Crux Language Specification

Lexical Semantics

A program written in Crux consists of a sequence of lexemes, each of which can be classified as a kind of token. The kinds of tokens, and the rules that govern their appearance are as follows:

- As in Java, comments begin with a double forward slash and continue until the end of the line on which they appear. Comments should be ignored by the scanner, because they do not constitute a lexeme.
- Whitespace should be ignored, as it does not constitute a lexeme.
- The following words are reserved types, but are recognized as IDENTIFIER tokens: void, bool, int.
- The following words are reserved keywords:

Name	Lexeme
AND	&&
OR	||
NOT	!
IF	if
ELSE	else
LOOP	loop
CONTINUE	continue
BREAK	break
TRUE	true
FALSE	false
RETURN	return

- The following character sequences have special meaning:

Name	Lexeme
OPEN_PAREN	(
CLOSE_PAREN	)
OPEN_BRACE	{
CLOSE_BRACE	}
OPEN_BRACKET	[
CLOSE_BRACKET	]
ADD	+
SUB	-
MUL	*
DIV	/
GREATER_EQUAL	>=
LESSER_EQUAL	<=
NOT_EQUAL	!=
EQUAL	==
GREATER_THAN	>
LESS_THAN	<
ASSIGN	=
COMMA	,
SEMICOLON	;

- The following patterns are reserved value literals:

Name	LexemePattern
INTEGER	`digit {digit}`
IDENTIFIER	`("_" | letter) { "_" | letter | digit }`

where

```
digit := "0" | "1" | ... | "9" .
lowercase-letter := "a" | "b" | ... | "z" .
uppercase-letter := "A" | "B" | ... | "Z" .
letter := lowercase-letter | uppercase-letter .
```

- The following special circumstances generate special tokens:

Name	Circumstance
ERROR	Any character sequence not otherwise reserved. For example, a "!" not followed by an "=".
EOF	The end-of-file marker.

Crux Grammar

The crux grammar is given in Wirth Syntax Notation .

```
literal := INTEGER | TRUE | FALSE .

designator := IDENTIFIER [ "[" expression0 "]" ] .
type := IDENTIFIER .

op0 := ">=" | "<=" | "!=" | "==" | ">" | "<" .
op1 := "+" | "-" | "||" .
op2 := "*" | "/" | "&&" .

expression0 := expression1 [ op0 expression1 ] .
expression1 := expression2
       | expression1 op1 expression2 .
expression2 := expression3
       | expression2 op2 expression3 .
expression3 := "!" expression3
       | "(" expression0 ")"
       | designator
       | call-expression
       | literal .
call-expression := IDENTIFIER "(" expression-list ")" .
expression-list := [ expression0 { "," expression0 } ] .

parameter := type IDENTIFIER .
parameter-list := [ parameter { "," parameter } ] .

variable-declaration := type IDENTIFIER ";" .
array-declaration := type IDENTIFIER "[" INTEGER "]" ";" .
function-definition := type IDENTIFIER "(" parameter-list ")" statement-block .
declaration := variable-declaration | array-declaration | function-definition .
declaration-list := { declaration } .

assignment-statement := designator "=" expression0 ";" .
call-statement := call-expression ";" .
if-statement := "if" expression0 statement-block [ "else" statement-block ] .
loop-statement := "loop" statement-block .
break-statement := "break" ";" .
continue-statement := "continue" ";" .
return-statement := "return" expression0 ";" .
statement := variable-declaration
           | call-statement
           | assignment-statement
           | if-statement
           | loop-statement
           | break-statement
           | continue-statement
           | return-statement .
statement-list := { statement } .
statement-block := "{" statement-list "}" .

program := declaration-list EOF .
```

Pre-defined Functions

- `int readInt()` - Prompts the user for an integer.
- `int readChar()` - Reads a character as an integer.
- `void printBool(bool arg)` - Prints a bool value to the screen.
- `void printInt(int arg)` - Prints an integer value to the screen.
- `void printChar(int arg)` - Prints an integer value as an ASCII character to the screen.
- `void println()` - Prints newline character to the screen.

Runtime Constraints

All valid crux programs have one function with the signature: `void main()`. This function represents the starting point of the crux program.

Symbol Semantics

- An identifier must be declared before use. Note that this rule means Crux does not support mutual recursion, but it does support direct recursion.
- Identifier lookup is based on name only (not name and type).
- Only unique names may exist within any one scope.
- Symbols in an inner scope shadow symbols in outer scopes with the same name. Crux offers no syntax for accessing names in an outer scope.
- Each scope (roughly) corresponds to a set of matching curly braces.
- Function parameters are scoped with the function body.

Type Semantics

- Crux has the following predefined types: `void, bool, int`.
- The relation operators (GreaterThan, LesserThan, GreaterEqual, LesserEqual, NotEqual, Equal) result in a boolean value.
- The boolean logic operations (&&, ||, !) can only operate on booleans.
- Mathematical operators (Add, Sub, Mul, Div) shall operate only on ints.
- A function with the void return type does not necessarily have to have a return statement.
- A function with any return type other than void must have all possible code paths return a value.
- The return value of a function must have the same type as that specified by the function declaration.
- A function is not allowed to have a void (or other erroneous) type for an argument.