Práce s maticemi

1 Teorie – proměnné a typy

1.1 Opakování základních pojmů

- Deklarace proměnné / parametru
  - Její popis, tedy jaké bude mít jméno a typ.
  - Používá se v parametrech funkcí.

```
void orejRadu(float pole[], int n);
```

- Deklarace typu
  - Dělá se mimo těla funkcí. (U modulů v hlavičkových souborech.)
  - Lze vyrobit zkratky (nové názvy) pro typy se zdlouhavým zápisem (typicky struktury, možno i pole, ale opatrně, není to zvykem).

```
typedef starý_typ nové_jméno_typu;
```

- Definice proměnné
  - *Příkaz* k vyhrazení paměti pro proměnnou.
  - Vypadá (většinou) stejně jako deklarace, ale dělá se v těle funkce.

```
void orej(float pole[], int n) {
  int m = 2*n;
  float pomocne[m];
  …
}
```

1.2 Typy proměnných

- Proměnná jednoduchého typu
  - *pojmenovaná* paměťová buňka o *velikosti* dané použitým typem
  - „šuplík vyhrazený pro věci stejného typu“
- Proměnná typu pole
  - *pojmenovaná souvislá oblast* paměťových buněk *stejného typu* (prvků pole) ležících vedle sebe
  - „skříň rozdělená na stejně velké šuplíky“
  - seskupuje *prvky* stejného typu ⇒ *homogenní typ*
  - dá se s ním pracovat jako s celkem (`mojePole`) nebo po jednotlivých prvcích (`mojePole[pořadí]`)
- Proměnná typu struktura
  - *pojmenovaná souvislá oblast* paměťových buněk *různého* typu
  - „skříň obsahující různě velké šuplíky na různé věci“
  - seskupuje *složky* různých typů ⇒ *heterogenní typ*
  - dá se s ní pracovat jako s celkem (`maStruktura`) nebo po jednotlivých složkách (`maStruktura.slozka`)

2 Typ pole

```
BazovyTyp nazev[DIMENZE]
```

- Dimenzí může být více. V praxi málokdy více než 3.
- Použití proměnných
  - `nazev` – pole jako celek, např. při použití jako argument funkce
    - **Pozor!** V C nejde pole přiřadit, či kopírovat přiřazením! Musí se to dělat ručně cyklem nebo funkcí `memcpy`.
  - `nazev[index]` – prvek pole v pořadí od začátku daném indexem
    - Prvek je proměnná bázového typu. Jde s ním dělat cokoli, co jde dělat s jinými proměnnými stejného typu → prvek je L-hodnota.

```
pole1 = pole2; // NELZE!
pole1[0] = pole1[1]; // ASI OK

float radek[10];
orej(~~radek[10]~~, 10); // CHYBA!
orej(radek, 10);     // OK
```

3 Matice

```
BazovyTyp nazev[RADKY][SLOUPCE]
```

- Pole o dvou dimenzích ⇒ matice
- Pole polí – prvkem pole je řádek
- Řádky a sloupce v tomto pořadí! Stejné pořadí i při indexování.
- Poznámka: Nepoužívat indexy **x, y** ⇒ vede to k chybným záměnám řádků za

sloupce při indexování. Vhodnější jsou indexy **r, s** (mnemotechnická pomůcka).

3.1 Definice matic

- Pro reálné problémy se matice alokují dynamicky na haldě (pomocí `malloc`).
  - Pomocí typu dvojitý ukazatel. Pozor! Matice se musí alokovat algoritmicky po jednotlivých řádcích!
  - Nejobecnější, ale pracné.

```
int radky = 10, sloupce = 3;
float **matice1;
matice1 = alokuj(radky, sloupce);
```

- Malé matice lze vyrobit dynamicky na zásobníku – jako lokální proměnné typu pole, u nichž budou dimenze zadány pomocí proměnných.
  - **Problém:** Zásobník má omezenou velikost (dělá to OS!). Od určitých rozměrů matice dojde k havarijnímu ukončení programu.
  - **Problém:** Tyto matice je obtížné předávat do podprogramů.
  - **Budeme využívat málokdy.**

```
int radky = 10, sloupce = 3;
float matice2[radky][sloupce];
```

- Pro školní úlohy budeme vyrábět matice pevných rozměrů, např. 100×100 prvků. Potom budeme pracovat jen s částí této matice – budeme si pamatovat počet využívaných řádků a sloupců.
  - **Nevýhoda:** Pro reálné programy zoufale neefektivní.
  - **Výhoda:** Snadněji se s tím pracuje. Pro demonstraci algoritmů to dostačuje.

```
#define MAXN 100
int radky = 10, sloupce = 3;
float matice3[MAXN][MAXN];
```

3.2 Zjednodušení pomocí struktury

- **Tohle budeme používat přednostně.**
- Všechny předchozí varianty vyžadují práci se 3 proměnnými na jednu matici.
- Při práci s více maticemi se rozměry snadno popletou.
- Řešení: zabalit matici do struktury.
- **Výhody**
  - Odpadá zdlouhavá deklarace parametrů funkcí.
  - Funkce má na jednu matici jediný parametr.
  - Časem lze statické pole vyměnit za dynamické bez předělávání funkcí.

```
#define MAXN 100
typedef struct {
  float prvek[MAXN][MAXN];
  int radku, sloupcu;
} Tmatice;
…
// výroba proměnné
Tmatice m4 = {.radku = 10,
  .sloupcu = 3};
…
// indexování prvků
m4.prvek[0][0] = 123;
```

