Viacrozmerné polia v jazyku C

Náhradná Prednáška 10, PROG-2, 2020
Pavol Zajac

Pozn.: Táto téma vyžaduje, aby ste rozumeli dobre pointerom a poliam. Na túto tému som našiel dobrú sériu prednášok
*https://www.youtube.com/playlist?list=PL2_aWCzGMAwLZp6LMUKI3cc7pgGsasm2_*
K dnešnej téme odporúčam do pozornosti hlavne videá týkajúce sa 2D a viac-rozmerných polí, ale oplatí sa pozrieť celá séria prednášok od začiatku.

Statické viacrozmerné polia

V jazyku C, pole je postupnosť prvkov rovnakého typu uložených v pamäti za sebou. Napríklad:

```
int pole[5] = {1,2,3};
```

rezervuje 5 za sebou idúcich prvkov v pamäti veľkosti `sizeof(int)` a naplní ich postupne binárne kódovanými hodnotami 1, 2, 3, 0, 0.

Keď v kóde použijeme identifikátor `pole`, vyhodnotí sa jeho hodnota ako **pamäťová adresa** prvého prvku (typu `int*`). Vďaka tomu vieme pristupovať k prvkom poľa ako cez operátor indexovania, tak aj operátor sprístupnenia. Taktiež vieme ľahko odkazom odoslať pole ako parameter funkcie.

Prvky poľa nemusia byť len základného typu (`int`, `double`, ...), ale v poli môžu byť uložené adresy (pole smerníkov), štruktúry, alebo prvkami poľa môžu byť iné polia. Ak prvkami poľa sú iné polia, hovoríme o (statickom) viacrozmernom poli. Príklad:

```
double sustava[3][4] = {
   {-1, 1, 0,0.5},
   { 0, 1, 1,2.5},
   { 2, 1,-1, -1}
};
```

Príkazom vytvoríme v pamäti trojprvkové pole `sustava`, ktorého prvky sú štvorprvkové polia `double` čísel. V príklade prvky poľa hneď aj inicializujeme hodnotami, používa sa podobná syntax ako pri poliach štruktúr.

Počet rozmerov môže byť prakticky ľubovoľný, ale v každom rozmere je potrebné špecifikovať rozsah indexov (koľkoprvkové pole je v danom rozmere). Prvý rozmer je možné vynechať, ak je pole inicializované (prekladač doplní podľa počtu prvkov), ale ostatné rozmery sa vynechať nedajú, lebo prekladač by nevedel ako správne ukladať prvky do pamäte a preložiť indexy na pamäťové adresy.

Príklad, 4-rozmerná Karnaugh-ova mapa:

```
int map[2][2][2][2] = {0,1,0,1, 0,0,0,0, 1,0,1,1, 1,1,0,0};
```

Pozn.: Pri inicializácii podobne ako v príklade môžeme použiť priamy zoznam hodnôt bez vnútorných zátvoriek, ale vo väčšine prípadov je používanie vnorených zátvoriek a oddelovačov prehľadnejšie (príklad `sustava`).

Viacrozmerné pole môže zaberať veľké množstvo pamäte (súčin rozmerov v každej dimenzii, krát veľkosť základného prvku), načo je potrebné dávať pozor, hlavne ak chceme vytvoriť lokálne premenné (pri veľkých viacrozmerných poliach je preto odporúčané použiť modifikátor `static`).

Sprístupnenie prvkov viacrozmerného poľa

Ak sa chceme dostať ku konkrétnemu prvku viacrozmerného poľa, je potrebné nastaviť všetky indexy. Opäť platí, že kontrola hraníc poľa je na programátorovi, v každom rozmere platný index je od 0 po MAX-1, kde MAX je rozsah počtu prvkov v danom rozmere.

