Dynamické datové struktury

- **Dynamické datové struktury**
  - slouží pro reprezentaci proměnlivého počtu prvků proměnlivé velikosti
  - složeny z jednotlivých dynamicky alokovaných prvků navzájem propojených ukazateli
  - definuje se pouze datový typ prvku
  - celá struktura je reprezentována jedním ukazatelem, na který jsou „pověšeny“ všechny vzájemně provázané prvky

- **Výhody**
  - velká pružnost struktur – vznikají za běhu programu a mohou měnit velikost podle potřeby
  - snadná změna vnitřního uspořádání prvků
  - dokáží věrně postihnout i dosti složité struktury

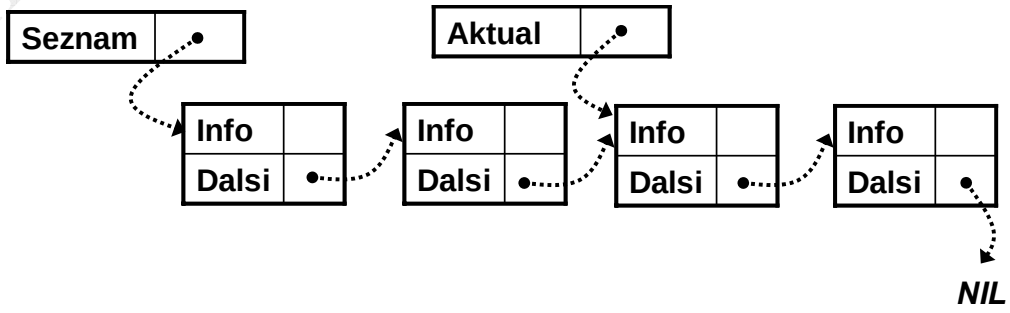
- **Nevýhody**
  - méně pohodlný přístup k jednotlivým prvkům – lze vstoupit do struktury pouze jediným hlavním ukazatelem
  - přístupové operace je nutné zvlášť programovat
  - určitá paměťová režie na propojovací ukazatele

Spojové seznamy

Jednosměrný lineární seznam

- **Jednosměrný lineární seznam**
  - nejjednodušší dynamická datová struktura
  - skupina položek propojených ukazateli do řady za sebou
  - každá položka nese kromě vlastní informace (***Info***) ještě ukazatel na svého následníka v seznamu - ***Dalsi***
  - celý seznam je zpřístupněn hlavním ukazatelem ***Seznam***, který ukazuje na první položku seznamu
  - poslední položka v seznamu má v ukazateli ***Dalsi*** hodnotu ***NIL*** (neukazuje nikam)
  - je definován aktuální prvek reprezentovaný ukazatelem ***Aktual***

- **Struktura seznamu:**

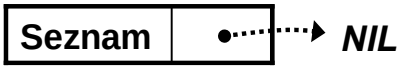


- **Deklarace seznamu:**

```
type UkPolozka = ↑Polozka;
Polozka = record
     Info : InfoTyp;
     Dalsi : UkPolozka;
end;
var Seznam, Aktual : UkPolozka
```

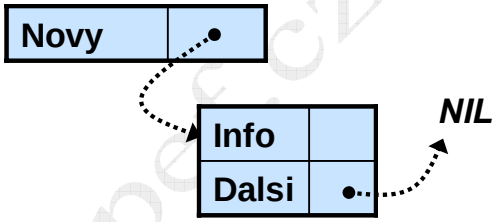
Operace s lineárním seznamem

- **Vytvoření prázdného seznamu**



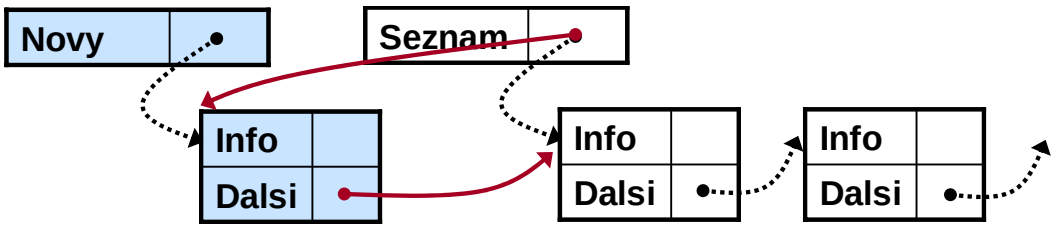

```
var Seznam : UkPolozka;
begin
     Seznam := nil;
end;
```

