Lab 2 - wideLang

Lexic.txt

Alphabet

- Uppercase (A-Z) or lowercase letters (a-z) of the English alphabet
- Underline character ‘_’
- Decimal digits

Special symbols

- arithmetic operators: ‘plus’, ‘minus’, times’, ‘dividedBy’, ‘takes’ (:=), ‘modulus’
- relational operators: ‘smallerThan’, ‘smallerThanOrEqualTo’, ‘equals’, ‘greaterThan’, ‘greaterThanOrEqualTo’
- boolean operators: ‘and’, ‘or’
- separators: space, ‘.’, {}, [], ?, ‘”’

```
separators = " " | "." | "{" | "}" | "[" | "]" | "?" | "\""
```

- reserved words:

```
array, integer, boolean, string
fun, return, with, as
plus, minus, times, dividedBy, takes, modulus
smallerThan, smallerThanOrEqualTo, equals, greaterThan, greaterT
inCase, otherwise, for, every, that, do, while
scream, chill, intimidateTheUserForInputInTheConsole
StartOfRunningProgram
absolutelyTrue, notTrue
```

Identifiers

- any combinations of letters (at least one) and digits that may or may not start with an underscore with the rule:

```
identifier = { "_" } letter {(letter | digit)}
letter = “A” | “B” | … | “Z” | “a” | “b” | … “z”
digit = “0” | “1” | … | “9”
```

Constants

- integer

```
integer = “0” | [”+” | “-” ] nonzerodigit {"0" | nonzerodigit }
nonzerodigit = "1" | ... | "9"
```

- boolean

```
boolean = "absolutelyTrue" | "notTrue"
```

- string

```
character = letter | "_" | digit | operator | separator
characters = character {character}
string = "\"" {characters} "\""
```

Token.in

Reserved words

```
array, integer, boolean, string
fun, return, with, as
plus, minus, times, dividedBy, takes, modulus
smallerThan, smallerThanOrEqualTo, equals, greaterThan, greater
```

```
inCase, otherwise, for, every, that, do, while
scream, chill, intimidateTheUserForInputInTheConsole
StartOfRunningProgram
absolutelyTrue, notTrue
```

Syntax.in

type = integer | boolean | string | array

integer = “0” | [”+” | “-”] nonzerodigit {"0" | nonzerodigit }

nonzerodigit = "1" | ... | "9"

identifier = { "_" } letter {(letter | digit)}

letter = “A” | “B” | … | “Z” | “a” | “b” | … “z”

digit = “0” | “1” | … | “9”

factor = expression | identifier | constant

term operator = “plus” | “minus” | “times” | “dividedBy” | “modulus”

term = factor {(term operator factor)}

expression operator = “plus” | “minus” | “times” | “dividedBy” | “modulus” | “takes”

relational operator = “smallerThan” | “smallerThanOrEqualTo” | “equals” | |greaterThan” | “ greaterThanOrEqualTo” | “and” | “or”

expression = term {(expression operator term)} | ternary expression

condition = expression {(expression operator expression)}

ternary expression = condition ? expression : expression

declaration statement = identifier “as” type [”takes” expression] “.”

assignment statement = identifier “takes” expression “.”

input statement = “scream” expression

output statement = “intimidateTheUserForInputInTheConsole” with expression

if statement = “inCase” condition ? statement list [”otherwise” statement list]

while statement = “while” condition “do” statement list

for statement = “for” “every” statement “that” condition” “do” statement “:”

statement = declaration statement | assignment statement | input statement | output statement | if statement

statement list = statement [statement list]