Syntaxí řízený překlad - Překlad

- **Definice:** Překlad je relace

  TRAN: $L_1 \rightarrow L_2$

  $L_1 \subseteq \Sigma^*$ Σ - vstupní abeceda

  $L_2 \subseteq \Delta^*$ Δ - výstupní abeceda

- Příklad
  - L_1 – jazyk infixových výrazů
  - L_2 – jazyk postfixových výrazů

Syntaxí řízený překlad - Překladová párová gramatika

- **Definice**: Překladová párová gramatika

  $V = (N, \Sigma, \Delta, P, S)$

  N - neterminály

  Σ - vstupní abeceda

  Δ - výstupní abeceda

  P – přepisovací pravidla

  S – startovací neterminál

  P: $A \rightarrow \alpha, \beta$ $\qquad \alpha \in (N \cup \Sigma)^*,\ \beta \in (N \cup \Delta)^*$

  Neterminály z β jsou permutacemi neterminálů z α

Syntaxí řízený překlad - Příklad (1)

E -> E+T, ET+
E -> T, T
T -> T*F, TF*
T -> F, F
F -> (E), E
F -> i, i

[E,E] => [E+T, ET+]
=> [T+T, TT+]
=> [F+T, FT+]
=> [i+T, iT+]
=> [i+T*F, iTF*+]
=> [i+F*F, FF*+]
=> [i+i*F, iiF*+]
=> [i+i*i,iii*+]

Syntaxí řízený překlad - Překladová gramatika

- Jsou-li neterminály ve vstupní i výstupní části pravé strany ve stejném pořadí, můžeme obě části spojit
  => **překladová gramatika**

- Musíme ale odlišit vstupní a výstupní symboly

Syntaxí řízený překlad - Příklad

$E \rightarrow E + T\ +$

$\quad (E \rightarrow E + T \oplus)$

$E \rightarrow T$

$T \rightarrow T * F\ *$

$T \rightarrow F$

$F \rightarrow (\ E\ )$

$F \rightarrow i\ \ i$

$E \Rightarrow E + T\ +$

$\Rightarrow T + T\ +$

$\Rightarrow F + T\ +$

$\Rightarrow i\ i + T\ +$

$\Rightarrow i\ i + T * F\ *\ +$

$\Rightarrow i\ i + F * F\ *\ +$

$\Rightarrow i\ i + i\ i * F\ *\ +$

$\Rightarrow i\ i + i\ i * i\ i\ *\ +$

Syntaxí řízený překlad - Homomorfismy

- Vstupní homomorfismus:

  $\iota(x) \quad = x, \quad x \in (N \cup \Sigma)$
  $\qquad\quad = \varepsilon, \quad x \in \Delta$

- Výstupní homomorfismus:

  $o(x) \quad = x, \quad x \in (N \cup \Delta)$
  $\qquad\quad = \varepsilon, \quad x \in \Sigma$

- Příklad
  - $\iota(E + T +) \qquad = E + T$
    $o(E + T +) \qquad = E\ T +$

  - $\iota(i\ i + i\ i * i\ i * +) = i + i * i$
    $o(i\ i + i\ i * i\ i * +) = i\ i\ i * +$

Syntaxí řízený překlad - Překlad

- **Definice**: Překlad generovaný překladovou gramatikou:

$$
\begin{aligned}
T(G) = \{(x,y) \mid\ & S \Rightarrow^{*} z, \\
& z \in (\Delta \cup \Sigma)^{*}, \\
& x = \iota(z), && \textit{vstupní věta} \\
& y = o(z)\} && \textit{výstupní věta}
\end{aligned}
$$

Syntaxí řízený překlad - Atributovaný překlad

- Rozšíření bezkontextové gramatiky o kontextové vlastnosti
  → „**gramatika s parametry**“

- Jednotlivé symboly mohou mít přiřazeny parametry – **atributy**

- Atributy se předávají podobně jako vstupní a výstupní argumenty funkcí

Syntaxí řízený překlad - Atributová překladová gramatika

- **Definice**: Atributová *překladová* gramatika
  $\mathbf{G}_{AT} = (G_T, A, F)$
  G_T – překladová gramatika
  ($\Delta = \emptyset$ -> **atributová gramatika**)
  A – množina atributů
  F – sémantická pravidla (pravidla pro výpočet hodnot atributů)

Syntaxí řízený překlad - Množina atributů

- $X \in (\Delta \cup \Sigma \cup N)$
  - I(X) - množina **dědičných** atributů
  - S(X) - množina **syntetizovaných** atributů

  - Dědičné atributy startovacího neterminálu jsou zadány předem.
  - Syntetizované atributy terminálních symbolů jsou zadány předem (lexikální analyzátor)

Syntaxí řízený překlad - Sémantická pravidla (1)

p_r: $X_0 \rightarrow X_1 X_2 \ldots X_n$ $\quad X_0 \in N,\ X_i \in (N \cup \Delta \cup \Sigma)$

a) $d = f_r^{d,k}(a_1, a_2, \ldots, a_n)$ $\quad d \in I(X_k),\ 1 \leq k \leq n$

b) $s = f_r^{s,0}(a_1, a_2, \ldots, a_n)$ $\quad s \in S(X_0)$

c) $s = f_r^{s,k}(a_1, a_2, \ldots, a_n)$ $\quad s \in S(X_k),\ X_k \in \Delta$

- a,b) a_i je atribut symbolu X_j, $0 \leq j \leq n$
- c) a_i je dědičný atr. symbolu X_f, $X_f \in \Delta$

Syntaxí řízený překlad - Sémantická pravidla (2)

- Výklad předchozích definic:

  a) Dědičné atributy symbolu na pravé straně pravidla

  b) Syntetizované atributy neterminálního symbolu na levé straně pravidla

  c) Syntetizované atributy výstupních symbolů na pravé straně pravidla

Syntaxí řízený překlad - Syntaxí řízená definice

- sémantické funkce v atributové gramatice nemohou mít vedlejší efekty
  - vstup/výstup (např. generovaného kódu)
  - globální proměnné (např. tabulka symbolů)

- *syntaxí řízená definice* – bez omezení

Syntaxí řízený překlad - Syntaxí řízené definice

- Syntaxí řízené definice
  - Základní idea je, že jednotlivým pravidlům přiřadíme množinu sémantických akcí.
  - Specifikace překladu na vysoké úrovni abstrakce.
  - Ukrývají mnoho implementačních detailů jako například pořadí vyhodnocování pravidel.
- Vyhodnocení syntaxí řízených definic
  - Nejprve spočítáme graf závislostí mezi definicemi.
  - Určíme pořadí v jakém budou pravidla vyhodnoceny.
  - Provedeme jednotlivé sémantické akce a naplníme hodnoty jednotlivých atributů.

