

## Standardní knihovny C. Rekurze.

Jan Faigl

Katedra počítačů  
Fakulta elektrotechnická  
České vysoké učení technické v Praze

Přednáška 07

B0B36PRP – Procedurální programování

### Přehled témat

- Část 1 – Standardní knihovny, čtení/zápis ze/do souboru  
Standardní knihovny  
Práce se soubory  
Zpracování chyb
- Část 2 – Rekurze  
Faktoriál  
Obrácený výpis  
Hanojské věže  
Rekurze  
Fibonacciho posloupnost
- Část 3 – Zadání 7. domácího úkolu (HW07)

S. G. Kochan: kapitola 16, Appendix B

Standardní knihovny

Práce se soubory

Zpracování chyb

## Část I

### Část 1 – Standardní knihovny, čtení/zápis ze souboru

Jan Faigl, 2022 B0B36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 1 / 53  
Standardní knihovny Práce se soubory Zpracování chyb

### Standardní knihovny

- Jazyk C sám osobě neobsahuje prostředky pro vstup/výstup dat, složitější matematické operace ani:
  - práci s textovými řetězci;
  - správu paměti pro dynamické přidělování;
  - vyhodnocení běhových chyb (run-time errors).
- Tyto a další funkce jsou obsaženy ve standardních knihovnách, které jsou součástí překladače jazyka C.
  - **Knihovny** – přeložený kód se připojuje k programu, např. `libc.so`.
  - **Hlavičkové soubory** – obsahují prototypy funkcí, definice typů, makra a konstanty a vkládají se do zdrojových souborů příkazem preprocesoru `#include <jmeno_knihovny.h>`.  
Viz např. `1dd.a.out`.  
Např. `#include <stdio.h>`

Jan Faigl, 2022 B0B36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 5 / 53  
Standardní knihovny Práce se soubory Zpracování chyb

### Základní práce se soubory – otevření souboru

- Knihovna `stdio.h`.
- Přístup k souboru je prostřednictvím ukazatele `FILE*`.
- Otevření souboru `FILE *fopen(char *filename, char *mode);`
- Práce s textovými a binárními (*modifikátor "b"*) soubory.
- Soubory jsou čteny/zapisovány sekvencně.
  - Se soubory se pracuje jako s proudem dat — postupně načítání/zápis.
  - *Aktuální „pozici“ v souboru si můžeme představit jako kurzor.*
  - *Při otevření souboru se kurzor nastavuje na začátek souboru.*
- Režim práce se souborem je dán hodnotou proměnné `mode`
  - `"r"` – režim čtení;  
`"r" – čtení textového souboru, "rb" – čtení binárního souboru`
  - `"w"` – režim zápisu;  
`"w" – režim zápisu; Vytvoří soubor, pokud neexistuje, jinak smaže obsah souboru`
  - `"a"` – režim přidávání do souboru.  
`"a" – režim přidávání do souboru. Kurzor je nastaven na konec souboru.`
- Můžeme kombinovat s dalšími režimy otevření souboru, např. `"r+"` pro otevření souboru pro čtení i zápis.  
`viz man fopen`

Jan Faigl, 2022 B0B36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 9 / 53

Jan Faigl, 2022 B0B36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 2 / 53  
Standardní knihovny Práce se soubory Zpracování chyb

### Standardní knihovny

- `stdio.h` – Vstup a výstup (formátovaný i neformátovaný)
- `stdlib.h` – Matematické funkce, alokace paměti, převod řetězců na čísla, řazení (`qsort`), vyhledávání (`bsearch`), generování náhodných čísel (`rand`)
- `limits.h` – Rozsahy číselných typů
- `math.h` – Matematické funkce
- `errno.h` – Definice chybových hodnot
- `assert.h` – Zpracování běhových chyb
- `ctype.h` – Klasifikace znaků (`char`)
- `string.h` – Řetězce, blokové přenosy dat v paměti (`memcpy`)
- `locale.h` – Internacionalizace
- `time.h` – Datum a čas

Jan Faigl, 2022 B0B36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 6 / 53  
Standardní knihovny Práce se soubory Zpracování chyb

