Základy programování v C++ 14. cvičení ukazatele a pole (ukazatelová aritmetika)

Zuzana Petříčková

10. listopadu 2019

Přehled

Ukazatele v C/C++ – pokračování

Ukazatel (pointer)

- proměnná, jejíž hodnotou je adresa v paměti počítače
  - ukazatel na data
  - ukazatel na funkci

Kdy ukazatele využijeme?

- **bylo minule:** úvod, předávání parametrů funkcí odkazem (alternativa referencí)
- **dnes:** efektivnější práce s poli
- **příště:** dynamická alokace paměti

Ukazatele a pole

- Pole v C++ je realizované jako ukazatel na první prvek:

```
int a[10] = {1,2,3,4,5,6,7,8,9,10};
int *ua = a;       // bude ukazovat na prvni prvek
int *va = &a[2];   // bude ukazovat na treti prvek

vypis(a, 10);  // vypis celeho pole
vypis(ua, 10); // vypis celeho pole
vypis(va, 8);  // vypis pole od indexu 2 do konce
```

- Funkce s parametrem typu pole
  - interně se předává ukazatel na první prvek, následující dva zápisy jsou proto ekvivalentní:

```
void nejakaFunkce(int a[], int delka);
void nejakaFunkce(int  *a, int delka);
```

Struktury a ukazatele

- Přístup ke složkám struktury pomocí operátoru -> nebo **(*u).**

```
...
Zlomek z = {2,3};
Zlomek *uz = &z;
(*uz).citatel = 4; // z.citatel = 4;
uz->citatel = 5;   // z.citatel = 5;
vypis(*uz);        // vypis(z)
...
```

Ukazatele v C/C++ ... cvičení (základy)

1. napište a zavolejte funkci **int* paty(int *a)**, která vrátí ukazatel na pátý prvek pole a.
2. napište a zavolejte funkci **void vypis(const Zlomek *z)**, která vypíše na konzoli Zlomek zadaný pomocí ukazatele

Ukazatele a funkce

- **void funkce1(double *a)** ... parametrem funkce je ukazatel (nebo pole), funkce může měnit skutečnou hodnotu parametru, na který ukazuje **a** (parametr je předán funkci odkazem)
- **void funkce2(const double *a)**... parametrem funkce je ukazatel na konstantu (nebo konstantní pole), funkce nemůže měnit skutečnou hodnotu parametru, na který ukazuje **a** (parametr je nicméně předán funkci odkazem)
- **double* funkce3(...)** ... funkce vrací ukazatel (nebo ,,pole")
- **const double* funkce4(...)** ... funkce vrací ukazatel na konstantu (nebo konstantní pole)

... obdobně pro reference ...

Reference a funkce

- **void funkce1(double &a)** ... parametrem funkce je reference, funkce může měnit skutečnou hodnotu parametru **a** (parametr je předán funkci odkazem)
- **void funkce2(const double &a)**... parametrem funkce je reference na konstantu, funkce nemůže měnit skutečnou hodnotu parametru, na který ukazuje **a** (parametr je nicméně předán funkci odkazem)
- **double& funkce3(...)** ... funkce vrací referenci
- **const double& funkce4(...)** ... funkce vrací referenci na konstantu

Ukazatel jako návratová hodnota funkce

- funkce by **NIKDY** neměla vracet ukazatel (nebo referenci) na lokální proměnnou!
  → proměnná po návratu z funkce zaniká a ukazatel ukazuje „někam na zásobník":

```
/* Funkce vrati ukazatel na nahodnou pamet na zasobniku*/
double* funkce(double *a)
{
    double x = a[3];
    return &x; // CHYBA!!
}
```

Ukazatel jako návratová hodnota funkce

Pokud je ukazatel („na nekonstantu") návratovou hodnotou funkce, mohu jeho dereferenci použít na levé straně přiřazovacího příkazu (jedná se o tzv. l-hodnotu).

```
/* Funkce vrati ukazatel na paty prvek pole */
double* funkce1(double *a)
{
    double *u = &a[4]; // paty prvek pole
    return u;
    // alternativne: return &a[4];
}
int main()
{
    double ad[5] = { 2.1, -5.3, 0.5, 1.1, 4.9 };
    double x = *funkce1(ad);
    *funkce1(ad) = 4;
}
```

Ukazatel jako návratová hodnota funkce

Příklad: maximum pole

```
/* Funkce vrati ukazatel na maximalni hodnotu pole a delky n
double* maximum1(double* a, int n)
{
    int im = 0; // index maxima
    for (int i = 1; i < n; i++)
    {
        if (a[i] > a[im])
            im = i;
    }
    return &a[im];
}
int main()
{
     double ad[5] = { 2.1, -5.3, 0.5, 1.1, 4.9 };
     cout << *maximum1(ad, 5) << endl;
     *maximum1(ad, 5) = 0; // nahradi maximum v poli nulou
     double *p = maximum1(ad, 5);
     *p = 0;  // nahradi maximum v poli nulou
     vypis(ad, 5);
```

