

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	7
6	Členění implementačního řešení	8
7	Závěr	11

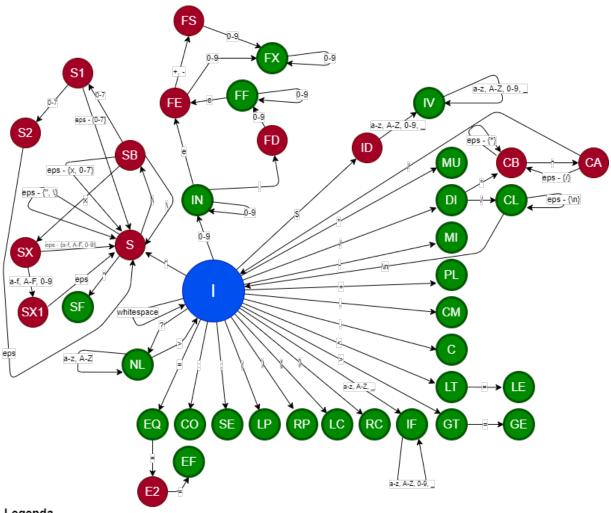
Rozdělení práce mezi členy týmu

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 2 s
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
FE = float_lit_e_s
                               SB = string_lit_backslash_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
DI = division f s
                               CB = com_block_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

```
ProgramBody ::= FceDefine ProgramBody
ProgramBody ::= Command ProgramBody
ProgramBody ::="
Command ::= DeclareVariable
Command ::= Condition
Command ::= While
Command ::= Return
Command ::= Exp semicolon
Command ::= semicolon
Command ::= FceCall
FceDefine ::= functionKey FceHeader curlyBraceLeft FunctionBody curlyBraceRight
FceHeader ::= identifierFunc leftPar FunctionDeclareParams rightPar colon FuncReturnCo-
lonType
FunctionDeclareParams ::="
FunctionDeclareParams ::= DataType DeclareParam CommaOrEpsParams
CommaOrEpsParams ::="
CommaOrEpsParams ::= comma DataType DeclareParam CommaOrEpsParams
DeclareParam ::= 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
Statement ::= nullKey
Exp ::= Statement
Exp ::= Exp minusOp Exp
Exp ::= Exp plusOp Exp
Exp ::= Exp divisionOp Exp
Exp ::= Exp multiplicationOp Exp
Exp ::= Exp concatenationOp Exp
Exp ::= Exp lesserThanOp Exp
Exp ::= Exp lesserEqOp Exp
```

Exp ::= Exp greaterThanOp Exp Exp ::= Exp greaterEqOp Exp

```
Exp ::= Exp eqOp Exp
Exp ::= Exp notEqOp 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

 $Condition ::= if Key \ left Par \ Exp \ right Par \ curly Brace Left \ Function Body \ curly Brace Right \ El-par \ Exp \ right Par \ Condition ::= if Key \ left Par \ Exp \ right Par \ curly Brace Left \ Function Body \ curly Brace Right \ El-par \ Exp \ right Par \ curly Brace Par \ Function Body \ curly Brace Right \ El-par \ Exp \ right Par \ curly Brace Par \ Function Body \ curly Brace Par \ Exp \ right Par \ curly Brace Par \ Function Body \ curly Brace Par \ Exp \ right Par \ curly Brace Par \ Function Body \ curly Brace Par \ Exp \ right Par \ curly Brace Par \ Function Body \ curly Brace Par \ Exp \ right Par \ curly Brace Par \ Function Body \ curly Brace Par \ Fun$

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

1	S	semicolon	curlyB	raceLeft	curlyBraceRight			fu	nctionKe	ey	
	S ::= ProgramBody	\$ S ::= ProgramBody \$			S	::= ProgramBody					
ProgramBody	ProgramBody ::= ε	ProgramBody ::= Command Progr	ramBody		P	rogramBody ::= F	ceDefine Progra	mBody			
Command		Command ::= semicolon									
FceDefine						ceDefine ::= FceI					
ceHeader						FceHeader ::= functionKey identifierFunc let			nctionDec	lareParams rightPar colon FuncR	eturnColonTy
ElseCond	ElseCond ::= ε				seCond ::= ε E	lseCond ::= ε					
ReturnExp FunctionBody		ReturnExp ::= ε FunctionBody ::= Command FunctionBody	tionBody	Fu	nctionBody ::= ε						
uncuoinbody .		identifierFu		10	ilicuoiiDody e	leftPa				rightPar	
2	C D		шс		C D I		<u> </u>			rigitear	
·		ogramBody \$	n 1		S ::= ProgramI						
ProgramBody	ProgramBody ::= Command ProgramBody				ProgramBody ::= Command ProgramBody						
Command	Command ::= FceCall				Command ::= Exp semicolon						
					Communication 2011						
FunctionDeclarePar	rams							FunctionDeclareParams ::= ε			
CommaOrEpsParar	ms							C	ommaC)rEpsParams ::= ε	
ceCall	FceCall ::= identifierFunc leftPar FirstFceParam right				ntPar						
irstFceParam									FirstFceParam ::= ε		
CommaOrEpsParar	m							C	ommaC)rEpsParam ::= ε	
DefVarAss		Ass ::= FceCall semicolon			DefVarAss ::=	Exp semicol	on			-	
ElseCond	ElseCor				ElseCond ::= ε						
ReturnExp	Ziscon	-			ReturnExp ::=	Exp					
FunctionBody	Function	nBody ::= Command Function	nBody		FunctionBody		FunctionBo	dv			
icuoiiDody	1 diledol	comma			identifierVar			voidKev		floatLiteral	
		- vmull	S	:= ProgramB				· Juney	S	::= ProgramBody \$	
ProgramBody			Pr	ogramBody ::	= Command ProgramB	ody				rogramBody ::= Command Progra	nBody
Command					eclareVariable				c	ommand ::= Exp semicolon	
CommaOrEpsParams	CommaOrEnsParan	ns ::= comma DeclareParam CommaC		ommand ::= E	xp semicoion				_		
uncReturnColonType	- Janua Si Lipsi di di		por mains				FuncReturnCo	lonType ::= voidI	Key		
irstFceParam				rstFceParam :	= Statement CommaO	rEpsParam			F	rstFceParam ::= Statement Comm	aOrEpsParar
CommaOrEpsParam Statement	CommaOrEpsParan	CommaOrEpsParam ::= comma Statement CommaOrEpsParam			entifierVar		1		0.	tatement ::= floatLiteral	
DeclareVariable					e ::= identifierVar equal	s DefVarAss			3	iatement noathterai	
DefVarAss					VarAss ::= Exp semicolon				D	efVarAss ::= Exp semicolon	
ElseCond				seCond ::= ε						lseCond ::= ε	
ReturnExp						rrnExp := Exp				eturnExp ::= Exp	-D - d
unctionBody				ilicuolii30dy	ctionBody ::= Command FunctionBody integerLiteral				FunctionBody ::= Command FunctionBody		
		stringLiteral							п	ullKey	ı
×	C D	D - d ¢		C D.		Literal		C D		6	,
5		ogramBody \$	n 1		ogramBody \$		N - 1 -	S ::= Progra	mBody		
S ProgramBody		ogramBody \$ nBody ::= Command Program	nBody				Body		mBody	\$ ommand ProgramBody	
	Progran		nBody	Progran	ogramBody \$	and ProgramE	Body		mBody dy ::= C	ommand ProgramBody	
Command	Progran Comma	nBody ::= Command Program nd ::= Exp semicolon		Program Comma	ogramBody \$ nBody ::= Comma nd ::= Exp semico	and ProgramE blon	-	ProgramBo	mBody dy ::= C	ommand ProgramBody	