### Testování – otevření/zavření souboru

- Testování otevření souboru.

```
1 char *fname = "file.txt";
2
3 if ((f = fopen(fname, "r")) == NULL) {
4     fprintf(stderr, "Error: open file '%s'\n", fname);
5 }
```
- Zavření souboru – `int fclose(FILE *file);`

```
1 if (fclose(f) == EOF) {
2     fprintf(stderr, "Error: close file '%s'\n", fname);
3 }
```
- Dosažení konce souboru – `int feof(FILE *file);`

Jan Faigl, 2022 B0B36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 10 / 53


Jan Faigl, 2022 B0B36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 3 / 53  
Standardní knihovny Práce se soubory Zpracování chyb

### Standardní knihovny (POSIX)

Komunikace s operačním systémem (OS).

POSIX – Portable Operating System Interface

- `stdlib.h` – Funkce využívají prostředků OS
- `signal.h` – Asynchronní události, vlákna
- `unistd.h` – Procesy, čtení/zápis souborů, ...
- `pthread.h` – Vlákna (POSIX Threads)
- `threads.h` – Standardní knihovna pro práci s vlákny (C11)

 Advanced Programming in the UNIX Environment, 3rd edition,  
W. Richard Stevens, Stephen A. Rago Addison-Wesley, 2013,  
ISBN 978-0-321-63773-4

Jan Faigl, 2022 B0B36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 7 / 53  
Standardní knihovny Práce se soubory Zpracování chyb

### Příklad – čtení souboru znak po znaku

- Čtení znaku: `int getc(FILE *file);`
- Hodnota znaku (`unsigned char`) je vrácena jako `int`.

```
1 int count = 0;
2 while ((c = getc(f)) != EOF) {
3     printf("Read character %d is '%c'\n", count, c);
4     count++;
5 }
```

`lec07/read_file.c`
- Pokud nastane chyba nebo konec souboru vrací funkce `getc` hodnotu `EOF`.
- Pro rozlišení chyby a konce souboru lze využít funkce `feof()` a `ferror()`.

Jan Faigl, 2022 B0B36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 11 / 53

Standardní knihovny Práce se soubory Zpracování chyb

### Formátované čtení z textového souboru

- int fscanf(FILE \*file, const char \*format, ...);
- Analogie formátovanému vstupu.

*Pro vyplnění hodnot proměnných předáváme ukazatel.*

```
record 1 13.4
```

- Vrací počet přečtených položek, například pro vstup

```
r = 3.
```

- Při čtení textového řetězce je nutné zajistit dostatečný paměťový prostor pro načítání textový řetězec, např. omezením velikosti řetězce.

```
char str[10];
int r = fscanf(f, "%9s %d %lf\n", str, &i, &d);
```

lec07/file\_scanf.c

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 12 / 53

Standardní knihovny Práce se soubory Zpracování chyb

### Zápis do textového souboru

- Zápis po znaku – int putc(int c, FILE \*file);
- Formátovaný výstup int fprintf(FILE \*file, const \*format, ...);

```
int main(int argc, char *argv[])
{
    char *fname = argc > 1 ? argv[1] : "out.txt";
    FILE *f;
    if ((f = fopen(fname, "w")) == NULL) {
        fprintf(stderr, "Error: Open file '%s'\n", fname);
        return -1;
    }
    fprintf(f, "Program arguments argc: %d\n", argc);
    for (int i = 0; i < argc; ++i) {
        fprintf(f, "argv[%d]='%s'\n", i, argv[i]);
    }
    if (fclose(f) == EOF) {
        fprintf(stderr, "Error: Close file '%s'\n", fname);
        return -1;
    }
    return 0;
}
```

lec07/file\_printf.c

- Identicky k stderr lze použít stdout a stdin pro čtení.

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 13 / 53

Standardní knihovny Práce se soubory Zpracování chyb

### Náhodný přístup k souborům – fseek()

- Nastavení pozice kurzoru v souboru relativně vůči whence v bajtech.
- int fseek(FILE \*stream, long offset, int whence);, kde whence
  - SEEK\_SET – nastavení pozice od začátku souboru;
  - SEEK\_CUR – relativní hodnota vůči současné pozici v souboru;
  - SEEK\_END – nastavení pozice od konce souboru.
- fseek() vrací 0 v případě úspěšného nastavení pozice.

