VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IFJ – protokol k projektu
Tým xjarol06, varianta TRP

Antonín Jarolím (xjarol06) - 28 bodů
Jakub Vlk (xvlkja07) - 26 bodů
Jan Brudný (xbrudn02) - 21 bodů
Jindřich Vodák (xvodak06) - 25 bodů

7. prosince 2022

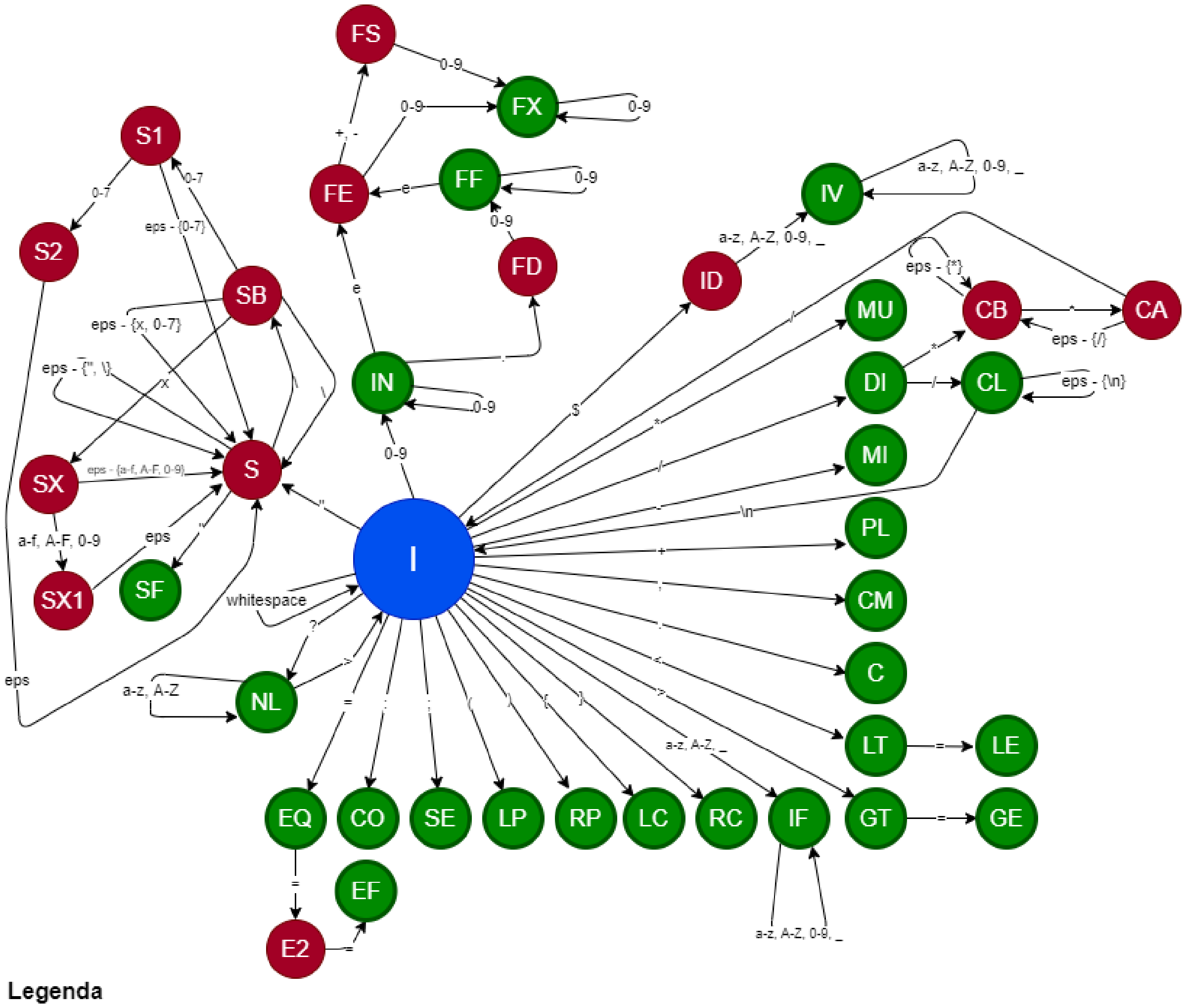
Obsah

Rozdělení práce mezi členy týmu

Antonín Jarolím - syntaktický analyzátor top-down, generace kódu, LL-tabulka, gramatika
Jakub Vlk - syntaktický analyzátor bottom-up, generace kódu, precedenční tabulka, gramatika
Jan Brudný - tabulka symbolů, generace kódu
Jindřich Vodák - lexikální analyzátor, psaní testů, dokumentace

