

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 4. domácího úkolu (HW4)

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

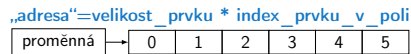
Část I

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 [];

- Závorky [] slouží také k přístupu (adresaci) prvku.
proměnná_typu_pole [index_prvku_pole]

Příklad definice proměnné typu pole hodnot typu int.
Alokace paměti pro až 10 prvků pole.

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

Tj. $10 \times sizeof(int)$

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. $n * sizeof(typ)$.
- Textový řetězec je pole typu **char**, kde poslední prvek je **'\0'**.

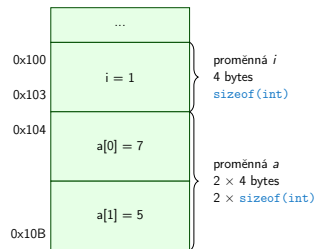
C nekontroluje za běhu programu, zdali je index platný!

Např. přístup do pole $a[1000]$.

Pole – Příklad vizualizace alokace přiřazení hodnot

- Proměnná typu pole označuje na **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 zpravidla alokovány od horní adresy k adresám nižším.

Pole – Příklad 1/3

- Definice jednorozměrného a dvourozměrného pole.
/ jednorozměrné pole prvku typu char */
char simple_array[10];

/* dvourozměrné 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>  
2  
3 int main(void)  
4 {  
5     int array[5];  
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] = 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  
3 int main(void)  
4 {  
5     int array[10];  
6  
7     for (int i = 0; i < 10; i++) {  
8         array[i] = i;  
9     }  
10  
11    int n = 5;  
12    int array2[n * 2];  
13  
14    for (int i = 0; i < 10; i++) {  
15        array2[i] = 3 * i - 2 * i * i;  
16    }  
17  
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 }
```

Size of array: 40
array[0]=0 array2[0]= 0
array[1]=+1 array2[1]= 1
array[2]=+2 array2[2]= -2
array[3]=+3 array2[3]= -9
array[4]=+4 array2[4]= -20
array[5]=+5 array2[5]= -35
array[6]=+6 array2[6]= -54
array[7]=+7 array2[7]= -77
array[8]=+8 array2[8]= -104
array[9]=+9 array2[9]= -135

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" };

```

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 11 / 53

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 vlastnosti 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.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 15 / 53

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 sílebníku jsou však zpravidla alokovány od horní adresy k adresám nížším.

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

- Ukazatele jsou proměnné, které uchovávají adresy jiných proměnných.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 18 / 53

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

```

lec04/fce_var_array.c

- Pole variabilní délky však nelze v definici inicializovat.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 12 / 53

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);
6
7 Value of a 10, address of a 0x7fffffff95c
8 Value of p 0x7fffffff95c, address of p 0x7fffffff950

```

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 16 / 53

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

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 19 / 53

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 paměťových třídách a dynamické alokaci v 5. přednášce.

- Pole může být argumentem funkce

```

void fce(int array[]);

