

Pole, ukazatel, textový řetězec, vstup a výstup programu

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 04

BAB36PRGA – Programování v C

Přehled témat

- Část 1 – Pole, ukazatele a řetězce
 - Pole
 - Ukazatele
 - Funkce a předávání parametrů
 - Vstup a výstup programu
 - Ukazatele a pole
 - Textové řetězce
- Část 2 – Zadání 3. domácího úkolu (HW3)

S. G. Kochan: kapitoly 7, 10, 11

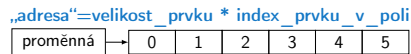
Část I

Pole a ukazatele

Pole

- Datová struktura pro uložení **více hodnot stejného typu**.
- Slouží k reprezentaci posloupnosti hodnot v paměti. *Hodnoty uloženy v souvislém bloku paměti.*
- Jednotlivé prvky mají identickou velikost a jejich relativní adresa vůči počátku pole je jednoznačně určena.
 - Prvky můžeme adresovat pořadím prvku v poli.

Relativní „adresa“ vůči prvnímu prvku.



- Proměnná typu pole reprezentuje adresu vyhrazeného paměťového prostoru, kde jsou hodnoty uloženy.
*Adresa_prvku = adresa_prvniho_prvku + velikost_typu * index_prvku_v_poli*
- Definicí proměnné dochází k alokaci paměti pro uložení definovaného počtu hodnot příslušného typu.
- **Velikost pole statické délky nelze měnit.**
Garance souvislého přístupu k položkám pole.

Definice pole

- Hodnota proměnné typu pole je odkaz (adresa) na místo v paměti, kde je pole uloženo.
- Definice proměnné typu pole se skládá z typu prvků, jména proměnné a hranatých závorek [].

typ proměnná [];

proměnná_typu_pole [index_prvku_pole]

- Závorky [] slouží také k přístupu (adresaci) prvku.
- Příklad definice proměnné typu pole hodnot typu int.
Alokace paměti pro až 10 prvků pole.

```
int array[10];
```

Tj. $10 \times \text{sizeof}(\text{int})$

```
printf("Size of array %lu\n", sizeof(array));  
printf("Item %i of the array is %i\n", 4, array[4]);
```

```
Size of array 40  
Item 4 of the array is -5728
```

Hodnoty pole nejsou inicializovány!

Pole (array)

- Pole je posloupnost prvků **stejného typu**.
- K prvkům pole se přistupuje pořadovým číslem prvku.
- **Index prvního prvku je vždy roven 0.**
- Prvky pole mohou být proměnné libovolného typu. *Též strukturované typy, viz další přednáška.*
- Pole může být jednorozměrné nebo vícerozměrné. *Pole poli (...) prvků stejného typu.*
- Prvky pole určuje: **jméno, typ, počet prvků**.
- **Prvky pole tvoří v paměti souvislou oblast!**
- Velikost pole (v bajtech) je dána počtem prvků pole n a **typem prvku**, tj. $n * \text{sizeof}(\text{typ})$.
- Textový řetězec je pole typu **char**, kde poslední prvek je `'\0'`.

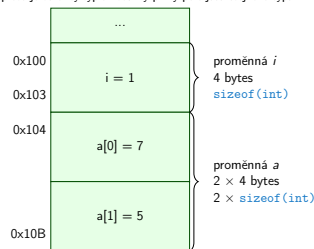
C nekontroluje za běhu programu, zdali je index platný!

Např. přístup do pole `a[1000]`.

Pole – Příklad vizualizace alokace přiřazení hodnot

- Proměnná typu pole označuje za **začátek paměti**, kde jsou alokovány jednotlivé prvky pole.
- Přístup k prvkům pole je prostřednictvím indexového operátoru [], který určí adresu prvku.
Jako začátek paměti + číslo prvku × paměťová velikost prvku, proto je důležitý typ a všechny prvky pole jsou stejného typu.

```
1 int i;  
2 int a[2];  
3  
4 i = 1;  
5  
6 a[1] = 5;  
7 a[0] = 7;
```



Pro účely vizualizace začíná alokace proměnných na adrese 0x100. Automatické proměnné na zásobníku jsou však správně alokovány od horní adresy k adresám nížším.

Pole – Příklad 1/3

- Definice jednorozměrného a **dvourozměrného** pole.
/ jednorozmerne pole prvku typu char */
char simple_array[10];
/* dvourozmerne pole prvku typu int */
int two_dimensional_array[2][2];*
- Přístup k prvkům pole `m[1][2] = 2*1`;
- Příklad definice pole a tisk hodnot prvků

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

```
3 int main(void)  
4 {  
5     int array[5];  
  
7     printf("Size of array: %lu\n", sizeof(array));  
8     for (int i = 0; i < 5; ++i) {  
9         printf("Item[%i] = %i\n", i, array[i]);  
10    }  
11    return 0;  
12 }
```

```
Size of array: 20  
Item[0] = 1  
Item[1] = 0  
Item[2] = 740314624  
Item[3] = 0  
Item[4] = 0
```

lec04/array.c

Pole – Příklad 2/3 – Definice pole

```
1 #include <stdio.h>  
2 array[0]=0 array[2][0]= 0  
3 array[1]=1 array[2][1]= 1  
4 {  
5     int array[10];  
6 array[4]=4 array[2][4]= -20  
7 array[5]=5 array[2][5]= -35  
8 array[6]=6 array[2][6]= -54  
9 array[7]=7 array[2][7]= -77  
10 array[8]=8 array[2][8]= -104  
11 array[9]=9 array[2][9]= -135  
  
14 int n = 5;  
15 int array2[n * 2];  
  
18 printf("Size of array: %lu\n", sizeof(array));  
19 for (int i = 0; i < 10; ++i) {  
20     printf("array[%i]=%+2i \t array2[%i]=%6i\n", i, array[i], i, array2[i]);  
21 }  
22 return 0;  
23 }
```

lec04/demo-array.c

Pole – Příklad 3/3 – Definice pole s inicializací

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int array[5] = {0, 1, 2, 3, 4};
6
7     printf("Size of array: %lu\n", sizeof(array));
8     for (int i = 0; i < 5; ++i) {
9         printf("Item[%i] = %i\n", i, array[i]);
10    }
11    return 0;
12 }

```

Size of array: 20
Item[0] = 0
Item[1] = 1
Item[2] = 2
Item[3] = 3
Item[4] = 4

lec04/array-init.c

- Inicializace pole