Body byly v týmu rozděleny s přihlédnutím k množství odvedené práce, aktivnímu zapojování se do práce na projektu v průběhu semestru a vyvinuté snaze. Rozdělování probíhalo postupně v průběhu semestru - v případě, že tři členové týmu shledali, že čtvrtý odvádí velmi dobrou práci, body byly přerozděleny tak, aby čtvrtý člen získal bod navíc. V opačném případě byl jeden bod čtvrtému členovi týmu odebrán.

Diagram konečného automatu



Legenda

I = init_s
S = string_lit_s
SF = string_lit_f_s
IN = integer_lit_f_s
FD = float_lit_dot_s
FE = float_lit_e_s
FS = float_lit_sign_s
FX = float_lit_exp_f_s
FF = float_lit_f_s
IF = identifier_func_f_s
ID = identifier_var_dollar_s
IV = identifier_var_f_s
MU = multiplication_f_s
DI = division_f_s
PL = plus_f_s
MI = minus_f_s
C = concatenation_f_s
LT = lesser_than_f_s
LE = lesser_eq_f_s
GT = greater_than_f_s
GE = greater_eq_f_s
EF = eq_f_s

E2 = eq_2_s
N1 = not_eq_1_s
N2 = not_eq_2_s
NF = not_eq_f_s
SB = string_lit_backslash_s
SX = string_lit_backslash_x_s
SX1 = string_lit_backslash_x_1_s
S1 = string_lit_backslash_1_s
S2 = string_lit_backslash_2_s
LP = left_par_f_s
RP = right_par_f_s
CL = com_line_f_s
CB = com_block_s
CA = com_block_ast_s
LC = left_curly_f_s
RC = right_curly_f_s
EQ = equals_f_s
CO = colon_f_s
SE = semicolon_f_s
CM = comma_f_s
NL = null_f_s

LL-gramatika

ProgramBody ::= FceDefine ProgramBody
ProgramBody ::= Command ProgramBody
ProgramBody ::=”

Command ::= DeclareVariable
Command ::= Condition
Command ::= While
Command ::= Return
Command ::= Exp semicolon
Command ::= semicolon
Command ::= FceCall

FceDefine ::= FceHeader curlyBraceLeft FunctionBody curlyBraceRight
FceHeader ::= functionKey identifierFunc leftPar FunctionDeclareParams rightPar colon FuncReturnColonType
FunctionDeclareParams ::=”
FunctionDeclareParams ::= DeclareParam CommaOrEpsParams
CommaOrEpsParams ::=”
CommaOrEpsParams ::= comma DeclareParam CommaOrEpsParams
DeclareParam ::= DataType identifierVar

FuncReturnColonType ::= DataType
FuncReturnColonType ::= voidKey

FceCall ::= identifierFunc leftPar FirstFceParam rightPar
FirstFceParam ::=”
FirstFceParam ::= Statement CommaOrEpsParam
CommaOrEpsParam ::=”
CommaOrEpsParam ::= comma Statement CommaOrEpsParam

Statement ::= identifierVar
Statement ::= floatLiteral
Statement ::= stringLiteral
Statement ::= integerLiteral

Exp ::= Statement
Exp ::= nullKey
Exp ::= Exp minusOp Exp
Exp ::= Exp plusOp Exp
Exp ::= Exp divisionOp Exp
Exp ::= Exp multiplicationOp Exp
Exp ::= Exp concatenationOp Exp
Exp ::= leftPar Exp rightPar

DataType ::= stringNullKey
DataType ::= floatNullKey
DataType ::= intNullKey

DataType ::= stringKey
DataType ::= floatKey
DataType ::= intKey

DeclareVariable ::= identifierVar equals DefVarAss
DefVarAss ::= Exp semicolon
DefVarAss ::= FceCall semicolon