- Nastavení pozice v souboru na začátek.
 

```
void rewind(FILE *stream);
```

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 14 / 53

Standardní knihovny Práce se soubory Zpracování chyb

### Binární čtení/zápis z/do souboru

- Otevření souboru s příznakem "b".
 

*Vliv na řetězec, řídicí znaky např. "\0", \n nebo EOF či EOT – Ctrl+Z.*
- Pro čtení a zápis bloku dat můžeme využít funkce fread() a fwrite() z knihovny stdio.h.
- Načtení nmemb prvků, každý o velikosti size bajtů.
 

```
size_t fread(void* ptr, size_t size, size_t nmemb, FILE *stream);
```
- Zápis nmemb prvků, každý o velikosti size bajtů.
 

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```
- Funkce vrací počet přečtených/zapsaných prvků o velikosti size.
- Pokud došlo k chybě nebo detekci konce souboru, funkce vrací menší než očekávaný počet bajtů.

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 15 / 53

Standardní knihovny Práce se soubory Zpracování chyb

### Zpracování chyb

- Základní chybové kódy jsou definovány v <errno.h>.
- Tyto kódy jsou ve standardních C knihovnách používány jako příznaky nastavené v případě selhání volání funkce v globální proměnné errno.
- Například otevření souboru fopen() vrací hodnotu NULL, pokud se soubor nepodařilo otevřít.
- Z této hodnoty, ale nepoznáme proč volání selhalo.
- Pro funkce, které nastavují errno, můžeme podle hodnoty identifikovat důvod chyby.
- Textový popis číselných kódů pro standardní knihovnu C je definován v <string.h>.
- Řetězec můžeme získat voláním funkce
 

```
char* strerror(int errnum);
```

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 17 / 53

Standardní knihovny Práce se soubory Zpracování chyb

### Příklad použití errno (chyba při otevření souboru)

```
1 #include <stdio.h>
2 #include <errno.h>
3 #include <string.h>
4
5 int main(int argc, char *argv[])
6 {
7     FILE *f = fopen("soubor.txt", "r");
8     if (f == NULL) {
9         int r = errno;
10        printf("Open file failed errno value %d\n", errno);
11        printf("String error '%s'\n", strerror(r));
12    }
13    return 0;
14 }
```

lec07/errno.c

- Výstup při neexistujícím souboru
 

```
Open file failed errno value 2
String error 'No such file or directory'
```
- Výstup při pokusu otevřít soubor bez práv přístupu k souboru
 

```
Open file failed errno value 13
String error 'Permission denied'
```

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 18 / 53

Standardní knihovny Práce se soubory Zpracování chyb

### Testovací makro assert()

- Do kódu lze přidat podmínky na nutné hodnoty proměnných.
- Testovat můžeme makrem assert(expr) z knihovny <assert.h>.
- Pokud není expr logická 1 (true) program se ukončí a vypíše jméno zdrojového souboru a číslo řádku.
- Makro vloží příslušný kód do programu.
 

*Získáme tak relativně jednoduchý způsob indikace případné chyby, např. nevhodným argumentem funkce.*
- Vložení makra lze zabránit kompilací s definováním makra NDEBUG.
 

man assert
- Makro assert má význam zejména při ladění, např. testování hodnot předávaných parametrů.
 

```
#include <stdio.h>
#include <assert.h>

int main(int argc, char *argv[])
{
    assert(argc > 1);
    printf("program argc: %d\n", argc);
    return 0;
}
```

lec07/assert.c

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 19 / 53

Standardní knihovny Práce se soubory Zpracování chyb

### Příklad použití makra assert()

- Kompilace s makrem a spuštění programu bez/s argumentem.
 

```
clang assert.c -o assert
./assert
Assertion failed: (argc > 1), function main, file assert.c, line 5.
zsh: abort ./assert
./assert 2
start argc: 2
```
- Kompilace bez makra a spuštění programu bez/s argumentem.
 

```
clang -DNDEBUG assert.c -o assert
./assert
program start argc: 1
./assert 2
program start argc: 2
```

