# Anotace

- Spojové seznamy,
- fronta, zásobník,
- uspořádaný spojový seznam,
- samoopravující seznamy,
- stromové datové struktury.

# Typologie spojových seznamů

- jednosměrný a obousměrný – prvek ukazuje jen na následníka, nebo na předka i následníka,
- s hlavou, bez hlavy, s ocasem, bez ocasu – hlava je první (prázdný) prvek, ocas je poslední (prázdný prvek). Vždy máme aspoň nějaký prvek seznamu.
- cyklický – poslední prvek si jako na následníka ukazuje na začátek seznamu.

# Fronta a zásobník

- Fronta je datová struktura organizující prvky způsobem FIFO.
- Je osazena funkcemi `enqueue` a `dequeue`.
- Zásobník organizuje prvky způsobem LIFO.
- Je vybaven funkcemi `push` a `pop`.
- Jak je implementovat pomocí spojových seznamů?

# Zásobník

## Implementace

```
type pzas=^zas;
zas=record
      hod:integer;
      next:pzas;
end;
var hlava:pzas;
procedure init;
begin
      hlava:=nil;
end;
```

# Zásobník

## Implementace

```
type pzas=^zas;
zas=record
      hod:integer;
      next:pzas;
end;
var hlava:pzas;
procedure push(co:integer);
var pom:pzas;
begin
      new(pom);
      pom^.hod:=co;
      pom^.next:=hlava;
      hlava:=pom;
end;
```

# Zásobník

## Implementace

```
function pop:integer;
var pom:pzas;
begin
      pom:=hlava;
      if hlava<>nil then
      begin pop:=hlava^.hod;
            hlava:=pom^.next;
            dispose(pom);
      end else
      begin writeln("Chyba!");
            pop:=-1;
      end;
end;
```

# Fronta

## Implementace

```
type pfr=^fronta;
fronta=record
      hod:integer;
      next:pfr;
end;
var hlava,ocas:pfr;
procedure init;
begin
      hlava:=nil;
      ocas:=nil;
end;
```

```
procedure enqueue(co:integer);
var pom:pfr;
begin if hlava=nil then
      begin new(hlava);
            ocas:=hlava;
            hlava^.next:=nil;
            hlava^.hod:=co;
      end else
      begin new(pom);
            pom^.next:=nil;
            pom^.hod:=co;
            hlava^.next:=pom;
            hlava:=pom;
      end;
end;
```

```
function dequeue:integer;
var pom:pfr;
begin if hlava=nil then
      begin dequeue:=-1;
      end else
      begin if hlava=ocas then
            begin dequeue:=ocas^.hod;
                  dispose(ocas);
                  hlava:=nil; ocas:=nil;
            end else
            begin dequeue:=ocas^.hod;
                  pom:=ocas;
                  ocas:=ocas^.next;
                  dispose(pom);
            end;
      end;
end;
```

# Prohození prvků ve spojovém seznamu

## Prohoď prvek s jeho následníkem v seznamu

```
procedure prohod(var hlava:seznam;co:seznam);
var pom:seznam;
begin pom:=hlava;
      if hlava=co then
      begin hlava:=hlava^.next;
            pom^.next:=hlava^.next;
            hlava^.next:=pom;
      end else
      begin while(pom^.next<>co) do
                  pom:=pom^.next;
            pom^.next:=co^.next;
            co^.next:=pom^.next^.next;
            pom^.next^.next:=co;
end; end;
```

# Dynamické datové struktury

- V příkladu je kvůli místu zanedbáno ošetření singulárních případů (prázdný seznam, prvek chybějící v seznamu, jednoprvkový seznam, prohazování posledního prvku a podobně). Vše je ale jen testování pointerů na `nil`.
- Cvičení na prohazování prvků ve spojovém seznamu: Bubblesort
- Organizace setříděného spojového seznamu (funkce insert, delete a member, které vkládají do setříděného spojového seznamu, mažou z něj a zjišťují, zda je hodnota ve spojovém seznamu.

# Uspořádaný spojový seznam

- Spojový seznam, ve kterém jsou prvky uspořádány v (BÚNO) neklesajícím pořadí.
- Nad spojovými seznamy typicky implementujeme funkce:
  - `member` – zjistí, zda je prvek s příslušným klíčem přítomen,
  - `insert` – přidá prvek s příslušným klíčem,
  - `delete` – smaže prvek s příslušným klíčem.
- Příklad viz web, resp. si ho napíšeme.

# Další datové struktury

- Samoopravující seznamy:
- Move-front rule, transposition rule:
- Pokud přistupujeme k prvku v seznamu, dotyčný prvek vytáhneme na začátek, resp. prohodíme s předchůdcem (myšlenka: pokud přistupujeme k prvku, zpravidla to děláme vícekrát za sebou).
- Má-li prvek více pointerů, vznikají stromové datové struktury.
- Binární vyhledávací strom (pokud možno co nejlépe vyvážený),
- Vyváženost se těžko vynucuje, proto AVL-stromy či červeno-černé stromy, kde se vyváženost vynucuje pomocí rotací.

# Reprezentace stromů

- Strom je (v této chvíli) datová struktura, ve které má každý vrchol nějaký počet synů a nejvýše jednoho rodiče.
- Vrchol bez rodiče je jen jeden a označujeme ho kořen.
- Ve stromě nesmějí vznikat cykly (ani slabé).
- Jak reprezentovat v Pascalu?
- Podobně jako spojový seznam, třeba takto:

```
type  strom:^vrchol;
      vrchol=record
               hod:longint;
               syn1:strom;
               syn2:strom;
               ...
      end;
```

# Binární vyhledávací strom

- Jedná se o binární strom, ve kterém pro každý vrchol všechny prvky v levém podstromě mají klíč menší než je klíč tohoto vrcholu a prvky v pravém podstromě mají klíč větší (než současný vrchol).
- V tomto stromě půjde snadno vyhledávat.
  Co výhody a nevýhody?
- Pokud se dobře postaví, je mnohem efektivnější, než spojový seznam.
- Problém je nepostavit ho špatně a při přidávání a ubírání nestrávit příliš času.
- Vyváženost odkazuje k tomu, jak moc se liší počty prvků v jednotlivých podstromech.