Experimental Report of lexical analyzer

1. Motivation/Aim

The purpose of this experiment is to generate the corresponding LR1 form from the given production. A code that has been lexically analyzed is then syntactically analyzed through the LR1 form. In turn, the correctness of the code can be verified. While completing the basic requirements, implementing some basic error message output needs to be included.

1. Content description

In the experimental code, it consists of three main parts.

The first part is a simplified lexical analyzer. This analyzer ignores the values corresponding to tokens and keeps only the tokens.

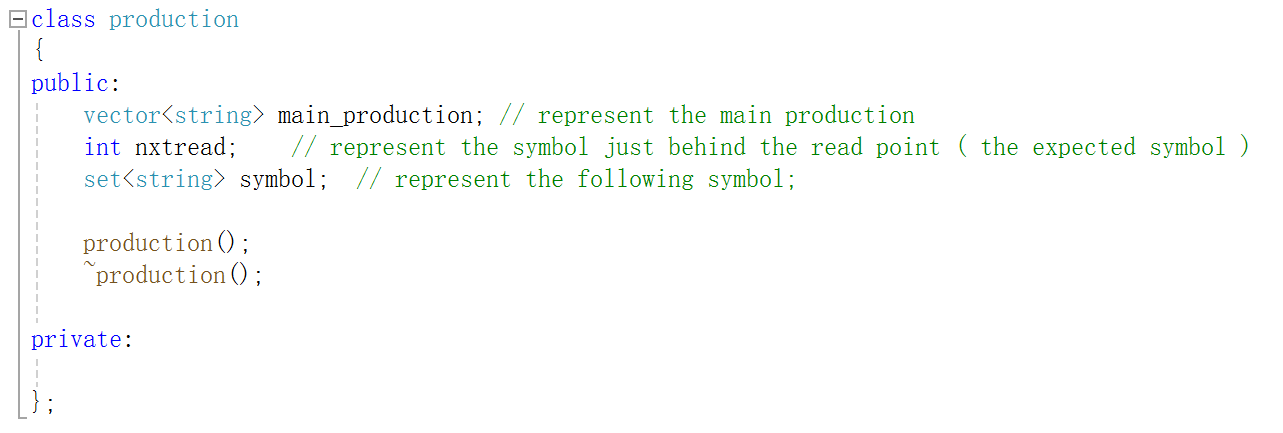
The second part is a generator for LR1 forms. This generator converts the input productions into LR1 forms. This form can be used for subsequent generative syntax analysis.

The third part is a grammar analyzer. The third part is a syntax analyzer that contains a file read and a stack, which enables the generation of syntax trees with the help of LR1 tables.

1. Ideas/Methods and Description of important Data Structures

The first part to be realized is the generation of the LR1 form through productions. In our experiment, we use vector<string> to store productions. Production stored in this form maintains the order of the symbols, while making it easy to extract each part of the symbols.

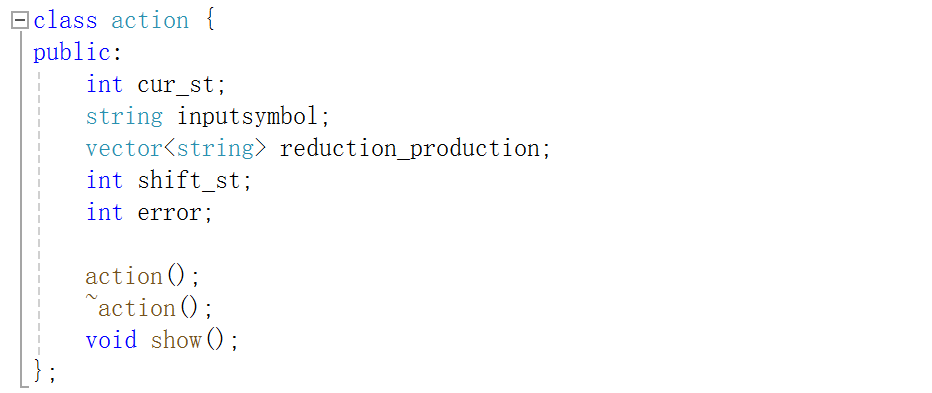
Inside the table in LR1, each production also contains information about the read head and the symbols that will be present just behind the production. So, we use a production class to describe the collection of these three pieces of information.



