Základy programování v C++ 16. cvičení

Zuzana Petříčková

4. prosince 2018

Přehled

Spojový seznam

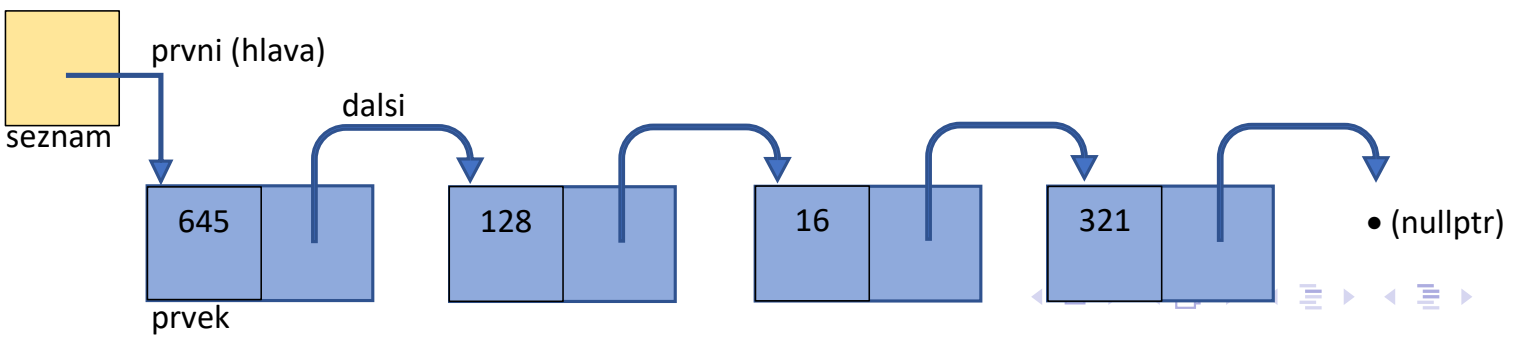
Jednosměrně zřetězený (lineární) spojový seznam

- dynamická datová struktura, jejíž prvky jsou stejného typu, ale na rozdíl od pole nejsou nutně v paměti umístěny za sebou

Narozdíl od dynamického pole:

- **„rychlejší"** vkládání nového prvku „na" požadovanou pozici
- **„pomalejší"** přístup k požadovanému prvku

Spojový seznam je velmi často používaná datová struktura, je součástí standardních knihoven většiny moderních programovacích jazyků (C++, Java,...)



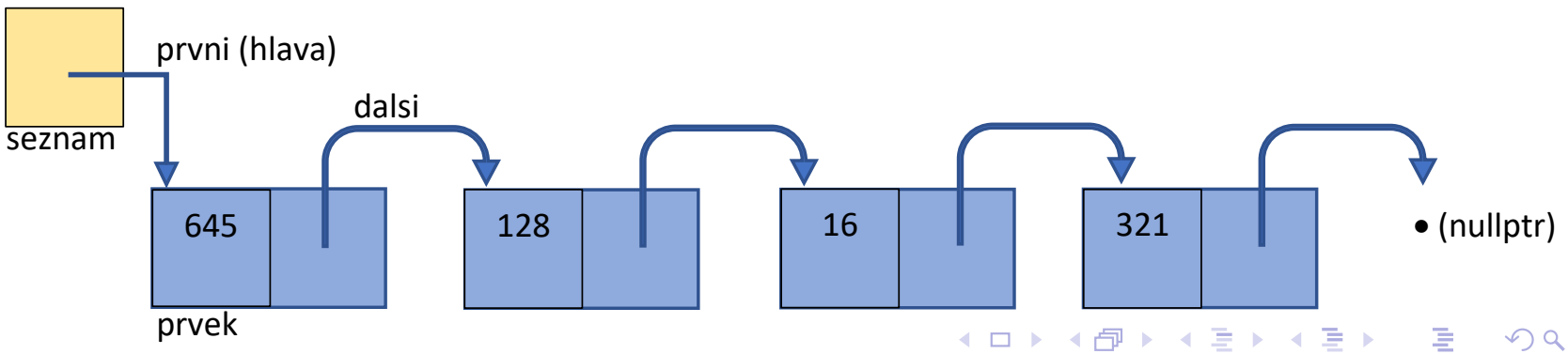
Spojový seznam

Struktura prvku

- vlastní data
- ukazatel na další prvek (**dalsi** / **next**) nebo **nullptr**

Struktura seznamu

- ukazatel na první prvek (**první** / **hlava** / **head**)
- prvky seznamu nacházející se za hlavou se někdy souhrnně nazývají jako **ocas** / **tail**



