Stephen Ostermiller's Blog

Finding Comments in Source Code Using Regular Expressions

Many text editors have advanced find (and replace) features. When I’m programming, I like to use an editor with regular expression search and replace. This feature is allows one to find text based on complex patterns rather than based just on literals. Upon occasion I want to examine each of the comments in my source code and either edit them or remove them. I found that it was difficult to write a regular expression that would find C style comments (the comments that start with /* and end with */) because my text editor does not implement the “non-greedy matching” feature of regular expressions.

First Try

When first attempting this problem, most people consider the regular expression:

```
/\*.*\*/
```

This seems the natural way to do it. `/\*` finds the start of the comment (note that the literal `*` needs to be escaped because `*` has a special meaning in regular expressions), `.*` finds any number of any character, and `\*/` finds the end of the expression.

The first problem with this approach is that `.*` does not match new lines.

```
/* First comment
 first comment—line two*/
/* Second comment */
```

Second Try

This can be overcome easily by replacing the `.` with `[^]` (in some regular expression packages) or more generally with `(.|[\r\n])`:

`/\*(.|[\r\n])*\*/`

This reveals a second, more serious, problem—the expression matches too much. Regular expressions are greedy, they take in as much as they can. Consider the case in which your file has two comments. This regular expression will match them both along with anything in between:

```
start_code();
/* First comment */
more_code();
/* Second comment */
end_code();
```

Third Try

To fix this, the regular expression must accept less. We cannot accept just any character with a `.`, we need to limit the types of characters that can be in our expressions:

`/\*([^*]|[\r\n])*\*/`

This simplistic approach doesn't accept any comments with a `*` in them.

```
/*
 * Common multi-line comment style.
 */
/* Second comment */
```

Fourth Try

This is where it gets tricky. How do we accept a `*` without accepting the `*` that is part of the end comment? The solution is to still accept any character that is not `*`, but also accept a `*` and anything that follows it provided that it isn't followed by a `/`:

`/\*([^*]|[\r\n]|(\*([^/]|[\r\n])))*\*/`

This works better but again accepts too much in some cases. It will accept any even number of `*`. It might even accept the `*` that is supposed to end the comment.

```
start_code();
/****
 * Common multi-line comment style.
 ****/
more_code();
/*
 * Another common multi-line comment style.
 */
end_code();
```

Fifth Try

What we tried before will work if we accept any number of `*` followed by anything other than a `*` or a `/`:

`/\*([^*]|[\r\n]|(\*+([^*/]|[\r\n])))*\*/`

Now the regular expression does not accept enough again. Its working better than ever, but it still leaves one case. It does not accept comments that end in multiple `*`.

```
/****
 * Common multi-line comment style.
 ****/
/****
 * Another common multi-line comment style.
 */
```

Solution

Now we just need to modify the comment end to allow any number of `*`:

`/\*([^*]|[\r\n]|(\*+([^*/]|[\r\n])))*\*+/`

We now have a regular expression that we can paste into text editors that support regular expressions. Finding our comments is a matter of pressing the find button. You might be able to simplify this expression somewhat for your particular editor. For example, in some regular expression implementations, `[^]` assumes the `[\r\n]` and all the `[\r\n]` can be removed from the expression.

This is easy to augment so that it will also find `//` style comments:

`(/\*([^*]|[\r\n]|(\*+([^*/]|[\r\n])))*\*+/)|(//.*)`

Tool	Expression and Usage	Notes
nedit	`(/\*([^*]|[\r\n]|(\*+([^*/]|[\r\n])))*\*+/)|(//.*)` Ctrl+F to find, put in expression, check the Regular Expression check box.	`[^]` does not include new line
grep	`(/\*([^*]|(\*+[^*/]))*\*+/)|(//.*)` grep -E "(/*([^*]|(*+[^*/]))**+/)|(//.*)" <files>	Does not support multi-line comments, will print out each line that completely contains a comment.
perl	`/((?:\/\*(?:[^*]|(?:\*+[^*\/]))*\*+\/)|(?:\/\/.*))/` perl -e "$/=undef;print<>=~/((?:\/*(?:[^*]|(?:*+[^*\/]))**+\/)|(?:\/\/.*))/g;" < <file>	Prints out all the comments run together. The `(?:` notation must be used for non-capturing parenthesis. Each `/` must be escaped because it delimits the expression. `$/=undef;` is used so that the file is not matched line by line like grep.
Java	`"(?:/\\*(?:[^*]|(?:\\*+[^*/]))*\\*+/)|(?://.*)"` System.out.println(sourcecode.replaceAll("(?:/\\*(?:[^*]|(?:\\*+[^*/]))\+/)|(?://.*)",""));	Prints out the contents of the string sourcecode with the comments removed. The `(?:` notation must be used for non-capturing parenthesis. Each `\` must be escaped in a Java String.

