

IFJ – protokol k projektu Tým xjarol06, varianta TRP

Antonín Jarolím (xjarol06) - 28 bodů

Jakub Vlk (xvlkja
07) - 26 bodů Jan Brudný (xbrudn
02) - 21 bodů Jindřich Vodák (xvodak
06) - 25 bodů

7. prosince 2022

Obsah

1	Rozdělení práce mezi členy týmu	2
2	Diagram konečného automatu	3
3	LL-gramatika	4
4	LL-tabulka	6
5	Precedenční tabulka	9
6	Členění implementačního řešení	10
7	Závěr	11

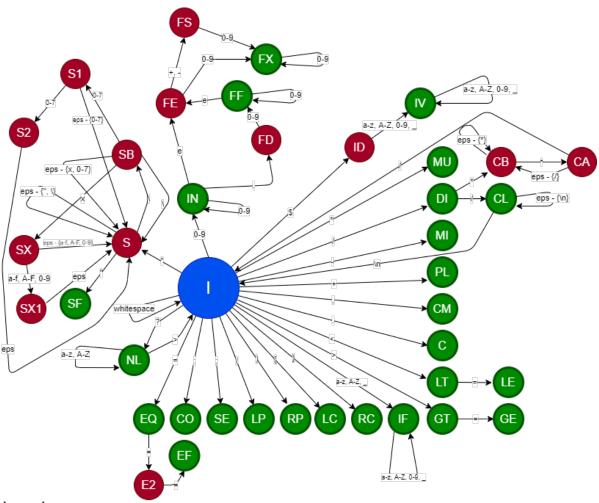
Rozdělení práce mezi členy týmu

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
                                E2 = eq_2s
SF = string_lit_f_s
                                N1 = not_eq_1_s
IN = integer_lit_f_s
                                N2 = not_eq_2_s
FD = float_lit_dot_s
                                NF = not_eq_f_s
SB = string_lit_backslash_s
FE = float_lit_e_s
FS = float_lit_sign_s
                                SX = string_lit_backslash_x_s
FX = float_lit_exp_f_s
                                SX1 = string_lit_backslash_x_1_s
FF = float_lit_f_s
                                S1 = string_lit_backslash_1_s
IF = identifier_func_f_s
                                S2 = string_lit_backslash_2_s
ID = identifier_var_dollar_s
                                LP = left_par_f_s
IV = identifier_var_f_s
                                RP = right_par_f_s
MU = multiplication_f_s
                                CL = com_line_f_s
                                CB = com_block_s
DI = division_f_s
PL = plus f s
                                CA = com_block_ast_s
MI = minus f s
                                LC = left_curly_f_s
C = concatenation_f_s
                                RC = right_curly_f_s
LT = lesser_than_f_s
                                EQ = equals_f_s
LE = lesser_eq_f_s
                                CO = colon_f_s
GT = greater_than_f_s
                                SE = semicolon f s
GE = greater_eq_f_s
                                CM = comma f s
EF = eq_f_s
                                NL = null f s
```

LL-gramatika

DataType ::= intNullKey

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

DeclareVariable ::= identifierVar equals DefVarAss

DefVarAss ::= Exp semicolon DefVarAss ::= FceCall semicolon

Condition ::= ifKey leftPar Exp rightPar curlyBraceLeft FunctionBody curlyBraceRight El-

seCond

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

ElseCond ::= ε

S										
5	s	semicolon		curlyBraceLeft	curlyBr	aceRight	n n 1 1		functionKey	
ProgramBody	S ::= ProgramBody \$ ProgramBody ::= ε	S ::= ProgramBody \$ ProgramBody ::= Command ProgramBody	amBodu.				S ::= ProgramBody \$ ProgramBody ::= FceDefine Progr	amBody.		
