

IFJ/IAL JAK NA PROJEKT

Zbyněk Křivka

Roman Lukáš

FRVŠ 673/2007/G1

Základní informace o projektu

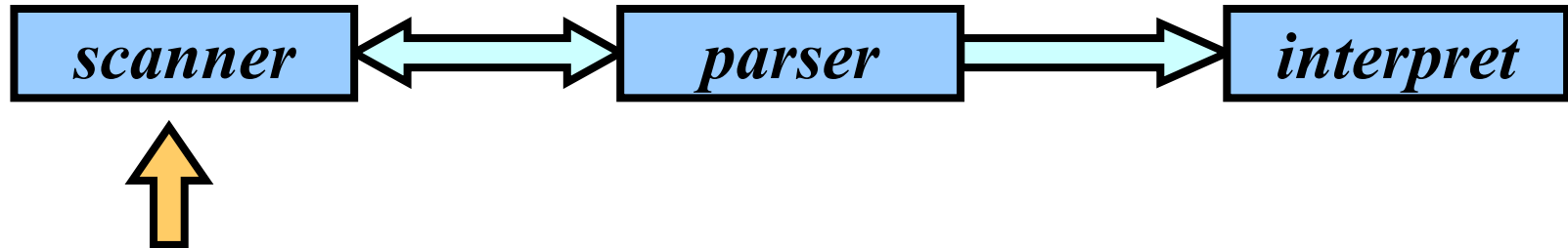
Cíle:

- porozumět základům interpretů a překladačů
- naučit se týmové spolupráci
- naučit se prezentovat svou práci a nápady
- pochopit rekurzi a volání funkcí

Samostatné úkoly na projektu:

- lexikální analyzátor
- syntaktický analyzátor (bez zpracování výrazů)
- syntaktický analyzátor pro výrazy
- interpret
- vestavěné funkce, testování, ...

Struktura projektu

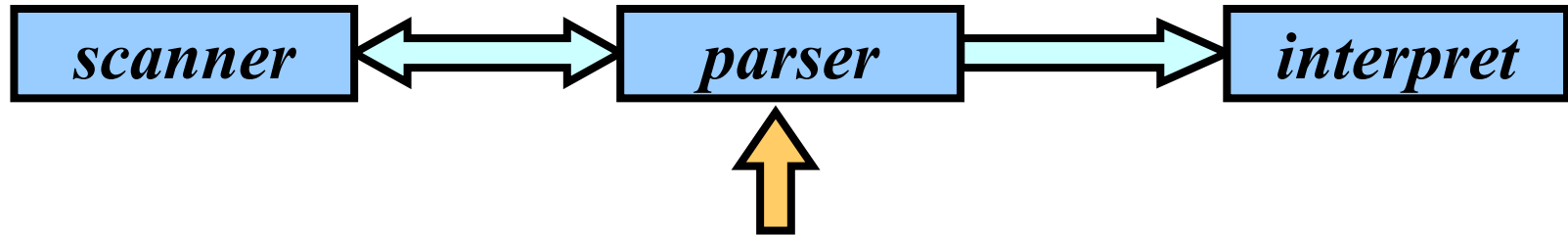


Implementace: Konečný automat

Úkol: Rozpoznat lexikální jednotky ve zdrojovém kódu a vytvořit korespondující tokeny.

Scanner musí být schopen rozpoznat a vrátit jeden token po zavolání funkce např. *get_token*.

Struktura projektu



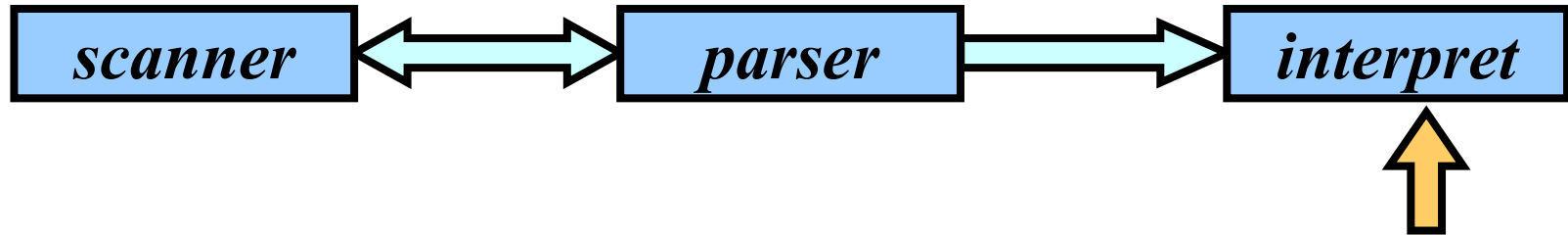
Implementace:

Konečný automat & precedenční synt. analyzátor

Úkol: Zkontrolovat syntaxi, provést sémantické akce, vygenerovat 3-adresný kód

SA je srdcem překladače a je to nejsložitější část tohoto projektu.

Struktura projektu



Implementace:

Smyčka procházející seznamem 3-adresného kódu

Úkol: Vykoná 3-adresný kód vygenerovaný syntaktickým analyzátořem.

Dobře si rozmyslete svoji instrukční sadu. Chytrá volba vám může ušetřit dost času při programování.

Tabulka symbolů

- Řešena pomocí hashovací tabulky, binárního vyhledávacího stromu nebo AVL stromu podle varianty zadání.
- **Struktura:**
 - Klíčem je název identifikátoru; data nesou další informace.
- **Další informace:**
 - *Proměnná*: typ; index (= pořadí v rámci struktury)
 - *Název funkce*: typy vstupních a výstupních parametrů; byla / nebyla již definována
 - *Návěští*: Návěští již bylo nalezeno / na název návěští byl nalezen zatím jen skok

Doporučení: Vytvořit dvě instance TS = lokální, globální

Deklarace glob. proměnných

- Uložit to tabulky symbolů globální úrovně
- Nagenarovat do interpretu instrukce, které alokují globální datový blok pro tyto typy

Příklad:

```
var int cislo, string ret,  
    double real;
```



TS globální úrovně:

Klíč: cislo **Data:** (Typ: i, pozice: 0)

Klíč: ret **Data:** (Typ: s, pozice: 1)

Klíč: real **Data:** (Typ: d, pozice: 2)

Nagenarované instrukce:

Alokuj globálně na poz 0 prostor pro int

Alokuj globálně na poz 1 prostor pro str

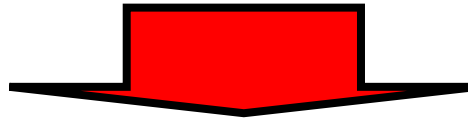
Alokuj globálně na poz 2 prostor pro dbl

Deklarace funkcí

- Sémantická kontrola parametrů, pokud už byla funkce definována/deklarována. Uložit do TS globální úrovně.
- Negerovat nic

Příklad:

```
void f(int, string, double)
int g(int, int)
```



TS globální úrovně:

Klíč: f **Data:** (Typy: „visd“, Definována: NE)

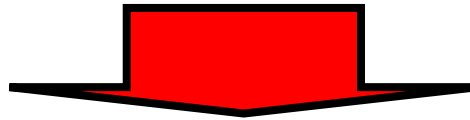
Klíč: g **Data:** (Typy: „iii“, Definována: NE)

Definice funkcí: Hlavička

- Sémantická kontrola parametrů, pokud už byla funkce definována/deklarována. Uložit do TS globální úrovně. Jednotlivé parametry uložit jako proměnné do TS lokální úrovně. Nagenerovat návěští pro skok na danou funkci.
-

Příklad:

```
void f(int a; string s)
```



TS globální úrovně:

Klíč: f **Data:** (Typy: „vis“, Definována: ANO)

TS lokální úrovně:

Klíč: a **Data:** (Typ: i, pozice: 0)

Klíč: s **Data:** (Typ: s, pozice: 1)

Nagenerované instrukce:

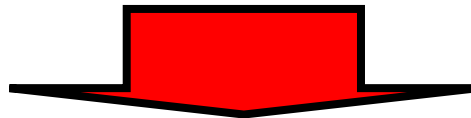
Návěští funkce f

Definice funkcí: Lokální proměnné

- Sémantická kontrola, jestli se název neshoduje s parametrem funkce... Jednotlivé proměnné uložit do TS lokální úrovně. Nagenerovat do interpretu instrukce, které alokují lokální datový blok pro tyto typy