lec07/assert.c

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 20 / 53

Standardní knihovny Práce se soubory Zpracování chyb

### Test alokace paměti a předčasné ukončení programu

- Dynamické přidělení paměti (malloc) je vhodné vždy kontrolovat.
- Pragmaticky můžeme očekávat typický průběh program bezchybný, včetně dynamické alokace paměti.
- Pak může být vhodné přidělení paměti kontrolovat, ale předčasně program ukončit v případě chyby.
- Nicméně stále je vhodné dát uživateli možnost dozvědět se, proč se program předčasně ukončil.
- Můžeme si tak napsat vlastní funkci (makro) my\_assert, které ovšem nelze vyřadit kompilací s -NDEBUG.

```
#ifndef __MY_ASSERT_H__
#define __MY_ASSERT_H__
...
int *a = malloc(SIZE * sizeof(*a));
my_assert(a, __LINE__, __FILE__);
...
#endif
```

lec07/demo-my\_assert.c

```
#define my_assert(x, line, file) \
if (!(x)) {\
    fprintf(stderr, "my_assert fail, line: %d,\
file %s\n", line, file);\
    exit(-1);\
}
```

lec07/my\_assert.h

- Při chybě dokážeme ve zdrojovém souboru dohledat místo a důvod chyb.

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze. 21 / 53

### Příkazy dlouhého skoku

- Příkaz `goto` je možné použít pouze v rámci jedné funkce.
- Knihovna `<setjmp.h>` definuje funkce `setjmp()` a `longjmp()` pro skoky mezi funkcemi.
- `setjmp()` uloží aktuální stav registrů procesoru a pokud funkce vrátí hodnotu různou od 0, došlo k volání `longjmp()`.
- Při volání `longjmp()` jsou hodnoty registrů procesoru obnoveny a program pokračuje od místa volání `setjmp()`.

*Kombinací `setjmp()` a `longjmp()` lze využít pro implementaci ošetření výjimečných stavů podobně jako `try-catch` v jiných programovacích jazycích.*

```

1 #include <setjmp.h>
2 jmp_buf jb;
3 int compute(int x, int y);
4 void error_handler(void);
5 if (setjmp(jb) == 0) {
6     r = compute(x, y);
7     return 0;
8 } else {
9     error_handler();
10    return -1;
11 }
12 int compute(int x, int y) {
13     if (y == 0) {
14         longjmp(jb, 1);
15     } else {
16         x = (x + y * 2);
17         return (x / y);
18     }
19 }
20 void error_handler(void) {
21     printf("Error\n");
22 }

```

## Část II Část 2 – Rekurse

### Výpočet faktoriálu

- Iterace  $n! = n(n-1)(n-2)\dots 2 \cdot 1$
- Rekurse  $n! = 1$  pro  $n \leq 1$   
 $n! = n(n-1)!$  pro  $n > 1$

```

int factorialI(int n)
{
    int f = 1;
    for (; n > 1; --n) {
        f *= n;
    }
    return f;
}

int factorialR(int n)
{
    int f = 1;
    if (n > 1) {
        f = n * factorialR(n-1);
    }
    return f;
}

```

lec07/demo-factorial.c

### Příklad výpis posloupnosti 1/3

- Vytvořte program, který přečte posloupnost čísel a vypíše ji v opačném pořadí.
- Rozklad problému:
  - Zavedeme abstraktní příkaz: „*obrat posloupnost*“.
  - Příkaz rozložíme do tří kroků:
    1. Přečti číslo; Číslo uložíme pro pozdější „obracený“ výpis.
    2. Pokud není detekován konec posloupnosti „*obrat posloupnost*“; Pokračujeme ve čtení čísel.
    3. Vypiš číslo. Vypíšeme uložené číslo.

### Příklad výpis posloupnosti 2/3

```

1 void reverse(void);
2 int main(void)
3 {
4     fprintf(stderr, "Enter a sequence of numbers (use Ctrl+D for the end of
5     the sequence)\n");
6     reverse();
7     printf("\n");
8     return 0;
9 }
10 void reverse(void)
11 {
12     int v;
13     if (scanf("%i", &v) == 1) {
14         reverse();
15         printf("%3d ", v);
16     }
17 }

```

lec07/demo-revert\_sequence.c

### Příklad výpis posloupnosti 3/3

- `lec07/demo-revert_sequence.c`
- Vytvoření posloupnosti.
 

```
./generate_numbers.sh | tr '\n' ' ' | cat > numbers.txt
clang demo-revert_sequence.c
./a.out < numbers.txt 2>/dev/null > numbers-r.txt
./a.out < numbers-r.txt 2>/dev/null > numbers-rr.txt
```
- Příkaz pro výpis obsahu souborů.
 

```
for i in numbers.txt numbers-r.txt numbers-rr.txt; do echo "$i"; cat
    $i; echo ""; done
```
- Výpis obsahu souborů.
 