Syntaxí řízený překlad - Syntaxí řízené definice

- Co když zavedeme následující omezení? Pomůžeme si?
  - Dědičný atribut symbolu na pravé straně vypočte akce umístěna před ním.
  - Akce nesmí používat syntetizované atributy symbolů vpravo od ní.
  - Hodnota syntetizovaného atributu neterminálu na levé straně pravidla se vypočte až po určení všech atributů, na které se odkazuje - obvykle až na konci pravidla

Syntaxí řízený překlad - L - atributová definice

L-atributová definice

$$A \rightarrow X_1 \ldots X_n$$

Dědičné atributy symbolu X_j závisejí jen na:

- atributech symbolů $X_1, \ldots, X_{j-1}$

  (tj. symbolech z téhož pravidla *vlevo* od symbolu X_j)
- dědičných atributech symbolu A

  (tj. symbolu na *levé* straně pravidla)

- ***Použití: LL(1) překlad – jasně definované pořadí pro vyhodnocení sémantických pravidel.***

Syntaxí řízený překlad - Příklad na výpočet hodnoty výrazu (1)

- Syntetizované atributy:
  - E.val - hodnota výrazu
  - T.val - hodnota podvýrazu
  - F.val - hodnota operandu

  - num.lexval – hodnota číselné konstanty

Syntaxí řízený překlad - Příklad na výpočet hodnoty výrazu (2)

- Gramatika

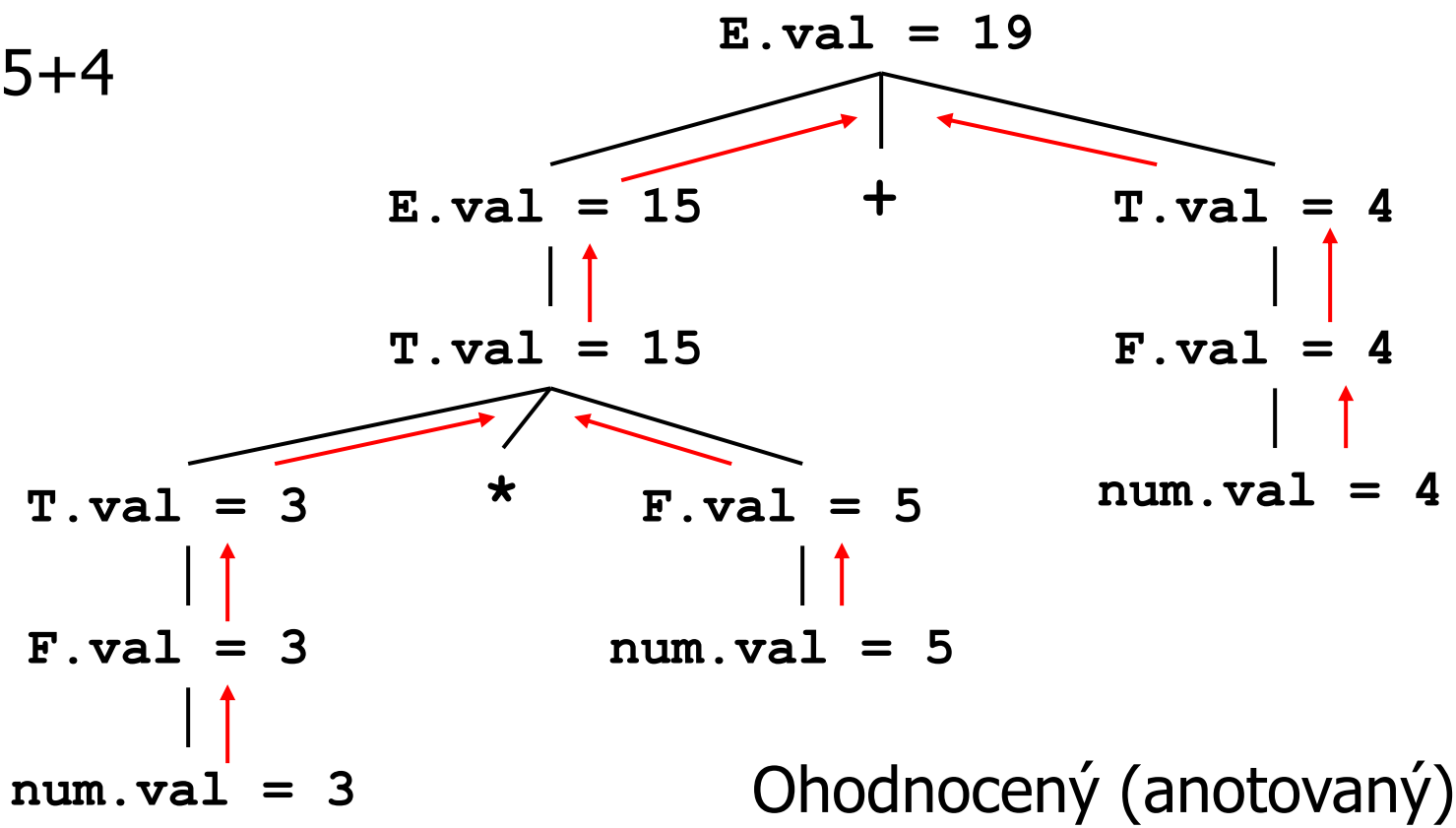
- E -> E’ + T
  E -> T
  T -> T’ * F
  T -> F
  F -> (E)
  F -> num

- Sémantická pravidla

- E.val = E’.val + T.val
  E.val = T.val
  T.val = T’.val * F.val
  T.val = F.val
  F.val = E.val
  F.val = num.lexval

Syntaxí řízený překlad - Příklad na výpočet hodnoty výrazu (3)

3*5+4



Ohodnocený (anotovaný) derivační strom

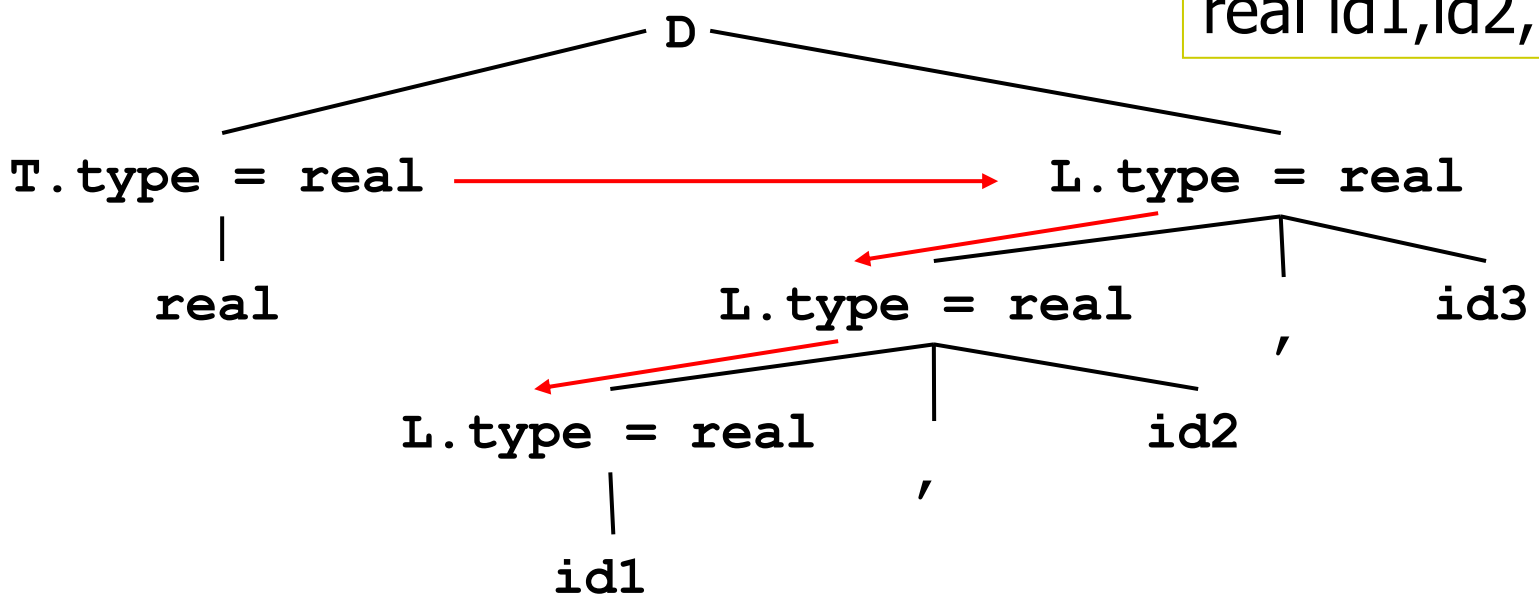
Syntaxí řízený překlad - Příklad deklarace proměnných (1)