Příklad:

```
void f(int a; string s)
var int c, double d;
```



TS lokální úrovně:

Klíč: a **Data:** (Typ: i, pozice: 0)

Klíč: s **Data:** (Typ: s, pozice: 1)

Klíč: c **Data:** (Typ: i, pozice: 2)

Klíč: d **Data:** (Typ: d, pozice: 3)

Nagenerované instrukce:

Alokuj lokálně na poz 2 prostor pro int

Alokuj lokálně na poz 3 prostor pro dbl

Konec jistého bloku: Sémantické akce

- **Konec funkce:**
- **Sémantické kontroly:**
 - Byly všechny návěští, které obsahuje tabulka symbolů, nalezeny v těle funkce? (Nebyl na neexistující návěští vytvořen pouze skok?)
- **Dále provést:**
 - Vyprázdnit tabulku symbolů lokální úrovně + nagenarovat instrukce, které dealokují datový blok lokálních proměnných
- **Konec programu:**
- **Sémantické kontroly:**
 - Byly všechny deklarované funkce i definované?
 - Byla nalezena funkce `main` a obsahovala správné parametry?
- **Dále provést:**
 - Vyprázdnit tabulku symbolů globální úrovně + nagenarovat instrukce, které dealokují datový blok globálních proměnných

Generování kódu pro jednotlivé příkazy

- *Poznámka:* Syntaktická a sémantická analýza výrazu bude popsána dále

```
id := <výraz>;
```

- **Sémantika:** kontrola typů $id \leftrightarrow \text{výraz}$, kontrola deklarace id
- Nagenarování instrukce, která zkopíruje hodnotu výrazu do paměti alokované pro hodnotu proměnné id .
- **POZOR!** Do paměti přistupovat pomocí indexů (známe z tabulky symbolů), rozlišovat lokální a globální proměnné!

```
id: & goto id; & if <výraz> goto id;
```

- Jen pro *id*: sémantická kontrola, zda se již návěští nevyskytovalo v dané funkci.
- případná registrace návěští do tabulky symbolů
- Nagenarování ekvivalentní instrukce
- **Pozor na jedinečnost názvu z hlediska globální úrovně!**

Generování kódu pro jednotlivé příkazy

return <výraz>; & **return;** & *Konec funkce*

- **Sémantika:**

- **return <výraz>** \leftrightarrow typ výrazu = typ návratové hodnoty funkce
- **return** \leftrightarrow typ výrazu = typ návratové hodnoty funkce je void

- **Generování instrukcí které:**

- Pro **return <výraz>** uloží hodnotu výrazu na dané místo v lokálních datech
- Z lokálního datového bloku typu „zásobník“ zjistí, na jakou adresu se má skočit (následník instrukce, ze které byla funkce volána)
- Z lokálního datového bloku odstraní všechny proměnné, které byly pro tuto funkci použity
- Na danou instrukci skočí.

Zpracování výrazu: Syntaktická analýza

- Provádí speciální syntaktický analyzátor založený na precedenční syntaktické analýze.
- Pouhé zavolání funkce je také chápáno jako výraz

Gramatika pro generování výrazů
včetně volání funkcí:

1. $E \rightarrow id()$
2. $E \rightarrow id(E)$
3. $E \rightarrow id(L)$
4. $L \rightarrow L, E$
5. $L \rightarrow E, E$
6. $E \rightarrow id$
7. $E \rightarrow const$
8. $E \rightarrow (E)$
9. $E \rightarrow E \text{ op1 } E$
10. $E \rightarrow E \text{ op2 } E$

...