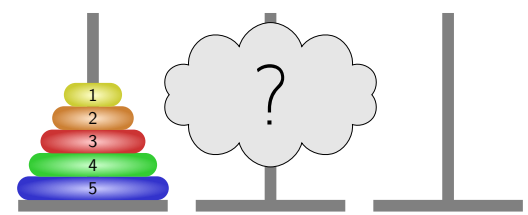
```
numbers.txt
10 4 20 8 8 5 18 6 7 7
numbers-r.txt
7 7 6 18 5 8 8 20 4 10
numbers-rr.txt
10 4 20 8 8 5 18 6 7 7
```

### Příklad Hanojské věže

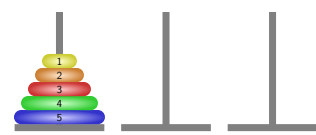


- Přemístit disky na druhou jehlu s použitím třetí (pomocné) jehly za dodržení následujících pravidel.
  1. V každém kroku můžeme přemístit pouze jeden disk a to vždy z jehly na jehlu. *Disky nelze odkládat mimo jehly.*
  2. Položit větší disk na menší není dovoleno.

### Hanojské věže – 5 disků



### Návrh řešení



- Zavedeme abstraktní příkaz `moveTower(n, 1, 2, 3)` realizující přesun  $n$  disků z jehly 1 na jehlu 2 s použitím jehly 3.
- Pro  $n > 0$  můžeme příkaz rozložit na tři jednodušší příkazy:
  1. `moveTower(n-1, 1, 3, 2);` Přesun  $n-1$  disků z jehly 1 na jehlu 3.
  2. „*přenes disk z jehly na jehlu 2*“; Přesun největšího disku na cílovou pozici.
  3. `moveTower(n-1, 3, 2, 1);` Přesun  $n-1$  disků na cílovou pozici.

Faktoriál    Obrácený výpis    Hanojské věže    Rekurze    Fibonacciho posloupnost

### Příklad řešení

```

1 void moveTower(int n, int from, int to, int tmp)
2 {
3     if (n > 0) {
4         moveTower(n-1, from, tmp, to); //move to tmp
5         printf("Move disc from %i to %i\n", from, to);
6         moveTower(n-1, tmp, to, from); //move from tmp
7     }
8 }
9
10 int main(int argc, char *argv[])
11 {
12     int numberOfDiscs = argc > 1 ? atoi(argv[1]) : 5;
13     moveTower(numberOfDiscs, 1, 2, 3);
14     return 0;
15 }

```

lec07/demo-towers\_of\_hanoi.c

Jan Faigl, 2022    BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze.    34 / 53

Faktoriál    Obrácený výpis    Hanojské věže    Rekurze    Fibonacciho posloupnost

### Příklad výpisu

```

■ lec07/demo-towers_of_hanoi.c

clang demo-towers_of_hanoi.c      clang demo-towers_of_hanoi.c
./a.out 3                          ./a.out 4
Move disc from 1 to 2              Move disc from 1 to 3
Move disc from 1 to 3              Move disc from 1 to 2
Move disc from 2 to 3              Move disc from 3 to 2
Move disc from 1 to 2              Move disc from 1 to 3
Move disc from 3 to 1              Move disc from 2 to 1
Move disc from 3 to 2              Move disc from 1 to 3
Move disc from 1 to 2              Move disc from 1 to 2
Move disc from 3 to 2              Move disc from 3 to 2
Move disc from 3 to 1              Move disc from 3 to 1
Move disc from 2 to 1              Move disc from 2 to 1
Move disc from 3 to 2              Move disc from 3 to 2
Move disc from 1 to 3              Move disc from 1 to 3
Move disc from 1 to 2              Move disc from 1 to 2
Move disc from 3 to 2              Move disc from 3 to 2

