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)

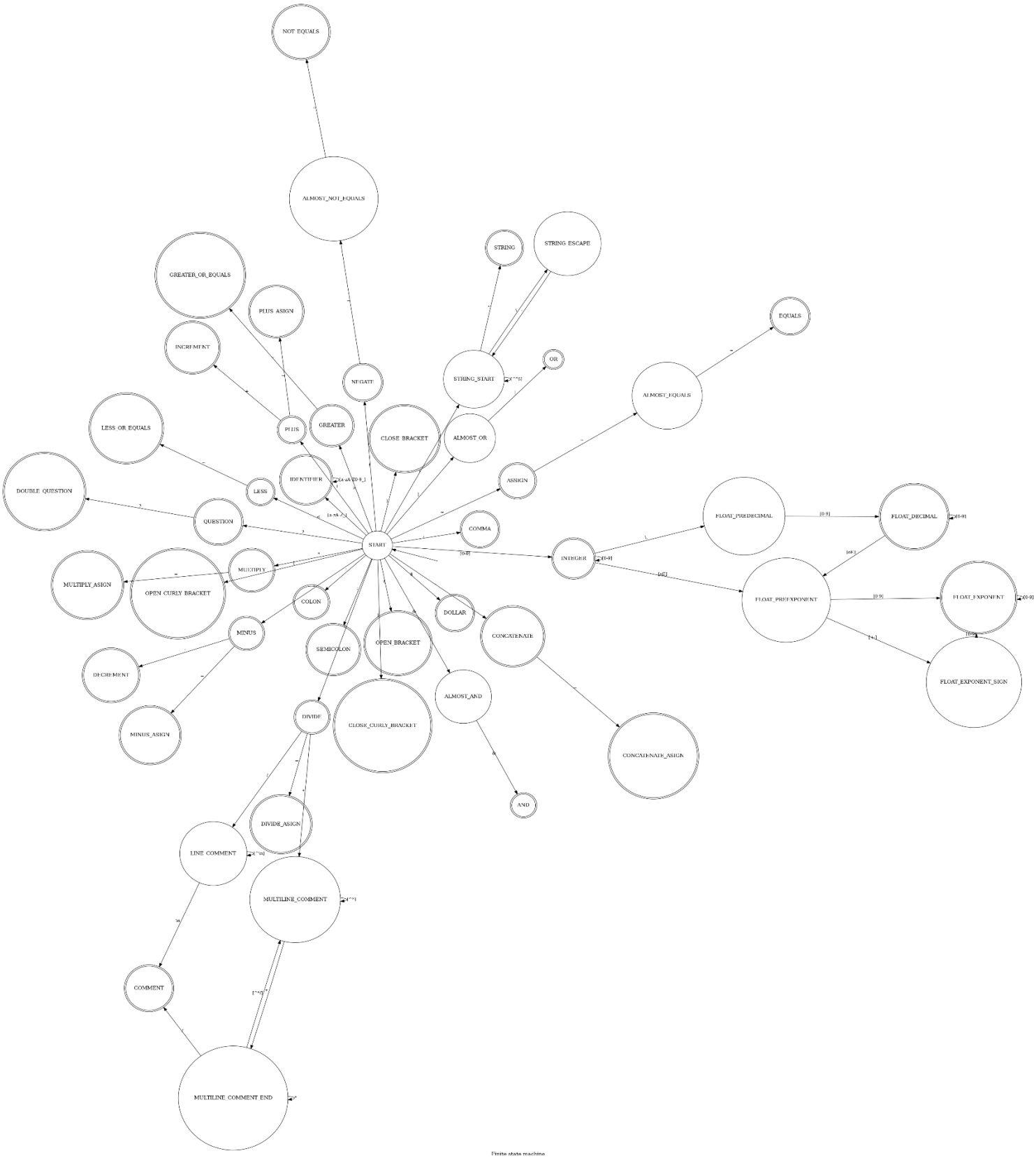
Rozdělení bodů:

