CSE 102 Programming Assignment 6 (200 points. Double Assignment)

If you complete this assignment you may get 200 points. Not submitting will be considered as "not submitting 2 assignments".

DUE

December 19, 2024, 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.

You are going to write a complete C program which implements the following functionality:

- your program reads two files:
  - `circuit.txt`
  - `input.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 `stdout`. After each output there should be a newline.

`circuit.txt`

This file describes a logic circuit. Below is the file format:

```
GATE <gate_type> <gate_name>
.
.
.
GATE <gate_type> <gate_name>
GATE <gate_type> <gate_name>
CONNECTION <from_gate> <to_gate>
.
.
.
CONNECTION <from_gate> <to_gate>
CONNECTION <from_gate> <to_gate>
```

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

```
GATE
CONNECTION
```

`GATE` line describes a logic gate instance in the circuit. The first word after the `GATE` keywords is the type of the gate. Possible types:

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

The second word is the name of the gate. Names are unique in the circuit.

`INPUT` type represents an input in the circuit. `OUTPUT` type represents an output in the circuit. `AND` type represents an `and` function. It takes 2 or more inputs. It has one output. The maximum input count is `10`. `OR` type represents an `or` function. It takes 2 or more inputs. It has one output. The maximum input count is `10`. `NOT` type represents a `not` function. It takes 1 input. It has one output. `FLIPFLOP` type represents a flipflop gate It takes 1 input. It has one output.

`CONNECTION` line describes a connection between two gates. The first word after the `CONNECTION` keywords is the name of the gate which creates the output. The second word after the `CONNECTION` keywords is the name of the gate which receives the input.

`input.txt`

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

```
1011
0111
0010
1001
```

- No gaps between the bits.
- These values will be assigned to `INPUT` gates in the order they are defined.
- The number of bits in a line will match with the number of `INPUT` gates defined in the circuit.

Example:

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

```
GATE INPUT a
GATE INPUT b
GATE INPUT c
GATE INPUT d
GATE AND and1
GATE OR or1
CONNECTION a and1
CONNECTION b and1
CONNECTION and1 or1
CONNECTION c or1
GATE NOT n1
CONNECTION d n1
GATE FLIPFLOP f1
CONNECTION n1 f1
GATE AND and2
CONNECTION or1 and2
CONNECTION f1 and2
GATE OUTPUT o
CONNECTION and2 o
```

- `input.txt` has the following content:

```
1101
1010
1110
```

- 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 `INPUT` gates in the order they defined in `circuit.txt`. According to the truth tables, outputs of gates are calculated. `OUTPUT` gates outputs are printed in the order they are defined in `circuit.txt`.
- For the input.txt given, the output of your program should be(There is only one `OUTPUT` gate):

```
0
1
0
```

- If there are 2 or more output gates in the circuit the output will show the outputs of the gates in their defined order. (No gaps in between)
- Example output of a circuit with 3 outputs. The circuit was simulated 4 times (each line is the result of one simulation):

```
101
111
100
```

