Základy programování v C++ 12. cvičení

Zuzana Petříčková

12. listopadu 2018

Přehled

Ukazatele v C/C++

Ukazatel (pointer)

- **laicky:** proměnná, která ,,ukazuje" na jinou proměnnou v paměti
- **lépe:** proměnná, jejíž hodnotou je adresa v paměti počítače
  - ukazatel na data
  - ukazatel na funkci

Kdy ukazatele využijeme?

- efektivnější práce s daty (např. s poli)
- předávání parametrů funkcí odkazem
- dynamická alokace paměti

Ukazatele v C/C++

Ukazatel (pointer) - deklarace:

```
domenovy_typ *jmeno; // kolem * mohou a nemusi byt mezery
```

Příklady

```
int i = 16;
cout << &i << endl; // adresa promenne i v pameti
int j = (int)(&i);  // adresa jako cislo v desitk. soust.

int *pi;            // pi je ukazatel na datovy typ int
pi = &i;            // do ukazatele pi vlozime adresu i
int *pj = &i;       // inicializace ukazatele pj na adresu i
double *pd, *pe;    // deklarace dvou ukazatelu na double
                    // (bez inicializace)
```

- operátor **&** ... získání adresy proměnné v paměti

Ukazatele v C/C++

Příklady - pokračování:

```
int i = 16;
int *pi;          // pi je ukazatel na datovy typ int
pi = &i;          // do ukazatele pi vlozime adresu i
int *pj = &i;     // inicializace ukazatele pj na adresu i

cout << &i << endl; // vytiskne adresu promenne i v pameti
cout << pi << endl; // vytiskne adresu promenne i v pameti

cout << i << endl; // vytiskne hodnotu promenne i
cout << *pi << endl; // vytiskne hodnotu promenne i
```

- operátor **&** ... získání adresy proměnné v paměti
- operátor * ... přístup k paměti, na kterou ukazatel ukazuje (tzv. dereferencování)
  symbol * je použit také v deklaraci ukazatele

Ukazatele v C/C++

Příklady - pokračování:

```
int i = 3, j = 5;
int *pi;            // pi je ukazatel na datovy typ int
pi = &i;            // do ukazatele pi vlozime adresu i

*pi = 10;           // do promenne i vlozime hodnotu 10
          // i = 10;

*pi = *pi + j;      // v promenne i bude hodnota 15
          // i = i + j;

pi = &j;            // do ukazatele pi vlozime adresu j
(*pi)++;            // v promenne j bude hodnota 6
          // j++;
```

Ukazatele v C/C++

Ukazatel nikam (nullpointer)

- neukazuje na žádnou platnou adresu

```
int *pi;
...
pi = nullptr; // od C++11, doporuceno
pi = 0;       // starsi zapis
pi = NULL;    // makro, knihovna stdef.h
```

Potenciální problémy ukazatelů

- ukazatel ukazuje „na náhodnou adresu v paměti" → mohu přepsat paměť, která mi nepatří
- ukazatel ukazuje „nikam" (nullpointer) → pokus o dereferenci vyvolá výjimku

```
int *pi;
...
pi = nullptr;
cout << (*pi) << endl // CHYBA!
```

Ukazatele v C/C++

Automatická konverze na logickou hodnotu

- ukazatel lze použít všude tam, kde se očekává logická hodnota
- ukazatel nikam (nullptr) se konvertuje na **false**
- nenulový ukazatel se konvertuje na **true**

```
int *pi;
...
if (pi)
    cout << (*pi) << endl;
```

Ukazatele v C/C++

Ukazatel bez doménového typu

- lze mu přiřadit hodnotu libovolného ukazatele
- pro opačné přiřazení je třeba použít přetypování
- nelze dereferencovat

```
int i=1, j=2;
int *pi = &i; // pi ukazuje na i
void *v = &j; // v ukazuje na j
pi = (int*)v; // pi ukazuje na j
// nelze i = (*v);
i = (*pi);    // i bude 2
...
```

Ukazatele v C/C++

Ukazatel jako parametr funkce

```
void vypis(int *x)
{
    cout << "Hodnota promenne je " << (*x) << endl;
    cout << "Adresa promenne je " << x << endl;
}
int main()
{
    int a = 56;
    int *pa = &a;
    vypis(&a);
    vypis(pa);
    ...
}
```

