Compiler Assignment 0

Kaartik Bhushan
150313
kbhushan@iitk.ac.in

Rohit Kumar Bose
150596
rohitkb@iitk.ac.in

Siddharth Mittal
150718
sidm@iitk.ac.in

January 2018

T-Diagram

Table 1: T diagram indicating our tuple

ObjPascal	x86
Python	

Grammar

Col1	Col2
⟨ Goal⟩	⟨ Program⟩ ;
⟨ Program⟩	['PROGRAM' ⟨ Ident⟩ ['(' ⟨ IdentList⟩ ')'] ';'] ⟨ ProgramBlock⟩ '.' ;
⟨ ProgramBlock⟩	⟨ Block⟩ ;
⟨ Block⟩	[⟨ DeclSection⟩] ⟨ CompoundStmt⟩ ;
⟨ DeclSection⟩	(⟨ ConstSection⟩ | ⟨ TypeSection⟩ | ⟨ VarSection⟩ | ⟨ ProcedureDeclSection⟩)*;
⟨ CompoundStmt⟩	BEGIN' ⟨ StmtList⟩ 'END';
⟨ StmtList⟩	⟨ Statement⟩ (';' ⟨ Statement⟩)*;
⟨ Statement⟩	[⟨ SimpleStatement⟩ | ⟨ StructStmt⟩];
⟨ SimpleStatement⟩	⟨ Designator⟩ ['(' ⟨ ExprList⟩ ')'] | ⟨ Designator⟩ ':=' ⟨ Expression⟩ | 'INHERITED' | '(' ⟨ Expression⟩ ')'

⟨ StructStmt⟩	⟨ CompoundStmt⟩ | ⟨ ConditionalStmt⟩ | ⟨ LoopStmt⟩
⟨ ExprList⟩	⟨ Expression⟩ ('_' ⟨ Expression⟩)*;
⟨ CompoundStmt⟩	BEGIN' ⟨ StmtList⟩ 'END';
⟨ ConditionalStmt⟩	⟨ IfStmt⟩ | ⟨ CaseStmt⟩ ;
⟨ CaseStmt⟩	CASE' ⟨ Expression⟩ 'OF' ⟨ CaseSelector⟩ (';' ⟨ CaseSelector⟩)* ['ELSE' ⟨ Statement⟩] [';'] 'END';
⟨ CaseSelector⟩	⟨ CaseLabel⟩ ('_' ⟨ CaseLabel⟩)* ':' ⟨ Statement⟩ ;
⟨ CaseLabel⟩	⟨ ConstExpr⟩ ['..' ⟨ ConstExpr⟩];
⟨ IfStmt⟩	IF' ⟨ Expression⟩ 'THEN' ⟨ Statement⟩ ['ELSE' ⟨ Statement⟩];
⟨ LoopStmt⟩	⟨ RepeatStmt⟩ | ⟨ WhileStmt⟩
⟨ RepeatStmt⟩	REPEAT' ⟨ Statement⟩ 'UNTIL' ⟨ Expression⟩ ;
⟨ WhileStmt⟩	WHILE' ⟨ Expression⟩ 'DO' ⟨ Statement⟩ ;
⟨ Expression⟩	⟨ SimpleExpression⟩ [⟨ RelOp⟩ ⟨ SimpleExpression⟩]*;
⟨ SimpleExpression⟩	['+' | '-'] ⟨ Term⟩ [⟨ AddOp⟩ ⟨ Term⟩]*;
⟨ Term⟩	⟨ Factor⟩ [⟨ MulOp⟩ ⟨ Factor⟩]*;
⟨ Factor⟩	⟨ Designator⟩ ['(' ⟨ ExprList⟩ ')'] | '@' ⟨ Designator⟩ | ⟨ Number⟩ | ⟨ String⟩ | 'NIL' | '(' ⟨ Expression⟩ ')' | 'NOT' ⟨ Factor⟩ | 'INHERITED' [⟨ Designator⟩] | ⟨ TypeID⟩ '(' ⟨ Expression⟩ ')' | '(' ⟨ LambdaFunction ⟩ ')' ;
⟨ Type⟩	⟨ TypeID⟩ | ⟨ SimpleType⟩ | ⟨ PointerType⟩ | ⟨ StringType⟩ | ⟨ ProcedureType⟩ ;
⟨ SimpleType⟩	(⟨ OrdinalType⟩ | ⟨ RealType⟩);
⟨ PointerType⟩	^ ⟨ Ident⟩
⟨ StringType⟩	STRING' | 'STRING' '[' ⟨ ConstExpr⟩ ']';
⟨ ProcedureType⟩	(⟨ ProcedureHeading⟩ | ⟨ FunctionHeading⟩) ['OF' 'OBJECT'];
⟨ RelOp⟩	⟩ ' | '⟨ ' | '⟨ =' | '⟩ =' | '⟨ ⟩ ' | '=';
⟨ AddOp⟩	+' | '-' | 'OR' | 'XOR';
⟨ MulOp⟩	*' | '/' | 'DIV' | 'MOD' | 'AND' | 'SHL' | 'SHR';
⟨ Designator⟩	⟨ Ident⟩ [⟨ DesignatorSubElement⟩]*;
⟨ DesignatorSubElement⟩	.' ⟨ Ident⟩ | '[' ⟨ ExprList⟩ ']' | '' ';'
⟨ ConstSection⟩	CONST' (⟨ ConstantDecl⟩ ';')*;
⟨ ConstantDecl⟩	⟨ Ident⟩ ('=' ⟨ ConstExpr⟩ | ':' ⟨ TypeID⟩ '=' ⟨ TypedConstant⟩)
⟨ TypedConstant⟩	(⟨ ConstExpr⟩ | ⟨ ArrayConstant⟩)