Kde vyrábět

- **Výroba matic – v `main`.** Tady to je nejjednodušší a nezpůsobuje to další problémy.
- Inicializace a zpracování – ve funkcích

3.3 Matice jako parametr funkcí

Varianta pole

- **Budeme využívat jen málokdy.**
- Parametr typu statické pole musí mít uvedeny rozměry dimenzí (poslední může zůstat prázdný, ale je lepší ho přesto uvádět).
- Skutečně využívané rozměry se musí zadávat jako další parametry.

```
void vlacej(~~float m[][]~~); // CHYBA!

void vlacej1(float m[MAXN][MAXN],
  int radku, int sloupcu){
  // práce s částí matice m
  m[0][0] = 123;
  …
}
```

Varianta struktura

- **Tohle budeme používat často.**
- Když je pole ve struktuře, je to jednodušší.
- Aby šlo pole uvnitř měnit, je **nutné předávat strukturu odkazem** (ukazatel) a **ke složkám chodit pomocí šipkového operátoru** (`m->radku`), namísto tečkového operátoru (`m.radku`).

```
void vlacej2(Tmatice *m){
  // práce s částí matice m->prvek
  m->prvek[0][0] = 123;
}
```

3.4 Matice jako argument funkcí při jejich volání

Varianta pole

- Pole se při volání funkce zadává pouze svým názvem!
- Je nutné předávat argumenty s rozměry matice.
- Operátorem [] se vybírá z pole konkrétní prvek. U matice jeden operátor (m[r]) vybírá celý řádek, použití dvou operátorů (m[r][s]) vybírá jeden prvek matice.
- Funkce vždy může měnit předávané pole. Zabránit tomu jde jen použitím klíčového slova **`const`** v hlavičce funkce před deklarací parametru typu pole.

```
float zahon[MAXN][MAXN];

vlacej1(~~zahon[][]~~, 10, 3);// CHYBA!
vlacej1(zahon, 10, 3); // OK

// jde i toto
orejRadu(zahon[5], 3);
// orej 6. řádek mého záhonu
// řádek je pole dlouhé 3 prvky
```

Varianta struktura

- Se strukturou je to jednodušší zvláště při opakovaném používání.
  - Místo tří parametrů bude parametr jen jeden.
  - Odpadá možný omyl s operátorem [].

Zásady

- Vyrábět v `main`.
- Aby šlo matici uvnitř měnit, předáváme vždy odkazem (operátor **&**).

```
Tmatice zahon; // žádný ukazatel!

// případná výroba a inicializace
vyrobMatici(&zahon, 10, 3);

vlacej2(&zahon);

// samotný řádek takto
orejRadu(zahon->prvek[5], 3);
// orej 6. řádek mého záhonu
// řádek je pole dlouhé 3 prvky
```

- Pozor ve funkcích, které už matici přes ukazatel dostaly a chtějí ji zpracovat jinou funkcí.

```
void pohnoj(Tmatice *m) {
  … // pohnojím

  vlacej2(m); // tady není &!!!
  // m je stejného typu, jako
  // vyžaduje funkce vlacej2
}
```

4 Úkol

Matice budeme tisknout a číst v textovém formátu.

```
3 5
 1.5  3.2   -.1   6.4 -2
-5    2.4  10     1.4  4.1
 1.5 -7.2   2   -64.1  7
```

Na prvním řádku jsou dvě celá čísla – počet řádků a počet sloupců. Pak následují prvky matice, jejichž počet odpovídá rozměrům. Mezery navíc mezi prvky nehrají roli.

1. Vytvoř funkci `vyrobMatici` pro inicializaci struktury typu `Tmatice`.
2. Vyrob funkce pro čtení a tisk matic s těmito hlavičkami:

```
void ctiM(FILE *in, Tmatice *m);
void tiskM(FILE *out, Tmatice *m);
```

Poznámka: Při testování použij místo souborů proměnné `stdin` a `stdout`,

když chceš pracovat se standardním vstupem (klávesnice, konzole).

Otestuj je ve funkci `testCteni`, kterou zavoláš v `main`.

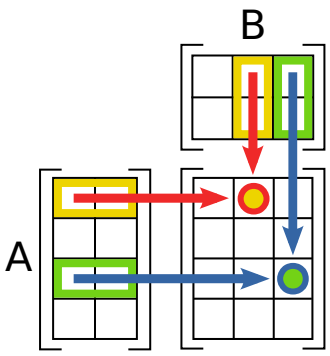
Poznámka: Použij typ `Tmatice`, ukázaný výše.

3. Vytvoř funkci `soucetM` pro přičtení jedné matice ke druhé (načti je ze souborů). Kontroluj, jestli mají obě matice stejné rozměry. Funkce nebude tisknout výsledek, musí ho nějak vrátit.

   Otestuj ji ve funkci `testSoucet`, kterou zavoláš v `main`.

4. Vytvoř funkci `soucinM` pro maticový součin dvou matic. Musíš testovat, zda vstupní matice mají správné rozměry. Výslednou matici musíš nově vyrobit na základě rozměrů obou vstupních matic. Funkce nebude tisknout výsledek, musí ho nějak vrátit (třetí parametr).

   Viz postup a obrázek z Wikipedie:



   Otestuj ji ve funkci `testSoucin`, kterou zavoláš v `main`.