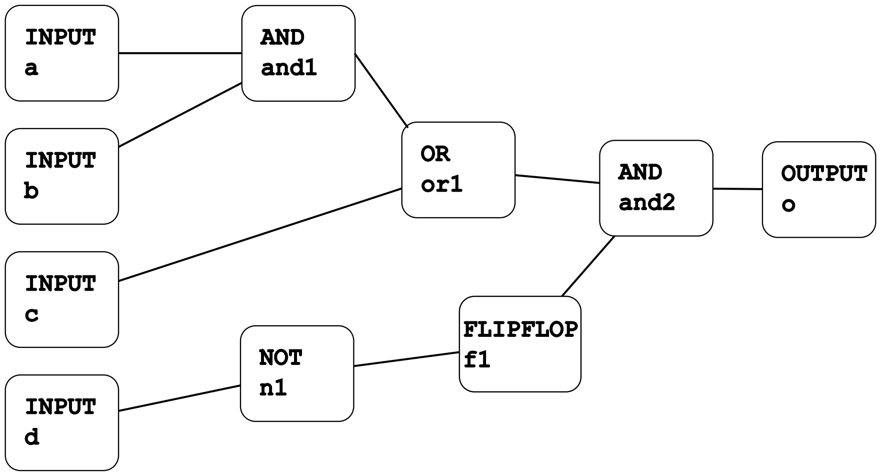

Figure 1: Example Logic Circuit

Remarks

- The number of gates is not known. There is no limit.
- Each identifier is **unique**. Max length is 20.
- There won't be any errors in the files.
- You have to use `dynamical memory allocation` and `struct`.
- You are encouraged to use function pointers.
- You cannot repeatedly read `circuit.txt`. If you do, you will get `0`.
- You cannot **repeatedly** search through arrays of gates and try to figure out the connections. Once you store a connection information, unaffected from the number of gates defined you should immediately reach the related gates. For this, you have to store the addresses of connected gates in the related gates.

- You cannot save `circuit.txt` content to an array and repeatedly use it. If you do, you will get `0`.
- `AND` and `OR` gates have at least 2 inputs(They can have more than 2. The maximum number of inputs is 10 for these gates.)
- A `CONNECTION` is defined after the required `GATE`.
- There won't be loops in the circuit.
- Output of a gate can connect to more than one gate.
- There may be more than one `FLIPFLOP`. (Each flipflop will have its own `former_out`. They don't share the same `former_out` value)
- `former_out` of each flipflop is preserved for successive simulations. Before the first simulation, every `former_out` will be `0`.
- **If you don't create a linked data structure(a graph) and use it in order to evaluate the circuit, your submission will not be accepted.**
- You cannot create arrays to store all the gate identifiers. But you can have an array to store pointers to gate structs.
- You can create a pointer array which stores the address of each gate struct created.

- Write comments in your code.
- If your code does not compile you will get `0`
- Do not share your code with your classmates.
- Do not print anything other than the expected output.
- You cannot use anything which is not covered in class.
- Do not submit any of the files you used for testing.
- One possible struct to represent a gate can be as follows:

```
struct gate
{
    int type;               /* type of the gate*/
    char* name[20];         /* name of the gate*/
    struct gate* input_gates[10]; /* partially_filled array. each element points to a gate
                                  /* which is connected to this gate as an input */
    int number_of_inputs;

    int output              /* stores the current output. this may be useful if the gate is
    ↪  connected to more than one gate.*/
    int former_out;         /* only for flipflop*/
    /*You can add other components if necessary.*/
    short evaluated;  /* a flag to store the state of evaluation */
                      /* if this is true, we just return the output */
                      /* we don't have to evaluate again for the current simulation */
                      /* before starting the next simulation, you should reset this */
    int (*characteristic_function)(int*, int); /* a function pointer which points to
                                                a function defining the logic of the gate */

}
```

- Lets try to define two functions for the `AND` and `OR` gates.

```
/* returns a 1 or 0 */
int and_function(int* inputs, int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        if(!inputs[i])
            return 0;
    }
    return 1;
}

int or_function(int* inputs, int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        if(inputs[i])
            return 1;
    }
    return 0;

}
```

- If we try to use the `struct` suggested above:

```
/* define gates */
/* These are not dynamically allocated */
/* You have to dynamically allocate gates in the actual solution */

struct gate g1;
g1.number_of_inputs = 2;
/* Lets make this gate an `AND` gate */
```

```
g1.characteristic_function = and_function;

struct gate g2;
g1.number_of_inputs = 2;
/* Lets make this gate an `OR` gate */
g2.characteristic_function = or_function;

struct gate g3;
g3.number_of_inputs = 2;
/* Lets make this gate an `AND` gate */
g3.characteristic_function = and_function;

/* Lets connect g2 and g3 to g1 */

g1.input_gates[0] = &g2;  /* The output of g2 will the first input of g1 */
g1.input_gates[1] = &g3;  /* The output of g3 will the second input of g1 */

/* if we want to calculate the output of g1 */
/* we can just call the characteristic function of g1 */
/* But, before doing this, we have to know the input values */
int output_g2 = g2.characteristic_function(/* you have to provide input values array
                                            and the number of inputs */);
int output_g3 = g3.characteristic_function(/* you have to provide input values array
                                            and the number of inputs */);
int inputs_g1[2];
inputs_g1[0] = output_g2;
inputs_g1[1] = output_g3;
int output_g1 = g1.characteristic_function(inputs_g1, 2);

/* As you can see, we can handle this sequence of function calls with recursion. */
/* Start with the output gates and repeatedly call
characteristic functions of gates until you reach input gates. */
```