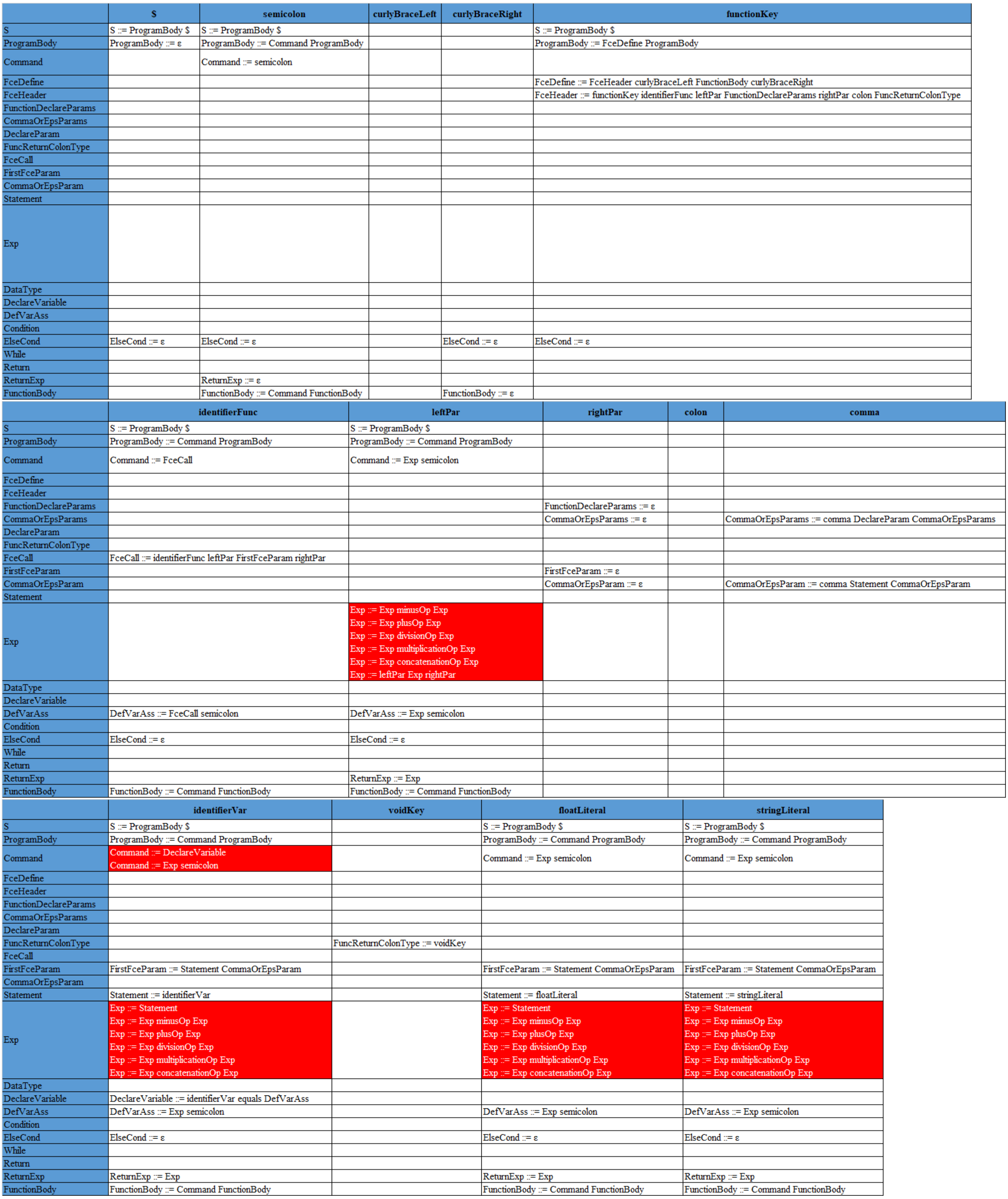
Condition ::= ifKey leftPar Exp rightPar curlyBraceLeft FunctionBody curlyBraceRight ElseCond
ElseCond ::= elseKey curlyBraceLeft FunctionBody curlyBraceRight
ElseCond ::="