	110gramDody e		amBody			Togrambody TeeDeline Trogi	aniiDouy			
ommand	Command ::= semicolon									
ceDefine						FceDefine ::= FceHeader curlyBraceLeft FunctionBody curlyBraceRight				
ceHeader					F	FceHeader ::= functionKey identifierFunc leftPar FunctionDeclareParams rightPar colon Fun			ncReturnColonTyp	
unctionDeclareParams CommaOrEpsParams										
DeclareParam										
uncReturnColonType										
ceCall										
irstFceParam										
ommaOrEpsParam tatement										
internent	enicii									
Exp										
)ataType										
eclareVariable										
efVarAss ondition										
lseCond	ElseCond ::= ε	ElseCond ::= ε			ElseCond ::	= ε F	ElseCond ::= ε			
Vhile										
eturn								-		
LeturnExp		ReturnExp ::= 8	onDe J		Enmation P	du c				
unctionBody		FunctionBody ::= Command Functi	oupody		FunctionBo	nuy– ε				
		identifierFunc			ftPar		rightPar	colon	comma	
D 1	S ::= ProgramBody \$	in n'		ProgramBody \$	1 P	D 1				
rogramBody	ProgramBody ::= Com	nand ProgramBody		ramBody ::= Comm		mBody				
command	Command ::= FceCall		Com	mand ::= Exp semic	colon					
ceDefine										
ceHeader										
unctionDeclareParams							FunctionDeclareParams ::= ε			
CommaOrEpsParams DeclareParam							CommaOrEpsParams ::= €		CommaOrEpsParams ::= comma Declare	Param CommaOrE
uncReturnColonType										
ceCall	FceCall ::= identifierFu	nc leftPar FirstFceParam rightPar								
irstFceParam							FirstFceParam ::= ε			
CommaOrEpsParam				CommaOrEpsParam ::= ε			CommaOrEpsParam ::= comma Statemen	t CommaOrEpsPar		
Statement			Eva minus On Eva							
			Exp :	= Exp minusOp Ex	KD.					
				::= Exp minusOp Ex ::= Exp plusOp Exp						
Екр			Exp :	::= Exp plusOp Exp ::= Exp divisionOp E	Exp					
Схр			Exp : Exp : Exp :	::= Exp plusOp Exp ::= Exp divisionOp E ::= Exp multiplication	Exp nOp Exp					
Exp			Exp : Exp : Exp :	::= Exp plusOp Exp ::= Exp divisionOp E ::= Exp multiplication ::= Exp concatenation	Exp nOp Exp onOp Exp					
			Exp : Exp : Exp :	::= Exp plusOp Exp ::= Exp divisionOp E ::= Exp multiplication	Exp nOp Exp onOp Exp					
DataType			Exp : Exp : Exp :	::= Exp plusOp Exp ::= Exp divisionOp E ::= Exp multiplication ::= Exp concatenation	Exp nOp Exp onOp Exp					
DataType DeclareVariable DefVarAss	DefVarAss ::= FceCall	semicolon	Exp: Exp: Exp: Exp:	::= Exp plusOp Exp ::= Exp divisionOp E ::= Exp multiplication ::= Exp concatenation	Exp nOp Exp onOp Exp tPar					
DataType DeclareVariable DefVarAss Condition		semicolon	Exp: Exp: Exp: Exp: Def\	:= Exp plusOp Exp := Exp divisionOp E := Exp multiplication := Exp concatenation := leftPar Exp right	Exp nOp Exp onOp Exp tPar					
DataType DeclareVariable DefVarAss Condition	DefVarAss ::= FceCall ElseCond ::= ε	semicolon	Exp: Exp: Exp: Exp: Def\	== Exp plusOp Exp == Exp divisionOp E == Exp multiplication == Exp concatenation == leftPar Exp right	Exp nOp Exp onOp Exp tPar					
DataType DeclareVariable DefVarAss Condition JseCond Vhile		semicolon	Exp: Exp: Exp: Exp: Def\	:= Exp plusOp Exp := Exp divisionOp E := Exp multiplication := Exp concatenation := leftPar Exp right	Exp nOp Exp onOp Exp tPar					
DataType DeclareVariable DecVarAss Condition UseCond Vhile Leturn LeturnExp	ElseCond ::= 8		Exp =	== Exp plusOp Exp == Exp divisionOp E == Exp multiplication == Exp concatenation == leftPar Exp right == VarAss :== Exp sem Cond :== e	Exp nOp Exp onOp Exp tPar sicolon					
