

Pole, ukazatel, textový řetězec, vstup a výstup programu

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 05

BOB36PRP – Procedurální programování

Přehled témat

■ Část 1 – Pole, ukazatele a řetězce

Pole

Ukazatele

Funkce a předávání parametrů

Vstup a výstup programu

Ukazatele a pole

Textové řetězce

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

■ Část 2 – Zadání 4. domácího úkolu (HW04)

Část I

Pole a ukazatele

Pole

- Datová struktura pro uložení **více hodnot stejného typu**.
- Slouží k reprezentaci posloupnosti hodnot v paměti. *Hodnoty uloženy v sousledném 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_prvního_prvku + velikost_typu * index_prvku_v_poli*
- Definicí proměnné dochází k alokaci paměti pro uložení definované 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 [];

- Závorčky [] 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];  
Tj. 10 × sizeof(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.

I strukturované typy, viz další přednáška.

- Pole může být jednorozměrné nebo vícerozměrné.

Pole polí (...) prvků stejného typu.

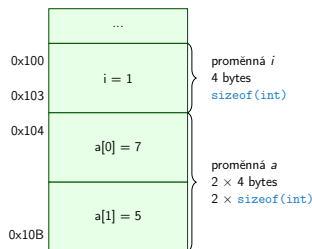
- Prvky pole určuje: **jméno, typ, počet prvků**.
- Prvky pole tvoří v paměti souvislou oblast!
- Velikost pole (v bajtech) je dána počtem prvků pole *n* a **typem** prvku, tj. **n * sizeof(typ)**.
- Textový řetězec je pole typu **char**, kde poslední prvek je **'\0'**.

C nekontroluje za běhu programu, zdali je index platný!

Pole – Příklad vizualizace alokace přiřazení hodnot

- Proměnná typu pole odkazuje na začátek paměti, kde jsou alokovány jednotlivé prvky pole.
- Přístup k prvkům je prostřednictvím indexového operátoru [], který určí adresu konkrétního prvku pole z typu proměnné.

```
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.
/ 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>  
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 }  
lec05/array.c  
Size of array: 20  
Item[0] = 1  
Item[1] = 0  
Item[2] = 740314624  
Item[3] = 0  
Item[4] = 0
```

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,  
21            array2[i]);  
22    }  
23 }  
lec05/demo-array.c  
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
```

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

```

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

lec05/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" };

```

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 ukazatele

- 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 neobzřetném používání mohou vést k chybám. Proto je důležité osvojit si princip nepřímého adresování a pochopit organizaci a přístup do paměti.

Ukazatele – Příklad vizualizace alokace přírazení 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.

- Ukazatele jsou proměnné, které uchovávají adresy jiných proměnných.

Pole variabilní délky

- 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.

```

void fce(int n)
{
    // int local_array[n] = { 1, 2 }; inicializace není dovolena
    int local_array[n]; // variable length array
    printf("sizeof(local_array) = %lu\n", sizeof(local_array));
    printf("length of array = %lu\n", sizeof(local_array) / sizeof(int));
    for (int i = 0; i < n; ++i) {
        local_array[i] = i * i;
    }
}
int main(int argc, char *argv[])
{
    fce(argc);
    return 0;
}

```

lec05/fce_var_array.c

- Pole variabilní délky však nelze v definici inicializovat.

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"`.
`int a = 10;`
`int *p = &a;`
`printf("Value of a %i, address of a %p\n", a, &a);`
`printf("Value of p %p, address of p %p\n", p, &p);`
Value of a 10, address of a 0x7fffffff95c
Value of p 0x7fffffff95c, address of p 0x7fffffff950

Ukazatel (pointer) – příklady 2/2

```

printf("i: %d -- pi: %p\n", i, pi); // 10 0x7fffffff8fc
printf("&i: %p -- *pi: %d\n", &i, *pi); // 0x7fffffff8fc 10
printf("*(&i): %d -- &(*pi): %p\n", *(&i), &(*pi));

printf("i: %d -- *pj: %d\n", i, *pj); // 10 10
i = 20;
printf("i: %d -- *pj: %d\n", i, *pj); // 20 20

printf("sizeof(i): %lu\n", sizeof(i)); // 4
printf("sizeof(pi): %lu\n", sizeof(pi)); // 8

long l = (long)pi;
printf("0x%lx %p\n", l, pi); /* print l as hex -- %lx */
// 0x7fffffff8fc 0x7fffffff8fc

l = 10;
pi = (int*)l; /* possible but it is nonsense */
printf("l: 0x%lx %p\n", l, pi); // 0xa 0xa

```

lec05/pointers.c

Pole ve funkci a jako argument funkce

- Lokálně definované pole ve funkci má rozsah platnosti pouze v rámci funkce (bloku).
`void fce(int n)`