(1) D -> T L	L.type := T.type
(2) T -> int	T.type := integer
(3) T -> real	T.type := real
(4) L -> L’ , id	L’.type := L.type dcl(id.ptr, L.type)
(5) L -> id	dcl(id.ptr, L.type)

Syntaxí řízený překlad - Příklad deklarace proměnných (1)

- Dědičné atributy: L.type
- Syntetizované atributy: T.type, id.ptr

real id1,id2,id3



Syntaxí řízený překlad - Překladové schéma

- Alternativou k syntaxí řízeným definicím je **překladové schéma**.
  - *Syntaxí řízená definice* se sémantickými akcemi umístěnými kdekoliv na pravé straně pravidla
  - Posloupnost sémantických akcí je přesně definována (syntaxí).

Syntaxí řízený překlad - Příklad překladového schéma

E -> T R

R -> + T **{ print(‘+’); }** R’
| ε

T -> num **{ print(num.val); }**

Syntaxí řízený překlad - Překlad při analýze rekurzivním sestupem (1)

- Syntaktický zásobník - řízení překladu
  Atributový zásobník - ukládání atributů

- Atributy
  - parametry podprogramů $(N \cup \Sigma)$
  - globální proměnné Δ

Syntaxí řízený překlad - Překlad při analýze rekurzivním sestupem (2)

- Atributy
  - dědičné - vstupní parametry
  - syntetizované - výstupní parametry
- Sémantické akce
  - Přímo na odpovídajících místech v podprogramech pro analýzu pravidel
- Hodnoty syntetizovaných atributů **musí** být definovány! (e-pravidla, zotavení)

Syntaxí řízený překlad - Příklad implementace rekurzivním sestupem (1)

E -> T *{R.i := T.val}* R *{E.val := R.s}*

R -> addop T

{if addop.op=add then

R_1*.i := R.i+T.val*

else R_1*.i := R.i-T.val }*

R_1 *{R.s :=* R_1*.s}*

| ε *{R.s := R.i}*

T -> num *{T.val := num.val}*

Syntaxí řízený překlad - Příklad implementace rekurzivním sestupem (2)

- addop.op … var lexop:char; ‘+’/‘-’
  num.val … lexval:integer;

- R.i - dědičný atribut (levý operand)
  R.s - dočasný výsledek, synt. atribut
  T.val - syntetizovaný attribute

Syntaxí řízený překlad - Příklad implementace rekurzivním sestupem (3)

```
void E(int& val) {
  int val1;
  T(val1);
  R(val1, val);
}
```

Syntaxí řízený překlad - Příklad implementace rekurzivním sestupem (4)

```
void R(int i, int& s) {
  if( sym == '+' || sym == '-' ) {
    char op=lexop;
    int val;
    T(val);
    if( op=='+' ) R(i+val,s)
              else R(i-val,s);
  } else
    s = i;
}
```

Syntaxí řízený překlad - Příklad implementace rekurzivním sestupem (5)

```
void T(int& val) {
  if( sym == num ) {
    val=lexval; // nestačí expect(num)
    lex();
  } else {
    error(); // po zotavení musí
    val=0;   // být val definováno
  }
}
```

Tabulka symbolů - Tabulka symbolů

- Reprezentace pojmenovaných entit
  - explicitně pojmenovaných uživatelem
  - implicitně pojmenovaných – standardní entity (typy, funkce, třídy, ...), dočasné proměnné
- Účel:
  - řešení kontextových vazeb (deklarace -> použití)
  - typová kontrola
  - ukládání informací pro generování kódu

Tabulka symbolů - Tabulka symbolů

- Interakce s lexikálním analyzátorem
  - ukládání identifikátorů během lexikální analýzy
  - využití kontextových informací

```
Expr ->   IdVar Index
        |  IdProc Args
        |  IdCon
        …
```

  - PASCAL: P(x,y) kontextově závislá syntaxe
    Write(x:3,y:2:5)

Tabulka symbolů - Funkce tabulky symbolů

- **Operace**:
  - init - inicializace
  - insert - vkládání
  - lookup - vyhledávání (častější)

- **Inicializace**
  - vytvoření prázdné tabulky
  - naplnění implicitními deklaracemi

Tabulka symbolů - Funkce tabulky symbolů

- **Vkládání** – nejprve vyhledá klíč v tabulce
  - není nalezen → vytvoří se nová položka
  - nalezen → obvykle chyba
    - deklarace / definice
    - implicitní deklarace (návěští a funkce v C)
- **Vyhledávání** – vyhledá klíč v tabulce
  - nalezen → vrátí odpovídající entitu
  - nenalezen → obvykle ohlásí chybu

Tabulka symbolů - Implementace tabulky symbolů (1)

- Neseřazené tabulky $O(n)$
  - jen pro malý počet prvků
- Seřazené tabulky s binárním vyhledáváním $O(\log_2 n)$
  - pro statické tabulky (např. klíčová slova)
- Vyhledávací stromy $O(n)$ - $O(\log_2 n)$
  - doba vyhledávání závisí na vyvážení stromu
  - časová náročnost vkládání není kritická
  - optimálně vyvážené stromy
    - příliš komplikované
  - suboptimální řešení
    - např. AVL stromy

Tabulka symbolů - Implementace tabulky symbolů (2)

- Tabulky s rozptýlenými položkami O(1)
  - mapovací funkce: *klíč* → *index*
  - řešení kolizí
    - seznamy/stromy synonym
    - rozptylovací funkce: *index* → *index*

  - rychlost závisí na zaplnění tabulky
  - problematický průchod položkami podle klíče
  - obtížně se řeší přetečení tabulky

Tabulka symbolů - Blokově strukturovaná tabulka (1)

- Vhodná pro jazyky se zanořenými deklaracemi (hierarchické struktury)
- Řeší rozsah platnosti a viditelnost jména v zanořených blocích
- Nové operace:
  - otevření rozsahu platnosti (open)
  - uzavření rozsahu platnosti (close)

Tabulka symbolů - Blokově strukturovaná tabulka (2)

- Vkládání
  - pracuje pouze s aktuální úrovní tabulky
  - jména na vyšších úrovních se neuvažují
- Vyhledávání
  - klíč se vyhledává nejprve v aktuální úrovni, pokud se nenajde, hledá o úroveň výš
  - neúspěch se hlásí až po prohledání nejvyšší úrovně (globální jména)
  - explicitní přístup na globální úroveň ::x

Tabulka symbolů - Implementace blokově strukturované tabulky symbolů

- Založena na některé z metod pro nestrukturovanou tabulku
  - vyhledávací stromy
  - tabulky s rozptýlenými položkami
- Přirozenou datovou strukturou pro reprezentaci hierarchie je **ZÁSOBNÍK**
  - Úrovně se nemohou překrývat

Tabulka symbolů - Příklad

	jméno entity	další atributy
8	y	
7	delta	
6	max	
5	num	
4	x	
3	fun	
2	y	
1	x	

TOP

7
4
1

index bloku

```
int x,y;
double fun(x,num)
{
   int max;
   …
   {
    double delta,y;
    …
   }
}
```

