

Základy programování (IZP)

Deváté počítačové cvičení

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
Gabriela Nečasová, inecasova@fit.vutbr.cz



- **Můj profil:** <http://www.fit.vutbr.cz/~inecasova/>
 - Kancelář: A221
 - Konzultační hodiny: po domluvě emailem
 - Karta Výuka → odkaz na osobní stránky:
IZP 2016/2017 Cvičení → Materiály
- **Komunikace:** email – prosím používejte předmět:
IZP - <předmět emailu>

- Seznámení se zadáním **druhého projektu**
- Funkce a řetězce
- Iterační výpočty

SEZNÁMENÍ S DRUHÝM PROJEKTEM

Iterační výpočty (max 7 bodů)

- Obhajoba: 21.11.2016
- Odevzdání: **27.11.2016, 23:59:59** do **WISu**
- Název: **proj2.c**
- Výstup:
 - standardní výstup pro **výpis výsledků** (na wiki)
 - standardní chybový výstup pro **výpis chyb**
- V rámci projektu můžete používat pouze matematické operace **+, -, *, /**
- **Překlad**: nutno využít přepínač **-lm**:

```
gcc -std=c99 -Wall -Wextra -Werror proj2.c  
-lm -o proj2
```

```
./proj2 --log x N  
./proj2 --pow x y N
```

- **--log**: vypočítá přirozený logaritmus čísla x v N iteracích
 - A) Pomocí Taylorova polynomu
 - B) Pomocí zřetězených zlomků (Continued Fractions)
- **--pow**: vypočítá exponenciální funkci z čísla y s obecným základem x v N iteracích
 - A) Pomocí Taylorova polynomu
 - B) Pomocí zřetězených zlomků (Continued Fractions)

- **Zakázáno:** používat funkce z `math.h`
- **Povoleno:**
 - Funkce `log()`, `isnan()`, `isinf()`
 - Konstanty `NAN`, `INFINITY`
- **Nezapomeňte důkladně testovat na merlinovi !!!**

```
double taylor_log(double x, unsigned int n);
```

- x = číslo
- n = počet členů Taylorova polynomu
- Pro $0 < x < 2$

$$\log(1 - x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \dots$$

- Pro $x > 1/2$

$$\log(x) = \sum_{n=1}^{\infty} \frac{\left(\frac{x-1}{x}\right)^n}{n}$$

- Doporučená mezní hodnota mezi polynomy je **1**


```
double cfrac_log(double x, unsigned int n);
```

- **x** = číslo (zde **z**)
- **n** = stupeň rozvoje zřetězeného zlomku

$$\log \left(\frac{1+z}{1-z} \right) = \frac{2z}{1 - \frac{z^2}{3 - \frac{4z^2}{5 - \frac{9z^2}{7 - \frac{16z^2}{9 - \frac{25z^2}{11 - \frac{36z^2}{13 - \dots}}}}}}}$$

```
log(X)      = LOG_X  
cfrac_log(X) = CFRAC_LOG_X  
taylor_log(X) = TAYLOR_LOG_X
```

- **X** – hodnota daná argumentem (printf **%g**)
- **LOG_X** hodnoty `log()` z `math.h`
- **CFRAC_LOG_** hodnoty vypočtené zřetězeným zlomkem
- **TAYLOR_LOG_** hodnoty vypočtené Taylorovým polynomem
- Všechny ***LOG_*** a ***POW** hodnoty odpovídají formátu **%.12g**

```
pow(X,Y)      = POW  
taylor_pow(X,Y) = TAYLOR_POW  
taylorcf_pow(X,Y) = TAYLORCF_POW
```

- **X,Y** – hodnota daná argumentem (printf **%g**)
- **POW** hodnota `pow()` z `math.h`
- **TAYLORCF_POW** hodnoty vypočtené zřetězeným zlomkem
- **TAYLOR_POW** hodnoty vypočtené Taylorovým polynomem
- Všechny ***LOG_*** a ***POW** hodnoty odpovídají formátu **%.12g**

```
double taylor_pow(double x, double y, unsigned int n);
```

```
double taylorcf_pow(double x, double y, unsigned int n);
```

