

Binární soubory

Jan Faigl

Katedra počítačů

Fakulta elektrotechnická

České vysoké učení technické v Praze

Přednáška 13

B0B36PRP – Procedurální programování



Přehled témat

- Část 1 – Binární soubory

 - Práce se soubory (standardní knihovna C)

 - Blokové čtení a zápis

 - Kódovací příklad – Načítání a ukládání složeného typu struct

- Část 2 – Příklady binárních souborů (obrázky)

 - Obrázky

 - Knihovna SDL

 - Načtení obrázku RGB

 - Uložení obrázku ve formátu JPEG

 - Načtení obrázku (dat) RGBA



Část I

Část 1 – Binární soubory



Obsah

Práce se soubory (standardní knihovna C)

Blokové čtení a zápis

Kódovací příklad – Načítání a ukládání složeného typu struct



Základní práce se soubory

- Knihovna `stdio.h` – `fopen()`, `fclose()`, `feof()`, `getc()`/`fgetc()`, `putc()`/`fputc()`, `fprintf()`, `fscanf()`, `fread()`, `fwrite()`.

`fseek()`, `rewind()`

- Přístup k souboru je prostřednictvím ukazatele `FILE*`.
- Otevření souboru `FILE *fopen(char *filename, char *mode);`.

Výchozí režim je textový – zápis/čtení '`\n`' dle OS.

- `"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.

- Binární soubory (*modifikátor* `"b"`).
- Soubory jsou čteny/zapisovány sekvenčně.

- Se soubory se pracuje jako s proudem dat — postupné načítání/zápis.
- Aktuální „pozici“ v souboru si můžeme představit jako kurzor.
- Při otevření souboru se kurzor nastavuje na začátek souboru.



Příklad - Zápis textového souboru

- Tisk chyby otevření souboru funkcí `perror()`.
- Návratová hodnota volání `fprintf()`.
- Rozdíl mezi OS.

<https://dev-cpp.com/Dev-Cpp-5/>

```

1  #include <stdio.h>
2  #include <stdlib.h>
4  int main(int argc, char *argv[])
5  {
6      char *fname = argc > 1 ? argv[1] : "text.txt";
7      FILE *f = fopen(fname, "w");
8      if (!f) perror("fopen");
9      int r = -100;
10     f && (r = fprintf(f, "Line1\nLine2\n"));
11     fprintf(stderr, "r: %d\n", r);
12     f && fclose(f);
13     return f ? EXIT_SUCCESS : EXIT_FAILURE;
14 }

```

lec13/create_txt.c

```

$ clang create_txt.c && ./a.out ; echo $?
r: 12
0
$ chmod a-w text.txt
$ ./a.out text.txt; echo $?
fopen: Permission denied
r: -100
1
$ wine ./create_txt.exe text-win.txt
r: 12
$ ls -ln text*.txt | tr -s ' ' | cut -d' ' -f5,9 -
14 text-win.txt
12 text.tx
$ xxd text.txt > text.hex
$ xxd text-win.txt text-win.hex
$ diff text.hex text-win.hex
1c1
< 00000000: 4c69 6e65 310a 4c69 6e65 320a
   Line1.Line2.
---
> 00000000: 4c69 6e65 310d 0a4c 696e 6532 0d0a
   Line1..Line2.

```



Příklad čtení ze souboru (textově a binárně)

- Čtení txt vs. bin souboru podle názvu spustitelného programu.
- Rozdíl mezi OS.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main(int argc, char *argv[])
6 {
7     _Bool bin = strstr(argv[0], "bin");
8     char *fname = argc > 1 ? argv[1] : "text.txt";
9
10    FILE *f = fopen(fname, bin ? "rb" : "r");
11    if (!f) perror("fopen");
12    size_t count = 0;
13    int r;
14    while (f && (r = getc(f)) != EOF) {
15        if (r == '\r')
16            fprintf(stderr, "DEBUG: carriage return character '\\r\\n");
17        if (r == '\n')
18            fprintf(stderr, "DEBUG: New line character '\\n\\n");
19        count += 1;
20    }
21    fprintf(stderr, "INFO: No. of read characters %lu\\n", count);
22    f && fclose(f);
23    return f ? EXIT_SUCCESS : EXIT_FAILURE;
24 }
```

lec13/read_file.c

Testujeme výskyt `bin` v názvu programu.

<https://www.winehq.org/>

```

./a.out text-win.txt
DEBUG: carriage return character '\r'
DEBUG: New line character '\n'
DEBUG: carriage return character '\r'
DEBUG: New line character '\n'
INFO: No. of read characters 14
```

```

$ wine read_file.exe
DEBUG: New line character '\n'
DEBUG: New line character '\n'
INFO: No. of read characters 12
$ wine read_file.exe text-win.txt
DEBUG: New line character '\n'
DEBUG: New line character '\n'
INFO: No. of read characters 12
$ cp read_file.exe read_file-bin.exe
$ wine read_file-bin.exe text-win.txt
DEBUG: carriage return character '\r'
DEBUG: New line character '\n'
DEBUG: carriage return character '\r'
DEBUG: New line character '\n'
INFO: No. of read characters 14
```



Příklad testu dosažení konce souboru 1/4

- Načtení posloupnosti dvojice celý čísel, každá dvojice na samostatném řádku.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[])
5 {
6     char *fname = argc > 1 ? argv[1] : "values.txt";
7     FILE *f = fopen(fname, "r");
8     if (!f) perror("fopen");
9     size_t count = 0;
10    int a, b;
11    int r = 0;
12
13    while (f && (r = fscanf(f, "%d %d\n", &a, &b) == 2)) {
14        count += 1;
15    }
16    fprintf(stderr, "INFO: No. of values pairs read: %lu\n",
17            count);
18    fprintf(stderr, "DEBUG: r value is %d\n", r);
19    f && fclose(f);
20    return f && r == EOF ? EXIT_SUCCESS : EXIT_FAILURE;
21 }
```