DataType DeclareVariable DecVarAss Condition UseCond Vhile Leturn LeturnExp			Exp =	:= Exp plusOp Exp := Exp divisionOp E = Exp multiplication := Exp concatenation := leftPar Exp right VarAss ::= Exp sem Cond ::= 8	Exp nOp Exp onOp Exp tPar sicolon	nBody				
DataType Declare Variable DefVarAss Condition IseCond Vhile Letturn	ElseCond ::= ε FunctionBody ::= Comr		Exp =	== Exp plusOp Exp == Exp divisionOp E == Exp multiplication == Exp concatenation == leftPar Exp right == VarAss :== Exp sem Cond :== e	Exp nOp Exp onOp Exp tPar sicolon	bnBody	floatLiteral		stringLiteral	
DataType Declare Variable DetVar Ass Ondition Lister Cond While Leturn Leturn Exp unctionBody	ElseCond ::= 8 FunctionBody ::= Comm is S ::= ProgramBody \$	mand FunctionBody	Exp =	== Exp plusOp Exp == Exp divisionOp Exp == Exp multiplicatio == Exp concatenatio == leftPar Exp right VarAss := Exp sem Cond := e rnExp := Exp tionBody := Comm	Exp mOp Exp onOp Exp Par sicolon	S ::= ProgramI	Body \$		gramBody \$	
DataType Declare Variable DetVar Ass Ondition Lister Cond While Leturn Leturn Exp unctionBody	ElseCond ::= \$ FunctionBody ::= Comm is S ::= ProgramBody \$ ProgramBody := Comm	mand FunctionBody dentifierVar nand ProgramBody	Exp =	== Exp plusOp Exp == Exp divisionOp Exp == Exp multiplicatio == Exp concatenatio == leftPar Exp right VarAss := Exp sem Cond := e rnExp := Exp tionBody := Comm	Exp mOp Exp onOp Exp Par sicolon	S ::= ProgramI				
DataType Declare Variable DelVar Ass Ondition Lister Cond While Leturn LeturnExp unctionBody ProgramBody	ElseCond ::= s FunctionBody ::= Comm S ::= ProgramBody \$ ProgramBody := Command := Declare\ Command := Declare\	mand FunctionBody dentifierVar mand ProgramBody 'ariable	Exp =	== Exp plusOp Exp == Exp divisionOp Exp == Exp multiplicatio == Exp concatenatio == leftPar Exp right VarAss := Exp sem Cond := e rnExp := Exp tionBody := Comm	Exp mOp Exp mOp Exp onOp Exp tPar sicolon	S ::= ProgramI ProgramBody	Body \$	Programl	gramBody \$	
DataType Declare Variable Del Var Ass Ondition IseCond Vale Leturn LeturnExp LeturnExp LeturnBody TogramBody Command	ElseCond ::= \$ FunctionBody ::= Comm is S ::= ProgramBody \$ ProgramBody := Comm	mand FunctionBody dentifierVar mand ProgramBody 'ariable	Exp =	== Exp plusOp Exp == Exp divisionOp Exp == Exp multiplicatio == Exp concatenatio == leftPar Exp right VarAss := Exp sem Cond := e rnExp := Exp tionBody := Comm	Exp mOp Exp mOp Exp onOp Exp tPar sicolon	S ::= ProgramI ProgramBody	Body \$::= Command ProgramBody	Programl	gramBody \$ Body ::= Command ProgramBody	
bataType beclareVariable berVarAss condition lseCond whale teturn teturnExp unctionBody rogramBody command ceDefine	ElseCond ::= s FunctionBody ::= Comm S ::= ProgramBody \$ ProgramBody := Command := Declare\ Command := Declare\	mand FunctionBody dentifierVar mand ProgramBody 'ariable	Exp =	== Exp plusOp Exp == Exp divisionOp Exp == Exp multiplicatio == Exp concatenatio == leftPar Exp right VarAss := Exp sem Cond := e rnExp := Exp tionBody := Comm	Exp mOp Exp mOp Exp onOp Exp tPar sicolon	S ::= ProgramI ProgramBody	Body \$::= Command ProgramBody	Programl	gramBody \$ Body ::= Command ProgramBody	