Command FirstFceParam	Program Comma FirstFce	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Comm		Program Comma FirstFce	ogramBody \$ nBody ::= Comma nd ::= Exp semico :Param ::= Statem	and ProgramE olon nent CommaC	-	ProgramBo	mBody dy ::= C	ommand ProgramBody	
Command FirstFceParam Statement	Program Comma FirstFce Stateme	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Commi ent ::= stringLiteral		Program Comma FirstFce Stateme	ogramBody \$ nBody ::= Comma nd ::= Exp semice Param ::= Statem ent ::= integerLiter	and ProgramE olon nent CommaC ral	-	ProgramBo Command ::	mBody dy ::= C := Exp s	ommand ProgramBody semicolon	
Command FirstFceParam Statement DefVarAss	Program Comma FirstFce Stateme DefVar	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Comme ent ::= stringLiteral Ass ::= Exp semicolon		Program Comma FirstFce Stateme DefVar	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem ent ::= integerLiter Ass ::= Exp semic	and ProgramE olon nent CommaC ral	-	ProgramBo Command :: DefVarAss	mBody dy ::= C := Exp s	ommand ProgramBody semicolon	
Command FirstFceParam Statement DefVarAss ElseCond	Program Comma FirstFce Stateme DefVar ElseCor	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Comme ent ::= stringLiteral Ass ::= Exp semicolon nd ::= \(\varepsilon \)		Program Comma FirstFce Stateme DefVar ElseCom	ogramBody \$ nBody ::= Comma nd ::= Exp semico Param ::= Statem ent ::= integerLiter Ass ::= Exp semio d ::= ε	and ProgramE olon nent CommaC ral	-	ProgramBo Command :: DefVarAss ElseCond ::=	mBody dy ::= C := Exp s ::= Exp = ε	ommand ProgramBody semicolon	
Command FirstFceParam Statement DefVarAss ElseCond	Program Comma FirstFce Stateme DefVar ElseCor	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Comme ent ::= stringLiteral Ass ::= Exp semicolon		Program Comma FirstFce Stateme DefVar ElseCom	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem ent ::= integerLiter Ass ::= Exp semic	and ProgramE olon nent CommaC ral	-	ProgramBo Command :: DefVarAss	mBody dy ::= C := Exp s ::= Exp = ε	ommand ProgramBody semicolon	
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp	Program Comma FirstFce Stateme DefVar ElseCor ReturnE	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Comme ent ::= stringLiteral Ass ::= Exp semicolon nd ::= \(\varepsilon \)	aOrEpsParam	Program Comma FirstFce Stateme DefVar ElseCon ReturnE	ogramBody \$ nBody ::= Comma nd ::= Exp semico Param ::= Statem ent ::= integerLiter Ass ::= Exp semio d ::= ε	and ProgramE olon nent CommaC ral colon)rEpsParam	ProgramBo Command :: DefVarAss ElseCond ::= ReturnExp	mBody dy ::= C ::= Exp s :::= Exp :::= Exp :::= Exp	ommand ProgramBody semicolon	
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody	Program Comma FirstFce Stateme DefVar ElseCon ReturnE Function	nBody ::= Command Program and ::= Exp semicolon aParam ::= Statement Comma and ::= stringLiteral Ass ::= Exp semicolon ad ::= s a	aOrEpsParam nBody	Program Comma FirstFce Stateme DefVar ElseCon ReturnE Function	ogramBody \$ nBody := Comma nd := Exp semice cParam := Statem ent := integerLiter Ass := Exp semica d := \$ d := \$ Exp := Exp nBody := Comma	and ProgramE olon nent CommaC ral colon and FunctionB	OrEpsParam Gody	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBo	mBody dy ::= C := Exp s ::= Exp = ε ::= Exp dy ::= C	ommand ProgramBody semicolon semicolon ommand FunctionBody stri	ingKey
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody unctionDeclareParams	Program Comma FirstFce Stateme DefVar ElseCor ReturnE FunctionDeclareParam	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Commit ent ::= stringLiteral Ass ::= Exp semicolon d ::= 8 Exp ::= Exp stringNullKey s := DeclareParam CommaOrEpsParam	aOrEpsParam nBody 	Program Comma FirstFce Stateme DefVar ElseCor ReturnF Function floatNu	ogramBody \$ nBody ::= Comma nd ::= Exp semico Param ::= Statem ent ::= integerLiter Ass ::= Exp semic d ::= 8 Exp ::= Exp int	and ProgramE olon ent CommaC ral colon and FunctionB	ody integral	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp FunctionBo SullKey eclareParam Com	mBody dy ::= C := Exp s ::= Exp = ε ::= Exp dy ::= C	semicolon osemicolon osemicolon osemicolon osemicolon osemicolon osemicolon osemicolon osemicolon osemicolon	eclareParam (
command irstFceParam otatement DefVarAss cliseCond ceturnExp functionBody metionDeclareParams celareParam	Program Comma FirstFce Stateme DefVar ElseCon ReturnE Function	nBody ::= Command Program nd ::= Exp semicolon dParam ::= Statement Comm: ext ::= stringLiteral Ass ::= Exp semicolon ad ::= & Exp ::= Exp Body ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar	aOrEpsParam nBody	Program Comma FirstFce Stateme DefVar ElseCon ReturnF Function floatNul trans:=Deck DataType iden	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem exp ::= Exp semic dd ::= E Exp ::= Exp nBody ::= Comma likey reParam CommaOrEpst littify ar	and ProgramE olon nent CommaC ral colon and FunctionBe	OrEpsParam Gody	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBookullKey eclareParam ComdentifierVar	mBody dy ::= C := Exp s ::= Exp = ε ::= Exp dy ::= C	ommand ProgramBody semicolon semicolon ommand FunctionBody stri	eclareParam identifierVar
