Programovací jazyk C++

Struktury

Abstraktní datové typy

Strukturované datové typy

- existují v jazyce C, nejde o novinku v C++
- struktura je heterogenní
  - proměnná typu struktura tedy obsahuje několik položek různých datových typů, které ovšem spolu logicky souvisejí
  - struktura umožňuje „společné“ pojmenování všech sdružených údajů, s nimiž se potom pohodlněji pracuje

Motivační příklad

- chceme v programu zpracovávat informace o bodech v rovině
  - každý bod má souřadnice [x, y]
  - naprogramujeme funkci, která vrací vzdálenost bodu od počátku

$$d = \sqrt{x^2 + y^2}$$

```
float vzdalenost(float x, float y)
{
  return sqrt(x*x+y*y);
}

int main()
{
  float xA,yA; // bod A
  float xB,yB; // bod B
  float d;
  xA = 3.5; yA = 2;
  d = vzdalenost(xA,yA);
}
```

Vadí nám na tom něco?

- bod chápeme jako dvojici souřadnic, ale toto „seskupení souřadnic“ ze zápisu programu nevidíme

Struktura

- deklarace proměnné bod typu struktura, která reprezentuje body v dvourozměrném prostoru:

```
struct {
   float x;
   float y;
} bod;
```

- deklarace nového typu `TBod`:

```
typedef struct {
 float x;
 float y;
} TBod;
```

- deklarace proměnné `bod1`, která je typu `TBod`:

```
TBod bod1;
```

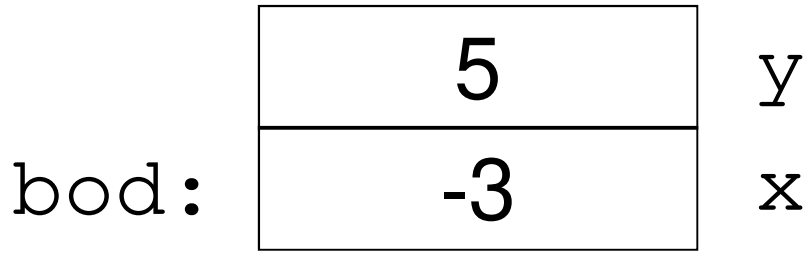
- deklarace pole „bodů“

```
TBod body[20];
```

- přístup k položkám se provádí pomocí tečkové notace:

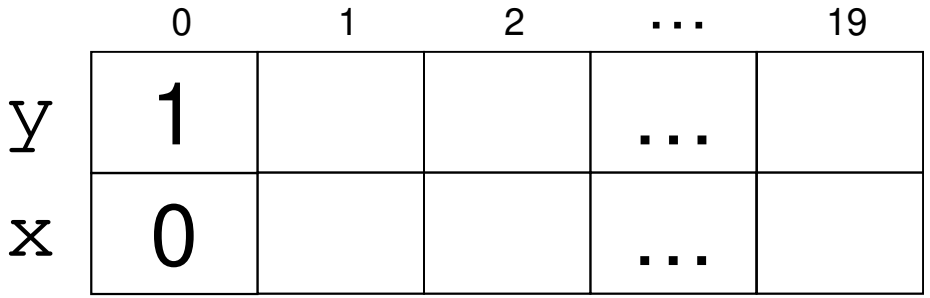
```
bod.x = -3; bod.y = 5;
bod1.x = 2; bod1.y = 0;
body[0].x= 0; body[0].y = 1;
```

- uložení v paměti:



- pole struktur `Bod body[20];`
  - uložení v paměti

`body:`



- stejně jako u pole je možné již při deklaraci inicializovat položky struktury (tzv. konstruktor):

```
TBod b1 = {3,-1};
```

- dynamická alokace proměnné typu struct v C:

```
TBod *b3;
b3 = (TBod*)malloc(sizeof(TBod));
```

resp. v C++

```
b3 = new TBod;
```

- přístup k položkám:

```
*b3.x = 5; *b3.y = 0;
```

nebo:

```
b3 -> x = 5;
```

- dynamická alokace pole struktur v C:

```
TBod *b4;
b4 = (TBod*)malloc(sizeof(TBod)*n);
```

resp. v C++

```
b4 = new TBod[n];
```

- přístup k položkám:

```
b4[0].x = 5; b4[0].y = 0;
```

- dealokace
  - v C:

```
free(b3);
free(b4)
```

  - v C++:

```
delete b3;
delete [] b4;
```

- do procedur a funkcí předáváme buď celé struktury
  - nevýhodné, uvědomme si, že se celá struktura kopíruje na zásobník
  - definice funkce

```
float vzdalenost(TBod b)
{
 return sqrt(b.x*b.x+b.y*b.y);
}
```

