Polia v jazyku C

Pavol Zajac
(náhradný materiál k prednáške 4/2020)

Pozn.: Tento materiál je doplnkový, sústreďuje sa na hlavné body, ktoré by boli prebrané na prednáške. Opäť pripomínam, že základom je poctivé štúdium z učebníc jazyka C.

- Polia (angl. *array*) v jazyku C umožňujú reprezentovať viac premenných naraz:

príklad: reprezentácia bodu v 3d priestore:

```
    int x, y, z;        //tri nezavisle premenne
    int suradnice[3];   //“trojica premennych so spolocnym menom”

//indexovanie sluzi ako pristup ku konkretnej premennej
// namiesto iba mena

    suradnice[0] = x;
    y = suradnice[1];

//vyhodou je, ze indexy mozu byt aj premenne
//  (lubovolny vyraz konvertovatelny na int):

    for (i = 0; i < 3; i++)
        suradnice[i] = 0;
```

Základné info:

- Polia v jazyku C môžu byť podobne ako premenné:
  - globálne: alokované mimo funkcií, nastavia sa pred spustením programu
  - lokálne statické: s použitím kľúčového slova `static`, nastavia sa pred spustením programu, nemenia sa medzi volaniami funkcie a vidí ich len funkcia, kde sú definované (*ak ich nesprístupnite inak, vrátením adresy začiatku poľa...*)
  - lokálne automatické: vytvoria sa vždy, keď sa spustí funkcia, keď funkcia skončí, pamäť sa uvoľní. *Dôležité! Ak by ste „vrátili“ lokálne automatické pole, jedná sa o neplatnú (uvoľnenú) pamäť.*
- *Existujú aj dynamické polia, tým sa budeme venovať neskôr.*
- *Automatické polia, na rozdiel od statických a globálnych, majú obmedzenú veľkosť (závisí na nastaveniach prekladača). Ak vám záhadne padá program, skontrolujte si, či nealokujete príliš veľké lokálne automatické polia.*

- Polia v jazyku C teda umožňujú riešiť niektoré úlohy, ktoré sa typicky v Pythone riešia pomocou listov

Dôležité rozdiely medzi C poľami a listami:

1. pole je iba úložisko dát rovnakého typu

Keď vytvárame pole, musíme špecifikovať typ prvku, všetky majú rovnaký typ

```
int suradnice[3];   //celociselne pole

double suradnice_f[3]; //pole desatinnych cisel
```

Prečo je to dôležité? V pozadí prekladač vyčlení súvislú pamäť, do ktorej sa musia zmestiť všetky dáta poľa. Keďže prvky sú rovnakého typu, prekladač hneď vie, koľko pamäte treba.

2. Na rozdiel od Pythonu nemáme vhodné operátory na prácu s poľom, napr. nemáme operátor *len*, nemáme *append*, *etc.*, polia nevieme zreťaziť (operátor +), nevieme robiť podpolia (*slicing*), iba pristupovať ku konkrétnym pozíciám.

Koľko pole zaberá v pamäti zistíme, na mieste kde je pole alokované, nasledovne:

```
sizeof(suradnice)

//pocet prvkov je teda:
sizeof(suradnice) / sizeof(suradnice[0])
```

POZOR! Platí to len pre globálne polia, a polia alokované priamo vo funkcii, kde tento spôsob použijeme. Ak pole použijeme v nejakej inej funkcii, ***dĺžku poľa nie je možné zistiť****. (viď ďalej)*

3. Na rozdiel od Pythonu, pole je pevnej dĺžky. Koľko miesta alokujeme pri vytvorení poľa, toľko má počas celej existencie (výnimka: dynamické polia, budeme sa venovať neskôr)

Je výhodné použiť makrá (pomenované konštanty cez define) na nastavenie dĺžky poľa. Výhodné to je, ak potrebujeme globálne túto dĺžku zmeniť (napr. pri testovaní zmenšiť/zväčšiť):

```
#define LEN 100

//vytvorime pole, a naplnime ho cislami 1 az LEN
int pole[LEN];
for (i = 0; i < LEN; i++)
     pole[i] = i+1;
```

Pozn.: nemusíte využívať celú veľkosť alokovaného poľa. Nemôžete však využívať viac prvkov, ako ste alokovali.