Vnitřní reprezentace - **Možnosti překladu**

- Interpretace
  - Okamžité provádění programu
- Překlad do instrukcí procesoru
  - Závislost na konkrétním typu procesoru
- **Překlad do vnitřní reprezentace**
  - Následuje interpretace nebo překlad do instrukcí procesoru

Vnitřní reprezentace - Výhody překladu do vnitřní reprezentace

- Strukturalizace překladače
- Mnohem jednodušší přenos na více typů procesorů
- Možnost optimalizace na úrovni vnitřní reprezentace
  - strojově nezávislé metody

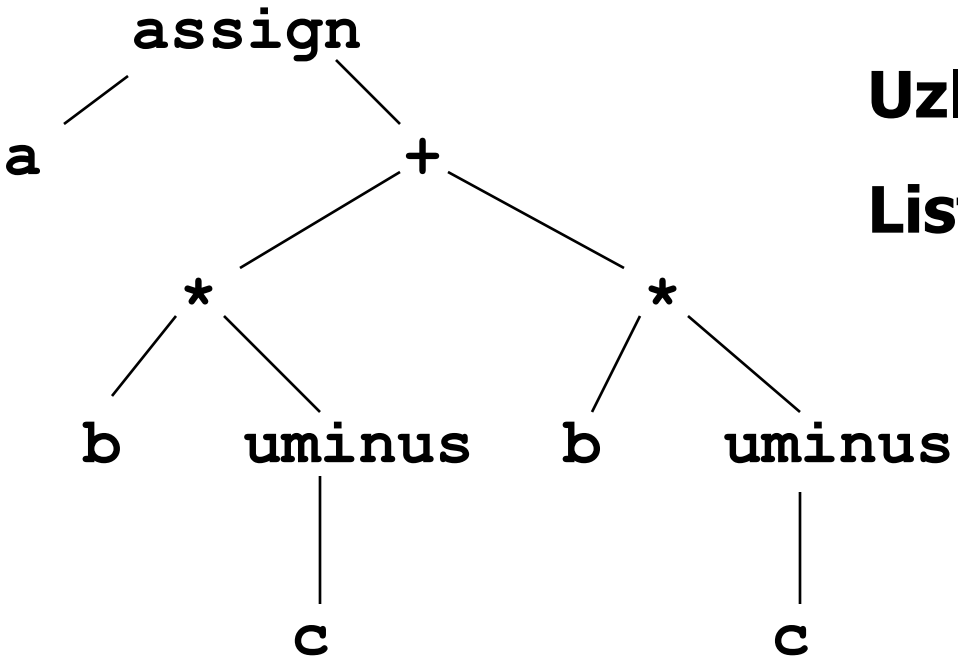
Vnitřní reprezentace - Formáty vnitřní reprezentace programu

- Grafová reprezentace
- Zásobníkový kód
- Tříadresový kód

Vnitřní reprezentace - Grafová reprezentace

- Abstraktní syntaktický strom (AST)

`a:=b*(-c)+b*(-c)`



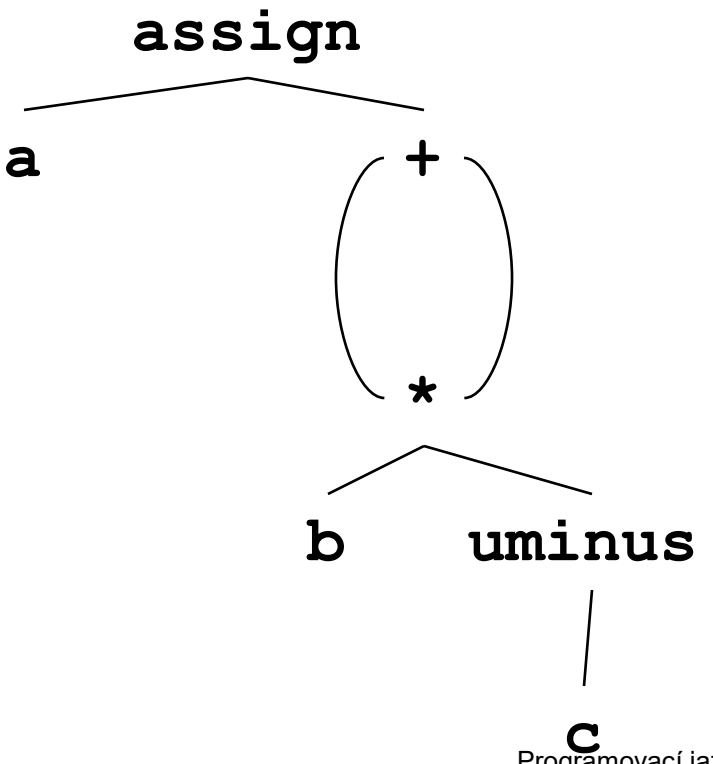
Uzly - operátory

Listy - operandy

Vnitřní reprezentace - Grafová reprezentace

- DAG (Directed Acyclic Graph)