```

{
    int array[n];
    // počítání s array
    {
        int array2[n*2];
    } // po skončení bloku array2 automaticky zaniká
    // zde již není array2 přístupné
} // po skončení funkce, pole array automaticky zaniká

```

 - Pole je automaticky vytvořeno a po skončení bloku (funkce) automaticky zaniká (paměť je uvolněna).
Více o paměťových třídách na 6. přednášce.
 - Lokální proměnné jsou ukládány na tzv. zásobník, který má zpravidla relativně malou velikost, proto pro velká pole může být vhodnější alokovat paměť dynamicky a použít **ukazatele**.
- Pole může být argumentem funkce
`void fce(int array[]);`
hodnota je však předávána jako **ukazatel!**

Proměnné typu ukazatel (pointer) – příklady

```

int i = 10; /* i -- promenna typu int
            &i -- adresa promenne i */

int *pi; /* definice promenne typu pointer
         pi -- pointer na promennou typu int
         *pi -- promenna typu int */

pi = &i; /* do pi se ulozi adresa promenne i */

int b; /* promenna typu int */

b = *pi; /* do promenne b se ulozi obsah adresy
         ulozene v ukazeteli pi */

```

Ukazatele (pointery), proměnné a jejich hodnoty

- Proměnné jsou názvy adres, kde jsou uloženy hodnoty příslušného typu.
- Kompilátor pracuje přímo s adresami.
Přestože se v případě kompilace zpravidla jedná o adresy relativní.
- 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.
- Operátor `&` vrací adresu, kde je uložena hodnota proměnné.

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 style="text-align: center;"><i>Pointer je pouze a 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 0 nebo <code>NULL</code>. <ul style="list-style-type: none"> <i>Např. <code>int *i = NULL;</code></i> 					
Jan Faigl, 2020	BOB36PRP – Přednáška 05: 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> <ul style="list-style-type: none"> <i>Přístup k proměnným prostředí funkci <code>getenv()</code> z knihovny <code><stdlib.h></code>.</i> <i>lec05/main_env.c</i> Rozšířená funkce o specifické parametry Mac OS X <pre>int main(int argc, char **argv, char **envp, char **apple);</pre> 					
Jan Faigl, 2020	BOB36PRP – Přednáška 05: 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, 2020	BOB36PRP – Přednáška 05: 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 inicializované na hodnotu předávanou funkci. <ul style="list-style-type: none"> <i>Více o volání funkcí a paměti v 6. 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 tak realizujeme volání odkazem. 					
Jan Faigl, 2020	BOB36PRP – Přednáška 05: 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 „čtete“ 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, 2020	BOB36PRP – Přednáška 05: 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> 2 3 int main(int argc, char *argv[]) 4 { 5 int ret = 0; 6 7 printf(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 \$? ./demo-stdout 2>stderr Program has been called as ./demo-stdout 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</pre> 					
Jan Faigl, 2020	BOB36PRP – Přednáška 05: 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>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);</pre> Výstup <pre>Before call a: 10 b: A After call a: 10 b: B</pre> 					
Jan Faigl, 2020	BOB36PRP – Přednáška 05: 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> 2 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 > 0 ? 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 vypsat příkazem <code>echo</code>. <code>>/dev/null</code> přeměruje standardní výstup do <code>/dev/null</code>. 					
Jan Faigl, 2020	BOB36PRP – Přednáška 05: Pole a ukazatele				28 / 53

Pole	Ukazatele	Funkce a předávání parametrů	Vstup a výstup programu	Ukazatele a pole	Textové řetězce
<h2>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 uloženi 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> prekladačem <code>clang</code></i> 					
Jan Faigl, 2020	BOB36PRP – Přednáška 05: Pole a ukazatele				32 / 53

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

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);

```

lec05/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 první prvek.

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 33 / 53

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

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.

```

lec05/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, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 34 / 53

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

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.
- Přřazení `p = a` je legitimní. *Kompilátor zajistí přřazení adresy prvního prvku do ukazatele.*
- Přístup ke druhému prvku lze `a[1]` nebo `p[1]`.
- Oběma přístupy 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, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 35 / 53

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

Ukazatelová (pointerová) aritmetika

- S ukazately (pointery) lze provádět aritmetické operace `+` a `-`, tj. 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 `lec05/pointers_and_array.c`

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 36 / 53

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

Příklad ukazatele a pole

```

1 int a[] = { 1, 2, 3, 4 };
2 int b[] = { [3] = 10, [1] = 1, [2] = 5, [0] = 0 };
3 //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 }
8
9 int *p = a; //you can use *p = &a[0], but not *p =
&a
10 a[2] = 99;
11
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 }

```

lec05/array_pointer.c

```

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

```

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 37 / 53

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

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>
2
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(array, sizeof(array)/sizeof(int)); // pocet prvku
16     return 0;
17 }

```

lec05/fce_pointer.c

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

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 38 / 53

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

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])
2 {
3     printf("Size of the array a is %lu\n", sizeof(a));
4     for (int i = 0; i < n; ++i) {
5         printf("array[%i]=%i\n", i, a[i]);
6     }
7 }
8
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 }

```

lec05/fce_vla.c

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 39 / 53

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

Vícerozměrná pole

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

```

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

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

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

```

lec05/matrix.c

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 40 / 53

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

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.