Spojový seznam

Typy spojových seznamů podle struktury seznamu

- seznam obsahuje jen ukazatel na první prvek
- seznam obsahuje ukazatele na první prvek a na poslední prvek
- seznam obsahuje tzv. zarážku (speciální poslední prvek, který nenese data)

Typy spojových seznamů podle zřetězení

- jednosměrně zřetězený
- obousměrně zřetězený ... každý prvek obsahuje:
  - vlastní data
  - ukazatel na další prvek
  - ukazatel na předchozí prvek

Spojový seznam

Příklad: Vytvořte jednosměrně zřetězený spojový seznam se zarážkou. Implementujte následující operace:

- vytvoření prázdného seznamu
- vyprázdnění seznamu
- zrušení seznamu
- test prázdnosti seznamu
- výpis obsahu seznamu
- přidání prvku
  - na začátek seznamu
  - na konec seznamu
  - za zadaný prvek
- nalezení prvku s požadovanými daty
- smazání (vyjmutí)
  - prvního prvku seznamu
  - posledního prvku seznamu
  - zadaného prvku

Spojový seznam

Další operace nad seznamem (pokud každý prvek má klíč a hodnotu)

- nalezení prvku s minimálním / maximálním klíčem
- setřídění prvků seznamu podle klíče
- ...

Spojový seznam ... definice datové struktury

```
// definuji typ hodnot ukladanych do seznamu:
using T = int;
// using T =  Zamestnanec;

struct Prvek
{
    T data;
    Prvek *dalsi=nullptr;
};

struct Seznam
{
    Prvek *hlava = nullptr;
    Prvek *zarazka = nullptr;
};
...
```

Spojový seznam ... definice datové struktury

```
struct Zamestnanec
{
    unsigned int ID;
    string jmeno;
    string prijmeni;
    unsigned int plat;
};
// using T = int;
using T =  Zamestnanec;

struct Prvek
{
    T data;
    Prvek *dalsi=nullptr;
};

struct Seznam
{
    Prvek *hlava = nullptr;
    Prvek *zarazka = nullptr;
};
```

Spojový seznam ... funkce main()

```
/* pro seznam prvku typu int */
int main()
{
    Seznam s;
    vytvor(s);

    vlozNaZacatek(s, 4);
    vypis(s);

    vyprazdni(s);
    if (jePrazdny(s))
        cout << "seznam je prazdny\n";

    vlozNaKonec(s, 5);
    vlozNaZacatek(s, 8);
    vypis(s);

    zrus(s);
    return 0;
}
```

Spojový seznam ... funkce main()

```
/* pro seznam prvku typu Zamestnanec */
int main()
{
    Zamestnanec pepa = {101,"Josef","Novak",50000 };
    Zamestnanec maruska = {23,"Marie","Pilna",60000 };
    Zamestnanec tereza = {78,"Terezie","Spurna",20000 };

    Seznam s;
    vytvor(s);

    vlozNaZacatek(s, pepa);
    vypis(s);
    vlozNaKonec(s, maruska);
    vlozNaZacatek(s, tereza);
    vypis(s);

    zrus(s);
    return 0;
}
```

Spojový seznam ... prvni funkce

```
void vytvor(Seznam &s);
void vyprazdni(Seznam &s);
void zrus(Seznam &s);

bool jePrazdny(Seznam &s);

void vlozNaZacatek(Seznam &s, const T&co);
void vlozNaKonec(Seznam &s, const T &co);

void vypis(Seznam &s);
```

