IFJ projekt dokumentace

**Tým xzajic22, varianta TRP**

**Rozšíření: BOOLTHEN, CYCLES, STRNUM, OPERATORS, FUNEXP**

# Seznam členů týmu:

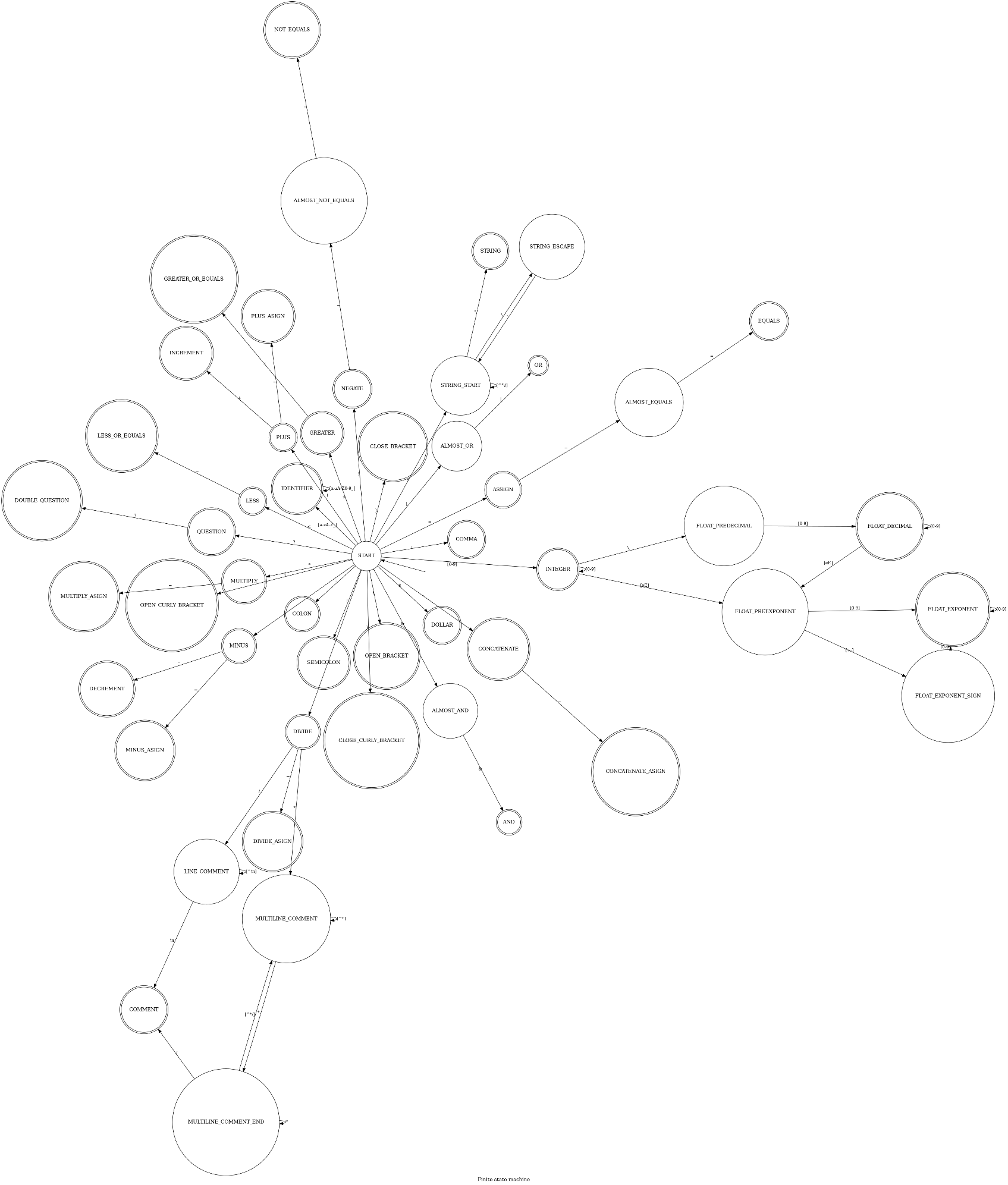
* Michal Cejpek – xcejpe05
* Jiří Gallo – xgallo04
* Jakub Kratochvíl – xkrato67
* Jan Zajíček – xzajic22 (vedoucí)

# Rozdělení bodů:

* xzajic22: 25%
* xkrato67: 25%
* xgallo04: 25%
* xcejpe05: 25%

# Rozdělení práce:

* xzajic22:
  + Programová dokumentace + komentáře
  + Rozšíření CYCLES
* xkrato67:
  + Tabulka symbolů
  + Práce na abstraktním syntaktickém stromu, generování kódu
  + Rozšíření STRNUM
  + Práce na OPERATORS
* xcejpe05:
  + Rozšíření BOOLTHEN
  + Gramatika
  + LL tabulka
  + Úprava syntaktického analyzátoru podle množiny FIRST
* xgallo04
  + Lexikální analyzátor
  + Syntaktický analyzátor
  + Generování kódu
  + Optimalizace
  + Abstraktní syntaktický strom
  + Výrazy
  + Rozšíření FUNEXP



# Gramatika

program -> START STATEMENT\_LIST\_MAIN .