An Easier Method Non-greedy Matching

Most regular expression packages support non-greedy matching. This means that the pattern will only be matched if there is no other choice. We can modify our second try to use the non-greedy matcher `*?` instead of the greedy matcher `*`. With this new tool, the middle of our comment will only match if it doesn't match the end:

`/\*(.|[\r\n])*?\*/`

Tool	Expression and Usage	Notes
nedit	`/\*(.|[\r\n])*?\*/` Ctrl+F to find, put in expression, check the Regular Expression check box.	`[^]` does not include new line
grep	`/\*.*?\*/` grep -E '/*.*?*/' <file>	Does not support multi-line comments, will print out each line that completely contains a comment.
perl	`/\*(?:.|[\r\n])*?\*/` perl -0777ne 'print m!/*(?:.|[\r\n])*?*/!g;' <file>	Prints out all the comments run together. The `(?:` notation must be used for non-capturing parenthesis. `/` does not have to be escaped because `!` delimits the expression. -0777 is used to enable slurp mode and -n enables automatic reading.
Java	`"/\\*(?:.|[\\n\\r])*?\\*/"` System.out.println(sourcecode.replaceAll("/\\*(?:.|[\\n\\r])*?\\*/",""));	Prints out the contents of the string sourcecode with the comments removed. The `(?:` notation must be used for non-capturing parenthesis. Each `\` must be escaped in a Java String.

Caveats Comments Inside Other Elements

Although our regular expression describes c-style comments very well, there are still problems when something appears to be a comment but is actually part of a larger element.

```
someString = "An example comment: /* example */";

// The comment around this code has been commented out.
// /*
some_code();
// */
```

The solution to this is to write regular expressions that describe each of the possible larger elements, find these as well, decide what type of element each is, and discard the ones that are not comments. There are tools called lexers or tokenizers that can help with this task. A lexer accepts regular expressions as input, scans a stream, picks out tokens that match the regular expressions, and classifies the token based on which expression it matched. The greedy property of regular expressions is used to ensure the longest match. Although writing a full lexer for C is beyond the scope of this document, those interested should look at lexer generators such as Flex and JFlex.

Leave a comment

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

3 thoughts on "Finding Comments in Source Code Using Regular Expressions"

Yandot

Reply ↓

January 2, 2016 at 6:22 pm

Hi Stephen. Good work! very useful. However it seems the perl expressions (both the solution one and the non-greedy version) do not render properly the end of lines. For example with this code: `const uint8_t* cmd_descriptor_block,/**<[in] */ int cmd_descriptor_block_size, /**[in] */ uint8_t* storage_for_ctx, /**<[in] */ int storage_max_size, /**<[in] */ uint8_t expected_data_xfer_len, /**<[in] */ struct context** command_struct, /**<[out] It uses storage_for_ctx memory. */ enum types* i_type /**<[out] */` I get this: `/**<[in] *//**[in] *//**<[in] *//**<[in] *//**<[in] *//**<[out] It uses storage_for_ctx memory. *//**<[out] */` Cheers!

Trevor Sundberg

Reply ↓

March 8, 2016 at 5:16 am

I've stumbled across this a few times, and I always find it incredibly useful. This is one of those cases where writing out the DFA that the regular expression builds is actually much easier to understand and to handle the edge cases (like multiple stars at the end). Thank you!

Roman

Reply ↓

July 7, 2016 at 4:13 am

Thanks a lot for this post! I could not go on the first try. Now it is work fine with r'(\/*(.|[\r\n])*?*\/)' in Python.

Post navigation

CSS Positioning →

Meta