

## Standardní knihovny C. Kódovací příklady.

Jan Faigl

Katedra počítačů  
Fakulta elektrotechnická  
České vysoké učení technické v Praze

Přednáška 07

BAB36PRGA – Programování v C

Jan Faigl, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 1 / 41  
Standardní knihovny Práce se soubory Zpracování chyb

### Standardní knihovny

- Jazyk C sám osobě neobsahuje prostředky pro vstup/výstup dat, složitější matematické operace ani:
  - práci z 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`.

*Viz např. ldd a.out.*

- Hlavičkové soubory** – obsahují prototypy funkcí, definice typů, makra a konstanty a vkládají se do zdrojových souborů příkazem preprocesoru `#include <jmeno_knihovny.h>`.

*Např. #include <stdio.h>*

Jan Faigl, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 5 / 41  
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 sekvenčně.
  - 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; *Vytvoří soubor, pokud neexistuje, jinak smaže obsah 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, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 9 / 41

### Přehled témat

- Část 1 – Standardní knihovny, čtení/zápis ze/do souboru  
Standardní knihovny  
Práce se soubory  
Zpracování chyb *S. G. Kochan: kapitola 16, Appendix B*
- Část 2 – Kódovací příklady  
Kódovací příklad – `goto`  
Kódovací příklad – `struct`  
Kódovací příklad – Načítání a ukládání složeného typu `struct`  
Makefile
- Část 3 – Zadání 6. domácího úkolu (HW6)

Jan Faigl, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 2 / 41  
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, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 6 / 41  
Standardní knihovny Práce se soubory Zpracování chyb

### Testování – otevření/zavření souboru

- Testování otevření souboru.

```
1 char *fname = "file.txt";  
  
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, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 10 / 41

Standardní knihovny Práce se soubory Zpracování chyb

## Část I

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

Jan Faigl, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 3 / 41  
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, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 7 / 41  
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, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 11 / 41

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.*

Vrací počet přečtených položek, například pro vstup  
record 1 13.4

příkaz: int r = fscanf(f, "%s %d %lf\n", str, &i, &d);

vrátí v případě úspěšného čtení hodnotu proměnné  
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, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 12 / 41

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, ...);

```
1 int main(int argc, char *argv[])
2 {
3     char *fname = argc > 1 ? argv[1] : "out.txt";
4     FILE *f;
5     if ((f = fopen(fname, "w")) == NULL) {
6         fprintf(stderr, "Error: Open file '%s'\n", fname);
7         return -1;
8     }
9     fprintf(f, "Program arguments argc: %d\n", argc);
10    for (int i = 0; i < argc; ++i) {
11        fprintf(f, "argv[%d]='%s'\n", i, argv[i]);
12    }
13    if (fclose(f) == EOF) {
14        fprintf(stderr, "Error: Close file '%s'\n", fname);
15        return -1;
16    }
17    return 0;
18 }
```

lec07/file\_printf.c

Identicky k stderr lze použít stdout a stdin pro čtení.

Jan Faigl, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 13 / 41

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, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 14 / 41

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, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 15 / 41

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, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 17 / 41

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, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 18 / 41

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 true, program se ukončí a vypíše jméno zdrojového souboru a číslo řádku.
- Makro vloží příslušný kód do programu.  
*Relativně jednoduchý způsob indikace chyby, např. nevhodným argumentem funkce, posloupností volání, ale jako strukturální chyba programu, nikoliv hodnot definovaných za běhu.*
- Vložení makra lze zabránit kompilací s definováním makra NDEBUG.
- Makro assert má význam zejména při ladění program.

```
1 #include <stdio.h>
2 #include <assert.h>
3
4 int main(int argc, char *argv[])
5 {
6     assert(argc > 1);
7     printf("program argc: %d\n", argc);
8     return 0;
9 }
```

lec07/assert.c

- Uvedený příklad pouze demonstruje použití assert().
- Není vhodné testovat proměnné definované za běhu programu.
- Test assert() nebude do programu vložen při kompilaci s NDEBUG.

Jan Faigl, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 19 / 41

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
- 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, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 20 / 41

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 předčasné 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.