`a:=b*(-c)+b*(-c)`



Společné podvýrazy jako sdílené uzly

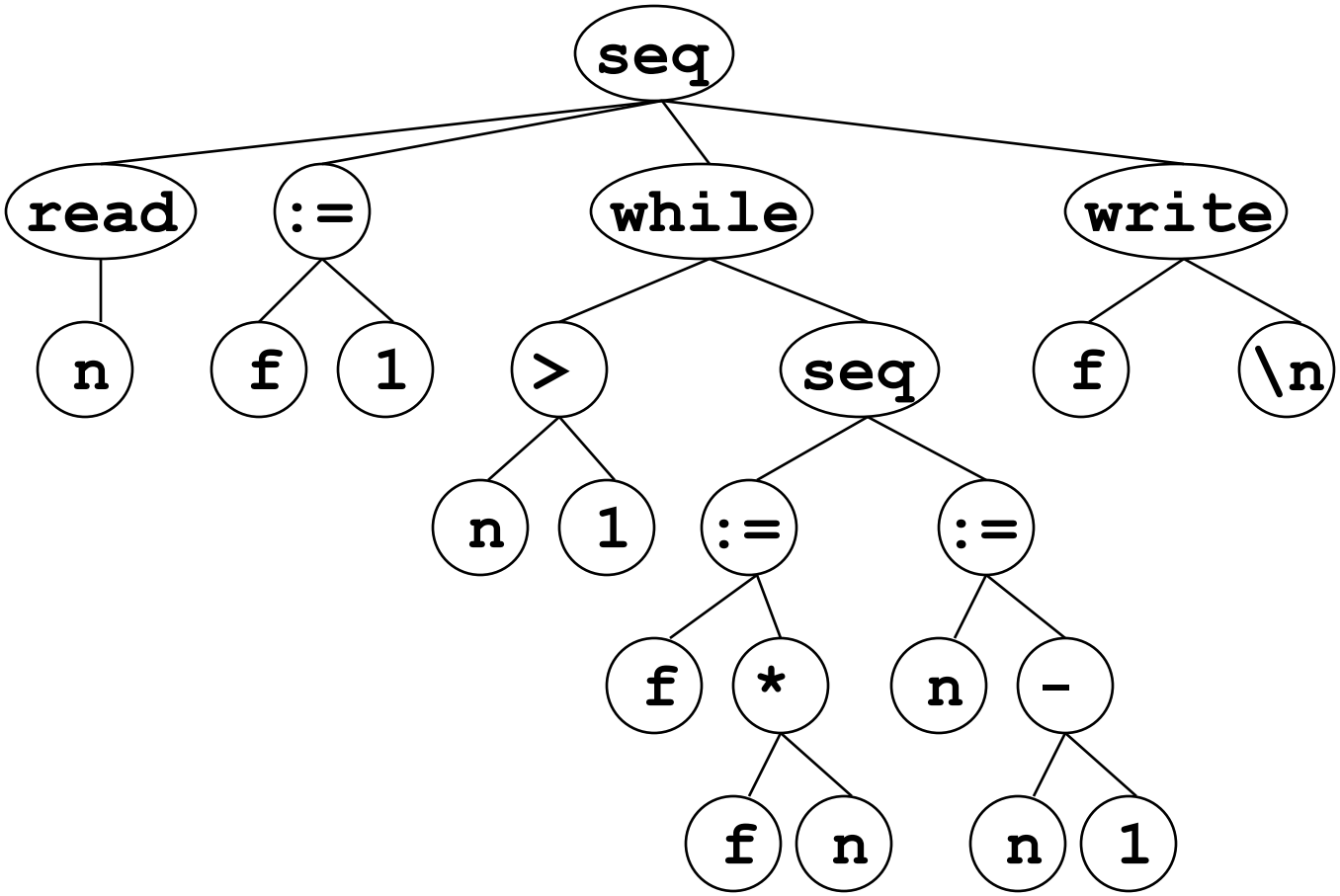
Používá se pro optimalizaci

Vnitřní reprezentace - Grafová reprezentace

- Obecný grafový model
  - OO modelování a návrh

```
var n,f:integer;
begin  read n;
       f:=1;
       while n>1 do begin
         f:=f*n; n:=n-1
       end;
       write f,"\n"
end.
```

Vnitřní reprezentace - Grafová reprezentace



Vnitřní reprezentace - Zásobníkový kód

- Postfixová notace

  `a b c uminus * b c uminus * + assign`

  - výsledek průchodu AST nebo DAG typu post-order
  - uzel následuje bezprostředně za svými následníky
  - žádný formální rozdíl mezi operandy a operátory

Vnitřní reprezentace - Zásobníkový kód

- Virtuální zásobníkový procesor

VAR b	(b)	vrchol zás.
VAR c	(b) (c)	
INV	(b) (-c)	
MUL	(b*-c)	
VAR b	(b*-c)(b)	
VAR c	(b*-c)(b)(c)	
INV	(b*-c)(b)(-c)	
MUL	(b*-c)(b*-c)	
ADD	(b*-c+b*-c)	
ASG a		

Vnitřní reprezentace - Zásobníkový kód

- Virtuální zásobníkový procesor
  - virtuální počítač s pamětí a zásobníkem
  - P-kód (Wirth)
    - přenositelný mezikód pro Pascal
    - specializované procesory
  - Java Virtual Machine
  - MSIL – Microsoft Intermediate Language (.NET)

Vnitřní reprezentace - Tříadresový kód

- **x := y op z**
  - x - (dočasná) proměnná
  - y, z - (dočasné) proměnné, konstanty
- Operandy nemohou být výrazy
  - rozklad na primitivní výrazy s dočasnými proměnnými
- Explicitní odkazy na operandy
  - možnost přesouvání příkazů při optimalizaci

Vnitřní reprezentace - Příkazy tříadresového kódu

- x := y op z — op – binární operátor
- x := op y — op - unární operátor (-, not, typová konverze, …)
- x := y
- goto L — nepodmíněný skok
- if x relop y goto L — podmíněný skok
- x := y[j]
  x[j] := y — indexování
- x := &y — reference (získání adresy)
- x := *y
  *x := y — dereference ukazatele

Vnitřní reprezentace - Příklad

- `a:=b*(-c)+b*(-c)`
-

```
t1 := -c
t2 := b*t1
t3 := -c
t4 := b*t3
t5 := t2+t4
a  := t5
```

```
t1 := -c
t2 := b*t1
t5 := t2+t2
a  := t5
```

- **syntaktický strom** **DAG**

- dočasné proměnné = (vnitřní) uzly stromu

Vnitřní reprezentace - Příklad – překlad výrazů (1)

- Atributy
  - *id.name* – jméno identifikátoru
  - *E.place* – proměnná obsahující hodnotu E
- Pomocné funkce
  - *newtemp* – vytvoří novou proměnnou
  - *lookup* – vyhledá proměnnou v tabulce
  - *emit* – generuje instrukci
  - *error* – hlášení chyby

Vnitřní reprezentace - Příklad – překlad výrazů (2)

- S -> id := E

```
{ p = lookup(id.name);
  if( p != null )
    emit(p,':=',E.place);
  else error(); }
```

- E -> E_1 + E_2

```
{ E.place = newtemp();
  emit(E.place,':=',
     E1.place,'+',E2.place);}
```

- E -> E_1 * E_2

```
{ E.place := newtemp();
  emit(E.place,':=',
     E1.place,'*',E2.place);}
```

Vnitřní reprezentace - Příklad – překlad výrazů (3)

- E -> - E_1

```
{ E.place = newtemp();
  emit(E.place,':=',
     'uminus',E1.place);}
```

- E -> (E_1)

```
{ E.place = E1.place; }
```

- E -> id

```
{ p := lookup(id.name);
  if( p != null )
    E.place := p;
  else error(); }
```

Vnitřní reprezentace - Překlad logických výrazů (1)

- E -> E **or** E
  | E **and** E
  | (E)
  | **not** E
  | id relop id
  | **true**
  | **false**

Vnitřní reprezentace - Překlad logických výrazů (2)

- Reprezentace logických hodnot celými čísly: true=1, false=0

- **a or b and not c**

  t1 := not c
  t2 := b and t1
  t3 := a or t2

  x<y

  1: if x<y goto 4
  2: t1 := 0
  3: goto 5
  4: t1 := 1
  5: …

Vnitřní reprezentace - Překlad logických výrazů (3)

- Zkrácené vyhodnocení
  - reprezentace logických hodnot pozicí v programu

- **a<b or c<d and e<f**

```
     if a<b goto Ltrue
     goto L1
L1:  if c<d goto L2
     goto Lfalse
L2:  if e<f goto Ltrue
     goto Lfalse
```

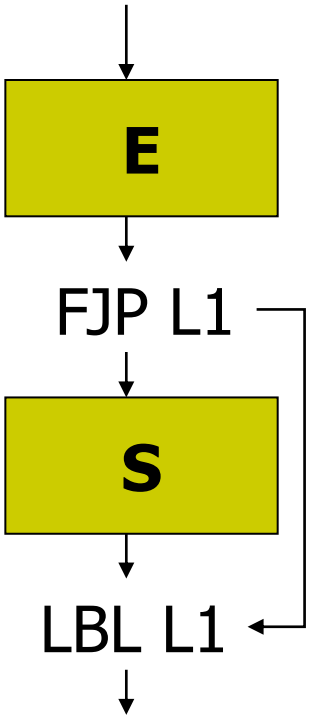
Vnitřní reprezentace - Překlad řídicích příkazů (1)

- Instrukce zásobníkového kódu:
  - LBL – definice návěští pro skok
  - JMP L – nepodmíněný skok na návěští L
  - FJP L - podmíněný skok na návěští L, pokud je na vrcholu zásobníku False
- Pomocné funkce:
  - getLbl() – vygeneruje nové číslo návěští

Vnitřní reprezentace - Překlad příkazu IF (1)

