CSE 102 Homework Assignment 5

DUE

December 20, 2020, 23:55

Description

- This is an individual assignment. Please do not collaborate
- If you think that this document does not clearly describe the assignment, ask questions before its too late.
- This assignment is about recursive functions. You can use pointers and functions. You **cannot** use dynamical memory allocation. You can use global variables. You can use standard string library operations.

This is a `C` programming assignment. You will write a `C` program according to the following description.

- **You have to use recursion for the core functionality. If you don't, you will get `0.0`.**
- Files to read: `input.txt`
- Files to create: `output.txt`
- `stdout`: **Do not** print anything.
- Your program will read a file named `input.txt`
- `input.txt` contains single line text. The text describes a tree structure. It follows the `Newick` format rules.
  - Each node is represented by a string (A string which does not include these chars: `(`, `)`, `,`). They are separated by `comma`. Branching is described by matching parentheses.
  - The length of the text will not be greater than `250`.
  - A string representing a node may have spaces in it.
  - A string representing a node **cannot** be an empty string.
  - You don't have to check for any errors. The given string will be a true `Newick` format description of a tree.
  - There may be repetitions.
  - Ignore spaces at the beginning and at the end of the input string. (Do not ignore spaces at any other places).
  - The maximum length of a node name is `10` chars.
- `output.txt` contains the visualization of the tree.
- Below is an example of a tree description:

```
(Ali,(c,Beee,ec ee),K,Dayi,e,(f,(d,Ali)))
```

- This tree can be visualized as follows:

```
-Ali
--c
--Beee
--ec ee
-K
-Dayi
-e
--f
---d
---Ali
```

- Another example:

```
(A,(A   ,    A,(A),   A  ),   A)
```

- Visualization (Do not print the explanations which are in `""`):

```
-A
--A                     "This node name has 3 spaces at the end."
--    A                 "This node name has 4 spaces before `A`."
---A
--   A                  "This node name has 3 spaces before `A`, and 2 spaces after."
-   A                   "This node name has 3 spaces before `A`."
```

- Another example:

```
A,(A,A,(A),A),A
```

- Visualization:

```
A
-A
-A
--A
-A
A
```

- Your program will read the simplified `Newick` formatted tree description from a file and write the visualization of the described tree to `output.txt`.
- Core part of your implementation (parsing and printing) should utilize **recursion** otherwise you will get `0pts`.
- Pay attention to the structure of the output. Don't print any debug messages or greeting texts. Don't add extra spaces, newlines, line numbers, commas etc...

Remarks:

- Please test your code extensively. Try limit values, extreme cases. For example:
  - What happens if the input file has this: `A` ? This tree has a single node. The output should be `A`.
  - What about this input?: `(((A)))` The visualization should be `---A`.
- You don't have to do error checking on the input file. You can safely assume that you will be given a proper input file which doesn't violate the described format.
- Be careful with the size of the array you allocate in program stack. Large arrays may not fit in program stack(stack size may be smaller on the test machine) and your program crashes.
- Make sure you can read input files with or without a tailing newline at the end. (If you are using a windows machine, newline is `CRLF`, on unix it is `LF`). You can alter this using advanced editors (i.e. Visual Studio Code). Test your code for every possible combination.
- Do not print anything other than the expected output. (For this assignment, your program prints NOTHING).
- You cannot use anything which is not covered in class.
- Do not submit any of the files you used for testing.
- Do not submit your output file.

Turn in:

- Source code of a complete C program. Name of the file should be in this format: `<full_name>_<id>.c`.
- Example: `gokhan_kaya_000000.c`. Please do not use any Turkish special characters.
- You don't need to use an IDE for this assignment. Your code will be compiled and run in a command window.
- Your code will be compiled and tested on a Linux machine(Ubuntu). GCC will be used.
- Make sure you don't get compile errors when you issue this command : `gcc <full_name>_<id>.c`.
- A script will be used in order to check the correctness of your results. So, be careful not to violate the expected output format.
- Provide comments unless you are not interested in partial credit. (If I cannot easily understand your design, you may loose points.)
- You may not get full credit if your implementation contradicts with the statements in this document.

Late Submission

- Late submission is NOT accepted.

Grading (Tentative)

- `Max Grade : 100`.

All of the followings are possible deductions from `Max Grade`.

- `#define HARD_CODED_VALUES -10`.
- No submission: `-100`. (be consistent in doing this and your overall grade will converge to `N/A`) (To be specific: if you miss `3` assignments you'll get `N/A`)
- Compile errors: `-100`.
- Irrelevant code: `-100`.
- Major parts are missing: `-100`.
- Unnecessarily long code: `-30`.
- Using language elements and libraries which are not allowed: `-100`.
- Not caring about the structure and efficiency: `-30`. (avoid using hard-coded values).
- Significant number of compiler warnings: `-10`.
- Not commented enough: `-5`. (Comments are in English).
- Source code encoding is not `UTF-8` and characters are not properly displayed: `-5`. (You can use 'Visual Studio Code', 'Sublime Text', 'Atom' etc... Check the character encoding of your text editor and set it to `UTF-8`).
- Fails at reading the file: `-30`.
- Fails reading an input with spaces: `-20`.
- Fails parsing node names with multiple characters. `-20`.
- Fails parsing identical node names. `-20`.
- Fails parsing multiple nested parenthesis: `-20`.
- Does not ignore the spaces at the beginning and at the end: `-20`.
- Fails during file write: `-30`.
- Wrong hierarchy at the output: `-30`.
- Wrong node names: `-30`.
- Output format is wrong: `-30`. (Be careful with spacing)
- Infinite loop: `-90`.
- Prints anything extra: `-30`.
- Unwanted chars and spaces in `output.txt`: `-30`.
- Submission includes files other than the expected: `-10`.
- Submission does not follow the file naming convention: `-10`.
- Sharing or inheriting code: `-200`.
- IF YOU DON'T FOLLOW THE FILE NAMING CONVENTIONS YOU WILL GET `0`.

Note: Some of these items are not independent. So, you cannot expect isolation of many of them. For example, if you cannot read input file correctly, you will fail to produce the correct output file. Partial grading is not guaranteed.