While ::= whileKey leftPar Exp rightPar curlyBraceLeft FunctionBody curlyBraceRight

Return ::= returnKey ReturnExp semicolon
ReturnExp ::= Exp
ReturnExp ::="

FunctionBody ::= Command FunctionBody
FunctionBody ::="

LL-tabulka



	$	semicolon	curlyBraceLeft	curlyBraceRight	functionKey
S	S ::= ProgramBody $	S ::= ProgramBody $			S ::= ProgramBody $
ProgramBody	ProgramBody ::= ε	ProgramBody ::= Command ProgramBody			ProgramBody ::= FceDefine ProgramBody
Command		Command ::= semicolon			
FceDefine					FceDefine ::= FceHeader curlyBraceLeft FunctionBody curlyBraceRight
FceHeader					FceHeader ::= functionKey identifierFunc leftPar FunctionDeclareParams rightPar colon FuncReturnColonType
FunctionDeclareParams					
CommaOrEpsParams					
DeclareParam					
FuncReturnColonType					
FceCall					
FirstFceParam					
CommaOrEpsParam					
Statement					
Exp					
DataType					
DeclareVariable					
DefVarAss					
Condition					
ElseCond	ElseCond ::= ε	ElseCond ::= ε		ElseCond ::= ε	ElseCond ::= ε
While					
Return					
ReturnExp		ReturnExp ::= ε			
FunctionBody		FunctionBody ::= Command FunctionBody		FunctionBody ::= ε	

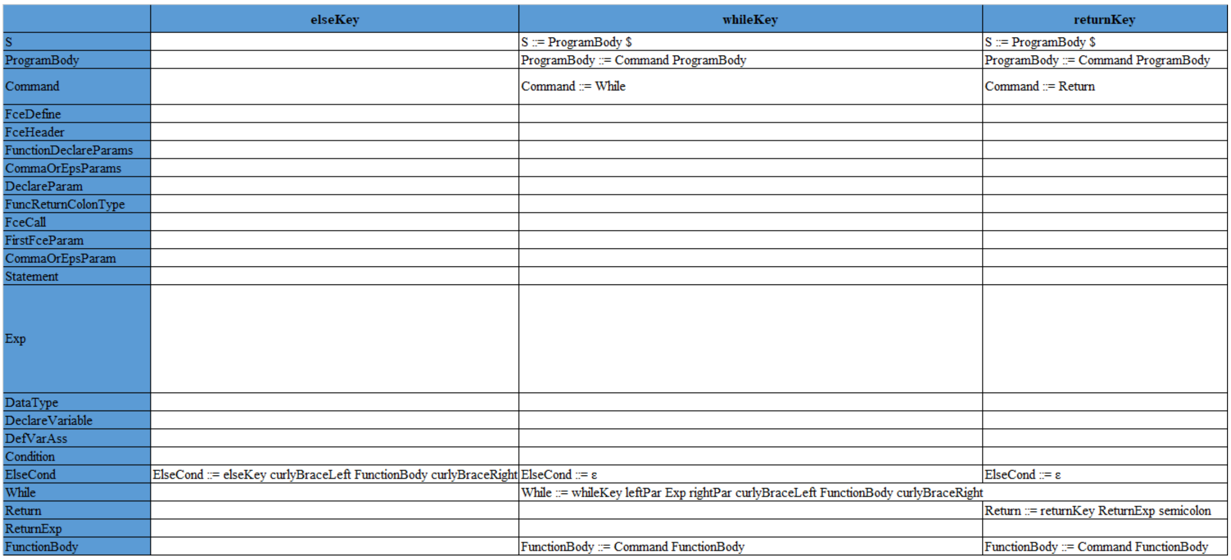
	identifierFunc	leftPar	rightPar	colon	comma
S	S ::= ProgramBody $	S ::= ProgramBody $			
ProgramBody	ProgramBody ::= Command ProgramBody	ProgramBody ::= Command ProgramBody			
Command	Command ::= FceCall	Command ::= Exp semicolon			
FceDefine					
FceHeader					
FunctionDeclareParams			FunctionDeclareParams ::= ε		
CommaOrEpsParams			CommaOrEpsParams ::= ε		CommaOrEpsParams ::= comma DeclareParam CommaOrEpsParams
DeclareParam					
FuncReturnColonType					
FceCall	FceCall ::= identifierFunc leftPar FirstFceParam rightPar				
FirstFceParam			FirstFceParam ::= ε		
CommaOrEpsParam			CommaOrEpsParam ::= ε		CommaOrEpsParam ::= comma Statement CommaOrEpsParam
Statement					
Exp		Exp ::= Exp minusOp Exp Exp ::= Exp plusOp Exp Exp ::= Exp divisionOp Exp Exp ::= Exp multiplicationOp Exp Exp ::= Exp concatenationOp Exp Exp ::= leftPar Exp rightPar			
DataType					
DeclareVariable					
DefVarAss	DefVarAss ::= FceCall semicolon	DefVarAss ::= Exp semicolon			
Condition					
ElseCond	ElseCond ::= ε	ElseCond ::= ε			
While					
Return					
ReturnExp		ReturnExp ::= Exp			
FunctionBody	FunctionBody ::= Command FunctionBody	FunctionBody ::= Command FunctionBody			

