B0B99PRPA – Procedurální programování

Pole, ukazatele

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Pole a ukazatele
  - Pole
  - Ukazatele
  - Předávání parametrů funkcím
  - Pole a ukazatele
  - Textové řetězce
- Část 2 – Textové řetězce
- Část 3 – Zadání 6. domácího úkolu

Část I

Pole a ukazatele

I. Pole a ukazatele

Motivace

- výpočet průměrné teploty z hodnot naměřených ve všedních dnech

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int t1, t2, t3, t4, t5, prumer;
6    printf("zadejte teploty\n");
7    scanf("%d", &t1);
8    scanf("%d", &t2);
9    scanf("%d", &t3);
10   scanf("%d", &t4);
11   scanf("%d", &t5);
12   prumer = (t1+t2+t3+t4+t5+t5)/5;
13   printf("%d\n", prumer);
14   return 0;
15 }
```

- řešení je těžkopádné
- bylo by ale ještě horší, kdyby vstupními daty byly teploty za měsíc nebo za celý rok
- vhodnější je využít stukturovaný datový typ – pole

Pole

- Datová struktura pro uložení více hodnot stejného typu

  Typicky reprezentace posloupností

- Hodnoty uloženy v souvislém bloku paměti

  Velikost bloku je určena počtem prvků a velikostí datového typu

- Proměnná typu pole reprezentuje adresu, kde blok začíná

  Definicí proměnné dochází k alokaci popřebné paměti

- Prvky pole mají stejnou velikost a známou relativní adresu

  Relativní adresa – posun vůči adrese prvního prvku

- Velikost pole statické délky nelze měnit
  - Ve starších standardech ($< C99$) je třeba, aby byla velikost pole známa v době překladu

Definice pole

- Definicí proměnné dojde k alokaci potřebné paměti

  K inicializaci ale dojde jen za určitých okolností

```
typ identifikátor []
```

- `[]` slouží také pro přístup k prvku pole (adresaci)

```
identifikátor [index]
```

Příklad

```
int array[10];

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

Vlastnosti pole

- Pole je posloupnost prvků stejného typu
- K prvkům pole se přistupuje pořadovým číslem (indexem) prvku

  Index je celé nezáporné číslo.
- Index prvního prvku je vždy roven 0
- C nekontroluje za běhu programu, zda je index platný!

  Je tedy možné adresovat paměť, která nebyla alokována.
- Prvky pole mohou být proměnné libovolného typu
- Pole může být jednorozměrné nebo vícerozměrné

Pole – příklad 1

- 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;
```

- Definice pole a tisk hodnot prvků

```
int array[5];

printf("Size of array: %lu\n", sizeof(array));

for (int i = 0; i < 5; ++i) {
  printf("Item[%i] = %i\n", i, array[i]);
}
```

Pole – příklad 2

- Motivační příklad pomocí pole

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int teploty[7], prumer = 0;
6
7    for (int i = 0; i < 7; i++)
8    {
9      scanf("%d", &teploty[i]);
10     prumer += teploty[i];
11   }
12   printf("%d\n", prumer);
13   return 0;
14 }
```

Pole – příklad 3

- Inicializace pole

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

Pole variabilní délky

- Standard C99 umožňuje určit velikost pole za běhu programu

  Ve starších standardech bylo třeba znát velikost v době překladu

- Nejedná se o možnost změny velikosti pole za běhu programu
- Pole variabilní délky nelze v definici inicializovat

Příklad

```
printf("zadej velikost pole: ");
scanf("%d", &n);

int pole[n];

for (int i = 0; i < n; i++) {
  pole[i] = i * i;
}
```

I. Pole a ukazatele

Adresování

- Každá proměnná v paměti má tři hlavní charakteristiky
  - Jméno (identifikátor)
  - Obsah (hodnota)
  - Adresa – jednoznačná identifikace obsazeného místa v paměti

    Vyjímkou jsou proměnné v paměťové třídě register.
- **Přímé adresování** – přístup k obsahu na adrese prostřednictvím jména proměnné
- **Nepřímé adresování** – přístup prostřednictvím jiné proměnné, která obsahuje adresu – tzv. reference
  - nepřímé adresování umožňuje funkcím přistupovat k paměti alokované mimo tělo funkce
  - nepřímé adresování umožňuje funkcím mít vic než jednu návratovou hodnotu (viz `scanf`)

Ukazatel (pointer)

- Ukazatel (pointer) je proměnná, jejíž hodnota je adresa v paměti
- Pointer může být inicializován adresou jiné proměnné

  Odkazuje na oblast paměti, kde je uložena hodnota proměnné.
- Ukazatel má typ proměnné, na kterou může ukazovat

  Důležité pro ukazatelovou aritmetiku
- Ukazatel může odkazovat na
  - proměnné všech typů – primitivní i strukturované
  - funkce
  - jiné 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`

Definice ukazatele

- Deklarace ukazatele

```
// ukazatel na promennou typu char
char *p1;
// ukazatel na promennou typu int
int *p2;
```