```

double d[] = { 0.1, 0.4, 0.5 }; // inicializace pole hodnotami
char str[] = "hallo"; // inicializace pole textovým literálem
char s[] = { 'h', 'a', 'l', 'l', 'o', '\0' }; //inicializace prvků
int m[3][3] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
char cmd[][10] = { "start", "stop", "pause" };

```

Pole variabilní délky (VLA – Variable Length Array)

- C99 umožňuje definovat tzv. pole variabilní délky – délka pole je určena za běhu programu.
V předchozích verzích bylo nutné znát délku při kompilaci.
- Délka pole tak může být, např. argument funkce.

```

1 void fce(int n);
2 int main(int argc, char *argv[])
3 {
4     fce(argc);
5     return 0;
6 }
7
8 void fce(int n)
9 {
10    // int local_array[n] = { 1, 2 }; inicializace není dovolena
11    int local_array[n]; // variable length array
12
13    printf("sizeof(local_array) = %lu\n", sizeof(local_array));
14    printf("length of array = %lu\n", sizeof(local_array) / sizeof(int));
15    for (int i = 0; i < n; ++i) {
16        local_array[i] = i + 1;
17    }
18 }

```

lec04/fce_var_array.c

- Pole variabilní délky však nelze v definici inicializovat.

Pole ve funkci a jako argument funkce

- Lokálně definované pole ve funkci má rozsah platnosti pouze v rámci funkce (bloku).

```

1 void fce(int n)
2 {
3     int array[n];
4     // počítání s array
5     {
6         int array2[n*2];
7     } // po skončení bloku array2 automaticky zaniká
8     // zde již není array2 přístupné
9 } // po skončení funkce, pole array automaticky zaniká

```

- Pole je automaticky vytvořeno a po skončení bloku (funkce) automaticky zaniká.
- Lokální proměnné jsou ukládány na tzv. zásobník, který má relativně malou velikost (jednotky/desítky MB). Pro velká pole je vhodnější alokovat paměť dynamicky a použít ukazatele.
Více o pamětových třídách a dynamické alokaci v 5. přednášce.
- Pole může být argumentem funkce

```

void fce(int array[]);

```

předávaná hodnota je adresa začátku pole – hodnota **ukazatele!**

Ukazatel (pointer)

- Ukazatel (pointer) je proměnná jejíž hodnota je adresa paměti jiné proměnné.
- Pointer *odkazuje* na jinou proměnnou.

Odkazuje na oblast paměti, kde je uložena hodnota proměnné

- Ukazatel má typ proměnné, na kterou může ukazovat.

Důležité pro ukazatelovou aritmetiku

- Ukazatel na hodnoty (proměnné) základních typů: `char, int, ...`
- „Ukazatel na pole“; *ukazatel na funkci, ukazatel na ukazatel*

- Ukazatel může být též bez typu (`void`).
- Velikost proměnné nelze z vlastností ukazatele určit.
- Pak může obsahovat adresu libovolné proměnné.

- Prázdná adresa ukazatele je definovaná hodnotou konstanty `NULL`.

Textová konstanta (makro) preprocesoru definovaná jako „null pointer constant“.

C99 – lze též použít „int“ hodnotu 0.

C za běhu programu nekontroluje platnost adresy (hodnoty) ukazatele.

Ukazatele umožňují psát efektivní kódy, při neobezřetném používání mohou vést k chybám. Proto je důležité osvojit si princip nepřímého adresování a pochopit organizaci a přístup do paměti.

Referenční a dereferenční operátor

- Referenční operátor – `&`
- Vrací adresu paměti, kde je uložena hodnota proměnné, před kterou je uveden.
- Dereferenční operátor – `*`
- Vrací **l-hodnotu** (l-value) odpovídající hodnotě na adrese ukazatele.

***proměnná typu ukazatel**

- Umožňuje číst a zapisovat hodnotu na adrese dané obsahem ukazatele, např. ukazatel na hodnotu typu `int` (tj. `int *`).

```

*p = 10; // zápis hodnoty 10 na adresu uloženou v proměnné p
int a = *p; // čtení hodnoty z adresy uložené v p

```

- Pro tisk hodnoty ukazatele (adresy) lze ve funkci `printf()` použít řídicí řetězec `“%p”`.

```

1 int a = 10;
2 int *p = &a;
3
4 printf("Value of a %i, address of a %p\n", a, &a);
5 printf("Value of p %p, address of p %p\n", p, &p);

```

Proměnné typu ukazatel (pointer) – příklady

```

1 int i = 10; /* i -- promenna typu int
2           &i -- adresa promenne i */
3
4 int *pi; /* definice promenne typu pointer
5         pi -- pointer na promennou typu int
6         *pi -- promenna typu int */
7
8 pi = &i; /* do pi se ulozi adresa promenne i */
9
10 int b; /* promenna typu int */
11
12 b = *pi; /* do promenne b se ulozi obsah adresy
13         ulozene v ukazeteli pi */

```

Ukazatele – Příklad vizualizace alokace přiřazení hodnot