lec13/read_values.c

```
$ clang read_values.c -o read_values
$ cat values.txt
1 2
3 4
4 1
$ ./read_values values.txt
INFO: No. of values pairs read: 3
DEBUG: r value is 0
$ ./read_values values-noeol.txt
INFO: No. of values pairs read: 3
DEBUG: r value is 0
$ cat values-e1.txt
1 2
3 4
4
$ ./read_values values-e1.txt
INFO: No. of values pairs read: 2
DEBUG: r value is 0
$ cat values-e2.txt
1 2
4 a
4 1
$ ./read_values values-e2.txt
INFO: No. of values pairs read: 1
DEBUG: r value is 0
```



Příklad testu dosažení konce souboru 2/4

- Netestujeme návratovou hodnotu `fscanf()`.

```
1 #include <stdio.h>
2 #include <stdlib.h>
4 int main(int argc, char *argv[])
5 {
6     char *fname = argc > 1 ? argv[1] : "values.txt";
7     FILE *f = fopen(fname, "r");
8     if (!f) perror("fopen");
9     size_t count = 0;
10    int a, b;
11    int r = 0;
13    while (f && (r = fscanf(f, "%d %d\n", &a, &b))) {
14        count += 1;
15    }
16    fprintf(stderr, "INFO: No. of values pairs read: %lu\n",
17        count);
17    fprintf(stderr, "DEBUG: r value is %d\n", r);
18    f && fclose(f);
19    return f && r == EOF ? EXIT_SUCCESS : EXIT_FAILURE;
20 }
```

lec13/read_values-f2.c

```
$ clang read_values-f1.c -o read_values-f1
$ ./read_values-f1 values.txt
^C % Program neskonci
$ ./read_values-f1 values-noeol.txt
^C % Program neskonci
$ ./read_values-f1 values-e1.txt
^C % Program neskonci
$ ./read_values-f1 values-e2.txt
INFO: No. of values pairs read: 2
DEBUG: r value is 0
```



Příklad testu dosažení konce souboru 3/4

- Přidáním dosažení konce souboru `feof()` nepostačuje.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[])
5 {
6     char *fname = argc > 1 ? argv[1] : "values.txt";
7     FILE *f = fopen(fname, "r");
8     if (!f) perror("fopen");
9     size_t count = 0;
10    int a, b;
11    int r = 0;
12
13    while (f && !feof(f) && (r = fscanf(f, "%d %d", &a, &b))) {
14        count += 1;
15    }
16    fprintf(stderr, "INFO: No. of values pairs read: %lu\n",
17             count);
18    fprintf(stderr, "DEBUG: r value is %d\n", r);
19    f && fclose(f);
20    return f && r == EOF ? EXIT_SUCCESS : EXIT_FAILURE;
21 }
```

lec13/read_values-f2.c

```
$ clang read_values-f2.c -o read_values-f2
$ cat values.txt
1 2
3 4
4 1
$ ./read_values-f2 values.txt; echo $?
INFO: No. of values pairs read: 4
DEBUG: r value is -1
0
$ ./read_values-f2 values-noeol.txt; echo $?
INFO: No. of values pairs read: 3
DEBUG: r value is 2
1
$ ./read_values-f2 values-e1.txt; echo $?
INFO: No. of values pairs read: 3
DEBUG: r value is 1
1
$ ./read_values-f2 values-e2.txt; echo $?
INFO: No. of values pairs read: 2
DEBUG: r value is 0
1
```



Příklad testu dosažení konce souboru 4/4

- Dále ještě přidáme čtení konce řádku.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[])
5 {
6     char *fname = argc > 1 ? argv[1] : "values.txt";
7     FILE *f = fopen(fname, "r");
8     if (!f) perror("fopen");
9     size_t count = 0;
10    int a, b;
11    int r = 0;
12
13    while (f && !feof(f) && (r = fscanf(f, "%d %d\n", &a, &b)))
14    {
15        count += 1;
16    }
17    fprintf(stderr, "INFO: No. of values pairs read: %lu\n",
18        count);
19    fprintf(stderr, "DEBUG: r value is %d\n", r);
20    f && fclose(f);
21    return f && r == EOF ? EXIT_SUCCESS : EXIT_FAILURE;
22 }
```

lec13/read_values-f3.c

```
$ clang read_values-f3.c -o read_values-f3
$ ./read_values-f3 values.txt; echo $?
INFO: No. of values pairs read: 3
DEBUG: r value is 2
1
$ ./read_values-f3 values-noeol.txt; echo $?
INFO: No. of values pairs read: 3
DEBUG: r value is 2
1
$ ./read_values-f3 values-e1.txt; echo $?
INFO: No. of values pairs read: 3
DEBUG: r value is 1
1
$ ./read_values-f3 values-e2.txt; echo $?
INFO: No. of values pairs read: 2
DEBUG: r value is 0
1
```



Obsah

Práce se soubory (standardní knihovna C)

Blokové čtení a zápis

Kódovací příklad – Načítání a ukládání složeného typu struct



Binární (blokové) čtení/zápis z/do souboru

- Otevření souboru s příznakem `"b"`.
- Čtení bloku dat funkcí `fread()` a zápis bloku funkcí `fwrite()`.
- 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.



