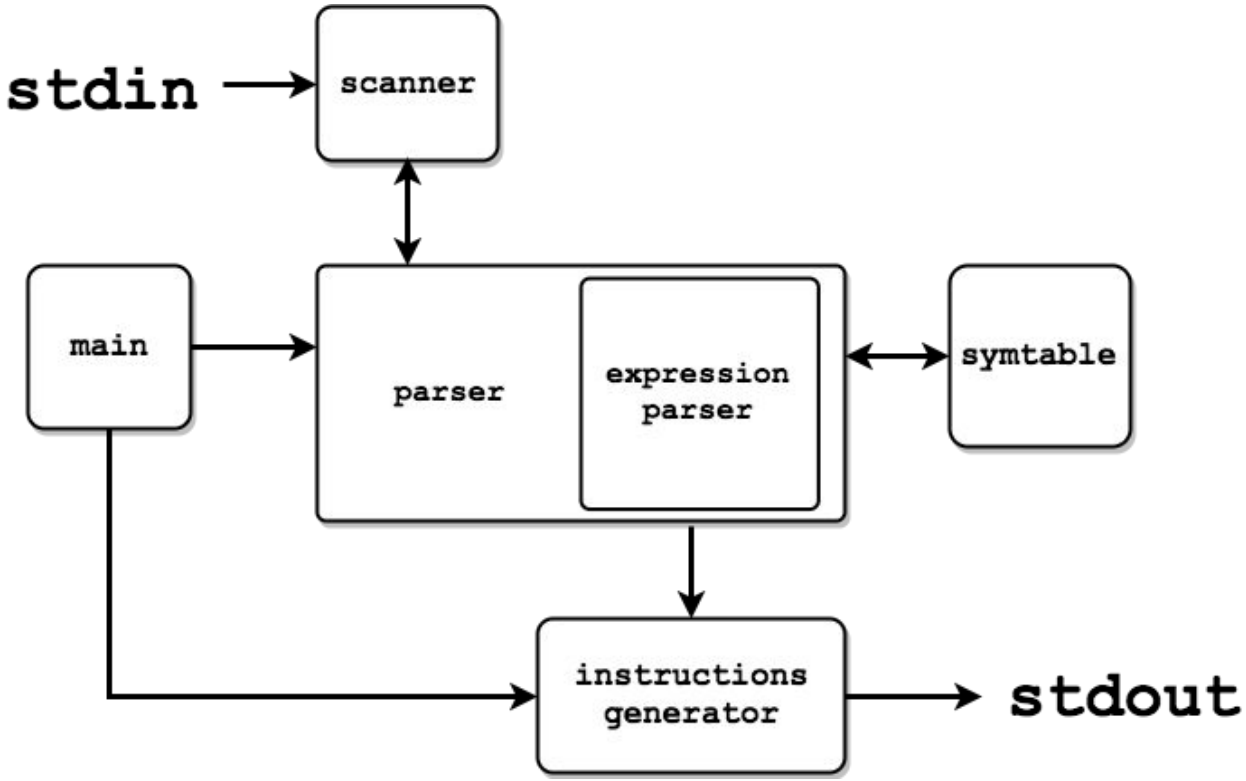
Implementace překladače imperativního jazyka IFJ17

Tým 025, varianta I

Naňo Andrej *(vedúci týmu)*	**xnanoa00**
Marko Peter	**xmarko15**
Švanda Jan	**xsvand06**
Mechl Stanislav	**xmechl00**