- **Vytvoření nového prvku**



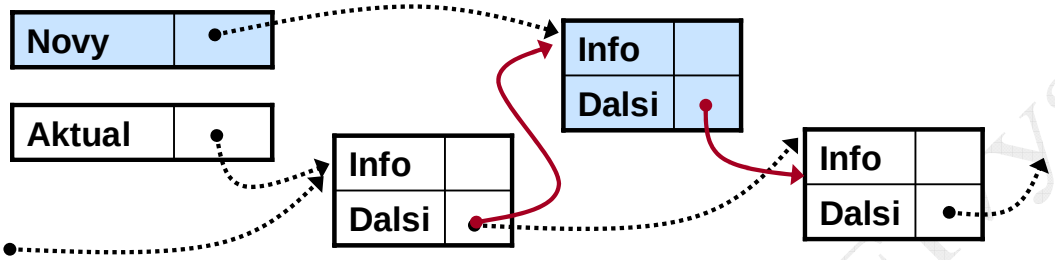

```
procedure VytvoritPrvek (var Novy:UkPolozka;
hodnota:InfoTyp);
var Uk:UkPolozka;
begin
     New(Uk);
     Uk↑.Info := Hodnota;
     Uk↑.Dalsi := nil;
     Novy := Uk;
end;
```

- **Vložení prvku na začátek seznamu**



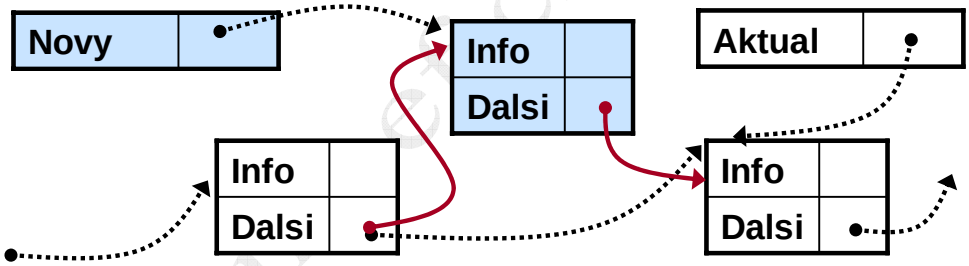
```
procedure VlozitNaZacatek(var Novy,Seznam:UkPolozka);
begin
     Novy↑.Dalsi := Seznam;
     Seznam := Novy;
end;
```

- Vložení prvku za aktuální prvek



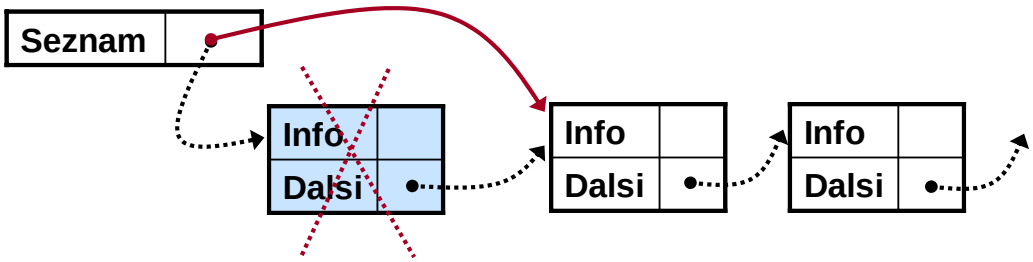

```
procedure VlozitZa(var Novy,Aktual:UkPolozka);
begin
     Novy↑.Dalsi := Aktual↑.Dalsi ;
     Aktual↑.Dalsi := Novy;
end;
```

