CodeStyliser

This is the Code Styliser utility.

The Code Styliser utility adds 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;
}
```

It is my suggestion to check all the code once this utility is finished running, as few issues may arise.

There is no guarantee that this software will work out of the box without any issues. Please check License for more details

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.

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.py [-h] (-f  file-name | -d  directory-name)

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

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.) is in the code after a keyword, which doesn't have curly braces (`{}`), the closing curly brace (`}`) is added in the wrong position by the utility.

For example,

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

becomes:

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

This doesnt happen if the block of code 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.

Hash ignore

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, i.e. *NO* braces will be added.

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

In the above case, the `for(...)` is not ignored, i.e. braces will be 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
```

License