# VYČÍSLENÍ INFIX / POSTFIX

Infixová, prefixová (polská), postfixová (obrácená polská) notace
(Jan Lukasiewić)

- Infixová ⇨ a + b
- prefixová ⇨ + a b

prefixová notace připomíná zápis funkce..?

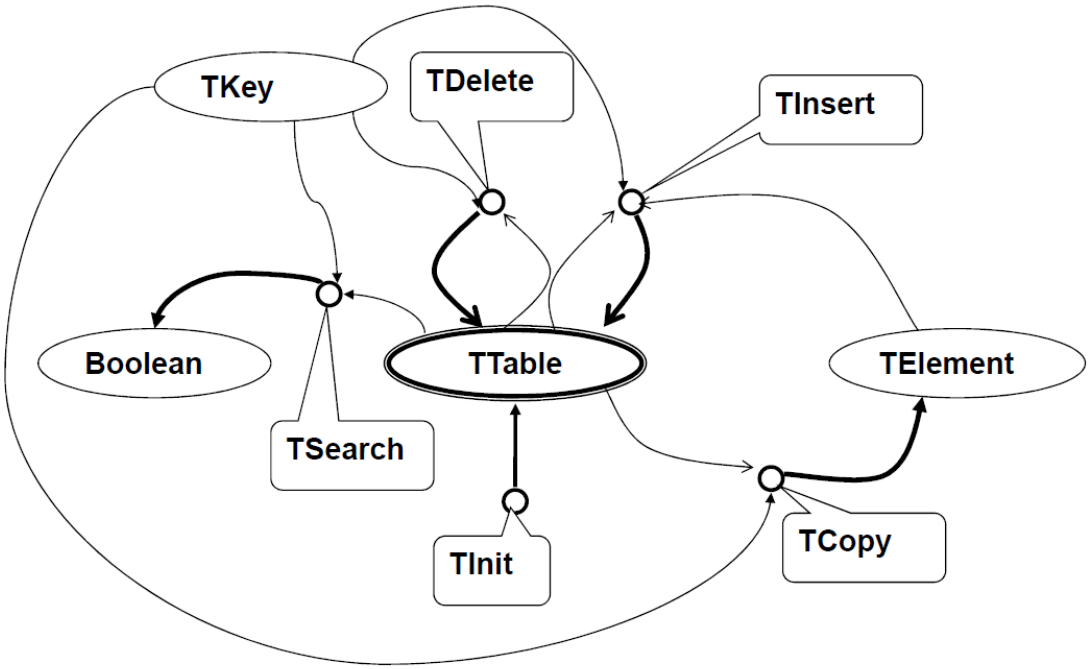
function ADD(a,b:integer):integer

- postfixová ⇨ a b +
- x + y= ⇨ x y + =
- (a+b)*(c-d)/(e+f)*(g-h)=⇨ab+cd-*ef+/gh-*=

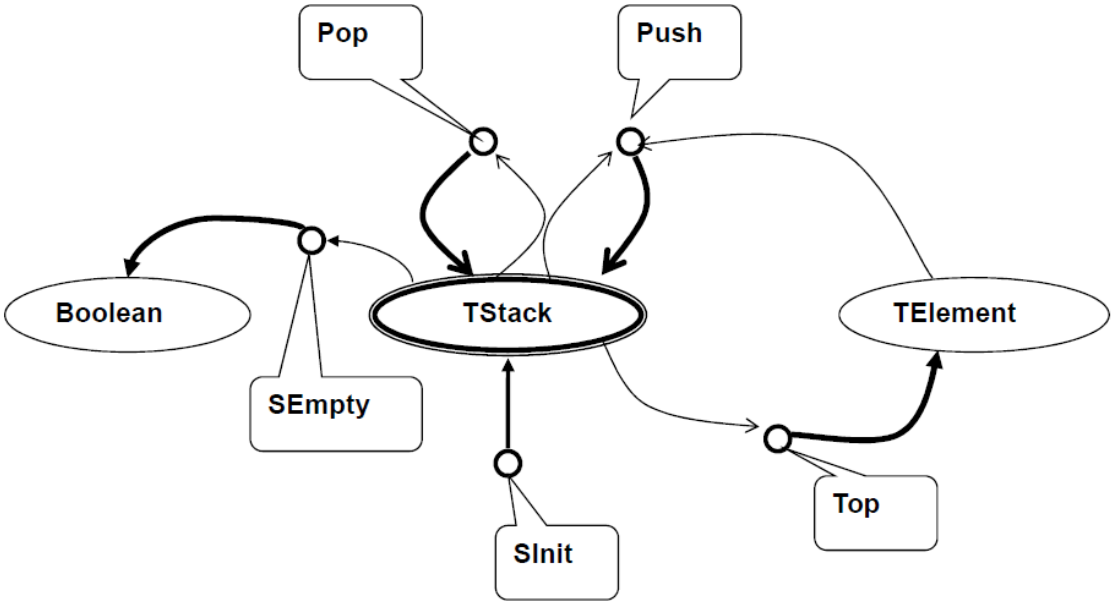
Algoritmus vyčíslení postfixového výrazu