- **n** = počet členů Taylorova polynomu
- **x, y** = odpovídají parametrům funkce **pow** z matematické knihovny

$$\begin{aligned} a^x &= e^{x \cdot \ln(a)} \\ &= 1 + \frac{x \cdot \ln(a)}{1!} + \frac{x^2 \cdot \ln^2(a)}{2!} + \frac{x^3 \cdot \ln^3(a)}{3!} + \dots \end{aligned}$$

pro $a > 0$

- **Funkce se liší pouze tím, jak se počítá přirozený logaritmus ☺**

FUNKCE A ŘETĚZCE

- Vyhledání znaku `c` v řetězci `s`, vrací ukazatel na vyhledaný znak, jinak NULL

```
char* strchr(char* s, int c);
```

- Kopie řetězce `src` do řetězce `dst`, vrací ukazatel na `dst`

```
char* strcpy(char* dst, char* src);
```

- Spojení řetězců `dst` a `src`, výsledek v `dst`

```
char* strcat(char* dst, char* src);
```

- Lexikografické porovnání řetězců `s1` a `s2`

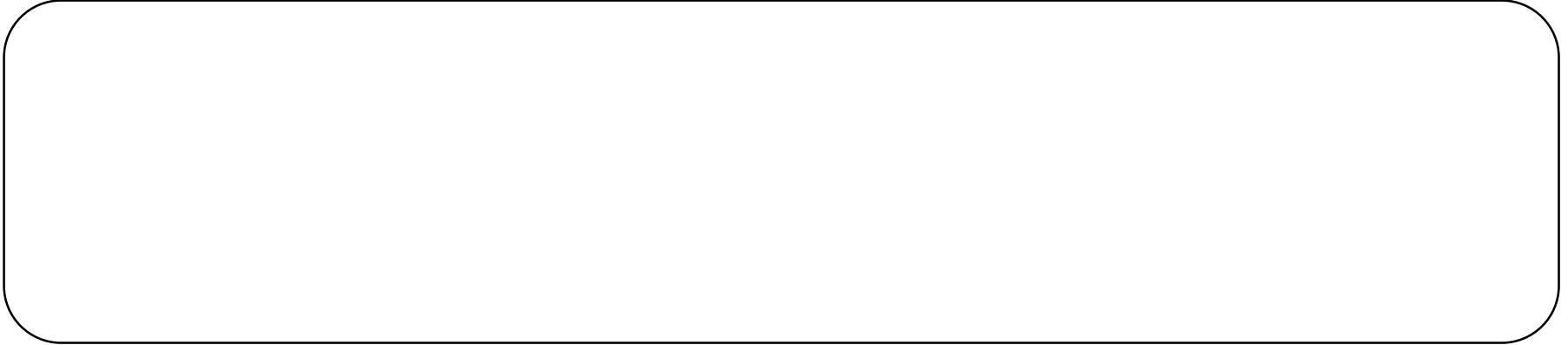
```
int strcmp(char* s1, char* s2);
```

ITERAČNÍ VÝPOČTY

- **Rekurentní problém:** výpočet nové hodnoty závisí na hodnotě výpočtu z předcházejícího kroku
- **Rekurentní vztah** obecně:
$$Y_{i+1} = F(Y_i)$$
- Pro výpočet hodnoty Y_{i+1} je nutné zjistit hodnotu Y_i

- Musí být dána počáteční hodnota Y_0
- Co musí platit pro hodnoty získané posloupnosti
 - $Y_{i+1} = F(Y_i)$ pro $y \geq 0$
 - $Y_i \neq Y_j$ pro všechna $i \neq j$
 - Y_i pro $i < N$ nesplňuje podmínky požadované hodnoty
 - Y_N splňuje podmínky hledané hodnoty

- **Algoritmické schéma** posloupnosti



- **Algoritmické schéma** posloupnosti

```
Y = y0; // Y - proměnná, y0 - počáteční hodnota
```

```
while( $\neg$ B(Y)) // dokud není splněna koncová podmínka
```

```
    Y = F(Y); // budeme počítat další prvek
```

```
    // posloupnosti
```

- **Algoritmické schéma** posloupnosti

```
Y = y0; // Y - proměnná, y0 - počáteční hodnota
```

```
while( $\neg$ B(Y)) // dokud není splněna koncová podmínka
```

```
    Y = F(Y);    // budeme počítat další prvek  
                // posloupnosti
```

- **Zápis v C** může vypadat např.