- Inicializace ukazatele

```
char a;
p1 = &a;  // p1 inicializovan adresou promenne a
int b;
p2 = &b;  // p2 inicializovan adresou promenne b
```

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
- Umožňuje číst a zapisovat hodnotu na adrese dané obsahem ukazatele

Příklad

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int a = 10;
6    int *p = &a;
7    printf("a = %i\n", a);
8    *p = 20;
9    printf("a = %i\n", a);
10
11   return 0;
12 }
```

Ukazatele, 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.
- Ukazatel je proměnná, ve které je uložena adresa. Na této adrese se pak nachází hodnota nějakého typu (např. int).
- Ukazatele realizují tzv. nepřímé adresování
- Dereferenční operátor přistupuje na proměnnou adresovanou hodnotou ukazatele
- Operátor & vrací adresu, kde je uložena hodnota proměnné

Ukazatele a kódovací styl

- Symbol * lze zapisovat u identifikátoru proměnné nebo typu
- Preferujeme zápis u proměnné, abychom předešli omylům

```
char* a, b, c;     // ukazatel je pouze a
char *a, *b, *c;   // vsechny promenne jsou ukazatele
```

- Ukazatel na ukazatel: `char **a;`
- Typ ukazatel: `char*`, `char**`
- Neplatná adresa má symbolické jmého `NULL`
  - makro preprocesoru
  - v C99 lze i 0
  - proměnné v C nejsou automaticky inicializovány a ukazatele tak mohou odkazovat na neplatnou paměť

I. Pole a ukazatele

Předávání parametrů funkcím

- V C jsou **parametry funkce předávány hodnotou**
- Parametry jsou lokální proměnné funkce (alokované na zásobníku), které jsou inicializované na hodnotu předávanou funkci

```
void fce(int a, char *b) { /*
a - lokalni promena typu int (na zasobniku)
b - lokalni promena typu ukazatel na promenou typu
char (hodnota je adresa, take na zasobniku) */}
```

- Lokální zmena hodnoty promenné neovlivnuje hodnotu promenné vne funkce
- Pri predání ukazatele, však máme prístup na adresu puvodní promenné, kterou mužeme menit
- Ukazatelem tak realizujeme volání odkazem

Předávání parametrů funkcím – 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("Pred volanim a: %d b: %c\n", a, b);
fce(a, \&b);
printf("Po volani a: %d b: %c\n", a, b);
```

Funkce `main` a její tvary

- Základní tvar funkce main

```
int main(int argc, char *argv[]) { ... }
```

- Alternativne pak také

```
int main(int argc, char **argv) { ... }
```

- Argumenty funkce nejsou nutné

```
int main(void) { ... }
```

- Rozšírená funkce o nastavení promenných prostredí

Unix a MS Windows

```
int main(int argc, char **argv, char **envp) { ... }
```

Prístup k promenným prostredí funkcí getenv() z knihovny ¡stdlib.h¿.

- Rozšírená funkce o specifické parametry Mac OS X

```
int main(int argc, char **argv, char **envp, char **
apple);
```

Argumenty funkce `main`

- Základní tvar funkce `main`

  ```
  int main(int argc, char *argv[]) { ... }
  ```

- `argc` – obsahuje počet argumentů programu

  Včetne jména spouštěného programu.

  - Argumenty jsou textové řetězce oddělené mezerou (bílým znakem).
- `argv` – pole ukazatelů na hodnoty typu `char`
  - Pole `argv` má velikost (pocet prvku) daný hodnotou `argc`
  - Každý prvek pole `argv[i]` obsahuje adresu, kde je uložen textový řetězec argumentů (tj. typ `char*`)
  - Textový řetězec (argument) je posloupnost znaku (typ `char`) zakoncený znakem `'\0'`
  - Alokace paměti pro uložení argumentů (textových retezců) je provedena pri spuštení programu

Argumenty programu

- Programu můžeme při spuštění předat argumenty příkazové rádky

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[])
4  {
5    printf("Pocet argumentu %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 }
```

- Voláním `return` ve funkci `main()` vracíme z programu návratovou hodnotu, se kterou můžeme dále pracovat

```
$ ./a.out >/dev/null; echo $?
$ 1
$ ./a.out first >/dev/null; echo $?
$ 0
```

Interakce programu s uživatelem

- Při spuštení programu lze předat parametry (textové řetězce)
- Při ukončení programu lze předat návratovou hodnotu

  Konvence: správné ukončení 0, jinak chybový kód.
- Při běhu programu lze číst ze standardního vstupu a zapisovat na standardní výstup
- Při spuštení programu lze vstup i výstup přesměrovat z/do souboru
- Každý terminálový program má standardní vstup (`stdin`) a výstup (`stdout`) a dále pak standardní chybový výstup (`stderr`), které lze v shellu presměrovat