```

1 char c;
2
3 c = 10;
4
5 char *pc;
6
7 pc = &c;
8
9 int i = 17;
10 int *pi = &i;
11
12 *pi = 15;
13 *pc = 2;
14
15 int **ppi = &pi;

```

Pro účely vizualizace začíná alokace proměnných na adrese 0x100. Automatické proměnné na zásobníku jsou však zpravidla alokovány od horní adresy k adresám nížším.

0x100	...	proměnná c 1 byte sizeof(char)
0x101	c = 2	proměnná pc 64-bit sizeof(char*)
0x108	pc = 0x100	proměnná i 4 bytes sizeof(int)
0x109	i = 15	proměnná pi 64-bit sizeof(int*)
0x10D	pi = 0x109	proměnná ppi 64-bit sizeof(int**)
0x114	ppi = 0x10D	
0x115		
0x11C		

- Ukazatele jsou proměnné, které uchovávají adresy jiných proměnných.

Ukazatel (pointer) – 2. příklad

```

1 printf("i: %d -- pi: %p\n", i, pi); // 10 0x7fffffff8fc
2 printf("&i: %p -- *pi: %d\n", &i, *pi); // 0x7fffffff8fc 10
3 printf("*(&i): %d -- &(*pi): %p\n", *(&i), &(*pi));
4
5 printf("i: %d -- *pj: %d\n", i, *pj); // 10 10
6 i = 20;
7 printf("i: %d -- *pj: %d\n", i, *pj); // 20 20
8
9 printf("sizeof(i): %lu\n", sizeof(i)); // 4
10 printf("sizeof(pi): %lu\n", sizeof(pi)); // 8
11
12 long l = (long)pi;
13 printf("0x%lx %p\n", l, pi); /* print l as hex -- %lx */
14 // 0x7fffffff8fc 0x7fffffff8fc
15
16 l = 10;
17 pi = (int*)l; /* possible but it is nonsense */
18 printf("l: 0x%lx %p\n", l, pi); // 0xa 0xa

```

lec04/pointers.c

Ukazatele (pointery), proměnné a jejich hodnoty

- Proměnné jsou názvy adres, kde jsou uloženy hodnoty příslušného typu.
- Kompilátor pracuje přímo s adresami.
V případě kompilace se zpravidla jedná o adresy relativní, které jsou absolutizovány při linkování nebo spouštění programu.
- Ukazatel (pointer) je proměnná, ve které je uložena adresa. Na této adrese se pak nachází hodnota nějakého typu (např. `int`).
- Ukazatele realizují tzv. **nepřímé adresování** (**indirect addressing**).
- Dereferenční operátor `*` přistupuje na proměnnou adresovanou hodnotou ukazatele.
 - Hodnota je získána z adresy, ve které je uložena v paměti, na kterou odkazuje hodnota proměnné typu ukazatel.
- Operátor `&` vrací adresu, kde je uložena hodnota proměnné.

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Ukazatele (pointery) a kódovací styl

- Typ ukazatel se značí symbolem `*`.
- `*` můžeme zapisovat u jména typu nebo jména proměnné.
- Preferujeme zápis u proměnné, abychom předešli omylům.
`char* a, b, c;` `char *a, *b, *c;`
Pointer je pouze a *Všechny tři proměnné jsou ukazatele.*
- Zápis typu ukazatele na ukazatel `char **a;`.
- Zápis pouze typu (bez proměnné): `char*` nebo `char**`.
- Ukazatel na proměnnou prázdného typu zapisujeme jako `void *ptr`.
- Prokazatelně neplatná adresa má symbolické jméno `NULL`.
Definovaná jako makro preprocesoru (C99 lze použít 0).
- Proměnné v C nejsou automaticky inicializovány a ukazatele tak mohou odkazovat na neplatnou paměť, proto může být vhodné explicitně inicializovat ukazatele na `0` nebo `NULL`.
*Např. int *i = NULL;*

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 21 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Funkce main a její tvary

- Základní tvar funkce `main`
`int main(int argc, char *argv[]) { ... }`
- Alternativně pak také
`int main(int argc, char **argv) { ... }`
- Argumenty funkce nejsou nutné
`int main(void) { ... }`
- Rozšířená funkce o nastavení proměnných prostředí
Pro Unix a MS Windows
`int main(int argc, char **argv, char **envp) { ... }`
Přístup k proměnným prostředí funkci `getenv()` z knihovny `<stdlib.h>`. `lec04/main_env.c`
- Rozšířená funkce o specifické parametry OS X
`int main(int argc, char **argv, char **envp, char **apple);`

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 25 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Interakce programu s uživatelem

- Funkce `int main(int argc, char *argv[])`
 - Při spuštění programu lze předat parametry (textové řetězce).
 - Při ukončení programu lze předat návratovou hodnotu.
Konvence 0 bez chyby, ostatní hodnoty chybový kód.
 - Při běhu programu lze číst ze standardního vstupu a zapisovat na standardní výstup.
Např. `scanf()` nebo `printf()`
 - Při spuštění programu lze vstup i výstup přeměrovat z/do souboru.
Program tak neceká na vstup uživatele (stisk klávesy „Enter“).
`./program <stdin.txt >stdout.txt 2>stderr.txt`
- Každý program (terminálový) má standardní vstup (`stdin`) a výstup (`stdout`) a dále pak standardní chybový výstup (`stderr`), které lze v shellu přeměrovat.
- Alternativou k `scanf()` a `printf()` lze využít `fscanf()` a `fprintf()`.
 - Funkce mají první argument soubor jinak, je syntax identická.
 - Soubory/proudy `stdin`, `stdout` a `stderr` jsou definovány v `<stdio.h>`.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 29 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Funkce a předávání parametrů

- V C jsou parametry funkce předávány hodnotou.
- Parametry jsou lokální proměnné funkce (alokované na zásobníku), které jsou inicializovány na hodnotu předávanou funkci.
Více o volání funkcí a paměti v 5. přednášce.