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody mictionDeclareParams seclareParam micReturnColonType	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function FunctionDeclareParam DeclareParam := Data	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Commit ent ::= stringLiteral Ass ::= Exp semicolon nd ::= 8 Exp ::= Exp Body ::= Command Function stringNollikey s := DeclareParam CommaOrEpsParam Type identifier Var := DataType Key	aOrEpsParam nBody	Program Comma FirstFce Stateme DefVar ElseCot ReturnF Function fontNu arms := Deck DataType iden ype := DataTy	ogramBody \$ nBody := Comma nd := Exp semico eParam := Statem ent := integerLiter Ass := Exp semi ed := 8 Exp := Exp mBody := Comma IKEY ureParam CommaOrEpsi tifierVar	and ProgramE olon nent CommaC ral colon and FunctionBe	ody intage Intage	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBookullKey eclareParam ComdentifierVar	mBody dy ::= C := Exp s ::= Exp = ε ::= Exp dy ::= C	command ProgramBody semicolon ommand FunctionBody stri Params FunctionDeclareParams := D DeclareParam := DataType FunckerunColonType := Dat DataType := stringKey	eclareParam identifierVar
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody unctionDeclareParams bucklearParam uncReturnColonType	Program Comma FirstFce Stateme DefVar ElseCon ReturnE FunctionDeclareParam:=Data FuncReturnColonType	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Commi nt ::= stringLiteral Ass ::= Exp semicolon nd ::= 8 Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar := DataType DataType	aOrEpsParam nBody FunctionDeclareP, DeclareParam := FuncReturnColon1	Program Comma FirstFce Stateme DefVar ElseCot ReturnF Function fontNu arms := Deck DataType iden ype := DataTy	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem exp ::= Exp semic dd ::= E Exp ::= Exp nBody ::= Comma likey reParam CommaOrEpst littify ar	and ProgramE olon nent CommaC ral colon and FunctionBe Param FunctionDec DeclarePara FuncRetunt DataType ::	oody int? clareParams := D im := DataType := Dat intNullKey	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBo NullKey colareParam Com dentifierVar aType	mBody dy ::= C := Exp s ::= Exp = ε ::= Exp dy ::= C	ommand ProgramBody semicolon ommand FunctionBody stri arams FunctionDeclareParam := DataType FuncketumColonType := Da	eclareParam (identifierVar
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody unctionDeclareParams beclareParams uncReturnColonType ataType	Program Comma FirstFce Stateme DefVar ElseCon ReturnE FunctionDeclareParam:=Data FuncReturnColonType	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Commit ent ::= stringLiteral Ass ::= Exp semicolon nd ::= 8 Exp ::= Exp Body ::= Command Function stringNollikey s := DeclareParam CommaOrEpsParam Type identifier Var := DataType Key	aOrEpsParam nBody FunctionDeclareP, DeclareParam := FuncReturnColon1	Program Comma FirstFce Stateme DefVar ElseCot ReturnF Function fontNu arms := Deck DataType iden ype := DataTy	ogramBody \$ nBody := Comma nd := Exp semico eParam := Statem ent := integerLiter Ass := Exp semi ed := 8 Exp := Exp mBody := Comma IKEY ureParam CommaOrEpsi tifierVar	and ProgramE colon and FunctionBe Param FunctionDeclarePar FuncRetum DataType	ody lareParams := DataType is colonType := DataType is colonType := Data intNullKey S := ProgramBo	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBot vullKey colar@ram Com dentifierVar aType	mBody dy ::= C := Exp s ::= Exp :::= Exp dy ::= C maOrEpsE	command ProgramBody semicolon ommand FunctionBody stri Params FunctionDeclareParams := D DeclareParam := DataType FunckerunColonType := Dat DataType := stringKey	eclareParam (identifierVar
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody unctionDeclareParams beclareParam uncReturnColonType ataType rogramBody	Program Comma FirstFce Stateme DefVar ElseCon ReturnE FunctionDeclareParam:=Data FuncReturnColonType	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Commit ent ::= stringLiteral Ass ::= Exp semicolon nd ::= 8 Exp ::= Exp Body ::= Command Function stringNollikey s := DeclareParam CommaOrEpsParam Type identifier Var := DataType Key	aOrEpsParam nBody FunctionDeclareP, DeclareParam := FuncReturnColon1	Program Comma FirstFce Stateme DefVar ElseCot ReturnF Function fontNu arms := Deck DataType iden ype := DataTy	ogramBody \$ nBody := Comma nd := Exp semico eParam := Statem ent := integerLiter Ass := Exp semi ed := 8 Exp := Exp mBody := Comma IKEY ureParam CommaOrEpsi tifierVar	and ProgramE colon and FunctionBe Param FunctionDee DeclarePara FuncRetum DataType	ody lareParams := D colonType := DataType is colonType := Data intNullKey S := ProgramBod ProgramBody :=	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBo collikey colareParam Com dentifierVar aType ddy \$ = Command Prog	mBody dy ::= C := Exp s ::= Exp :::= Exp dy ::= C maOrEpsE	command ProgramBody semicolon ommand FunctionBody stri Params FunctionDeclareParams := D DeclareParam := DataType FunckerunColonType := Dat DataType := stringKey	eclareParam (identifierVar
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody Statement DeclareParam uncReturnColonType DataType StrogramBody Command	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function FunctionDeclareParam := Data FuncReturnColonType DataType := stringNull	nBody ::= Command Program nd ::= Exp semicolon dParam ::= Statement Commant i:= stringLiteral Ass ::= Exp semicolon ad ::= & Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifier Var ::= DataType Key floatKey	aOrEpsParam Body FunctionDeclareP DeclareParam = FuncReturnColon1 DataType := float	Program Comma FirstFce Stateme DefVar ElseCor Returni Function floatNul rams:= Deck DataType iden ype := DataTy NullKey	ogramBody \$ nBody ::= Comma nd ::= Exp semico cParam ::= Statem ent ::= integerLiter cAss ::= Exp semico d ::= & d ::= & d ::= Exp nBody ::= Comma likey urcParam CommaOrEpst liferVar trpe intKey	and ProgramE colon and FunctionB Param FunctionDeclarePar FuncReturn DataType	ody lareParams := D colonType := Data intNullKey S := ProgramBody := Command := Co	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBo collikey colareParam Com dentifierVar aType ddy \$ = Command Prog	mBody dy ::= C := Exp s ::= Exp :::= Exp dy ::= C maOrEpsE	command ProgramBody semicolon ommand FunctionBody stri Params FunctionDeclareParams := D DeclareParam := DataType FunckerunColonType := Dat DataType := stringKey	eclareParam (identifierVar