```

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

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 13 / 53

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 */

```

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 17 / 53

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, která je uložena v paměti, na kterou odkazuje hodnota proměnné typu ukazatel.
- Operátor `&` vrací adresu, kde je uložena hodnota proměnné.

Jan Faigl, 2024 BAB36PRGA – Přednáška 04: Pole a ukazatele 20 / 53

Pole	Ukazatele	Funkce a předávání parametrů	Vstup a výstup programu	Ukazatele a pole	Textové řetězce
<h2>Ukazatele (pointery) a kódovací styl</h2> <ul style="list-style-type: none"> Typ ukazatel se značí symbolem <code>*</code>. <code>*</code> můžeme zapisovat u jména typu nebo jména proměnné. Preferujeme zápis u proměnné, abychom předešli omylům. <ul style="list-style-type: none"> <code>char* a, b, c;</code> <code>char *a, *b, *c;</code> <p><i>Pointer je pouze a</i> <i>Všechny tři proměnné jsou ukazatele.</i></p> <ul style="list-style-type: none"> Zápis typu ukazatele na ukazatel <code>char **a;</code>. Zápis pouze typu (bez proměnné): <code>char*</code> nebo <code>char**</code>. Ukazatel na proměnnou prázdného typu zapisujeme jako <code>void *ptr</code>. Prokazatelně neplatná adresa má symbolické jméno <code>NULL</code>. <ul style="list-style-type: none"> <i>Definovaná jako makro preprocesoru (C99 lze použít 0).</i> Proměnné v C nejsou automaticky inicializovány a ukazatele tak mohou odkazovat na neplatnou paměť, proto může být vhodné explicitně inicializovat ukazatele na <code>0</code> nebo <code>NULL</code>. <ul style="list-style-type: none"> <i>Např. <code>int *i = NULL;</code></i> 					
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
<h2>Funkce main a její tvary</h2> <ul style="list-style-type: none"> Základní tvar funkce <code>main</code> <pre>int main(int argc, char *argv[]) { ... }</pre> Alternativně pak také <pre>int main(int argc, char **argv) { ... }</pre> Argumenty funkce nejsou nutné <pre>int main(void) { ... }</pre> Rozšířená funkce o nastavení proměnných prostředí <pre>int main(int argc, char **argv, char **envp) { ... }</pre> <i>Průběh k proměnným prostředí funkci <code>getenv()</code> z knihovny <code><stdlib.h></code>.</i> <ul style="list-style-type: none"> <code>lec04/main_env.c</code> Rozšířená funkce o specifické parametry Mac OS X <pre>int main(int argc, char **argv, char **envp, char **apple);</pre> 					
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
<h2>Interakce programu s uživatelem</h2> <ul style="list-style-type: none"> Funkce <code>int main(int argc, char *argv[])</code> <ul style="list-style-type: none"> Při spuštění programu lze předat parametry (textové řetězce). Při ukončení programu lze předat návratovou hodnotu. <ul style="list-style-type: none"> <i>Konvence 0 bez chyby, ostatní hodnoty chybový kód.</i> Při běhu programu lze číst ze standardního vstupu a zapisovat na standardní výstup. <ul style="list-style-type: none"> <i>Např. <code>scanf()</code> nebo <code>printf()</code></i> Při spuštění programu lze vstup i výstup přeměrovat z/do souboru. <ul style="list-style-type: none"> <i>Program tak nečeká na vstup uživatele (stisk klávesy „Enter“).</i> Každý program (terminálový) má standardní vstup (<code>stdin</code>) a výstup (<code>stdout</code>) a dále pak standardní chybový výstup (<code>stderr</code>), které lze v shellu přeměrovat. <pre>./program <stdin.txt >stdout.txt 2>stderr.txt</pre> Alternativou k <code>scanf()</code> a <code>printf()</code> lze využít <code>fscanf()</code> a <code>fprintf()</code>. <ul style="list-style-type: none"> Funkce mají první argument soubor jinak, je syntax identická. Soubory/proudy <code>stdin</code>, <code>stdout</code> a <code>stderr</code> jsou definovány v <code><stdio.h></code>. 					
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
<h2>Funkce a předávání parametrů</h2> <ul style="list-style-type: none"> V C jsou parametry funkce předávány hodnotou. Parametry jsou lokální proměnné funkce (alokované na zásobníku), které jsou inicializovány na hodnotu předávanou funkci. <ul style="list-style-type: none"> <i>Více o volání funkci a paměti v 5. přednášce.</i> <pre>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 také na zásobníku)*/ }</pre> <ul style="list-style-type: none"> 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