```
void fce(int a, char *b)
{ /*
a - je lokální proměna typu int (uložena na zásobníku)
b - je lokální proměna typu ukazatel na proměnou
typu char (hodnota je adresa a je take na zásobníku)*/
}
```

- Lokální změna hodnoty proměnné neovlivňuje hodnotu proměnné vně funkce.
- Při předání ukazatele, však máme přístup na adresu původní proměnné, kterou můžeme měnit.
- Ukazatelem v podstatě realizujeme „volání odkazem.“

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 23 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Argumenty funkce main

- Základní tvar funkce `main`
`int main(int argc, char *argv[]) { ... }`
- `argc` – obsahuje počet argumentů programu.
Včetně jména spouštěného programu.
 - Argumenty jsou textové řetězce oddělené mezerou (bílým znakem).
- `argv` – pole ukazatelů na hodnoty typu `char`.
Typ „čteme“ zprava doleva.
 - Pole `argv` má velikost (počet prvků) daný hodnotou `argc`.
 - Každý prvek pole `argv[i]` obsahuje adresu, kde je uložen textový řetězec argumentu (tj. typ `char*`).
 - Textový řetězec (argument) je posloupnost znaků (typ `char`) zakončený znakem `'\0'`.
„null character“ – konec textového řetězce
 - Alokace paměti pro uložení argumentů (textových řetězců) je provedena při spuštění programu.
V případě programu pro OS zajišťuje zavaděč programu („loader“) a standardní knihovna C.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 26 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Příklad programu s výstupem na stdout a přeměrováním

```
#include <stdio.h>
int main(int argc, char *argv[])
{
int ret = 0;
if (argc > 1) {
printf(stdout, "1st argument is %s\n", argv[1]);
} else {
printf(stdout, "1st argument is not given\n");
printf(stderr, "At least one argument must be given!\n");
ret = -1;
}
return ret;
}
```

lec04/demo-stdout.c

- Příklad výstupu – `clang demo-stdout.c -o demo-stdout`
`./demo-stdout; echo $?` `./demo-stdout 2>stderr`
Program has been called as ./demo- Program has been called as ./demo-stdout
1st argument is not given 1st argument is not given
- At least one argument must be given! `./demo-stdout ARGUMENT 1>stdout; echo $?`
255 0

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 30 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Funkce a předávání parametrů – příklad

- Proměnná `a` realizuje volání hodnotou, proměnná `b` realizuje „volání odkazem“.

```
void fce(int a, char* b)
{
a += 1;
(*b)++;
}
int a = 10;
char b = 'A';
printf("Before call a: %d b: %c\n", a, b);
fce(a, &b);
printf("After call a: %d b: %c\n", a, b);
```

- Výstup
Before call a: 10 b: A
After call a: 10 b: B

lec04/function_call.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 24 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Předávání parametrů programu

- Při spuštění programu můžeme předat parametry programu prostřednictvím argumentů.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
printf("Number of arguments %i\n", argc);
for (int i = 0; i < argc; ++i) {
printf("argv[%i] = %s\n", i, argv[i]);
}
return argc > 1 ? 0 : 1;
}
```

lec04/demo-arg.c

- Voláním `return` ve funkci `main()` vracíme z programu návratovou hodnotu, se kterou můžeme dále pracovat.
`./arg >/dev/null; echo $?`
1 *Návratová hodnota programu je uložena v proměnné \$?, kterou lze vypsat příkazem echo.*
`./arg first >/dev/null; echo $?`
0 *>/dev/null přeměruje standardní výstup do /dev/null.*

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 28 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Ukazatele (pointery) a pole

- Pointer ukazuje na vyhrazenou část paměti proměnné.
Předpokládáme správné použití.
- Pole je označení souvislého bloku paměti.
`int *p; //ukazatel (adresa) kde je uložena hodnota int`
`int a[10]; //souvislý blok pameti pro 10 int hodnot`
`sizeof(p); //pocet bytu pro ulozeni adresy (8 pro 64bit)`
`sizeof(a); //velikost alokovaneho pole je 10*sizeof(int)`
- Obě proměnné odkazují na paměť, kompilátor s nimi však pracuje rozdílně.
 - Proměnná typu pole je symbolické jméno pro místo v paměti, kde jsou uloženy hodnoty prvků pole.
Kompilátor nahrazuje jméno přímo pamětovým místem.
 - Ukazatel obsahuje adresu, na které je příslušná hodnota (nepřímé adresování).
- Při předávání pole jako parametru funkce je předáváno pole jako pointer (ukazatel).
Viz kompilace souboru `main_env.c` překladačem `clang`.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 32 / 53

Příklad kompilace funkce s předáváním pole 1/2

- Argument funkce je pole.

```

1 void fce(int array[])
2 {
3     int local_array[] = { 2, 4, 6 };
4     printf("sizeof(array) = %lu -- sizeof(local_array) = %lu\n",
5           sizeof(array), sizeof(local_array));
6     for (int i = 0; i < 3; ++i) {
7         printf("array[%i]=%i local_array[%i]=%i\n", i, array[i], i,
8               local_array[i]);
9     }
10 ...
11 int array[] = { 1, 2, 3 };
12 fce(array);

```

lec04/fce_array.c

- Po příkladu (`gcc -std=c99`) na amd64
 - `sizeof(array)` vrátí velikost 8 bajtů (64-bitová adresa);
 - `sizeof(local_array)` vrátí velikost 12 bajtů (3×4 bajty – int).
- Pole se funkcím předává jako ukazatel na adresu prvního prvku.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 33 / 53

Příklad kompilace funkce s předáváním pole 2/2

- Kompilátor `clang` (ve výchozím nastavení) upozorňuje na záměnu `int*` za `int[]`.

```

$ clang fce_array.c
fce_array.c:7:16: warning: sizeof on array function parameter will return
size
of 'int*' instead of 'int []' [-Wsizeof-array-argument]
    sizeof(array), sizeof(local_array));
    ^
fce_array.c:3:14: note: declared here
void fce(int array[])
    ^
1 warning generated.

```

lec04/fce_array.c

- Program lze zkompilovat, ale u předáváného pole se nelze spoléhat na velikost `sizeof`.
- Ukazatel nenese informaci o velikosti alokované paměti!

