June 2022

Language Agnostic Code Highlighting in Word Processors

Barak Ben Noon

Gregory George Galante

Behnoosh Hariri

Tomer Aberbach

Blake Kaplan

See next page for additional authors

Inventor(s)
Barak Ben Noon, Gregory George Galante, Behnoosh Hariri, Tomer Aberbach, Blake Kaplan, and Emily Cahill

Language Agnostic Code Highlighting in Word Processors

ABSTRACT

This disclosure describes techniques for highlighting code snippets included in text documents edited via a word processor. A text block containing code is received and analyzed using a tokenizer to identify specific words included in the text block. The words are classified by the tokenizer into a finite set of types (categories) by matching the words with a list of words defined for different computer languages. Words or characters are colorized based on whether the word is a language specific reserved keyword or a user-defined identifier. Multiple coding languages can be supported, with low maintenance, since only the active dictionary of reserved words needs to be updated when adding a language. The techniques can support live updates, highlighting code even as the user enters text. Incremental highlighting can be implemented with relatively minimal additional effort by analyzing only a small block of code near the altered text character(s).

KEYWORDS

- Word processor
- Code highlighting
- Language syntax
- Code snippet
- Reserved keyword
- Text colorizing
- Text readability
- Tokenizer

BACKGROUND

Display of computer program code commonly utilizes highlighting wherein colorized words and characters are colorized or otherwise highlighted based on the language syntax to improve readability. Code highlighting is used in text articles that include code fragments, in integrated development environments (IDE), in design documents utilized by programmers for collaboration, etc. Highlighting of code text blocks in a document is based on syntax of the language (grammar). Language syntax specific highlighting is not supported by general purpose word processors or other document editing software.

DESCRIPTION

This disclosure describes techniques for highlighting computer program code in text documents displayed in a word processor/ document editor, as part of a website, etc. Each text block of code (code fragment) in a document is analyzed to determine specific words included in the text block. The specific words are classified into types (categories) based on a list of reserved words in specific computer languages. The words in the text block are highlighted, e.g., colorized, based on the category of each word.

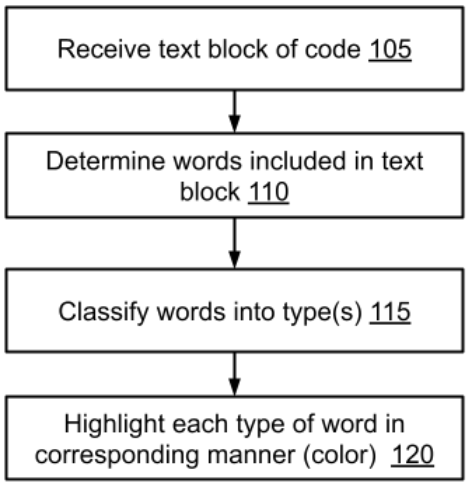

Fig. 1: Method for highlighting text blocks of code

Fig. 1 depicts an example method for the highlighting of text blocks of code, per techniques of this disclosure. A text block of code (code fragment) is received (105). The text block is analyzed using a tokenizer to identify (110) specific word(s) included in the text block. For example, a stream of characters may be scanned to locate words included in the stream by collecting characters until a delimiter (or end of word) is encountered. For example, a whitespace, comma, or parenthesis may be utilized as a delimiter that signals the end of a word.

The identified words are classified (115) by the tokenizer into a finite set of types (categories). Such classification can be performed by matching the words with a list of words defined for each computer language. The tokenizer is provided with a list of possible strings for each category. Examples of types (categories) include:

- *Literals* - string, number, Boolean, etc.
- *Reserved words by the language* - if, for, switch, while, etc.
- *Casing* - Snake case, camel case, upper snake case, etc.
- *Punctuation* - {, “, [, (, ; etc.
- *Operators* - +, -, |, ?