<h2>Argumenty funkce main</h2> <ul style="list-style-type: none"> Základní tvar funkce <code>main</code> <pre>int main(int argc, char *argv[]) { ... }</pre> <code>argc</code> – obsahuje počet argumentů programu. <ul style="list-style-type: none"> <i>Včetně jména spouštěného programu.</i> Argumenty jsou textové řetězce oddělené mezerou (bílým znakem). <code>argv</code> – pole ukazatelů na hodnoty typu <code>char</code>. <ul style="list-style-type: none"> <i>Typ „čteme“ zprava doleva.</i> Pole <code>argv</code> má velikost (počet prvku) daný hodnotou <code>argc</code>. Každý prvek pole <code>argv[i]</code> obsahuje adresu, kde je uložen textový řetězec argumentu (tj. typ <code>char*</code>). Textový řetězec (argument) je posloupnost znaků (typ <code>char</code>) zakončený znakem <code>'\0'</code>. <ul style="list-style-type: none"> <i>„null character“ – konec textového řetězce</i> Alokace paměti pro uložení argumentů (textových řetězců) je provedena při spuštění programu. <ul style="list-style-type: none"> <i>V případě programu pro OS zajišťuje zavaděč programu („loader“) a standardní knihovna C.</i> 					
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
<h2>Příklad programu s výstupem na stdout a přeměrováním</h2> <pre>1 #include <stdio.h> 3 int main(int argc, char *argv[]) 4 { 5 int ret = 0; 7 fprintf(stdout, "Program has been called as %s\n", argv[0]); 8 if (argc > 1) { 9 fprintf(stdout, "1st argument is %s\n", argv[1]); 10 } else { 11 fprintf(stdout, "1st argument is not given\n"); 12 fprintf(stderr, "At least one argument must be given!\n"); 13 ret = -1; 14 } 15 return ret; 16 }</pre> <ul style="list-style-type: none"> Příklad výstupu – <code>clang demo-stdout.c -o demo-stdout</code> <pre>./demo-stdout; echo \$? Program has been called as ./demo-stdout 1st argument is not given At least one argument must be given! 255</pre> <code>./demo-stdout 2>stderr</code> <pre>Program has been called as ./demo-stdout 1st argument is not given</pre> <code>./demo-stdout ARGUMENT 1>stdout; echo \$?</code> <pre>0</pre> 					
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
<h2>Funkce a předávání parametrů – příklad</h2> <ul style="list-style-type: none"> Proměnná <code>a</code> realizuje volání hodnotou, proměnná <code>b</code> realizuje „volání odkazem“. <pre>1 void fce(int a, char* b) 2 { 3 a += 1; 4 (*b)++; 5 } 6 int a = 10; 7 char b = 'A'; 8 printf("Before call a: %d b: %c\n", a, b); 9 fce(a, &b); 10 printf("After call a: %d b: %c\n", a, b);</pre> <ul style="list-style-type: none"> Výstup <pre>Before call a: 10 b: A After call a: 10 b: B</pre> 					
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
<h2>Předávání parametrů programu</h2> <ul style="list-style-type: none"> Při spuštění programu můžeme předat parametry programu prostřednictvím argumentů. <pre>1 #include <stdio.h> 3 int main(int argc, char *argv[]) 4 { 5 printf("Number of arguments %i\n", argc); 6 for (int i = 0; i < argc; ++i) { 7 printf("argv[%i] = %s\n", i, argv[i]); 8 } 9 return argc > 1 ? 0 : 1; 10 }</pre> <ul style="list-style-type: none"> Voláním <code>return</code> ve funkci <code>main()</code> vracíme z programu návratovou hodnotu, se kterou můžeme dále pracovat. <ul style="list-style-type: none"> <i>Např. v interpretu příkazů (shellu).</i> <pre>./arg >/dev/null; echo \$? 1 ./arg first >/dev/null; echo \$? 0</pre> <ul style="list-style-type: none"> Návratová hodnota programu je uložena v proměnné <code>\$?</code>, kterou lze vpsát příkazem <code>echo</code>. <ul style="list-style-type: none"> <code>>/dev/null</code> přeměruje standardní výstup do <code>/dev/null</code>. 					
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
<h2>Ukazatele (pointery) a pole</h2> <ul style="list-style-type: none"> Pointer ukazuje na vyhrazenou část paměti proměnné. <ul style="list-style-type: none"> <i>Předpokládáme správné použití.</i> Pole je označení souvislého bloku paměti. <pre>int *p; //ukazatel (adresa) kde je uložena hodnota int int a[10]; //souvislý blok paměti pro 10 int hodnot sizeof(p); //pocet bytu pro ulozeni adresy (8 pro 64bit) sizeof(a); //velikost alokovaneho pole je 10*sizeof(int)</pre> Obě proměnné odkazují na paměť, kompilátor s nimi však pracuje rozdílně. <ul style="list-style-type: none"> Proměnná typu pole je symbolické jméno pro místo v paměti, kde jsou uloženy hodnoty prvků pole. <ul style="list-style-type: none"> <i>Kompilátor nahrazuje jméno přímo pamětovým místem.</i> Ukazatel obsahuje adresu, na které je příslušná hodnota (nepřímé adresování). Při předávání pole jako parametru funkce je předáváno pole jako pointer (ukazatel). <ul style="list-style-type: none"> <i>Viz kompilace souboru <code>main_env.c</code> překladačem <code>clang</code>.</i> 					
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, local_array[i]);
8     }
9 }
10 ...
11 int array[] = { 1, 2, 3 };
12 fce(array);