- **Algoritmické schéma** posloupnosti

```
Y = y0; // Y - proměnná, y0 - počáteční hodnota
```

```
while(¬B(Y)) // dokud není splněna koncová podmínka  
    Y = F(Y); // budeme počítat další prvek  
              // posloupnosti
```

- **Zápis v C** může vypadat např.

```
double y = y0;
```

```
while(!b(y)) // dokud není splněna koncová podmínka  
    y = f(y); // budeme počítat další prvek  
              // posloupnosti
```

```
return y;
```

- Běžně se iterační výpočet ukončí, pokud

$$|Y_i - Y_{i-1}| \leq EPS$$
- To se dá v C zapsat např.

```
double y = y0; // aktuální člen
double yp;     // předchozí člen

do {
    yp = y;     // uložíme hodnotu předchozího členu
    y = f(y);   // vypočítáme další člen
} while (fabs(y - yp) > eps);
```

- Algoritmické schéma lze použít pro výpočet číselných řad (Taylorův rozvoj), kterými lze aproximovat funkce

Posloupnosti

BODOVANÝ ÚKOL 1

- Implementujte funkci, která vypočítá druhou odmocninu \sqrt{x} Newtonovou metodou

$$y_{i+1} = \frac{1}{2} * \left(\frac{x}{y_i} + y_i \right)$$

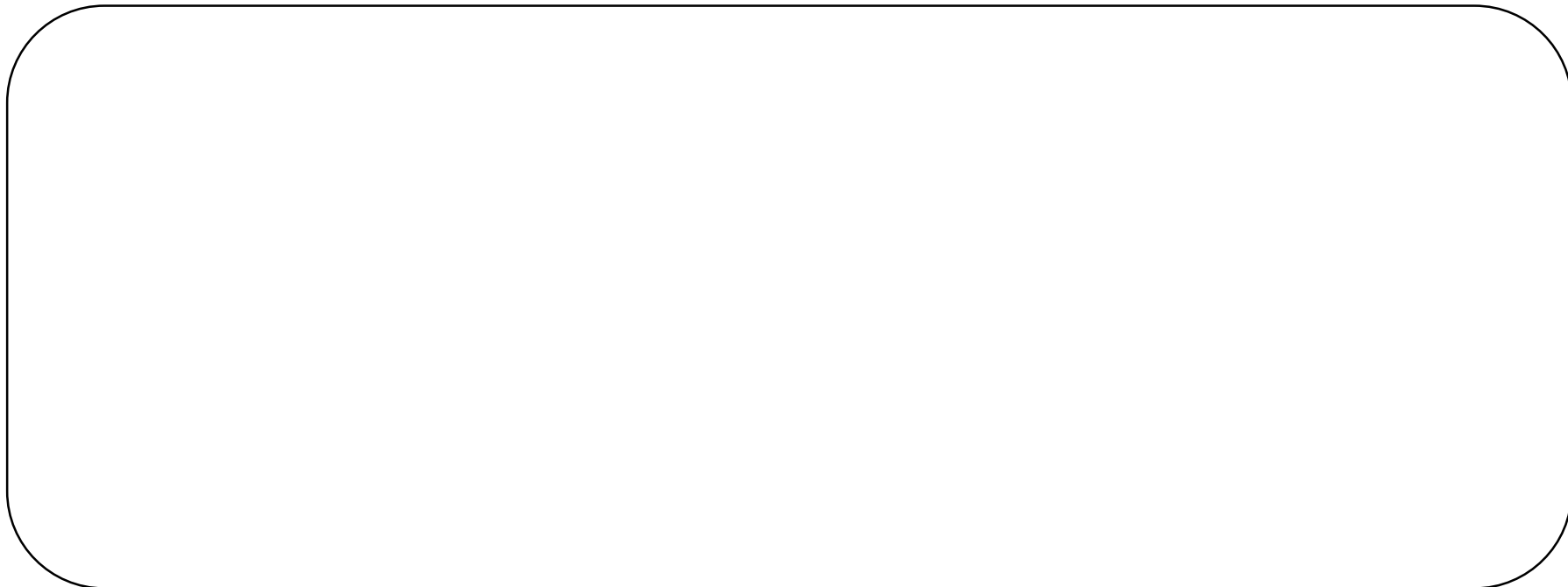
- Prototyp funkce si vhodně zvolte
- **`x = 7`**
- **`eps=1e-19`**
- **Můžete** použít funkci `sqrt(x)` pro ověření funkčnosti řešení
- Výpis: `printf("%.10f", vysledek);`