  - volání

```
vzdalenost(b1);
vzdalenost(*b3);
```

- nebo raději přes ukazatele
  - na zásobník se kopíruje pouze adresa, úsporné
  - definice funkce

```
float vzdalenost(TBod *b)
{
 return sqrt(b->x*b->x+b->y*b->y);
}
```

  - volání

```
vzdalenost(&b1);
vzdalenost(b3);
```

- strukturu jako parametr lze deklarovat také pomocí reference
  - předá se interně také adresa
  - definice funkce

```
float vzdalenost(TBod &b)
{
 return sqrt(b.x*b.x+b.y*b.y);
}
```

  - volání

```
vzdalenost(b1);
vzdalenost(*b3);
```

- parametr lze deklarovat také jako *konstantní parametr typu reference*
  - definice funkce

```
float vzdalenost(const TBod &b) {…}
```

  - volání je stejné

```
vzdalenost(b11);
vzdalenost(*b3);
```

- pak není možné ve funkci měnit položky struktury (překladač ohlásí chybu)
  - používá se jako ochrana, aby programátor omylem neměnil hodnotu parametru, který považuje za vstupní

Poznámka

- **struktura** odpovídá záznamu (record) v Pascalu
- existuje ještě datový typ **sjednocení (union)**
  - analogie variantního záznamu v Pascalu
  - prezentace je k dispozici ke stažení pro zájemce

Jiný příklad

```
typedef struct
{
   char znacka_vozidla[30];
   char RZ[10];
   int objem_valcu;
} Auto;

Auto auto1;
```

- přístup k položkám:

```
auto1.objem_valcu = 1221;
strcpy(auto1.znacka_vozidla,"Škoda");
strcpy(auto1.RZ,"1A1 01 01");
```

- v klasickém C musíme je řetězec reprezentován pole znaků, nelze uložit text do pole pomocí přiřazení, musíme použít knihovní funkci `strcpy`

- naštěstí máme v C++ typ **`string`**

```
typedef struct
{
  string znacka_vozidla;
  string RZ;
  int objem_valcu;
} TAuto;

TAuto auto1;

auto1.objem_valcu = 1221;
auto1.znacka_vozidla = "Škoda";
auto1.RZ = "1A1 01 01";
```

Uložení datumu jako struktura

```
typedef struct
{
   int den;
   int mesic;
   int rok;
} Datum;

Datum datum_narozeni;
datum_narozeni.den = 5;
datum_narozeni.mesic = 12;
```

Položkou struktury může být struktura

```
typedef struct
{
   string jmeno;
   string prijmeni;
   Datum dat_nar;
   ...
} Osoba;
```

Položkou struktury může být struktura

```
Osoba student;
student.jmeno = "Vit";
student.dat_nar.den = 5;
student.dat_nar.mesic = 5;
```

Příklad – formát BMP

- grafický formát pro uložení obrázků bmp (Bitmap) obsahuje úvodní hlavičku o délce 14 bytů.

Položka	Délka položky	Popis
typ	2 byty	2 znaky „BM“
velikost	4 byty	celková velikost souboru
rezervováno1	2 byty	rezervováno pro pozdější použití, musí být na 0
rezervováno2	2 byty	rezervováno pro pozdější použití, musí být na 0
posun	4 byty	posun obrazových dat od začátku této hlavičky (struktury)

- předpokládejme, že pracujeme na platformě Intel
  - uložení dat little-endian, tj. slabiky nižšího řádu jsou uloženy na nižších adresách),
  - `sizeof(`**`unsigned short`**`) ==2`
  - `sizeof(`**`unsigned int`**`) == 4`

- strukturu, která odpovídá hlavičce souboru, nadeklarujeme následovně:

```
typedef struct
{
 unsigned char B;
 unsigned char M;
 unsigned int velikost;
 unsigned short res1;
 unsigned short res2;
 unsigned int posun;
}  THeadBMP;
```