- 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

FSM – Diagram lexikálního analyzátoru



Finite state machine

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

	(	$	UNARY_OPERATOR
program			
START			
STTMT_LIST_MAIN			
STTMT_IF			
STTMT_IF2	STTMT_IF2 → EPS	STTMT_IF2 → EPS	STTMT_IF2 → EPS
STTMT_IF3			
STTMT_WHILE			
STTMT_BREAK			
STTMT_BREAK2			
STTMT_CONTINUE			
STTMT_CONTINUE2			
STTMT_FOR			
STTMT_FOR2	STTMT_FOR2 → STTMT_EXP	STTMT_FOR2 → STTMT_EXP	STTMT_FOR2 → STTMT_EXP
STTMT_LIST	STTMT_LIST → STTMT_EXP ; STTMT_LIST	STTMT_LIST → STTMT_EXP ; STTMT_LIST	STTMT_LIST → STTMT_EXP ; STTMT_LIST
STTMT_RETURN			
STTMT_RETURN2			
STTMT_FC			
FC_PARAMETER_LIST			
FC_PARAMETER_LIST2			
STTMT_EXP	STTMT_EXP → (STTMT_EXP)STTMT_EXP → EXP_BINARY_OPERATOR	STTMT_EXP → EXP_VARIABLESTTMT_EXP → EXP_BINARY_OPERATOR	STTMT_EXP → EXP_UNARY_OPERATORSTTMT_EXP → EXP_BINARY_OPERATOR
EXP_FC_CALL			
PARAMETER_LIST	PARAMETER_LIST → STTMT_EXP PARAMETER_LIST2	PARAMETER_LIST → STTMT_EXP PARAMETER_LIST2	PARAMETER_LIST → STTMT_EXP PARAMETER_LIST2
PARAMETER_LIST2			
EXP_CONSTANT			
EXP_VARIABLE		EXP_VARIABLE → $ IDENTIFIER	
EXP_UNARY_OPERATOR			EXP_UNARY_OPERATOR → UNARY_OPERATOR STTMT_EXP
EXP_BINARY_OPERATOR	EXP_BINARY_OPERATOR → STTMT_EXP BINARY_OPERATOR STTMT_EXP	EXP_BINARY_OPERATOR → STTMT_EXP BINARY_OPERATOR STTMT_EXP	EXP_BINARY_OPERATOR → STTMT_EXP BINARY_OPERATOR STTMT_EXP
EPS	EPS → ε	EPS → ε	EPS → ε

	<?php	STTMT_EXP;	if(	){	}
program	program → START STTMT_LIST_MAIN				
START	START → <?php declare(strict_types=1);				
STTMT_LIST_MAIN		STTMT_LIST_MAIN → STTMT_EXP; STTMT_LIST_MAIN	STTMT_LIST_MAIN → STTMT_IF STTMT_LIST_MAIN		
STTMT_IF			STTMT_IF → if(STTMT_EXP){ STTMT_LIST } STTMT_IF2		
STTMT_IF2		STTMT_IF2 → EPS	STTMT_IF2 → EPS		STTMT_IF2 → EPS
STTMT_IF3					
STTMT_WHILE					
STTMT_BREAK					
STTMT_BREAK2					
STTMT_CONTINUE					
STTMT_CONTINUE2					
STTMT_FOR					
STTMT_FOR2				STTMT_FOR2 → EPS	
STTMT_LIST			STTMT_LIST → STTMT_IF STTMT_LIST		STTMT_LIST → EPS
STTMT_RETURN					
STTMT_RETURN2		STTMT_RETURN2 → STTMT_EXP;			
STTMT_FC					
FC_PARAMETER_LIST					
FC_PARAMETER_LIST2					
STTMT_EXP					
EXP_FC_CALL					
PARAMETER_LIST					
PARAMETER_LIST2					
EXP_CONSTANT					
EXP_VARIABLE					
EXP_UNARY_OPERATOR					
EXP_BINARY_OPERATOR					
EPS		EPS → ε	EPS → ε	EPS → ε	EPS → ε

	continue	for(	,	;	return
program					
START					
STTMT_LIST_MAIN		STTMT_LIST_MAIN → STTMT_FOR STTMT_LIST_MAIN			STTMT_LIST_MAIN → STTMT_RETURN STTMT_LIST_MAIN
STTMT_IF					
STTMT_IF2	STTMT_IF2 → EPS	STTMT_IF2 → EPS			STTMT_IF2 → EPS
STTMT_IF3					
STTMT_WHILE					
STTMT_BREAK					
STTMT_BREAK2					
STTMT_CONTINUE	STTMT_CONTINUE → continue STTMT_CONTINUE2				
STTMT_CONTINUE2					
STTMT_FOR		STTMT_FOR → for(STTMT_FOR2 , STTMT_FOR2 , STTMT_FOR2){ STTMT_LIST }			
STTMT_FOR2			STTMT_FOR2 → EPS		
STTMT_LIST	STTMT_LIST → STTMT_CONTINUE STTMT_LIST				STTMT_LIST → EPS
STTMT_RETURN					STTMT_LIST → STTMT_RETURN STTMT_LIST
STTMT_RETURN2				STTMT_RETURN2 → ;	
STTMT_FC					
FC_PARAMETER_LIST					
FC_PARAMETER_LIST2					
STTMT_EXP					
EXP_FC_CALL					
PARAMETER_LIST					
PARAMETER_LIST2					
EXP_CONSTANT					
EXP_VARIABLE					
EXP_UNARY_OPERATOR					
EXP_BINARY_OPERATOR					
EPS	EPS → ε	EPS → ε	EPS → ε		EPS → ε

