Phase 1. Scanner/Screener

The Compildres

Arthur Margulies

Harold Treen

Paul Wells

February 4, 2015

**Modifications**

Three files were changed in this phase: stdIdentifiers, scanner.ssl and parser.pt.

**stdIdentifiers**

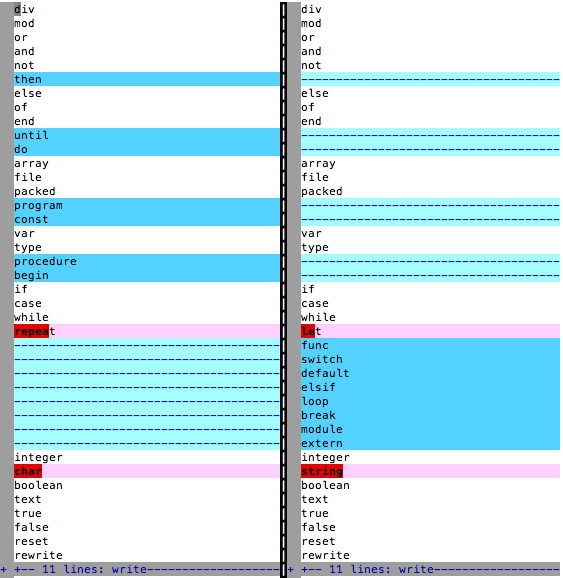
The file stdIdentifiers is a list of the keywords used in the language. We simply removed the keywords that were in PT Pascal and will not be in Drift:

*then, until, do, program, const, procedure, begin, repeat*

and added the keywords that will be new for Drift:

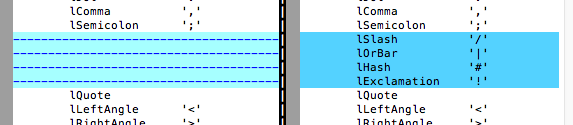
*let, func, switch, default, elsif, loop, break, module, extern*

We also replaced the keyword *char* with the keyword *string*.

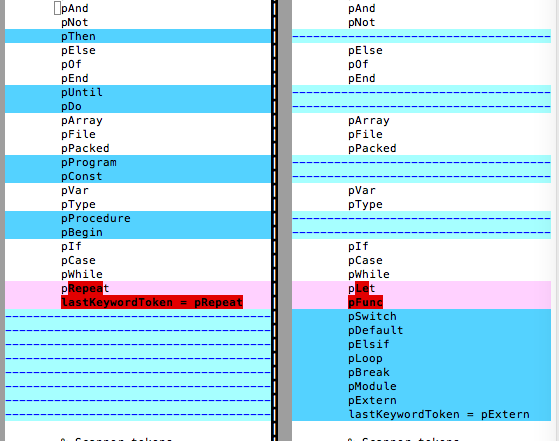


**scan.ssl**

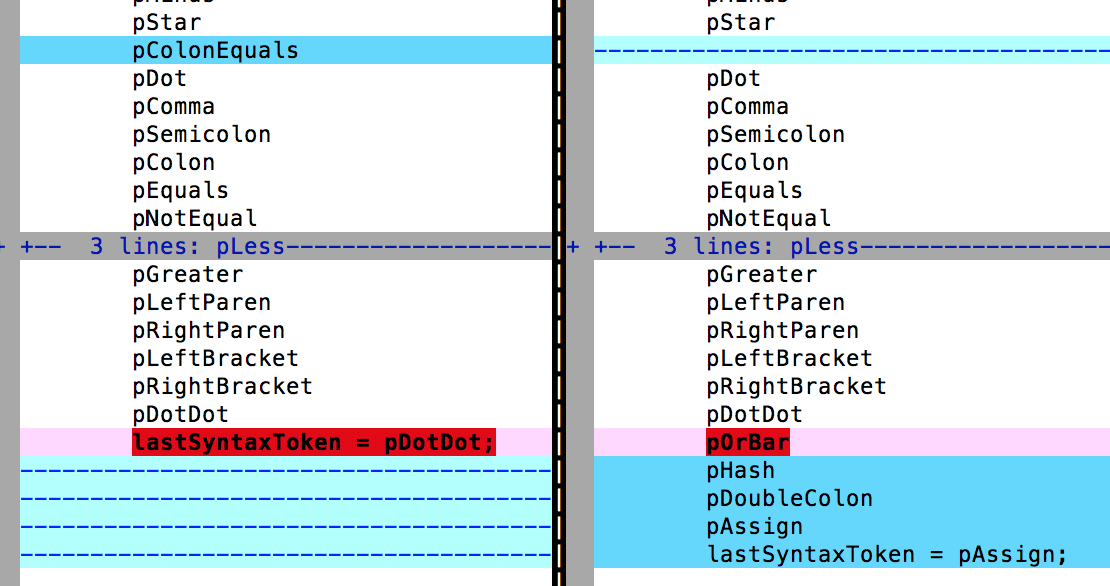
The first change we made to scan.ssl was adding character classes. We added the classes lSlash, lOrBar, lHash and lExclamation. We had to add these because PT Pascal did not use these characters at all. The symbols ‘=’ and ‘:’ for example were used already so even though we’ll be using them to make new syntax tokens, their character classes already exist so we didn’t need to add them.