Kódovací příklad – struct 1/3

- Implementujme 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
16         terminating array
17     };
18     fprintf(stderr, "DEBUG: size of struct
19         %lu\n", sizeof(struct record));
20     fprintf(stderr, "DEBUG: size of records
21         %lu\n", sizeof(records));
```

- Položka `username` je o jeden znak delší, uložení `'\0'`.
- Položka `number` je zápis čísla o maximální hodnotě 9999 (počet řádů 4) v textové podobě (0000–9999).
- Velikost složeného typu je dána velikostí jednotlivých položek.
- Pole záznamů inicializujeme se zarážkou (poslední prvek obsahuje prázdný řetězec v položce `username`).

\$ clang struct.c && ./a.out

DEBUG: size of struct 13

DEBUG: size of records 6

- Na příkladu si ukážeme, jak převést celé číslo na textovou reprezentaci, k čemuž použijeme několik pomocných funkcí.
- Implementujeme si funkce pro tisk záznamu a pole záznamů.
- Položku `number` naplníme programově z celého čísla s kontrolou, zdali se číslo vejde do `NUMBER_DIGITS`.
- 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__
4 #define USERNAME_SIZE 8
5 #define NUMBER_DIGITS 4
7 struct record {
8     char username[USERNAME_SIZE + 1];
9     char number[NUMBER_DIGITS];
10 };
12 void print_record(const struct record * record);
13 void print(const struct record * const records);
15 unsigned int fill_numbers(struct record * const
    records);
17 #endif
    lec13/record.h
```

- V C nemůžeme přetěžovat jména funkcí, proto máme funkci `print_record()` a `print()`.
- Funkce `fill_numbers()` vyplní položky `numbers` v posloupnosti hodnot typu `struct record`.

```
1 #include <stdio.h>
2 #include <limits.h>
3 #include <assert.h> // strukturální testy
5 #include "record.h"
    lec13/record.c

33 void print_record(const struct record * record)
34 {
35     if (record) {
36         printf("Record\n");
37         printf("|- username: \"%s\"\n", record->username);
38         printf("|- number: ");
39         print_chars(NUMBER_DIGITS, record->number);
40         printf("\n\n");
41         // printf("|- number: %s\n\n"); // Vyzkoušejte!
42     }
43 }

45 void print(const struct record * const records)
46 {
47     struct record const *cur = records;
48     while (cur && cur->username[0]) {
49         print_record(cur);
50         cur += 1; // pointer arithmetic
51     }
52 }
    lec13/record.c
```



Kódovací příklad – struct 3/3

```
7 static unsigned short get_max_number(unsigned int digits)
8 {
9     unsigned int number = 1;
10    assert(sizeof(short) < sizeof(int)); // 16 vs. 32?
11    for (unsigned int i = 0; i < digits; ++i) {
12        number *= 10;
13    }
14    assert(number <= USHRT_MAX); // short
15    return number - 1;
16 }
17
18 static void fill_record_number(unsigned short n, int
19                               digits, char *number)
20 { //number needs to be at least digits large
21     for (int i = digits - 1; i >= 0; --i) {
22         number[i] = (n % 10) + '0';
23         n = n / 10;
24     }
25 }
```

lec13/record.c

- V programu máme snahu testovat velikost paměťové reprezentace.

```
26 static void print_chars(size_t digits, const char *number)
27 { // number je ukazatel na konstantní hodnotu
28     for (size_t i = 0; i < digits; ++i, ++number) {
29         putchar(*number);
30     }
31 }
```

lec13/record.c

```
54 unsigned int fill_numbers(struct record * const records)
55 {
56     struct record *cur = records;
57     unsigned short n = 0;
58     const short max_number = get_max_number(NUMBER_DIGITS);
59     while (cur && cur->username[0]) {
60         assert(n <= max_number); // Vyplňujeme programově
61         fill_record_number(n, NUMBER_DIGITS, cur->number);
62         n += 1;
63         cur += 1;
64     }
65     return n;
66 }
```

lec13/record.c

- Lokální pomocné funkce jsou **static**.
- Hodnoty položky **number** vyplňujeme programově inkrementálně od hodnoty 0000.



Obsah

Práce se soubory (standardní knihovna C)

Blokové čtení a zápis

Kódovací příklad – Načítání a ukládání složeného typu struct



Kódovací příklad – Načítání a ukládání struct 1/4

```

1  #include <stdio.h>
2  #include <stdlib.h>
4  #include "record.h"
6  int main(void) {
7      int ret = EXIT_SUCCESS;
9      struct record records[] = {
10         { .username = "user01" },
11         { .username = "user02" },
12         { .username = "admin" },
13         { .username = "root" },
14         { .username = "jf" },
15         { .username = '\0' } // null terminating array
16     };
18     fprintf(stderr, "DEBUG: size of struct %lu\n", sizeof(struct
        record));
19     fprintf(stderr, "DEBUG: size of records %lu\n", sizeof(records
        ));
21     unsigned int n = fill_numbers(records); // number!
22     print(records);

```

```

24     const char *fname = "records.dat";
25     FILE *fd = fopen("records.dat", "w"); // uložení records
26     if (fd) {
27         size_t saved = fwrite(records, sizeof(struct record), n, fd);
28         size_t size = n * sizeof(struct record);
29         printf("DEBUG: saved bytes %lu out of %lu\n", saved * sizeof(struct record
        ), size);
30     } else {
31         fprintf(stderr, "ERROR: Cannot open \"%s\" for writting\n", fname);
32         ret = EXIT_FAILURE;
33     }
34     return ret;
35 };

```

lec13/save_struct.c

```

$ clang -c record.c -o record.o
$ clang save_struct.c record.o -o save && ./save
DEBUG: size of struct 13
DEBUG: size of records 78
Record
|- username: "user01"
|- number: 0000
...
Record
|- username: "jf"
|- number: 0004
DEBUG: saved bytes 65 out of 65