Spojový seznam ... vytvor a zrus

```
void vytvor(Seznam &s)
{
    // vytvor "prazdny" seznam (= seznam s jedinnym prvkem,
    //  zarazkou)
    // 1. do hlavy vloz novy prvek (pomoci operatoru new)
    // 2. do zarazky vloz ukazatel na tento prvek
    // 3. naslednikem zarazky bude nullptr
}
void vyprazdni(Seznam &s)
{
    // 1. dokud seznam neni "prazdny",
    // 1.1. adresu hlavy uloz do pomocneho ukazatele
    // 1.2. posun hlavu na dalsi prvek
    // 1.3. puvodni hlavu odstran (pomoci operatoru delete)
}
void zrus(Seznam &s)
{
   // 1. vyprazdni seznam
   // 2. odstran zarazku (operator delete)
   // 3. nastav vsechny parametry na 0
}
```

Spojový seznam

```
bool jePrazdny(Seznam &s)
{
    // seznam je prazdny, pokud obsahuje prave jeden prvek
    // (zarazku)
}

void vlozNaZacatek(Seznam &s, const T &co)
{
   // 1. vytvor novy prvek (pomoci operatoru new)
   // 2. uloz do nej data (co)
   // 3. jako jeho naslednika nastav hlavu seznamu
   // 4. do hlavy seznamu uloz ukazatel na nove vytvoreny prvek
}

void vlozNaKonec(Seznam &s, const T &co)
{
   // 1. data (co) uloz do zarazky
   // 2. vytvor novy prvek jako naslednika zarazky
   // 3. presun zarazku na nove vytvoreny prvek
   // 4. naslednikem nove zarazky bude nullptr
}
```

Spojový seznam

```
void vypis(int x)
{
    cout << x << " ";
}

void vypis(const Zamestnanec&z)
{
    cout << z.ID << " " << z.jmeno << " " << z.prijmeni
         << " " << z.plat << endl;
}

void vypis(Seznam &s)
{
    // 1. adresu hlavy uloz do pomocneho ukazatele
    // 1. dokud se s timto ukazatelem nedostanes k zarazce:
    // 1.1. vypis obsah prvku (zavolej pomocnou funkci)
    // 1.2. posun pomocny ukazatel na dalsi prvek
}
```

Odbočka ... využití maker a direktiv preprocesoru k "zakomentování" části kódu:

```
// definuji makro:
#define MOJE_MAKRO
#define N 100
...
// rusim makro:
#undef MOJE_MAKRO
#undef N
```

hlavičkový soubor:

```
#define ZAMESTNANEC

// prikazy se provedou, jen je-li makro definovano:
#ifdef ZAMESTNANEC
 using T = Zamestnanec;
#else
 using T = int;
#endif
```

Odbočka ... využití maker a direktiv preprocesoru k "zakomentování" části kódu:

```
#ifdef ZAMESTNANEC
int main()
{
        Zamestnanec pepa = { 456,"Josef","Novak",50000 };
        Seznam s;
        vytvor(s);
        vlozNaZacatek(s, pepa);
        ...
        zrus(s);
        return 0;
}
#else
int main()
{
        Seznam s;
        vytvor(s);
        vlozNaZacatek(s, 16);
    ...
}
#endif
```

Spojový seznam ... pokračování

```
/* pro seznam prvku typu int */
...
Seznam s;
vytvor(s);
...
if(!jePrazdny(s))
    cout << vyjmiPrvni(s) << endl;
smazPrvni(s);
int *px = najdi1(s,6);
if (px)
{
   cout << "nalezeno " << *px << endl;
   *px = 16;
}
najdi2(s,8) = 18;
int y = 11;
Prvek *p = najdi(s,5);
vlozZa(s,p,y);
smazPrvekZa(s,p);
smazPrvek(s,najdi(s,5));
...
zrus(s);
```

Spojový seznam ... pokračování

```
T vyjmiPrvni(Seznam &s);
void smazPrvni(Seznam &s);

Prvek *najdi(Seznam &s, const T &co);
T *najdi1(Seznam &s, const T &co);
T &najdi2(Seznam &s, const T &co);

void vlozZa(Seznam &s, Prvek *zaKtery, const T &co);
T vyjmiPrvekZa(Seznam &s, Prvek *predchazejici);
void smazPrvekZa(Seznam &s, Prvek *predchazejici);

void smazPrvek(Seznam &s, Prvek *mazany);

T* maximum(Seznam& s);
void setrid(Seznam &s);
```