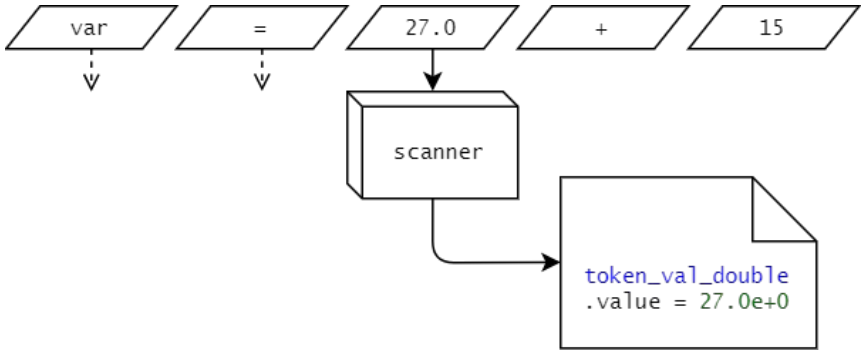
Štruktúra prekladača



Lexikálny analyzátor

```
bool get_next_token(FILE *f, Token_t *token);
```

- Konečný automat
- Čtení lexémů
- Vytváření tokenů



Syntaktický analyzátor

- Postupne volá scanner
- Kontroluje syntax tokenov
- Robí sémantické kontroly
- Vytvára nové inštrukcie

Syntaktický analyzátor / LL(1) gramatika

NONTERMINALS

<Program>	➡	<Head> <Scope> token_eof
<Head>	➡	ε
<Head>	➡	<FunctionDecl> <Head>
<Head>	➡	<FunctionDef> <Head>
<Scope>	➡	token_scope <CompoundStmt> token_endscope
<FunctionDecl>	➡	token_declare <Function>
<FunctionDef>	➡	<Function> token_eol <CompoundStmt> token
<Function>	➡	token_function token_identifier token_lbr
<ParamList>	➡	ε
<ParamList>	➡	<Param> <NextParam>
<NextParam>	➡	token_comma <Param> <NextParam>
<NextParam>	➡	ε
<Param>	➡	token_identifier token_as token_datatype

```
void NT_Program();
void NT_Head();
void NT_Scope();
void NT_CompoundStmt();
void NT_FunctionDec();
void NT_FunctionDef();
void NT_ParameterList();
void NT_ParamList();
void NT_Param();
void NT_NextParam();
char *NT_VarDec();
void NT_VarDef();
void NT_AssignStmt();
void NT_IfStmt();
void NT_WhileStmt();
void NT_InputStmt();
void NT_PrintStmt();
void NT_ReturnStmt();
void NT_TermList();
void NT_ExprList();
void NT_NextExpr();
void NT_CallExpr();
```

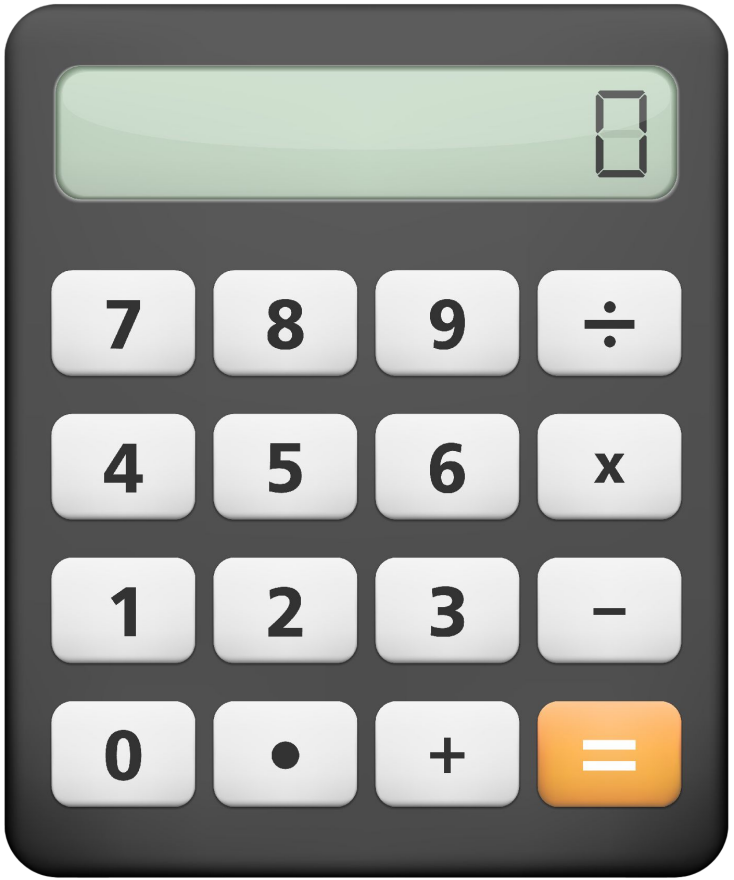
Syntaktický analyzátor / funkcia match()

```
bool match(token_type expected_token_type);
```