```



Kódovací příklad – Načítání a ukládání struct 2/4

- Obsah uloženého souboru můžeme zkusit přímo otevřít v textovém editoru nebo použijeme program `hexdump`.

```
$ clang -c record.c -o record.o
$ clang save_struct.c record.o -o save
$ ./save 1>/dev/null
DEBUG: size of struct 13
DEBUG: size of records 78
$ hexdump -C records.dat
00000000 75 73 65 72 30 31 00 00 00 30 30 30 75 73 65 |user01...0000use|
00000010 72 30 32 00 00 00 30 30 30 31 61 64 6d 69 6e 00 |r02...0001admin.|
00000020 00 00 00 30 30 30 32 72 6f 6f 74 00 00 00 00 00 |...0002root.....|
00000030 30 30 30 33 6a 66 00 00 00 00 00 00 30 30 30 |0003jf.....000|
00000040 34                                     |4|
00000041
```

- V případě, že vytvořenému souboru `records.dat` odebereme práva zápisu, např. `chmod 0 records.dat`, program selže.

```
$ chmod 0 records.dat
$ ./save 1> /dev/null; echo $?
DEBUG: size of struct 13
DEBUG: size of records 78
ERROR: Cannot open file "records.dat" for writing
1
```



Kódovací příklad – Načítání a ukládání struct 3/4

```
1 #include <stdio.h>
2 #include <stdlib.h>
4 #include "record.h"
6 #define NUM_RECORDS 2
8 int main(void) {
9     const char *fname = "records.dat";
11    FILE *fd = fopen(fname, "r");
12    if (!fd) {
13        perror("Error open file");
14        goto error;
15    }
16    struct record *records = malloc(NUM_RECORDS * sizeof(
17        struct record));
18    if (!records) {
19        perror("Error malloc");
20        fclose(fd); // Korektně soubor zavíráme.
21        goto error;
22    }
23    ssize_t loaded;
```

lec13/load_struct.c

- Načtení bloku dat funkcí `fread()`.

```
23 while ((loaded = fread(records, sizeof(struct record), NUM_RECORDS, fd))) {
24     fprintf(stderr, "DEBUG: loaded records %lu\n", loaded);
25     for (size_t i = 0; i < loaded; ++i) {
26         print_record(&records[i]);
27     }
28 }
29 free(records);
30 fclose(fd);
31 goto leave;
32 error:
33     fprintf(stderr, "ERROR: during reading from the file \"%s\"\n", fname);
34     return 1;
35 leave:
36     return 0;
37 }
```

lec13/save_struct.c

```
$ clang load_struct.c record.o -o load && ./load
```

```
DEBUG: loaded records 2
```

```
...
```

```
Record
```

```
|- username: "root"
```

```
|- number: 0003
```

```
DEBUG: loaded records 1
```

```
Record
```

```
|- username: "jf"
```

```
|- number: 0004
```



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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "record.h"
4 #define NUM_RECORDS 2
5 enum { ERROR_FILE = 101, ERROR_MEM = 102};
6 void print_error(int error);
7 int main(void) {
8     int ret = EXIT_SUCCESS;
9     const char *fname = "records.dat";
10    FILE *fd = fopen(fname, "r");
11    if (!fd && (ret = ERROR_FILE)) { // ret!
12        goto leave;
13    }
14    struct record *records = malloc(
15        NUM_RECORDS * sizeof(struct record));
16    if (!records && (ret = ERROR_MEM)) { // ret!
17        goto leave;
18    }
19    ssize_t loaded;
```

```
25     while ((loaded = fread(records, sizeof(struct record), NUM_RECORDS, fd)) {
26         fprintf(stderr, "DEBUG: loaded records %lu\n", loaded);
27         for (size_t i = 0; i < loaded; ++i) {
28             print_record(&records[i]);
29         }
30     }
31     free(records);
32 leave:
33     if (fd) {
34         fclose(fd);
35     }
36     print_error(ret);
37     return ret;
38 }
39 void print_error(int error)
40 {
41     switch (error) {
42     case ERROR_FILE:
43         fprintf(stderr, "ERROR: open file\n");
44         break;
45     case ERROR_MEM:
46         fprintf(stderr, "ERROR: mem allocation\n");
47         break;
48     }
49 }
50 }
51 }
```



Část II

Část 2 – Příklady binárních souborů (obrázky)



Obsah

Obrázky

Knihovna SDL

Načtení obrázku RGB

Uložení obrázku ve formátu JPEG

Načtení obrázku (dat) RGBA



Paměťová reprezentace obrázků

- Obrázek je reprezentován jako matice pixelů, souvislý blok paměti (1D pole).
- Každý pixel může být reprezentován jedním nebo více bajty, v závislosti na typu a kódování obrázku (šedotonový - *grayscale*, 8-bitový s paletou, RGB nebo také s průhledností RGBA).
- Formát obrázku může být neztrátový, např. BMP, PNG, PPM, nebo ztrátový, např. JPEG.
- Formát obrázku definuje jak jsou data uložena na disku.
- V paměti je pro zobrazení obrázek reprezentován souvislým blokem o velikosti $\text{šířka} \times \text{výška} \times \text{počet bajtů na pixel}$.
- Pro načítání a ukládání obrázky typicky využijeme existující knihovnu třetí strany.
- Podobně pro vizualizaci využijeme nějakou dostupnou knihovnu.



Textová reprezentace obrázku PPM P6

- Formát PPM P6 je textově binární formát obsahující textovou hlavičku následovanou souvislým blokem bajtů reprezentující jednotlivé pixel po složkách RGB.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #ifndef WIDTH
5  #define WIDTH 640
6  #endif
7
8  #ifndef HEIGHT
9  #define HEIGHT 480
10 #endif
11
12 #define BYTES_PER_PIXEL 3
13 #define MAX_COLOR_VALUE 255
14
15 char *create_image(int w, int h);
16
17 int save_ppm_p6(char *img, int w, int h, char *fname); // return 1 on
    success
18
19 int main(int argc, char *argv[])
20 {
21     char *fname = argc > 1 ? argv[1] : "image.ppm";
22     char *img = create_image(WIDTH, HEIGHT);
23
24     int ok = img && ( // HEIGHT <=> WIDTH if three args are given
25         argc > 2 ?
26             save_ppm_p6(img, HEIGHT, WIDTH, fname) :
27             save_ppm_p6(img, WIDTH, HEIGHT, fname)
28         );
29     if (img)
30         free(img);
31     return ok ? EXIT_SUCCESS : EXIT_FAILURE;
32 }