- Vložení prvku před aktuální prvek



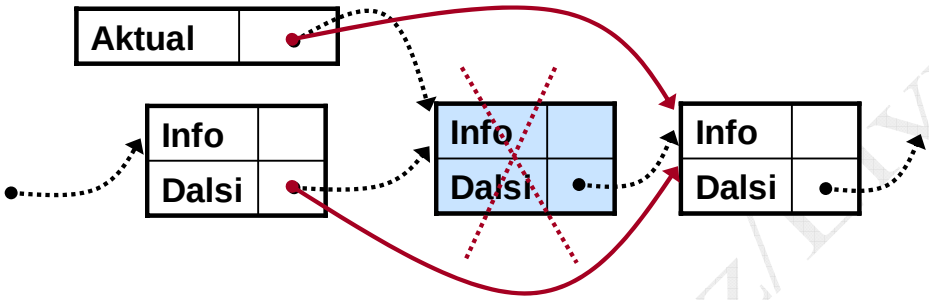

```
procedure VlozitPred (Novy,Aktual:UkPolozka;
var Seznam:UkPolozka);
var Za:UkPolozka;
begin
     Za := Seznam;
     while Za↑.Dalsi <> Aktual do Za := Za↑.Dalsi;
     VlozitZa (Novy, Za);
end;
```

- Vyřazení prvku na začátku seznamu



```
procedure VyradZacatek (var Seznam:UkPolozka);
var Uk:UkPolozka;
begin
     Uk := Seznam;
     Seznam := Seznam↑.Dalsi;
     Dispose (Uk);
end;
```

Vyřazení aktuálního prvku




```
procedure VyradAktual (var Aktual,Seznam:UkPolozka);
var Uk:UkPolozka;
begin
     Uk := Seznam;
     while Uk↑.Dalsi <> Aktual do Uk := Uk↑.Dalsi;
     Uk↑.Dalsi := Aktual↑.Dalsi; Uk := Aktual;
     Aktual := Aktual↑.Dalsi;
     Dispose (Uk);
end;
```

Vypsání prvků seznamu

iterativní metoda

```
procedure I_Vypis (var Seznam:UkPolozka);
var Uk:UkPolozka;
begin
     Uk := Seznam;
     while Uk <> nil do begin
          writeln (Uk↑.Info);
          Uk := Uk↑.Dalsi;
     end;
end;
```

- rekurzivní metoda

```
procedure R_Vypis (var Seznam:UkPolozka);
begin
     if Seznam <> nil then begin
          writeln (Seznam↑.Info);
          R_Vypis(Seznam↑.Dalsi);
     end;
end;
```

Hledání hodnoty v seznamu

- iterativní metoda

```
function I_Hledej(Hodnota:Infotyp;
var Seznam:UkPolozka):UkPolozka;
var Uk:UkPolozka;
begin
     Uk := Seznam;
     while (Uk <> nil) and (Uk↑.Info <> Hodnota)
          do Uk := Uk↑.Dalsi;
     I_Hledej := Uk;
end;
```

- rekurzivní metoda

```
function R_Hledej (Hodnota:Infotyp;
var Seznam:UkPolozka):UkPolozka;
begin
     if (Seznam <> nil)and(Seznam↑.Info <> Hodnota) then
          R_Hledej:=R_Hledej(Seznam↑.Dalsi,Hodnota)
     else
          R_Hledej:=Seznam;
end;
```

Zrušení celého seznamu

- iterativní metoda

```
procedure I_Zrus (var Seznam:UkPolozka);
var Uk:UkPolozka;
begin
     while Seznam <> nil do begin
          Uk := Seznam;
          Seznam := Seznam↑.Dalsi);
          Dispose(Uk);
     end;
end;
```

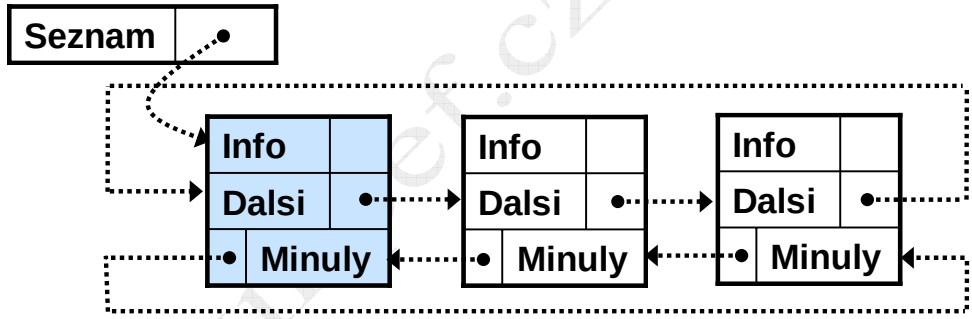
- **rekurzivní metoda**

```
procedure R_Zrus (var Seznam:UkPolozka);
begin
     if Seznam↑.Dalsi <> nil then R_Zrus(Seznam↑.Dalsi);
     Dispose(Seznam);
end;
```

