The programming language definition exists in two parts. The first is in the Actions.cfg/pre-parser where the symbol table gets loaded. The second is in the symbol table/parser where the program is actually executed. Essentially the way in which the symbol table is loaded and then executed defines how the programming language works.

Loading the symbol table:

Each entry in the actions.cfg file has four fields: command, action, link and syntax
The command is a name that can be used to identify the command in a string
The action defines which actions are taken during pre-parsing and parsing.
The link is used in preprocessing only to identify corresponding code groups i.e. if related to elseif, else, and eif. this allows for the preprocessor to correctly load the symbol table even with nested loops/ifs using a nested count.
The syntax defines how each command is used as well as identifying parameters for the command.

Actions: in-depth look

Within the actions.cfg file the syntax's for all of the languages functionality are defined. Being this is a relatively simple language the actions taken for any given syntax are broken down into five categories: declare, update, evaluate, marker, and goto. Every syntax falls into one of these categories.

declare:

pre-parse: ignore declares during pre-parse as variables may be initialized on the fly with dynamic values.

parse: create an entry in the symbol table of the command type with name <name> and value <value>. Only supported types will be allowed. For this project this means ints and floats, but functionality for other types could be added. If a name already exists in the symbol table it should be overwritten with the new value.

update:

pre-parse: updates are ignored during pre-parsing as they do not require an entry in the symbol table parse: during parsing updates will lookup <name> and changes its value in the symbol table to <value>

evaluate:

pre-parse: depending on the command each evaluate will have a different pre-parse consequence. But each evaluate will have some resulting entry in the symbol table.

parse: during parsing commands identified as evaluate will have their <condition> evaluated. In the case of positive outcome, execution continues, with negative outcome lookup to the symbol table will be made to find the target of the next executable line of code. i.e. for if statements if the <condition> results in true, then the next line will be executed, if the condition results in <false> a lookup to find the next else or eif line# in the symbol table will be made. In the case of the for loop, the increment will be done following the evaluate prior to the next line of code being executed upon position results from the <condition>

marker:

pre-parse: marker commands will have an entry in the symbol table made with their corresponding index number (line#)

parse: markers are ignored during parsing as they serve only as a goto mark

goto:

pre-parse: goto commands will have an entry in the symbol table made with their target line#

parse: the program pointer will be moved to the index in the program array corresponding with the goto target retrieved from the symbol table. Care must be taken when exiting loops and conditions such that they exit beyond their corresponding ends otherwise their end markers will loop them back to their start endlessly

The current actions.cfg file:

```
#command:   action:     link:             syntax:
int         declare     nill              int <name>=<value>
float       declare     nill              float <name>=<value>
=           update      nill              <name>=<value>
if          evaluate    [eif|elseif|else] if(<condition>)
eif         marker      nill              eif
elseif      evaluate    [eif|elseif|else] elseif(<condition>)
else        marker      nill              else
for         evaluate    efor              for(<name>;<condition>;<value>)
efor        goto        nill              efor
do          goto        nill              efor
edo         evaluate    do                edo(<condition>)
while       evaluate    ewhile            while(<condition>)
ewhile      goto        nill              ewhile
```

Parsing using the symbol table:

Once loaded the symbol table will use its entries as the means by which program execution will be controlled. Lines will be rereferenced by the actions table. Anything matching actions.cfg syntaxs will be processed using a combinition of the syntaxs action.cfg action and the previously created symbol table entry. The symbol table has four fields for each entry: name, value, location and count.

For each command name, an the value expected to be loaded by the pre-parser are as follows:

name:	**value:**
int	int value
float	float value
if	next else line# or eif line# if no else present
eif	if line#
elseif	next else line# or eif if last else statement
else	next else line# or eif if last else statement
for	efor line#
efor	for line#
do	edo line#
edo	do line#
while	ewhile line#
ewhile	while line#

The location field for any given command in the symbol table corresponds to the line number of th command found in the actual program.

Each command group i.e variable declare, if/else/eif, for/efor will be identified by a the count field. This allows ifs and loops to correctly reference their subsequent next-steps. These could be the end or beginning of the loop, or the next else if in an if statement.

Notes:

As the programming language currently exists primaraly as the comments collected from the actions.cfg and the parser/pre-parser, it will be important that changes in functionality be recognized in this document as a means to keep the defnintion somwhat centralized.