```

```

56 char *create_image(int w, int h)
57 {
58     char *img = malloc(w * h * BYTES_PER_PIXEL * sizeof *img);
59     unsigned char color[BYTES_PER_PIXEL] = {0, 0, 0}; // RGB
60     for (int y = 0; img && y < h; ++y) { // all rows
61         color[0] = (color[0] + 1) % 256; // alternate R
62         for (int x = 0; x < w; ++x) // all columns
63             for (int c = 0; c < BYTES_PER_PIXEL; ++c)
64                 img[(y * w + x) * BYTES_PER_PIXEL + c] = color[c];
65     }
66     return img;
67 }
68
69 int save_ppm_p6(char *img, int w, int h, char *fname)
70 {
71     if (!img) return 0;
72     FILE *fd = fopen(fname, "wb");
73     if (!fd) {
74         perror("Error save_ppm fopen()");
75         return 0;
76     }
77     fprintf(fd, "P6\n %s\n %d\n %d\n %d\n", "Test image", w, h,
78         MAX_COLOR_VALUE);
79     int ret = fwrite(img, BYTES_PER_PIXEL, w * h, fd) == w * h;
80     fclose(fd);
81     return ret;
82 }

```

lec13/img/write_ppm_image.c



Uložení obrázku ve formátu PPM P6

```
$ clang write_ppm_image.c -o ppm
$ ./ppm
$ head -n 5 image.ppm
P6
Test image
640
480
255
$ du -h image.ppm
960K image.pp
$ xview image.ppm
```



```
$ ./ppm image2.ppm rot
$ head -n 5 image2.ppm
P6
Test image
480
640
255
$ ls -ln image*.ppm | tr -s ' ' | cut -d' ' -f5,9 -
921630 image.ppm
921630 image2.ppm
$ xview image2.ppm
```



Obsah

Obrázky

Knihovna SDL

Načtení obrázku RGB

Uložení obrázku ve formátu JPEG

Načtení obrázku (dat) RGBA



Simple DirectMedia Layer – SDL

- SDL je multiplatformní multimediální C/C++ knihovna s nízkoúrovňovým přístupem ke grafice (2D a 3D), zvuku, klávesnici, atd.

<https://www.libsdl.org/>

- Používá se pro tvorbu her a multimediálních aplikací, např. Angry Birds, Unreal Tournament (Linux OpenGL), OpenTTD.

- SDL, SDL2 (<https://wiki.libsdl.org/SDL2/Tutorials>, SDL3.

https://mrlvsb.github.io/upr-skripta/c/aplikovane_ulohy/sdl.html

- V PRP pro zjednodušení použijeme knihovnu `xwin_sdl.h` s rozhráním pro:

- vytvoření grafického okna: `int xwin_init(int w, int h);`
- zavření okna: `void xwin_close(void);`
- překreslení okna: `void xwin_redraw(int w, int h, unsigned char *img);`
- načtení RGB obrázku: `unsigned char *xwin_load_image(const char *filename, int *width, int *height);`
- a další funkce jako je
 - načtení RGBA obrázku `unsigned char *xwin_load_image_bytes(const char *filename, size_t *size, int *Rshift, int *Gshift, int *Bshift, int *Ashift);`
 - vyprázdnění fronty událostí `void xwin_poll_events(void);`



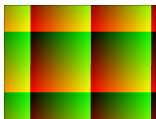
Příklad animace

- Animace je opakované překreslování modifikovaného obrázku (matice).

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "xwin_sdl.h"
5
6  #define WIDTH 640
7  #define HEIGHT 480
8
9  #define ANIM 100
10 #define DELAY_MS 10
11
12 unsigned char* create_image(int w, int h);
13 void update_image(unsigned char *img, int w, int h);
14
15 int main(int argc, char *argv[])
16 {
17     unsigned char *img = create_image(WIDTH, HEIGHT);
18     if (img && !xwin_init(WIDTH, HEIGHT)) {
19         for (int i = 0; i < ANIM; ++i) {
20             update_image(img, WIDTH, HEIGHT);
21             xwin_redraw(WIDTH, HEIGHT, img);
22             delay(DELAY_MS);
23         }
24         xwin_close();
25     }
26
27     if (img) free(img);
28     return EXIT_SUCCESS;
29 }

```



```

38 unsigned char* create_image(int w, int h)
39 {
40     unsigned char *img = malloc(3 * w * h * sizeof *img); // 3 bytes
41     per pixel
42     unsigned char *cur = img;
43     for (int y = 0; y < h; ++y)
44         for (int x = 0; x < w; ++x) {
45             *(cur++) = 255; // red
46             *(cur++) = 0;
47             *(cur++) = 0;
48         }
49     return img;
50 }
51
52 void update_image(unsigned char *img, int w, int h)
53 {
54     static int i = 0;
55     for (int y = 0; y < h; ++y) {
56         for (int x = 0; x < w; ++x) {
57             const int idx = y * WIDTH * 3 + x * 3; // pixel position
58             *(img + idx + 0) = (y + i) % 255; // R
59             *(img + idx + 1) = (x + i) % 255; // G
60             *(img + idx + 2) = 0; // B
61         }
62         i += 1;
63     }
64 }

```

lec13/img/xwin_anim.c



Sestavení programu (xwin_sdl)

- `xwin_sdl` volá funkce SDL2.
- SDL2 nastavení kompilátoru lze získat z `sdl2-config`, `CFLAGS` a `LDFLAGS`.

