Assignment #02: Implement Bracket Linter

0. Basic Information

- Class:
  - Computer Programming (Spring 2024, Prof. Hyeonsang Eom)
- Version info:
  - Last modified time: `Wed Apr 17 19:50:18 KST 2024`
- Contact info:
  - Email: haeram.kim1@snu.ac.kr

1. Lint Rules

1.1. Indent and Line Break

- Given a string containing some bracket characters (`'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`), lint the string based on the rules described below:
  1. After the opening brackets (`'('`, `'{'`, and `'['`), a new line *MUST* be added and the indent level *MUST* be increased.
  2. For the closing brackets (`')'`, `'}'`, and `']'`), the indent level *MUST* be decreased beforehand, and then the given closing bracket *MUST* be shown, and then a new line *MUST* be added.
- For example, for the string:

```
{{}}
```

- After applying rule no.1:

```
{
    {
        }}
```

- And after applying rule no.2:

```
{
    {
    }
}
```

- Additional lint examples are listed in section 2.1.

1.2. Parentheses Validity Check

- When the parentheses are invalid, an error message *MUST* be printed.
- Possible invalid parentheses are as follows:
    1. Pair not matched (e.g. opening bracket `'{'` is closed with `']'`.)
    2. Missing pair (e.g. corresponding closing bracket for an opening bracket `'{'` is missing or vice versa.)
- For example, for the string:

```
[(})]
```

- As opening bracket `'{'` is missing for the closing bracket `'}'`, so the following message *MUST* be printed:

```
error: invalid parentheses
```

- Additional lint error examples are listed in section 2.2.

2. Input & Output Examples

2.1. Lint Examples

2.1.1.

- Input:

```
{{}}
```

- Output:

```
{
    {
    }
}
```

2.1.2

- Input:

```
{}{}{}
```

- Output:

```
{
}
{
}
{
}
```

2.1.3

- Input:

```
[{()}()]
```

- Output:

```
[
    {
        (
        )
    }
    (
    )
]
```

2.2. Lint Error Examples

2.2.1.

- Input:

```
[(})]
```

- Output:

```
error: invalid parentheses
```

2.2.2.

- Input:

```
[({))]
```

- Output:

```
error: invalid parentheses
```

2.2.3.

- Input:

```
{}{}[{()}([])]
```

- Output:

```
error: invalid parentheses
```

3. Implementation Guide

3.1. Scope

- In this assignment, students have to implement bracket linter in two ways:
  - Lint string using a stack (Level 1)
  - Lint string using a queue (Level 2)
- The scope of this assignment is implementing two functions:
  - `lint` function in `StackLinter` class
  - `lint` function in `QueueLinter` class
- All required functionalities are already implemented in the skeleton code except for these two functions.
- *DO NOT* modify the code outside of these functions; Submission which modified the pre-implemented code will be scored 0.
  - Minor modifications (space or new line) are ok.

3.2. Input and Output

- Each function has no parameters for the input.
- Instead, the input string is provided in the form of a character queue (`queue<char> q`) inside of the class variable.
- And that same character queue is used to store the output string.
  - It *DOES NOT* mean that the input and output are stored together; Only the output *MUST* be stored after the lint.
- A return value for the function is the status code.
  - If linting is done without error, it *MUST* return 0.
  - And 1 *MUST* be returned when there is some error.
- Error output is done simply with `setError()` pre-implemented function.
  - Students *MUST* use this function to output the error message (`"error: invalid parentheses"`)
  - Note that this function takes no required parameters.

3.3. Stack and Queue

- In `StackLinter` class, students *MUST* use `stack<char> stk` class member variable to implement the `lint` function.
- Also, in `QueueLinter` class, students *MUST* use `queue<char> q_stk` class member variable.
- Defining and using another queue is not allowed in `StackLinter` class's `lint` function and

defining and using another stack is not allowed in `QueueLinter` class's `lint` function.

3.4. `Linter` Parent Class

- Each class inherits a pre-implemented parent class named `Linter`.
- In this class, various members that can be used in each child class are already implemented:
  - `TAB` and `NEWLINE` constant character: Students *MUST* use this constant character to represent the indents and new lines (or manually, using `'\t'` and `'\n'` characters is also allowed).
  - `q` character queue variable: As mentioned above, the input or output string *MUST* be stored in this same queue.
  - `setError()` function: As mentioned above, one *MUST* use this function to output an error message.

3.5. Main Function

- When compiling and running the code, the input string is provided with the user keyboard input, and after pressing enter, a string is processed with the `lint` function, and finally, the lint result is shown on the screen.
- All processes through keyboard input and screen output are done by the `main` function and it is already implemented (PLZ do not modify!).

4. Score Policy

- Deadline: Apr 30 23:59, 2024
  - Late submission: `pts -1` for 1 day delay, `0` after May 8 00:00, 2024
- Submission:
  - Filename: `학번_assignment02.cc`
  - Location: online ETL
- Scores (total 8 pts):
  - Level 1: 4 pts
  - Level 2: 4 pts
- Fails (0 pts):
  - Code other than two functions (`StackLinter::lint`, `QueueLinter::lint`) is modified.
  - Cannot compile the code.
  - Runtime error (e.g. segment failure, infinite loop)

- Lint result is not matched with the test cases.