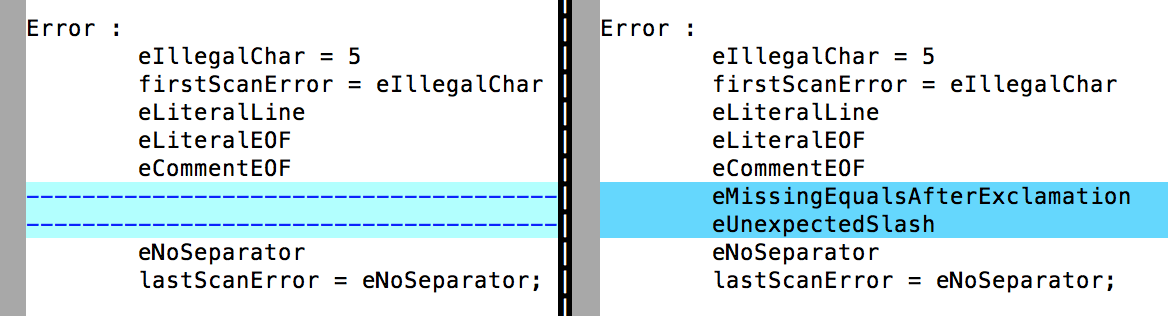
Next we updated the syntax tokens. We removed the tokens for the old keywords and added tokens for the new keywords.



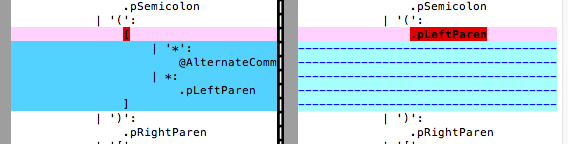
Next we added new syntax tokens for the new character classes and the new symbol ‘::’. We removed pColonEquals since ‘:=’ is being removed from the language. The new syntax token for assignment (using ‘=’) is called pAssign. Even though the symbols for pEquals and pNotEquals are changing we did not have to change the character classes.



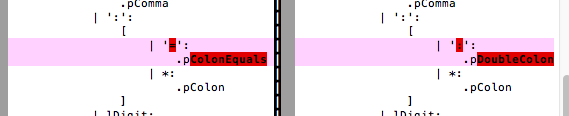
We added a new error statement called ‘eMissingEqualsAfterExclamation’. This is for the case when a ‘!’ is not followed by a ‘=’ since ‘!=’ is valid drift code but a single ‘!’ is not. We also added an error statement called ‘eUnexpectedSlash’ for when a slash is not followed by another slash.



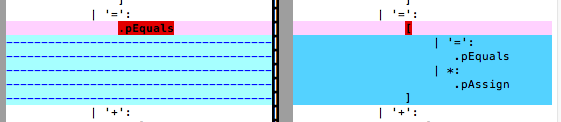
Next we changed the logic of the Scanner. We’ll list the changes we made in order of top to bottom in the code. First, since ‘(\*’ is no longer used to start a comment we removed the choice operator and simply emitted a pLeftParen.



In Drift, when a ‘:’ is encountered it is either a pColon (same as PT) or it’s a pDoubleColon (in PT it was pColonEquals). We changed the logic to reflect this.



In PT, when a ‘=’ was encountered it could only be the equality comparison operator ‘=’ but in Drift it is either the comparison operator ‘==’ or the assignment operator ‘=’. We made the changes below to reflect this.