```
1 #ifndef __MY_ASSERT_H__
2 #define __MY_ASSERT_H__
3
4 #include <stdio.h> //because of fprintf()
5 #include <stdlib.h> //because of exit() and malloc
6
7 #define my_assert(x, line, file) \
8     if (!(x)) {\
9         fprintf(stderr, "my_assert fail, line: %d, \
10        file %s\n", line, file);\
11        exit(-1);\
12    }
13 #endif
```

lec07/demo-my\_assert.c

lec07/my\_assert.h

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

Jan Faigl, 2024 BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady. 21 / 41

### 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()`.

Kombinaci `setjmp()` a `longjmp()` lze využít pro implementace 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
13 int compute(int x, int y) {
14     if (y == 0) {
15         longjmp(jb, 1);
16     } else {
17         x = (x + y * 2);
18         return (x / y);
19     }
20 }
21 void error_handler(void) {
22     printf("Error\n");
23 }

```

## Část II

### Část 2 – Kódovací příklady

### Kódovací příklad – Ne úplně čitelné použití goto

- Použití `goto` souvisí zejména s čitelností kódu, může jej využít reakci na návratové hodnoty.

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int c = getchar();
6     if (c >= '0' && c <= '9') {
7         goto print_digit;
8     }
9     goto print_error;
10    print_digit:
11    printf("User input '%c' that has ASCII code value '%d\n", c, c);
12    goto leave;
13    print_error:
14    fprintf(stderr, "ERROR: Expected input value is '0'-'9'\n");
15    leave:
16    return 0;
17 }

```

Low Level Learning: why is it illegal to use "goto"?

- Uvedený příklad je funkční, ale načítatelnosti úplně nepřívadlivá.

V našem případě se plně obejdeme bez `goto`, obecně ale může být jeho použití užitečné, např. pokud testujeme postupné volání funkcí.

### Kódovací příklad – struct 1/3

- Implementujeme složený typ s dvěma položkami typu pole znaků `username` a `number`, kde první položku chceme interpretovat jako textový řetězec (`null terminated`), ale ve druhém případě pouze jako pole znaků.

Ukázkový příklad, jehož hlavní motivace je uložení paměti do souboru a náhled na obsah souboru.

```

1 #define USERNAME_SIZE 8
2 #define NUMBER_DIGITS 4
3
4 struct record {
5     char username[USERNAME_SIZE + 1];
6     char number[NUMBER_DIGITS];
7 };
8
9 int main(void) {
10    struct record records[] = {
11        { .username = "user01" },
12        { .username = "user02" },
13        { .username = "admin" },
14        { .username = "root" },
15        { .username = '\0' } // null terminating array
16    };
17
18    fprintf(stderr, "DEBUG: size of struct %lu\n",
19            sizeof(struct record));
20    fprintf(stderr, "DEBUG: size of records %lu\n",
21            sizeof(records));
22 }
23
24 # Položka username je o jeden znak delší, uložení '\0'.
25 # Položka number je zápis čísla o maximální hodnotě 9999 (počet řádů 4) v textové podobě (0000-9999).
26 # Velikost složeného typu je dána velikostí jednotlivých položek.
27 # Pole záznamů inicializujeme se zarážkou (poslední prvek obsahuje prázdný řetězec v položce username).
28 # Na příkladu si ukážeme, jak převést celé číslo na textovou reprezentaci, k čemuž použijeme několik pomocných funkcí.
29 # Implementujeme si funkce pro tisk záznamu a pole záznamů.
30 # Položku number naplníme programově z celého čísla s kontrolou, zdali se číslo vejde do NUMBER_DIGITS.
31 # Složený typ a implementaci funkcí realizujeme v samostatném modulu record.h a record.c.

```

### Kódovací příklad – struct 2/3

