ASSIGNMENT - 1

Design a preprocessor on top of the base compiler infrastructure provided to you.

Preprocessing is a stage in the pipeline before lexing begins. Preprocessors take the source code as input and output code after processing any preprocessor directives inside the code. An example of this would be the `#include` preprocessor directive in C code, which the preprocessor replaces with the contents of the file specified as an argument to the directive. You should use ***Flex*** for this purpose.

The specifications of the required preprocessor are as follows:

- The preprocessor performs text substitution.
- Directive #def: `#def IDENTIFIER text`. This directive is similar to the `#define` directive in C. After a `#def` directive, the preprocessor replaces all instances of `IDENTIFIER` in the source code with `text`. Examples regarding all possible variations of this directive are mentioned below.
- Directive #undef: `#undef IDENTIFIER`. Undefines the defined IDENTIFIER macro. The macro is not expanded after this directive. If IDENTIFIER is already undefined before this directive, this directive is ignored.
- It also matches and removes any comments in the code. There are two types of comments, single line comments (starting with `//`) and multiline comments (enclosed within `/*` and `*/`). A multiline comment ends at the first `*/` instance, i.e. nesting multiline comments is not permitted.

Modify the driver code in `main()` to fully preprocess the code before any other stage of the compiler, then pass the processed code as input to the other stages.

Example #def macros:

1) #def TEN 10
    - Simplest macro possible, expands to the body.

2) #def STATEMENTS let a = 5; \
   let x = a * 16; \
   dbg a + b * x
    - Multiline macro: STATEMENTS expands to the three statements wherever it is used.
    - Example usage:
      let b = 10;
      STATEMENTS;

      The above two lines will be preprocessed into:
      let b = 10;
      let a = 5;

```
let x = a * 16;
dbg a + b * x;
```

3) #def NOBODY
    - Macro without a body. By default it expands to 1, i.e. it is the same as `#def NOBODY 1`.

4) #def ABC 4
   #def DEF ABC + 5
   #def GHI ABC * DEF

   dbg GHI;
    - Nested macros: Nested macros should be supported. The above dbg statement should expand to:
      dbg 4 * 4 + 5;
    - Circular dependencies should cause the preprocessor to throw an error.

5) #def ABC 1
   dbg ABC;
   #def ABC 15
   dbg ABC;
    - This code expands to:
      dbg 1;
      dbg 15;

6) #def ABC 2
   #undef ABC
   dbg ABC;
    - Throws an error because ABC is not expanded by the preprocessor, and the parser sees no variable `ABC` declared.

NOTE: Macros should only be expanded in statements that are below them. For example:
dbg ABC;
#def ABC 5

Should not be expanded to `dbg 5;`