4. Pobodne ako v Pythone, polia sa dajú naplniť prvkami aj explicitne (vymenovaním). Syntax na vymenovanie je uviesť prvky oddelené čiarkou v krútených zátvorkách (viď príklad). Prvky musia byť všetky rovnakého typu, ako je typ poľa (resp. sa skonvertujú).

Ak uvádzame prvky explicitne, nemusíme nastaviť dĺžku poľa. Prekladač ju zistí podľa počtu zadaných prvkov:

```
//vyuzijeme pole vo funkcii pocet dni:
//PRE: 1 <= m && m <= 12
int pocet_dni(int m)
{
     int pocty[] = {31,28,31,30,31,30,31,31,30,31,30,31};

     return pocty[m-1];
}
```

Deklarácia s počtom prvkov a vymenovaním sa dá kombinovať:

a) ak sa uvedie iba dĺžka, obsah poľa je definovaný iba pri globálnych a statických poliach (je nulový), lokálne automatické polia budú obsahovať „náhodné dáta“ (závisí na prekladači)

b) ak sa uvedie inicializácia a dĺžka (aspoň taká ako počet inicializovaných prvkov), zvyšok sa nastaví na nulu:

```
int pole[100] = {1}; //pole[0] == 1, ostatne: pole[i] == 0
```

5. Pole sa dá indexovať ľubovoľným výrazom, ktorý sa dá vyhodnotiť na int, ale na rozdiel od Pythonu sa nekontrolujú hranice poľa! (*v skutočnosti sa vypočíta nejaká adresa v pamäti a k tej sa pristúpi, bez ohľadu na to, či je vyčlenená pre pole, alebo nie*).
   POZOR: na rozdiel od Pythonu, záporné indexy sú neplatné.

   Vždy sa teda indexuje od 0 po LEN-1

```
#define LEN 100

//vytvorime pole, a naplnime ho cislami 1 az LEN
int pole[LEN];

//prvy prvok pola nastavíme na 5:
pole[0] = 5;

//posledny prvok pola nastavíme na hodnotu druhého prvku
pole[LEN - 1] = pole[1];

//nastavi kazdy druhy prvok na nulu.
// POZOR: vsetky prvky na parnych indexoch zostanu povodne
for (i = 1; i < LEN; i += 2)
    pole[i] = 0;
```

Špecifické poznámky týkajúce sa globálnych a lokálnych polí:

- globálne a statické polia môžu mať veľkosť definovanú len konštantou (známou pri preklade). Ak nepoznáte vopred veľkosť poľa, musíte alokovať toľko, aby bolo dosť miesta v najhoršom prípade. Ak alokujete menej ako potrebujete, a zapíšete/čítate mimo hranice poľa, jedná sa o závažnú chybu s bezpečnostnými dôsledkami.
- nie je vhodné spoliehať sa (len) na globálne polia: keďže s nimi môže manipulovať každá funkcia, zvyšuje sa previazanosť kódu a náročnosť hľadania chýb. Tiež existuje len jedna kópia globálneho (aj statického) poľa v celom programe! (viď funkcie a polia)
- lokálne automatické polia môžu mať veľkosť danú premennou (*iba v novom štandarde C, nemusí fungovať v starších Visual Studiach, resp. ak má súbor príponu cpp*), táto musí mať známu hodnotu pred vytvorením poľa /najčastejšie ako parameter funkcie/:

```
int pocet_dni(int n)
{
    int docasne[n];

    nacitaj(docasne, n);

    return sucet(docasne, n);
}
```

Funkcie a polia:

- na efektívnu prácu s poliami je potrebné si pripraviť vhodné funkcie: neexistuje štandardný výpis polí, načítanie, map /Python list comprehension/
- je dobré poznať niektoré knižničné funkcie: memcmp, memcpy, ale keďže je potrebné poznať smerníky, budeme sa zaoberať nimi neskôr
- polia sa do funkcií **odovzdávajú odkazom**. Čo to znamená: funkcia nepracuje s kópiou poľa /ako pri individuálnych hodnotách premenných/, ale priamo s pôvodným poľom /*funkcia získa jeho adresu do lokálneho smerníka, budeme hovoriť neskôr*/.

