Staticky alokovaná pole

16. října 2012

Pole jsou datové struktury skládající se z mnoha prvků stejného datového typu. V tuto chvíli se budeme věnovat polím typu `int` a `double`, pole prvků typu `char` se nazývají řetězce a práce s nimi je trochu odlišná, proto se jim budeme věnovat podrobněji v závěru kurzu. Pole mohou být jednorozměrná (zejména pro definici vektorů) či vícerozměrná (např. pro definici matic používáme pole dvourozměrná). Z hlediska alokace paměti rozlišujeme v jazyce C dva základní typy polí:

- **staticky alokovaná pole**, jejichž **velikost musí být známá v okamžiku překladu programu**,
- **dynamicky alokovaná pole**, jejichž velikost je možné určit až za běhu programu, ale musí být známa v okamžiku alokace paměti.

Podmínka znalosti velikosti pole v době překladu je v mnoha situacích příliš omezující, ale na druhou stranu je práce se staticky alokovanými poli snazší. Proto se nejprve zaměříme právě na staticky alokovaná pole.

Definice staticky alokovaných polí vypadá následovně:

```
double vektor[4];     //jednorozmerne pole o ctyrech prvcich
double matice[3][4];  //dvourozmerne pole o trech radcich a
ctyrech sloupcich
```

V hranatých závorkách musí být konstanty a v žádném případě se tam nesmí objevit proměnné, takže následující definice není přípustná:

```
int n=4;
double vektor[n];
```

Přesto se Vám může stát, že uvedenou definici některé překladače dovolí. Nicméně některé překladače, včetně toho od firmy Borland, který používáme, tuto definici nepovolí v souladu s normou ANSI/ISO. Proto byste měli v zájmu přenositelnosti Vámi vytvořeného kódu normu respektovat.

Kromě číselných hodnot ovšem můžeme použít i konstantu, kterou si sami definujeme. Globální konstanty definujeme v úvodu programu po vložení knihoven pomocí klíčového slova `define`. Tato definice není příkaz, proto mezi názvem konstanty a její hodnotou nepíšeme rovnítko a na konci nepíšeme středník.

```
#include<stdio.h>

#define POCET 5

int main (void)
{
  int a[POCET];

  return 0;
}
```

Definici globálních konstant bychom si ale vždy měli dobře rozmyslet a neměli bychom je nikdy definovat, pokud k tomu nemáme opravdu dobrý důvod.

K jednotlivým prvkům pole pak přistupujeme pomocí názvu pole a příslušného indexu. **Pozor: Index prvního prvku má v jazyce C vždy hodnotu 0 a poslední prvek má pak index o jedničku menší než je velikost pole!** Jedna z nejčastějších chyb je pokus přistupovat k prvku s indexem rovným velikosti pole, což vede v lepším případě k pádu programu, v horším případě k nevyzpytatelným výsledkům. Příklady přiřazení hodnoty prvkům polí:

```
vektor[0] = 1;        //priradi hodnotu 1 do 1. prvku pole vektor
vektor[3] = 4;        //priradi hodnotu 4 do posledniho prvku pole vektor
vektor[4] = 5;        //chyba, hodnota 5 se ulozi do pameti za alokovane pole vektor
matice[0][3] = 1;     //priradi hodnotu 1 do 1. radku a 4. sloupce pole matice
```

Na rozdíl od definice pole je v případě přiřazení hodnoty možné použít v hranatých závorkách proměnnou. Toho využijeme zejména v kombinaci s cykly. Následující kód například přiřadí do pole vektor hodnoty od 1 do 4.

```
int i, vektor[4];

for ( i=0; i<4; i++ ) {
  vektor[i] = i+1;
}
```

Stejně jako proměnné, i pole můžeme definovat a zároveň hodnoty jejich prvků inicializovat. Při inicializaci uvedeme po sobě jdoucí prvky pole oddělené čárkou a uzavřené do složených závorek. V případě definice s inicializací je možně vynechat hodnotu určující velikost jednorozměrného pole a ponechat hranaté závorky prázdné. U vícerozměrných polí je možné vynechat vždy pouze první rozměr, velikost pole v ostatních rozměrech musí být uvedena.

```
int a[4] = {1, 2, 3, 4};
int b[] = {1, 2, 3, 4};
int A[][] = {{ 1, 1, 1, 1},{2, 2, 2, 2},{3, 3, 3, 3}}; //chyba, nutné uvést 2. rozměr
int B[][4] = {{ 1, 1, 1, 1},{2, 2, 2, 2},{3, 3, 3, 3}};
int C[3][4] = {{ 1, 1, 1, 1},{2, 2, 2, 2},{3, 3, 3, 3}};
```