⟨ Array⟩	ARRAY' ['[' ⟨ OrdinalType⟩ ['..' ⟨ OrdinalType⟩] ']'] 'OF' ⟨ TypeArray⟩
⟨ TypeArray⟩	[⟨ TypeID⟩ | ⟨ PointerType⟩]
⟨ ArrayConstant⟩	(' (⟨ TypedConstant⟩ [',' ⟨ TypedConstant⟩]*) ')';
⟨ ConstExpr⟩	? An expression which evaluates to a constant at compilation time ?;
⟨ Ident⟩	⟨ Identifier⟩ ;
⟨ Identifier⟩	⟨ AlphaChar⟩ (⟨ AlphaChar⟩ | ⟨ NumericChar⟩)*;
⟨ AlphaChar⟩	A'..'Z';
⟨ NumericChar⟩	0'..'9';
⟨ IdentList⟩	⟨ Ident⟩ [⟨ TypeArgs⟩] ('.' ⟨ Ident⟩ [⟨ TypeArgs⟩])*;
⟨ TypeArgs⟩	⟨ ' (⟨ TypeID⟩ | ⟨ String⟩) '⟩ ';
⟨ TypeID⟩	INTEGER' | 'REAL' | 'CHAR'
⟨ String⟩	⟨ Character⟩ * ;"
⟨ Character⟩	32..255;
⟨ OrdinalType⟩	INTEGER'
⟨ RealType⟩	DOUBLE'
⟨ TypeSection⟩	TYPE' (⟨ TypeDecl⟩ ';')*;
⟨ TypeDecl⟩	⟨ Ident⟩ '=' ['TYPE'](⟨ Type⟩ | ⟨ RestrictedType⟩);
⟨ RestrictedType⟩	⟨ ObjectType⟩ | ⟨ ClassType⟩
⟨ VarSection⟩	VAR' (⟨ VarDecl⟩ ';')*;
⟨ VarDecl⟩	⟨ IdentList⟩ ':' ⟨ Type⟩ [('ABSOLUTE' (⟨ Ident⟩ | ⟨ ConstExpr⟩)) | '=' ⟨ ConstExpr⟩]
⟨ ProcedureDeclSection⟩	⟨ ProcedureDecl⟩ | ⟨ FunctionDecl⟩ | ⟨ ConstructorDecl⟩ | ⟨ LambdaFunctionDecl ⟩ ;
⟨ ConstructorDecl⟩	⟨ ConstructorHeading⟩ ';' ⟨ Block⟩ ';';
⟨ ConstructorHeading⟩	CONSTRUCTOR' ⟨ Ident⟩ [⟨ FormalParameters⟩];
⟨ FunctionDecl⟩	⟨ FunctionHeading⟩ ';' ⟨ Block⟩ ';';
⟨ FunctionHeading⟩	FUNCTION' ⟨ Ident⟩ '(' [⟨ FormalParameters⟩] ')' ':' ⟨ SimpleType⟩
⟨ FormalParameters⟩	⟨ IdentList⟩
⟨ ProcedureDecl⟩	⟨ ProcedureHeading⟩ ';' ⟨ Block⟩ ';';
⟨ ProcedureHeading⟩	PROCEDURE' ⟨ Ident⟩ [⟨ FormalParameters⟩];
⟨ LambdaFunctionDecl ⟩	'LAMBDA' ⟨ Ident⟩ : ⟨ SimpleExpression⟩
⟨ LambdaFunction⟩	⟨ Ident⟩ '(' ⟨ ConstExpr⟩ ')'