```

1 #ifndef __RECORD_H__
2 #define __RECORD_H__
3
4 #define USERNAME_SIZE 8
5 #define NUMBER_DIGITS 4
6
7 struct record {
8     char username[USERNAME_SIZE + 1];
9     char number[NUMBER_DIGITS];
10 };
11
12 void print_record(const struct record * record);
13 void print(const struct record * const records);
14 unsigned int fill_numbers(struct record * const records);
15 #endif
16
17 #include <stdio.h>
18 #include <limits.h>
19 #include <assert.h> // strukturální testy
20 #include "record.h"
21
22 void print_record(const struct record * record)
23 {
24     if (record) {
25         printf("Record\n");
26         printf("%s\n", record->username);
27         printf("%s\n", record->number);
28         printf("Number: %s\n", record->number);
29         printf("%s\n", record->number);
30         printf("Number: %s\n", record->number);
31         printf("Number: %s\n", record->number);
32     }
33 }
34
35 void print(const struct record * const records)
36 {
37     struct record const *cur = records;
38     while (cur && cur->username[0]) {
39         print_record(cur);
40         cur++; // pointer arithmetic
41     }
42 }
43
44 unsigned int fill_numbers(struct record * const records)
45 {
46     struct record const *cur = records;
47     while (cur && cur->username[0]) {
48         print_record(cur);
49         cur++; // pointer arithmetic
50     }
51 }
52
53 # V C nemůžeme přetěžovat jména funkcí, proto máme funkci print_record() a print().
54 # Funkce fill_numbers() vyplní položky numbers v posloupnosti hodnot typu struct record.

```

### Kódovací příklad – struct 3/3

```

1 static unsigned short get_max_number(unsigned int digits)
2 {
3     unsigned int number = 1;
4     assert(sizeof(short) < sizeof(int)); // 16 vs. 32?
5     for (unsigned int i = 0; i < digits; ++i) {
6         number *= 10;
7     }
8     assert(number <= USHRT_MAX); // short
9     return number - 1;
10 }
11
12 static void fill_record_number(unsigned short n, int digits, char *number)
13 {
14     // number needs to be at least digits large
15     for (int i = digits - 1; i >= 0; --i) {
16         number[i] = (n % 10) + '0';
17         n = n / 10;
18     }
19 }
20
21 static void print_chars(size_t digits, const char *number)
22 {
23     for (size_t i = 0; i < digits; ++i) {
24         putchar(number[i]);
25     }
26 }
27
28 unsigned int fill_numbers(struct record * const records)
29 {
30     struct record *cur = records;
31     unsigned short n = 0;
32     const short max_number = get_max_number(NUMBER_DIGITS);
33     while (cur && cur->username[0]) {
34         fill_record_number(n, NUMBER_DIGITS, cur->number);
35         n++;
36         cur++;
37     }
38     return n;
39 }
40
41 # V programu máme snahu testovat velikost paměťové reprezentace.
42 # Lokální pomocné funkce jsou static.
43 # Hodnoty položky number vyplňujeme programově inkrementálně od hodnoty 0000.

```

### Kódovací příklad – Načítání a ukládání struct 1/4

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "record.h"
5
6 int main(void) {
7     int ret = EXIT_SUCCESS;
8
9     struct record records[] = {
10        { .username = "user01" },
11        { .username = "user02" },
12        { .username = "admin" },
13        { .username = "root" },
14        { .username = "jif" },
15        { .username = '\0' } // null terminating array
16    };
17
18    fprintf(stderr, "DEBUG: size of struct %lu\n",
19            sizeof(struct record));
20    fprintf(stderr, "DEBUG: size of records %lu\n",
21            sizeof(records));
22
23    unsigned int n = fill_numbers(records); // number!
24    print(records);
25 }
26
27 const char *fname = "records.dat";
28 FILE *fd = fopen(records, "w"); // uložení records
29 if (fd) {
30     size_t saved = fwrite(records, sizeof(struct record), n, fd);
31     size_t size = n * sizeof(struct record);
32     printf("DEBUG: saved bytes %lu out of %lu\n", saved + sizeof(struct record), size);
33 } else {
34     fprintf(stderr, "ERROR: Cannot open \"%s\" for writing!\n", fname);
35     ret = EXIT_FAILURE;
36     return ret;
37 }
38
39 $ clang -c record.c -o record.o
40 $ clang save_struct.c record.o -o save && ./save
41 DEBUG: size of struct 13
42 DEBUG: size of records 78
43 Record
44 |- username: "user01"
45 |- number: 0000
46 ...
47 Record
48 |- username: "jif"
49 |- number: 0004
50
51 DEBUG: saved bytes 65 out of 65