Príklad, vypísanie obsahu poľa `sustava` v tvare troch rovníc o troch neznámych, štvrtý stĺpec sa berie ako pravá strana:

```
for (int i = 0; i < 3; i++)
{
     for (int j = 0; j < 3; j++)
     {
          printf("%4.1lf ", sustava[i][j]);
     }
     printf("| %4.1lf\n", sustava[i][3]);
}
/* Vysledok:
-1.0  1.0  0.0 |  0.5
 0.0  1.0  1.0 |  2.5
 2.0  1.0 -1.0 | -1.0
 */
```

Poznámka: V príklade používame prvý index (i) ako číslo riadku, a druhý index (j), ako číslo stĺpca. Dvojrozmerné polia sa často chápu ako matice, a používa sa táto konvencia. Programátor však indexom a rozmerom priraďuje význam v kóde podľa situácie, napr. v príklade je index 3 v druhom rozmere špeciálny (pravé strany sústavy). Keby sme chceli, môžeme pole vypísať transponovane (zamenené riadky/stĺpce) bez prehadzovania údajov v poli:

```
for (int i = 0; i < 4; i++)
{
     for (int j = 0; j < 3; j++)
     {
          //vnutorny index cyklu je teraz prvy rozmer
          // hranice indexov v rozmeroch musia sediet!
          printf("%4.1lf ", sustava[j][i]);
     }
     printf("\n");
}
/* Vysledok:
-1.0  0.0  2.0
 1.0  1.0  1.0
 0.0  1.0 -1.0
 0.5  2.5 -1.0
 */
```

Viacrozmerné polia a funkcie

Pole `sustava` je pre programátora logicky dvojrozmerné pole, reprezentujúce maticu sústavy 3 rovníc s troma neznámimi združenej spolu s pravými stranami, kde obsahom poľa sú 3x4 čísla typu `double`. Pre prekladač sa však jedná o rezervované trojprvkové pole, ktorého obsahom sú štvorprvkové polia s prvkami typu `double`.

V kóde výraz `sustava` sa preloží ako adresa prvého prvku poľa, ale nie je typu pointer na double, teda `double*`, ale je typu pointer na pole 4 double prvkov, t.j. typu `double (*)[4]`.

Rozdiel je v tom, že ak zvýšime pointer typu `double*` o 1 pointrovou aritmetikou, zvýši sa fyzicky adresa o sizeof(double). Naopak, ak zvýšime pointer typu `double (*)[4]` o 1, zvýši sa fyzicky adresa o `4*sizeof(double)`. To je ale presne to, čo potrebujeme:

`sustava[0]` je pole prvých 4 double prvkov, prvý riadok matice,
`sustava[1]` je pole nasledujúcich 4 double prvkov, druhý riadok matice,

Oba výrazy, `sustava[0]` aj `sustava[1]` sú už typu `double*` a odkazujú na prvý prvok v riadku matice.

Napokon k samotnému prvému prvku v prvom riadku sa dostaneme výrazom `sustava[0][0]`, atď.

Keďže prekladač potrebuje vedieť, koľko prvkov má preskočiť, keď sa zvýši prvý index o 1, musia byť rozsahy indexov v nasledujúcich rozmeroch pevne dané. Keď odovzdávame viacrozmerné polia do funkcie, musíme tieto rozmery uviesť:

```
void vypis_sustavy(double sustava[3][4]);
```

Pozn.: pri lepšiu čitateľnosť odporúčam použiť na rozsahy pomenované konštanty, napr.:

```
#define RIADKY 100
#define STLPCE  50
void vypis_maticu(double matica[RIADKY][STLPCE]);
```

Pozn.: aj keď maximálne rozmery môžu byť pevne dané, podobne ako pri jednorozmerných poliach nemusíme využiť celé pole na platné prvky. Príklad:

```
#define MAX_RIADKY 100
#define MAX_STLPCE 100

//prvy rozmer matice mozeme vynechat, ostatne nie
void vypis(double matica[][MAX_STLPCE], int riadky, int stlpce) {
     for (int r = 0; r < riadky; r++) {
         printf( "[%lf", matica[r][0] );
         for (int s = 1; s < stlpce; s++) {
            printf( ", %lf", matica[r][s] );
         }
         printf( "]\n" );
     }
}
```

TODO: Pozrite si ukážku použitia funkcie v príklade `p10-02.c`. Na rozdiel od výpisu posledného riadku, funkcia `vypis` sa nedá použiť na vypísanie iba posledného stĺpca matice v ukážke. Prečo?

