CodeStyliser version 0.12

The Code Styliser utility fixes MISRA-C-2012 Rule 15.6 warnings automatically, by adding curly braces (`{}`) in C-source (`.c`) files wherever needed.

It adds braces after single line `if()`, `for()`, `while()`, `else` and `else if()` statements.

For example,

```
if(condition)
  statement;
```

will become

```
if(condition) {
  statement;
}
```

and,

```
if (condition)
  statement;
else
  statement;
```

becomes

```
if(condition) {
  statement;
} else {
  statement;
}
```

It is my suggestion to check all the code once this utility is finished running.

NOTE: This utility will replace all `\r\n` line endings with `\n`.

Usage

This utility is written in Python 3.7.7, 64-Bit. However, you **dont** need to have python installed on your machine.

NOTE: The binary file attached works only for 64-Bit Linux systems.

The binary has been tested on **Ubuntu 16.04 LTS**

The usage is as follows:

- Download the binary file attached into the directory you want to run the utility from.
- Now, just run the file in the following manner
    - If you want to run it for a file:

    `$ ./codeStyliser -f file-name`

    - Or, if you want to run it for a directory:

    `$ ./codeStyliser -d directory-name`
- The complete usage of this binary is as follows:

```
usage: codeStyliser [-h] (-f  file-name | -d  directory-name)

The Code Styliser Utility, Created by Pranjal Rastogi. Copyright (c) 2020,
Pranjal Rastogi All rights Reserved

optional arguments:
  -h, --help            show this help message and exit
  -f  file-name, --file  file-name
                        name of file to format
  -d  directory-name, --directory  directory-name
                        directory name, under which to format all files
```

Known issues

Encoding issue

The utility only changes UTF-8 encoded files, it will ignore all other file encodings.

Non-keyword block of code

If a block of code, that is not a keyword(`for()`, `if()`, `while()`, `do`, etc.) appears immediately after a detected keyword (not containing curly braces), the closing curly brace (`}`) is added in the wrong position.

For example,

```
for (...)
  NOT_A_KEYWORD () {
    statement 1;
    statement 2;
    ...
  }
```

becomes:

```
for (...) {
  NOT_A_KEYWORD () {
    statement 1;
}
    statement 2;
    ...
  }
```

This doesnt happen if the inner block starts on a keyword.

Tab indentation error

If the characters used to indent keywords, are tabs (`\t`), then the output code will not be indented properly.

The indented characters must always use spaces to indent.

For example,

```
      for(...)
      statement;
```

becomes

```
      for(...) {
        statement;
  }
```

if the characters used to indent the `for(...)` are tabs, and not spaces.

Ignoring '#' keywords

The code will ignore all keywords that have a `#` on the next immediate valid line.

A valid line is a line which has no comment and isnt a blank line.

For example,

```
      for(...)
    #endif

      for(...)

      // comment

    #endif

    for(...) statement;
    #endif
```

Here, the `for(...)` is ignored and no braces will be added.

```
 for(...)
foo;
#endif
```

In the above case, the `for(...)` is not ignored and no braces are added.

Issues with Comments

Comments like the following will cause issues, as explained in this block:

```
 // the following for(); isnt detected

/*comment*/ for(...)
              statement;

// this will result in an error, and this line will be ignored

for( /*comment*/ )
  statement;

// this for is also not detected
/*comment*/ for(...) //comment
          statement;

// Similiarly, this for is also not detected
/*comment*/ for() /*comment*/
              statement;

// Here, the code results in an UnboundLocalError

/*
foo */ for(...) statement; //bar
```

Copyright