START -> <?php declare(strict\_types=1); .

STATEMENT\_LIST\_MAIN -> EPSILON .

STATEMENT\_LIST\_MAIN -> STATEMENT\_IF STATEMENT\_LIST\_MAIN .

STATEMENT\_LIST\_MAIN -> STATEMENT\_EXPRESSION; STATEMENT\_LIST\_MAIN .

STATEMENT\_LIST\_MAIN -> STATEMENT\_WHILE STATEMENT\_LIST\_MAIN .

STATEMENT\_LIST\_MAIN -> STATEMENT\_FOR STATEMENT\_LIST\_MAIN .

STATEMENT\_LIST\_MAIN -> STATEMENT\_RETURN STATEMENT\_LIST\_MAIN .

STATEMENT\_LIST\_MAIN -> STATEMENT\_BREAK STATEMENT\_LIST\_MAIN .

STATEMENT\_LIST\_MAIN -> STATEMENT\_CONTINUE STATEMENT\_LIST\_MAIN .

STATEMENT\_LIST\_MAIN -> STATEMENT\_FUNCTION STATEMENT\_LIST\_MAIN .

STATEMENT\_IF -> if( STATEMENT\_EXPRESSION ){ STATEMENT\_LIST } STATEMENT\_IF2 .

STATEMENT\_IF2 -> EPSILON .

STATEMENT\_IF2 -> elseif( STATEMENT\_EXPRESSION ){ STATEMENT\_LIST } STATEMENT\_IF2 .

STATEMENT\_IF2 -> STATEMENT\_IF3 .

STATEMENT\_IF3 -> else{ STATEMENT\_LIST } .

STATEMENT\_WHILE -> while( STATEMENT\_EXPRESSION ){ STATEMENT\_LIST } .

STATEMENT\_BREAK -> break STATEMENT\_BREAK2 .

STATEMENT\_BREAK2 -> CONSTANT\_INTEGER .

STATEMENT\_CONTINUE -> continue STATEMENT\_CONTINUE2 .

STATEMENT\_CONTINUE2 -> CONSTANT\_INTEGER .

STATEMENT\_FOR -> for( STATEMENT\_FOR2, STATEMENT\_FOR2, STATEMENT\_FOR2 ){ STATEMENT\_LIST } .

STATEMENT\_FOR2 -> STATEMENT\_EXPRESSION .

STATEMENT\_FOR2 -> EPSILON .

STATEMENT\_LIST -> EPSILON .

STATEMENT\_LIST -> STATEMENT\_IF STATEMENT\_LIST .

STATEMENT\_LIST -> STATEMENT\_EXPRESSION ; STATEMENT\_LIST .

STATEMENT\_LIST -> STATEMENT\_WHILE STATEMENT\_LIST .

STATEMENT\_LIST -> STATEMENT\_RETURN STATEMENT\_LIST .

STATEMENT\_RETURN -> return STATEMENT\_RETURN2 .

STATEMENT\_RETURN2 -> STATEMENT\_EXPRESSION; .

STATEMENT\_RETURN2 -> ; .

STATEMENT\_FUNCTION -> function IDENTIFIER( FUNCTION\_PARAMETER\_LIST ): RETURN\_TYPE { STATEMENT\_LIST STATEMENT\_RETURN } .

FUNCTION\_PARAMETER\_LIST -> TERM\_TYPE IDENTIFIER FUNCTION\_PARAMETER\_LIST2 .

FUNCTION\_PARAMETER\_LIST2 -> EPSILON .

FUNCTION\_PARAMETER\_LIST2 -> ,TERM\_TYPE IDENTIFIER FUNCTION\_PARAMETER\_LIST2 .

STATEMENT\_EXPRESSION -> EXPRESSION\_FUNCTION\_CALL .

EXPRESSION\_FUNCTION\_CALL -> IDENTIFIER( PARAMETER\_LIST ) .

PARAMETER\_LIST -> STATEMENT\_EXPRESSION PARAMETER\_LIST2 .