Command CirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody unctionDeclareParams reclareParam uncReturnColonType statType rogramBody command unctionDeclareParams	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function FunctionDeclareParam:= Data FuncReturnColonType DataType:= stringNull FunctionDeclareParam	nBody ::= Command Program nd ::= Exp semicolon Param ::= Statement Commit ent ::= stringLiteral Ass ::= Exp semicolon nd ::= 8 Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar ::= DataType Key floatKey ams ::= DeclareParam CommaOrEpsParam Type identifierVar	aOrEpsParam nBody FunctionDeclareParam := FuncReturnColl DataType := float	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function floatNu trams := Declar DetVar Type := DataTy NullKey	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem exp ::= integerLiter Ass ::= Exp semic nd ::= Exp semic nd ::= Exp nBody ::= Comma IKEY intervar inte	and ProgramE colon and FunctionB Param FunctionDeclarePar FuncReturn DataType	ody lareParams := D colonType := Data intNullKey S := ProgramBody := Command := Co	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBo collikey colareParam Com dentifierVar aType ddy \$ = Command Prog	mBody dy ::= C := Exp s ::= Exp :::= Exp dy ::= C maOrEpsE	command ProgramBody semicolon ommand FunctionBody stri Params FunctionDeclareParams := D DeclareParam := DataType FunckerunColonType := Dat DataType := stringKey	eclareParam identifierVar
Command irstFceParam itatement DefVarAss CiseCond CeturnExp UnctionBody microinDeclareParams mcReturnColonType ataType rogramBody ommand unctionDeclareParams eclareParam	Program Comma FirstFce Stateme DefVar ElseCor ReturnE FunctionDeclareParam:= Data FuncReturnColonType DataType:= stringNull FunctionDeclareParam:= Data FunctionDeclareParam:= Data	nBody ::= Command Program nd ::= Exp semicolon param ::= Statement Commi ent ::= stringLiteral Ass ::= Exp semicolon ad ::= & Exp ::= Exp Body ::= Command Function stringNullKey s ::= DeclareParam CommaOrEpsParam Type identifierVar ::= DataType Key floatKey ams ::= DeclareParam CommaOrEps ataType identifierVar	aOrEpsParam nBody FunctionDeclarePr DeclarePram = FuncReturnColonI DataType := float	Program Comma FirstFce Stateme DefVar ElseCon Return Function Fontallype in DataType iden ppe := DataType iden program colored by the program	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem eParam ::= Statem Ass ::= Exp semic dd ::= & Exp ::= Exp nBody ::= Comma Ilkey intKey ::= DeclareParam Com Type identifierVar	and ProgramE colon and FunctionB Param FunctionDeclarePar FuncReturn DataType	ody lareParams := D colonType := Data intNullKey S := ProgramBody := Command := Co	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBo collikey colareParam Com dentifierVar aType ddy \$ = Command Prog	mBody dy ::= C := Exp s ::= Exp :::= Exp dy ::= C maOrEpsE	command ProgramBody semicolon ommand FunctionBody stri Params FunctionDeclareParams := D DeclareParam := DataType FunckerunColonType := Dat DataType := stringKey	eclareParam identifierVar
Command CirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody muctionDeclareParams seclareParam macReturnColonType ataType rogramBody rogramBody rommand muctionDeclareParams seclareParam seclareParams seclareParams seclareParams seclareParams seclareParams seclareParams seclareParams seclareParams seclareParams	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function FunctionDeclareParam:= Data FuncReturnColonType DataType:= stringNull FunctionDeclareParam	nBody ::= Command Program nd ::= Exp semicolon dParam ::= Statement Commant i::= stringLiteral Ass ::= Exp semicolon ad ::= 8 Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar ::= DataType MontKey mans ::= DeclareParam CommaOrEpsParam Type identifierVar pe ::= DataType analype identifierVar pe ::= DataType	aOrEpsParam nBody FunctionDeclarePr DeclarePram = FuncReturnColonI DataType := float	Program Comma FirstFce Stateme DefVar ElseCot Returni Function floatNul rams:= Deck DataType iden ype:= DataTy vullKey ecclareParams ram := DataTn nColonType:	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem eParam ::= Statem Ass ::= Exp semic dd ::= & Exp ::= Exp nBody ::= Comma Ilkey intKey ::= DeclareParam Com Type identifierVar	and ProgramE colon and FunctionB Param FunctionDeclarePar FuncReturn DataType	ody lareParams := D colonType := Data intNullKey S := ProgramBody := Command := Co	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBo collikey colareParam Com dentifierVar aType ddy \$ = Command Prog	mBody dy ::= C := Exp s ::= Exp :::= Exp dy ::= C maOrEpsE	command ProgramBody semicolon ommand FunctionBody stri Params FunctionDeclareParams := D DeclareParam := DataType FunckerunColonType := Dat DataType := stringKey	eclareParam identifierVar