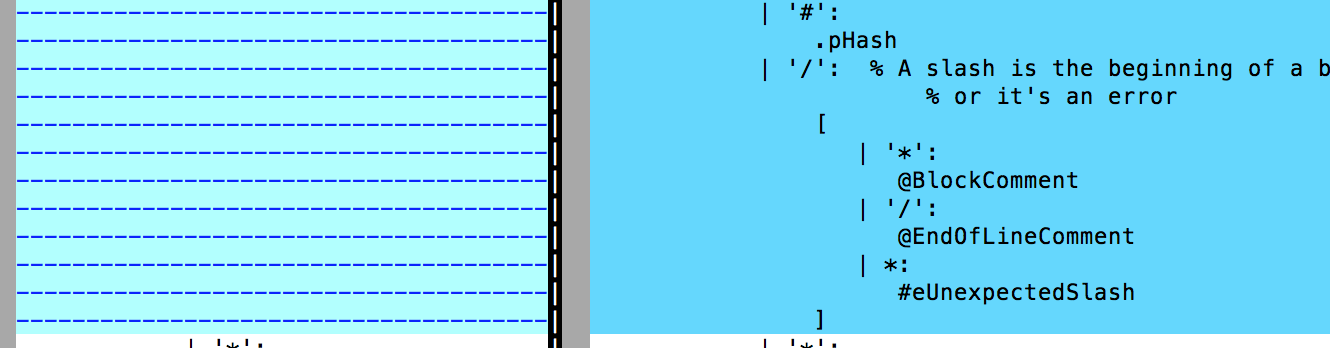
We added a choice to handle ‘!’ symbols. If it’s followed by a ‘=’ then it emits pNotEqual. Otherwise, it throws an error since a lone exclamation is not a valid Drift symbol.



We removed the ‘{‘ symbol since it is no longer used and added choices for ‘|’ and ‘#’ which simply emit their respective character classes.



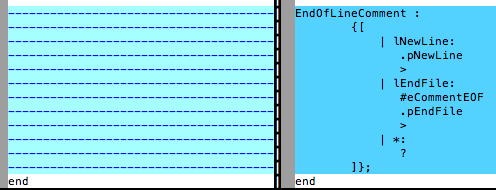
The ‘/’ character is now used to either start a block comment with ‘/\*’ or a line comment with ‘//’. Otherwise an error is emitted.



The block comments in Drift behave exactly like both versions of block comments in PT other then using a different symbol. We deleted one of the versions in PT and then adapted the other to use the /\* syntax.



Finally, we added a rule for end of line comments ‘//’. Once a double slash has been encountered then skip everything except for new lines and end of file characters. If it’s a new line character then emit pNewLine and exit the loop. If it’s an end of file character then throw an error, emit pEndFile and exit the loop.



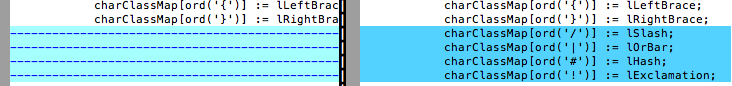
**parser.pt**

Most of the changes to parser.pt involved updating the integer values associated with the character classes, syntax tokens and error codes. These values were generated by running ‘ssl scan.ssl’ which produced the file scan.def. The numbers from scan.def were copied and pasted into parser.pt. The code diffs for these changes are emitted because they are relatively straightforward but long.

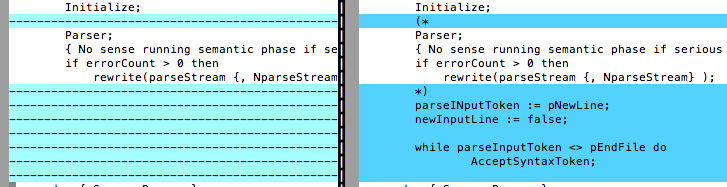
The identifier used for quotes was changed from double quotes to single quotes.

Macintosh HD:Users:paulwells:Desktop:Screen Shot 2015-02-01 at 1.20.41 PM.png

The new character classes were added to the charClassMap.



Finally the code which runs the semantic phase was commented out so that the scanner/screener could be ran on it’s own.



