odstraní** (strip) trasovací informace a tabulku symbolů a binárka bude menší. Pak tedy např.

  ```
  gcc -s -O2 tab.c -o tab -lm
  ```
- Parametr `-static` vytvoří **statickou binárku** (tj. bez závislosti na dynamických knihovnách):

  ```
  gcc -s -O2 -static tab.c -o tab -lm
  ```

Překlad programu – upozornění na možné chyby

- Při vývoji programu je vhodné, aby překladač kontroloval i případné možné chyby, ač to nejsou přímo chyby v syntaxi jazyka. Jedná se např. o souhlas formátovacích parametrů %g a %i vzhledem k typu proměnné v příkazu `printf()`, použití neinicializované proměnné (např. proměnná používaná pro součet řady, kterou ale nenastavíme na začátku na nulu), apod. Dosáhneme tohoto parametrem `-Wall` (W znamená warnings):

  ```
  gcc tab.c -o tab -lm -Wall
  ```

  Použití tohoto přepínače pro „All Warnings“ můžeme více než doporučit.
- Nebo použijeme překladač clang, který má přívětivější hlášení chyb

  ```
  clang tab.c -o tab -lm -Wall
  ```

Knihovna matematických funkcí math.h

- Pokud zapomenete na

```
1 #include <math.h>
```

  tak bude překladač křičet něco o
  `implicit declaration of function (sin, exp, ...)`, protože nepozná funkce exp() a sin().

- Jaké funkce a konstanty jsou v `math.h` deklarovány? Viz učebnice C, dokumentace, příkaz `man math.h` a nyní též i „někde“ na webu.
  Na unixových systémech najdete tento hlavičkový soubor např. zde:

  `/usr/include/math.h`

  Pozn.: zkuste příkaz `locate math.h` nebo `locate /math.h`.

Základní cyklus: while

- **Cyklus while – obecně:**

```
while (podminka) {
    prikaz1;
    ...
}
```

příklad:

```
x = 0.5;
while (x < 10) {
  x += 1.7;
  printf("x = %g\n", x);
}
```

- **Logické výrazy – porovnávání matematických výrazů:**

```
(1 < 2)  (3 > 2)  (4==4)  (x=>1 && x<=10)  (x>0 || y<0)
```

- Cyklus for pro **cyklení jediného příkazu**:

```
while (podminka)
    JedinyPrikaz;
```

- Cyklus while, který **nikdy neproběhne**:

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

- Cyklus while **nikdy nekončící**:

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

- Cyklus while **nic nedělající**:

```
while (PlatnaPodminka) ;
        NejakyPrikaz;
```

Cyklus while vs cyklus for

- **Jak spolu cykly `while` a `for` souvisí?**
  Obecný zápis a příklad:

```
1 for (poc; podminka; krok) {          for (x=0; x<=10; x++) {
2     nejake; prikazy;                     y=2*x; z=x+y;
3 }                                    }
```

je totéž co

```
1 poc;                                 x=0;
2 while (podminka) {                   while (x<=10) {
3     nejake; prikazy;                     y=2*x; z=x+y;
4     krok;                                x++;
5 }                                    }
```

Nejmenší reálné kladné číslo typu double

- Úkol: Najděte nejmenší kladné číslo $x > 0$ pro typ double.

Nejmenší reálné kladné číslo typu double

- Úkol: Najděte nejmenší kladné číslo $x > 0$ pro typ double.

```
1  #include <stdio.h>
2  int main ( )
3  {
4      double x = 1.0;      // nejaka startovni hodnota
5      double x_predesle;
6      while (x > 0.0) {
7          x_predesle = x; // zapamatuj si predeslou hodnotu
8          x = x*0.5;      // novy odhad zmensenim cisla
9      }
10     x = x_predesle;      // vysledek je v predesle hodnote x
11     printf("Nejmensi kladne cislo typu double je %.8g\n", x);
12     return 0;
13 }
```

Strojové epsilon

- Úkol: Najděte nejmenší číslo ϵ (zvané **strojové epsilon – machine epsilon**) takové, pro které ještě platí $1.0 + \epsilon > 1.0$.

```
1  #include <stdio.h>
2  int main ( )
3  {
4      double sum, eps;
5      sum = 12345; // startovni hodnota: cislo vetsi nez 1.0
6      eps = 1.0;
7      while (sum > 1.0) {
8          eps /= 2;
9          sum = 1.0 + eps;
10     }
11     eps *= 2;
12     printf("Spoctene strojove epsilon: %.8g\n", eps);
13     return 0;
14 }
```

Strojové epsilon

- Výsledek pro různé přesnosti:

```
Presnost float - velikost: 4 B
Spoctene strojove epsilon: 1.1920929e-07 = 2^-23

Presnost double - velikost: 8 B
Spoctene strojove epsilon: 2.220446e-16 = 2^-52

Presnost long double - velikost: 16 B
Spoctene strojove epsilon: 1.0842022e-19 = 2^-63
```