The matching can be performed without assuming that the words are in any particular order or arranged as defined by grammar rules for a specific computer language. Each category of word is highlighted (120), e.g., shown in a different color, to enable a reader to easily distinguish the types of words and provide improved readability.

Per techniques of this disclosure, specific language syntax is not taken into account; instead, words or characters are colorized based on whether the word is a language specific reserved keyword (e.g., the word "class" in Java) or is a user-defined identifier (name). Additionally, a set of words included within a string category is highlighted as a single block

corresponding to the string category rather than as individual words, even if individual words within the string can be classified into one of the types of words.

For example, consider a text block of JavaScript code: `const myVar = "class Browser"`. Even though the text block includes words that correspond to different categories ("class" is a reserved word and "Browser" is a user-defined identifier), the text block is classified as a string and highlighted accordingly.

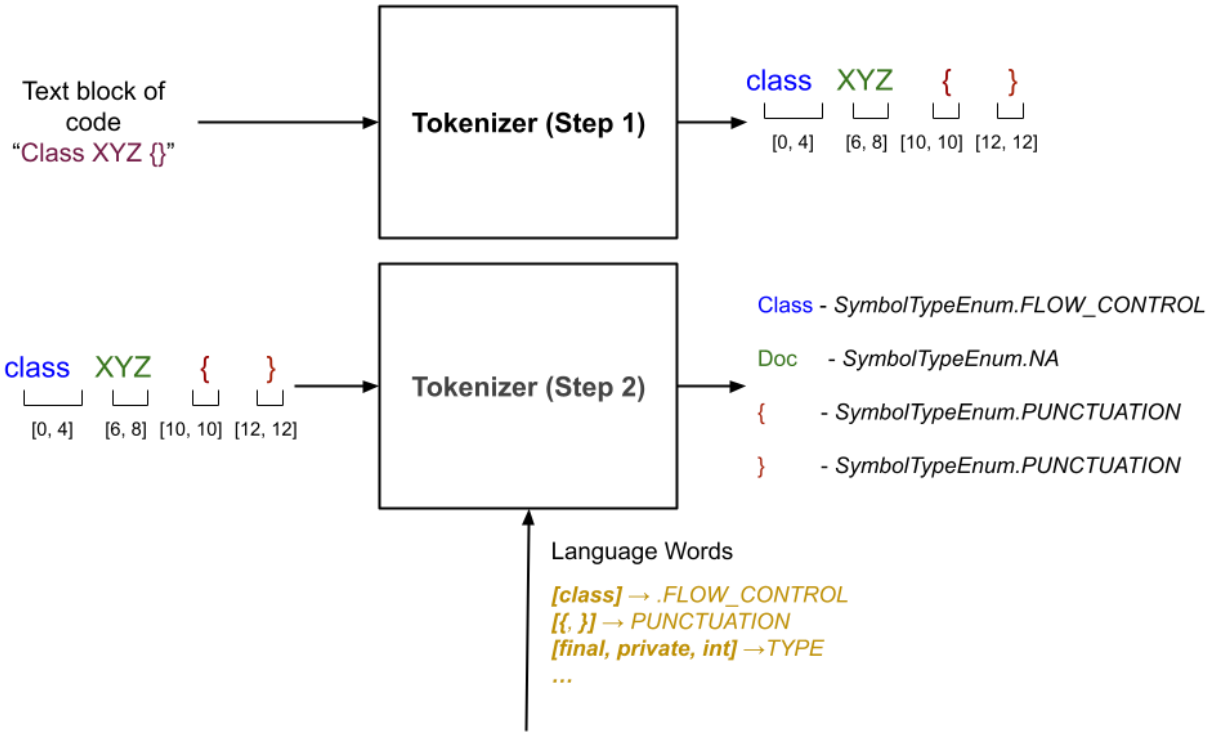

Fig. 2: A tokenizer is utilized to detect and classify words included in code

Fig. 2 depicts a tokenizer utilized to detect and classify words included in text blocks, per techniques of this disclosure. As depicted in Fig. 2, an input text block of code "Class XYZ {}" is received at the tokenizer. The tokenizer is utilized to detect specific words included in the text block - "class", "XYZ", "{", and "}". Based on a list of language words provided to the tokenizer, categories are determined for each word. Words that are not classified by the tokenizer are presumed to be user coined words (and categorized, for example as "NA").