- Očakávaný token vracia **TRUE**, inak **FALSE**
- Vyčistí pamäť alokovanú tokenom
- Zavolá scanner a presunie sa na ďalší token

Spracovanie výrazov

- Infix na postfix
- Zasobnikové inštrukcie
- Volanie funkcí
- Implicitné konverzie
- Typová kontrola



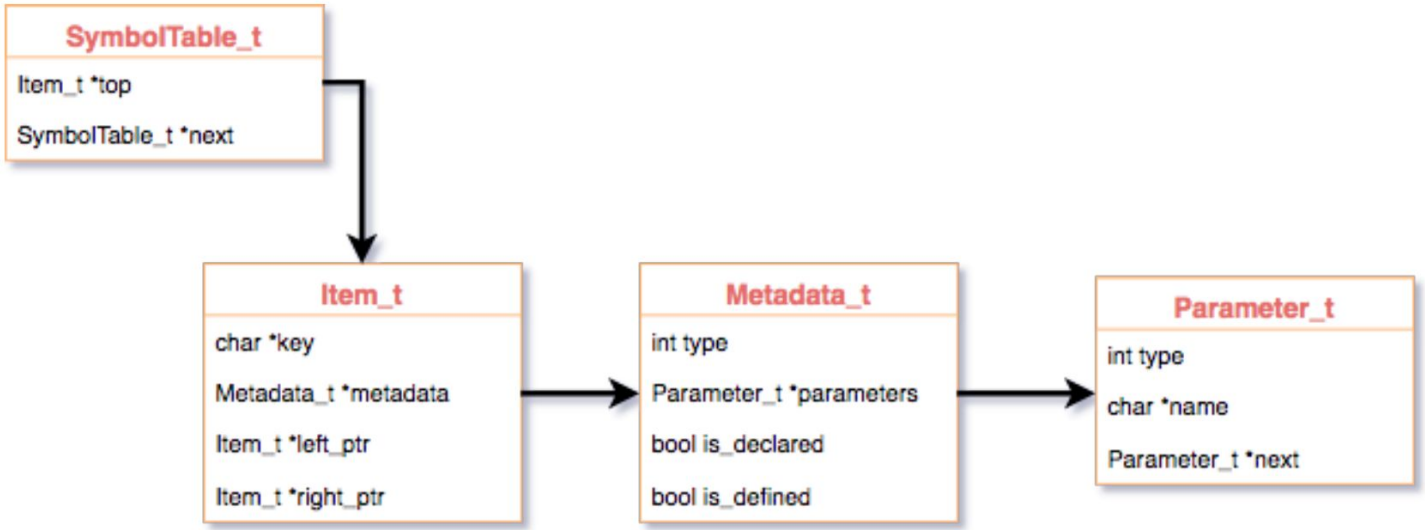
Sémantická analýza

- Pokus o redefiníciu
- Rovnaké parametre v predpise funkcie
- Nezhodná definícia s deklaráciou

Príklad : *Porovnanie parametrov deklarácie a definície funkcie*

```
// COMPARE PARAMETER LISTS
if (param_list_cmp(function_metadata→parameters,  function_parameters) == false )
{
    raise_error(E_SEM_DEF, "Function definition parameters do not match with it's declaration.");
    definition_error = true;
}
```