Pole ano „hlídá za nás kompilátor.“

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 34 / 53

Ukazatele a pole

- Proměnná pole `int a[3] = {1,2,3};`
a odkazuje na adresu prvního prvku pole.
- Proměnná ukazatel `int *p = a;`
Ukazatel p obsahuje adresu prvního prvku pole.
- Hodnota `a[0]` přímo reprezentuje hodnotu na adrese `0x10`.
- Hodnota `p` je adresa `0x10`, kde je uložena hodnota prvního prvku pole.
- Přiřazení `p = a` je legitimní. Kompilátor zajistí přiřazení adresy prvního prvku do ukazatele.
- Přístup ke druhému prvku lze `a[1]` nebo `p[1]`.
- Oběma způsoby se dostaneme na příslušné prvky pole, způsob je však odlišný — ukazatele využívají tzv. *pointerovou aritmetiku*.

variable names memory

a	→	1	0x10
int a[3]={1,2,3};	→	2	0x14
	→	3	0x18
p	→	0x10	0x1C

p=a;

<http://e11.thegreenplace.net/2009/10/21/are-pointers-and-arrays-equivalent-in-c>

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 35 / 53

Příklad ukazatele a pole

```

1 int a[] = { 1, 2, 3, 4 };
2 int b[] = { [3] = 10, [1] = 1, [2] = 5, [0] = 0 };
//initialization

4 // b = a; It is not possible to assign arrays
5 for (int i = 0; i < 4; ++i) {
6     printf("a[%i] =%3i b[%i] =%3i\n", i, a[i], i,
7           b[i]);
8 }

9 int *p = a; //you can use *p = &a[0], but not *p =
10 &a
11 a[2] = 99;

12 printf("\nPrint content of the array 'a' with
13 pointer arithmetic\n");
14 for (int i = 0; i < 4; ++i) {
15     printf("a[%i] =%3i p[%i] =%3i\n", i, a[i], i,
16           *(p+i));
17 }

```

lec04/array_pointer.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 36 / 53

Příklad předání ukazatele na pole

- Předáním pole jako ukazatele nemáme informaci o počtu prvků.
- Proto můžeme explicitně předat počet prvků v proměnné `n`.

```

1 #include <stdio.h>

3 void fce(int n, int *array);

5 int main(void)
6 {
7     int array[] = {1, 2, 3};
8     fce(sizeof(array)/sizeof(int), array); // pocet prvku
9     return 0;
10 }

12 void fce(int n, int *array) // array je lokální proměnná
13 { // typu ukazatel, můžeme změnit obsah paměti proměnné definované v main()
14     int local_array[] = {2, 4, 6};
15     printf("sizeof(array) = %lu, n = %i -- sizeof(local_array) = %lu\n",
16           sizeof(array), n, sizeof(local_array));
17     for (int i = 0; i < 3 && i < n; ++i) { //testujeme take n!
18         printf("array[%i]=%i local_array[%i]=%i\n", i, array[i], i, local_array[i]);
19     }
20 }

```

lec04/fce_pointer.c

- Přes ukazatel `array` v `fce()` máme přístup do pole z `main()`.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 37 / 53

Příklad předání pole včetně velikosti využitím VLA

- VLA (Variable Length Array) – délka pole určena za běhu programu. Pole je však stále předáváno jako ukazatel. Získáváme tak především přehlednost kódu.

```

1 void print_array(int n, int a[n]);

3 int main(int argc, char *argv[])
4 {
5     int n = 10;
6     if (argc > 1 && sscanf(argv[1], "%d", &n) != 1) {
7         fprintf(stderr, "Warning: cannot parse number from argv[1]!\n");
8     }
9     printf("Size of the array is %lu\n", sizeof(array));
10    int array[n]; //via array size depends on n
11    for (int i = 0; i < n; ++i) {
12        array[i] = 2*i;
13    }
14    print_array(n, array);
15    return 0;
16 }

18 void print_array(int n, int a[n])
19 {
20     printf("Size of the array a is %lu\n", sizeof(a));
21     for (int i = 0; i < n; ++i) {
22         printf("array[%i]=%i\n", i, a[i]);
23     }
24 }

```

lec04/fce_vla.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 38 / 53

Vícerozměrná pole

- Pole můžeme definovat jako vícerozměrná, např. 2D matice.

```

1 int m[3][3] = {
2     { 1, 2, 3 },
3     { 4, 5, 6 },
4     { 7, 8, 9 }
5 };

7 printf("Size of m: %lu == %lu\n", sizeof(m), 3 * 3 * sizeof(int));

9 for (int r = 0; r < 3; ++r) {
10     for (int c = 0; c < 3; ++c) {
11         printf("%3i", m[r][c]);
12     }
13     printf("\n");
14 }

```

lec04/matrix.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 39 / 53

Vícerozměrná pole a vnitřní reprezentace

- Vícerozměrné pole je **vždy** souvislý blok paměti.
Např. `int a[3][3]`; reprezentuje alokovanou paměť o velikosti `9*sizeof(int)`, tj. zpravidla 36 bajtů. Operátor `[]` nám tak především zjednodušuje zápis programu.

```

1 int *pm = (int *)m; // ukazatel na souvislou oblast m
2 printf("m[0][0]=%i m[1][0]=%i\n", m[0][0], m[1][0]); // 1 4
3 printf("pm[0]=%i pm[3]=%i\n", m[0][0], m[1][0]); // 1 4

```

lec04/matrix.c

- Dvourozměrné pole lze také definovat jako ukazatel na ukazatele (pole ukazatelů) na hodnoty konkrétního typu, např.
 - `int **a;` – ukazatel na ukazatele.
 - V obecném případě však takový ukazatel nemusí odkazovat na souvislou oblast, kde jsou alokovány jednotlivé prvky.
 - Proto při přístupu jako do jednorozměrného pole `int *b = (int *)a;` nelze garantovat přístup do druhého řádku jako v přechodném příkladě.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 40 / 53

