Základy programování v C++ 16. cvičení Dynamické datové struktury - pokračování

Zuzana Petříčková

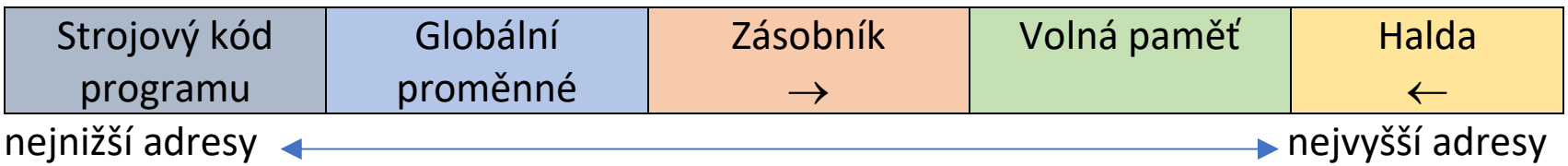
19. listopadu 2019

Přehled

Dynamické proměnné v C/C++

- nevznikají deklarací, ale jsou vytvářeny (**alokovány**) a rušeny (**dealokovány**)
  - vytvoříme je na místě programu, kde je potřebujeme
  - zrušíme je ve chvíli, kde je přestaneme potřebovat
- uložené v části paměti zvané halda (**heap**)
- nemají jména, zacházíme s nimi pouze pomocí ukazatelů
- umožňují práci s daty, o kterých předem nevíme, jak budou velká

Celková paměť vyhrazená pro program a jeho data:



Vytvoření a zrušení dynamické proměnné

```
int *ui;
ui = new int;
...
*ui = 5;
cout << *ui;
...
delete ui;
ui = nullptr;
```

Operátor **new** ... vytvoření (alokace) proměnné

- vyhradí v paměti místo pro proměnnou daného typu a vrátí ukazatel na toto místo

Operátor **delete** ... zrušení (dealokace) jedné proměnné

- uvolní paměť, na kterou ukazuje ukazatel (ale ukazatel nevynuluje → potenciální nebezpečí)

Vytvoření a zrušení jednorozměrného dynamického pole

```
int n;
cout << "Zadejte pocet prvku pole: " << endl;
cin >> n;

int    *ui = new int[n];
...
delete [] ui;
ui = nullptr;
```

Operátor **new T[n]** ... vytvoření dynamického pole délky n s prvky typu **T**

- vyhradí v paměti místo pro pole daného typu a vrátí ukazatel na toto místo

Operátor **delete []** ... zrušení celého pole

- operátor **delete** by uvolnil paměť jen prvního prvku pole

Příklad: dynamicky se zvětšující pole

Příklad

- cílem je implementovat pole, které se bude dynamicky zvětšovat podle potřeby (ve chvíli, kdy uživatel bude chtít přidat další prvek do již plného pole).

Příklad: dynamicky se zvětšující pole

Nástřel

- cílem je implementovat pole, které se bude dynamicky zvětšovat podle potřeby (ve chvíli, kdy uživatel bude chtít přidat další prvek do již plného pole).

```
int n = 1;                  // aktualni delka pole
float *a= new float[n];
a[0] = 3.4;

/* n predame jako referenci (popr. ukazatel)
   abychom ho mohli ve funkci zmenit (zvysit o 1)
*/
a = pridej(a,n,4.5);
a = pridej(a,n,6.7);
...
delete[] a;
```

Příklad: dynamicky se zvětšující pole

Nástřel

```
/* Funkce prida prvek na konec jiz plneho dynamickeho pole
* a ... puvodni pole (funkce ho zrusi)
* n ... pocet prvku pole (bude zmenen)
* x ... vkladana hodnota
* funkce vrati nove vytvorene zvetsene pole
*/
float* pridej(float *a, int &n, float x)
{
  // 1. vytvor nove pole delky (n+1) (operator new)
  // 2. prekopiruj do nej vsechny prvky pole a
  // 3. na index n vloz x
  // 4. zvetsi n o jednicku
  // 5. dealokuj puvodni pole (operator delete)
  // 6. vrat nove pole
}
```

Příklad: dynamicky se zvětšující pole

Nástřel ... alternativně:

```
/* Funkce prida prvek na konec jiz plneho dynamickeho pole
* a ... reference na ukazatel na pole
*       (ukazatel bude zmenen)
* n ... delka pole (pocet prvku pole) (bude zmenena)
* x ... vkladana hodnota
*/
void pridej1(float *&a, int &n, float x)
{
  // 1. vytvor nove pole delky (n+1) (operator new)
  // 2. prekopiruj do nej vsechny prvky pole a
  // 3. na index n vloz x
  // 4. zvetsi n o jednicku
  // 5. dealokuj puvodni pole (operator delete)
  // 6. do a vloz adresu noveho pole
}
```

Příklad: dynamicky se zvětšující pole

Lépe:

- pro pole si pamatuji jeho aktuální kapacitu (k) a počet uložených prvků (n)
- při naplnění kapacity pole nezvětším o 1, ale např.
  - o x prvků, kde x je vhodně zvolená konstanta
  - x-krát (typicky dvakrát) ... obecně efektivnější

```
int n = 0; // aktualni pocet ulozenych prvku
int k = 1; // aktualni kapacita pole
float *a= new float[k];
pridej(a,n,k,4.5); // a, n a k predame jako referenci
                   // (popr. ukazatel)
pridej(a,n,k,6.7);
...
delete[] a;
```