```
sudo apt install libsdl2-dev
```

- Načítání obrázků je v `xwin_sdl` realizováno voláním `SDL2_image`. `libsdl2-image-dev`

```
CFLAGS+=$(shell sdl2-config --cflags)
LDFLAGS+=$(shell sdl2-config --libs)
LDFLAGS+=-lSDL2_image
```

[lec13/img/Makefile](#)

```
$ make
clang -c xwin_anim.c -O2 -std=gnu99 -pedantic -Wall -I/usr/local/include -I/usr/local/include
    /SDL2 -D_REENTRANT -D_THREAD_SAFE -o xwin_anim.o
clang -c xwin_sdl.c -O2 -std=gnu99 -pedantic -Wall -I/usr/local/include -I/usr/local/include/
    SDL2 -D_REENTRANT -D_THREAD_SAFE -o xwin_sdl.o
clang xwin_anim.o xwin_sdl.o -L/usr/local/lib -lSDL2 -lSDL2_image -o xwin_anim
```



Obsah

Obrázky

Knihovna SDL

Načtení obrázku RGB

Uložení obrázku ve formátu JPEG

Načtení obrázku (dat) RGBA



Načtení obrázku `xwin_load_image()`

- Dle formátu obrázku nemusí posloupnost bajtů pixelů odpovídat RGB, ale například BGR (BMP).

```

86 unsigned char *xwin_load_image(const char *filename, int *width, int *height)
87 {
88     unsigned char *ret = NULL;
89     SDL_Surface *img = IMG_Load(filename);
90     if (img) {
91         *width = img->w;
92         *height = img->h;
93         const int bytesPerPixel = img->format->BytesPerPixel;
94 #ifndef NDEBUG
95         fprintf(stderr, "DEBUG(xwin_sdl): bytesPerPixel: %d\n", bytesPerPixel);
96 #endif
97         require(bytesPerPixel == 3);
98         ret = (unsigned char *)malloc(img->w * img->h * 3 *
99             sizeof(unsigned char));
100         require(ret != NULL);
101         unsigned char *dst = ret;
102         for (int y = 0; y < img->h; ++y) {
103             for (int x = 0; x < img->w; ++x) {
104                 const int idx = (y * img->w + x) * img->format->BytesPerPixel;
105                 Uint8 *px = (Uint8*)img->pixels + idx;
106                 *(dst++) = *(px + img->format->Rshift / 8);
107                 *(dst++) = *(px + img->format->Gshift / 8);
108                 *(dst++) = *(px + img->format->Bshift / 8);
109             }
110         }
111         SDL_FreeSurface(img);
112     }
113     return ret;
114 }
```

```

16 void require_fail(const char *expr, const char *fname,
17     int line, const char *fname);
18 #define require(cond) \
19     do { \
20         if (!(cond)) {\
21             require_fail(#cond, __FILE__, __LINE__,
22                 __func__); \
23         } \
24     } while(0)
25 // - function -----
26 void require_fail(const char *expr, const char *fname,
27     int line, const char *fname)
28 {
29     fprintf(stderr, "ERROR: require failed: %s @ %s in %s
30         :%d\n", expr, fname, fname, line);
31     exit(EINVAL);
32 }
```



Příklad načtení obrázku view()

```

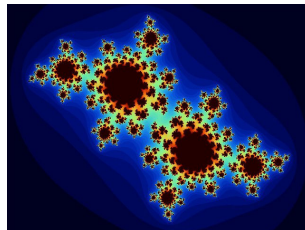
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "xwin_sdl.h"
5
6  #define BYTES_PER_PIXEL 3
7  #define VIEW_WAIT_MS 5000 // 5 sec
8
9  int main(int argc, char *argv[])
10 {
11     int ret = EXIT_SUCCESS;
12     char *fname = argc > 1 ? argv[1] : "image.png";
13     int w, h;
14     unsigned char *img = xwin_load_image(fname, &w, &h);
15     if (!img) {
16         fprintf(stderr, "ERROR: Cannot load image %s\n", fname);
17         ret = EXIT_FAILURE;
18     } else if (!xwin_init(w, h)) {
19         xwin_redraw(w, h, img);
20         fprintf(stderr, "INFO: View loaded image %s for %d ms\n", fname,
21             VIEW_WAIT_MS);
22         delay(VIEW_WAIT_MS);
23         xwin_close();
24     } else {
25         fprintf(stderr, "ERROR: Cannot init window for the image %s of the
26             size %d x %d\n", fname, w, h);
27         ret = EXIT_FAILURE;
28     }
29 }

```

```

$ make
clang -c xwin_sdl.c -O2 -std=gnu99 -pedantic -Wall -I/usr/
local/include -I/usr/local/include/SDL2 -D_REENTRANT -
D_THREAD_SAFE -o xwin_sdl.o
clang xwin_sdl.o xwin_anim.o -L/usr/local/lib -lSDL2 -
lSDL2_image -o xwin_anim
clang -c view.c -O2 -std=gnu99 -pedantic -Wall -I/usr/local/
include -I/usr/local/include/SDL2 -D_REENTRANT -
D_THREAD_SAFE -o view.o
clang xwin_sdl.o view.o -L/usr/local/lib -lSDL2 -lSDL2_image -
o view
$ ./view julia.png
DEBUG(xwin_sdl): bytesPerPixel: 3
INFO: View loaded image julia.png for 5000 ms

```



Příklad animace obrázku

- V programu `view.c` přidáme funkci `move()`, která postupně přesune obrázek za pravý okraj.