Dvousměrný cyklický seznam s hlavou

- **Dvousměrný cyklický seznam s hlavou**
  - každá položka nese kromě vlastní informace (***Info***) dva ukazatele – na svého následníka (***Dalsi***) a předchůdce (***Minuly***)
  - následníkem poslední položky seznamu je začátek, předchůdcem počáteční položky je poslední prvek
  - do seznamu je přidána navíc jedna fiktivní položka, nazvaná ***Hlava***, která není z hlediska informací součástí seznamu, ale slouží pouze k usnadnění práce se seznamem (vyhledávání, …)
  - není nikdy prázdný, obsahuje vždy alespoň Hlavu

- **Struktura seznamu:**

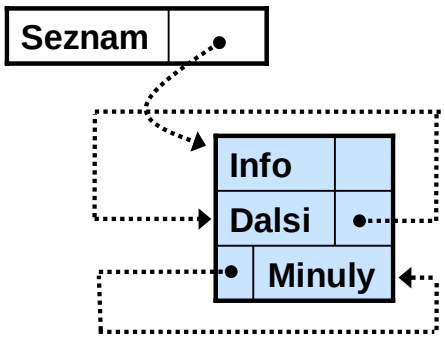


- **Deklarace seznamu:**

```
type UkPolozka = ↑Polozka;
Polozka = record
     Info : InfoTyp;
     Dalsi,Minuly : UkPolozka;
end;
var Seznam : UkPolozka
```

Operace s cyklickým seznamem

- **Vytvoření prázdného seznamu**

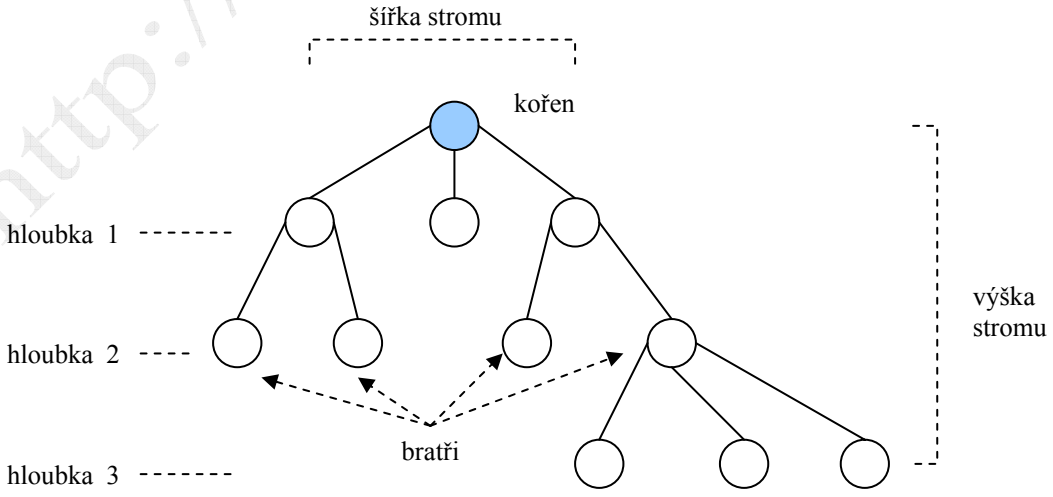



```
var Seznam : UkPolozka;
begin
  New (Seznam);
  Seznam↑.Dalsi := Seznam;
  Seznam↑.Minuly := Seznam;
end;
```

- **Ostatní operace jsou obdobné jako u jednosměrného seznamu**

Stromy a grafy

- **Strom T** je konečná množina jednoho nebo více prvků (**uzlů**), z nichž jeden je označen jako ***r*** **– kořen** (*root*) a zbývající uzly jsou rozděleny do $k \geq 0$ disjunktních podmnožin $T_1, T_2, \ldots T_k$, které jsou také stromy a jejichž kořeny $r_1, r_2, \ldots, r_k$ jsou následníky kořene ***r***.
  - Strom je graf, ve kterém existuje pouze jedna cesta z uzlu ***r*** (kořene), do kteréhokoliv dalšího uzlu grafu (neobsahuje cykly)