- Konstrukce tabulky pro výrazy:
(viz přednášky)

• **Změny pro precedenční tabulku zahrnující i volání funkcí:**

- Operátor “,” chápán jako levě asociativní a s nejmenší prioritou

- V tabulce navíc políčko: $id = ($

Proč? ☺

Sémantické akce pro jednotlivá pravidla

$E \rightarrow id$

- **Sémantická akce:** kontrola deklarace proměnné id
 - Přiřadit atributu nonterminálu E index proměnné id a její typ
-

$E \rightarrow const$

- Nagenerovat novou lokální proměnnou typu, který specifikuje konstanta $const$ + naplnění hodnoty: Nagenerovat odpovídající instrukce. Dále řešit jako minulý případ.
-

$E_1 \rightarrow (E_2)$

- Pouze kopie atributů mezi proměnnými
-

$E_1 \rightarrow E_2 \text{ op } E_3$

- **Sémantická akce:** kontrola, zda je operace povolena nad danými typy. (popř. po typové konverzi) necht' t = typ výsledku.
- Nagenerovat novou lokální proměnnou typu t a atributu nonterminálu E_1 přiřadit její index a typ. Nagenerovat instrukci provádějící danou operaci (indexy proměnných jsou známy z atributů E_2 a E_3)

Sémantické akce pro volání funkce

1. $E \rightarrow id()$ 2. $E \rightarrow id(E)$ 3. $E \rightarrow id(L)$ 4. $L \rightarrow L, E$ 5. $L \rightarrow E, E$

Sémantická kontrola:

- Postupně „sbírat“ typy jednotlivých parametrů. Na závěr zkontrolovat s parametry definice/deklarace funkce.

Nagenerování instrukcí, které postupně na „zásobník“ reprezentující lokální data vloží:

- aktuální pozici instrukce (bude potřeba při návratu z funkce)
- místo proměnnou, kam bude uložen výsledek funkce
- hodnoty jednotlivých výrazů (tedy proměnných, ve kterých je uložen vždy výsledek výrazu) do lokální datové části.

Nagenerování instrukce, která „skočí“ na návěští odpovídající dané funkci

Ukázkový zdrojový kód

```
int f(int i) {  
    if (i > 1) goto calc;  
    return 1;  
    calc:  
    return i * f(i-1);  
}
```

```
void main() {  
    printstring("5! = "+int2str(f(5)));  
}
```

Scanner - příklad

ZDROJOVÝ KÓD

```
int f(int i) {
    if (i > 1) goto calc;
    return 1;
    calc:
    return i * f(i-1);
}

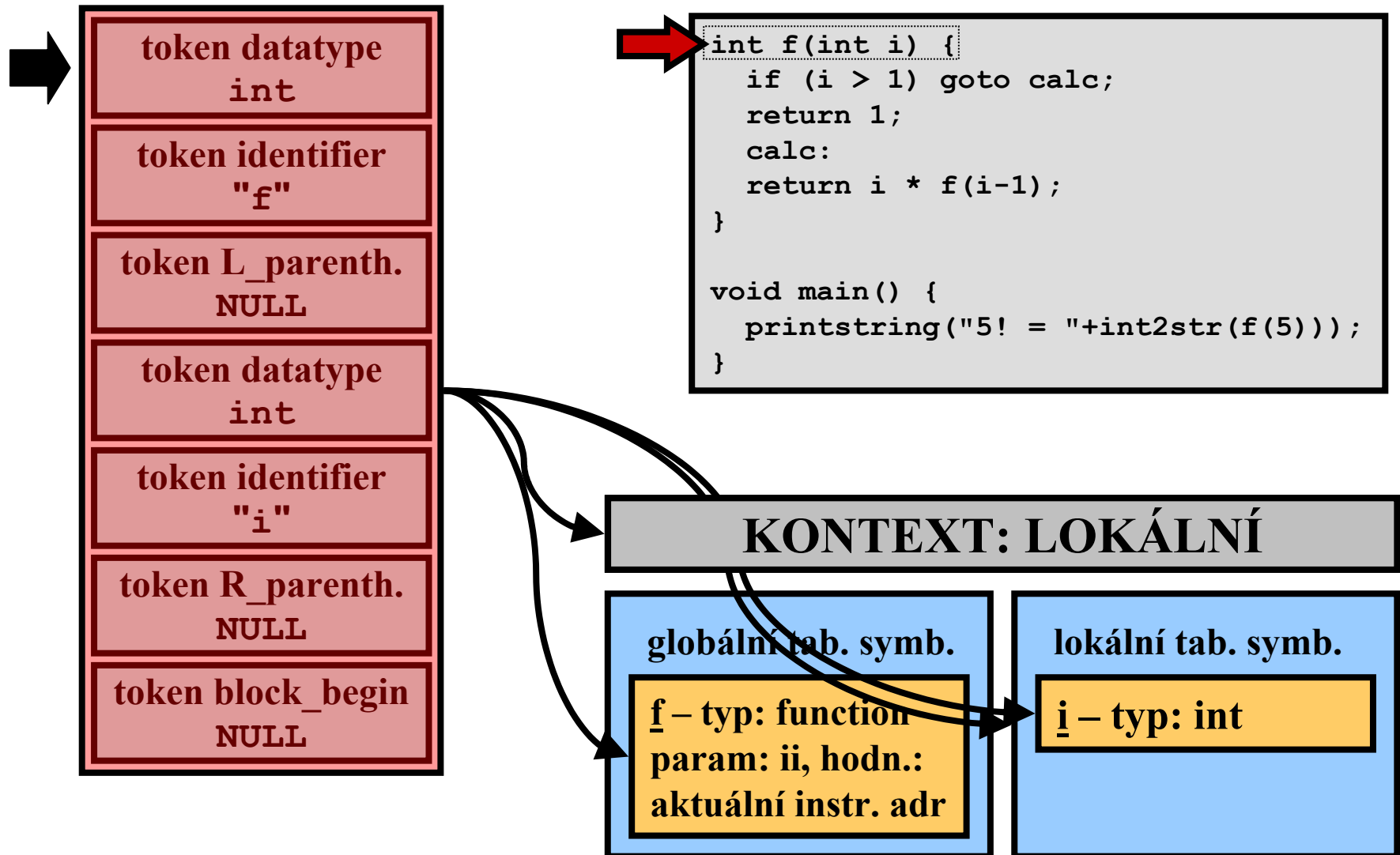
void main() {
    printstring("5! = "+int2str(f(5)));
}
```