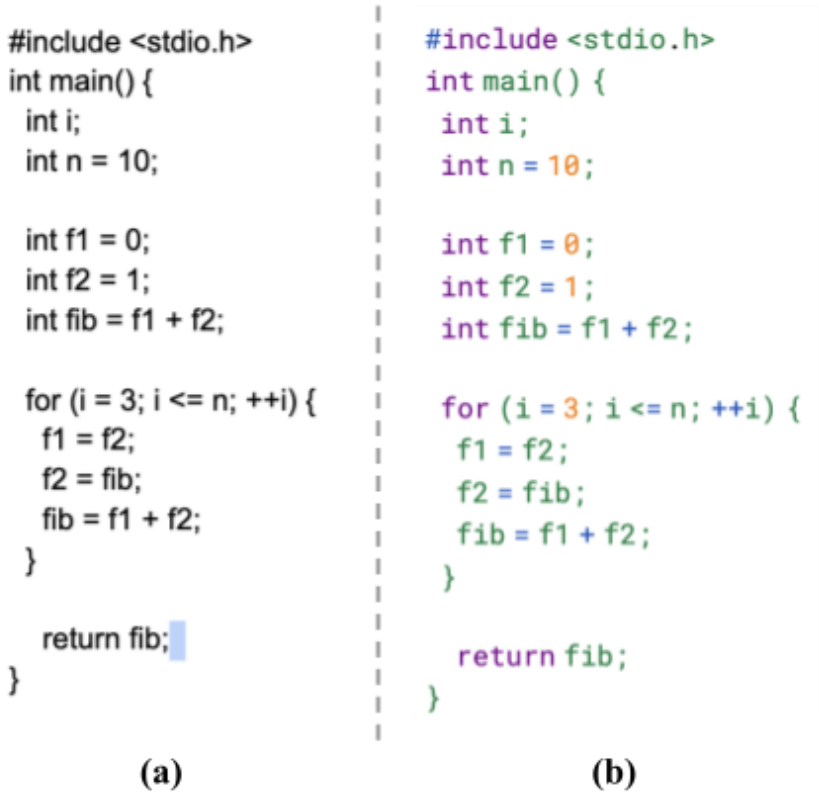

Fig. 3: Example output of highlighted code

Fig. 3 depicts an input code snippet and a corresponding output highlighted code snippet. Fig. 3(a) depicts the input code snippet and Fig. 3(b) depicts the corresponding highlighted (colorized) version. As can be seen, different colors are automatically utilized for different categories of words within the input code snippet thus improving readability.

The techniques described herein can be utilized in any platform and support consistent rendering of highlighted code on mobile devices, desktops, or other types of devices. Multiple coding languages can be supported, with low maintenance, since only the active dictionary of reserved words needs to be updated when adding a language. The techniques can support live updates, highlighting code even as the user enters text. Incremental highlighting can be implemented with relatively minimal additional effort by analyzing only a small block of code near the altered text character(s).

CONCLUSION

This disclosure describes techniques for highlighting code included in text documents. A text block of code is received and analyzed using a tokenizer to determine specific words included in the text block. The words are classified by the tokenizer into a finite set of types (categories) by matching the words with a list of words defined for each computer language. Words or characters are colorized based on whether the word is a language specific reserved keyword or is a user-defined identifier. Multiple coding languages can be supported, with low maintenance when it comes to language changes, since only an active dictionary of reserved words needs to be updated. The techniques can be utilized to support incremental code updates, even as the user is typing the code text. Incremental highlighting can be implemented with minimal additional effort by analyzing a block of code surrounding the altered text character(s).