- S -> **if** (E) S
  - <E>
  - FJP L1
  - <S>
  - LBL L1

- S -> **if** (E) *{FJP}* S *{LBL}*
  FJP.l = LBL.l = getLbl();



Vnitřní reprezentace - Překlad příkazu IF (2)

- S -> **if** (E) S^1 **else** S^2

S -> X *{LBL}*

LBL.l = X.l

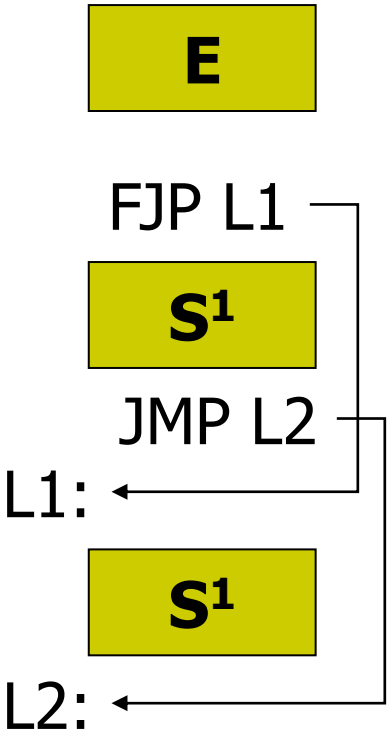
S -> X *{JMP}* *{LBL1}* **else** S *{LBL2}*

JMP.l = LBL2.l = getLbl();

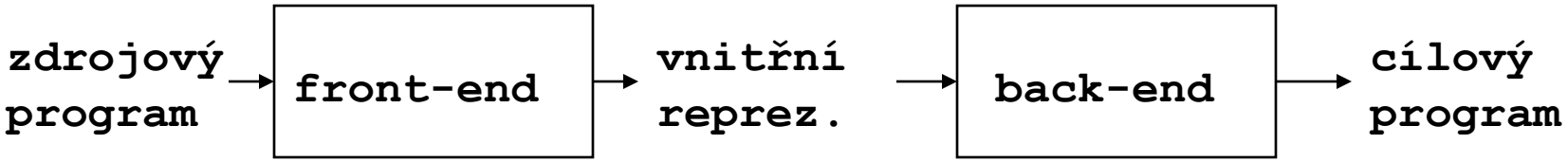
LBL1.l = X.l

X -> **if** (E) *{FJP}* S

FJP.l = getLbl()



Optimalizace – Možnosti optimalizace



uživatel	překladač	překladač
změna algoritmu	strojově	strojově
transformace	nezávislé	závislé
profilování	optimalizace	optimalizace

„Optimalizace" = transformace programu směřující ke zlepšení jeho vlastností za běhu (rychlost, prostor)

Optimalizace – Možnosti optimalizace

- Optimalizace zdrojového programu
  - často největší vliv – např. změna algoritmu
  - možnost profilování programu
- Strojově nezávislé optimalizace
  - na úrovni vnitřní reprezentace
- Strojově závislé optimalizace
  - na úrovni cílového programu
  - co nejlepší využití instrukční sady procesoru a jeho zdrojů (registry, paměť, ...)

Optimalizace – Kritéria pro transformace

- musí zachovat význam programu
  - "bezpečný" přístup --- raději neoptimalizovat!
- musí program v průměru významně zrychlit
  - někdy nám jde o velikost paměti
  - v některých případech se může efektivita programu i zhoršit
- musí stát za vynaložené úsilí
  - vyplatí se pro často spouštěné programy

Optimalizace – Volba vnitřní reprezentace programu pro optimalizaci

- Zásobníkový kód
  - obtížná změna struktury programu
- **Tříadresový kód**
  - explicitně vyjádřené závislosti
  - blízký architektuře RISC i klasickým architekturám procesorů
- Grafová reprezentace
  - grafové transformace, přepisovací systémy

Optimalizace – Organizace optimalizujícího překladače

- Analýza toku řízení
  - rozdělení programu na *základní bloky*
  - vytvoření *grafu toku řízení*
- Analýza toku dat
  - živé proměnné
  - dosahující definice
  - dostupné výrazy
- Transformace

Optimalizace – Základní blok

Nejdelší posloupnost příkazů s jedním vstupem a jedním výstupem

- pouze poslední příkaz může být skok
- cílovým místem skoků může být pouze první příkaz bloku

- můžeme jej považovat za jedinou instrukci
- určuje meze lokálních optimalizací
- snížení množství uchovávaných dat
  - např. 16B/instrukci + 8KB/data pro optimalizaci
  - lze ukládat jednou pro základní blok a ostatní dopočítat

Optimalizace – Typy optimalizací

- Lokální optimalizace
  - na úrovni *základního bloku*
  - nevyžaduje znalost kontextu
- Globální optimalizace
  - v rámci podprogramů
  - v rámci programu (interprocedurální)

Optimalizace – Příklady lokálních optimalizací (1)

- odstranění společných podvýrazů

  a[i,j] = a[i,j]+1;

- vyhodnocení konstantních výrazů

  a = 2 * 3 + 5;

- šíření kopií

  f = g;

- šíření konstant

  f = 2 * 3; if(f > 10) …

Optimalizace – Příklady lokálních optimalizací (2)

- odstranění mrtvého kódu

  debug = false; if (debug) printf("…");

  goto L; x = 10; L: …

- algebraické transformace

  x * 1 + 0 => x

- redukce síly operací

  x ** 2 => x * x

  5 * x => (x << 2) + x

  length(x+y) => length(x)+length(y)

Optimalizace – Optimalizace cyklů

- přesun kódu mimo cyklus
  - invarianty cyklu lze přesunout před tělo cyklu
- redukce indukčních proměnných
  - generování posloupnosti hodnot
  - např. indexování pole proměnnou cyklu => použití ukazatele na aktuální prvek

- PROBLÉM: detekce cyklů

Optimalizace – Globální optimalizace programu

- překladač potřebuje informace o programu jako celku
  - obtížně proveditelné v modulárních programech
  - často jen v rámci jednoho podprogramu (intraprocedurální optimalizace)
- tyto informace se připojují k základním blokům

Optimalizace – Příklady globálních optimalizací

- využití informací o *živosti proměnných* k přidělování registrů a ke kontrole správné inicializace proměnných
- odstranění redundantních výpočtů analýzou *společných podvýrazů*
- šíření konstant pomocí *dosahujících definic*
- nahrazení volání podprogramu jeho tělem (otevřené podprogramy – inline)

Generování cílového kódu - Činnost generátoru

- vytvoření sémanticky ekvivalentního programu
- výběr ‘optimální’ varianty

Generování cílového kódu - Cílový jazyk (1)

- Absolutní strojový kód
  - nejefektivnější výstup
  - program se musí přeložit celý najednou (složitá vazba na knihovny)
  - vhodné pro malé programy, výukové překladače, specializované procesory
- Přemistitelný strojový kód
  - běžné používaný v klasických překladačích
  - nutnost použití sestavovacího programu

Generování cílového kódu - Cílový jazyk (2)

- Jazyk symbolických instrukcí
  - jednoduché generování
    - adresové výpočty, makroinstrukce
  - vyžaduje další dva průchody asembleru
- Jazyk virtuálního procesoru
  - Java VM
  - .NET MSIL

Generování cílového kódu - Požadavky na generátor cílového programu

- Rychlý překlad -> rychlé algoritmy
- Bezpečný kód -> kontroly v době běhu
- Dobrá detekce chyb za běhu
- Vlízký vztah mezi zdrojovým a cílovým programem -> jednodušší ladění
- Efektivita cílového programu (rychlost+velikost)