	identifierVar	voidKey	floatLiteral	stringLiteral
S	S ::= ProgramBody $		S ::= ProgramBody $	S ::= ProgramBody $
ProgramBody	ProgramBody ::= Command ProgramBody		ProgramBody ::= Command ProgramBody	ProgramBody ::= Command ProgramBody
Command	Command ::= DeclareVariable Command ::= Exp semicolon		Command ::= Exp semicolon	Command ::= Exp semicolon
FceDefine				
FceHeader				
FunctionDeclareParams				
CommaOrEpsParams				
DeclareParam				
FuncReturnColonType		FuncReturnColonType ::= voidKey		
FceCall				
FirstFceParam	FirstFceParam ::= Statement CommaOrEpsParam		FirstFceParam ::= Statement CommaOrEpsParam	FirstFceParam ::= Statement CommaOrEpsParam
CommaOrEpsParam				
Statement	Statement ::= identifierVar		Statement ::= floatLiteral	Statement ::= stringLiteral
Exp	Exp ::= Statement Exp ::= Exp minusOp Exp Exp ::= Exp plusOp Exp Exp ::= Exp divisionOp Exp Exp ::= Exp multiplicationOp Exp Exp ::= Exp concatenationOp Exp		Exp ::= Statement Exp ::= Exp minusOp Exp Exp ::= Exp plusOp Exp Exp ::= Exp divisionOp Exp Exp ::= Exp multiplicationOp Exp Exp ::= Exp concatenationOp Exp	Exp ::= Statement Exp ::= Exp minusOp Exp Exp ::= Exp plusOp Exp Exp ::= Exp divisionOp Exp Exp ::= Exp multiplicationOp Exp Exp ::= Exp concatenationOp Exp
DataType				
DeclareVariable	DeclareVariable ::= identifierVar equals DefVarAss			
DefVarAss	DefVarAss ::= Exp semicolon		DefVarAss ::= Exp semicolon	DefVarAss ::= Exp semicolon
Condition				
ElseCond	ElseCond ::= ε		ElseCond ::= ε	ElseCond ::= ε
While				
Return				
ReturnExp	ReturnExp ::= Exp		ReturnExp ::= Exp	ReturnExp ::= Exp
FunctionBody	FunctionBody ::= Command FunctionBody		FunctionBody ::= Command FunctionBody	FunctionBody ::= Command FunctionBody