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody unctionDeclareParams DeclareParam uncReturnColonType DataType TogramBody Command UnctionDeclareParams DeclareParam UnctionDeclareParams DeclareParams DeclarePa	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function DeclareParam := Data FunctionDeclareParam:= D FunctionDeclareParam:= D FunctionDeclareParam:= D	nBody ::= Command Program nd ::= Exp semicolon dParam ::= Statement Commant i::= stringLiteral Ass ::= Exp semicolon ad ::= 8 Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar ::= DataType MontKey mans ::= DeclareParam CommaOrEpsParam Type identifierVar pe ::= DataType analype identifierVar pe ::= DataType	aOrEpsParam Body FunctionDeclareP DeclareParam = FuncReturnColon1 DataType := float DeclarePa FuncReturnColon2	Program Comma FirstFce Stateme DefVar ElseCot Returni Function floatNul rams:= Deck DataType iden ype:= DataTy vullKey ecclareParams ram := DataTn nColonType:	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem eParam ::= Statem Ass ::= Exp semic dd ::= & Exp ::= Exp nBody ::= Comma Ilkey intKey ::= DeclareParam Com Type identifierVar	and ProgramE olon nent CommaC ral colon and FunctionB DeclarePar FuncRetunt DataType:	ody intage large params in Dispersion in State Params in Dispersion in State Program Body in Command in Command in Command in Command in Command in Information In	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp: FunctionBo Soulkey ecclareParam Com dentifierVar aType dedy \$ = Command Prog ondition	mBody dy := Cxp s := Exp s ::= Exp = \(\epsilon \) := Cxp := Exp dy := C	command ProgramBody semicolon ommand FunctionBody stri Params FunctionDeclareParams := D DeclareParam := DataType FunckerunColonType := Dat DataType := stringKey	eclareParam identifierVar taType
Command CirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody MunctionDeclareParams MarketurnColonType Marketurn	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function DeclareParam := Data FunctionDeclareParam:= D FunctionDeclareParam:= D FunctionDeclareParam:= D	nBody ::= Command Program nd ::= Exp semicolon dParam ::= Statement Commant i::= stringLiteral Ass ::= Exp semicolon ad ::= 8 Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar ::= DataType MontKey mans ::= DeclareParam CommaOrEpsParam Type identifierVar pe ::= DataType analype identifierVar pe ::= DataType	aOrEpsParam Body FunctionDeclareP DeclareParam = FuncReturnColon1 DataType := float DeclarePa FuncReturnColon2	Program Comma FirstFce Stateme DefVar ElseCot Returni Function floatNul rams:= Deck DataType iden ype:= DataTy vullKey ecclareParams ram := DataTn nColonType:	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem eParam ::= Statem Ass ::= Exp semic dd ::= & Exp ::= Exp nBody ::= Comma Ilkey intKey ::= DeclareParam Com Type identifierVar	and ProgramE solon ment CommaC ral colon and FunctionBe DeclarePar FuncReturn DataType: amaOrEpsParams	ody lareParams := D colonType := Dat intNullKey S := ProgramBody := Command := Co	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBot Valikey celareParam Com dentifierVar aType ddy \$ = Command Prog ondition	mBody dy := Cxp s ::= Exp s	semicolon semicolon ommand FunctionBody stri Params FunctionDeclareParams := D DeclareParam := DataType FuncRetumColonType := Data DataType := stringKey ifKey	eclareParam identifierVar taType
Command CirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody MunctionDeclareParams MarketurnColonType Marketurn	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function DeclareParam := Data FunctionDeclareParam:= D FunctionDeclareParam:= D FunctionDeclareParam:= D	nBody ::= Command Program nd ::= Exp semicolon cParam ::= Statement Common nd ::= stringLiteral Ass ::= Exp semicolon nd ::= 8 Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar pe DataType Key floatKey ams ::= DeclareParam CommaOrEps statType identifierVar pe ::= DataType ey ::= DataType ey	aOrEpsParam Body FunctionDeclareP DeclareParam = FuncReturnColon1 DataType := float DeclarePa FuncReturnColon2	Program Comma FirstFce Stateme DefVar ElseCot Returni Function floatNul rams:= Deck DataType iden ype:= DataTy vullKey ecclareParams ram := DataTn nColonType:	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem eParam ::= Statem Ass ::= Exp semic dd ::= & Exp ::= Exp nBody ::= Comma Ilkey intKey ::= DeclareParam Com Type identifierVar	and ProgramE olon nent CommaC ral colon and FunctionBe Param FunctionDec DeclarePar FuncRetum DataType ::	ody lareParams := DataType is ColonType := Dat intNullKey S := ProgramBody := Command := Co	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp: FunctionBo Soulkey ecclareParam Com dentifierVar aType dedy \$ = Command Prog ondition	mBody dy := Cxp s ::= Exp s	semicolon semicolon semicolon ommand FunctionBody stri arams FunctionDeclareParams := D DeclareParam := DataType FuncketumColonType := Da DataType := stringKey iffKey	veclareParam identifierVar taType