DataType Declare Variable Def Var Ass Ondition Use Cond While Leturn Leturn Eleturn Eleturn Eleturn Program Body Command CceDefine CceHeader unction Declare Params	ElseCond ::= s FunctionBody ::= Comm S ::= ProgramBody \$ ProgramBody := Command := Declare\ Command := Declare\	mand FunctionBody dentifierVar mand ProgramBody 'ariable	Exp =	== Exp plusOp Exp == Exp divisionOp Exp == Exp multiplicatio == Exp concatenatio == leftPar Exp right VarAss := Exp sem Cond := e rnExp := Exp tionBody := Comm	Exp mOp Exp mOp Exp onOp Exp tPar sicolon	S ::= ProgramI ProgramBody	Body \$::= Command ProgramBody	Programl	gramBody \$ Body ::= Command ProgramBody	
ataType eclareVariable erVarAss ondition lseCond fale etturn etturnExp unctionBody rogramBody ommand ceDefine ceHeader ametionDeclareParams ommaOrEpSParams	ElseCond ::= s FunctionBody ::= Comm S ::= ProgramBody \$ ProgramBody := Command := Declare\ Command := Declare\	mand FunctionBody dentifierVar mand ProgramBody 'ariable	Exp =	== Exp plusOp Exp == Exp divisionOp Exp == Exp multiplicatio == Exp concatenatio == leftPar Exp right VarAss := Exp sem Cond := e rnExp := Exp tionBody := Comm	Exp mOp Exp mOp Exp onOp Exp tPar sicolon	S ::= ProgramI ProgramBody	Body \$::= Command ProgramBody	Programl	gramBody \$ Body ::= Command ProgramBody	
ataType ceclareVariable efVarAss ondition secCond hale tturn tturnExp metionBody ogramBody ogramBody ommand teDefine efFeader metionDeclareParams celareParams	ElseCond ::= s FunctionBody ::= Comm S ::= ProgramBody \$ ProgramBody := Command := Declare\ Command := Declare\	nand FunctionBody dentifierVar mand ProgramBody 'ariable colon	Exp	== Exp phasOp Exp == Exp divisionOp E == Exp multiplicatio == Exp concatenatic == leftPar Exp right VarAss := Exp sem Cond := & unExp := Exp tionBody := Comm voidKey	Exp mOp Exp onOp Exp Par iicolon	S ::= ProgramI ProgramBody	Body \$::= Command ProgramBody	Programl	gramBody \$ Body ::= Command ProgramBody	
ataType eclareVariable efVarAss ondition seCond 'hale eturn eturnExp anctionBody rogramBody ommand ecDefine elFeader anctionDeclareParams ommaOrEpsParams eclareParam ancReturnColonType	ElseCond ::= s FunctionBody ::= Comm S ::= ProgramBody \$ ProgramBody := Command := Declare\ Command := Declare\	nand FunctionBody dentifierVar mand ProgramBody 'ariable colon	Exp	== Exp plusOp Exp == Exp divisionOp Exp == Exp multiplicatio == Exp concatenatio == leftPar Exp right VarAss := Exp sem Cond := e rnExp := Exp tionBody := Comm	Exp mOp Exp onOp Exp Par iicolon	S ::= ProgramI ProgramBody	Body \$::= Command ProgramBody	Programl	gramBody \$ Body ::= Command ProgramBody	
ataType eclareVariable efVarAss ondition secOnd fale etturn etturnExp anctionBody rogramBody ommand ceDefine ceHeader ametionDeclareParams ommaOrEpsParams eclareParam mcReturnColonType ecCall	ElseCond := 8 FunctionBody := Comm is := ProgramBody \$ ProgramBody := Comm Command := Declare\ Command := Exp semi	nand FunctionBody dentifierVar mand ProgramBody 'ariable colon	Exp	== Exp phasOp Exp == Exp divisionOp E == Exp multiplicatio == Exp concatenatic == leftPar Exp right VarAss := Exp sem Cond := & unExp := Exp tionBody := Comm voidKey	Exp mOp Exp onOp Exp Par sicolon	S ::= ProgramB ProgramBody Command ::= 1	Body \$ == Command ProgramBody Exp semicolon	Programl Comman	gramBody \$ Body ::= Command ProgramBody	