Iný príklad viacrozmerného poľa je viacrozmerné pole znakov, kde jednotlivé riadky poľa môžeme používať ako reťazce. Žiaden z reťazcov v poli však nesmie prekročiť alokovanú kapacitu (musí sa zmestiť na príslušný riadok vrátane koncovej '\0'). Príklad:

```
//nacita slova do zadaneho pola slov, kazde je dlzky MAX_LEN
int read_words(char words[][31], int max_words) {
     int nwords = 0;
     while (nwords < max_words) {
          //tu words[nwords] je adresa typu char*
          //ukazuje na miesto, kde sa da ulozit MAX_LEN dlhy retazec
          if (scanf("%30s", words[nwords]) < 1)
               break;
          nwords++;
     }
     return nwords;
}
```

TODO: Pozrite si príklad `p10-03.c`. Je tam uvedené, aj spôsob ako môžete mať šírku slov predpísanú makrom (resp. dvomi), aby ste ju vedeli prípadne ľahko zväčšiť.

Alternatívy pre viacrozmerné polia

Syntax viacrozmerných polí v jazyku C je pomerne neflexibilná (vo vyšších programovacích jazykoch sú naozajstné viacrozmerné polia, aj s kontrolami indexov), ale veľké množstvo algoritmov vyžaduje viacrozmerné polia (matice, grafy, relácie, grafika, ...).

Viacrozmerné pole viete algoritmicky emulovať v jednorozmernom poli, prepočítavaním indexov. Príklad:

```
void print2d(int array[], int rows, int cols)
{
     for (int r = 0; r < rows; r++) {
       for (int s = 0; s < rows; s++) {
          printf("%3i ", array[r*cols + s]);
       }
       printf("\n");
     }
}
int data[] = {1,2,3,4,5,6,7,8,9,10,11,12};
print2d(data, 3, 4);
print2d(data, 4, 3);
print2d(data, 2, 2);
```

TODO: Vyskúšajte si, čo vypíše funkcia s rôznymi parametrami. Čo sa stane, ak bude `rows*cols` väčšie ako počet prvkov v poli?

Všimnite si vzorec vo výpočte: `index = r*cols + s`
Podobným spôsobom prekladač počíta pamäťové adresy pri štandardných viacrozmerných poliach: `r`-krát preskočíme počet stĺpcov, a potom sa posunieme ešte o `s` miest.

Matematická úloha: Ako by ste z indexu `index` spätne určili `r` a `s`?

Ďalšie alternatívy:

- vytvoríme si polia štruktúr, ktoré v sebe obsahujú polia
- vytvoríme pole smerníkov, smerníky ukazujú na polia (zväčša dynamické, alebo do väčšej súvislej pamäti)

Príklad:

```
struct _row    { int ncols; int cols[MAXCOLS]; };
struct _matrix { int nrows; struct _row rows[MAXROWS]; };

void printMatrix(struct _matrix M)
{
      for (int r = 0; r < M.nrows; r++) {
        for (int s = 0; s < M.rows[r].ncols; s++) {
           printf("%3i ", M.rows[r].cols[s]);
        }
        printf("\n");
      }
}
```

V príklade máme štruktúru _row (riadok matice), ktorá obsahuje nejaký počet platných stĺpcov (ncols) a obsah stĺpcov uložených v poli cols. Matica je reprezentovaná štruktúrou _matrix, ktorá obsahuje nejaký počet platných riadkov nrows, uložených v poli rows. Prístup k jednotlivým prvkom je syntakticky zložitejší:
`M.rows[r].cols[s]`: prečítaj z matice `M` premennú `rows`, t.j. adresu začiatku poľa riadkov, posuň sa o offset `r`, dostaneš objekt typu `_row`, prečítaj z neho adresu začiatku poľa stĺpcov `cols` (v danom riadku), a posunom o offset `s` sa dostaneš k želanému číslu.
Výhodou je, že celá matica so všetkými informáciami o nej je zabalená do jednej štruktúry. Je potrebné však dávať pozor na to, že štruktúry sa odovzdávajú funkciám kópiou (a teda sa nedajú modifikovať pôvodné objekty vo funkcii). Ak je MAXCOLS a MAXROWS veľké, prejaví sa to aj v rýchlosti volania funkcie `printMatrix`. Riešením je použiť odovzdanie matice odkazom (pozor na syntax so smerníkmi na štruktúry):

```
void printMatrix(const struct _matrix *pM)
{
      for (int r = 0; r < pM->nrows; r++) {
        for (int s = 0; s < pM->rows[r].ncols; s++) {
           printf("%3i ", pM->rows[r].cols[s]);
        }
        printf("\n");
      }
}
```