```

lec04/fce_array.c

- Po překladu (`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 zkompileovat, ale u předávaného pole se nelze spoléhat na velikost `sizeof`.
- Ukazatel nese 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 příslušný prvek 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, b[i]);
7 }
9 int *p = a; //you can use *p = &a[0], but not *p = &a
10 a[2] = 99;
12 printf("\nPrint content of the array 'a' with pointer arithmetic\n");
13 for (int i = 0; i < 4; ++i) {
14     printf("a[%i] =%3i p[%i] =%3i\n", i, a[i], i, *(p+i));
15 }

```

a[0] = 1 b[0] = 0
a[1] = 2 b[1] = 1
a[2] = 3 b[2] = 5
a[3] = 4 b[3] = 10

Print content of the array 'a' using pointer arithmetic
a[0] = 1 p+0 = 1
a[1] = 2 p+1 = 2
a[2] = 99 p+2 = 99
a[3] = 4 p+3 = 4

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) // array je lokální proměnná
4 { // typu ukazatel, můžeme změnit obsah paměti proměnné definované v main()
5     int local_array[] = { 2, 4, 6 };
6     printf("sizeof(array) = %lu, n = %i -- sizeof(local_array) = %lu\n",
7           sizeof(array), n, sizeof(local_array));
8     for (int i = 0; i < 3 && i < n; ++i) { //testujeme take n!
9         printf("array[%i]=%i local_array[%i]=%i\n", i, array[i], i, local_array[i]);
10    }
11 }
12 int main(void)
13 {
14     int array[] = { 1, 2, 3 };
15     fce(sizeof(array)/sizeof(int), array); // pocet prvku
16     return 0;
17 }

```

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.** Ziskáváme tak především přehlednost kódu.

```

1 void print_array(int n, int a[n])
2 {
3     printf("Size of the array is %lu\n", sizeof(a));
4     for (int i = 0; i < n; ++i) {
5         printf("array[%i]=%i\n", i, a[i]);
6     }
7 }
9 int main(int argc, char *argv[])
10 {
11     int n = 10;
12     if (argc > 1 && sscanf(argv[1], "%d", &n) != 1) {
13         fprintf(stderr, "Warning: cannot parse number from argv[1]!\n");
14     }
15     printf("Size of the array is %lu\n", sizeof(array));
16     int array[n]; //vla array size depends on n
17     for (int i = 0; i < n; ++i) {
18         array[i] = 2*i;
19     }
20     print_array(n, array);
21     return 0;
22 }