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody unctionDeclareParams beclareParam uncReturnColonType ataType TogramBody Togram	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function DeclareParam := Data FunctionDeclareParam:= D FunctionDeclareParam:= D FunctionDeclareParam:= D	nBody ::= Command Program nd ::= Exp semicolon dParam ::= Statement Commant i::= stringLiteral Ass ::= Exp semicolon ad ::= 8 Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar ::= DataType MontKey mans ::= DeclareParam CommaOrEpsParam Type identifierVar pe ::= DataType analype identifierVar pe ::= DataType	aOrEpsParam Body FunctionDeclareP DeclareParam = FuncReturnColon1 DataType := float DeclarePa FuncReturnColon2	Program Comma FirstFcc Stateme DefVar ElseCon ReturnF Function Function Function Function Function PostNull Function Fun	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem exp ::= Exp semic d ::= Exp mBody ::= Comma IKEY intervar pre intKey ::= DeclareParam Com type identifierVar = DataType	and ProgramE solon ment CommaC ral colon and FunctionBe DeclarePar FuncReturn DataType: amaOrEpsParams	ody lareParams := DataType is ColonType := Dat intNullKey S := ProgramBody := Command := Co	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBot Valikey celareParam Com dentifierVar aType ddy \$ = Command Prog ondition	mBody dy := Cxp s ::= Exp s	semicolon semicolon semicolon ommand FunctionBody params FunctionDeclareParam := DataType FuncketumColonType := DataType FuncketumColonType := DataType FuncketumColonType := StringKey ifKey returnKe	veclareParam identifierVar taType
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody unctionDeclareParams DeclareParam uncReturnColonType ataType Command UnctionDeclareParams DeclareParam uncketurnColonType ataType Command UnctionDeclareParams DeclareParam UncketurnColonType ataType Ondition UnctionBody Statistics St	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function Function DeclareParam:= Data FunctionDeclareParam:= D FunctionDeclareParam:= D FunctionOcclareParam:= D FunctionOcclareParam:= D FunctionOcclareParam:= D FunctionOcclareParam:= D FuncReturnColonTy	nBody ::= Command Program nd ::= Exp semicolon cParam ::= Statement Common nd ::= stringLiteral Ass ::= Exp semicolon nd ::= 8 Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar pe DataType Key floatKey ams ::= DeclareParam CommaOrEps statType identifierVar pe ::= DataType ey ::= DataType ey	aOrEpsParam Body FunctionDeclareP DeclareParam = FuncReturnColon1 DataType := float DeclarePa FuncReturnColon2	Program Comma FirstFcc Stateme DefVar ElseCon Returnf Function floatNui rams:= Deck DataType iden ype := DataTy vullKey ecclareParams ram := DataTn colonType ::= intKey	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem eParam ::= Statem Ass ::= Exp semic dd ::= & Exp ::= Exp nBody ::= Comma likey reParam CommaOrEpst infied Var ppe intKey ::= DeclareParam Com Type identifierVar = DataType gramBody \$	and ProgramE olon nent CommaC ral colon and FunctionBe DeclarePara FuncRetund DataType ::	ody lareParams := DataType is ColonType := Dat intNullKey S := ProgramBody := Command := Co	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBot Valikey celareParam Com dentifierVar aType ddy \$ = Command Prog ondition	mBody dy := Cxp s ::= Exp s	semicolon semicolon semicolon ommand FunctionBody stri params FunctionDeclareParams := D DeclareParam := DataType FuncketurnColonType := Da DataType := stringKey ifKey ifWey returnKe S := ProgramBody S	veclareParam identifierVar taType
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function Function DeclareParam:= Data FunctionDeclareParam:= D FunctionDeclareParam:= D FunctionOcclareParam:= D FunctionOcclareParam:= D FunctionOcclareParam:= D FunctionOcclareParam:= D FuncReturnColonTy	nBody ::= Command Program nd ::= Exp semicolon cParam ::= Statement Common nd ::= stringLiteral Ass ::= Exp semicolon nd ::= 8 Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar pe DataType Key floatKey ams ::= DeclareParam CommaOrEps statType identifierVar pe ::= DataType ey ::= DataType ey	aOrEpsParam Body FunctionDeclareP DeclareParam = FuncReturnColon1 DataType := float DeclarePa FuncReturnColon2	Program Comma FirstFcc Stateme DefVar ElseCon Returnf Function floatNui rams:= Deck DataType iden ype := DataTy vullKey ecclareParams ram := DataT nColonType := intKey S ::= Pro Program	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem eParam ::= Statem Ass ::= Exp semic dd ::= & Exp ::= Exp nBody ::= Comma Ilkey intKey ::= DeclareParam CommaOrEpst intiger ar prope intKey ::= DeclareParam CommaOrEpst inty ar prope inty	and ProgramE olon nent CommaC ral colon and FunctionBe DeclarePara FuncRetund DataType ::	ody lareParams := DataType is ColonType := Dat intNullKey S := ProgramBody := Command := Co	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBot Valikey celareParam Com dentifierVar aType ddy \$ = Command Prog ondition	mBody dy := Cxp s ::= Exp s	semicolon semicolon semicolon ommand FunctionBody stri params FunctionDeclareParams := D DeclareParam := DataType FuncketurnColonType := Da DataType := stringKey ifKey ifKey returnKe S ::= ProgramBody := Comman	veclareParam identifierVar taType
DeclareParam umcRetumColonType DataType ProgramBody Command FunctionDeclareParams DeclareParam FuncRetumColonType DataType Condition ElseCond	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function Function DeclareParam:= Data FunctionDeclareParam:= D FunctionDeclareParam:= D FunctionOcclareParam:= D FunctionOcclareParam:= D FunctionOcclareParam:= D FunctionOcclareParam:= D FuncReturnColonTy	nBody ::= Command Program nd ::= Exp semicolon cParam ::= Statement Common nd ::= stringLiteral Ass ::= Exp semicolon nd ::= 8 Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar pe DataType Key floatKey ams ::= DeclareParam CommaOrEps statType identifierVar pe ::= DataType ey ::= DataType ey	aOrEpsParam Body FunctionDeclareP DeclareParam = FuncReturnColon1 DataType := float DeclarePa FuncReturnColon2	Program Comma FirstFcc Stateme DefVar ElseCon Returnf Function floatNui rams:= Deck DataType iden ype := DataTy vullKey ecclareParams ram := DataT nColonType := intKey S ::= Pro Program	ogramBody \$ nBody ::= Comma nd ::= Exp semice eParam ::= Statem eParam ::= Statem Ass ::= Exp semic dd ::= & Exp ::= Exp nBody ::= Comma likey reParam CommaOrEpst infied Var ppe intKey ::= DeclareParam Com Type identifierVar = DataType gramBody \$	and ProgramE olon nent CommaC ral colon and FunctionBe DeclarePara FuncRetund DataType ::	ody lareParams := DataType is ColonType := Dat intNullKey S := ProgramBody := Command := Co	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBot Valikey celareParam Com dentifierVar aType ddy \$ = Command Prog ondition	mBody dy := Cxp s ::= Exp s	semicolon semicolon semicolon ommand FunctionBody stri params FunctionDeclareParams := D DeclareParam := DataType FuncketurnColonType := Da DataType := stringKey ifKey ifWey returnKe S := ProgramBody S	veclareParam of dentifierVar taType
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody FunctionBody FunctionDeclareParams FuncReturnColonType DataType Command FuncReturnColonType DataType Condition ElseCond Condition ElseCond Condition Condition ElseCond Condition C	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function DeclareParam:= Data FuncReturnColonType DataType:= stringNull FunctionDeclareParam:= D FuncReturnColonType DataType:= floatK	nBody ::= Command Program nd ::= Exp semicolon cParam ::= Statement Common nd ::= stringLiteral Ass ::= Exp semicolon nd ::= 8 Exp ::= Exp nBody ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar pe DataType Key floatKey ams ::= DeclareParam CommaOrEps statType identifierVar pe ::= DataType ey ::= DataType ey	aOrEpsParam nBody FunctionDeclarePr DeclareParam := FuncReturnColonI DataType := float Params FunctionD DeclarePa FuncRetur DataType	Program Comma FirstFcc Stateme DefVar ElseCon Return Function Func	ogramBody \$ nBody ::= Comma nd ::= Exp semico eParam ::= Statem eAs ::= Exp semic d ::= Exp nBody ::= Comma IKEY intervar pre intKey ::= DeclareParam Con Type identifierVar = DataType gramBody \$ Body ::= Command I d ::= While	and ProgramE olon nent CommaC ral colon and FunctionBe DeclarePara FuncRetund DataType ::	ody lareParams := DataType is ColonType := Dat intNullKey S := ProgramBody := Command := Co	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBot Valikey celareParam Com dentifierVar aType ddy \$ = Command Prog ondition	mBody dy := Cxp s ::= Exp s	semicolon semicolon semicolon ommand FunctionBody stri params FunctionDeclareParams := D DeclareParam := DataType FuncketurnColonType := Da DataType := stringKey ifKey ifKey returnKe S ::= ProgramBody := Comman	veclareParam of dentifierVar taType
Command FirstFceParam Statement DefVarAss ElseCond ReturnExp FunctionBody	Program Comma FirstFce Stateme DefVar ElseCor ReturnE Function DeclareParam:= Data FuncReturnColonType DataType:= stringNull FunctionDeclareParam:= D FuncReturnColonType DataType:= floatK	nBody ::= Command Program nd ::= Exp semicolon cParam ::= Statement Comm. ent ::= stringLiteral Ass ::= Exp semicolon dd ::= & Exp ::= Exp Body ::= Command Function stringNullKey s := DeclareParam CommaOrEpsParam Type identifierVar ::= DataType Key floatKey ams ::= DeclareParam CommaOrEps ataType identifierVar pe ::= DataType entire identifierVar pe ::= DataType ey elseKey	nBody FunctionDeclarePr DeclareParam: FuncReturnColonI DataType: FuncRetur DataType DeclareParams FunctionD DeclareParams FuncRetur DataType	Program Comma FirstFcc Stateme DefVar ElseCor ReturnF Function foatNul rams:=DecloatType iden pype := DataTy NullKey S:=Pro Program Commar ght ElseCon	ogramBody \$ nBody ::= Comma nd ::= Exp semico eParam ::= Statem eAs ::= Exp semic d ::= Exp nBody ::= Comma IKEY intervar pre intKey ::= DeclareParam Con Type identifierVar = DataType gramBody \$ Body ::= Command I d ::= While	and ProgramE olon nent CommaC ral colon and FunctionBe Param FunctionDec DeclarePar FuncRetume DataType: whilek ProgramBody	ody clareParams := Do m := DataType := Do clorType := Data intVullKey S := ProgramBody := Command := Co Condition := ifK ElseCond := e FunctionBody := Cey	ProgramBo Command :: DefVarAss ElseCond :: ReturnExp : FunctionBot NullKey adv \$ Command Prog ondition ey leftPar Exp ri	mBody dy := Cxp s ::= Exp	semicolon ommand ProgramBody semicolon ommand FunctionBody stri grams FunctionDeclareParams := D DeclareParam := DataType FuncketunColonType := Da DataType := stringKey ifKey returnKe s := ProgramBody curlyB returnKe S := ProgramBody := Comman Command := Return ElseCond := e	eclareParam identifierVar taType taType braceRight Els y d ProgramB

Precedenční tabulka

	+-	*/	ID lit.	•	()	boolOP	\$
+-	>	<	<	>	<	>	>	>
* /	>	>	<	>	<	>	>	>
ID lit.	>	>		>		>	>	>
•			<		<	>	>	>
(>	'	'	'	<	=	\	<
)	>	>		>		>	>	>
boolOP	<	<	<	<	<	>		>
\$	<	v	'	'	<		<	

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.

Syntaktický analyzátor je hlavní řídící jednotkou překladače a podobně jako lexikální analyzátor je implementován ve dvou souborech - hlavičkovém *parser.h* a zdrojovém *parser.c.* Využívá prediktivní analýzu řízenou LL-tabulkou. Tento postup byl zvolen z důvodu své relativní jednoduchosti.

Analyzátor využívá dynamicky uloženou tabulku vytvořenou voláním funkce createLLTable() v souboru *LLTable.c.* Ta volá staticky zapsanou sadu funcí, které vkládají pravidla do tabulky. Nabízí poté rozhraní, které voláním funce getLLMember() zavolá. Buňka tabulky je uložena

jako struktura tableMember, která ukládá neterminál (výčtový typ nonTerminalType), terminál (výčtový typ lexType) a pravidla (vnořená struktura).