	integerLiteral	nullKey	minusOp	plusOp	divisionOp	multiplicationOp	concatenationOp
S	S ::= ProgramBody $	S ::= ProgramBody $					
ProgramBody	ProgramBody ::= Command ProgramBody	ProgramBody ::= Command ProgramBody					
Command	Command ::= Exp semicolon	Command ::= Exp semicolon					
FceDefine							
FceHeader							
FunctionDeclareParams							
CommaOrEpsParams							
DeclareParam							
FuncReturnColonType							
FceCall							
FirstFceParam	FirstFceParam ::= Statement CommaOrEpsParam						
CommaOrEpsParam							
Statement	Statement ::= integerLiteral						
Exp	Exp ::= Statement Exp ::= Exp minusOp Exp Exp ::= Exp plusOp Exp Exp ::= Exp divisionOp Exp Exp ::= Exp multiplicationOp Exp Exp ::= Exp concatenationOp Exp	Exp ::= nullKey Exp ::= Exp minusOp Exp Exp ::= Exp plusOp Exp Exp ::= Exp divisionOp Exp Exp ::= Exp multiplicationOp Exp Exp ::= Exp concatenationOp Exp					
DataType							
DeclareVariable							
DefVarAss	DefVarAss ::= Exp semicolon	DefVarAss ::= Exp semicolon					
Condition							
ElseCond	ElseCond ::= ε	ElseCond ::= ε					
While							
Return							
ReturnExp	ReturnExp ::= Exp	ReturnExp ::= Exp					
FunctionBody	FunctionBody ::= Command FunctionBody	FunctionBody ::= Command FunctionBody					

	stringNullKey	floatNullKey	intNullKey	stringKey
S				
ProgramBody				
Command				
FceDefine				
FceHeader				
FunctionDeclareParams	FunctionDeclareParams ::= DeclareParam CommaOrEpsParam	FunctionDeclareParams ::= DeclareParam CommaOrEpsParam	FunctionDeclareParams ::= DeclareParam CommaOrEpsParams	FunctionDeclareParams ::= DeclareParam CommaOrEpsParar
CommaOrEpsParams				
DeclareParam	DeclareParam ::= DataType identifierVar	DeclareParam ::= DataType identifierVar	DeclareParam ::= DataType identifierVar	DeclareParam ::= DataType identifierVar
FuncReturnColonType	FuncReturnColonType ::= DataType	FuncReturnColonType ::= DataType	FuncReturnColonType ::= DataType	FuncReturnColonType ::= DataType
FceCall				
FirstFceParam				
CommaOrEpsParam				
Statement				
Exp				
DataType	DataType ::= stringNullKey	DataType ::= floatNullKey	DataType ::= intNullKey	DataType ::= stringKey
DeclareVariable				
DefVarAss				
Condition				
ElseCond				
While				
Return				
ReturnExp				
FunctionBody				

	floatKey	intKey	equals	ifKey
S				S ::= ProgramBody $
ProgramBody				ProgramBody ::= Command ProgramBody
Command				Command ::= Condition
FceDefine				
FceHeader				
FunctionDeclareParams	FunctionDeclareParams ::= DeclareParam CommaOrEpsParams	FunctionDeclareParams ::= DeclareParam CommaOrEpsParams		
CommaOrEpsParams				
DeclareParam	DeclareParam ::= DataType identifierVar	DeclareParam ::= DataType identifierVar		
FuncReturnColonType	FuncReturnColonType ::= DataType	FuncReturnColonType ::= DataType		
FceCall				
FirstFceParam				
CommaOrEpsParam				
Statement				
Exp				
DataType	DataType ::= floatKey	DataType ::= intKey		
DeclareVariable				
DefVarAss				
Condition				Condition ::= ifKey leftPar Exp rightPar curlyBraceLeft FunctionBody curlyBraceRight ElseCon
ElseCond				ElseCond ::= ε
While				
Return				
ReturnExp				
FunctionBody				FunctionBody ::= Command FunctionBody



	elseKey	whileKey	returnKey
S		S ::= ProgramBody $	S ::= ProgramBody $
ProgramBody		ProgramBody ::= Command ProgramBody	ProgramBody ::= Command ProgramBody
Command		Command ::= While	Command ::= Return
FceDefine			
FceHeader			
FunctionDeclareParams			
CommaOrEpsParams			
DeclareParam			
FuncReturnColonType			
FceCall			
FirstFceParam			
CommaOrEpsParam			
Statement			
Exp			
DataType			
DeclareVariable			
DefVarAss			
Condition			
ElseCond	ElseCond ::= elseKey curlyBraceLeft FunctionBody curlyBraceRight	ElseCond ::= ε	ElseCond ::= ε
While		While ::= whileKey leftPar Exp rightPar curlyBraceLeft FunctionBody curlyBraceRight	
Return			Return ::= returnKey ReturnExp semicolon
ReturnExp			
FunctionBody		FunctionBody ::= Command FunctionBody	FunctionBody ::= Command FunctionBody