Pole a vícerozměrná pole jako parametr funkce

- Parametr funkce je ukazatel na pole, např. typu `int`

```

1 int (*p)[3] = m; // pointer to array of int
// Size of p: 8
// Size of *p: 12

3 printf("Size of p: %lu\n", sizeof(p));
4 printf("Size of *p: %lu\n", sizeof(*p)); // 3 * sizeof(int) = 12

```

- Funkci nelze deklarovat s argumentem typu `[][]`, např. `int fce(int a[][]);` neboť kompilátor nemůže určit adresu pro přístup na `a[i][j]`, neboť se používá adresová aritmetika odpovídající 2D poli.
Pro `int m[row][col]` totiž `m[i][j]` odpovídá hodnotě na adrese `*(m + col * i + j)`
- Je však možné funkci deklarovat například jako
 - `int g(int a[]);` což odpovídá deklaraci `int g(int *a);`
 - `int fce(int a[][13]);` – je znám počet sloupců
 - nebo `int fce(int a[3][3]);`.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 41 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Inicializace pole

- Při definici můžeme hodnoty prvků pole inicializovat postupně nebo indexovaně.
- Při částečné inicializaci jsou ostatní prvky nastaveny na 0. 2D pole jsou inicializována po řádcích.

```

1 #define ROWS 3
2 #define COLS 3
3 void print(int rows, int cols, int m[rows][cols])
4 {
5     for (int r = 0; r < rows; ++r) {
6         for (int c = 0; c < cols; ++c) {
7             printf("%i ", m[r][c]);
8         }
9         printf("\n");
10    }
11 }
12
13 int m0[ROWS][COLS];
14 int m1[ROWS][COLS] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
15 int m2[ROWS][COLS] = { 1, 2, 3 };
16 int m3[ROWS][COLS] =
17 { [0][0] = 1, [1][1] = 2, [2][2] = 3 };
18
19 print(ROWS, COLS, m0);
20 print(ROWS, COLS, m1);
21 print(ROWS, COLS, m2);
22 print(ROWS, COLS, m3);

```

lec04/array-inits.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 42 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Řetězcové literály

- Formát – posloupnost znaků a řídicích znaků (escape sequences) uzavřená v uvozovkách. "Řetězcová konstanta s koncem řádku\n"
- Řetězcové konstanty oddělené oddělovači (white spaces) se sloučí do jediné, např. "Řetězcová konstanta " s koncem řádku\n"
- Typ
 - Řetězcová konstanta je uložena v poli typu char a zakončena znakem '\0'. Např. řetězcová konstanta "word" je uložena jako

```

'w' 'o' 'r' 'd' '\0'

```

Pole tak musí být vždy o 1 položku delší než je vlastní text!

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 44 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Textový řetězec

- Textový řetězec můžeme inicializovat jako pole znaků, tj. char[].

```

1 char str[] = "123";
2 char s[] = {'5', '6', '7'};
3
4 printf("Size of str %lu\n", sizeof(str));
5 printf("Size of s %lu\n", sizeof(s));
6 printf("str '%s'\n", str);
7 printf(" s '%s'\n", s);

```

lec04/array_str.c

- Pokud není řetězec zakončen znakem '\0', jako v případě proměnné char s[], pokračuje výpis řetězce až do nejbližšího znaku '\0'.
- Na textový řetězec lze odkazovat ukazatelem na znak char*.

```

1 char *sp = "ABC";
2 printf("Size of ps %lu\n", sizeof(sp));
3 printf(" ps '%s'\n", sp);

```

Size of ps 8
ps 'ABC'

- Velikost ukazatele je 8 bytů (pro 64-bit architekturu).
- Textový řetězec musí být zakončen znakem '\0'.

Alternativně lze řešit vlastní implementaci s explicitním uložením délky řetězce.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 45 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Načítání textových řetězců

- Správnost alokace vstupních argumentů je zajištěna při spuštění.
- Načtení textového řetězce funkcí scanf().
- Použitím %s může dojít k přepisu paměti.
- Načtení maximálně 4 znaků zajistíme řídicím řetězcem "%4s".

```

1 char str0[4] = "PRG"; // +1 \0
2 char str1[5]; // +1 for \0
3 printf("String str0 = '%s'\n", str0);
4 printf("Enter 4 chars: ");
5 scanf("%s", str1);
6 printf("You entered string '%s'\n", str1);
7 printf("String str0 = '%s'\n", str0);

```

Příklad výstupu programu:
String str0 = 'PRG'
Enter 4 chars: 1234567
You entered string '1234567'
String str0 = '67'
lec04/str_scanf-bad.c

```

1 char str0[4] = "PRG";
2 char str1[5];
3 ...
4 scanf("%4s", str1);
5 printf("You entered string '%s'\n", str1);
6 printf("String str0 = '%s'\n", str0);

```

Příklad výstupu programu:
String str0 = 'PRG'
Enter 4 chars: 1234567
You entered string '1234'
String str0 = 'PRG'
lec04/str_scanf-limit.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 46 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Řetězce, ukazatele a pole – Příklad vizualizace

```

1 char s0[4] = "PRG";
2 char s1[2];
3 char *str = NULL;
4
5 str = s0; // str je ukazatel
6
7 str[2] = 'G'; // nepřímé adresování
8
9 // s1 zatím není null-terminated
10 s1[0] = 'X';
11 s1[1] = '\0';
12
13 set_string(s0) // pole jako ukazatel
14
15 // pohled na řetězce jako pole znaků
16 str = s1;
17 set_string(str);

```

predchozi data s0: char[4] s1

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 47 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Zjištění délky textového řetězce

- Textový řetězec v C je posloupnost znaků zakončená znakem '\0'. Je to souvislý blok paměti zakončený '\0'.
- Paměť (blok znaků) alokujeme jako proměnnou typu pole char[].
- Řetěz je ukazatel, hodnota je adresa v paměti, kde začíná posloupnost znaků, typ char*. Např. textový literál char *str = "Textový literál";
- Délku textového řetězce lze zjistit sekvenčním procházením odkazované části paměti znak po znaku až k '\0'.
- Funkce strlen() ze standardní knihovny <string.h> pro práci s řetězi.
- Z principu má takový dotaz na délku řetězce lineární složitost O(n).

```

1 int getLength(char *str)
2 {
3     int ret = 0;
4     while (str && str[ret] != '\0') {
5         ret ++;
6     }
7     return ret;
8 }