```

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

```

lec05/matrix.c

- Dvouzměrné 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, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 41 / 53

Pole a vícerozměrná pole jako parametr funkce

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

```
int (p)[3] = m; // pointer to array of int
printf("Size of p: %lu\n", sizeof(p));
printf("Size of *p: %lu\n", sizeof(*p)); // 3 * sizeof(int) = 12
```

Funkci nelze deklarovat s argumentem typu `int []`, 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[rows][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, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 42 / 53

Textový řetězec

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

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

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*`.

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

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.

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 46 / 53

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. `scanf()`.

Více viz `man strcpy, strcmp, strtol, strtod, scanf`.

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 49 / 53

Inicializace pole

- Při definici můžeme hodnoty prvků pole inicializovat postupně nebo indexovaně. 2D pole jsou inicializována po řádcích.

Při částečné inicializaci jsou ostatní prvky nastaveny na 0.

```
#define ROWS 3
#define COLS 3
void print(int rows, int cols, int m[rows][cols])
{
    for (int r = 0; r < rows; ++r) {
        for (int c = 0; c < cols; ++c) {
            printf("%4i", m[r][c]);
        }
        printf("\n");
    }
}

int m0[ROWS][COLS];
int m1[ROWS][COLS] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int m2[ROWS][COLS] = { 1, 2, 3 };
int m3[ROWS][COLS] =
{ [0][0] = 1, [1][1] = 2, [2][2] = 3 };

print(ROWS, COLS, m0);
print(ROWS, COLS, m1);
print(ROWS, COLS, m2);
print(ROWS, COLS, m3);
```

lec05/array-inits.c

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 43 / 53

Načítání textových řetězců

- Správnost alokace vstupních argumentů je zajištěna při spuštění.

```
int main(int argc, char *argv[])
{
    // Načtení textového řetězce funkcí scanf().
    // Použitím %s může dojít k přepsu paměti.

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

    // Načtení maximálně 4 znaků zajistíme řídicím řetězcem "%4s".

    char str0[4] = "PRP";
    char str1[5];
    ...
    scanf("%4s", str1);
    printf("You entered string '%s'\n", str1);
    printf("String str0 = '%s'\n", str0);
```

Příklad výstupu programu:

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

lec05/str_scanf-bad.c

Příklad výstupu programu:

```
String str0 = 'PRP'
Enter 4 chars: 1234567
You entered string '1234'
String str0 = 'PRP'
```

lec05/str_scanf-limit.c

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 47 / 53

Část II

Část 2 – Zadání 4. domácího úkolu (HW04)

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 50 / 53

Řetězcové literály

- Formát – posloupnost znaků a řídicích znaků (escape sequences) uzavřená v uvozovkách.
 - Řetězcová konstanta s koncem řádku `"Řetězcová konstanta s koncem řádku\n"`
 - Řetězcové konstanty oddělené oddělovači (white spaces) se sloučí do jediné, např. `"Řetězcová konstanta" " s koncem řádku\n"` se sloučí do `"Řetězcová konstanta s koncem řádku\n"`.
- Typ
 - Řetězcová konstanta je uložena v poli typu `char` a zakončená 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, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 45 / 53

Zjištění délky textového řetězce

- Textový řetězec v C je pole (`char []`) nebo ukazatel (`char*`) odkazující na část paměti, kde je uložena příslušná posloupnost znaků.
- Textový řetězec je zakončen znakem `'\0'`.
- Délku textového řetězce lze zjistit sekvenčním procházením znak po znaku až k `'\0'`.

```
int getLength(char *str)
{
    int ret = 0;
    while (str && (*str++) != '\0') {
        ret += 1;
    }
    return ret;
}

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

lec05/string_length.c

Nebo jen `while (*str++) ret += 1;`

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 48 / 53

Zadání 4. domácího úkolu HW04

Téma: Prvočíselný rozklad

Povinné zadání: 2b; Volitelné zadání: 3b; 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/b0b36prp/hw/hw04>
 - Načtení posloupnosti kladných celých čísel (menších než 10^6) zakončených nulou a jejich rozklad na prvočinitele.
 - Volitelné zadání rozšiřuje rozsah hodnot vstupní čísel až do 10^8 (celá čísla v rozsahu 64-bitů). S ohledem na výpočetní náročnost řešení vyžaduje sofistikovanější přístup výpočtu s využitím techniky *Eratothenova síta*.
 - Bonusové zadání dále úlohu rozšiřuje zpracování čísel s až 100 ciframi. Řešení vyžaduje implementaci vlastní reprezentace velkých celých čísel spolu s operacemi celočíselného dělení se zbytkem.
- Termín odevzdání:** 30.10.2021, 23:59:59 PDT.
- Bonusová úloha:** 08.01.2022, 23:59:59 PST.

PDT – Pacific Daylight Time
PST – Pacific Standard Time

Jan Faigl, 2020 BOB36PRP – Přednáška 05: Pole a ukazatele 51 / 53

Shrnutí přednášky

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**