ataType eclareVariable efVarAss ondition secCend fale eturn eturnExp metionBody rogramBody oromand ecDefine eclifeader metionDeclareParams eclareParam meReturnColorType eccall stsfeeParam	ElseCond ::= \$\varepsilon\$ FunctionBody ::= Comm is ::= ProgramBody S ProgramBody ::= Command ::= Declare\ Command ::= Exp semi FirstFceParam ::= State	mand FunctionBody dentifierVar mand ProgramBody /ariable colon ement CommaOrEpsParam	Exp	== Exp phasOp Exp == Exp divisionOp E == Exp multiplicatio == Exp concatenatic == leftPar Exp right VarAss := Exp sem Cond := & unExp := Exp tionBody := Comm voidKey	Exp mOp Exp mOp Exp mOp Exp licolon	S ::= ProgramBody ProgramBody Command ::= 1	Body \$:= Command ProgramBody Exp semicolon a ::= Statement CommaOrEpsPara	Programl Comman m FirstFceF	gramBody \$ Body := Command ProgramBody d := Exp semicolon Param ::= Statement CommaOrEpsParam	
bataType seclareVariable seclareVariable setVarAss ondition lseCond shile seturn seturnExp unctionBody rogramBody ommand ceDefine ceHeader ceHeader ceHeader sommanOrEpsParams ommanOrTpsParam ommaOrTpsParam	ElseCond := 8 FunctionBody := Comm is := ProgramBody := Comm Command := Declare Command := Exp semi FirstFceParam ::= State Statement ::= identifier	mand FunctionBody dentifierVar mand ProgramBody /ariable colon ement CommaOrEpsParam	Exp	== Exp phasOp Exp == Exp divisionOp E == Exp multiplicatio == Exp concatenatic == leftPar Exp right VarAss := Exp sem Cond := & unExp := Exp tionBody := Comm voidKey	Exp mOp Exp onOp Exp iicolon	S ::= ProgramB ProgramBody Command ::= 1 FirstFceParam	Body \$:= Command ProgramBody Exp semicolon i:= Statement CommaOrEpsPara BoatLiteral	Programl Comman m FirstFceF	gramBody \$ Body := Command ProgramBody d := Exp semicolon Param := Statement CommaOrEpsParam t := stringLiteral	
DataType DeclareVariable DeclareVariable DefVarAss Condition DescCond While Letturn LetturnExp UnctionBody ProgramBody Command CeDefine CeHeader UnctionDeclareParams CommaOrEpsParams DeclareParam UnceReturnColonType CeCCall IrristTeCParam CommaOrEpsParam	ElseCond ::= s FunctionBody ::= Comm is S ::= ProgramBody S ProgramBody ::= Comm Command := Declare V Command := Exp sem FirstFceParam ::= State Statement ::= identifier Exp := Statement	mand FunctionBody dentifierVar mand ProgramBody ariable colon ement CommaOrEpsParam	Exp	== Exp phasOp Exp == Exp divisionOp E == Exp multiplicatio == Exp concatenatic == leftPar Exp right VarAss := Exp sem Cond := & unExp := Exp tionBody := Comm voidKey	Exp mOp Exp mOp Exp mOp Exp sicolon and Function	S := ProgramBody ProgramBody Command := 1 FirstFceParam Statement := 1 Exp := Statem	Body \$:= Command ProgramBody Exp semicolon ::= Statement CommaOrEpsPara doatLiteral ent	Programl Comman m FirstFceF Statemen Exp := S	gramBody \$ 30dy := Command ProgramBody d := Exp semicolon Param := Statement CommaOrEpsParam t := stringLiteral atement	
DataType DeclareVariable DeclareVariable DefVarAss Omdition DiseCond While Letturn LetturnExp UnctionBody TogramBody TogramB	ElseCond ::= \$\varepsilon\$ FunctionBody ::= Comm S ::= ProgramBody \$\varepsilon\$ ProgramBody ::= Comm Command ::= Exp sent FirstFceParam ::= State Statement ::= identifer Exp := Statement Exp := Statement	mand FunctionBody dentifierVar mand ProgramBody /ariable colon ement CommaOrEpsParam Var	Exp	== Exp phasOp Exp == Exp divisionOp E == Exp multiplicatio == Exp concatenatic == leftPar Exp right VarAss := Exp sem Cond := & unExp := Exp tionBody := Comm voidKey	Exp nOp Exp nOp Exp nOp Exp licolon	S := ProgramBody ProgramBody Command := 1 FirstFceParam Statement := f Exp := Statem Exp := Exp miles	Body \$:= Command ProgramBody Exp semicolon := Statement CommaOrEpsPara := Statement CommaOrEpsPara := statement CommaOrEpsPara	Programl Comman FirstFceF Statemen Exp := S Exp := E	gramBody \$ Sody := Command ProgramBody d := Exp semicolon Param ::= Statement CommaOrEpsParam t ::= stringLiteral tatement xp minusOp Exp	