- nadeklarujeme proměnnou pro hlavičku:

```
THeadBMP hlavicka;
```

- hlavičku načteme ze soubor vst, který byl otevřen pomocí fopen v *binárním* módu:

```
fread((void*)&hlavicka,sizeof(THeadBMP),1,vst)
```

- *upozornění:*
  - některé překladače v rámci optimalizace přístupu do paměti neukládají položky struktur těsně za sebou, ale zarovnávají je na adresy dělitelné 4. V takovém případě by mezi položkami M a velikost byla v paměti mezera a velikost struktury by byla větší než velikost hlavičky. Museli bychom tedy v překladači tuto optimalizaci (zarovnání položek na adresy dělitelné 4) vypnout.

Úloha 1

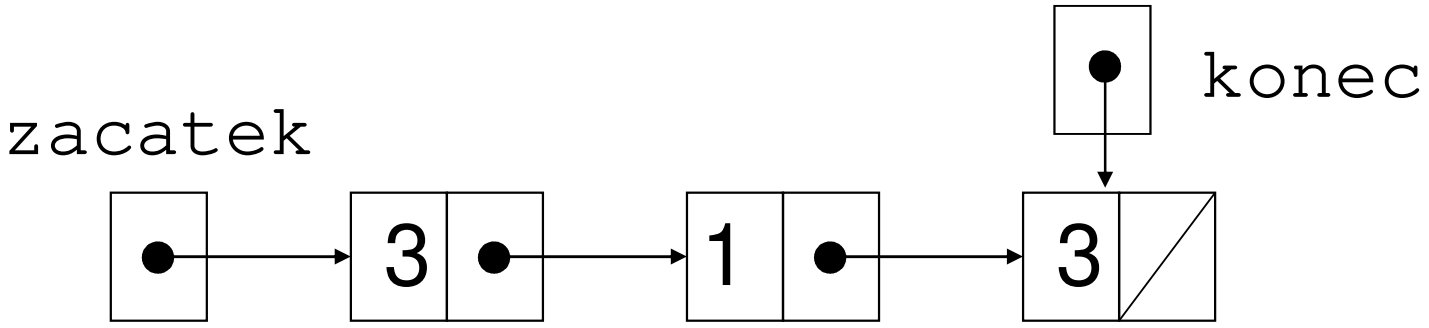
Napište program, kde deklarujete strukturovaný typ pro uložení bodu v prostoru. Napište dvě funkce, jedna vrací vzdálenost bodu od počátku (inspirujte se funkcí vzdalenost na snímcích 15 a 16), druhá funkce má dva body jako parametry a vrací vzdálenost mezi nimi. V hlavním programu deklarujte dva body (dvě proměnné Vašeho typu), jejichž souřadnice načtete z klávesnice. Ověřte chování naprogramovaných funkcí.

- pokud se při deklaraci struktury odkazujeme na ni, napíšeme jméno ještě před „{“

```
typedef struct TPolozka {
  int cislo;
  TPolozka *dalsi;
} TPolozka;

TPolozka *seznam;
```

Spojový seznam

- abstraktní datový typ (data + operace)
- určen pro dynamický seznam (lexikon), kde není znám počet prvků seznamu a jejich počet se často mění (vkládání, mazání)