	elseif(	else{	while(	break	CONSTANT_INTEGER
program					
START					
STTMT_LIST_MAIN			STTMT_LIST_MAIN → STTMT_WHILE STTMT_LIST_MAIN		
STTMT_IF					
STTMT_IF2	STTMT_IF2 → elseif(STTMT_EXP){ STTMT_LIST } STTMT_IF2	STTMT_IF2 → STTMT_IF3	STTMT_IF2 → EPS	STTMT_IF2 → EPS	STTMT_IF2 → EPS
STTMT_IF3		STTMT_IF3 → else{ STTMT_LIST }			
STTMT_WHILE			STTMT_WHILE → while(STTMT_EXP){ STTMT_LIST }		
STTMT_BREAK				STTMT_BREAK → break STTMT_BREAK2	
STTMT_BREAK2					STTMT_BREAK2 → CONSTANT_INTEGER
STTMT_CONTINUE					
STTMT_CONTINUE2					STTMT_CONTINUE2 → CONSTANT_INTEGER
STTMT_FOR					
STTMT_FOR2					STTMT_FOR2 → STTMT_EXP
STTMT_LIST			STTMT_LIST → STTMT_WHILE STTMT_LIST	STTMT_LIST → STTMT_BREAK STTMT_LIST	STTMT_LIST → STTMT_EXP ; STTMT_LIST
STTMT_RETURN					
STTMT_RETURN2					
STTMT_FC					
FC_PARAMETER_LIST					
FC_PARAMETER_LIST2					
STTMT_EXP					STTMT_EXP → EXP_CONSTANTSTTMT_EXP → EXP_BINARY_OPERATOR
EXP_FC_CALL					
PARAMETER_LIST					PARAMETER_LIST → STTMT_EXP PARAMETER_LIST2
PARAMETER_LIST2					
EXP_CONSTANT					EXP_CONSTANT → CONSTANT_INTEGER
EXP_VARIABLE					
EXP_UNARY_OPERATOR					
EXP_BINARY_OPERATOR					EXP_BINARY_OPERATOR → STTMT_EXP BINARY_OPERATOR STTMT_EXP
EPS			EPS → ε	EPS → ε	EPS → ε

	)	,STTMT_EXP	CONSTANT_FLOAT	CONSTANT_STRING	CONSTANT_BOOL
program					
START					
STTMT_LIST_MAIN					
STTMT_IF					
STTMT_IF2			STTMT_IF2 → EPS	STTMT_IF2 → EPS	STTMT_IF2 → EPS
STTMT_IF3					
STTMT_WHILE					
STTMT_BREAK					
STTMT_BREAK2					
STTMT_CONTINUE					
STTMT_CONTINUE2					
STTMT_FOR					
STTMT_FOR2			STTMT_FOR2 → STTMT_EXP	STTMT_FOR2 → STTMT_EXP	STTMT_FOR2 → STTMT_EXP
STTMT_LIST			STTMT_LIST → STTMT_EXP ; STTMT_LIST	STTMT_LIST → STTMT_EXP ; STTMT_LIST	STTMT_LIST → STTMT_EXP ; STTMT_LIST
STTMT_RETURN					
STTMT_RETURN2					
STTMT_FC					
FC_PARAMETER_LIST					
FC_PARAMETER_LIST2					
STTMT_EXP			STTMT_EXP → EXP_CONSTANTSTTMT_EXP → EXP_BINARY_OPERATOR	STTMT_EXP → EXP_CONSTANTSTTMT_EXP → EXP_BINARY_OPERATOR	STTMT_EXP → EXP_CONSTANTSTTMT_EXP → EXP_BINARY_OPERATOR
EXP_FC_CALL					
PARAMETER_LIST			PARAMETER_LIST → STTMT_EXP PARAMETER_LIST2	PARAMETER_LIST → STTMT_EXP PARAMETER_LIST2	PARAMETER_LIST → STTMT_EXP PARAMETER_LIST2
PARAMETER_LIST2	PARAMETER_LIST2 → EPS	PARAMETER_LIST2 → , STTMT_EXP PARAMETER_LIST2			
EXP_CONSTANT			EXP_CONSTANT → CONSTANT_FLOAT	EXP_CONSTANT → CONSTANT_STRING	EXP_CONSTANT → CONSTANT_BOOL
EXP_VARIABLE					
EXP_UNARY_OPERATOR					
EXP_BINARY_OPERATOR			EXP_BINARY_OPERATOR → STTMT_EXP BINARY_OPERATOR STTMT_EXP	EXP_BINARY_OPERATOR → STTMT_EXP BINARY_OPERATOR STTMT_EXP	EXP_BINARY_OPERATOR → STTMT_EXP BINARY_OPERATOR STTMT_EXP
EPS	EPS → ε		EPS → ε	EPS → ε	EPS → ε