```

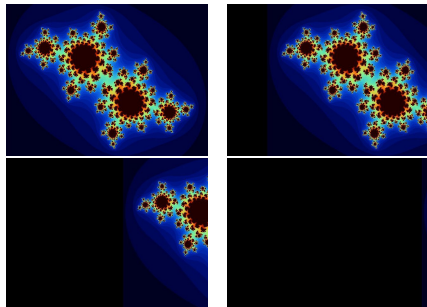
1 void move(unsigned char *img, int w, int h);
2 void h_shift(unsigned char *img, int w, int h);
3 ...
4     delay(VIEW_WAIT_MS);
5     if (move_to_right)
6         move(img, w, h);
7 ...
38 void move(unsigned char *img, int w, int h)
39 {
40     for (int i = 0; i < w; ++i) {
41         h_shift(img, w, h);
42         xwin_redraw(w, h, img);
43     }
44 }
46 void h_shift(unsigned char *img, int w, int h)
47 { // shift image by one pixel from left to right
48     for (int y = 0; y < h; ++y) {
49         for (int x = w-1; x > 0; --x) {
50             unsigned char *src = &img[(y * w + x - 1)*BYTES_PER_PIXEL];
51             unsigned char *dst = &img[(y * w + x)*BYTES_PER_PIXEL];
52             for (int i = 0; i < 3; ++i) {
53                 *(dst + i) = *(src + i);
54             }
55         }
56         for (int i = 0; i < 3; ++i) {
57             img[(y * w) * BYTES_PER_PIXEL + i] = 0; // black
58         }
59     }
60 }

```

```

$ ./view julia.png move
DEBUG(xwin_sdl): bytesPerPixel: 3
INFO: View loaded image julia.png for 5000 ms

```



Obsah

Obrázky

Knihovna SDL

Načtení obrázku RGB

Uložení obrázku ve formátu JPEG

Načtení obrázku (dat) RGBA



Uložení obrázku v JPEG

- Obrázek je posloupnost bajtů (1D pole) o délce odpovídající rozlišení a počtu bajtů na pixel.
- Formát JPEG je ztrátová komprese, obrázek tak odpovídá souboru o menší velikosti než původní (nekomprimovaný) obrázek.
- Komprese do JPEG formátu, je vytvoření jiného (menšího) 1D pole.
- Práci s JPEG obrázky je možné založit například na knihovně jpeg-turbo ([libjpeg.so.8](https://libjpeg-turbo.org/)).

```
28 void save_image_jpeg(int w, int h, unsigned char *img, char *fname)
29 {
30     unsigned long jsize = 0;
31     unsigned char *jbuf = NULL;
32     jpeg_compress(img, w * h, w, h, &jbuf, &jsize, 85, 3);
33     FILE *fd = fopen(fname, "wb");
34     require(fwrite(jbuf, 1, jsize, fd) == jsize);
35     fclose(fd);
36     free(jbuf);
37 }
```

lec13/img/save_jpeg.c



Příklad uložení obrázku v JPEG formátu

- Formát souboru (obrázku) není závislý na jeho jméně, ale obsahu.

```
1 #include <stdio.h>
2 #include <stdlib.h>
4 #include "xwin_sdl.h"
5 #include "save_jpeg.h"
7 int main(int argc, char *argv[])
8 {
9     int ret = EXIT_FAILURE;
10    char *fname = argc > 1 ? argv[1] : "image.png";
11    char *fname_jpeg = argc > 2 ? argv[2] : NULL;
12    int w, h;
13    unsigned char *img = xwin_load_image(fname, &w, &h);
14    if (!img) {
15        fprintf(stderr, "ERROR: Cannot load image %s\n", fname);
16    } else if (!fname_jpeg) {
17        fprintf(stderr, "ERROR: JPEG filename must be given\n");
18    } else {
19        save_image_jpeg(w, h, img, fname_jpeg);
20        ret = EXIT_SUCCESS;
21    }
22    free(img);
23    return ret;
24 }
```

```
$ make
clang jpeg.o save_jpeg.o xwin_sdl.o -L/usr/local/lib -
    lSDL2 -lSDL2_image -ljpeg -o jpeg
$ ./jpeg
ERROR: Cannot load image image.png
$ ./jpeg julia.png
DEBUG(xwin_sdl): bytesPerPixel: 3
ERROR: JPEG filename must be given
$ ./jpeg julia.png julia.jpeg
DEBUG(xwin_sdl): bytesPerPixel: 3
$ du -h julia.png julia.jpeg
344K julia.png
72K julia.jpeg
$ ./jpeg julia.png j
DEBUG(xwin_sdl): bytesPerPixel: 3
$ sha1 julia.jpeg j
SHA1 (julia.jpeg) = 8425
    d3f03deee275b9398ba9cde36dd68057c8b6
SHA1 (j) = 8425d3f03deee275b9398ba9cde36dd68057c8b6
$ xview j
```



Obsah

Obrázky

Knihovna SDL

Načtení obrázku RGB

Uložení obrázku ve formátu JPEG

Načtení obrázku (dat) RGBA



Načtení obrázku RGBA – `xwin_load_image_bytes()`

- V případě obrázku s alfa kanálem (transparentnost) odpovídá jeden pixel 32-bitové hodnotě.
- Data z načteného obrázku můžeme dekodovat do definované posloupnosti RGBA.