- *Protichůdné požadavky!*

Generování cílového kódu - Řešení

- Sada překladačů - spíš historie
  - IBM FORTRAN - FORTRAN H
  - PL/I Checkout compiler - diagnostika
    Full compiler
    Optimizing compiler
- Parametrizovatelný překladač
  - Zapnutí/vypnutí optimalizací
  - Výběr prováděných optimalizací

Generování cílového kódu - Klasické metody generování cílového programu

- Princip
  - dekompozice prvků jazyka na nezávislé podproblémy (např. součet dvou hodnot)
  - každý podproblém řešíme tou nejefektivnější metodou
- Implementace
  - sada procedur pro konkrétní podproblémy

Struktura programu v době běhu - Obsah

- Vztah mezi zdrojovým programem a činností přeloženého programu
  - reprezentace dat
  - správa paměti
  - aktivace podprogramů
    - parametry
    - rekurze

Struktura programu v době běhu - Systém řízení běhu programu (1)

- Inicializace programu
  - převzetí parametrů od systému
  - alokace paměti pro data
  - otevření standardních souborů (Pascal)
- Finalizace programu
  - uzavření souborů
  - uvolnění paměti
  - předání stavového kódu

Struktura programu v době běhu - Systém řízení běhu programu (2)

- Zpracování chyb
  - přerušení a výjimky (definované uživatelem, systémové a aplikační chyby, ...)
  - automatické kontroly (přetečení zásobníku, ukazatele, indexy, parametry, ...)
- Dynamické přidělování paměti
  - zásobník
  - volná paměť
- Komunikace s operačním systémem
  - soubory, čas, …
- Volací a návratové posloupnosti
- Podpora pro ladění programu

Struktura programu v době běhu – **Podprogramy**

- Procedury
  - jméno -> příkaz
- Funkce
  - jméno -> výraz

```
void main() {
  int fib(int n) {
    if( n < 2 ) return 1;
    return fib(n-2) + fib(n-1);
  }
  void print(int n) {
    for(int k = 1; k <= n; k++)
      printf("%d: %d", k, fib(k));
  }
  int max;
  scanf("%d", &max);
  print(max);
}
```

Struktura programu v době běhu – Definice podprogramu

- Jméno
- Formální parametry (parametry)
  - jméno
  - typ
  - způsob předávání
- Tělo

Struktura programu v době běhu – Aktivace podprogramu

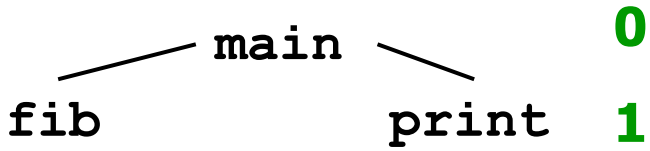
- Skutečné parametry (argumenty)
- Doba života aktivace
  - od zavolání podprogramu po návrat zpět
  - obvykle se nepřekrývají nebo jsou do sebe vnořeny (výjimka: vlákna, procesy)
- Rekurzivní podprogramy
  - nová aktivace se může spustit před ukončením předcházející

Struktura programu v době běhu – Úroveň zanoření

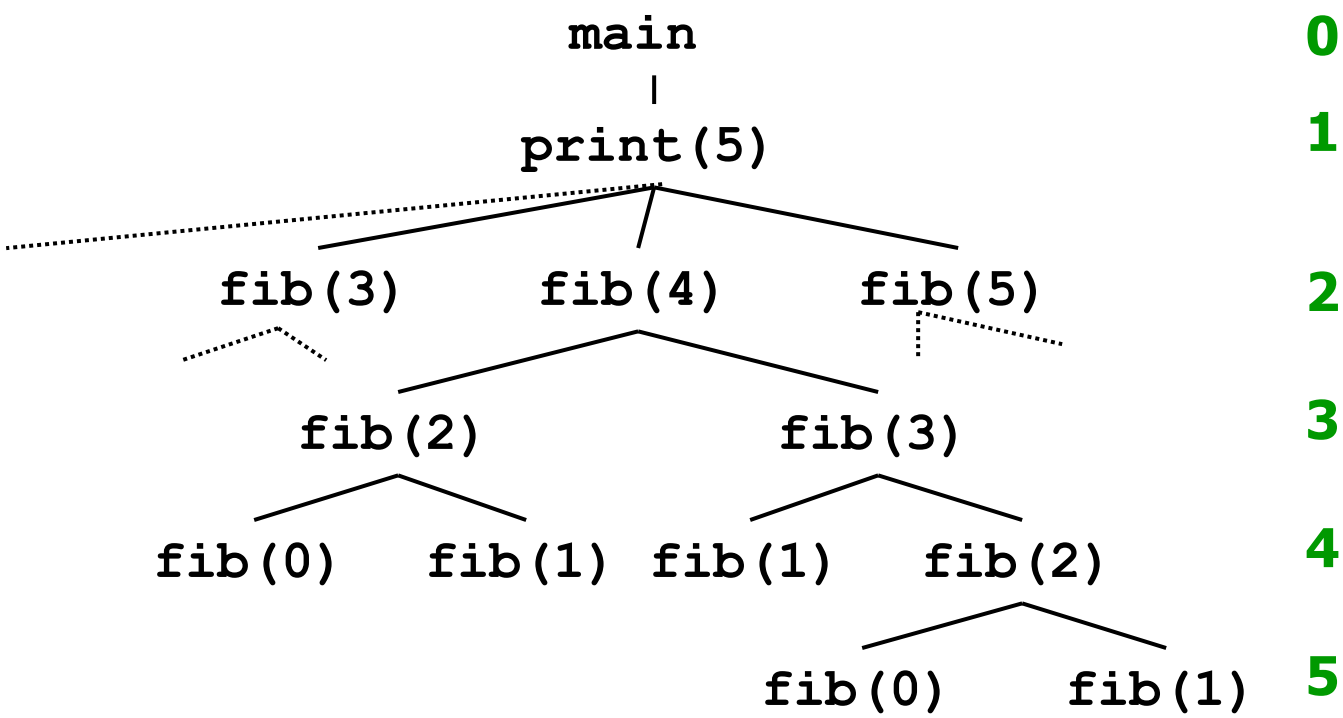
- Statická úroveň zanoření
  - je definována zdrojovým programem
  - C, Java – pouze jedna úroveň
- Dynamická úroveň zanoření
  - je definována zanořováním aktivací
  - kolik aktivací je současně rozpracováno

- Úroveň zanoření je dána vzdáleností uzlu od kořene stomu

Struktura programu v době běhu – **Příklad**



Statické zanoření



Dynamické zanoření

Struktura programu v době běhu – Lokální data aktivace (1)

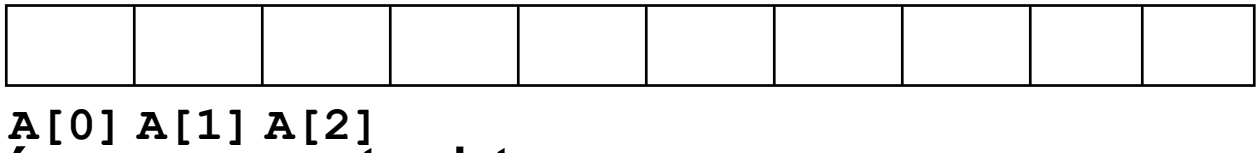
- Aktivační záznam
  - lokální proměnné (+dočasné proměnné)
  - návratová adresa
  - uschovaný obsah registrů
- Jazyky bez rekurzivních podprogramů
  - aktivační záznamy mohou být alokovány staticky v době překladu (FORTRAN)
- Jazyky s rekurzivními podprogramy
  - zásobník aktivačních záznamů
  - řídicí zásobník
    - obsahuje všechny aktivační záznamy od kořene stromu aktivací až po aktivní uzel (podprogram)

