- 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
  ```

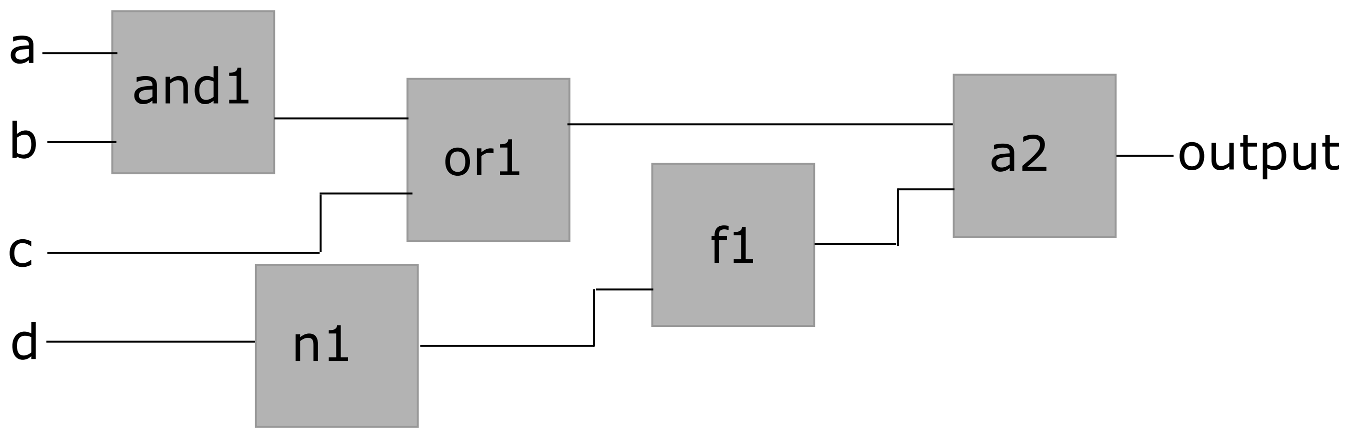

Figure 1: Example Logic Circuit

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.

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