Reference jako návratová hodnota funkce

Pokud je reference („na nekonstantu") návratovou hodnotou funkce, mohu ji použít na levé straně přiřazovacího příkazu (jedná se o tzv. l-hodnotu).

```
/* Funkce vrati referenci na paty prvek*/
double& funkce1(double *a)
{
    double &u = a[4];
    return u;
    // alternativne: return a[4];
}
int main()
{
    double ad[5] = { 2.1, -5.3, 0.5, 1.1, 4.9 };
    funkce1(ad) = 4; // totez jako ad[4]=4;
    double &r = funkce1(ad);
    r = 3;
}
```

Reference jako návratová hodnota funkce

Příklad: maximum pole

```
/* Funkce vrati referenci na maximalni hodnotu pole a delky n
double& maximum2(double* a, int n)
{
    int im = 0; // index maxima
    for (int i = 1; i < n; i++)
    {
        if (a[i] > a[im])
            im = i;
    }
    return a[im];
}
int main()
{
    double ad[5] = { 2.1, -5.3, 0.5, 1.1, 4.9 };
    cout << maximum2(ad, 5) << endl;
    maximum2(ad, 5) = 0; // nahradi maximum v poli nulou
    vypis(ad, 5);
}
```

Ukazatelová aritmetika

- pro efektivnější práci s poli
- operátory **==,!=** pro porovnání dvou ukazatelů stejného typu
  - **u1 == u2**, jestliže oba ukazatele obsahují (tj. ukazují na) stejnou adresu
- operátory **<,<=,>,>=** pro porovnání ukazatelů stejného typu (ukazující na prvky stejného pole)
  - **u1 <u2**, jestliže **u1** obsahuje nižší adresu než **u2**

Příklad:

```
int a[10] = {1,2,3,4,5,6,7,8,9,10};
int *ua = a, *va = &a[3];
if (ua == a)
   cout << "ukazuji na stejny prvek\n";
if (ua < va)
   cout << "ua ukazuje na prvek s nizsim indexem\n";
```

Ukazatelová aritmetika

- k ukazatelům, které ukazují na prvky pole, mohu přičítat či odečítat celá čísla (operátory **+,− ,++,− −,+=,−=**)
  - **u = u + n** ... k ukazateli **u** na typ **T** přičtu celé číslo **n**:
    → adresa, na kterou ukazuje **u**, se zvětší o **n*sizeof(T)**

Příklad

```
int a[10] = {1,2,3,4,5,6,7,8,9,10};
int *ua = a;        // totez jako ua = &a[0];
int *va = a+9;      // totez jako va = &a[9];
ua +=3;             // bude ukazovat na ctvrty prvek
ua--;               // bude ukazovat na treti prvek
int x = *(a+2);     // totez jako x = a[2];
x = *(va-1);        // totez jako x = a[9-1];
                    // (ziskam index prvku)
```

Ukazatelová aritmetika

- ukazatele téhož typu, které ukazují na prvky pole, mohu od sebe odečítat (operátor −), výsledkem je **celé číslo**
  - **n = u - v** ... celé číslo **n** je rozdíl indexů prvků, na které ukazují ukazatele **u** a **v**

Příklad

```
int a[10] = {1,2,3,4,5,6,7,8,9,10};
int *ua = a+2;      // totez jako ua = &a[0];
int *va = a+9;      // totez jako va = &a[9];
int n   = va - ua; // n bude 9-2=7
n       = va - a;  // n bude 9-0=9
```

Ukazatelová aritmetika - Příklady

1. Dokončete příklady ze strany 6.
2. Upravte následující dříve implementované funkce tak, aby místo s indexy pracovali s ukazateli s pomocí ukazatelové aritmetiky:
   - **void vypis(const double *a, int n)**, která vypíše na konzoli všechny prvky pole **a** délky **n**.
   - **double* maximum(double *a, int n)**, která vrátí ukazatel na maximalni prvek pole **a** délky **n**.

   Dobrovolně
   - **void otoc(double *a, int n)**, která otočí pořadí prvků v poli **a** délky **n**.

Ukazatelová aritmetika

Příklad 1 ... možné řešení pomocí indexů:

```
/* Vypis vsech prvku pole a delky n na konzoli. */
void vypis(const double *a, int n)
{
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << endl;
}
```

Ukazatelová aritmetika

Příklad 2 ... řešení pomocí indexů:

```
/* Funkce vrati ukazatel na maximalni hodnotu pole a delky n
double* maximum1(double* a, int n)
{
    int im = 0; // index maxima
    for (int i = 1; i < n; i++)
    {
        if (a[i] > a[im])
            im = i;
    }
    return &a[im];
}
int main()
{
     double ad[5] = { 2.1, -5.3, 0.5, 1.1, 4.9 };
     cout << *maximum1(ad, 5) << endl;
     *maximum1(ad, 5) = 0; // nahradi maximum v poli nulou
     vypis(ad, 5);
}
```

Ukazatelová aritmetika

Příklad 3 ... možné řešení pomocí indexů:

```
/* Funkce prohodi v poli a prvky na indexech i a j. */
void prohod(double *a, int i, int j)
{
    double pom = a[i];
    a[i] = a[j];
    a[j] = pom;
}

/* Funkce otoci pole a delky n. */
void otoc(double *a, int n)
{
    int i, j;
    for (i = 0, j = n - 1; i < j; i++, j--)
        prohod(a, i, j);
}
```