- K výpočtu řad se používají **částečné součty**
- Pro řadu $t_0, t_1, t_2, t_3, \dots$, kde $t_i = f(t_{i-1})$ můžeme napsat řadu částečných součtů

$$s_0, s_1, s_2, s_3, \dots, \text{ kde } s_i = \sum_{j=0}^i t_j$$

- Můžeme je opět řešit **rekurentně**:
 - $s_0 = t_0$
 - $s_1 = s_0 + t_1 = t_0 + t_1$
 - $s_i = s_{i-1} + t_i$
 - \rightarrow částečný součet pro aktuální člen je částečný součet pro předchozí člen + hodnota aktuálního členu

- **Algoritmické schéma řady**



- **Algoritmické schéma řady**

```
T = t0; // první člen řady
S = T;  // součet = první člen řady
while(¬B(S, T))
{
    T = f(T); // vypočítáme nový člen řady
    S = S + T; // tento člen přičteme k aktuálnímu
               // částečnému součtu
}
```

- Je nutné si vždy zjistit, jak se od sebe liší jednotlivé členy řady
- **Pozor:** Některé řady konvergují nejrychleji jen v omezeném definičním oboru funkce

```
Y = y0; // Y - proměnná, y0 - počáteční hodnota
```

```
while(¬B(Y)) // dokud není splněna koncová podmínka  
    Y = F(Y); // budeme počítat další prvek  
                // posloupnosti
```

```
T = t0; // první člen řady
```

```
S = T; // součet = první člen řady
```

```
while(¬B(S, T))
```

```
{
```

```
    T = f(T); // vypočítáme nový člen řady
```

```
    S = S + T; // tento člen přičteme k aktuálnímu  
                // částečnému součtu
```

```
}
```

Částečné součty (řady)

BODOVANÝ ÚKOL 2

Pomocí **částečných součtů** implementuje výpočet **exponenciální funkce** e^x . (Taylorova) řada má následující tvar:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

Výsledek **porovnejte** s matematickou knihovnou `math.h` a **obě hodnoty vypište** na standardní výstup

- Výpis: `printf("%.10f", vysledek);`

Nemůžete použít **mocninu**, **faktoriál** a funkci `exp(x)` z matematické knihovny `math.h`.

Můžete použít funkci `fabs()` pro výpočet absolutní hodnoty a `exp(x)` pro ověření funkčnosti řešení

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

```
x = 4.2;
eps = 0.01;
double t = t0; // první člen řady
double s = t;  // součet = první člen řady
int i = 1;     // index aktuálního členu řady
while (fabs(t) > eps) {
    t = f(t, i); // vypočítáme nový člen řady
    s = s + t;   // tento člen přičteme k
                // aktuálnímu částečnému součtu
}
return s;
```

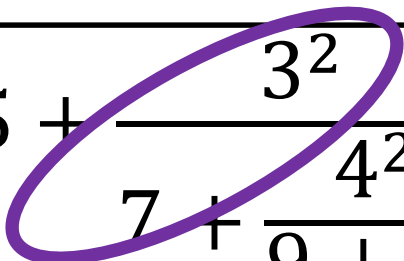
Continued Fractions

ZŘETĚZENÉ ZLOMKY

$$\pi = \frac{4}{1 + \frac{1^2}{3 + \frac{2^2}{5 + \frac{3^2}{7 + \frac{4^2}{9 + \dots}}}}}$$

$$\pi = \frac{4}{1 + \frac{1^2}{3 + \frac{2^2}{5 + \frac{3^2}{7 + \frac{4^2}{9 + \dots}}}}}$$

n = 4



$$\pi = \frac{4}{1 + \frac{1^2}{3 + \frac{2^2}{5 + \frac{3^2}{7 + \frac{4^2}{9 + \dots}}}}}$$