- Zpracovávej řetězec zleva doprava
- Je-li zpracovávaným prvkem operand, vlož ho do zásobníku
- Je-li zpracovávaným prvkem operátor, vyjmi ze zásobníků tolik operandů, kolika-adický je operátor (pro dyadické operátory dva operandy), proveď danou operaci a výsledek uloží na vrchol zásobníku
- Je-li zpracovávaným prvkem omezovač ‘=‘, je výsledek na vrcholu zásobníku

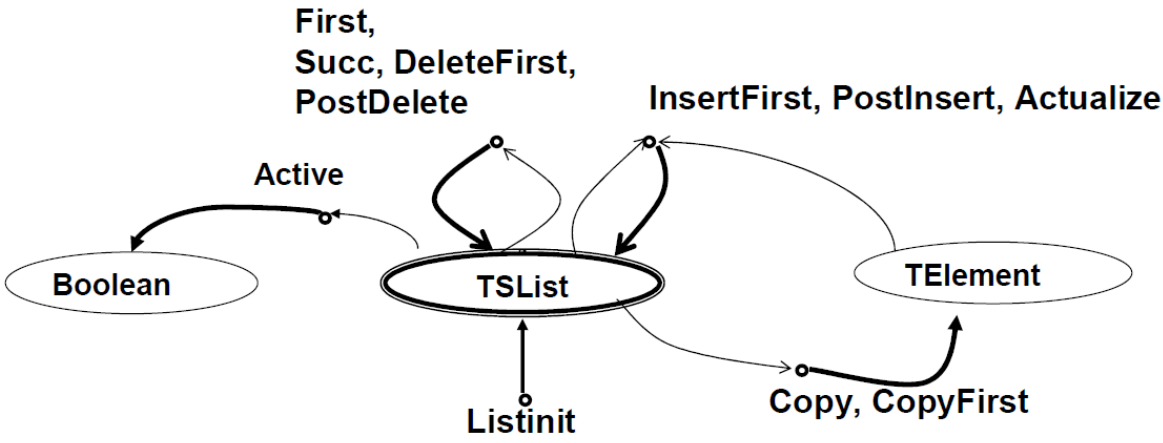
# DIAGRAM SIGNATURY – TABULKA




# DIAGRAM SIGNATURY - ADT ZÁSOBNÍK

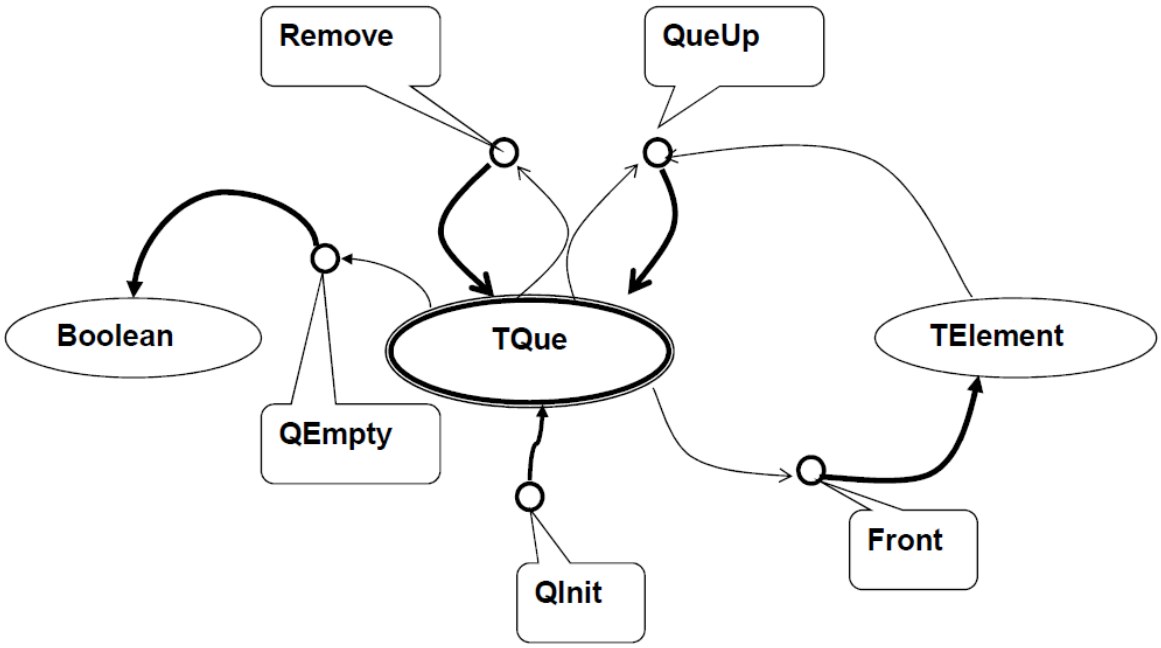


# DIAGRAM SIGNATURY - ADT SEZNAM




**Symetricky doplňkové operace pro dvousměrný seznam:**
**Last, Pred, DeleteFirst, PostDelete, InsertLast, PreInsert, CopyLast**

