Design Overview for Programming Language

Name: Denver Lacey
Student ID: 103306345

Summary of Program

Describe what you want the program to do… one or two paragraphs.
The program will be a tree walker interpreter for its own programming language. It will open and parse a given source file into an Abstract Syntax Tree (AST) which will then be executed by walking the tree.
The programming language will be simple dynamically typed object-oriented programming language similar to Python.

Include a sketch of sample output to illustrate your idea.
A sample of the language's syntax:

```
class Pos
        fn init(x, y)
                self.x = x
                self.y = y
        fn add(other)
                Pos(self.x + other.x, self.y + other.y)

var p1 = Pos(4.5, 2.1)
var p2 = Pos(7.6, 9.9)
var p3 = p1.add(p2)
print p3
```

The output of this program would be:

```
Pos(x: 12.1, y: 12)
```

Required Roles

Describe each of the classes and interfaces you will create using the following table (one per record).

Table 1: Interpreter class details

Responsibility	Type Details	Notes
Turn source code into tokens	Tokenizer	Pre-pass for parsing source code into AST
Parse tokens and generate AST	Parser	
Visits each node of AST and executes it	VM	VM is short for Virtual Machine. Will keep track of execution state
Interpret a source file	void Interpret(string filepath)	
Opens source file and reads it into a string	File-path string. source code string	reading in entire file into one string simplifies parsing

Table 2: Tokenizer class details

Responsibility	Type Details	Notes
Source code to tokenize	Source code string	
Record starting index of current token	Integer index that records beginning position of current token	
Record current tokens length	Integer representing current tokens length in ASCII characters	Sticking with ASCII to make parsing simple
Record indentation of current token	Integer recording indentation level for current token	indentation based syntax means we need to give each token an indentation level
Transforms source code into a list of tokens	List<Token> Tokenize(string source)	Accumulates tokens until reached EOF then returns resulting list
Skips irrelevant whitespace	void SkipWhitespace()	Gets called before actually processing characters so something like '1 + 2' is treated the same as '1+2' or '1 + 2'
Skip over indentation and record it	void ProcessIndentation()	Called at the beginning of each line to get indentation level for that line
Turns small chunk of source into one token	Token ProcessToken()	delegates to more specific processing methods based on first character
Turns small chunk of source into one number token	Token ProcessNumber()	Called by 'ProcessToken()' when first character is a digit
Turns small chunk of source into one string token	Token ProcessString()	Called by 'ProcessToken()' when first character is a double-quote

Tokenizer class continued

Responsibility	Type Details	Notes
Turns small chunk of source into one character token	Token ProcessCharacter()	Called by 'ProcessToken()' when first character is a single-quote
Turns small chunk of source into either an identifier or keyword token	Token ProcessIdentifierOrKeyword()	Called by 'ProcessToken()' when first character is a letter
Turns small chunk of source into one operator token	Token ProcessOperator()	Called by 'ProcessToken()' when first character is not a digit, double-quote, single-quote or letter

Table 3: Token struct details

Responsibility	Type Details	Notes
Represents the kind of token	TokenKind	By using a struct with a type tag we keep the complexity of what to do with each token in what context can be kept in the Parser while also keeping that complexity to a minimum
Represents token's indentation level	Integer	Will be used in parsing to identify when a block of code ends because of indentation-based syntax
Copy of source code of the token	String	Will be used to get identifiers and for error messages
References Literal Value for Literal tokens	Value. Used in the runtime as base class for all value types	

Table 4: Parser class details

Responsibility	Type Details	Notes
Gives parser access to generated tokens	List of Tokens	This list is returned from Tokenizer.Tokenize(). Having it in a list makes it easy to iterate and peek ahead
Flags when an error occurred	Boolean flag. true if error has occurred otherwise false	
Represents current place in list of tokens	Integer index used to access current and previous token in the list	
Current Root node of generated AST	An AST node	This is used to access previously generated AST to make it the child of a parent expression
Parses list of tokens into a list of AST nodes	List<IAST> Parse(List<Token> tokens)	
Increments the current index integer	void Advance()	Will be wrapped in more specific methods
Checks if current token is of a given kind	bool Check(TokenKind)	

Checks current token and advances if given kind	bool Next(TokenKind)	Basically a wrapper for:
		if (Check(kind)) {
		  Advance();
		  // ...
		}

Similar to Next except if will throw an exception if Check fails	void Expect(TokenKind, errMsg, args)	The errMsg and args are used to generate an error message
Attempts to parse a declaration before a statement	void ParseDeclaration()	A declaration for a variable, class or function

Parser class continued

Responsibility	Type Details	Notes
Attempts to parse a statement before an expression	void ParseStatement()	An If statement. while and for loops etc. are statements
Attempts to parse an expression. Errors if invalid syntax	void ParseExpression()	Wrapper for ParsePrecedence(Precedence.Assignment)
Parses an expression at a given precedence level	void ParsePrecedence(Precedence)	Main function for parsing expressions. Calls more specific parsing functions based on current token