Spojový seznam ... vyjmutí / smazání prvního prvku

```
T vyjmiPrvni(Seznam &s)
{
    // 1. over, ze seznam neni prazdny
    //      (pokud je, hod vyjimku nebo vrat spec. hodnotu)
    // 2. adresu hlavy uloz do pomocneho ukazatele
    // 3. posun hlavu na nasledujici prvek
    // 4. data z puvodni hlavy uloz do pomocne promenne
    // 5. puvodni hlavu odstran (pomoci operatoru delete)
    // 6. vrat data
}

void smazPrvni(Seznam &s)
{
    // ...
}
```

Spojový seznam ... nalezení prvku s požadovanými daty

```
Prvek *najdi(Seznam &s, const T &co);
T *najdi1(Seznam &s, const T &co);
T &najdi2(Seznam &s, const T &co);
bool najdi3(Seznam &s, const T &co);

// pro strukturu (Zamestnanec) ma vetsi smysl:
Prvek *najdi(Seznam &s, unsigned int ID);
T *najdi1(Seznam &s, unsigned int ID);
T &najdi2(Seznam &s, unsigned int ID);
bool najdi3(Seznam &s, unsigned int ID);

Prvek *najdi(Seznam &s, string prijmeni);
T *najdi1(Seznam &s, string prijmeni);
T &najdi2(Seznam &s, string prijmeni);
bool najdi3(Seznam &s, string prijmeni);
```

Spojový seznam ... nalezení prvku s požadovanými daty

```
Prvek *najdi(Seznam &s, const T &co)
{
    // 1. adresu hlavy uloz do pomocneho ukazatele
    // 2. dokud se s timto ukazatelem nedostanes k zarazce:
    // 2.1. pokud jsi nalezl hledany prvek, vrat ho
    // 2.2  posun pomocny ukazatel na dalsi prvek
    // 3. vrat nullptr (prvek nebyl nalezen)
}
T* najdi1(Seznam &s, const T &co)
{
    // ...
}

T& najdi2(Seznam &s, const T &co)
{
    // ...
}

bool najdi3(Seznam &s, const T &co)
{
    // ...
}
```

Spojový seznam ... nalezení prvku s požadovanými daty

```
T* najdi1(Seznam &s, const T &co)
{
    Prvek *p = najdi(s, co);
    if (p)
        return &(p->data);
    return nullptr;
    // return (p) ? &(p->data) ? nullptr;
}

T& najdi2(Seznam &s, const T &co)
{
    Prvek *p = najdi(s, co);
    if (p)
        return (p->data);
    throw 1; //throw exception("Prvek v seznamu nenalezen.");
}

bool najdi3(Seznam &s, const T &co)
{
    return najdi(s, co);
}
```

Spojový seznam ... nalezení prvku s požadovanými daty

```
/* Hledani v seznamu Zamestnancu:
   aby slo pouzit Prvek *najdi(Seznam &s, const T &co), je treba
   nejak definovat operator == pro Zamestnance, napriklad: */
bool operator == (const Zamestnanec &x, const Zamestnanec &y)
{
    return (x.ID == y.ID);
}

/* Hledani v seznamu Zamestnancu podle ruznych klicu analogicky:
   (rozdil bude jen v bodu 2.1) */
Prvek *najdi(Seznam &s, unsigned int ID)
{
   // 1. adresu hlavy uloz do pomocneho ukazatele
   // 2. dokud se s timto ukazatelem nedostanes k zarazce:
   // 2.1. pokud jsi nalezl hledany prvek, vrat ho
   // 2.2  posun pomocny ukazatel na dalsi prvek
   // 3. vrat nullptr (prvek nebyl nalezen)
}

Prvek *najdi(Seznam &s, string prijmeni);
```

Spojový seznam ... vkládání a vyjmutí prvku

```
void vlozZa(Seznam &s, Prvek *zaKtery, const T &co)
{
    // 0. proved kontrolu prvku zaKtery
    //       (neni to nullptr ani zarazka)
    // 1. vytvor novy prvek a vloz do nej data
    // 2. do jeho polozky dalsi vloz ukazatel na naslednika
    //    prvku zaKtery
    // 3. nasledovnikem prvku zaKtery bude novy prek

}
T vyjmiPrvekZa(Seznam &s, Prvek *predchazejici)
{
    // 0. proved kontrolu prvku predchazejici
    //      (neni to nullptr, zarazka ani posledni prvek seznamu)
    // 1. adresu mazaneho prvku uloz do pomocneho ukazatele
    // 2. novym nasledovnikem prvku predchazejici bude naslednik
    //    mazaneho prvku
    // 3. data mazaneho prvku uloz do pomocne promenne
    // 4. odstran mazany prvek (operator delete)
    // 5. vrat data
}
```