Viacrozmerné polia ako polia smerníkov

V predošlom príklade s maticou by potenciálne mohol každý riadok matice mať inú logickú veľkosť, stanovenú zložkou `ncols`. Fyzicky by však zaberal každý riadok rovnaké miesto:

```
sizeof(struct _row)
```

Štandardné viacrozmerné pole predstavuje v pamäti za sebou uložené prvky základného typu:

```
int A[2][3];
```

A[0][0]	A[0][1]	A[0][2]	A[1][0]	A[1][1]	A[1][2]		
A[0]			A[1]				

V príklade so štruktúrou dostávame za sebou uložené štruktúryv rámci štruktúry M:

<table>
<tr><td>M.nrows</td><td>
<table>
<tr><td colspan="3">M.rows[0].ncols</td><td colspan="3">M.rows[1].ncols</td></tr>
<tr><td>M.rows[0].
cols[0]</td><td>M.rows[0].
cols[1]</td><td>M.rows[0]
.cols[2]</td><td>M.rows[1]
.cols[0]</td><td>M.rows[1].
cols[1]</td><td>M.rows[1]
.cols[2]</td></tr>
</table>
</td></tr>
</table>

V prípade uloženia 2d údajov do jednorozmerného poľa počítame pozície ručne:

```
int d[6];
```

d[0*3+0]	d[0*3+1]	d[0*3+2]	d[1*3+0]	d[1*3+1]	d[0*3+2]		
d[0*3+0]			d[1*3 + 0]				

Čo takto vytvoriť si dodatočné pole smerníkov, do ktorého uložíme `&d[0*3+0]`, `&d[1*3 + 0]` ?

```
int *p[2];
p[0] = &d[0];
p[1] = &d[3];
```

Teraz `p[0]` odkazuje na začiatok úseku pamäte `d[0*3+0], d[0*3+1], d[0*3+2]`, t.j. prvý logický riadok, a `p[1]` odkazuje na úsek pamäte `d[1*3+0], d[1*3+1], d[0*3+2]`, t.j. druhý logický riadok.

Ak teraz napíšeme výraz `p[i][j]` (pre `i, j` v stanovenom rozsahu), sprístupníme prvky pôvodne jednorozmerného poľa `d` cez dvojité indexovanie. Ako keby sme mali dvojrozmerné pole bez dodatočných zložitých výpočtov.

Viacrozmerné pole sa teda dá emulovať poliami smerníkov. Pozor, keby sme chceli pridať tretí rozmer, museli by sme mať pole smerníkov na smerníky, atď.

Dynamické viacrozmerné polia

Vytvorenie dynamického dovj- a viac-rozmerného poľa by už teraz malo byť ľahké: alokujeme si pamäť na pole smerníkov (budú odkazovať na riadky matice), a do každého smerníka v poli zapíšeme adresu novo alokovaného poľa základného typu (stĺpce v príslušnom riadku):

