Debugging the Tokeniser to Support the 'IncrementOne' Development

Tokeniser Tests

During January 2019, the Tokeniser was tested to ensure that it could handle the

Meta operator, and
<==Attribute
<==Key
<--Remove assignments

without further development.

As expected - because ***the operator and assignments all conform to the RAQUEL syntax - the Tokeniser handled them all correctly, and needed no debugging or extension to cope with them correctly***.

However in the course of testing, several bugs were found that had not been discovered before. All were fixed.

In order to carry out the testing and debugging, a copy from the prototype of the InputStack class's Tokeniser member function was made as a standalone function. It was this copy that was tested and debugged. The resulting bug fixes in that copy were then applied to the InputStack class's Tokeniser member function.

Being able to run and inspect the Tokeniser in isolation made it far easier to test and debug. Within the prototype, it is one of many objects related together in complex ways. This makes it very difficult to see what outcome is due solely to the Tokeniser and what might be the result of some other interaction.

A 'main' program was written to provide a test harness for the Tokeniser. 'main' calls the Tokeniser and provides a means of :

1. Enabling the tester to input Raquel statements that are then fed to the Tokeniser.
2. Displaying the resulting word tokens (actually a vector of 'Lex Token' structs) created by the Tokeniser.
3. Displaying the resulting errors (actually a vector of integer error numbers) found by the Tokeniser.

The Tokeniser uses a set of utility functions that are also member functions of the prototype's InputStack class. They are placed in 'NumUtil', a .cpp and corresponding .h file.

The debugged version of the Tokeniser uses an additional function, 'issingle(ch)', which has to be included as an additional InputStack member function.

'TokWOutil', a .cpp and corresponding .h file, contains the Tokeniser function *per se*. Its .h file also contains those items closely bound to the Tokeniser, but found in a variety of locations within the prototype :

1. Enumerated type 'EModuleID' - the RaquelDBMS class .h file

2. Enumerated type 'ETokeniserErrors' - the InputStack class .h file.
3. Enumerated type 'ELexTokenType' - the TokenTypes .h file.
4. 'SLexToken' struct definition - the TokenTypes .h file.
5. 'LexTokenVector' typedef - the TokenTypes .h file.

For ease of use by the 'main' program, the test Tokeniser function was modified to return a standard vector of error numbers called 'Errors'.

In the prototype, the InputStack class includes a protected data member called 'm_pErrors', which is a standard vector of pointers to error numbers; the prototype'sTokeniser uses this instead, and does not return an explicit result.

The InputStack's Tokeniser is called 'TokeniseQuery'. The name 'Tokeniser' or 'Tokenise' is preferred, since all forms of Raquel statement are tokenised in the same way, not just query statements.

The GNU 'g++' compiler was used, with the '–g' flag (so that the 'gdb' debugger could be used with the compiled code) and the '-Wall' flag to reveal any potentially undesirable code formulations.

The executable result was called 'ErrToken', as it can output both error numbers and word tokens.

Bugs Found

They were :

1. Any whitespace character occurring at the beginning of parameter text, i.e. immediately after the initial '[' bracket, was omitted.
2. The '(' and ')' parentheses were not always handled correctly if they were not separated from other characters in the statement by a space character; they were always handled correctly if there was a space character separation. The specific problems were :
   - ')' was ignored if it was not preceded by a space character.
   - ')' was ignored if it was not followed by a space character, except at the end of a statement where it was accepted.
   - '(' was ignored if it was not preceded by a space character. '(' was still accepted if it was not followed by a space character.

   The errors reported by the Tokeniser when it failed to handle parentheses correctly were 'Unbalanced Left Parenthesis' or 'Unbalanced Right Parenthesis'.
3. The state variables ''@' and '#' were ignored. These are not planned to be implemented in the near future, so have not been considered before. (Note that '@' is convenient for use with the **<--Remove** assignment).
4. The '-Wall' flag :
   - shows 2 variables that were declared within the Tokeniser but never used by it. They were :
     - Int **SP** = 0; //Start position of new token.
     - eLexTokenType **eCurrentType** = eLEXTOKEN_KEYWORD.

- suggests that the 'while' statement used to go through each character of the input statement, viz.
  ```
  while (ch = szQuery[ iStatementoffset++ ])
  ```
  should have an extra pair of parentheses around the 'while' condition.

Bug Fixes

The following table summarises the fixes used to solve the bugs :-

Bug No.	Corrective Action	Date Applied to Repo
4	The 2 variables deleted. 'while' condition parentheses added.	18th January 2019 1st February 2019
1	At state **Enter?, Event** *whitespace*, the following **Action** added to existing actions : "Append (whitespace) character to parameter token."	18th January 2019
2	At state **Exit?, Event '('**, the "P ← P + 1" **Action** added to existing **ELSE** actions. At state **Exit?, Event ')'**, the "P ← P - 1" **Action** added to existing **ELSE** actions . Their previous omission meant parentheses were not always correctly counted, so their correct balancing and nesting was not always achieved.	1st February 2019
3	In the **Standard** and **Keyword** states, the *standalone* events did not always correctly handle the ***single*** characters **{** @ # , ; : **}** which are only ever meaningful on their own, as opposed to the remaining *standalone* characters where they may or may not appear in combination with other characters. The ***single*** characters are now treated separately, a word token being created for each one. The remaining *standalone* characters are treated as before.	1st February 2019

The 'Logical Design of the Tokeniser' has been amended appropriately

The bug fixes were tested on a number of sample RAQUEL statements to ensure that both syntactically correct statements were tokenised correctly and syntactically invalid statements returned the correct error numbers.

The changes were made to the **InputStack.cpp** file in the **RaquelDBMS/Branches/AddMeta/src** directory of the repository on Sourceforge.

The extra work of extracting code from the prototype, and carrying across the bug fixes back to the prototype, was considered well worthwhile because of the ease of testing and debugging provided.

All the bugs were located within the internals of the Tokeniser function and not in its interfaces; thus no problems arose from copying the bug fixes into the code of the prototype's InputStack class.

Note

Use of the '–Wall' flag of the 'g++' compiler also warned of the "Unknown Escape Sequence" '\A' in text "\AE\AE". '\AE' is the ASCII character expressed as the decimal value '174' and hex value 'AE'.

The warning is because the character is not part of the basic ASCII character set expressible in 128 bits.

It is in the extended part of the ASCII character set and represents the character '®'.

The logical design of the Tokeniser calls for its output be a sequence of word tokens that include an initial and a terminating word token. The terminating word token was specifically designed to include the text "®®", because '®' suggests the 'Return' key and hence the end of a statement. (The ASCII character '®' is now considered to represent a registered trademark). Therefore the text "\AE\AE" is not an error.

It could be replaced by the basic ASCII character expressed as the decimal value '13' and hex value '0D', and used to represent 'CR' or 'enter/carriage return'.

However the text "®®" is expected by the Compactor and passed through to the Parser. Therefore the change would have to be applied throughout the Tokeniser, Compactor and Parser, or an error would be created.