Při překladu syntaktický analyzátor zpracovává tokeny, které mu předá lexikální analyzátor, a používá strukturu stack definovanou v hlavičkovém souboru stack.h, kam poté ukládá struktury stackMem_t může být terminál, neterminál nebo precedent. Pokud analyzátor detekuje, že na vrcholu zásobníku je neterminál Expression nebo BoolExp, předává řízení precedenčnímu analyzátoru. Pro správné zpracování se zde se používá funce ungetToken().

TODO

Precedenční analyzátor používá precedenční tabulku, která je součástí tohoto dokumentu. Je tvořen opět dvěma soubory - zdrojovým expParse.c a hlavičkovým expParse.h. Precedenční analyzátor načítá tokeny ze zdroje skrze funkci getToken() z lexikálního analyzátoru a poté vyhledává v precedenční tabulce asociativitu (tabulka je implementována jako dvojrozměrné pole). Stack může obsahovat asociativní pravidla, token, případně neterminály (výraz nebo pravdivostní výraz). Pokud je detekována levá asociativita, jsou prohledána všechna pravidla pro výraz. Pokud není pravidlo nalezeno, je vrácena chyba. Takto se pokračuje, dokud se na stacku nenachází pouze znak \$. Poté je řízení předáno zpět syntaktickému analyzátoru.

LL-tabulka je v našem případě implementována jako dvojrozměrné pole. Bohužel se nám nepodařilo vymyslet způsob, kterým by bylo možno tabulku vytvořit staticky takovým způsobem, aby byla dostatečně přehledná. Proto je tabulka vytvářena za běhu pomocí funkce insertMember(), která vkládá pravidla na příslušné místo v tabulce. Kostrou LL-tabulky je zdrojový soubor *LLtable.c* a hlavičkový soubor *LLtable.h*.

Gramatika a její návrh byl jednou z prvních věcí, které se náš tým při práci na projektu věnoval. Návrh probíhal jako týmová aktivita, kdy jsme procházeli jednotlivé části syntaxe jazyka IFJ22 a vzájemně si zpochybňovali své navrhy. Postupně jsme tato pravidla sepsali do souboru *LLgrammar.txt* a v průběhu práce na projektu gramatiku průběžně vylepšovali. Tabulka je poměrně obsáhlá a vyskytuje se v ní mnoho pravidel, domníváme se však, že značnou část syntaktických chyb se našemu překladači daří odchytávat právě díky tomu.

Generace kódu je implementována tak, že kdekoliv, kde je provedena derivace, se rozhodne (na základě provedené derivace), která funkce pro generování kódu bude použita. Hodnoty a další informace o literálech a proměnných jsou ukládány na stack, odkud jsou při generaci načteny a následně párovány s konkrétními instrukcemi z IFJcode22. Kvůli nutnosti definice proměnných před cykly (aby nedošlo k redefinici proměnných při interpretaci, což by vyústilo v chybu) byl použit buffer, do kterého se prvně generuje kód. V bufferu je možné měnit pořadí jednotlivých příkazů. Tento buffer je implementován jako lineární seznam. Po dokončení překladu jedné komponenty je celý buffer vypsán na standardní výstup.

Pomocné soubory: Tato implementace překladače obsahuje mimo jiné také zdrojový soubor *common.c* a jeho hlavičkový soubor *common.h*. Jde o pomocnou knihovnu obsahující různé pomocné funkce a makra, která napomáhají běhu programu ve všech jeho součástech. Mimo jiné tyto soubory obsahují např. definice všech možných chyb, a také funkce, které vypisují příslušnou chybu v případě, že některá část programu selže - například se zde vyskytuje funkce PrintErrorExit(), která je zodpovědná za výpis chyby s odpovídajícím chybovým kódem a následné korektní ukončení programu. Mimo to se zde objevují kontrolní prvky pro některé z testovacích souborů, které jsme vytvořili a posléze využili k testování jednotlivých částí pro-

gramu. Pomocné soubory rovněž zahrnují například definice všech terminálů a neterminálů.

Pomocné soubory jsou v programu využívány např. pro ladicí výpisy. To se ukázalo jako nesmírně efektivní cesta k debugování programu - kdykoliv se vyskytla chyba, z chybového výpisu se dalo vyčíst mnoho údajů, které pomohly chybu eliminovat. V souvislosti s tím jsme se zároveň naučili používat variadická makra; práce s proměnným počtem argumentů nám velmi pomohla právě při psaní ladicích funkcí.

Závěr

Pro náš tým šlo o poměrně velmi náročný projekt, který zabral spoustu času a práce. Všichni členové týmu se však shodli na tom, že je projekt velmi přínosný a člověk, který jej dokončí, získá spoustu důležitých znalostí a zkušeností. Velkou výhodou projektu je, že zadání nabízí relativně velkou svobodu a spoustu věcí může člověk implementovat libovolným způsobem či metodou. Samotný rozsah projektu nutí členy týmu, aby skutečně spolupracovali, společně rozvrhovali čas a úkoly, což náš tým hodnotí pozitivně.

Naše práce na projektu probíhala formou četných schůzek a plánování, společného programování ať už online či offline a vzájemnou kontrolou. Jednotlivé dílčí části projektu byly rozděleny mezi členy týmu dle zájmu a schopností. Pro správu projektu byl využit populární verzovací systém Git, respektive cloudové úložiště GitHub. Lepší schopnost práce s Gitem a celkově větší porozumění tomuto systému jsou dalším ovocem práce na tomto projektu, což je dobře, jelikož je Git skutečně mocný nástroj a v týmu se bez něj či jeho alternativ prakticky nelze obejít.

Během práce na projektu se náš tým setkal s mnohými problémy, které zahrnovaly jak špatnou práci s Gitem či problémy vzniklé nesprávným zacházením s dynamickou pamětí, tak i špatné rozvržení času. Skrze týmovou práci se nám však podařilo tyto problémy překonat a z chyb se také poučit (například právě v souvislosti s prací s Gitem). Díky tomu považujeme projekt za velmi přínosnou zkušenost a jsme rádi, že jsme mohli tuto práci absolvovat.