Ukazatel jako parametr funkce

Předávání parametrů hodnotou

```
void funkce(int a, int b)
{
    ...
}
```

- funkce pracují s **lokální kopií** svých parametrů, tyto kopie jsou po ukončení funkce zrušeny
  → **funkce** nemůže měnit skutečné hodnoty parametrů **a** a **b**

Předávání parametrů odkazem (ukazatelem)

```
void funkce(int *a, int *b)
{
    ...
}
```

- funkce může měnit skutečné hodnoty parametrů **a** a **b** (změna se projeví „i venku")

Ukazatel jako parametr funkce - příklad:

```
void zmen(int x)
{
    x++;
    cout << "Hodnota promenne uvnitr je " << x << endl;
}
void zmen(int *px)
{
    (*px)++;
    cout << "Hodnota promenne uvnitr je " << (*px) << endl
}
int main()
{
    int a = 56;
    zmen(a);
    cout << "Hodnota promenne je " << a << endl;
    zmen(&a);
    cout << "Hodnota promenne je " << a << endl;
    ...
}
```

Ukazatele v C/C++

Ukazatele ... cvičení (základy)

1. vytvořte proměnnou typu double, vypište její adresu v paměti, vytvořte ukazatel ukazající na tuto proměnnou a pomocí něj změňte hodnotu této proměnné a vypište ji
2. sečtěte dvě proměnné pomocí ukazatelů
3. napište funkci **prohod()**, která prohodí obsah svých skutečných parametrů
4. napište funkci **serad()**, která seřadí své dva skutečné parametry podle velikosti (první bude menší)
5. napište funkci **void petkrat(int *x)**, která vynásobí skutečný parametr **x** pěti

Ukazatele v C/C++

Ukazatele ... cvičení (základy)

- vytvořte proměnnou typu double, vypište její adresu v paměti, vytvořte ukazatel ukazající na tuto proměnnou a pomocí něj změňte hodnotu této proměnné

```
double x = 3.25;
cout << "hodnota x je " << x << endl;
cout << "adresa x je " << &x << endl;
double *px;
px = &x;
// nebo  double *px = &x;
(*px) = 4.67;
(*px) *= -1;
cout << "nova hodnota x je " << x << endl;
```

Ukazatele v C/C++

Ukazatele ... cvičení (základy)

- sečtěte dvě proměnné pomocí ukazatelů

```
double x = 3.25;
double y = -1.75;
double z;
double *px = &x, *py = &y, *pz = &z;
(*pz) = (*px) + (*py);
//    cout << "hodnota z je " << (*pz) << endl;
cout << "hodnota z je " << z << endl;
```

Ukazatele v C/C++

Funkce, která neprohodí obsah svých parametrů

```
void neprohod(int a, int b)
{
    int c = a;
    a = b;
    b = c;
}
int main()
 {
    int a = 5, b = 15;
    cout << a << " a " << b << endl;
    neprohod(a,b);
    cout << a << " a " << b << endl;
    return 0;
 }
```

Ukazatele v C/C++

Funkce, která prohodí obsah svých parametrů

```
void prohod(int *pa, int *pb)
{
    int c = *pa;
    *pa = *pb;
    *pb = c;
}
  int main()
 {
    int a = 5, b = 15;
    cout << a << " a " << b << endl;
    prohod(&a,&b);
    cout << a << " a " << b << endl;
    ...
 }
```

Ukazatele v C/C++

Funkce, která seřadí své parametry podle velikosti

```
void serad(int *pa, int *pb)
{
    if ((*pa) > (*pb))
        prohod(pa,pb);
}
int main()
{
    int a = 23, b = 15;
    cout << a << " a " << b << endl;
    serad(&a,&b);
    cout << a << " a " << b << endl;
    ...
}
```

Funkce a předávání hodnot odkazem

1. **pomocí ukazatelů** (bylo již v jazyce C)
   - trochu nepohodlné (jiný zápis, musíme dereferencovat)
   - potenciálně nebezpečné (co když je **pa** nullptr nebo "ukazuje jinam"?)

```
void prohod(int *pa, int *pb)
{
    int c = *pa;
    *pa = *pb;
    *pb = c;
}
```

2. **pomocí tzv. referencí** (v jazyce C++)
   - bezpečnější a pohodlnější způsob
   - interně se jedná o konstantní statické ukazatele, ale syntaxe je stejná jako pro typ, na který reference odkazuje

