Source Code Documentation

Classes:

`class switch(object)`

Aids in creating a switch-case syntax using Python.

Functions:

`case(*args)`

Aids in creating a switch-case syntax using Python.

`getChar()`

Reads a character and identifies its character class.

`addChar()`

Append the next character to the lexeme string.

`lookup(ch)`

Classifies character when its class is unknown.

`getNoneBlank()`

Ignores all whitespace.

`lex()`

Assigns a value to `nextToken` depending on the value of `lexeme`.

`A()`

Parses a declaration line.

`expr()`

Parses either `[expr][+|-][term]` or `[term]`.

`term()`

Parses either `[term][*|/|%][factor]` or `[factor]`.

`factor()`

Parses either `([expr])` or `[var|int_lit]`.

`expr2()`

Similar function as `expr()` but modified to aid in generating translated code.

`term2()`

Similar function as `term()` but modified to aid in generating translated code.

`factor2()`

Similar function as `factor()` but modified to aid in generating translated code.

`program()`

Directs general parsing of the entire code. Returns an error if end of file is reached prematurely.

`outsideprog()`

Directs parsing when the end of file is not the end of program (e.g. blank lines at the end of code).

`func()`

Parses a `#function` line for function declaration. Form should be `[var]()`, otherwise error. Calls `block1()` afterwards.

`block1()`

Handles the declaration block, i.e., declaration of integers and floats. Calls `block()` afterwards.

`declare()`

Handles parsing of variable declaration. For integers, form should be `[var][assign_op]([+|-])[int_lit]`. For floats, form should be `[var][assign_op]([+|-])[int_lit][float_pt][int_lit]`.

`constant()`

Parses a constant during declaration of variables. Should be of the form `[+/-][int_lit]` or `[int_lit]`.

`param()`

Parses a parameter that must be of the form `[var|int_lit]`. Calls `param1()` if parsing of parameters is not yet done (expects `[comma|right_bracket]` afterwards).

`param1()`

Parses commas to aid parameter parsing. Calls `param()` if parsing of parameters is not yet done (expects `[var|int_lit]` afterwards).

`block()`

Parses a block. Calls `function()` to parse the current line and calls `block()` again to 'end' the parsing in that line and move on to the next one.

`function()`

Classifies a line and its function depending on its start. Calls the corresponding parser.

`print_out()`

Parses a `#print` statement. Must be of the form `[quote]#end[EOL]`, otherwise returns an error.

`read_in()`

Parses a **#**`read` statement. Must be of the form `[var]#end[EOL]`, otherwise returns an error.

`strexpr()`

Parses a string. Must be of the form "`[var]`", otherwise returns an error.

`comment_out()`

Parses a comment. Must be of the form `?[var|int_lit]?`, otherwise returns an error.

`call_function()`

Parses a call to a function. Must be of the form `[var]()`, otherwise returns an error.

`if_cont()`

Parses a #`if` condition. Must be of the form `([var][rel_op][var|int_lit])`, otherwise returns an error.

`elif_cont()`

Parses a #`elif` condition. Must be of the form `([var][rel_op][var|int_lit])`, otherwise returns an error.

`else_cont()`

Parses the succeeding block if the line is of the form `#else[EOL]`, otherwise returns an error.

`for_cont()`

Parses a #`for` line. Must be of the form `([var][assign_op][var|int_lit];[var][rel_op][var|int_lit];[var][++|--])`, otherwise returns an error.

`cond()`

Parses a general condition statement. Must be of the form `([var][rel_op][var|int_lit])`, otherwise returns an error.

`deletecontent ( pfile )`

Re-initializes the translated file.

`error()`

Parses the hash file until EOF.

`start()`

The main program. Initializes global variables and opens files for parsing. Calls `program()`.