# DIAGRAM SIGNATURY - ADT FRONTA



# EKVIVALENCE SEZNAMU REKURZIVNĚ

# DÉLKA SEZNAMU REKURZIVNĚ

# DOPLNĚNÍ KÓDU DELETEDMA

```
procedure deleteDMA(var L:TList; Ptr:TListPtr);
(* Ptr ukazuje na rušený prvek. Procedura používá pascalovské ukazatele  *)
begin
  if Ptr<>nil then begin  (* je ukazatel rušeného nenilový? *)
     if (Ptr=L.Frst) and (Ptr=L.Lst)
     then begin    (* rušený je jediným prvkem *)
         L.Frst:=nil;  L.Lst:=nil; (* rušení jediného *)
     end else begin (* rušený není jediným prvkem*)
       if (Ptr=L.Frst)
       then begin (* rušený je prvním prvkem *)
         L.Frst:=Ptr^.RPtr;
          Ptr^.RPtr^.LPtr:=Ptr^LPtr; (* :=nil *)
       end else begin
         if (Ptr=L.Lst)
         then begin (* rušený je posledním prvkem *)
           L.Lst:=Ptr^.LPtr;
           Ptr^.LPtr^.RPtr:=Ptr^.RPtr (* :=nil*)
         end else begin (* rušený má oba sousedy *)
           Ptr^.LPtr^.RPtr:=Ptr^.RPtr;
           Ptr^.RPtr^.LPtr:=Ptr^.LPtr
         end (* Ptr=L.Lst *)
       end (* Ptr=L.Frst *)
     end; (* (Ptr=L.Frst) and (Ptr=L.Lst) *)
     dispose(Ptr);
  end (* Ptr<>nil *)
end;
```

# DOPLNĚNÍ FUNKCE QUEUE (FULL, REMOVE, QUEUP)

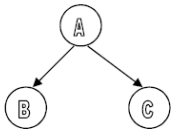
```
procedure QueUp (var Q:TQeue; El:TElem);
begin
   Q.QPole[Q.QKon] := El;
   Q.QKon:=Q.QKon + 1;
    if Q.QKon > QMax
    then Q.QKon := 1;(* Ošetření kruhovosti seznamu *)
end;

procedure Remove (var Q:TQueue);
begin
   if Q.QZac<>Q.QKon
   then begin
      Q.QZac:=Q.Qzac + 1;
       if Q.QZac > QMax
       then Q.QZac:=1; (* Ošetření kruhovosti pole*)
   end (* if *)
```

```
function QFull(Q: TQueue): Boolean;
begin
   QFull:= (Q.Zac=1) and (Q.Kon=QMax) or
((Q.Zac - 1) = Q.Kon)
end;
```

# STROM, VYPSAT SYSTÉM PREORDER, INORDER, POSTORDER

Mějme kořen BS se třemi uzly A,B, a C ve tvar

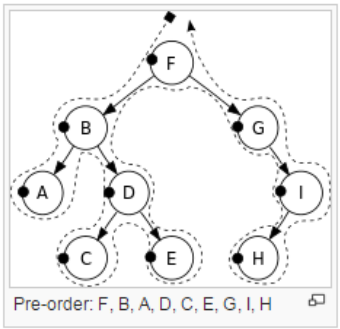



Pak průchod PreOrder má tvar A,B,C
průchod InOrder má tvar B,A,C
průchod Postorder má tvar B,C,A

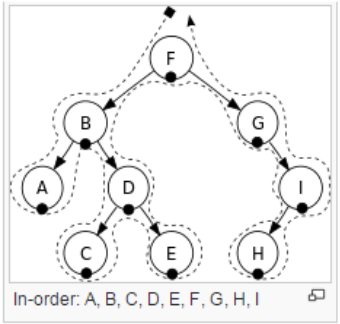
Inverzní průchody mají obrácené pořadí synovských uzlů:
InvPreOrder má tvar A,C,B
InvInOrder má tvar C,A,B
InvPostOrder má tvar C,B,A



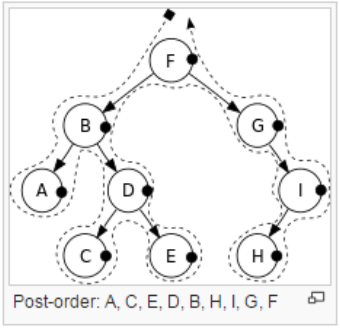
PostOrder je invertovaný InvPreOrder




**Podle puntiku vlevo**




**Podle puntiku dole**




**Podle puntiku vpravo**

**U invertovanych se meni smer prochazeni a umisteni puntiku - zleva doprava, popr. z prava do leva**