Table 5: VM class details

Responsibility	Type Details	Notes
Reference to Parent VM	VM	Each VM encapsulates its own scope of variables and constants. To access values in different scopes we lookup through the parents
Reference to Global VM	VM	This is where all top-level code is executed and declarations are stored. Top-level declarations are accessible anywhere in the program
Map of identifiers to variables	Dictionary<string, Value>	Used to access and reassign named values in the program
Map of identifiers to constants	Dictionary<string, Value>	Used to access constant values like classes and functions in the program
Used to set VM's parent and global references	void SetEnvironment(VM, VM)	

VM class continued

Responsibility	Type Details	Notes
Begins execution of AST	void Execute(List<IAST> program)	Calls the 'Execute()' method for each root node in the list

Table 6: IAST interface details

Responsibility	Type Details	Notes
Executes node with VM as its environment and returns resulting value to parent nodes	Value Execute(VM)	Every type of AST node returns a value. A node usually calls its children's 'Execute()' method and works with the result values

Table 7: Unary abstract class details, implements IAST

Responsibility	Type Details	Notes
Reference to subexpression node	IAST	In an expression like '!false'. 'false' would be the subexpression
Executes the unary operation	Value Execute(VM)	Is abstract. Overriden in derived classes.

Table 8: Binary abstract class details, implements IAST

Responsibility	Type Details	Notes
Reference to left hand side node	IAST	In an expression like '1 + 2'. '1' would be the left hand side node
Reference to right hand side node	IAST	In an expression like '1 + 2'. '2' would be the right hand side node
Executes the binary operation	Value Execute(VM)	Is abstract. Overriden in derived classes

Table 9: Value abstract class details

Responsibility	Type Details	Notes
Flag for what type of data is stored in the value	ValueType	Used to assert that values are the expected type when executing AST nodes
Checks if two Values are of the same type	bool TypesMatch(Value, Value)	Is a static method
Checks if two values are equal. Assumes values are of same type	bool UncheckedEqual(Value)	Is abstract
Checks if two values are equal	bool Equal(Value)	Is virtual. Calls TypesMatch before calling UncheckedEqual
Checks if value is nil	bool IsNil()	Is virtual. Default implementation returns false. NilValue overrides and returns true

Table 10: TokenKind enum details

Value	Notes
EOF	Denotes the end of the file
Error	Used to pass errors found in the tokenizer to the parser
EndStatement	Denotes the end of a statement
Comma	Denotes a comma
DelimOpenParenthesis	Denotes a '('
DelimCloseParenthesis	Denotes a ')'
DelimOpenBracket	Denotes a '['
DelimCloseBracket	Denotes a ']'
LiteralNil	Denotes 'nil'
LiteralBoolean	Denotes 'true' or 'false'
LiteralNumber	Denotes a number like '12'
LiteralString	Denotes text like '"Hello"'
LiteralChar	Denotes a character like 'A'
KeywordVar	Denotes keyword 'var'
KeywordFn	Denotes keyword 'fn'
KeywordClass	Denotes keyword 'class'
KeywordIf	Denotes keyword 'if'
KeywordElif	Denotes keyword 'elif'
KeywordElse	Denotes keyword 'else'
KeywordWhile	Denotes keyword 'while'
KeywordFor	Denotes keyword 'for'
KeywordIn	Denotes keyword 'in'
KeywordBreak	Denotes keyword 'break'
KeywordContinue	Denotes keyword 'continue'
KeywordReturn	Denotes keyword 'return'

TokenKind enum continued

Value	Notes
OpBang	Denotes a '!'
OpPlus	Denotes a '+'
OpDash	Denotes a '-'
OpStart	Denotes a '*'
OpSlash	Denotes a '/'
OpEqual	Denotes a '='
OpDoubleEqual	Denotes a '=='
OpBangEqual	Denotes a '!='
OpOr	Denotes a 'or'
OpAnd	Denotes an 'and'
OpLeftAngle	Denotes a '<'
OpRightAngle	Denotes a '>'
OpDot	Denotes a '.'
OpDoubleDot	Denotes a '..'
OpDoubleDotEqual	Denotes a '..='

Table 11: Precedence enum details

Value	Notes
None	Precedence level of things that aren't part of expressions
Assignment	Precedence level of assignment operators
Or	Precedence level of 'or' operator
And	Precedence level of 'and' operator
Equality	Precedence level of equality operators
Comparison	Precedence level of comparison operators

Precedence enum continued

Value	Notes
Term	Precedence level of '+' and '-' operators
Factor	Precedence level of '*' and '/' operators
Unary	Precedence level of unary operators
Call	Precedence level of '.' operator, function calls and list subscript
Primary	Parsing algorithm increments precedence level as it goes. this is used to parsing without having to check precedence level

Table 12: Precedence enum details

Value	Notes
Unknown	Used as a default that is no type
Nil	Valid nothing type
Boolean	Runtime boolean type
Number	Runtime number type
String	Runtime string type
Char	Runtime character type
Lambda	Runtime function type. Also covers anonymous functions
List	Runtime list type
Class	Runtime class type
Instance	Runtime instance of a class type
Range	A range like '1..10'