Struktura programu v době běhu – Volací a návratové posloupnosti

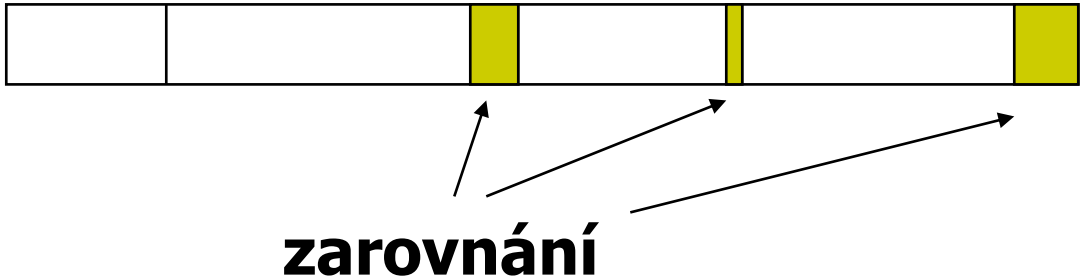
- Definice adresy aktivačního záznamu
  Uložení stavu (registry, příznaky, návratová adresa)
  Inicializace lokálních dat
- …
- Uložení výsledku
  Obnovení předchozího stavu
  Obnovení adresy aktivačního záznamu volajícího
  Návrat za místo volání

Struktura programu v době běhu – Reprezentace datových typů

- Primitivní datové typy:
  - char, int, double
  - výčet, interval – stejně jako int
- Pole



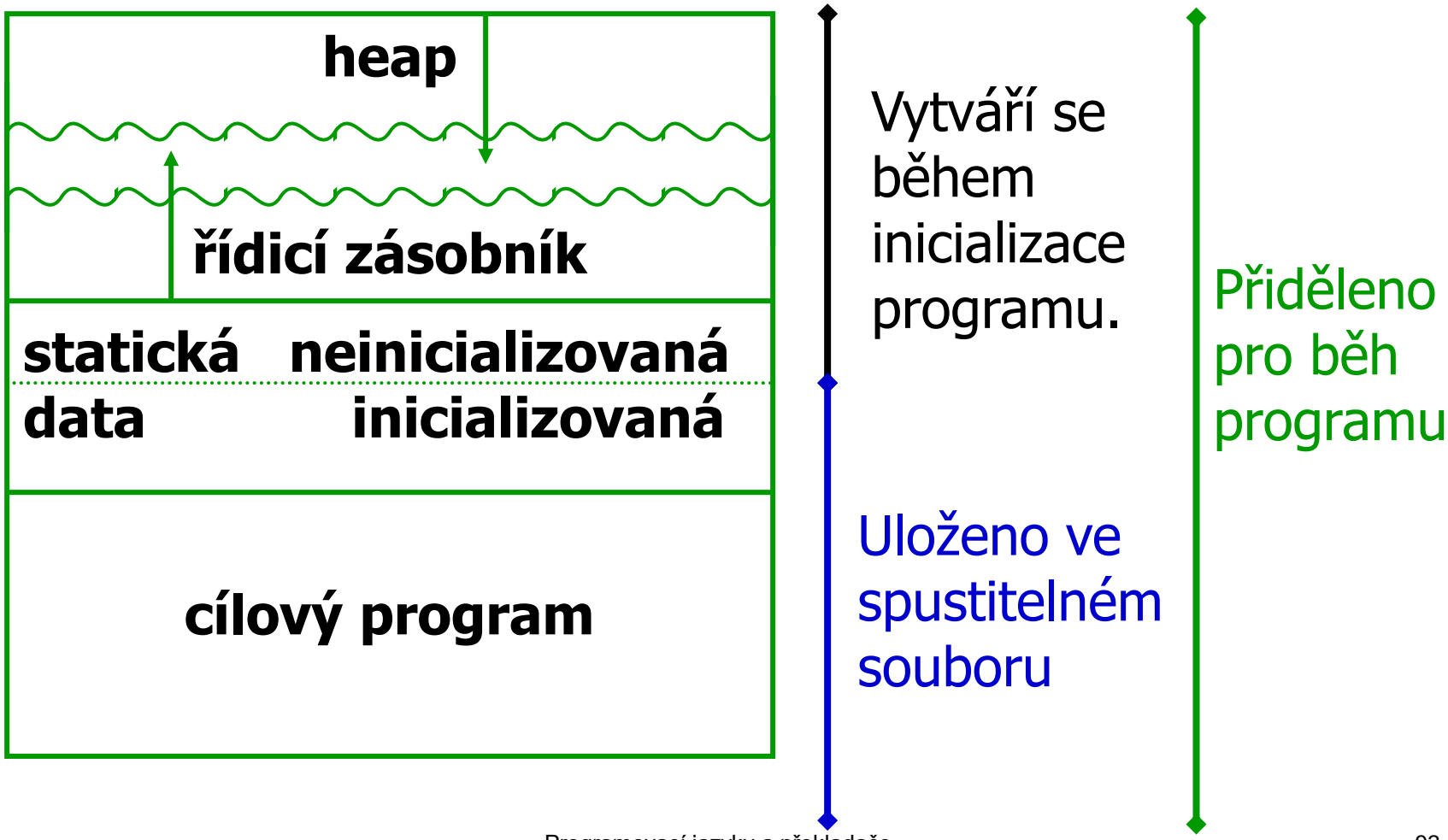
- Záznamy, struktury



Struktura programu v době běhu – **Struktura paměti**

- Cílový program program, konstanty
- Statická data globální a statické proměnné
- Řídicí zásobník aktivační záznamy (lokální proměnné)
- Volná paměť dynamicky alokovaná paměť (hromada – heap)

Struktura programu v době běhu – Schéma obsazení paměti



Struktura programu v době běhu – Přidělování paměti pro aktivační záznamy (1)

- Statické přidělování paměti
  - adresy všech proměnných jsou známy v době překladu
  - lokální proměnné mohou přežít volání programu (viz static v C)
  - počet a velikost prvků musíme znát v době překladu
  - omezená možnost rekurze (sdílení proměnných)
    - obsahy předchozích aktivačních záznamů můžeme odkládat na zásobník

Struktura programu v době běhu – Přidělování paměti pro aktivační záznamy (2)

- Přidělování na zásobníku
  - vhodné pro jazyky s rekurzí
  - paměť se přidělí v okamžiku volání
  - uvolnění paměti proběhne při návratu
  - relativní adresování proměnných – offsety jsou známy v době překladu
  - neznáme-li velikost parametrů -> deskriptor [adresa, délka]
    - Příklad: řetězce, otevřená pole

Struktura programu v době běhu – Přidělování paměti pro aktivační záznamy (3)

- Přidělování z volné paměti
  - lokální proměnné mohou přežít aktivaci podprogramu, resp. aktivace volaného podprogramu může přežít aktivaci volajícího podprogramu -> **nelze použít zásobník**
  - alokace paměti při inicializaci aktivace, finalizace až ve chvíli, kdy není potřebná
  - implementace paralelních jazyků