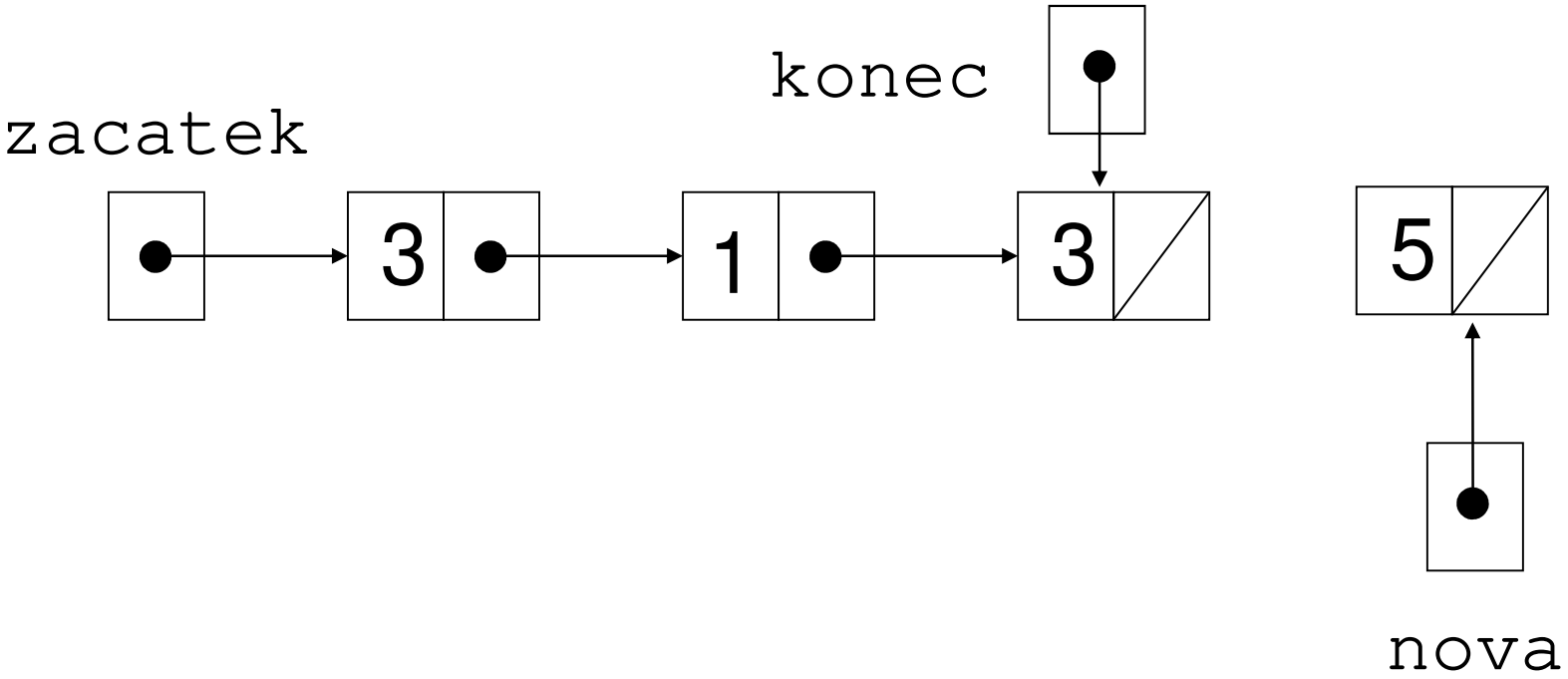
- implementace v C:

```
typedef struct TPolozka {
  int prvek;
  TPolozka *dalsi;
} TPolozka;

typedef struct {
  TPolozka *zacatek;
  TPolozka *konec;
} TSeznam;
```

```
TSeznam sez1;

void Init(TSeznam *seznam)
{
  seznam -> zacatek = NULL;
  seznam -> konec = NULL;
}
```