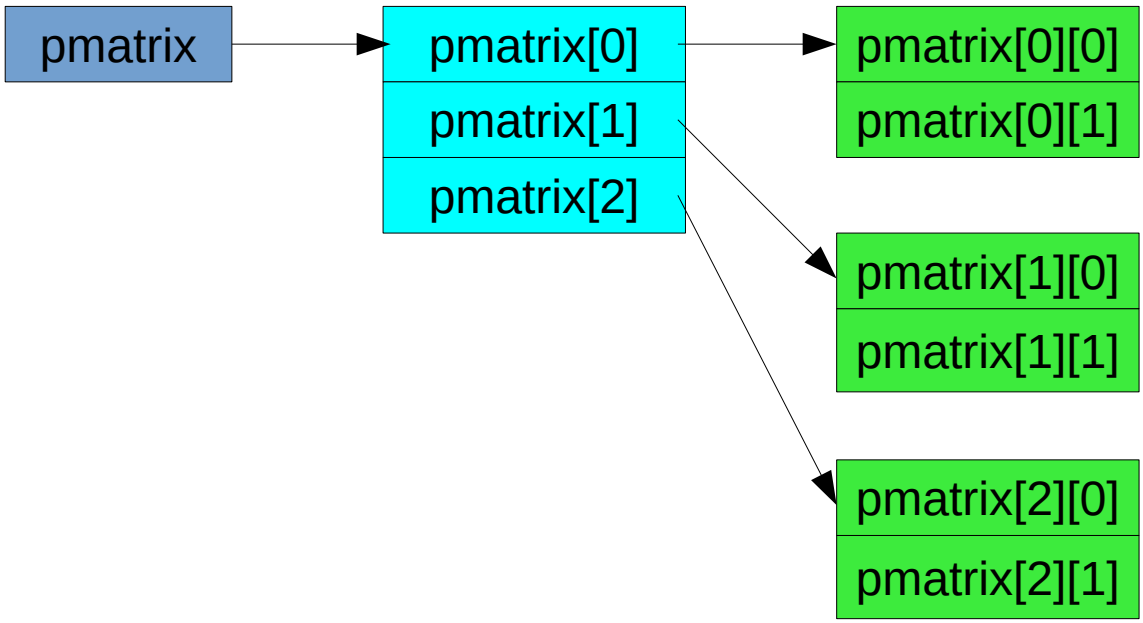
```
int** create_matrix(int nrows, int ncols)
{
    int** pmatrix;   //sem dame adresu dynamickeho pola smernikov

    //alokacia cez malloc, mame miesto na nrows smernikov
    pmatrix = (int**) malloc(nrows * sizeof(int*));

    //alokacia jednotlivych riadkov
    for (int row = 0; row < nrows; row++) {
        //pouzijem calloc, aby v riadkoch boli vynulovane hodnoty
        pmatrix[row] = (int*) calloc(ncols, sizeof(int));
    }

    return pmatrix;
}
```

V pamäti sme vyrobili dátovú štruktúru, ktorá vyzerá nasledovne:



Farebne sú odlíšené rôzne typy, `pmatrix` je typu `int**`, `pmatrix[i]` je typu `int*`, a `pmatrix[i][j]` je typu `int`. Jednotlivé dynamické polia sú alokované nezávisle, takže nemusia za sebou v pamäti nasledovať za sebou (ale prvky jednotlivých polí áno, viď obrázok).

Výhoda takejto dynamickej štruktúry je jednoduché používanie: do funkcií pošleme hodnotu z `pmatrix` a rozsahy indexov v oboch rozmeroch, a ona vie pristupovať vďaka smerníkovej štruktúre k pmatrix ako obyčajnému dvojrozmernému poľu s dvomi indexami. Navyše, riadky sú samostatné smerníky, keď napr. chceme zameniť poradie riadkov `i` a `j` (napr. pri riešení sústav rovníc), stačí

vymeniť smerníky `pmatrix[i]` a `pmatrix[j]`. Výhoda samostatných riadkov je tiež možnosť používať rôzne dĺžky riadkov (vhodné pre polia reťazcov), a samostatne ich realokovať.

Nevýhoda takejto dynamickej štruktúry je zložitejšia správa pamäte, vrátane jej uvoľňovania. V príklade `p10-04.c` je ukážka alokácie, použitia a uvoľnenia pamäte (možnosť `vytvor1`/`uvolni1`). Je tam tiež ukážka pokročilejšia (`vytvor2`), kde alokujeme dynamické 2d pole jediným volaním malloc a nastavením smerníkov. V tomto druhom príklade sa alokuje súvislý úsek pamäte, a časť z neho sa použije na smerníky odkazujúce na riadky (ako pole `p` v predošlej časti), a časť na samotné dáta (uložené podobne ako v prípade poľa `d` v predošlej časti). Keďže sa pole alokovalo jedným volaním, stačí jedno volanie `free` na uvoľnenie pamäte. Nevýhodou je však problematická realokácia, toto riešenie sa hodí skôr na matice (polia s nemeniacim sa rozmerom po ich vytvorení).

TODO: Do zbierky príkladov pribudli úlohy na prácu s 2d poľami. Vyskúšajte si ich preriešiť s rôzne reprezentovanými poľami podľa prednášky.