  ```
  $ ./a.out <stdin.txt >stdout.txt 2>stderr.txt
  ```
- Pro práci s chybovým výstupem lze využít funkci `fprintf`
  - Prvním argumentem soubor, jinak stejná syntaxe jako `printf`
  - Soubory `stdout`, `stdin` a `stderr` jsou definovány v `<stdio.h>`

Příklad programu s výstupen na stderr

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[])
4  {
5    int ret = 0;
6
7    fprintf(stdout, "Jmeno programu je %s\n", argv[0]);
8
9    if (argc > 1)
10   {
11     fprintf(stdout, "Prvni argument je %s\n", argv[1]);
12   }
13   else
14   {
15     fprintf(stdout, "Prvni argument nebyl zadan.\n");
16     fprintf(stderr, "Musi byt zadan alespon jeden argument!\n");
17     ret = -1;
18   }
19   return ret;
20 }
```

I. Pole a ukazatele

Pole a ukazatele

- Ukazatel ukazuje na vyhrazenou část paměti proměnné

```
int *p;      // ukazatel (adresa)
sizeof(p);   // velikost promenne
```

- Pole je označení souvislého bloku paměti

```
int a[10];   // souvisly blok pameti
sizeof(a);   // 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 paměťovým místem.
  - Ukazatel obsahuje adresu, na které je příslušná hodnota (nepřímé adresování)
- Při předávání pole jako parametru funkce je předáváno pole jako ukazatel.

Pole a ukazatele – příklad

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int *p;   // ukazatel na integer
6      int (*ptr)[5];  // ukazatel na pole peti prvku int.
7      int arr[5];
8
9      p = arr;    // ukazatel na prvni prvek pole
10     ptr = &arr;   // ukazatel na cele pole
11
12     printf("p = %p, ptr = %p\n", p, ptr);
13
14     p++;
15     ptr++;
16
17     printf("p = %p, ptr = %p\n", p, ptr);
18
19     return 0;
20 }
```

Funkce s polem jako argumentem

```
void fce(int A[]) {
  int B[] = { 2, 4, 6 };
  printf("sizeof(A) = %lu -- sizeof(B) = %lu\n", sizeof(B),
     sizeof(B));
  for (int i = 0; i < 3; ++i) {
    printf("array[%i]=%i B[%i]=%i\n", i, A[i], i, B[i]);
  }
}
...
int array[] = { 1, 2, 3 };
fce(array);
```

- Po překladu (`gcc -std=c99`) na **amd64**
  - `sizeof(A)` vrátí velikost 8 bajtů (64-bitová adresa)
  - `sizeof(B)` vrátí velikost 12 bajtů (3×4 bajty – `int`)
- Pole se funkcím predává jako ukazatel na první prvek

I. Pole a ukazatele

Řetězcové literály

- Formát – posloupnost znaků a řídicích znaků uzavřená v uvozovkách

  `"Sud kulatý, rys tu pije!\n"`

  - Řetězcové konstanty oddělené oddělovači (white spaces) se sloučí

    `"Tu je kára," " ten to ryje!\n"`

    se sloučí do

    `"Tu je kára, ten to ryje!\n"`
- Typ – pole typu char zakončené znakem `'\0'`
  - Pole pro uložení řetězce musí být o jeden prvek větší než délka samotn0ho řetězcového literálu

    ```
    char text[10]; // text o delce 9 znaku
    ```

Textový řetězec

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

```
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);
```

- 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 OS)
  - Textový retezec musí být zakončen znakem `'\0'`

Načtení textového řetězce funkcí `scanf()`

- Použitím `%s` může dojít k přepisu paměti

```
char str0[4] = "PRP";
char str1[5];
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);
```

Práce s textovými řetězci

- Základní operace jsou definovány v knihovne `<string.h>`

```
int strlen (char *s);
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 retezcu:

```
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);
```

Část II

Zadání domácího úkolu

Zadání 4. domácího úkolu (HW04)

Téma: Maticové počty

- **Motivace:** Práce s dvourozměrným polem
- **Cíl:** Práce s polem variabilní délky a předávání ukazatelů
- **Zadání:** `https://cw.fel.cvut.cz/wiki/courses/b0b99prpa/hw/hw04`
  - Načtení vstupních hodnot dvou matic a znaku operace ('*' – násobení).
  - Volitelné zadání rozširuje úlohu o další operace s maticemi sčítání ('+') a odčítání ('-'), které mohou být zapsány ve výrazu.
- **Termín odevzdání:** **17.11.2018, 23:59:59**

Shrnutí přednášky

Diskutovaná témata

- Pole
- Ukazatele
- Předávání parametrů funkcím
- Pole a ukazatele, ukazatelová aritmetika
- Řetězce

- Příště: ukazatele, práce s pamětí, ladění

Diskutovaná témata

- Pole
- Ukazatele
- Předávání parametrů funkcím
- Pole a ukazatele, ukazatelová aritmetika
- Řetězce

- Příště: ukazatele, práce s pamětí, ladění