Tabuľka symbolov

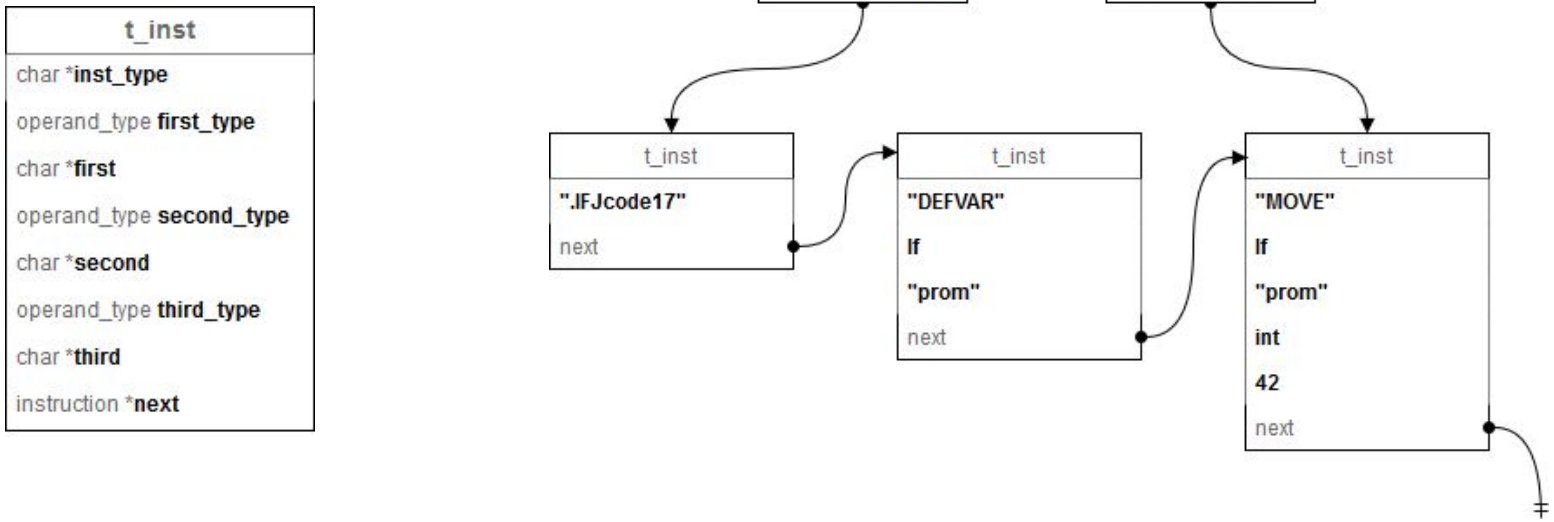


Tabuľka symbolov / datové štruktúry a algoritmy

- Binárny vyhľadávací strom
- Výpočtová zložitosť vyhľadávania a vkladania
  - Priemerná **O(log n)**
  - V najhoršiom prípade ak strom nieje vyvážený **O(n)**

Generování kódu / datové struktury

- Instrukce jsou uspořádány do jednosměrně vázaného seznamu



Generování kódu / tisk do výstupního souboru

- Hlavička
- Využité vestavěné funkce
- Zbytek instrukcí uložených v seznamu

Spracovanie chýb

```
void raise_error(int error_code, const char *msg, ... );
```

- Nastavenie globálnej chyby
- Program vráti vždy kód prvej chyby
- Vypisuje chybové správy pre všetky chyby

```
factorial.ifj:6:ERROR[2][Syntax error]: Wrong expression syntax expected EOL
```

Pomocné debug výpisy

```
[        --- START---
[
[DEBUG] match token_declare
[DEBUG] match token_function
[DEBUG] match token_identifier >> 'factorial'
[DEBUG] match token_lbrace
[DEBUG] match token_identifier >> 'n'
[DEBUG] match token_as
[DEBUG] match token_integer
[DEBUG] match token_rbrace
[DEBUG] match token_as
[DEBUG] match token_integer

        + New item in FUNCTIONS symtable [first item]
                [KEY] :  'factorial'
               [TYPE] :  Integer
```

GCC s nastavením **-DDEBUG**

\0

Ďakujeme za pozornosť