Spojový seznam ... vyjmutí / smazani prvku

```
void smazPrvekZa(Seznam &s, Prvek *predchazejici)
{
    try
    {
        vyjmiPrvekZa(s,predchazejici);
    }
    catch (const int& x)
    {
        // cout << "Mazany prvek nenalezen";
        return;
    }
}
```

Spojový seznam ... vyjmutí / smazani prvku

```
void smazPrvek(Seznam &s, Prvek *mazany)
{
    // 0. proved kontrolu prvku mazany:
    //       pokud je to nullptr ani zarazka => konec
    //       (popr. hod vyjimku)
    // 1. pokud je naslednik mazaneho prvku zarazka:
    // 1.1. odstran puvodni zarazku (operator delete)
    // 1.1. vloz do zarazky adresu prvku mazany
    // 2. jinak:
    // 2.1. prekopiruj data z naslednika mazaneho prvku do mazaneho
    // 2.2. smaz nasledovnika mazaneho (zavolej smazPrvekZa())
}
```

Spojový seznam ... nalezeni minima/maxima

```
T* maximum(Seznam& s)
{
    // 0. pokud je seznam prazdny, vrat nullptr
    // 1. adresu hlavy uloz do pomocneho ukazatele
    //    (hlava bude pocatecni kandidat na maximum)
    // 2. adresu naslednika hlavy vloz do jineho pomocneho
    //    ukazatele
    // 3. dokud se s timto ukazatelem nedostanes k zarazce:
    // 3.1. pokud jsi nalezl vetsi prvek
    // 3.1.1 bude to novy kandidat na maximum
    // 2.2  posun pomocny ukazatel na dalsi prvek
    // 3. vrat adresu dat obsazenych v kandidatovi na maximum
}
/* pro Zamestnance: analogicky (jedina zmena bude v 3.1)
  (popr. predefinuji operator > pro Zamestnance)
// T* nejbohatsi(Seznam& s);
```

Spojový seznam ... setrideni seznamu

```
// setrideni seznamu metodou bublinkoveho trideni
void setrid(Seznam &s);
{
    // 0. pokud je seznam prazdny, konec
    // 1. zaved si pomocnou promennou typu bool (indikator,
    //    zda v dane iteraci cyklu byly nejake prvky prohozeny)
    // 1. provadej nasledujici prikazy, dokud
    //    "v dane iteraci cyklu byly nejake prvky prohozeny":
    // 1.1. poznamenej si, ze (zatim) zadne prvky prohozeny nebyly
    // 1.1. adresu hlavy uloz do pomocneho ukazatele
    // 1.2. dokud se s timto ukazatelem nedostanes k zarazce:
    // 1.2.1. pokud je aktualni prvek vesti nez nasledovnik:
    // 1.2.1.1. prohod data v aktualnim prvku a v jeho nasledovniku
    //          ("stara znama" funkce prohod())
    // 1.2.2.2. poznamenej si, ze nejake prvky byly prohozeny
    // 1.2.2.   posun pomocny ukazatel na dalsi prvek
}
/* pro Zamestnance: analogicky (jedina zmena bude v 1.2.1)
   (popr. predefinuji operator > pro Zamestnance) */
```

Spojový seznam ... další funkce

```
// na procviceni:
void smazPosledni(Seznam &s);
T vyjmiPosledni(Seznam &s);

// varianty funkci pro Zamestnance:
Prvek* najdi(Seznam &s, string prijmeni);
T* najdi(Seznam &s, string prijmeni);
T* nejbohatsi(Seznam& s);
void setridDlePlatu(Seznam &s);
void setridDleID(Seznam &s);

/* zbyva: ulozeni seznamu do souboru a jeho nacteni ze souboru
*/
void ulozSeznam(string soubor);
void nactiSeznam(string soubor);
```