Expression Language

CE305 Assignment 2 by Henry Lewis (Student ID: 1703977)

Main Features of the Compiler

- Can compile code from external text files or from code typed into the console. Compiling to .py files
- User can define the names of the python file saved to.
- Language supports variable decleration and assignment, along with type checking.
- Language supports all basic arithmetic & boolean operations on variables.

Basic Language Syntax

- All statements should be ended with ';'
- Language has support for both Integers and Float values. Conversion between these is implicitly done.
- Statements such as "WHILE" and "IF", "ELIF", "ELSE" are structured with conditions in '(' ')' brackets, and code to be executed inside '{' '}' curly braces.
- `COMPILE` must be called when typing in code to be compiled in the console. This is so the compiler knows the program has been fully typed out.
  - It is not required for program-code loaded from file.

Variables:

- There are three types of variable supported in my language:
  - INT
  - FLOAT
  - BOOL
- Boolean values are either `True` or `False`.
- Integer values are simply any whole positive / negative number.
- Float values are any positive/ negative number with some decimal value attatched.
- Variables must be declared before use.
- All variable names must be in lower-case.
- Variables are declared
  i.e: `TYPE variable-name;`.
- Declerations can just specify type, in which case a null value is assigned, or they may be fully declared & assigned in a single statement.

i.e. `INT a = 3;`

- Assignments are done after a variable is declared, using '='
  i.e: `a = 3;`
- Integer and Float values can be implicitly converted in assignments, but booleans cannot.

Assigning expressions

- Variables can be assigned expressions, i.e.
  `BOOL b = (4 < 5);`
- Expressions assigned need to be surrounded by brackets.

Functions

- Functions are declared with a name, variables for that function inside brackets, then all content of the function within a pair of curly braces.
  i.e: `DEF main (INT apple) { *some code* }`
- Arguments supplied to the function may be omitted, if none are required.
- The `RETURN` statement may be used to return values from the function. This must be the last line in the function if used.

Operations:

Arithmetic Operations

- Numbers & number-type variables can have the following operations applied:
  - `+` - Sum
  - `-` - Subtract
  - `*` - Multiply
  - `\\` - Divide
  - `%` - Modulo
- Operations can be strung together, i.e:
  `4 + 5 + 6`
- As mentioned above, operations can be used as variables themselves - but need to be encapsulated in '(' and ')' brackets.

Boolean Logic

- Number & number-type variables can be compared using the following operators:
  - `>` - Greater Than
  - `<` - Less Than

  - `>=` - Greater Than or Equal To
  - `<=` - Less Than or Equal To

- All variable types can be compared using the following:
  - `==` - Equal To
  - `!=` - Not Equal To
- These comparisons are similar to my arithmetic expressions, in that they can be chained one after the other i.e:
  ```
  4 > 5 > 6
  ```
- They can also be treated as boolean variables when encapsulated in '(' and ')' brackets.
- Logical comparisons can also be applied to boolean-type variables (including aformentioned arithmetic comparators)
  - `&` - AND
  - `|` - OR
- These can also be chained together or treated as a variable when encapsulated in '(' and ')' brackets.

Expressions

While Loops

- While loops are structured in the following manner:
  ```
  WHILE (variable or condition) { expression(s); };
  ```
- If you are using basic boolean values i.e: 'True' the '(' and ')' brackets may be omitted.
- No indentation is required as any statements are encapsulated within the '{' and '}' curly-braces, as in Java.

If-Else Statements

- If-Else statements are structured in the following manner:
  ```
  IF (variable or condition) { expression(s); };
  ELIF (variable or condition) { expression(s); };
  ELSE { expression(s); };
  ```
- Both `ELIF` and `ELSE` statements are not required after an `IF`
- There can be many `ELIF` statements but only one `ELSE`
- Like with While-loops, no indentation is required since all expressions / statements end with `;` and are encapsulated by the '{' and '}' curly-braces.

Language Tokens

My language identifies the following Tokens:

- **LineEnd** - this is a `;` character, used to denote the end of statements.
- **Assign** - this is a `=` used to denote when a variable is being assigned
- **Open_Bracket** - this is a `(` character used to encapsulate expressions for variables
- **Close_Bracket** - this is a `)` character
- **Open_Brace** - this is a `{` character, used to start encapsulation for nesting
- **Close_Brace** - this is a `}` character, used to end encapsulation for nesting
- **IF** - `IF` used to denote beginning of an IF statement
- **ELIF** - `ELIF` used to denote beginning of an ELSE-IF statement
- **ELSE** - `ELSE` used to denote beginning of an ELSE statement
- **WHILE** - `WHILE` used to denote beginning of a While-Loop
- **PRINT** - `PRINT` used to define a Print-Statement
- **COMPILE** - `COMPILE` the phrase used to mark end of typed code when compiling console-typed code.

- **Char** - `[a-z]` this is any set / string of lower case characters, excluding spaces.
- **Int** - any string of number characters `[0-9]`
- **Float** - this is any string of number character followed by a '.' and at least one number character.
  i.e. `5.5`
- **Bool** - Represents a True / False value, which have a capital first-letter
  i.e: `True` or `False`

- **Sum** - a `+` character
- **Subtract** - a `-` character
- **Multiply** - a `*` character
- **Divide** - a `\` character
- **Modulo** - a `%` character

- **Function** - `DEF` used to define a function
- **Return** - `RETURN` used to return a value from a function

- **DecInt** - `INT` used in declaring an integer-type variable
- **DecFloat** - `FLOAT` used in declaring a float-type variable
- **DecBool** - `BOOL` used in declaring a boolean-type variable

- **ET** - `==` Equal-To symbol

- **NET** - `!=` Not-Equal-To symbol
- **GT** - `>` Greater-Than symbol
- **LT** - `<` Less-Than symbol
- **GTET** - `>=` Greater-Than or Equal-To symbol
- **LTET** - `<=` Less-Than or Equal-To symbol
- **AND** - `&` Logical-AND symbol (*ampersand*)
- **OR** - `|` Logical-OR symbol (*pipe*)

- **Whitespace** - any blank space or character, this is ignored by the program & removed in pretty-print.
  i.e. (4 + 5)