```

Jan Faigl, 2022    BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze.    35 / 53

Faktoriál    Obrácený výpis    Hanojské věže    Rekurze    Fibonacciho posloupnost

### Rekurzivní algoritmy

- Rekurzivní funkce jsou přímou realizací rekurzivních algoritmů.
- Rekurzivní algoritmus předepisuje výpočet „shora dolů“.
- V závislosti na velikosti vstupních dat je výpočet předepsán:
  - Pro nejmenší (nejjednodušší) vstup je výpočet předepsán přímo;
  - Pro obecný vstup je výpočet předepsán s využitím téhož algoritmu pro menší vstup.
- Výhodou rekurzivních funkcí je jednoduchost a přehlednost.

Jan Faigl, 2022    BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze.    37 / 53

Faktoriál    Obrácený výpis    Hanojské věže    Rekurze    Fibonacciho posloupnost

### Rekurzivní vs iteračními algoritmy

- Nevýhodou rekurzivních algoritmů může být časová náročnost způsobená např. zbytečným opakováním výpočtu.
- Řadu rekurzivních algoritmů lze nahradit iteračními, které počítají výsledek „zdola nahoru“, tj. od menších (jednodušších) vstupních dat k větším (složitějším).
- Pokud algoritmus výpočtu „zdola nahoru“ nenajdeme, např. při řešení problému Hanojských věží, lze rekurzi odstranit využitím zásobníku.  
*Např. zásobník využijeme pro uložení stavu řešení problému.*

Jan Faigl, 2022    BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze.    38 / 53

Faktoriál    Obrácený výpis    Hanojské věže    Rekurze    Fibonacciho posloupnost

### Rekurze

*“To iterate is human, to recurse divine.”*

L. Peter Deutsch

<http://www.devtopics.com/101-great-computer-programming-quotes>

Jan Faigl, 2022    BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze.    39 / 53

Faktoriál    Obrácený výpis    Hanojské věže    Rekurze    Fibonacciho posloupnost

### Elegance vs obtížnost rekurze

*I've often heard people describe understanding recursion as one of those "got it" moments, when the universe opened its secret stores of knowledge and gifted the mind of a burgeoning developer with a very powerful tool. For me, recursion has always been hard. Each time I'm able to peer more into its murky depths, I am humbled to see how little I feel like I really appreciate and understand its power and elegance.*

Rick Winfrey, 2012

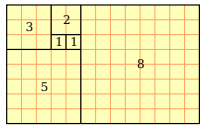
<http://selfless-singleton.rickwinfrey.com/2012/11/27/to-iterate-is-human-to-recurse-divine>

Jan Faigl, 2022    BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze.    40 / 53

Faktoriál    Obrácený výpis    Hanojské věže    Rekurze    Fibonacciho posloupnost

### Fibonacciho posloupnost

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...  
*Nebo 0, 1, 1, 2, 3, 5, ...*
- $F_n = F_{n-1} + F_{n-2}$
- pro  $F_1 = 1, F_2 = 1$   
*Nebo  $F_1 = 0, F_2 = 1$*



- Nekonečná posloupnost přirozených čísel, kde každé číslo je součtem dvou předchozích.
- Limita poměru dvou následujících čísel Fibonacciho posloupnosti je rovna **zlatému řezu**.
  - Section aurea – ideální poměr mezi různými délkami.
  - Rozdělení úsečky na dvě části tak, že poměr větší části ku menší je stejný jako poměr celé úsečky k větší části  $\varphi = \frac{1+\sqrt{5}}{2} \approx 1,618\ 033\ 988\ 749\ 894\ 848\ \dots$

Jan Faigl, 2022    BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze.    42 / 53

Faktoriál    Obrácený výpis    Hanojské věže    Rekurze    Fibonacciho posloupnost