```
110 unsigned char *xwin_load_image_bytes(const char *filename, size_t *size, int *Rshift, int *Gshift, int *Bshift, int *Ashift)
111 {
112     unsigned char *ret = NULL;
113     SDL_Surface *img = IMG_Load(filename);
114     if (img) {
115         #ifndef NDEBUG
116             fprintf(stderr, "DEBUG(xwin_sdl): %s (%s:%d) image size %d x %d with BytesPerPixel: %d\n", __func__, __FILE__, __LINE__,
117                     img->w, img->h, img->format->BytesPerPixel);
118         #endif
119         require(img->format->BytesPerPixel == 4);
120         *size = ((size_t)img->w * (size_t)img->h * (size_t)img->format->BytesPerPixel);
121         ret = malloc(*size * sizeof(unsigned char));
122         require(ret != NULL);
123         memcpy(ret, img->pixels, *size * sizeof(unsigned char));
124         *Rshift = img->format->Rshift;
125         *Gshift = img->format->Gshift;
126         *Bshift = img->format->Bshift;
127         *Ashift = img->format->Ashift;
128         SDL_FreeSurface(img);
129     }
130     return ret;
131 }
```



Dekódování dat z RGBA obrázku

- V obrázku můžeme uložit libovolná data jako posloupnost bajtů.
- Obsah `input.txt` uložíme jako posloupnost bajtů na pozici RGB, ale obrázek uložíme jako RGBA ve formátu PNG, který následně dekódujeme načtením funkcí `xwin_load_image_bytes()` s uvažováním pouze RGB hodnot.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "xwin_sdl.h"
5
6 int main(int argc, char *argv[])
7 {
8     int ret = EXIT_SUCCESS;
9     char *fname = argc > 1 ? argv[1] : "input.png";
10    size_t size;
11    int Rshift, Gshift, Bshift, Ashift;
12    unsigned char *img = xwin_load_image_bytes(fname, &size, &
        Rshift, &Gshift, &Bshift, &Ashift);
13    if (!img) {
14        fprintf(stderr, "ERROR: Cannot load image %s\n", fname);
15        ret = EXIT_FAILURE;
16    } else {
17        int shift[3] = { Rshift/8, Gshift/8, Bshift/8 };
18        for (size_t i = 0; i < size; i += 4) {
19            for (int k = 0; k < 3; ++k) {
20                if (!img[i + shift[k]]) break; // 0 is end of txt
21                putchar(img[i + shift[k]]);
22            }
23        }
24    }
25    free(img);
26    return ret;
27 }

```

```

$ make
clang -c decode.c -O2 -std=gnu99 -pedantic -Wall -I/usr/
    local/include -I/usr/local/include/SDL2 -D_REENTRANT -
    D_THREAD_SAFE -o decode.o
clang decode.o xwin_sdl.o -L/usr/local/lib -lSDL2 -
    lSDL2_image -o decode

$ ./decode input.png > output.txt
$ diff input.txt output.txt
$ sha1 input.txt output.txt
SHA1 (input.txt) = e5e6f2d5ac2db91e18aa9d2ba8570a5695fdc2f7
SHA1 (output.txt) = e5e6f2d5ac2db91e18aa9d2ba8570a5695fdc2f7
$ du -h input.txt output.txt
4.0K  input.txt
4.0K  output.txt
$ head -c 20 output.txt
Lorem ipsum dolor s

```

[lec13/img/decode.c](#)



Respektování RGBA vs. BGRA

- Pokud bychom nezohlednili vnitřní kódování obrázku, nezískáme původní data v případě použití jiného formátu obrázku.
- Podobně dekódování nebude fungovat při ztrátové kompresi obrazových dat.*

```
$ magick convert input.png input.bmp
$ du -h input.png input.bmp
4.0K  input.png
28K  input.bmp

$ ./decode input.png > output-png.txt
DEBUG(xwin_sdl): xwin_load_image_bytes (xwin_sdl.c:116)
    image size 82 x 82 with BytesPerPixel: 4
$ ./decode input.bmp > output-bmp.txt
DEBUG(xwin_sdl): xwin_load_image_bytes (xwin_sdl.c:116)
    image size 82 x 82 with BytesPerPixel: 4
$ diff output-png.txt output-bmp.txt
$ sha1 input.txt output-png.txt output-bmp.txt
SHA1 (input.txt) = e5e6f2d5ac2db91e18aa9d2ba8570a5695fdc2f7
SHA1 (output-png.txt) =
e5e6f2d5ac2db91e18aa9d2ba8570a5695fdc2f7
SHA1 (output-bmp.txt) =
e5e6f2d5ac2db91e18aa9d2ba8570a5695fdc2f7
```

lec13/img/decode.c

```
12 ...
13     unsigned char *img = xwin_load_image_bytes(fname, &size, ... );
14     if (!img) {
15         ...
16     } else {
17         for (size_t i = 0; i < size; i += 4) {
18             for (int k = 0; k < 3; ++k) {
19                 if (!img[i + k]) break; // 0 is end of txt
20                 putchar(img[i + k]);
21             }
22         }
23     ...

$ ./decode2 input.png 2>/dev/null > output2-png.txt
$ ./decode2 input.bmp 2>/dev/null > output2-bmp.txt
$ sha1 input.txt output2-png.txt output2-bmp.txt
SHA1 (input.txt) = e5e6f2d5ac2db91e18aa9d2ba8570a5695fdc2f7
SHA1 (output2-png.txt) =
e5e6f2d5ac2db91e18aa9d2ba8570a5695fdc2f7
SHA1 (output2-bmp.txt) = 6
f092912345ff64ab1eefbfbfe803639e6eeb5a8

$ head -c 10 output2-png.txt
Lorem ipsu
$ head -c 10 output2-bmp.txt
roL mespi
```

lec13/img/decode2.c



Shrnutí přednášky



Diskutovaná témata

- Binární soubory.
- Příklad načítání a ukládání obrázků.