	FC	IDENTIFIER(	):	TERM_TYPE	,TERM_TYPE
program					
START					
STTMT_LIST_MAIN	STTMT_LIST_MAIN → STTMT_FC STTMT_LIST_MAIN				
STTMT_IF					
STTMT_IF2	STTMT_IF2 → EPS	STTMT_IF2 → EPS			
STTMT_IF3					
STTMT_WHILE					
STTMT_BREAK					
STTMT_BREAK2					
STTMT_CONTINUE					
STTMT_CONTINUE2					
STTMT_FOR					
STTMT_FOR2		STTMT_FOR2 → STTMT_EXP			
STTMT_LIST		STTMT_LIST → STTMT_EXP ; STTMT_LIST			
STTMT_RETURN					
STTMT_RETURN2					
STTMT_FC	STTMT_FC → FC IDENTIFIER(FC_PARAMETER_LIST): RETURN_TYPE { STTMT_LIST STTMT_RETURN }				
FC_PARAMETER_LIST				FC_PARAMETER_LIST → TERM_TYPE IDENTIFIER FC_PARAMETER_LIST2	
FC_PARAMETER_LIST2			FC_PARAMETER_LIST2 → EPS		FC_PARAMETER_LIST2 → ,TERM_TYPE IDENTIFIER FC_PARAMETER_LIST2
STTMT_EXP		STTMT_EXP → EXP_FC_CALLSTTMT_EXP → EXP_BINARY_OPERATOR			
EXP_FC_CALL		EXP_FC_CALL → IDENTIFIER(PARAMETER_LIST)			
PARAMETER_LIST		PARAMETER_LIST → STTMT_EXP PARAMETER_LIST2			
PARAMETER_LIST2					
EXP_CONSTANT					
EXP_VARIABLE					
EXP_UNARY_OPERATOR					
EXP_BINARY_OPERATOR		EXP_BINARY_OPERATOR → STTMT_EXP BINARY_OPERATOR STTMT_EXP			
EPS	EPS → ε	EPS → ε	EPS → ε		

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.

Precedenční tabulka:

	i	(	)	+-, ++--!x	*/	+-.	<>	!=	&&	||	??	+-.*/=	$
i			>	>	>	>	>	>	>	>	>	>	>
(	<	<	=	<	<	<	<	<	<	<	<	<	
)			>	>	>	>	>	>	>	>	>	>	>
+-, ++--!x	<	<	>	>	>	>	>	>	>	>	>	>	>
*/	<	<	>	<	>	>	>	>	>	>	>	>	>
+-.	<	<	>	<	<	>	>	>	>	>	>	>	>
<>	<	<	>	<	<	<	>	>	>	>	>	>	>
!=	<	<	>	<	<	<	<	>	>	>	>	>	>
&&	<	<	>	<	<	<	<	<	>	>	>	>	>
||	<	<	>	<	<	<	<	<	<	>	>	>	>
??	<	<	>	<	<	<	<	<	<	<	<	>	>
+-.*/=	<	<	>	<	<	<	<	<	<	<	<	<	>
$	<	<		<	<	<	<	<	<	<	<	<	

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 metodami 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.