### Fibonacciho posloupnost – historie

- Indičtí matematici (450 nebo 200 BC).
- Leonardo Pisano (1175–1250) popis růstu populace králiků.  
*Italský matematik známý také jako Fibonacci.*
  - $F_n$  – velikost populace po  $n$  měsících za následujících předpokladů.
    - První měsíc se narodí jediný pár.
    - Narozené páry jsou produktivní od 2. měsíce svého života.
    - Každý měsíc zplodí každý produktivní pár jeden další pár.
    - Králíci nikdy neumírají, nejsou nemocní atd.
- Henry E. Dudeney (1857–1930) – popis populace krav.
  - „Jestliže každá kráva vyprodukuje své první tele (jalovici) za rok a poté každý rok jednu další jalovici, kolik budete mít krav za 12 let, jestliže žádná nezemře a na počátku budete mít jednu krávu?“  
*Po 12 let je k dispozici jeden či více byků.*

Jan Faigl, 2022    BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze.    43 / 53

Faktoriál    Obrácený výpis    Hanojské věže    Rekurze    Fibonacciho posloupnost

### Fibonacciho posloupnost – rekurzivně

- Platí:
 
$$f_0 = 1$$

$$f_1 = 1$$

$$f_n = f_{n-1} + f_{n-2}, \text{ pro } n > 1$$

```

1 int fibonacci(int n) {
2     return n < 2
3         ? 1
4         : fibonacci(n - 1) + fibonacci(n - 2);
5 }

```

**Zápis je elegantní, jak je však takový výpočet efektivní?**

Jan Faigl, 2022    BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurze.    44 / 53

Faktorál Obrácený výpis Hanojské věže Rekurse Fibonacciho posloupnost

### Fibonacciho posloupnost – příklad 1/2

- Počet operací při výpočtu Fibonacciho čísla  $n$ .

```

1 long counter; // store number of individual operations
2
3 long fibonacciRecursive(int n) {
4     counter += 1; // jedno porovnání
5     return n < 2 ? 1 : fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2);
6 }
7
8 long fibonacciIterative(int n) {
9     long fibM2 = 1;
10    long fibM1 = 1;
11    long fib = 1;
12    for (int i = 2; i <= n; ++i) {
13        fibM2 = fibM1;
14        fibM1 = fib;
15        fib = fibM1 + fibM2;
16        counter += 3; // dvě přiřazení, jeden součet
17    }
18    return fib;
19 }

```

lec07/demo-fibonacci.c

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurse. 45 / 53

Faktorál Obrácený výpis Hanojské věže Rekurse Fibonacciho posloupnost

### Fibonacciho posloupnost – rekurzivně 2/2

```

1 int main(int argc, char *argv[])
2 {
3     int n = argc > 1 ? atoi(argv[1]) : 25;
4     counter = 0; // reset counter
5     long fibR = fibonacciRecursive(n);
6     long counterR = counter;
7
8     counter = 0; // reset counter
9     long fibI = fibonacciIterative(n);
10    long counterI = counter;
11
12    printf("Fibonacci number recursive: %li\n", fibR);
13    printf("Fibonacci number iteration: %li\n", fibI);
14    printf("Counter recursive: %li\n", counterR);
15    printf("Counter iteration: %li\n", counterI);
16
17    return 0;
18 }

```

lec07/demo-fibonacci.c

```

clang demo-fibonacci.c && ./a.out 30
Fibonacci number recursive: 1346269
Fibonacci number iteration: 1346269
Counter recursive: 2692537
Counter iteration: 87

```

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurse. 46 / 53

Faktorál Obrácený výpis Hanojské věže Rekurse Fibonacciho posloupnost

### Fibonacciho posloupnost – rekurzivně vs iteračně

- Rekurzivní výpočet:
  - Složitost roste exponenciálně s  $n \sim 2^n$ .
- Iterační algoritmus:
  - Počet operací je proporcionální  $n \sim 3n$ .
- Skutečný počet operací závisí na konkrétní implementaci, programovacím jazyku, překladači a hardware.
- Složitost algoritmů proto vyjadřujeme asymptoticky jako funkci velikosti vstupu.
  - Například v tzv. „Big O“ notaci:
    - rekurzivní algoritmus výpočtu má složitost  $O(2^n)$ ;
    - iterační algoritmus výpočtu má složitost  $O(n)$ .