Pseudo_code (A suggestion):

- You don't have to follow the algorithm below. It is just suggested.

```
Main:

- Allocate gates pointer array for `input_gates`. (pre-allocate for 10 pointers)
- Allocate gates pointer array for `output_gates`. (pre-allocate for 10 pointers)
- Allocate gates pointer array for `other_gates`. (pre-allocate for 10 pointers)

- open circuit.txt
- Loop until EOF:
    - read a line. extract 3 strings
    - if the first string is `GATE`
    - call `create_gate`
    - if the first string is `CONNECTION`
    - call `create_connection`
- close circuit.txt
- open input.txt
- Loop unti`l EOF:
    - call `reset_gates`
    - read a line. extract 1s and 0s in the given order
    - define input_values
    - store the extracted values in `input_values`.
    - Loop until the end of `input_values`.
        - assign an input value to an input gate in `input_gates`.
    - define `output_values`
    - Loop until the end of `output_gates`
        - call `evaluate` for a gate in `output_gates`
        - store return value in output_values`
    - print `output_gates`
- close input.txt

reset_gates:
- Loop input_gates, other_gates and output_gates:
    set `evaluated` to FALSE


evaluate:
 - if `evaluated` return `output`
 - if type `INPUT` return input_value assigned
 - define input_values
 - Loop until the end of input_gates:
    - call evaluate for an input gate
    - store the return value in `input_values`
 - set `output` to the return value of characteristic_function which takes `input_values`
 - set `evaluated` to TRUE
 - return `output`


create_gate:
 - dynamically allocate a gate struct
 - if it is an input gate, store the address in `input_gates`
    - if `input_gates` is full, reallocate it with twice the previous size
    - copy the contents to the new allocation
    - free the old allocation
 - if it is an output gate, store the address in `output_gates`
    - if `output_gates` is full, reallocate it with twice the previous size
    - copy the contents to the new allocation
    - free the old allocation
 - if it is another type of gate, store the address in `other_gates`
    - if `other_gates` is full, reallocate it with twice the previous size
    - copy the contents to the new allocation
    - free the old allocation

 - Store the name of the gate in the struct.


create_connection:
 - Loop until the end of `output_gates`:
    - find a <to_gate> by trying to match the name component.
    - if <to_gate> gate is not found in `output_gates:
        - Loop until the end of `other_gates`:
            - find a <to_gate> by trying to match the name component.
 - <to_gate> string is found in struct instance `to_gate`
 - Loop until the end of `input_gates`:
    - find a <from_gate> by trying to match the name component.
    - if <from_gate> gate is not found in `input_gates:
        - Loop until the end of `other_gates`:
            - find a <from_gate> by trying to match the name component.
 - <from_gate> string is found in struct instance `from_gate`
 - store the address of `from_gate` in the `input_gates` array of `to_gate`
```

Turn in:

- Source code of a complete C program. Name of the file should be in this format: `<full_name>_PA6.c`.
- Example: `guthrie_govan_PA6.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>_PA6.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:

- 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` : `200`.

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

- No submission: `-200`.
- Compile errors: `-200`.
- Irrelevant code: `-200`.
- Major parts are missing: `-200`.
- Unnecessarily long code: `-30`.
- Using language elements and libraries which are not allowed: `-200`.
- Not caring about the structure and efficiency: `-100`. (avoid using hard-coded values).
- Significant number of compiler warnings: `-10`.
- Not commented enough: `-10`. (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 `circuit.txt`: `-150`.
- Fails at reading `input.txt`: `-150`.
- The result is wrong: `-100`.
- Output format is wrong: `-30`. (Be careful with spacing)
- Infinite loop or recursion: `-150`.
- Prints anything extra: `-30`.
- Unwanted chars and spaces in the output: `-30`.
- Submission includes files other than the expected: `-10`.
- Submission does not follow the file naming convention: `-10`.
- Sharing or inheriting code: `-400`.
- 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.