- Keďže polia sa odovzdávajú odkazom, funkcie môžu meniť hodnoty v poli. Ak tomu chceme zabrániť, musíme parameter pole deklarovať ako `const`. Ak neuvedieme const, volaná funkcia môže z poľa hodnoty čítať, ale aj do poľa ľubovoľne zapisovať. Taktiež môže preposlať odkaz na pole ďalším funkciám.
- Polia z funkcií sa nedajú vrátiť. *Dá sa vrátiť odkaz na začiatok poľa /smerníky, neskôr ;)/, ale ak je pole lokálne, tak tento odkaz po skončení funkcie je neplatný! Ak je pole statické a vráti sa odkaz naň, tak sa manipuluje s pôvodným poľom vo funkcie, nie s kópiou.*
- Funkcie nepoznajú veľkosti polí (dostanú iba odkaz na začiatok). Preto programátor by mal explicitne pridať aj veľkosť poľa ako parameter (ak sa nejako inak nedá identifikovať posledný prvok, ako pri reťazcoch).

```
//priklad funkcii, ktore pracuje s polom
// pri parametri pole mozeme uviest pevnu (konstantnu) velkost
//  vhodnejsie je vsak pouzit nespecifikovanu velkost []
//  a pridat parameter s konkretnou velkostou,
//    ktoru nastavi volajuca funkcia

//scita hodnoty pola, hodnoty sa nemenia
int sucet(const int pole[], int n)
{
     int i, sum = 0;

     for (i = 0; i < n; i++)
         sum += pole[i];

     return sum;
}

//nastavi nahodne hodnoty mensie ako m do pola, povodne sa prepisu
// funkcia vracia cisla v ramci zmeneneho pola,
// cize nemusi vratit nic ine, je void
void random(int pole[], int n, int m)
{
     int i;

     for (i = 0; i < n; i++)
         pole[i] = rand() % m;
}

//priklad pouzitia:

int random_sum(int n)
{
     //docasne vyclenime pamat, nema definovany obsah
     int docasne[n];

     //zavolame funkciu, ktora pole naplni
     //pri volani funkcie sa [] nepouzivaju, iba nazov pola
     random( docasne, n, 6 );

     //spocitame, co sme dostali
     //po returne docasne hodnoty zaniknu
     return sucet( docasne, n );
}
```

Pozn.: Keď už máme funkcie hotové, môžeme ich používať v rámci ľubovoľných iných funkcií, a môžeme im ako parameter dávať odkazy na ľubovoľné polia (aj globálne, alebo tie, čo funkcia dostala ako parameter:

```
void vypis_so_suctom(int pole[], int n)
{
     int i, sum;

     sum = sucet(pole, n);

     //vypis prvkov mohla byt aj zvlast funkcia:
     for (i = 0; i < n; i++)
         printf("%i\n", pole[i]);

     printf("Sucet: %i\n", sum);
}

#define LEN 100
int global[LEN];

int main()
{
    //na jednom riadku vieme specifikovat pole
    // aj premenne rovnakeho typu
    int local[LEN], j, suml, sumg;

    random(global, LEN, 6);

    //pozn. nemusime vzdy zadat celu dlzku...
    // potom bude funkcia robit len s prislusnou castou pola
    random(local, 10, 6);

    suml = sucet(local, 10);
    for (j = 0; j < LEN-10; j++)
    {
// ak chceme adresovat podpole, pouziva sa smernikova aritmetika
// global + j vrati cast pola od j-tej pozicie
// pre funkciu sa to javi ako standardne pole,
//   ale programator musi dat pozor na hranice indexov!
// viac nabuduce :)
        sumg = sucet(global+j, 10);

        if (suml > sumg)
            printf("Local > global at: %i", j);
    }

    return 0;
}
```