DataType DeclareVariable DeclareVariable DeclareVariable DeclareVariable DefVarAss Ondition Declared DeclareVariable DeclareData DeclareDa	ElseCond ::= 8 FunctionBody ::= Comm S ::= ProgramBody \$ ProgramBody := Comm Command := Declare Command := Exp semi FirstFceParam ::= State Statement ::= identifier Exp ::= Exp minsOp E Exp ::= Exp phinsOp Exp ::= Exp phisOp Exp	mand FunctionBody dentifier Var mand ProgramBody variable coolen ement CommaOrEpsParam Var	Exp	== Exp phasOp Exp == Exp divisionOp E == Exp multiplicatio == Exp concatenatic == leftPar Exp right VarAss := Exp sem Cond := & unExp := Exp tionBody := Comm voidKey	Exp mOp Exp onOp Exp ilPar iicolon	S := ProgramBody ProgramBody Command := I FirstFceParam Statement := f Exp := Statem Exp := Exp mi Exp := Exp mi Exp := Exp mi	Body \$:= Command ProgramBody Exp semicolon 1::= Statement CommaOrEpsPara loatLiteral ent inusOp Exp 150 Exp	Programl Comman FirstFceF Statemen Exp ::= S Exp ::= E Exp ::= E	gramBody \$ Body := Command ProgramBody d := Exp semicolon d := E	
ataType seclareVariable seclareVariable secVariable secVariable secVariable secVariable secVariable seturn seturnExp unctionBody secVariable secVariab	ElseCond ::= E FunctionBody ::= Comm S ::= ProgramBody S ProgramBody ::= Comm Command ::= Declaret Command ::= Exp seni FirstFceParam ::= State Statement ::= identifier Exp ::= Exp minus Op E Exp := Exp minus Op E Exp :	mand FunctionBody dentifierVar mand ProgramBody /ariable coolon ement CommaOrEpsParam Var xxp p Exp onOp Exp	Exp	== Exp phasOp Exp == Exp divisionOp E == Exp multiplicatio == Exp concatenatic == leftPar Exp right VarAss := Exp sem Cond := & unExp := Exp tionBody := Comm voidKey	Exp nOP Exp onOP Exp onOP Exp iicolon	S := Programl ProgramBody Command := 1 FirstFceParam Exp := Statem Exp := Exp ph Exp := Exp ph Exp := Exp ph Exp := Exp dr Exp := Exp dr	Body \$ == Command ProgramBody Exp semicolon a := Statement CommaOrEpsPara doatLiteral ent musOp Exp usOp Exp usionOp Exp usionOp Exp	Programi Comman FirstFceF Statemen Exp == 8 Exp == E Exp == E Exp == E Exp == E	gramBody \$ Body := Command ProgramBody d := Exp semicolon error := Statement CommaOrEpsParam t := stringLiteral atacment xp minusOp Exp xp plusOp Exp xp divisionOp Exp xp multiplicationOp Exp xp multiplicationOp Exp	
ataType seclareVariable seclareVariable seclareVariable setVarAss ondition lseCond while seturn seturnExp unctionBody ommand ceDefine ceHeader unctionDeclareParams ommaOrEpsParams seclareParam uncReturnColonType ceCall irsFceParam tatement	ElseCond ::= s FunctionBody ::= Comm is S ::= ProgramBody S ProgramBody ::= Comm Command := Declare V Command ::= Exp sem FirstFceParam ::= State Statement ::= identifier Exp ::= Statement Exp ::= Exp minus Op E Exp ::= Exp physion Op	mand FunctionBody dentifierVar mand ProgramBody /ariable coolon ement CommaOrEpsParam Var xxp p Exp onOp Exp	Exp	== Exp phasOp Exp == Exp divisionOp E == Exp multiplicatio == Exp concatenatic == leftPar Exp right VarAss := Exp sem Cond := & unExp := Exp tionBody := Comm voidKey	Exp nOP Exp onOP Exp onOP Exp iicolon	S := Programl ProgramBody Command := 1 FirstFceParam Exp := Statem Exp := Exp ph Exp := Exp ph Exp := Exp ph Exp := Exp dr Exp := Exp dr	Body \$:= Command ProgramBody Exp semicolon 1:= Statement CommaOrEpsPara doatLiteral ent entsOp Exp visionOp Exp visionOp Exp	Programi Comman FirstFceF Statemen Exp == 8 Exp == E Exp == E Exp == E Exp == E	gramBody \$ Sody := Command ProgramBody d := Exp semicolon earam := Statement CommaOrEpsParam t := stringLiteral tatement xp minusOp Exp xp plusOp Exp xp divisionOp Exp	