**Testing**

To make our testing easier, we broke down our test routine into the following sections:

1. Moving from recognizing single quotes to double quotes
2. Adding Character classes/syntax tokens for the new operators “/“ “|” “#” and “!”
3. Adding syntax support for the keywords “let”, “func”, “switch”, “default”, “elsif”, “loop”, “break”, “module”, “extern” and “string”
4. Removing support for “then”, “until”, “do”, “program”, “const”, “procedure”, “begin”, “repeat”, and “char”
5. Adding new syntax tokens for the operators “::”, “==”, and “!=“
6. Remove syntax token for the former operator “:=”
7. Changing the syntax token for the operator “=“
8. Adding support for c-style block comments and line comments, and removing support for the pt-pascal-style block comments

# Moving from recognizing single quotes to double quotes

The first change we made was having string literals be denoted by double quotes instead of the single quotes present in PT-Pascal. If our code is functioning correctly, we expect the double quotes to be identified as quotes and the first line to output a pLiteral, while the second line is has the single quotes not recognized at all (lIIlegal on the -i output) whereas the series of letters outputs a pIdentifier. Our simple test file for this is:

[characterClassesReplaced.pt](http://characterClassesReplaced.pt)

—————————————————————————————————————————

"hello"

‘hello’

—————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lRightBracket) | [ (lQuote) | .pExtern | .pLiteral |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | ? (lLetter) | .pInteger | .pIdentifier |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | ? (lLetter) | .pNewLine | .pEndFile |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lRightBracket) | ? (lLetter) |  |  |
| [ (lRightBracket) | [ (lLetter) |  |  |
| [ (lLeftBrace) | ? (lLetter) |  |  |
| [ (lSlash) | [ (lLetter) |  |  |
| [ (lLetter) | ? (lLetter) |  |  |
| ? (lLetter) | [ (lQuote) |  |  |
| [ (lLetter) | [ (lNewLine) |  |  |
| ? (lLetter) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lIllegal) |  |  |
| ? (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| ? (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| ? (lLetter) | [ (lLetter) |  |  |
| [ (lSlash) | [ (lIllegal) |  |  |
| [ (lLeftBrace) | [ (lIllegal) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lIllegal) | [ (lEndFile) |  |  |

# Adding Character classes/syntax tokens for the new operators “/“, “|”, “#” and “!”

Once we added in the string literals, the next step was to add the character classes and syntax tokens for the operators, “|”, “#”, and “!”. Due to us expecting that the “|” and “#” characters to both be recognized and output a corresponding syntax token whereas the “/” and “!” characters make no sense on their own, and simply output error tokens once they are recognized. To accommodate this, we created three separate test files:

characterClassesAdded.pt

—————————————————————————————————————————

|

#

—————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lRightBracket) | [ (lOrBar) | .pLiteral | .pOrBar |
| [ (lLeftBrace) | [ (lNewLine) | .pNewLine | .pNewLine |
| [ (lRightBracket) | [ (lHash) |  | .pHash |
| [ (lIllegal) | [ (lEndFile) |  | .pEndFile |

characterClassesSlashAdded.pt

—————————————————————————————————————————

/

—————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lRightBracket) | [ (lSlash) | .pNewLine |  |
| [ (lIllegal) | [ (lEndFile) |  |  |

characterClassesExclamationAdded.pt

—————————————————————————————————————————

!

—————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lRightBracket) | [ (lExclamation) | .pNewLine |  |
| [ (lIllegal) | [ (lEndFile) |  |  |

# Adding syntax support for the keywords “let”, “func”, “switch”, “default”, “elsif”, “loop”, “break”, “module”, “extern” and “string”

Next we added new keywords and their relevant syntax tokens to the compiler. To test if they were properly supported, we created a test file with each of the new keywords on each line. We expect each of the keywords to be initially identified as a series of letters but output a pIdentifier screened to its corresponding syntax token.

keywordsAdded.pt

——————————————————————————————————————————

let

func

switch

default

elsif

loop

break

module

extern

string

——————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lLetter) | [ (lLetter) | .pExtern | .pIdentifier (screened to pLet) |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern | .pIdentifier (screened to pFunc) |
| [ (lLeftBrace) | [ (lNewLine) | .pLiteral | .pNewLine |
| [ (lLeftBrace) | [ (lNewLine) | .pExtern | .pIdentifier (screened to pSwitch) |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern | .pIdentifier (screened to pDefault) |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern | .pIdentifier (screened to pElsif) |
| [ (lLeftBrace) | [ (lNewLine) | .pLiteral | .pNewLine |
| [ (lLeftBrace) | [ (lNewLine) | .pExtern | .pIdentifier (screened to pLoop) |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern | .pIdentifier (screened to pBreak) |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern | .pIdentifier (screened to pModule) |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern | .pIdentifier (screened to pExtern) |
| [ (lLeftBrace) | [ (lNewLine) | .pLiteral | .pNewLine |
| [ (lLeftBrace) | [ (lNewLine) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern | .pIdentifier |
| [ (lLetter) | [ (lLetter) | .pNewLine | .pEndFile |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lIllegal) | [ (lEndFile) |  |  |
| [ (lIllegal) | [ (lEndFile) |  |  |