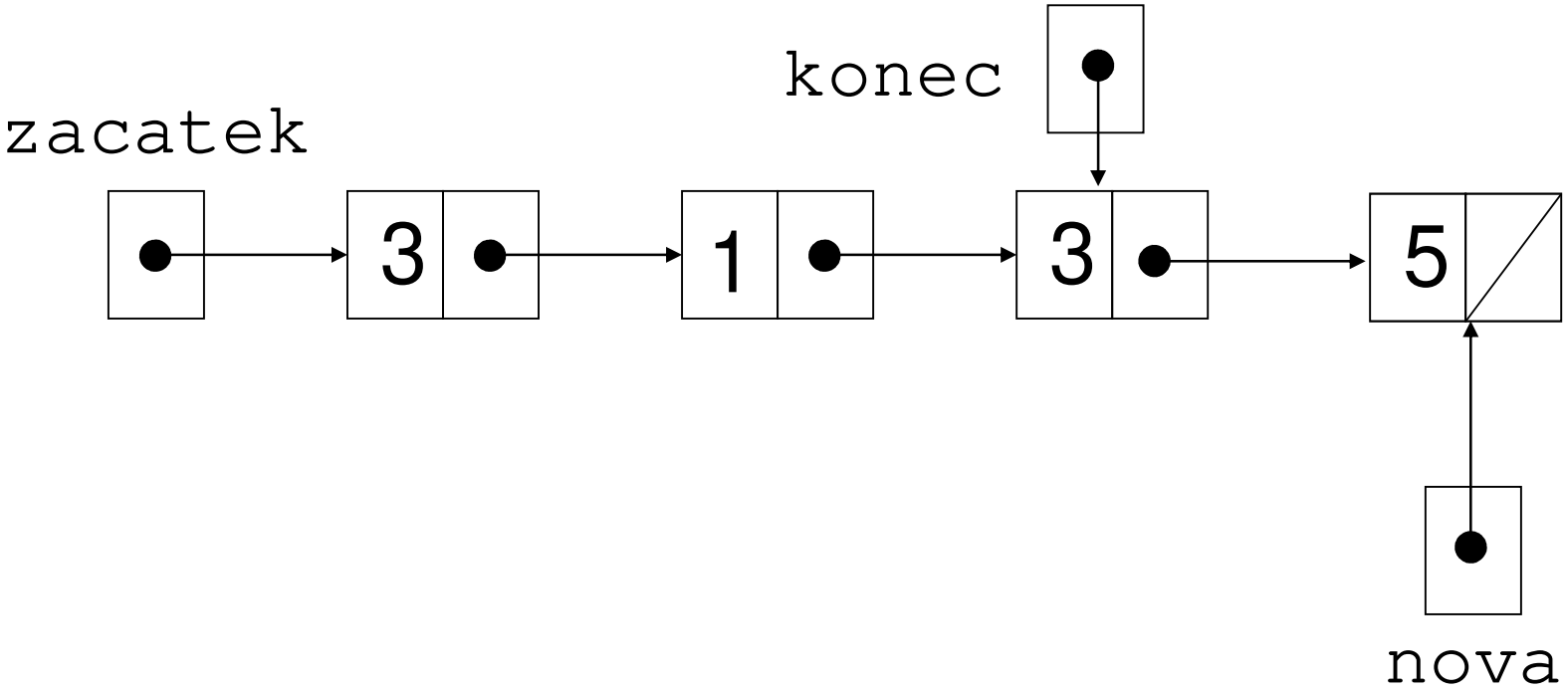
Vložení na konec seznamu

1) dynamicky vytvořím novou položku
   (ukazatel na další položku nastavím na NULL)