```

### Kódovací příklad – Načítání a ukládání struct 2/4

```

1 # Obsah uloženého souboru můžeme zkusit přímo otevřít v textovém editoru nebo použijeme program hexdump.
2
3 $ clang -c record.c -o record.o
4 $ clang save_struct.c record.o -o save
5 $ ./save 1>/dev/null
6 DEBUG: size of struct 13
7 DEBUG: size of records 78
8 $ hexdump -C records.dat
9 00000000 75 73 65 72 30 31 00 00 00 30 30 30 30 31 61 64 64 69 6e 00 |user01...0000use|
10 00000010 72 30 32 00 00 00 30 30 30 31 61 64 64 69 6e 00 |r02...0001admin.|
11 00000020 00 00 00 30 30 32 72 6f 6f 74 00 00 00 00 |...0002root.....|
12 00000030 30 30 33 6a 66 00 00 00 00 00 00 30 30 |00003jf.....000|
13 00000040 34
14 00000041
15
16 # V případě, že vytvořenému souboru records.dat odebereme práva zápisu, např. chmod 0 records.dat, program selže.
17
18 $ chmod 0 records.dat
19 $ ./save 1>/dev/null; echo $?
20 DEBUG: size of struct 13
21 DEBUG: size of records 78
22 ERROR: Cannot open file "records.dat" for writing
23

```

### Kódovací příklad – Načítání a ukládání struct 3/4

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "record.h"
5
6 #define NUM_RECORDS 2
7
8 int main(void) {
9     const char *fname = "records.dat";
10
11    FILE *fd = fopen(fname, "r");
12    if (!fd) {
13        perror("Error open file");
14        goto error;
15    }
16
17    struct record *records = malloc(NUM_RECORDS * sizeof(struct record));
18    if (!records) {
19        perror("Error malloc");
20        fclose(fd); // Korektně soubor zavíráme.
21        goto error;
22    }
23
24    $size_t loaded;
25
26    while (loaded = fread(records, sizeof(struct record), NUM_RECORDS, fd)) {
27        printf(stderr, "DEBUG: loaded records %lu\n", loaded);
28        for (size_t i = 0; i < loaded; ++i) {
29            print_record(&records[i]);
30        }
31        free(records);
32        fclose(fd);
33        goto leave;
34    }
35    error:
36    fprintf(stderr, "ERROR: during reading from the file \"%s\"!\n", fname);
37    return 1;
38    leave:
39    return 0;
40 }
41
42 $ clang load_struct.c record.o -o load && ./load
43 DEBUG: loaded records 2
44 ...
45 Record
46 |- username: "root"
47 |- number: 0003
48
49 DEBUG: loaded records 1
50 Record
51 |- username: "jif"
52 |- number: 0004
53
54 # Načtení bloku dat funkcí fread().

```

Kódovací příklad – goto      Kódovací příklad – struct      Kódovací příklad – Načítání a ukládání složeného typu struct      Makefile

### Kódovací příklad – Načítání a ukládání struct 4/4 (lépe)

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "record.h"
5 #define NUM_RECORDS 2
6
7 enum { ERROR_FILE = 101, ERROR_MEM = 102};
8
9 void print_error(int error);
10
11 int main(void) {
12     int ret = EXIT_SUCCESS;
13     const char *fname = "records.dat";
14
15     FILE *fd = fopen(fname, "r");
16     if (!fd && (ret = ERROR_FILE)) { // ret!
17         goto leave;
18     }
19     struct record *records = malloc(
20         NUM_RECORDS * sizeof(struct record));
21     if (!records && (ret = ERROR_MEM)) { // ret!
22         goto leave;
23     }
24     ssize_t loaded;
25
26     while ((loaded = fread(records, sizeof(struct record), NUM_RECORDS, fd)) > 0) {
27         fprintf(stderr, "DEBUG: loaded records %ju\n", loaded);
28         for (size_t i = 0; i < loaded; ++i) {
29             print_record(records[i]);
30         }
31         free(records);
32     }
33 leave:
34     if (fd) {
35         fclose(fd);
36     }
37     print_error(ret);
38     return ret;
39 }
40
41 void print_error(int error)
42 {
43     switch (error) {
44     case ERROR_FILE:
45         fprintf(stderr, "ERROR: open file\n");
46         break;
47     case ERROR_MEM:
48         fprintf(stderr, "ERROR: mem allocation\n");
49         break;
50     }
51 }