```

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 }

```

Size of m: 36 == 36
1 2 3
4 5 6
7 8 9

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

```

Size of p: 8
Size of *p: 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("%4i", m[r][c]);
8         }
9         printf("\n");
10    }
11 }
12 int m0[ROWS][COLS];
13 int m1[ROWS][COLS] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
14 int m2[ROWS][COLS] = { 1, 2, 3 };
15 int m3[ROWS][COLS] = { 1, 2, 3 };
16 { [0][0] = 1, [1][1] = 2, [2][2] = 3 };
17
18 print(ROWS, COLS, m0);
19 print(ROWS, COLS, m1);
20 print(ROWS, COLS, m2);
21 print(ROWS, COLS, m3);

```

m0 - not initialized
-584032767743694227
0 1 0
740314624 0 0

m1 - init by rows
1 2 3
4 5 6
7 8 9

m2 - partial init
1 2 3
0 0 0
0 0 0

m3 - indexed init
1 0 0
0 2 0
0 0 3

lec04/array-inits.c 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.
- Řetězcové konstanty oddělené oddělovači (white spaces) se sloučí do jediné, např.
- 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[]`.
- 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*`.
- Velikost ukazatele je 8 bytů (pro 64-bit architekturu).
- Textový řetězec musí být zakončen znakem `'\0'`.

Alternativně lze řešit vlastní implementací s explicitním uložením délky řetězce.

```

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);

```

Size of str 4
Size of s 3
str '123'
s '567123'
lec04/array_str.c

```

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'

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

- Správné alokace vstupních argumentů je zajištěna při spuštění.
- Načtení textového řetězce funkcí `scanf()`.
- Použitím `%s` může dojít k přepisu paměti.

Příklad výstupu programu:

```

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);

```

String str0 = 'PRG'
Enter 4 chars: 1234567
You entered string '1234567'
String str0 = '67'

lec04/str_scanf-bad.c

- Načtení maximálně 4 znaků zajistíme řídicím řetězcem `"%4s"`.

Příklad výstupu programu:

```

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);

```

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 // pohled na řetězec jako pole znaků
15 str = s1;
16 str = s1;
17 set_string(str);
18 ...
19
20 void set_string(char *s)
21 {
22     strcpy(s, "123");
23 }
24

```

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'`.
- Paměť (blok znaků) alokujeme jako proměnnou typu pole `char[]`.
- Řetěz je ukazatel, hodnota je adresa v paměti, kde začíná posloupnost znaků, typ `char*`.
- 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 += 1;
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

Část II

Část 2 – Zadání 4. domácího úkolu (HW4)

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

Zadání 4. domácího úkolu HW4

Téma: Caesarova šifra Povinné zadání: 3b; Volitelné zadání: není; Bonusové zadání: 5b

- Motivace:** Získat zkušenosti s dynamickou alokací paměti. Implementovat výpočetní úlohu optimalizačního typu.
- Cíl:** Osvojit si práci s dynamickou alokací paměti.
- Zadání:** <https://cw.fel.cvut.cz/wiki/courses/bab36prga/hw/hw4>
 - Načtení dvou vstupních textů a tisk dekodované zprávy na výstup.
 - Zakódovaný text i (spatně) odposlechnutý text mají stejné délky.
 - Nalezení největší shody dekodovaného a odposlechnutého textu na základě hodnoty posunu v Caesarově šifře.
 - Optimalizace hodnoty Hammingovy vzdálenosti. https://en.wikipedia.org/wiki/Hamming_distance
- Volitelné zadání rozšiřuje úlohu o uvažování chybějících znaků v odposlechnutém textu, což vede na využití Levenštejnovy vzdálenosti. https://en.wikipedia.org/wiki/Levenshtein_distance
- Termín odevzdání: 06.04.2024, 23:59:59 PDT.**
- Bonusová úloha: 24.05.2024, 23:59:59 CEST.**

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