PARAMETER\_LIST2 -> EPSILON .

PARAMETER\_LIST2 -> ,STATEMENT\_EXPRESSION PARAMETER\_LIST2 .

STATEMENT\_EXPRESSION -> EXPRESSION\_CONSTANT .

EXPRESSION\_CONSTANT -> CONSTANT\_INTEGER .

EXPRESSION\_CONSTANT -> CONSTANT\_FLOAT .

EXPRESSION\_CONSTANT -> CONSTANT\_STRING .

EXPRESSION\_CONSTANT -> EXPRESSION\_CONSTANT\_BOOL .

STATEMENT\_EXPRESSION -> EXPRESSION\_VARIABLE .

EXPRESSION\_VARIABLE -> $ IDENTIFIER .

STATEMENT\_EXPRESSION -> EXPRESSION\_UNARY\_OPERATOR .

EXPRESSION\_UNARY\_OPERATOR -> UNARY\_OPERATOR STATEMENT\_EXPRESSION .

STATEMENT\_EXPRESSION -> EXPRESSION\_BINARY\_OPERATOR .

EXPRESSION\_BINARY\_OPERATOR -> STATEMENT\_EXPRESSION BINARY\_OPERATOR STATEMENT\_EXPRESSION .

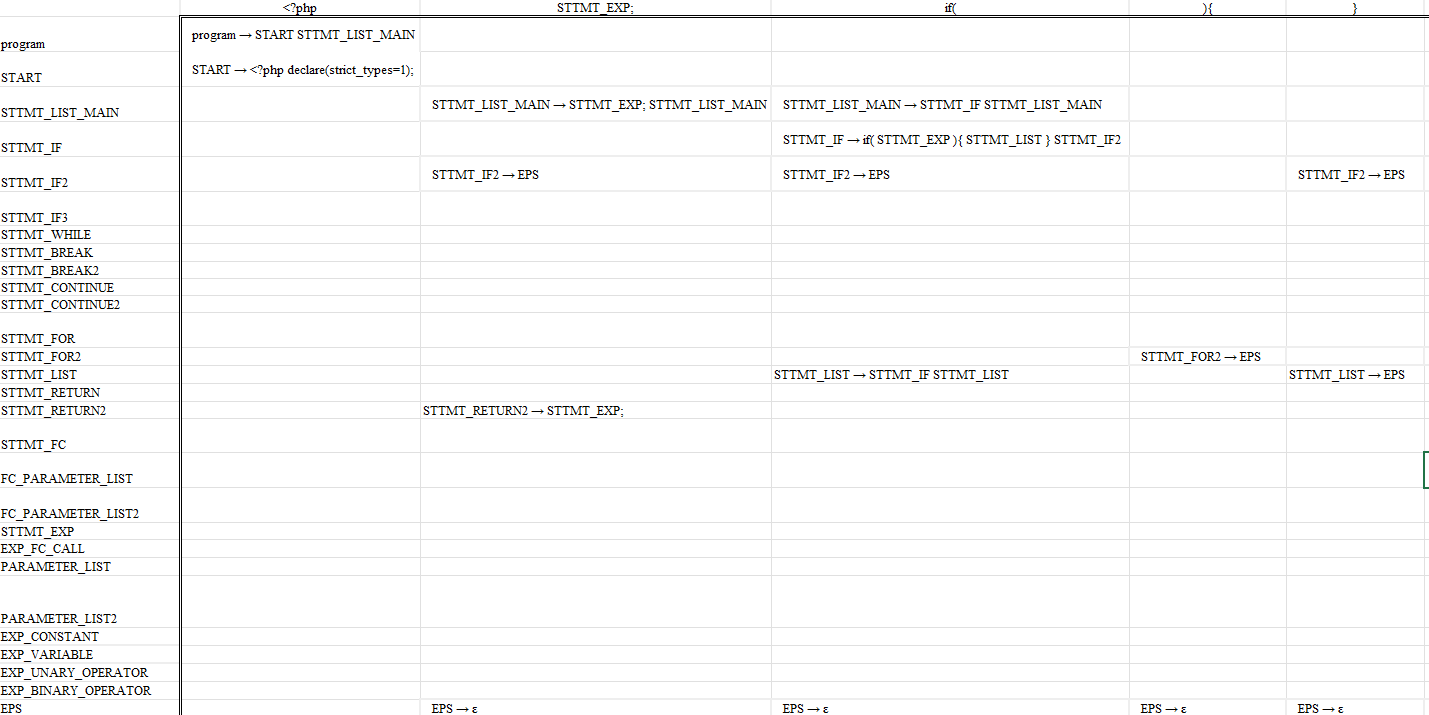
EPSILON -> .

