CSE 102 Homework Assignment 7

DUE

December 18, 2018, 23:55

Description

- This is an individual assignment. Please do not collaborate.
- If you think that this document does not clearly describes the assignment, ask questions before its too late.

For this assignment, you are expected to write a program which simulates a logic circuit for a set of given inputs.

In this assignment, you are going to use structure and dynamic memory allocation.

Be careful with file names. You won't given a chance to correct any mistakes.

- Your program reads two files:
  - `circuit.txt`
  - `input.txt`
- Your program creates a text file:
  - `output.txt`
- According to content in `circuit.txt`, the program **dynamically** creates necessary structures for a logic circuit and evaluates the cases listed in `input.txt`.
- Your program prints the output to `output.txt`. Each output step should be on a separate line.

`circuit.txt`

- Each line starts with a `keyword`. Possible `keywords`:

  ```
  INPUT
  AND
  OR
  NOT
  FLIPFLOP
  ```

- The first line specifies input labels. Labels are separated by spaces. Example:

  ```
  INPUT a input2 c3 k
  ```

- Here there are 4 inputs are defined. Each has an identifier. `a`, `input2`, `c3`, `k`.
- `AND` keyword specifies that there is an `and` gate defined. `AND` keyword follows the identifier for the gate and two other identifiers for the inputs. Example:

  ```
  AND gate_A c3 another_id
  ```

- Here the `and` gate is identified by the string `gate_A`. Its inputs are identified `c3` and `another_id`. These identifiers can be input identifiers or identifiers for other gates.
- `OR` keyword specifies that there is an `or` gate defined. `OR` keyword follows the identifier for the gate and two other identifiers for the inputs. Example:

  ```
  OR gate_B ck id3
  ```

- Here the `or` gate is identified by the string `gate_B`. Its inputs are identified `ck` and `id3`. These identifiers can be input identifiers or identifiers for other gates.

- `NOT` keyword specifies that there is an `not` gate defined. `NOT` keyword follows the identifier for the gate and one other identifier for its input. Example:

  ```
  NOT gate_C c5
  ```

- Here the `not` gate is identified by the string `gate_C`. It has only one input an it is identified by the string `c5`.
- `FLIPFLOP` keyword specifies that there is an `flip-flop` gate defined. `FLIPFLOP` keyword follows the identifier for the gate and one other identifier for its input. Example:

  ```
  FLIPFLOP gate_F c6
  ```

- Here the `flip-flop` gate is identified by the string `gate_F`. Its input is identified by `c6`.

`input.txt`

- Each line is a list of `1` and `0`. Example:

```
1 0 1 1
0 1 1 1
0 0 1 0
1 0 0 1
```

Example:

- Suppose that `circuit.txt` is has the following content:

```
INPUT a b c d
AND and1 a b
OR or1 and1 c
NOT n1 d
FLIPFLOP f1 n1
AND a2 or1 f1
```

- `input.txt` has the following content:

```
1 1 0 1
1 0 1 0
1 1 1 0
```

- Assume that initially `former-out` of any `FLIPFLOP` is `0`.
- Any `FLIPFLOP`s should preserve the state throughout the evaluation of the whole `input.txt`.
- Each line in `input.txt` is assigned to identifiers `a`, `b`, `c`, `d`, defined in `circuit.txt`. According to the truth tables, outputs of gates are calculated.
- For the `input.txt` given, the contents of `output.txt` should be:

```
0
1
0
```

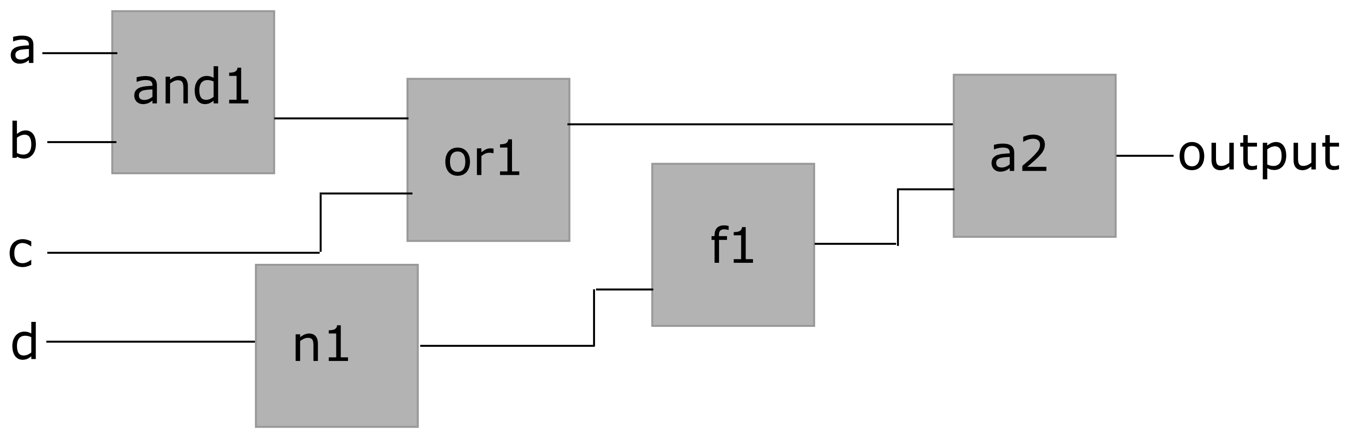

Figure 1: Example Logic Circuit

Remarks

- Output is not defined explicitly. It is your job to figure out the output pin. There will always going to be one output pin.
- Each identifier is unique. Max length of each identifier is `10` chars.
- Assume there won't be multiple spaces separating the identifiers, keywords or data. Tokenize it according to this assumption.
- There won't be any errors in the files.
- You **have to** use `dynamical memory allocation` and `struct`. You **have to** use `struct` in order to represent the gates of the circuit.
- Do not submit your code without testing it with several different scenarios. Try cases with multiple gates of the same type. Try cases with single gate. Try `input.txt` with thousands of lines. Try cases with many gates. Try cases where there are many inputs.

- You can use `c structs`, unions, arrays, c strings, pointers, recursion.
- Write comments in your code.
- Do not print anything to `stdout` and `stderr`.
- 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.

Truth Tables:

- AND

a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

- OR

a	b	out
0	0	0
0	1	1
1	0	1
1	1	1

- NOT

a	out
0	1
1	0

- FLIPFLOP

a	former_out	out
0	0	0
0	1	1
1	0	1
1	1	0

Late Submission

- Late submission is **NOT** accepted.

Grading (Tentative)

- `Max Grade` : `100`.
- Multiple tests(at least `5`) will be performed. The tests may be completely different from the examples given in this document.

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, avoid hard-to-follow expressions, avoid code repetition, avoid unnecessary loops).
- 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`).
- Missing or wrong output values: `Fails the test`.
- Output format is wrong: `-30`.
- Infinite loop: `Fails the test`.
- Segmentation fault: `Fails the test`.
- Fails 5 or more random tests: `-100`.
- Fails the test: `deduction up to 20`.
- 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 `0.0`.**

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.