TYPY TOKENŮ

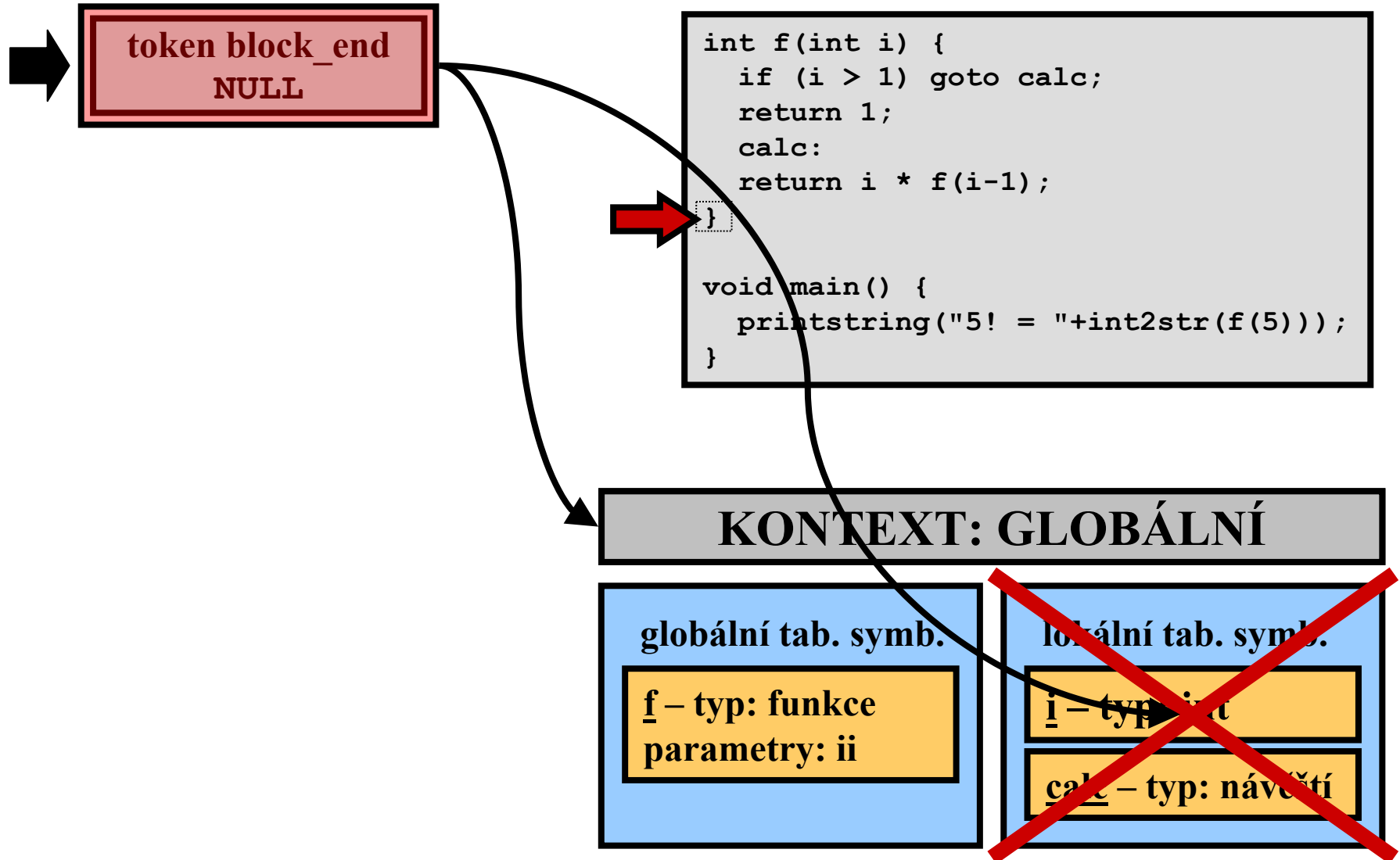
identifikátor
datový typ
klíč.slovo
literál int/flt/str
L/R_závorka
blok_záč/konec
operátor +/-/...
návěští
ukončovač
čárka
přiřazení
EOF