# Removing support for “then”, “until”, “do”, “program”, “const”, “procedure”, “begin”, “repeat”, and “char”

We then removed the pt-pascal keywords that were no longer welcome in drift. To test if they were removed with prejudice, we created a test file with each line containing one of them. We hope that they are identified as a series of letters, and output a pIdentifier that no longer screens to any other syntax token.

keywordsRemoved.pt

——————————————————————————————————————————

then

until

do

program

const

procedure

begin

repeat

char

——————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lLetter) | [ (lLetter) | .pExtern (screened to pElse) | .pIdentifier |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern (screened to pFile) | .pIdentifier |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLeftBrace) | [ (lNewLine) | .pExtern (screened to pPacked) | .pIdentifier |
| [ (lLeftBrace) | [ (lNewLine) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern (screened to pCase) | .pIdentifier |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern (screened to pWhile) | .pIdentifier |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern (screened to pSwitch) | .pIdentifier |
| [ (lLeftBrace) | [ (lNewLine) | .pLiteral | .pNewLine |
| [ (lLeftBrace) | [ (lNewLine) | .pExtern (screened to pDefault) | .pIdentifier |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern (screened to pModule) | .pIdentifier |
| [ (lLeftBrace) | [ (lNewLine) | .pLiteral | .pNewLine |
| [ (lLeftBrace) | [ (lNewLine) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern | .pIdentifier |
| [ (lLetter) | [ (lLetter) | .pNewLine | .pEndFile |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lIllegal) | [ (lEndFile) |  |  |
| [ (lIllegal) | [ (lEndFile) |  |  |

# Adding new syntax tokens for the operators “::”, “==”, and “!=“

Fresh off the success of removing adding and then removing keywords, we decided to tackle adding syntax tokens head on. We expect that our modified compiler should identify each of the operators’ characters’ character classes normally, including the exclamation mark which the old compiler did not recognize. We further expect that our compiler will output the correct syntax token for each of the added operators.

syntaxTokensAdded.pt——————————————————————————————————————————

::

==

!=

——————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lColon) | [ (lColon) | .pColon | .pDoubleColon |
| [ (lColon) | [ (lColon) | .pColon | .pNewLine |
| [ (lColon) | [ (lNewLine) | .pLiteral | .pEquals |
| [ (lLeftBrace) | [ (lEquals) | .pEquals | .pNewLine |
| [ (lLeftBrace) | [ (lEquals) | .pEquals | .pNotEqual |
| [ (lEquals) | [ (lNewLine) | .pLiteral | .pNewLine |
| [ (lEquals) | [ (lExclamation) | .pEquals | .pEndFile |
| [ (lLeftBrace) | [ (lEquals) | .pLiteral |  |
| [ (lRightBracket) | [ (lNewLine) | .pNewLine |  |
| [ (lEquals) | [ (lEndFile) |  |  |
| [ (lLeftBrace) |  |  |  |
| [ (lIllegal) |  |  |  |

# Remove syntax token for the former operator “:=”

We no longer require the services of the operator “:=”, and as such it has been terminated. To verify that it is indeed no longer with us, we created a test file with just it as the contents. We expect that the individual characters will be identified correctly, but will no longer combine to create the syntax token pStar but will instead output the two syntax tokens pColon and pAssign.

syntaxTokensRemoved.pt

——————————————————————————————————————————

:=

——————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lColon) | [ (lColon) | .pStar | .pColon |
| [ (lEquals) | [ (lEquals) | .pLiteral | .pAssign |
| [ (lLeftBrace) | [ (lEquals) | .pNewLine | .pNewLine |
| [ (lIllegal) | [ (lNewLine) |  | .pEndFile |
|  | [ (lNewLine) |  |  |
|  | [ (lEndFile) |  |  |