$n = 3$

$n = 4$

$$\pi = \frac{4}{1 + \frac{1^2}{3 + \frac{2^2}{5 + \frac{3^2}{7 + \frac{4^2}{9 + \dots}}}}}$$

$n = 2$ (blue)
 $n = 3$ (green)
 $n = 4$ (purple)

The diagram shows the continued fraction expansion of pi. The terms are arranged in a nested structure. The first term is 4. The denominator is 1 plus a fraction. The next level has a denominator of 3 plus a fraction. The next level has a denominator of 5 plus a fraction. The next level has a denominator of 7 plus a fraction. The next level has a denominator of 9 plus a fraction. The ellipses indicate the pattern continues. Three nested ellipses are drawn around the terms: a blue ellipse around the first three terms (4, 1, 3), a green ellipse around the next three terms (1, 3, 5), and a purple ellipse around the next three terms (3, 5, 7). To the left of the diagram, the values n=2, n=3, and n=4 are listed in blue, green, and purple respectively, corresponding to the ellipses.

$$\begin{array}{lcl}
 \pi = & & 4 \\
 & & \hline
 n = 1 & 1 + & \frac{1^2}{4} \\
 & & \hline
 n = 2 & 3 + & \frac{2^2}{3 + \frac{3^2}{5 + \frac{4^2}{7 + \frac{4^2}{9 + \dots}}}} \\
 & & \hline
 n = 3 & 5 + & \frac{3^2}{5 + \frac{4^2}{7 + \frac{4^2}{9 + \dots}}} \\
 & & \hline
 n = 4 & 7 + & \frac{4^2}{9 + \dots}
 \end{array}$$

Implementujte výpočet čísla π pomocí zřetězeného zlomku:

$$\pi = \frac{4}{1 + \frac{1^2}{3 + \frac{2^2}{5 + \frac{3^2}{7 + \frac{4^2}{9 + \dots}}}}}$$

PŘÍKLADY K PROCVIČENÍ

- Implementujte funkci

```
void getMax(int *pole, int len, int *max);
```

která vyhledá v poli maximální hodnotu a vrátí ji přes ukazatel

- Implementujte funkci

```
void getMax(int *pole, int len, int *max);
```

která vyhledá v poli maximální hodnotu a vrátí ji přes ukazatel

- Jak inicializujeme pole?

- Implementujte funkci

```
void getMax(int *pole, int len, int *max);
```

která vyhledá v poli maximální hodnotu a vrátí ji přes ukazatel

- Jak inicializujeme pole?

```
int pole[10]={7,2,3,9,15,20,-1,42,100,-75};
```

- Implementujte funkci

```
void getMax(int *pole, int len, int *max);
```

která vyhledá v poli maximální hodnotu a vrátí ji přes ukazatel

- Jak inicializujeme pole?

```
int pole[10]={7,2,3,9,15,20,-1,42,100,-75};
```

- Jak zavoláme funkci `getMax()` ?

- Implementujte funkci

```
void getMax(int *pole, int len, int *max);
```

která vyhledá v poli maximální hodnotu a vrátí ji přes ukazatel

- Jak inicializujeme pole?

```
int pole[10]={7,2,3,9,15,20,-1,42,100,-75};
```

- Jak zavoláme funkci `getMax()` ?

```
int maximum = 0;  
getMax(pole, &maximum);
```

Implementujte výpočet čísla π pomocí zřetězeného zlomku:

$$\frac{4}{\pi} = 1 + \frac{1}{2 + \frac{9}{2 + \frac{25}{2 + \dots}}}$$

Čitatele se vypočítají podle vztahu
 $(2 * n - 1)^2$

- Implementujte si vlastní funkci pro lexikografické porovnání dvou řetězců

```
int strcmpMy(char* s1, char* s2);
```

- Funkce vrátí:
 - 0 pokud `s1 == s2`
 - -1 pokud `s1 < s2`
 - 1 pokud `s1 > s2`
- Vyzkoušejte implementovat i další funkce z knihovny `string.h`

Děkuji Vám za pozornost!