Parser – začátek funkce - příklad



Parser – konec funkce - příklad



Parser – příklad výrazu

token return
NULL

*precedenční
synt. analýza*

viz IFJ09 – strana 4

token terminator
NULL

```
int f(int i) {
    if (i > 1) goto calc;
    return 1;
calc:
    return i * f(i-1);
}

void main() {
    printstring("5! = "+int2str(f(5)));
}
```

KONTEXT: LOKÁLNÍ

globální tab. symb.

f – typ: funkce
parametry: ii

lokální tab. symb.

i – typ: int

calc – typ: návěští

Parser – příklad generování 3AC

```
int f(int i) {
    if (i > 1) goto calc;
    return 1;
calc:
    return i * f(i-1);
}
```

```
void main() {
    printstring("5! = "+int2str(f(5)));
}
```

Zaváděcí instrukce:

CALL  main

EXIT

Páska 3-adresného kódu

LABEL

, , ,

GR_CMP

i, 1, T1

GOTO

T1, , calc

RETURN

1, , ret

LABEL

, , ,

SUB

i, 1, T2

PUSH

T2, ,

CALL

f, , T3

MUL

i, T3, T4

RETURN

T4, , ret

...

Interpreter – volání podprogramů

PUSH
param

CALL
f, ,ret_addr

1. Označit současný vrchol zásobníku jako vrchol zásobníku funkce a uložit předchozí vrchol zásobníku jako nadřazený datový zásobník
2. Vložit parametry volání funkce na zásobník
3. Uložit návratovou adresu instrukční pásky
4. Uložit adresu pro uložení návratové hodnoty do nadřazeného datového zásobníku (pokud funkce má návratovou hodnotu)
5. Naalokovat místo pro lokální proměnné

Interpret – volání podprogramů

PUSH
param

CALL
f, ,ret_addr

...

vykonat tělo funkce...

Interpret – volání podprogramů

PUSH
param

CALL
f,,ret_addr

...

RETURN
value,,ret

1. Uložit návratovou hodnotu na adresu v nadřazeném datovém zásobníku
2. Obnovit nadřazený datový zásobník do současného vrcholu zásobníku (= uvolnit lokální proměnné)
3. Vrátit se na uloženou pozici v instrukční pásce