```

Jan Faigl, 2024      BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady.      34 / 41

### Pravidla překladačů v gmake / make

- Pro řízení překladačů použijeme pravidlový předpis programu GNU make. `make` nebo `gmake`
- Pravidla se zapisují do souboru `Makefile`. <http://www.gnu.org/software/make/make.html>
- Pravidla jsou deklarativní ve tvaru definice cíle, závislostí cíle a akce, která se má provést.
 

cíl : závislosti	dvojtečka
akce	tabulátor
- Cíl (podobně jako závislosti) může být například symbolické jméno nebo jméno souboru.
 

```
tload.o : tload.c
clang -c tload.c -o tload.o
```
- Předpis může být napsán velmi jednoduše.
 

*Například jako v uvedené ukázce.*

Flexibilita použití však spočívám především v použití zavedených proměnných, vnitřních proměnných a využití vzorů, neboť většina zdrojových souborů se překládá identicky.

Jan Faigl, 2024      BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady.      36 / 41

### Příklad – Makefile

- Definujeme pravidlo pro vytvoření souborů `.o` z `.c` z aktuálních souborů v pracovním adresáři s koncovkou `.c`.
 

```

CC=ccache $(CC)
CFLAGS+=-O2

OBJ=$(patsubst %.c,%.o,$(wildcard *.c))

TARGET=tload

bin: $(TARGET)

$(OBJ): %.o: %.c
$(CC) -c $(CFLAGS) $(CPPFLAGS) -o $@

$(TARGET): $(OBJ)
$(CC) $(OBJ) $(LDFLAGS) -o $@

clean:
$(RM) $(OBJ) $(TARGET)
            
```

`CC=clang make` vs `CC=gcc make`
- Při linkování záleží na pořadí souborů (knihoven)!**
- Jednou z výhod dobrých pravidel je možnost paralelního překladačů nezávislých cílů.

Jan Faigl, 2024      BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady.      37 / 41

## Část III

### Část 3 – Zadání 6. domácího úkolu (HW6)

Jan Faigl, 2024      BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady.      38 / 41

### Zadání 6. domácího úkolu HW6

**Téma: Maticové počty**

Povinné zadání: **4b**; Volitelné zadání: **4b**; Bonusové zadání: **5b**

- Motivace:** Získání zkušenosti s dvojrozměrným polem.
- Cíl:** Osvojit si práci s polem variabilní délky a předávání ukazatelů.
- Zadání:** <https://cw.fel.cvut.cz/wiki/courses/bab36prga/hw/hw6>
  - Načtení vstupních hodnot dvou matic a znaku operace (`*` – násobení).
  - Volitelné zadání** rozšiřuje úlohu o další operace s maticemi sčítání (`+`) a odčítání (`-`), které mohou být zapsány ve výrazu.
  - Bonusové zadání** pak řeší zpracování celého výrazu, ve kterém jsou však jednotlivé matice uvedeny jako symboly, které jsou nejdříve definovány načtením hodnot matic ze standardního vstupu.

Využití `struct` a dynamické alokace může být výhodnou, není však nutné.
- Termín odevzdání:** 20.04.2024, 23:59:59 PDT.
- Bonusová úloha:** 24.05.2024, 23:59:59 CEST.

Jan Faigl, 2024      BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady.      39 / 41

Diskutovaná témata

## Shrnutí přednášky

Jan Faigl, 2024      BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady.      40 / 41

### Diskutovaná témata

- Standardní knihovny C
- Čtení a ukládání z/do souborů
- Ošetření chybových stavů - `assert()`, `errno`, `setjmp()`, `longjmp()`
- Kódovací příklady

**Příště: Spojové struktury.**

Jan Faigl, 2024      BAB36PRGA – Přednáška 07: Standardní knihovny C. Kódovací příklady.      41 / 41