Precedenční tabulka

Členění implementačního řešení

Lexikální analyzátor sestává ze dvou souborů - hlavičkového souboru *lex.h*, ve kterém jsou definovány všechny důležité struktury a hlavičky funkcí používaných dále v programu, a zdrojového souboru *lex.c*, jehož kód vykonává samotnou lexikální analýzu. Lexikální analyzátor generuje tokeny ve formě struktury definované v hlavičkovém souboru, která obsahuje typ lexému (využit výčtový typ `lexType` definovaný rovněž v hlavičkovém souboru), informace o pozici v textu (pro ladění a chybové výpisy) a vnořenou datovou strukturu `data_t`. Tato vnořená struktura je typu union a jejím účelem je uchovávat datový obsah tokenu (pokud jej tedy token má) - například v případě celočíselného literálu je do proměnné `valueInteger` datového typu `int` uvnitř struktury uložena hodnota daného literálu. V případě řetězcového literálu využívá struktura `data_t` pomocné knihovny *dynstring.c*, která obsahuje speciální datový typ `dynStr_t` a funkce výrazně usnadňující práci s dynamickými řetězci. To s sebou přináší značné výhody nejen skrze jednodušší ladění, ale také například při práci s escape sekvencemi.

Pro vygenerování tokenu ze zdrojových dat je nutno zavolat hlavní a největší funkci celého lexikálního analyzátoru `getToken()`. Tato funkce načítá vstupní data znak po znaku pomocí jednoduché funkce `getNextChar()` a skrze vnitřní konečný automat nalezne pro lexém odpovídající stav. Pokud je detekován identifikátor či literál, je při čtení dat zároveň aktivován také buffer, který všechny přečtené znaky ukládá, a pokud jde o řetězcový literál obsahující jednu nebo více escape sekvencí, je zároveň aktivován druhý buffer uchovávající danou escape sekvenci. Tato escape sekvence je okamžitě zpracována dle svého typu, konvertovaný znak je uložen do datového bufferu a sekvenční buffer je vyčištěn. Při dosažení separátoru (dle lexému může jít o bílý znak, speciální znak či jen začátek dalšího lexému) je konečný stav a buffer předán druhé části analyzátoru, která na základě stavu rozhodne o typu tokenu a uloží do něj jeho data, a pokud je to nutné (v případě přerušení začátkem nového lexému), na začátek vstupního proudu je pomocí funkce `ungetNextChar()` vrácen poslední načtený znak.

Lexikální analyzátor podporuje několik zajímavých funkcí, které slouží buď ke zjednodušení procesu ladění nebo k usnadnění práce při dalších fázích překladu. Jako první příklad uveďme funkci `printTokenData()`, jejímž vstupem je token a výstupem informace o typu tokenu, jeho obsahu a jeho pozici ve zdrojovém textu. Jde o velmi jednoduchou funkci, která se ale ukázala jako zcela nepostradatelná i v posledních fázích práce na projektu. Pro účely testování vznikl jednoduchý program obsahující tuto funkci, který byl schopen načíst data ze zdrojového souboru a cyklicky vypsat informace o všech zpracovaných tokenech. Tento program sloužil jako důležitá pomoc při práci na lexikálním analyzátoru a pomohl vyřešit spoustu obtížně zachytitelných chyb.

Druhým příkladem je funkce `ungetToken()`, která funguje analogicky s `ungetNextChar()` - na výstup lexikálního analyzátoru vrátí poslední zpracovaný token. Samotný lexikální analyzátor obsahuje buffer, v němž je vždy uložen poslední zpracovaný token, a přepínač uchovávající informaci o tom, zda se má při volání funkce `getToken()` zpracovat nový lexém či právě vrátit tento poslední token. Jde o funkci důležitou pro syntaktický analyzátor, který se tak nemusí starat o uchovávání tokenů a stačí mu funkci jednoduše zavolat.

Závěr