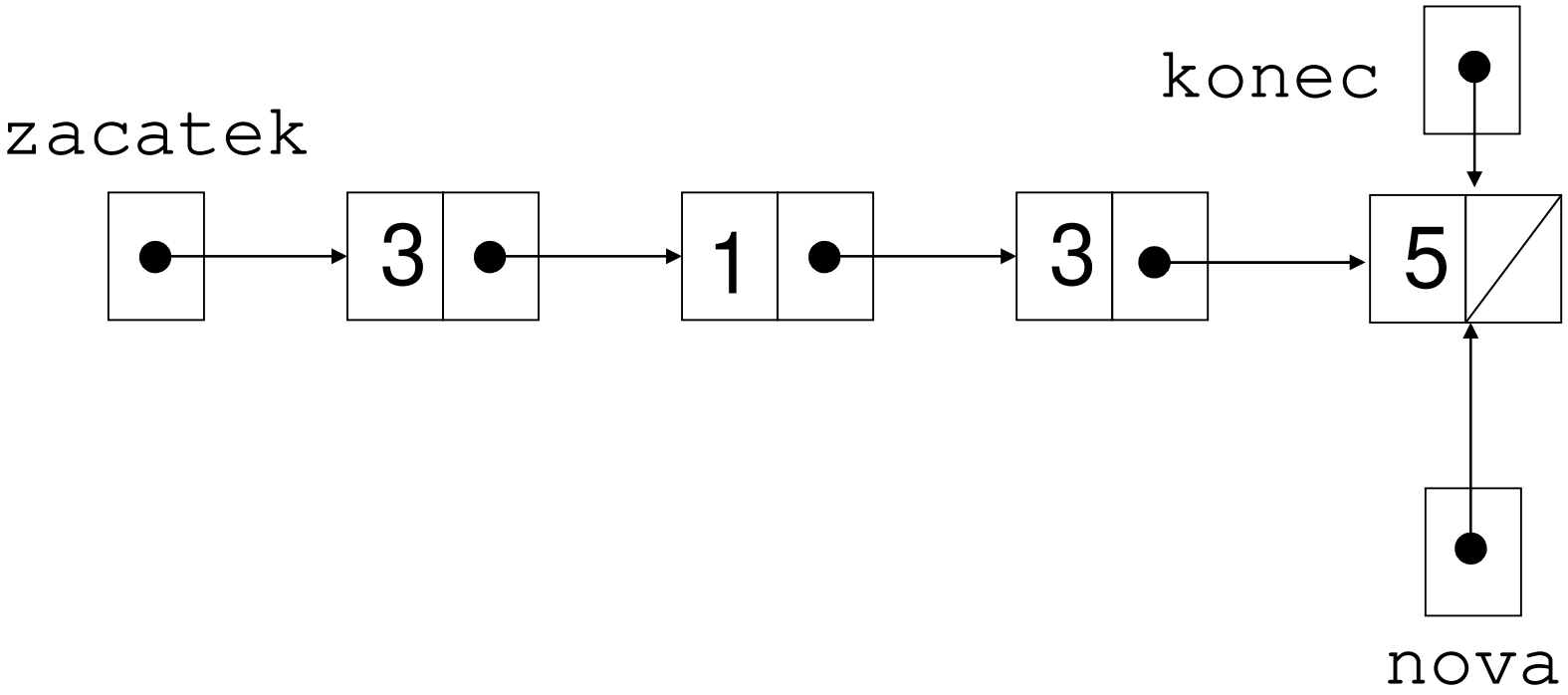
Vložení na konec seznamu

2) ukazatel `dalsi` u posledního prvku seznamu nastavím na novou položku



Vložení na konec seznamu

3) posunu ukazatel na konec seznamu

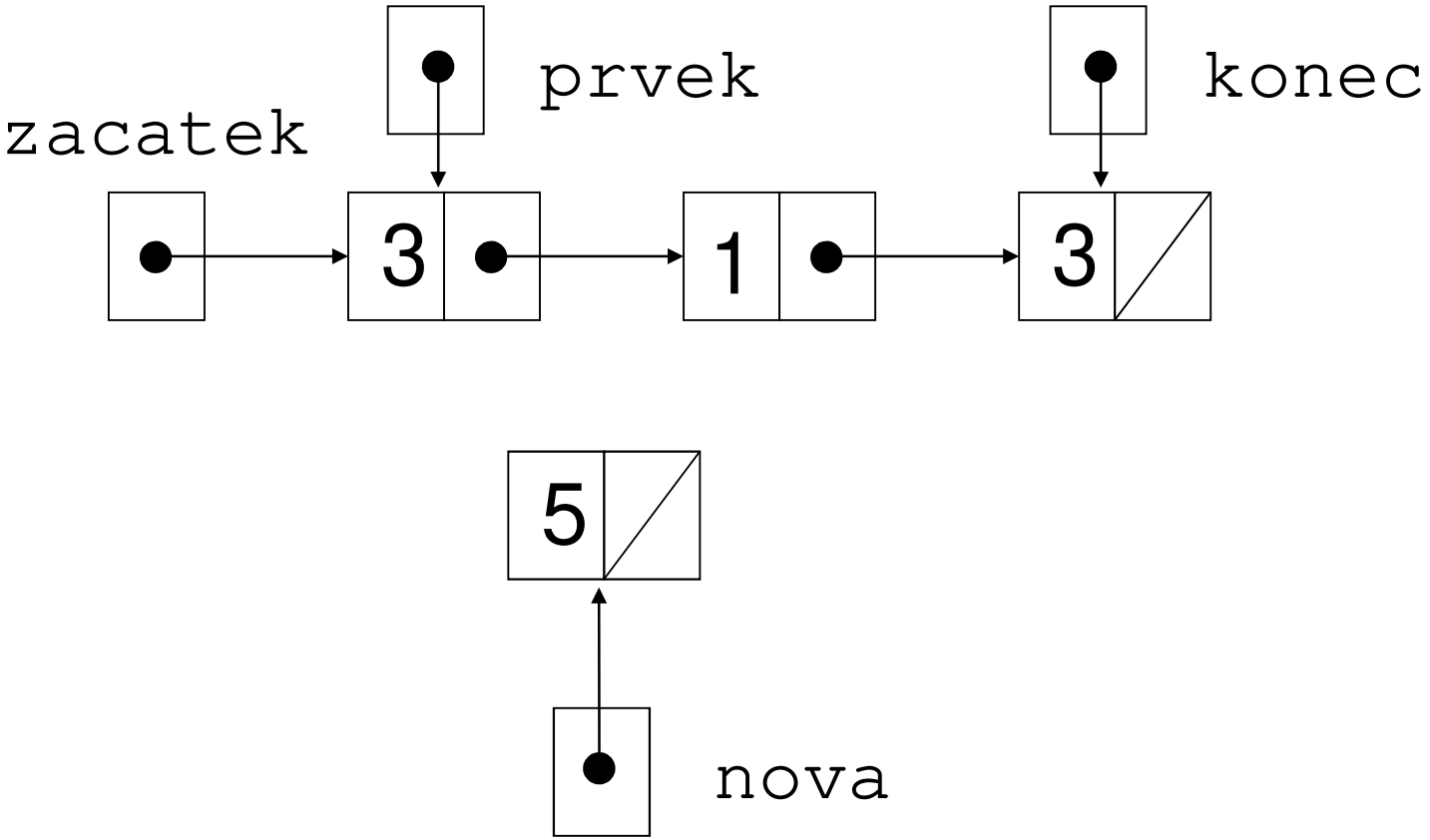


Pozor, vkládám-li do prázdného seznamu

```
void Vloz_na_konec(TSeznam *seznam, int
cislo)
{
  TPolozka *nova;
  nova = new TPolozka;
  nova -> prvek = cislo; nova -> dalsi=NULL;
  if (seznam -> zacatek != NULL)
  {
    seznam -> konec -> dalsi = nova;
    seznam -> konec = nova;
  }
  else { seznam -> konec = nova;
         seznam -> zacatek = nova; }
}
```