- Uzel, kterým se vstupuje do celého stromu (**kořen**) nemá žádného předchůdce
- Uzly, které nemají žádné syny se nazývají **listy**
- Cesta od kořene k některému z uzlů se nazývá **větev**
- Délka (počet hran) maximální větve se nazývá **výška stromu**
- Počet následovníků (synů) kořene se nazývá **šířka stromu**
- Délka větve (počet hran) od kořene k uzlu udává **hloubku uzlu**
- Uzly se stejnou hloubku (počtem hran od kořene) se nazývají **bratři**

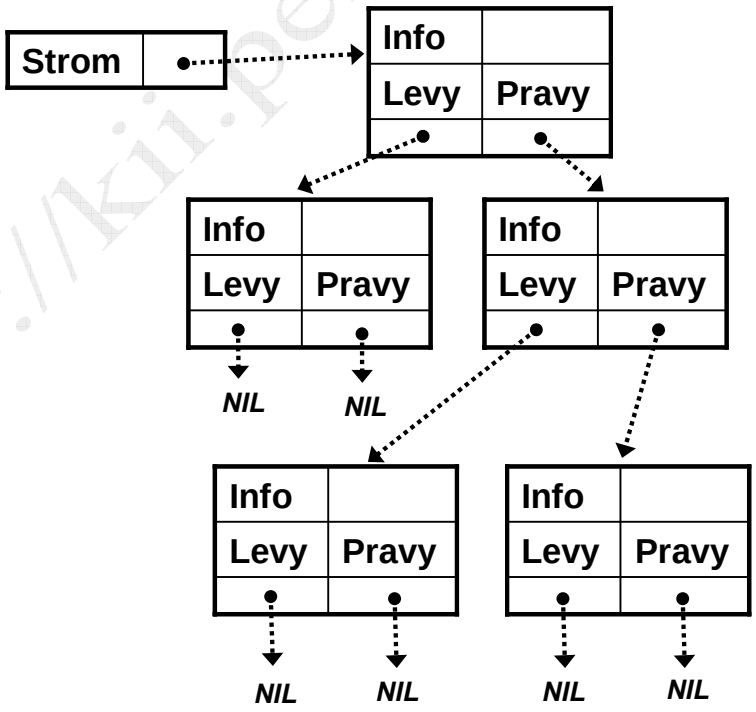
Binární strom

- **Binární strom**
  - skládá se z **uzlů** a **hran** (reprezentovány ukazateli)
  - každý **uzel** nese kromě vlastní informace (***Info***) dva ukazatele na své syny (***Levy*** a ***Pravy***)

- **Deklarace stromu:**

```
type UkUzel = ↑Uzel;
Uzel = record
     Info : InfoTyp;
     Levy,Pravy : UkUzel;
end;
var Strom : UkUzel;
```

- **Struktura binárního stromu:**



Graf

- **Graf**
  - skládá se z **uzlů** a **hran** (reprezentovány ukazateli)
  - každý **uzel** nese kromě vlastní informace (***Info***) ukazatele na jeden nebo více jiných uzlů (***Hrana_1 … Hrana_N***)
  - ***Graf*** je reprezentován ukazatelem na začátek seznamu uzlů
  - pro **orientovaný graf** zařazujeme do seznamu hran uzlu $\mathbf{U_i}$ jen hrany, pro které je uzel $\mathbf{U_i}$ počátečním uzlem
  - pro **ohodnocený graf** obsahuje každá hrana i ohodnocení (***H***)

- **Deklarace grafu:**

```
type UkHrana = ↑THrana;
     UkUzel = ↑TUzel;
     TUzel = record
          Info:InfoTyp;
          Hrana: UkHrana;
          DalsiU: UkUzel;
     end;
     THrana = record
          Hod : HodTyp;
          Uzel: UkUzel;
          DalsiH: UkHrana;
     end;
var Graf : UkUzel;
```

- **Struktura grafu:**