⟨ ObjectType⟩	OBJECT' [⟨ ObjHeritage⟩] ([⟨ ObjVisibility⟩] [⟨ ObjTypeSection⟩] [⟨ ObjConstSection⟩] [⟨ ObjVarSection⟩] [⟨ ObjMethodList⟩])* 'END';
⟨ ObjVisibility⟩	['PUBLIC'];
⟨ ObjTypeSection⟩	⟨ TypeSection⟩ ;
⟨ ObjConstSection⟩	⟨ ConstSection⟩ ;
⟨ ObjVarSection⟩	⟨ VarSection⟩
⟨ ObjMethodList⟩	(⟨ ObjectMethodHeading⟩ (⟨ ObjectMethodHeading⟩ ';')*);
⟨ ObjectMethodHeading⟩	⟨ ProcedureHeading⟩ | ⟨ FunctionHeading⟩ ;
⟨ ClassType⟩	CLASS' [⟨ ClassHeritage⟩] ([⟨ ClassVisibility⟩] [⟨ ClassTypeSection⟩] [⟨ ClassConstSection⟩] [⟨ ClassVarSection⟩] [⟨ ClassMethodList⟩])* 'END';
⟨ ClassHeritage⟩	(' ⟨ IdentList⟩ ')';
⟨ ClassVisibility⟩	PUBLIC';
⟨ ClassTypeSection⟩	⟨ TypeSection⟩ ;
⟨ ClassConstSection⟩	⟨ ConstSection⟩ ;
⟨ ClassVarSection⟩	⟨ VarSection⟩ ;
⟨ ClassMethodList⟩	(⟨ ClassMethodHeading⟩ (';'⟨ ClassMethodHeading⟩)*);
⟨ ClassMethodHeading⟩	⟨ ProcedureHeading⟩ | ⟨ FunctionHeading⟩ ;
⟨ Input⟩	READ' | 'READLN' '(' ⟨ IdentList⟩ ')';
⟨ Output⟩	WRITE' | 'WRITELN' '(' ⟨ IdentList⟩ ')';

New Constructs Added

Lambda : A lambda expression is a function definition that is not bound to an identifier. These functions are ubiquitous in functional programming languages. Example code :

$$g = x : x * 2$$

Here, the variable g will store the value $2 * x$.

Rules Removed

- Try-except, raise and assembler statements
- Label, goto
- Records
- File I/O
- Units
- Private and protected visibility inside class
- Class variable section, field list and property section within class
- Structures
- Arrays having more than 2 dimensions
- Recursion with depth more than 2 level
- For loops
- Boolean data type
- Type parameters in type declarations
- Nested Case
- Deconstructor Declaration
- Operator declaration
- Restricted Identity List
- Directives
- Interface, Implement methods
- RTTI Attributes
- Sets
- Export statements
- Portability Directives
- Enum, subrange

Tools

We are going to use the python library **PLY**(Python Lex Yacc) for lexing/parsing.