Funkce a předávání hodnot odkazem

2. pomocí tzv. referencí (v jazyce C++)

- bezpečnější a pohodlnější způsob
- interně se jedná o konstantní statické ukazatele, ale syntaxe je stejná jako pro typ, na který reference odkazuje

```
void prohod(int &a, int &b)
{
    int c = a;
    a = b;
    b = c;
}
...
int main()
{
    int a = 23, b = 15;
    cout << a << " a " << b << endl;
    prohod(a,b);
    cout << a << " a " << b << endl;
    return 0;
}
```

Reference

Deklarace

- podobně jako ukazatel, místo znaku * znak **&** (neplést s operátorem **&** pro získání adresy proměnné)

```
int x = 7, y = 5;
int &rx = x;
rx++;   // totez jako x++;
rx = y; // totez jako x = y;
x += y;
cout << x << endl;
cout << rx << endl;
```

Omezení

- referenci je třeba inicializovat ihned při deklaraci na nějakou l-hodnotu
- odkazovanou proměnnou nelze později změnit
  → na referenci se mohu dívat jako na nové jméno pro existující proměnnou

Reference

Reference může odkazovat i na prvek pole:

```
int a[5] = { 1,2,3,4,5 };
int &atri = a[3];
atri = 8; // totez jako a[3]=8
cout << a[3] << endl;
cout << atri << endl;
```

Reference

Význam

- předávání parametrů funkcí odkazem
- jiné jméno pro existující proměnnou / prvek pole
- návratový typ funkce (funkce vracející l-hodnotu, ukážeme později)

Další omezení

- nelze deklarovat ukazatel na referenci, referenci na referenci a pole referencí
- naopak lze deklarovat referenci na pole nebo referenci na ukazatel
- reference z principu ,,nezná" nullptr (někdy výhoda)

Reference

Další využití reference - struktura jako parametr funkce:

```
void vypis(Zamestnanec z)
{
    ...
}
```

- zbytečně se vytváří kopie struktury na zásobníku → lépe:

```
void vypis(const Zamestnanec &z)
{
    ...
}
```

případně:

```
void vypis(const Zamestnanec *uz)
{
    ...
}
```

Struktury - další příklady na procvičení

- Definujte strukturu **Zlomek** s celočíselnými složkami **citatel** a **jmenovatel**.
- Napište funkce pro základní operace se zlomky (sčítání, odčítání, násobení, dělení). Funkce převedou výsledek do základního tvaru.
  Příklad: součet 2/3 a -1/6 je 1/2
- **Pomocné funkce**
  - **int nsd(int x, int y)**, která spočítá největšího společného dělitele čísel x a y.
  - **void zakladni_tvar(Zlomek &x)**, která převede zlomek do základního tvaru (vydělí čitatele i jmenovatele jejich největším společným dělitelem a opraví znaménko minus, aby bylo pouze u čitatele).
  - **void vypis_zlomek(const Zlomek &x)**, která vypíše zlomek na konzoli.

Struktury - další příklady na procvičení

Euklidův algoritmus pro výpočet největšího společného dělitele (pro $x, y > 0$)

$$nsd(x, y) = \begin{cases} x, & x == y \\ nsd(x - y, y), & x > y \\ nsd(x, y - x), & x < y \end{cases}$$

Seznam funkcí k implementaci

```
// Zlomek secti(Zlomek x,Zlomek y};
Zlomek secti(const Zlomek &x,const Zlomek &y);
Zlomek odecti(const Zlomek &x,const Zlomek &y);
Zlomek vynasob(const Zlomek &x,const Zlomek &y);
Zlomek vydel(const Zlomek &x,const Zlomek &y);
int nsd(int x, int y);
void zakladni_tvar(Zlomek &z);
void vypis_zlomek(const Zlomek &x);
```

Pro struktury mohu definovat (přetížit) i operátory

klíčové slovo **operator**:

```
...
Zlomek operator + (const Zlomek& x,const Zlomek& y)
{
  return secti(x, y);
}
Zlomek operator - (const Zlomek& x,const Zlomek& y)
{
  return odecti(x, y);
}
int main()
{
  Zlomek x= {2, 3}, y ={-2, 12}, z;
  z = x + y;
  ...
}
```