# Množina FIRST

* Terminal\_expression
  + CONSTANT\_INTEGER
  + IDENTIFIER
  + CONSTANT\_FLOAT
  + CONSTANT\_STRING
  + CONSTANT\_BOOL
  + (
  + $
* Expression
  + CONSTANT\_INTEGER
  + IDENTIFIER
  + CONSTANT\_FLOAT
  + CONSTANT\_STRING
  + CONSTANT\_BOOL
  + (
  + $
  + UNARY\_OPERATOR
* Statement
  + TOKEN\_IF
  + TOKEN\_WHILE
  + TOKEN\_RETURN
  + TOKEN\_FOR
  + TOKEN\_BREAK
  + TOKEN\_CONTINUE
  + TOKEN\_OPEN\_BRACKET
  + Expression
* Statement list main
  + STATEMENT\_EXPRESSION
  + if
  + while
  + for
  + return
  + function
* Statement list
  + if
  + while
  + break
  + CONSTANT\_INTEGER
  + continue
  + return
  + IDENTIFIER
  + CONSTANT\_FLOAT
  + CONSTANT\_STRING
  + CONSTANT\_BOOL
  + (
  + $
  + UNARY\_OPERATOR

# LL Tabulka

Graphical user interface, text, application, email

Popis byl vytvořen automaticky

Graphical user interface, text

Popis byl vytvořen automatickyGraphical user interface, text, application

Popis byl vytvořen automaticky

Graphical user interface, text, application, email

Popis byl vytvořen automatickyGraphical user interface, application

Popis byl vytvořen automaticky

# Struktura projektu

* **Root**
  + dokumentace.pdf
    - Dokumentace projektu
  + ast.c
    - Abstraktní syntaktický strom
  + code\_generator.c
    - Generování kódu
  + emmiter.c
    - Pomocný soubor pro generování kódu, vypisuje instrukce na výstup
  + pointer\_hashtable.c
    - Hashovací tabulka pro optimalizace
  + lexer\_processor.c
    - Získává další token ze souboru
  + lexer.c
    - Lexikální analýza
  + optimizer.c
    - Optimalizace výstupního kódu
  + parser.c
    - Syntaktický analyzátor rekurzivního sestupu shora dolů
  + main.c
    - Volá syntaktickou analýzu
  + string\_builder.c
    - Sestavuje výstupní řetězec
  + symtable.c
    - Tabulka symbolů

# Precedenční analýza

Pro zpracování výrazů se používá metoda precedence climbing bez použití precedenční tabulky, místo které je použita priorita operátorů. Nejdříve se provede zpracování prefix operátorů, které sdílí prioritu s ostatními operátory. Následně se provede načtení "ukončujícího výrazu" včetně postfix operátorů a poté pokud se dále vyskytují binární operátory, tak se provede jejich zpracování. Ukončující výraz je buďto proměnná, volání funkce nebo závorky – pro obsah závorek nebo seznamu parametrů se pouští precedenční analýza od znova s počáteční prioritou 0. Postfix operátory nepodporují prioritu a jsou tedy aplikovány pouze na "ukončující výraz" nebo další postfix operátor.

# Abstraktní syntaktický strom

Pro návrh stromu je využito principů tříd OOP včetně virtuálních metod.

Hierarchie tříd je následující

* Statement (s metodami serialize, getChildren, duplicate, free)
  + StatementList (s metoddami addStatement, append)
  + Expression (s metodou getType)
    - Expression\_\_Constant
    - Expression\_\_Variable
    - Expression\_\_FunctionCall (s metodou addArgument)
    - Expression\_\_BinaryOperator
    - Expression\_\_PrefixOperator
    - Expression\_\_PostfixOperator
  + StatementIf
  + StatementWhile
  + StatementFor
  + StatementReturn
  + StatementExit
  + StatementContinue
  + StatementBreak
  + Function (s metodou addParameter)

# Optimalizátor

Optimalizátor pracuje pouze na úrovni syntaktického stromu a obsahuje následující optimalizace:

* Přetypování konstant
* Výpočet konstantních výrazů
* Vykonání příkazu if s konstantní podmínkou
* Vykonání vestavěných funkcí s konstantními parametry
* Odstranění kódu po příkazech return, break a continue
* Vyhodnocení některých výrazů s nedefinovanou proměnou jako chyba
* Rozvinutí smyček
* Odstranění zbytečných přiřazení
* Propagaci konstant
* Spojování příkazů write dohromady

Tyto optimalizace jsou prováděny, dokud nejsou všechny hotové nebo dokud nevyprší časovač omezující množství optimalizací.