Předpověď typů

Pro zlepšení rychlosti generovaného kódu se provádí určení co nejvíce přesných informací o typech za běhu, což následně omezí množství generovaných typových kontrol. Množina možných datových typů proměnné (včetně informace, jestli je proměnná definována) je uložena v mezipaměti typů proměnných. Je to z důvodu vysoké náročnosti určování typů. Mezipaměť typů je generovana několikrát za běhu, konkrétně při každém rozvinutí cyklů nebo nemožnosti pokračovat při optimalizacích a při finálním generování kódu.

Předpověď probíhá vytvořením tabulky typů proměnných a výsledných typů. Tabulka proměnných je naplněna nejdříve nedefinovanými typy pro veškeré proměnné, kromě případu, kdy se jedná o funkci, kde jsou parametry inicializovány na jejich počáteční typy. Následně se začne provádět postupně veškerý kód. Provádí se i obě větve podmíněných příkazů s rozdělenými tabulkami typů proměnných, které se po provedení spojí a následně se uvažuje že proměnná může mít kterýkoliv s typů, který do ni byl přiřazen po provedení jakékoliv podmíněné větve. Podobně se provádí i cykly, kdy se uvažují typy po provedení nula iterací, jedné iterace a dalších iterací až do chvíle, kdy už nedochází v tabulce k rozšíření o další typy. Pro podmíněné příkazy existuje optimalizace, že levá i pravá strana porovnání musí být stejná – to umožňuje redukci typů porovnávané proměnné nebo dokonce propagaci konstanty. Zároveň se při průchodu vytváří tabulka výsledných typů, kde různé výskyty jedné proměnné můžou nabývat

různých typů. Tato tabulka slouží také jako mezipaměť a jsou z ní dodávány informace o typech optimalizátoru a generátoru programu.

Rozšíření

FUNEXP

Toto rozšíření nevyžaduje z hlediska implementace oproti běžné implementaci nic navíc kromě nutnosti ukládat návratové hodnoty z dočasných rámců do lokálních rámců u vyhodnocení parametrů a zavolání od znova precedenční analýzy, když se narazí na argument funkce.

OPERATORS

Operátory +=, -=, *=, /=, .= nemají třídy prvky v syntaktickém stromu kvůli zmenšení množství kódu pro generování výsledného kódu a předpovídání typů. Místo toho je operátor $x OP= $y implementován jako $x = $x OP $y. Podomně jsou implementování i prefix operátory +$x a -$x jako $x = $x OP 1. A také prefix operátory ++$x a --$x jako $x = $x OP 1. Tento trik není možné využít pro postfixové operátory ++ a --, takže mají třídy v syntaktickém stromu, to stejné platí i pro operátor ??. Operátory = a operátor ?? se oproti ostatním operátorům liší v tom, že jsou pravě asociativní.

BOOLTHEN

Pro toto rozšíření jsme vytvořili nový typ operátoru (unární), kvůli operaci „!“. Nakonec jsme unární operátory využili i v rozšíření OPERATORS, kde jsme je dále rozčlenili na prefix a postfix operátory.

elseif v syntaktické analýze bereme jako if v else větvi nadřazeného if. Máme naimplementované zkratování v operátorech AND a OR pomocí skoků, pokud je výsledek operace AND false, nebo výsledek operace OR true. Toto rozšíření vyžadovalo vytvoření tokenů pro funkci for a pro funkce break a continue.

CYCLES

For k implementaci vyžadoval jen trochu více práce než while kvůli inkrementu a parametrům, které nejsou povinné. Funkce break a continue potřebovaly ke správné implementaci jejich parametrů také funkce pro pole řetězců, které si do sebe ukládá identifikační klíče jednotlivých cyklů, jak do nich vstupuje a vystupuje. Pokud se jim nepředá žádný parametr, jejich hodnota je 1 a vynoří se z posledního vnořeného cyklu.

STRNUM

U rozšíření strnum se převážně jednalo o generování kódu pro interpret a úpravu typových chyb. Jelikož se kód pro interpret velice podobá strojovému kódu, bylo to převážně o zamyšlení, jak daný převod vyřešit. Většina těchto převodů, po troše osvěžení strojového kódu, byly jednoduché. Nejsložitější převod byl z řetezce na desetinné číslo, protože byla nutnost podporovat nejenom desetinnou čárku, ale i tvar desetinného čísla s exponentem, který má mnoho různých možností, jak daný tvar může vypadat.