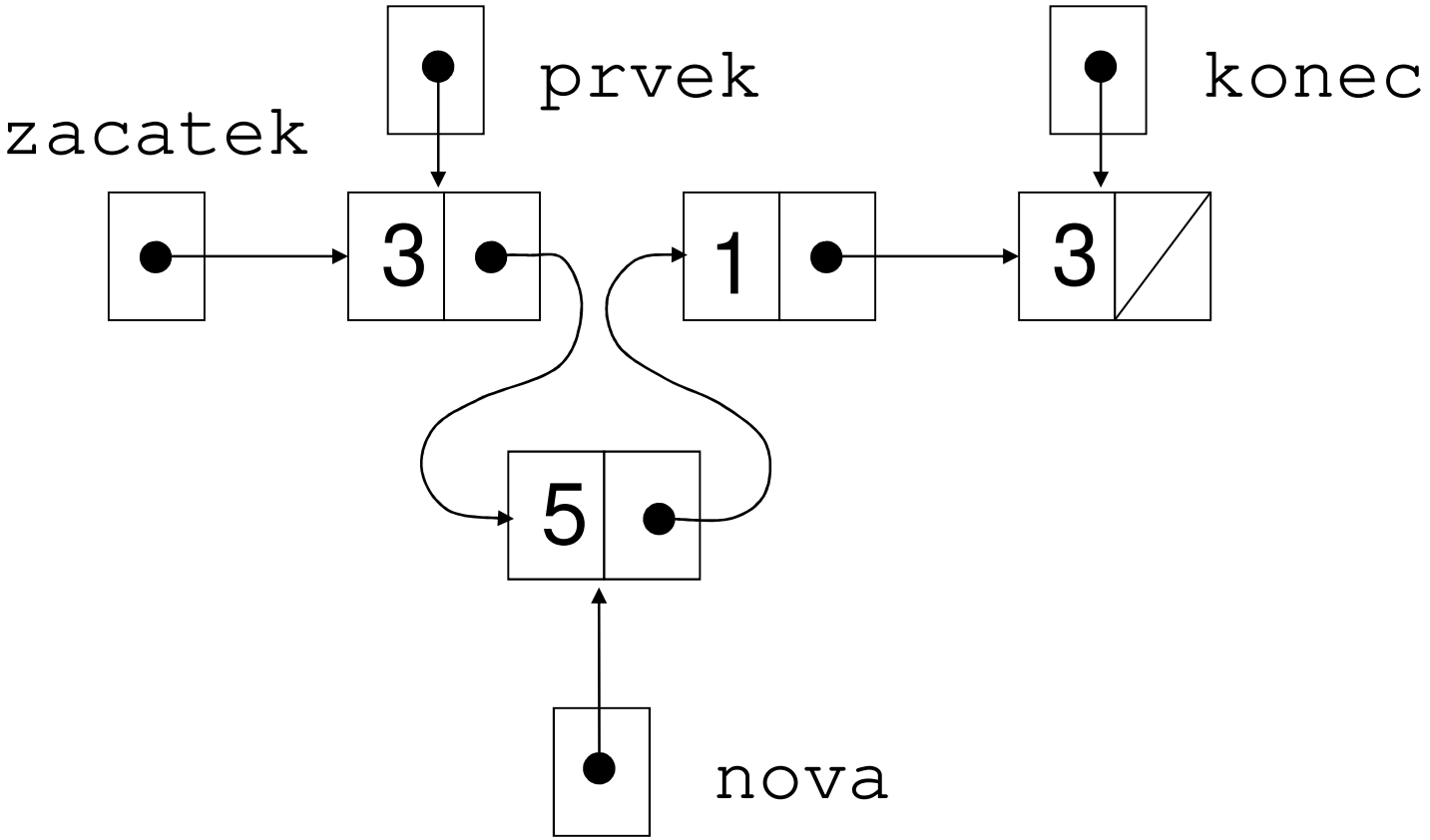
Vložení do seznamu za prvek

1) dynamicky vytvořím novou položku



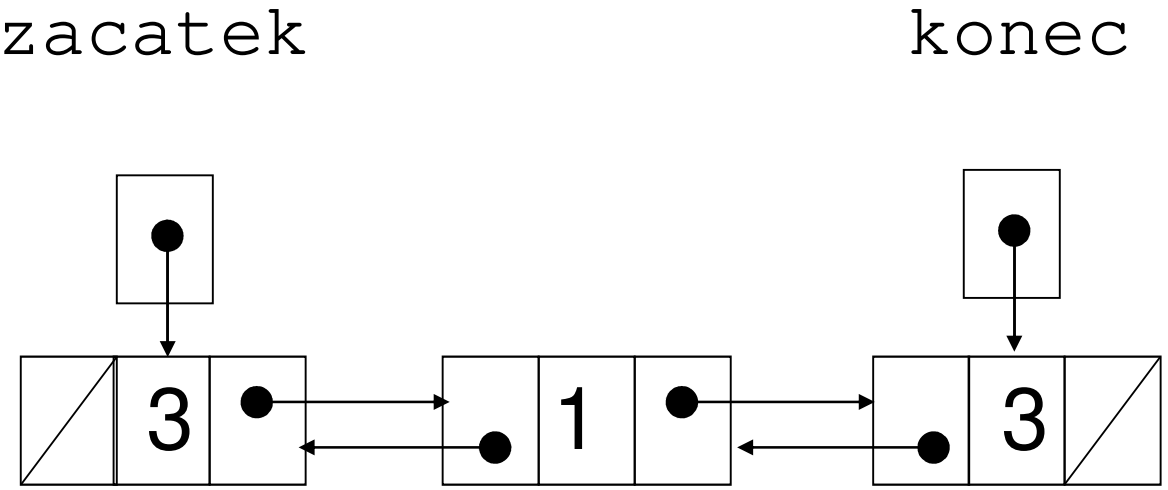
Vložení do seznamu za prvek

2) prováži nový prvek se seznamem



Obousměrný spojový seznam

- položka obsahuje ukazatel na předchůdce i následníka



Úloha 2

- naimplementujte spojový seznam (jednosměrně nebo obousměrně vázaný), do kterého se ukládají jména a telefonní čísla
- naimplementujte tyto procedury a funkce:
  - inicializace prázdného seznamu
  - test, zda je seznam prázdný
  - vložení záznamu na konec seznamu
  - vložení záznamu za prvek (parametrem je ukazatel na prvek, za který se vkládá, a nové jméno a telefonní číslo)

- hledání záznamu (podle jména i telef. čísla) - vrací ukazatel na záznam, pokud není v seznamu, vrací NULL
- zjištění ukazatele na první záznam
- zjištění ukazatele na následující záznam
- výmaz prvku (parametrem je ukazatel na existující prvek, který se má vymazat)
- zrušení celého seznamu

- napište jednoduchou konzolovou aplikaci pro otestování implementace seznamu; data zadávejte z klávesnice
  - nemusíte využít všechny funkce