```

```

1 for (int i = 0; i < argc; ++i) {
2     printf("argv[%i]: getLength = %i -- strlen = %lu\n",
3           i, getLength(argv[i]), strlen(argv[i]));
4 }

```

\$ clang string_length.c && ./a.out
argv[0]: getLength = 7 -- strlen = 7

lec04/string_length.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 48 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Práce s textovými řetězci

- V C jsou řetězce pole znaků zakončené znakem '\0'.
- Základní operace jsou definovány v knihovně <string.h>, například pro kopírování nebo porovnání řetězců.
 - char* strcpy(char *dst, char *src);
 - int strcmp(const char *s1, const char *s2);
 - Funkce předpokládají dostatečný rozsah alokovaných polí
 - Funkce s explicitním limitem na maximální délku řetězců: char* strncpy(char *dst, char *src, size_t len); int strncmp(const char *s1, const char *s2, size_t len);
- Převod řetězce na číslo – <stdlib.h>
 - atoi(), atof() – převod celého a necelého čísla.
 - long strtol(const char *nptr, char **endptr, int base);
 - double strtod(const char *nptr, char **restrict endptr);

Funkce atoi() a atof() jsou „obsolete“, ale mohou být rychlejší.

- Alternativně také např. sscanf().

Více viz man strcpy, strcmp, strtol, strtod, sscanf.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 49 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Část II

Část 2 – Zadání 3. domácího úkolu (HW3)

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 50 / 53

Pole Ukazatele Funkce a předávání parametrů Vstup a výstup programu Ukazatele a pole Textové řetězce

Zadání 3. domácího úkolu HW3

Téma: Prvočíselný rozklad

Povinné zadání: 3b; Volitelné zadání: není; Bonusové zadání: 5b

- **Motivace:** Rozvinout znalost použití cyklů, proměnných a jejich reprezentace ve výpočetní úloze.
- **Cíl:** Osvojit si algoritmické řešení výpočetní úlohy
- **Zadání:** <https://cw.fel.cvut.cz/wiki/courses/bab36prga/hw/hw3>
 - Načtení posloupnosti kladných celých čísel (v rozsahu 64-bitů znaménkového typu) zakončených nulou a jejich rozklad na prvočinitele. S ohledem na výpočetní náročnost řešení vyžaduje sofistikovanější přístup výpočtu s využitím techniky *Eratostenova síta*.
 - **Bonusové zadání** dále úlohu rozšiřuje zpracování čísel s až 100 ciframi. Řešení vyžaduje implementaci *vlastní reprezentace velkých celých čísel* spolu s *operacemi* celocíselného dělení se zbytkem.
- **Termín odevzdání:** 30.03.2024, 23:59:59 PDT.
- **Bonusová úloha:** 24.05.2024, 23:59:59 CEST.

PDT – Pacific Daylight Time

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 51 / 53

Shrnutí přednášky

Kódovací příklad – hexdump – 1/4

- Implementujme program, který vytiskne vstup načtený ze stdin na stdout v hexa formátu.

Obdobu programu hexdump.

```
# cat hw01-2.out
# hexdump -C hw01-2.out
00000000 44 65 73 69 74 6b 6f 6e 76 61 20 73 6f 75 73 74 61 [Data: hexout]
00000010 76 61 3a 20 33 37 35 39 20 24 31 30 30 30 0a [va: 3759 -10000]
00000020 53 65 73 74 6e 61 6b 6f 76 61 20 73 6f 75 73 6f 75 [Data:actovce sousta]
00000030 73 74 61 76 61 6b 6f 76 61 6b 6f 66 66 66 66 64 [Data: eaf fffff]
00000040 38 66 30 0a 53 6f 76 63 65 74 3a 20 33 37 35 39 [Hex:Soucet: 3759]
00000050 20 2b 20 24 31 30 30 30 30 20 34 20 24 36 32 34 [i = -10000 = -654]
00000060 31 0a 52 6f 7a 64 69 6c 3a 20 33 37 35 39 20 24 [i:Soucet: 3759 -]
00000070 20 24 31 30 30 30 30 20 34 20 31 33 37 35 39 0a [-10000 = 13759 -]
00000080 53 6f 76 63 69 6a 3a 20 33 37 35 39 20 24 20 34 [Soucet: 3759 * -1]
00000090 31 30 30 30 30 30 30 20 24 33 37 35 39 30 30 30 [10000 = -375900]
000000a0 30 0a 50 6f 64 69 6c 3a 20 33 37 35 39 20 2f 20 [0: Puskli: 3759 / -]
000000b0 2d 31 30 30 30 30 20 34 20 30 0a 50 72 75 64 65 [i: -10000 = 0: Pruse]
000000c0 72 3a 20 2d 33 31 32 30 2b 35 0a [i: -3120.5 /]
000000cb
```