Příklad: dynamicky se zvětšující pole... lépe:

```
/* Funkce prida prvek na konec dynamickeho pole (za posledni)
* a ... reference na ukazatel na pole
*       (ukazatel bude zmenen)
* n ... pocet prvku ulozenych v poli (bude zmenen)
* k ... kapacita pole (bude zmenena)
* x ... vkladana hodnota
*/
void pridej(float *&a, int &n, int &k, float x)
{
  // 1. pokud je pole plne (k == n)
  // 1.1. vytvor nove pole delky (2*k) (operator new)
  // 1.2. prekopiruj do nej vsechny prvky pole a
  // 1.3. dealokuj puvodni pole (operator delete)
  // 1.4  do a vloz adresu noveho pole
  // 1.5  do k vloz novou kapacitu pole
  // 2. do pole a na index n vloz x
  // 3. do n vloz novy pocet prvku pole
}
```

Příklad: dynamicky se zvětšující pole

Ještě lépe a přehledněji:

- pole, jeho aktuální délku a kapacitu obalíme do struktury

```
struct ChytrePole
{
    float *a = nullptr;
    int n = 0;
    int k = 0;
};
int main()
{
    ChytrePole s;

    vytvor(s); // s predame jako referenci
    ...                // (popr. ukazatel)
    pridej(s,4.5);
    pridej(s,6.7);
    ...
    zrus(s);
}
```

Příklad: dynamicky se zvětšující pole

Implementujte nad chytrým polem následující funkce:

```
void zrus(ChytrePole &s);
void vytvor(ChytrePole &s);

/* pridej prvek x na konec dynamickeho pole s */
void pridej(ChytrePole &s, float x);

/* smaz prvek na indexu i */
void smaz(ChytrePole &s, int i);

/* vypis obsah pole s */
void vypis(const ChytrePole &s);
```

Příklad: dynamicky se zvětšující pole

```
void zrus(ChytrePole &s)
{
  // 1. dealokuj pole s.a (operator delete)
  // 2. nastav pocet prvku i kapacitu pole na 0,
  //    do s.a vloz nullptr
 }

void vytvor(ChytrePole &s)
{
  // 1. pokud pole neni prazdne, tak ho vyprazdni (zrus)
  // 2. do s.a vloz adresu noveho pole delky napr. 1
  // 3. do s.n vloz pocet prvku ulozenych v novem poli (t.j.,
        do s.k vloz kapacitu pole
 }
```

Příklad: dynamicky se zvětšující pole

```
/* pridej prvek x na konec dynamickeho pole s */
void pridej(ChytrePole &s, float x)
{
  // 1. pokud je pole plne (s.k == s.n)
  // 1.1. vytvor nove pole delky (2*s.k) (operator new)
  // 1.2. prekopiruj do nej vsechny prvky pole s.a
  // 1.3. dealokuj puvodni pole (operator delete)
  // 1.4  do s.a vloz adresu noveho pole
  // 1.5  do s.k vloz novou kapacitu pole
  // 2. do pole s.a na index s.n vloz x
  // 3. do s.n vloz novy pocet prvku pole
}
```

Příklad: dynamicky se zvětšující pole

Další oprace nad polem (na procvičení):

```
/* smaz prvek na indexu i */
void smaz(ChytrePole &s, int i)
{
  // 1. pokud i neni ani moc velke ani moc male:
  // 1.1. vsechny prvky za indexem i prekopiruj
          o jednu pozici zpet
  // 1.2. do s.n vloz novy pocet prvku pole
          (kapacita se nemeni)
}
```

Příklad: dynamicky se zvětšující pole

Další oprace nad polem (na procvičení):

```
/* vrati index prvku (vrati -1, pokud ho nenajde) */
int najdi(ChytrePole &s, float x);
{
  ...
}
/* pokud je prvek v poli, tak jeho prvni vyskyt smaze */
void smaz(ChytrePole &s, float x)
{
  ... (zavola predchozi dve funkce)
}

/* pokud je prvek v poli, tak vsechny jeho vyskyty smaze */
void smazVse(ChytrePole &s, float x)
{
  ... (slo by resit s vyuzitim predchozich funkci,
       nebo lepe - efektivneji)
}
```

Příklad: dynamicky se zvětšující pole

Ukázka: Povídací prográmek nad dynamickým polem:

```
Pole: 4.5 6.7 0.1
 pro pridani prvku na konec zadej znak 'a'
 pro smazani prvniho vyskytu prvku zadej znak 'c'
 pro smazani vsech vyskytu prvku zadej znak 'd'
 pro zmenu prvku na indexu zadej znak 'z'
 pro ukonceni prace zadej jiny znak (napr. 'k')
Zadej operaci, kterou chces provest: a
Zadej cislo, ktere chces vlozit na konec pole: 7.8

Pole: 4.5 6.7 0.1 7.8
 pro pridani prvku na konec zadej znak 'a'
 pro smazani prvniho vyskytu prvku zadej znak 'c'
 pro smazani vsech vyskytu prvku zadej znak 'd'
 pro zmenu prvku na indexu zadej znak 'z'
 pro ukonceni prace zadej jiny znak (napr. 'k')
Zadej operaci, kterou chces provest: k
```