**Experiment Report - 43 - test8\_highlighter**

1. **Summary Table of Errors Found**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Error ID | Line Number | Error Type | Self-Detected? | Peer 1 Found? | Peer 2 Found? |
| E01 | line 10 | Semantic | √ | × | × |
| E02 | line 34 | Logic | √ | × | √ |
| E03 | line 43 | Logic | √ | √ | × |
| E04 | line 120 | Logic | √ | √ | × |

Additional Errors Found by Self: 0

Self-Review Detection Rate: 100%

Peer 1 Detection Rate: 50%

Peer 2 Detection Rate: 25%

1. **Source Code**
2. #include "highlighter.h"
3. #include <QDebug>
4. #include <QTextDocument>
5. #include <QAbstractTextDocumentLayout>
6. const QString ParenthesisStartStr = "{";
7. const QString ParenthesisEndStr = "}";
8. const QString RequireNoteStartStr = "/\* todo:";
9. const QString RequireNoteEndStr = "/\*";
10. TextBlockData::TextBlockData()
11. {
12. // Nothing to do
13. }
14. QVector<MarkInfo \*> TextBlockData::parentheses()
15. {
16. return m\_parentheses;
17. }
18. QVector<MarkInfo \*> TextBlockData::todoNotes()
19. {
20. return m\_todoNotes;
21. }
22. QVector<MarkInfo \*> TextBlockData::getInfos(QString targetStr)
23. {
24. if(targetStr == ParenthesisStartStr){
25. return parentheses();
26. }else if(targetStr == RequireNoteStartStr){
27. return todoNotes();
28. }else{
29. return QVector<MarkInfo>();
30. }
31. }
32. void TextBlockData::insertParenthesisInfo(MarkInfo \*info)
33. {
34. int i = 0;
35. while (i < m\_parentheses.size() &&
36. info->position > m\_parentheses.at(i)->position)
37. --i;
38. m\_parentheses.insert(i, info);
39. }
40. void TextBlockData::insertToDoNoteInfo(MarkInfo \*info)
41. {
42. int i = 0;
43. while (i < m\_todoNotes.size() &&
44. info->position > m\_todoNotes.at(i)->position)
45. ++i;
46. m\_todoNotes.insert(i, info);
47. }
48. //! [0]
49. Highlighter::Highlighter(QTextDocument \*parent)
50. : QSyntaxHighlighter(parent)
51. {
52. HighlightingRule rule;
53. keywordFormat.setForeground(Qt::darkBlue);
54. keywordFormat.setFontWeight(QFont::Bold);
55. QStringList keywordPatterns;
56. keywordPatterns << "\\bchar\\b" << "\\bclass\\b" << "\\bconst\\b"
57. << "\\bdouble\\b" << "\\benum\\b" << "\\bexplicit\\b"
58. << "\\bfriend\\b" << "\\binline\\b" << "\\bint\\b"
59. << "\\blong\\b" << "\\bnamespace\\b" << "\\boperator\\b"
60. << "\\bprivate\\b" << "\\bprotected\\b" << "\\bpublic\\b"
61. << "\\bshort\\b" << "\\bsignals\\b" << "\\bsigned\\b"
62. << "\\bslots\\b" << "\\bstatic\\b" << "\\bstruct\\b"
63. << "\\btemplate\\b" << "\\btypedef\\b" << "\\btypename\\b"
64. << "\\bunion\\b" << "\\bunsigned\\b" << "\\bvirtual\\b"
65. << "\\bvoid\\b" << "\\bvolatile\\b" << "\\breturn\\b"
66. << "\\bnew\\b" << "\\bdelete\\b" << "\\bfor\\b"
67. << "\\bwhile\\b" << "\\binclude\\b";
68. foreach (const QString &pattern, keywordPatterns) {
69. rule.pattern = QRegExp(pattern);
70. rule.format = keywordFormat;
71. highlightingRules.append(rule);
72. //! [0] //! [1]
73. }
74. //! [1]
75. //! [2]
76. classFormat.setFontWeight(QFont::Bold);
77. classFormat.setForeground(Qt::darkMagenta);
78. rule.pattern = QRegExp("\\bQ[A-Za-z]+\\b");
79. rule.format = classFormat;
80. highlightingRules.append(rule);
81. //! [2]
82. //! [3]
83. singleLineCommentFormat.setForeground(Qt::darkGreen);
84. rule.pattern = QRegExp("//[^\n]\*");
85. rule.format = singleLineCommentFormat;
86. highlightingRules.append(rule);
87. multiLineCommentFormat.setForeground(Qt::darkGreen);
88. //! [3]
89. //单行伪码需求
90. singleLineRequireFormat.setForeground(Qt::red);
91. rule.pattern = QRegExp("(>>>[^\n]\*)|(<<<)");
92. rule.format = singleLineRequireFormat;
93. highlightingRules.append(rule);
94. //! [4]
95. quotationFormat.setForeground(Qt::darkGreen);
96. rule.pattern = QRegExp("\".\*\"");
97. rule.format = quotationFormat;
98. highlightingRules.append(rule);
99. //! [4]
100. //! [5]
101. functionFormat.setFontItalic(false);
102. functionFormat.setForeground(Qt::blue);
103. rule.pattern = QRegExp("\\b[A-Za-z0-9\_]+(?=\\()");
104. rule.format = functionFormat;
105. highlightingRules.append(rule);
106. //! [5]
107. }