*Efektivní algoritmy mají polynomiální složitost.*

lec07/demo-fibonacci-stats.c, lec07/fibonacci.sh

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurse. 47 / 53

Faktorál Obrácený výpis Hanojské věže Rekurse Fibonacciho posloupnost

### Vsuvka – Vykreslení grafu

- Jednoduchou úpravou vypíšeme počty operací na řádek.

```

printf("%u\t%.3e\t%.3e\n", n,
(double)counterR, (double)counterI);

```

lec07/demo-fibonacci-stats.c

- Program zkompilujeme a spustíme.

```

clang demo-fibonacci-stats.c
./a.out 10 > fibonacci.dat &&
./fibonacci.sh fibonacci.dat fibonacci-n10.pdf

```

fibonacci-n10.pdf

```

./a.out 20 > fibonacci.dat &&
./fibonacci.sh fibonacci.dat fibonacci-n20.pdf

```

fibonacci-n20.pdf

Přeměřovaný standardní výstup do souboru fibonacci.dat představuje tabulku hodnot, která je vykreslena do grafu skriptem fibonacci.sh s využitím nástroje R – <https://www.R-project.org/>.

lec07/demo-fibonacci-stats.c, lec07/fibonacci.sh

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurse. 48 / 53

Faktorál Obrácený výpis Hanojské věže Rekurse Fibonacciho posloupnost

### Vsuvka – Kompilace s optimalizací

- Můžeme rekurzivní výpočet urychlit kompilací s optimalizacemi kódu?

```

clang demo-fibonacci.c && time ./a.out 50
Fibonacci number recursive: 20365011074
Fibonacci number iteration: 20365011074
Counter recursive: 40730022147
Counter iteration: 147

```

```

clang -O2 -march=native demo-fibonacci.c && time ./a.out 50
Fibonacci number recursive: 20365011074
Fibonacci number iteration: 20365011074
Counter recursive: 40730022147
Counter iteration: 147

```

real	1m35.912s	real	1m16.042s
user	1m35.824s	user	1m15.968s
sys	0m0.000s	sys	0m0.008s

- Ano, můžeme. V tomto případě je však zrychlení zanedbatelné.
- V tomto případě je rozhodující asymptotická složitost  $O(2^n)$  vs  $O(n)$ .

*Obecně se pro odladěné a výpočetně náročné programy vyplatí kompilovat s optimalizací. Nárůst výkonu může být i několikanásobný.*

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurse. 49 / 53

Faktorál Obrácený výpis Hanojské věže Rekurse Fibonacciho posloupnost

### Část III

## Část 2 – Zadání 7. domácího úkolu (HW07)

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurse. 50 / 53

Faktorál Obrácený výpis Hanojské věže Rekurse Fibonacciho posloupnost

### Zadání 7. domácího úkolu HW07

**Téma: Hledání textu v souborech**

Povinné zadání: 3b; Volitelné zadání: 3b; Bonusové zadání: není

- Motivace:** Dekomponovat výpočetní úlohu na dílčí výpočetní kroky.
- Cíl:** Osvojit si práci se soubory.
- Zadání:** <https://cw.fel.cvut.cz/wiki/courses/b0b36prp/hw/hw07>
  - Zpracování vstupu po řádcích a detekce textového řetězce ve vstupním souboru.
  - Volitelné zadání** rozšiřuje úlohu o zpracování tří základních kvantifikátorů regulárních výrazů (pouze pro předcházející znak).
    - Znaky pro kvantifikátory: ?, \*, +.
- Termin odevzdání:** 03.12.2022, 23:59:59 PST.

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurse. 51 / 53

Faktorál Obrácený výpis Hanojské věže Rekurse Fibonacciho posloupnost

Diskutovaná témata

## Shrnutí přednášky

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurse. 52 / 53

Faktorál Obrácený výpis Hanojské věže Rekurse Fibonacciho posloupnost

Diskutovaná témata

- Standardní knihovny C
- Čtení a ukládání z/do souboru
- Ošetření chybových stavů - assert(), errno, setjmp(), longjmp()
- Rekurse a rekurzivní řešení problémů
- Kompilace programu s optimalizacemi
- Příště: Spojové struktury.**

Jan Faigl, 2022 BOB36PRP – Přednáška 07: Standardní knihovny C. Rekurse. 53 / 53