Kódovací příklad – hexdump – 4/4

- Program spustíme s přeměrováním stdin ze souboru, např. hw01-2.out a přeměrováním výstupu stdout do souboru hex.out.
- Při kompilaci definujeme počet hodnot na řádek MAX_WIDTH=20 a program spustíme s přeměrováním stderr do souboru hex.err.

```
# clang hexdump-2.c -o hexdump2k ./hexdump.c -hw01-2.out > hex.out
# hexdump2k
# cat hex.out
44 65 73 69 74 6b 6f 6e 76 61 20 73 6f 75 73 74 61
76 61 3a 20 33 37 35 39 20 24 31 30 30 30 0a
53 65 73 74 6e 61 6b 6f 76 61 20 73 6f 75 73 6f 75
73 74 61 76 61 6b 6f 76 61 6b 6f 66 66 66 66 64
38 66 30 0a 53 6f 76 63 65 74 3a 20 33 37 35 39
20 2b 20 24 31 30 30 30 30 20 34 20 24 36 32 34
31 0a 52 6f 7a 64 69 6c 3a 20 33 37 35 39 20 24
20 24 31 30 30 30 30 20 34 20 31 33 37 35 39 0a
53 6f 76 63 69 6a 3a 20 33 37 35 39 20 24 20 34
20 24 31 30 30 30 20 24 33 37 35 39 30 30 30
30 0a 50 6f 64 69 6c 3a 20 33 37 35 39 20 2f 20
2d 31 30 30 30 30 20 34 20 30 0a 50 72 75 64 65
72 3a 20 2d 33 31 32 30 2b 35 0a
```

```
# clang MAX_WIDTH=20 hexdump.c -o hexdump2k ./hexdump.c -hw01-2.out > hex.err
# cat hex.err
44 65 73 69 74 6b 6f 6e 76 61 20 73 6f 75 73 74 61
24 31 30 30 30 20 0a 53 65 73 74 6e 61 6b 6f 76 61 6b
6f 66 66 66 66 30 0a 53 6f 76 63 65 74 3a 20 33 37 35
20 2b 20 24 31 30 30 30 30 20 34 20 24 36 32 34 31
0a 52 6f 7a 64 69 6c 3a 20 33 37 35 39 20 24 20 34
7a 64 69 6c 3a 20 33 37 35 39 20 2d 20 2d 31 30 30
30 20 34 20 31 33 37 35 39 0a 53 6f 76 63 69 6a 3a
20 33 37 35 39 30 30 30 20 24 33 37 35 39 30 30 30
30 0a 50 6f 64 69 6c 3a 20 33 37 35 39 20 2f 20 2d
31 30 30 30 30 20 34 20 30 0a 50 72 75 64 65 72
3a 20 2d 33 31 32 30 2b 35 0a
```

- Obsah stderr není přeměrován, proto se vypíše na terminál.
 - Obsah stderr je přeměrování, proto se nevypíše na terminál.
- Vyzkoušejte program s přeměrováním binárního souboru na stdin našeho hexdump.*

Diskutovaná témata

- Jednorozměrná a vícerozměrná pole a jejich inicializace
- Ukazatel
- Textový řetězec
- Rozdíl mezi polem a ukazatelem
- Předávání polí funkcím
- Vstup a výstup programu - argumenty programy a návratová hodnota

- Příště: Ukazatele, paměťové třídy a volání funkcí.**

Kódovací příklad – hexdump – 2/4

- Program napíšeme nejdříve kreativně, ale s počtem hodnot na řádek MAX_WIDTH.

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

#ifdef MAX_WIDTH
#define MAX_WIDTH 16
#endif

int main(void)
{
    int ret = EXIT_SUCCESS;
    int n = 0; // initialize
    int c; // we do not need to init.

    while ((c = getchar()) != EOF) {
        if (n >= MAX_WIDTH) {
            n = 0;
            putchar('\n');
        }
        if (n > 0) {
            printf(", ");
        }
        printf("%02x", c);
        n += 1;
    } //end while loop
    if (n > 0) {
        putchar('\n');
    }
    printf(stderr, "HEXDUMP END\n");
    return ret;
}
```

Ukazatelová (pointerová) aritmetika

- S ukazately (pointery) lze provádět aritmetické operace + a - (přičítat nebo odčítat celé číslo).
- ukazatel = ukazatel stejného typu + (nebo -) a celé číslo (int).
- Nebo lze používat zkrácený zápis např. ukazatel += 1 a unární operátory např. ukazatel++.
- Aritmetické operace jsou užitečné pokud ukazatel odkazuje na více položek daného typu (souvislý blok paměti).
 - Např. pole položek příslušného typu;
 - Dynamicky alokovaný souvislý blok paměti.
- Přičtením hodnoty celého čísla k pointeru „posouváme“ hodnotu pointeru na další prvek, např.

```
int a[10];
int *p = a;

int i = *(p+2); //odkazuje na hodnotu 3. prvku pole a
```

 - Podle typu ukazatele se hodnota adresy příslušně zvýší.
 - (p+2) je ekvivalentní adrese p + 2*sizeof(int).
 - Příklad použití viz lec04/pointers_and_array.c

Část IV

Appendix

Kódovací příklad – hexdump – 3/4

- Program upravíme redukcí počtu zanořených vyjmutím (dekompozice).

```
void print_hex(int c, int *n);

int main(void)
{
    int n = 0;
    int c; // we do not need to init

    fprintf(stderr, "HEXDUMP:\n");
    while ((c = getchar()) != EOF) {
        print_hex(c, &n);
    } //end while loop
    if (n > 0) {
        putchar('\n'); // final EOL
    }
    fprintf(stderr, "HEXDUMP END\n");
    return EXIT_SUCCESS; // always ok
}
```

- Předávání ukazatele (adresy) proměnné n je nutné.
- Vyzkoušejte chování programu s předáváním hodnoty n.

```
void print_hex(int c, int n);
```

Příklad ukazatelové aritmetiky

- Ukazatel je proměnná jejíž hodnota je adresa v paměti.
- Ukazatelová aritmetika zohledňuje typ proměnné, její velikost v paměti.
- Přičtením hodnoty 1 k proměnné typu ukazatel je vypočtena adresa následujícího prvku, adresa je zvětšena o hodnotu odpovídající sizeof() příslušného typu.

<pre>int getLength(char *str) { int ret = 0; while (str && str[ret] != '\0') { ret += 1; } return ret; }</pre>	<pre>int getLengthPtr1(char *str) { int ret = 0; while (str && (*str++) != '\0') { ret += 1; } return ret; }</pre>	<pre>int getLengthPtr2(char *str) { int ret = 0; while (str && (str++) ret += 1; return ret; }</pre>
--	--	--

- Textový řetězec můžeme interpretovat jako pole znaků char[] nebo ukazatel char*.

```
lec04/string_length.c
```