hataType heclareVariable heclareVariable her Variable her	ElseCond ::= \$ FunctionBody ::= Comm is S ::= ProgramBody \$ ProgramBody ::= Comm Command := Declare V Command ::= Exp sem FirstFceParam ::= State Statement ::= identifier Exp ::= Exp minusOp E Exp ::= Exp divisionOp Exp ::= Exp multipReat Exp ::= Exp multipReat Exp ::= Exp multipReat Exp ::= Exp concatenat	mand FunctionBody dentifierVar mand ProgramBody ariable colon ement CommaOrEpsParam Var xp p Exp onOp Exp ionOp Exp	Exp	== Exp phasOp Exp == Exp divisionOp E == Exp multiplicatio == Exp concatenatic == leftPar Exp right VarAss := Exp sem Cond := & unExp := Exp tionBody := Comm voidKey	Exp nOP Exp onOP Exp onOP Exp iicolon	S := Programl ProgramBody Command := 1 FirstFceParam Exp := Statem Exp := Exp ph Exp := Exp ph Exp := Exp ph Exp := Exp dr Exp := Exp dr	Body \$ == Command ProgramBody Exp semicolon a := Statement CommaOrEpsPara doatLiteral ent musOp Exp usOp Exp usionOp Exp usionOp Exp	Programi Comman FirstFceF Statemen Exp == 8 Exp == E Exp == E Exp == E Exp == E	gramBody \$ Body := Command ProgramBody d := Exp semicolon error := Statement CommaOrEpsParam t := stringLiteral atacment xp minusOp Exp xp plusOp Exp xp divisionOp Exp xp multiplicationOp Exp xp multiplicationOp Exp	
DataType DeclareVariable DefVarAss	ElseCond ::= \$ FunctionBody ::= Comm is S ::= ProgramBody \$ ProgramBody ::= Comm Command := Declare V Command ::= Exp sem FirstFceParam ::= State Statement ::= identifier Exp ::= Exp minusOp E Exp ::= Exp divisionOp Exp ::= Exp multipReat Exp ::= Exp multipReat Exp ::= Exp multipReat Exp ::= Exp concatenat	mand FunctionBody IdentifierVar mand ProgramBody Ariable colon ement CommaOrEpsParam Var Arp p Exp onOp Exp iniOp Exp mtifierVar equals DefVarAss	Exp	== Exp phasOp Exp == Exp divisionOp E == Exp multiplicatio == Exp concatenatic == leftPar Exp right VarAss := Exp sem Cond := & unExp := Exp tionBody := Comm voidKey	Exp nOp Exp onOp Exp iicolon	S := Programl ProgramBody Command := 1 FirstFceParam Exp := Exp mi Exp := Exp di Exp := Exp di Exp := Exp do Exp := Exp do Exp := Exp co	Body \$ == Command ProgramBody Exp semicolon a := Statement CommaOrEpsPara doatLiteral ent musOp Exp usOp Exp usionOp Exp usionOp Exp	Programi Comman TirstFceF Stateme Exp == E	gramBody \$ Body := Command ProgramBody d := Exp semicolon error := Statement CommaOrEpsParam t := stringLiteral atacment xp minusOp Exp xp plusOp Exp xp divisionOp Exp xp multiplicationOp Exp xp multiplicationOp Exp	

ElseCond ::= ε

ElseCond ::= ε

	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 phusOp 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	ε ElseCond ::= ε					
While							
Return							
ReturnExp	ReturnExp ::= Exp	ReturnExp ::= Exp					
FunctionBody	FunctionBody ::= Command FunctionBody	FunctionBody ::= Command FunctionBody					
	-4-d	ataingNullVan			in (No.11)		

	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 ::= 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
While		While ::= whileKey leftPar Exp rightPar curlyBraceLeft FunctionBody curlyBraceRigh	
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