# Changing the syntax token for the operator “=“

Now that we have added in the operator “==” to represent the query for equality, we will change what the operator “=” represents. We wish for it to now output pAssign instead of pEquals. As such we have created the following test file with the expectation that it will indeed result in pEquals syntax tokens where appropriate

singleEquals.pt——————————————————————————————————————————

a=b

=——————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lLetter) | [ (lLetter) | .pExtern | .pIdentifier |
| [ (lEquals) | [ (lEquals) | .pEquals | .pAssign |
| [ (lEquals) | [ (lEquals) | .pExtern | .pIdentifier |
| [ (lLetter) | [ (lLetter) | .pLiteral | .pNewLine |
| [ (lLeftBrace) | [ (lLetter) | .pEquals | .pAssign |
| [ (lLeftBrace) | [ (lNewLine) | .pLiteral | .pNewLine |
| [ (lEquals) | [ (lNewLine) | .pNewLine | .pEndFile |
| [ (lLeftBrace) | [ (lEquals) |  |  |
| [ (lIllegal) | [ (lNewLine) |  |  |
|  | [ (lNewLine) |  |  |
|  | [ (lEndFile) |  |  |

# Adding support for c-style block comments and inline comments, and removing support for the PT Pascal-style block comments

We have reached the final addition to the compiler: comments. We wish to support c-style block and line comments, and remove support for the PT Pascal style block comments. As such, we have created four test files: one to verify that c-style block comments work as expected, two to verify that each of the PT Pascal block comments styles no longer function, and finally one to verify that the inline comments functionality works.

We expect that the c-style block comment will simply be removed in its entirety as if it never existed, and impacting newline outputs as such. We also expect that inline comments would not remove the newline for the line they subsist on, while also having everything after (and including) the “//” operator be removed

commentsBlock.pt——————————————————————————————————————————

/\*x\*/

——————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lRightBracket) | [ (lSlash) | .pMinus | .pEndFile |
| [ (lStar) | [ (lStar) | .pExtern |  |
| [ (lLetter) | [ (lLetter) | .pMinus |  |
| [ (lStar) | ? (lLetter) | .pNewLine |  |
| [ (lStar) | [ (lStar) |  |  |
| [ (lRightBracket) | [ (lSlash) |  |  |
| [ (lIllegal) | [ (lEndFile) |  |  |

commentsBlockOld.pt——————————————————————————————————————————

{x}

——————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lLeftParen) | [ (lLeftBrace) | .pNewLine |  |
| [ (lLetter) | ?lBlank (lLeftBrace) |  |  |
| ? (lLetter) |  |  |  |
| [ (lRightParen) |  |  |  |
| [ (lIllegal) |  |  |  |

commentsBlockOldAlt.pt——————————————————————————————————————————

(\*x\*)

——————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lExclamation) | [ (lLeftParen) | .pNewLine | .pLeftParen |
| [ (lStar) | [ (lStar) |  | .pStar |
| [ (lLetter) | [ (lLetter) |  | .pIdentifier |
| ? (lLetter) | [ (lStar) |  | .pStar |
| [ (lStar) | [ (lStar) |  | .pRightParen |
| [ (lQuote) | [ (lRightParen) |  | .pEndFile |
| [ (lIllegal) | [ (lEndFile) |  |  |

commentsLine.pt——————————————————————————————————————————

//x

x//y

xy//

/x

——————————————————————————————————————————

| **Old Compiler -i Output** | **Drift Compiler -i Output** | **Old Compiler -e Output** | **Drift Compiler -e Output** |
| --- | --- | --- | --- |
| [ (lRightBracket) | [ (lSlash) | .pExtern | .pNewLine |
| [ (lRightBracket) | [ (lSlash) | .pLiteral | .pIdentifier |
| [ (lLetter) | [ (lLetter) | .pExtern | .pNewLine |
| [ (lLeftBrace) | ? (lLetter) | .pExtern | .pIdentifier |
| [ (lLeftBrace) | [ (lNewLine) | .pLiteral | .pNewLine |
| [ (lLetter) | [ (lLetter) | .pExtern |  |
| [ (lRightBracket) | [ (lSlash) | .pLiteral |  |
| [ (lRightBracket) | [ (lSlash) | .pExtern |  |
| [ (lRightBracket) | [ (lSlash) | .pLiteral |  |
| [ (lLetter) | [ (lLetter) | .pNewLine |  |
| [ (lLeftBrace) | ? (lLetter) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lRightBracket) | [ (lSlash) |  |  |
| [ (lRightBracket) | [ (lSlash) |  |  |
| [ (lRightBracket) | [ (lSlash) |  |  |
| [ (lLeftBrace) | [ (lNewLine) |  |  |
| [ (lRightBracket) | [ (lSlash) |  |  |
| [ (lLetter) | [ (lLetter) |  |  |
| [ (lLeftBrace) |  |  |  |
| [ (lLeftBrace) |  |  |  |
| [ (lIllegal) |  |  |  |