CSE 241 Programming Assignment 4

NOTE

The weight-point of this assignment is two times the weight-point of a regular programming assignment.

DUE

May 19, 2021, 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.

This assignment is about implementing and testing classes for a logic circuit simulator.

- Your program reads two files:
  - `circuit.txt`
  - `input.txt`
- According to content in `circuit.txt`, the program **dynamically** creates necessary objects for a logic circuit and evaluates the states listed in `input.txt`.
- Your program prints the output to `stdout`. Each line in `input.txt` is a state. For each state, there should be a line of output printed.

`circuit.txt`

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

```
INPUT
OUTPUT
AND
OR
NOT
FLIPFLOP
DECODER
```

- 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`.
- The second line specifies output labels. Labels are separated by spaces. Example:

```
OUTPUT d1 d2 d3 d4
```

- Here there are `4` outputs are defined. Each has an identifier. `d1`, `d2`, `d3`, `d4`.
- `AND` keyword specifies that there is an `and` gate defined. `AND` keyword follows the identifier for its output and two other identifiers for the inputs. Example:

```
AND gate_A c3 another_id
```

- Here the `and` gate has an output 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 its output and two other identifiers for the inputs. Example:

```
OR gate_B ck id3
```

- Here the `or` gate has an output 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 a `not` gate defined. `NOT` keyword follows the identifier for its output and one other identifier for its input. Example:

```
NOT gate_C c5
```

- Here the `not` gate has an output 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 a `flip-flop` gate defined. `FLIPFLOP` keyword follows the identifier for its output and one other identifier for its input. Example:

```
FLIPFLOP gate_F c6
```

- Here the `flip-flop` gate has an output identified by the string `gate_F`. Its input is identified by `c6`.
- `DECODER` keyword specifies that there is a `decoder` gate defined. `DECODER` keyword follows the identifiers for its outputs(o1, o2, o3, o4) and identifiers for its inputs(a1, a2). Example:

```
DECODER d1 d2 d3 d4 g1 another_identifier
```

- Here the `decoder` gate has outputs identified by strings `o1`, `o2`, `o3`, `o4`. Its inputs are identified by `g1` and `another_identifier`.

`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
OUTPUT d1 d2 d3 d4
AND and1 a b
OR or1 and1 c
NOT n1 d
FLIPFLOP f1 n1
AND a2 or1 f1
DECODER d1 d2 d3 d4 a2 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 output of your program should be:

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

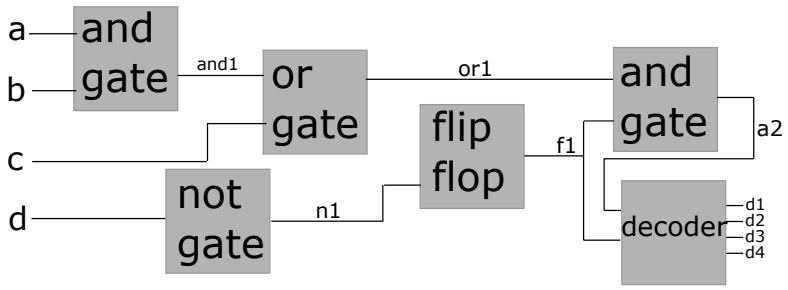

Figure 1: Example Logic Circuit

Remarks

- Each identifier is unique. You can assume that their length is limited to 15 characters. You can use `std::string`.
- There won't be any errors in the files.
- You have to use `dynamic memory allocation` and C++ classes.
- You have use inheritance and polymorphism. Find a class hierarchy and try to use abstraction.
- You are not allowed to used `std::vectors`.
- You cannot use components which are not covered in class. (such as templates etc..)

Turn in:

- Source code of a complete C++ program and a suitable makefile. You should use c++11 standard. Your code will be tested in a linux-gcc environment.
- 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 cannot 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

- DECODER

a1	a2	d1	d2	d3	d4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Late Submission

- Not accepted.

Grading (Tentative)

- `Max Grade` : `100`.
- Multiple tests(at least `5`) will be performed.

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

- Do **NOT** use hard-coded values. If you use you will loose `10pts`.
- 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`.
- Inefficient implementation: `-20`.
- 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: `-10`. (Comments are in English. Turkish comments are not accepted).
- 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: `-30`.
- Submission includes files other than the expected: `-10`.
- Submission does not follow the file naming convention: `-10`.
- Sharing or inheriting code: `-200`.