Jednorozměrná pole

Základy programování 1

Martin Kauer

Motivace

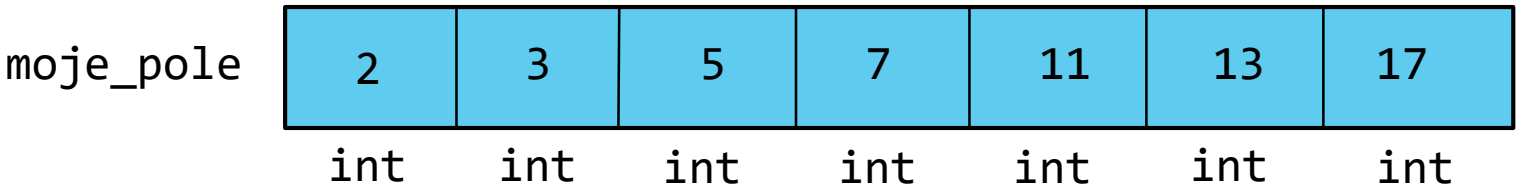
- Dostaneme zadání, že od uživatele máme načíst 5 čísel:

```
int a,b,c,d,e;
scanf(“%d”,&a);
scanf(“%d”,&b);
...
```

- Co když chceme načíst 50 čísel nebo 500?

(Jednorozměrné) pole

- Datová struktura
  - Lineární
  - Homogenní = prvky stejného datového typu
  - Statická = předem určený počet prvků
- Pole umožňuje pohodlně zpracovat větším množství dat



Vytvoření pole

- Pole je před použitím je potřeba vytvořit
- bez inicializace:
  typ identifikátor[*velikost*];
- s inicializací:
  typ identifikátor[*velikost*] = {*inicializační hodnoty*};
- Inicializační hodnoty oddělujeme čárkou
- Hodnot může být i méně, než jaká je velikost pole
- V případně neuvedení velikosti se tato určí podle počtu uvedených hodnot
- Příklady:

```
int cisla[5];
int cisla[5] = { 1, 2, 3, 4, 5 };
int cisla[] = { 1, 2, 3, 4, 5 };
int cisla[10] = { 1, 2, 3, 4, 5 };
double desetinna[4];
double desetinna[] = { 1.1, 5.1, 3.2, 1e5 };
```

Používání pole

- K prvkům přistupujeme pomocí operátoru indexu [ ]
- Prvky mají indexy od 0 do velikosti pole – 1
- Příklad:

```
int i;
int pole[10];

for (i = 0; i < 10; i++){
    pole[i] = i + 1;
}

for (i = 0; i < 10; i++){
    printf("%i, ", pole[i]);
}
```

Velikost pole

- Velikost pole nelze zjistit
- Při práci s polem obvykle máme v nějaké proměnné / konstantě i jeho velikost
- V ANSI C:

```
#define VELIKOST 10
...
int i;
int pole[VELIKOST];
for (i = 0; i < VELIKOST; i++){
    pole[i] = i + 1;
}
```

- Od C99 funguje také:

```
int i;
const int velikost = 10;
int pole[velikost];
for (i = 0; i<velikost; i++){
    pole[i] = i + 1;
}
```

Textové řetězce

- Pole znaků ukončených znakem ‘\0’
- ‘\0’ má ASCII hodnotu 0
- Velikost pole je tedy délka řetězce + 1
- Funkce pro práci s řetězci najdete v knihovně **string.h**
- Příklad vytvoření:

```
char retezec[] = "ahoj";
char retezec[] = { 'a', 'h', 'o', 'j', '\0' };
```

- Příklad využití zarážky:

```
char retezec[] = "ahoj";
char znak;
int i = 0;
while (znak = retezec[i]){
    printf("%c ", znak);
    i++;
}
```

Vybrané funkce ze string.h

- Délka řetězce: `strlen`
- Kopírování řetězce: `strcpy`, `strncpy`
- Spojení řetězců: `strcat`, `strncat`
- Porovnávání řetězců (lexikografické): `strcmp`, `strncmp`
- Hledání znaku v řetězci: `strchr` (zleva), `strrchr` (zprava)
- Hledání podřetězce v řetězci: `strstr`
- A mnoho dalších…

Poznámky

- Používání prvků v poli, bez jejich inicializace může mít neočekávané výsledky.

```
int pole[5];
if (pole[0] > 0) {}
```

- Nesmíme sahat za hranice pole tzn. můžeme přistupovat pouze na indexy **0** až **(velikost pole - 1)**
- Klíčové slovo `sizeof` **není určeno** pro získání velikosti pole!
- Při zadávání velikosti pole určujeme celkový počet prvků v poli. Neplést si s indexací pole od 0.
- Zbytečné průchody cyklem: řídící proměnná cyklu se nemusí měnit pouze jednoduchou inkrementací/dekrementací. Můžeme použít libovolnou změnu např: i+=2;

Cvičení

Vytvořte program, který pomocí algoritmu Eratosthenova síta určí a **čitelně** vypíše všechna prvočísla menší než `N`, což bude hodnota, kterou bude možné změnit na jediném místě v kódu.

Algoritmus Eratosthenova síta:

1. Vytvoříme pole všech čísel až do maximálního zkoumaného čísla.
2. Vše, co je menší než 2, vyškrtneme.
3. Procházíme pole od začátku, dokud nenajdeme nevyškrtnuté číslo. Toto číslo je prvočíslem.
4. Vyškrtáme z pole všechny násobky právě nalezeného prvočísla.
5. Pokračujeme v procházení pole (krok 3), dokud nedorazíme na jeho konec.

Cvičení příklad

Výstup programu pro **`N = 10`**:

```
2       3       5       7
```

Výstup programu pro **`N = 100`**:

```
2       3       5       7       11      13      17      19      23      29
31      37      41      43      47      53      59      61      67      71
73      79      83      89      97
```