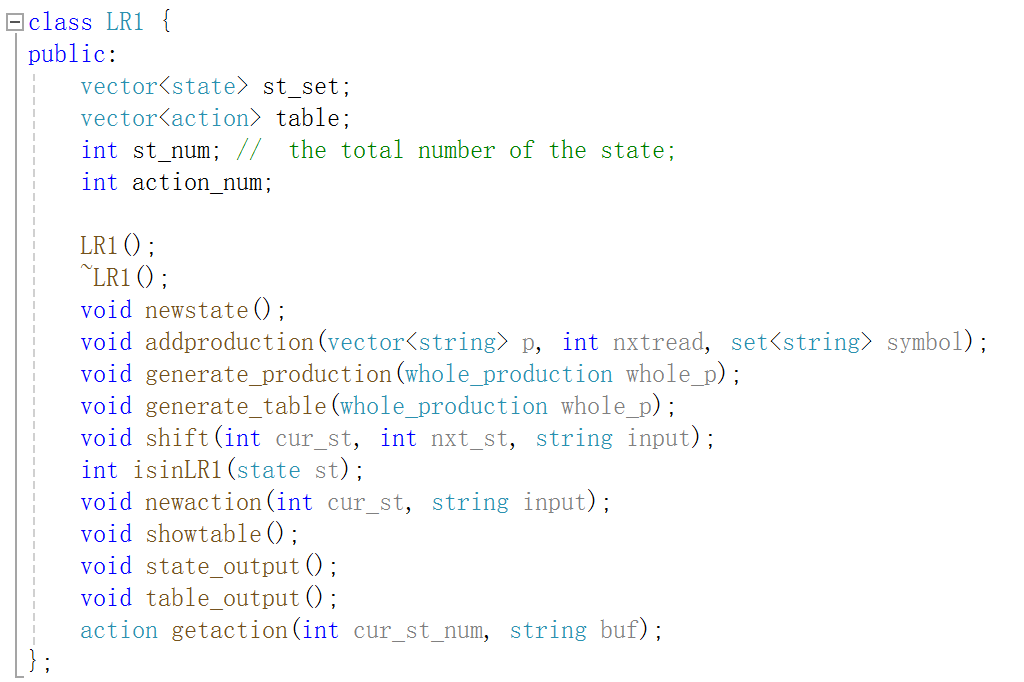
The main forms of production are stored in main\_production. The location of the read head is indicated in nxtread. The symbols in this position are the symbols that are possible to read when the state is transferred. Symbol represents the possible following symbol just behind the main production.

The next step is to create the state data structure. The data for a state contains a collection of productions and a serial number for the state. The collection of productions is also implemented as a vector. Although unorganized production is not conducive to later state comparisons, it is also a good option.

Similarly, we need a class that describes the LR1 form. The data in the LR1 form is not just a collection of states, but also a collection of actions. We generally think of only two actions in a given state when there is an input non-terminating symbol. One is a reduction and the other is a shift. The members of the action class have a state, an input, a state to transfer to and a production for reductions.



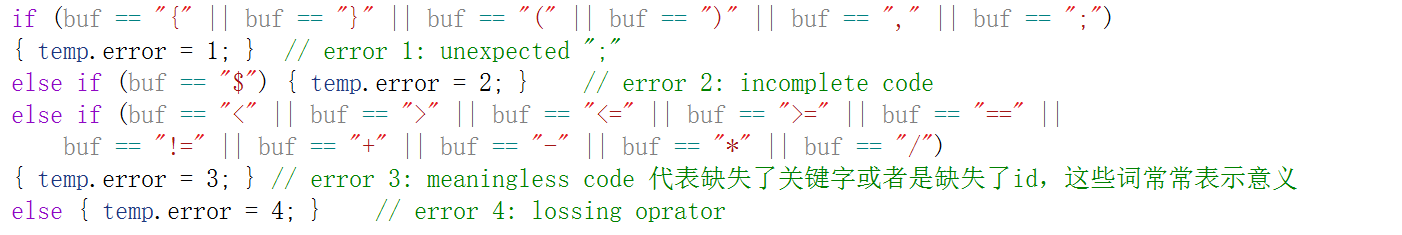
In order to make it easier to generate errors, there is also an error state there, with different values representing different error states. This value is generally 0 and is only used when returning an action to determine if the transfer operation is available in the table. The contents of the LR1 table are shown below.



Construct LR1 table and store actions using the method of solving for table state transfers. The table can then be used to perform a syntactic analysis of the lex file.

We use the LR1 stack class for lex file analysis. The nodes inside this stack class all have two main members, current symbol and the current state. Error handling is included in the process of syntax analysis. During syntax analysis, the corresponding action is returned. When the error status of the action is not 0, the error is handled and the program exits.

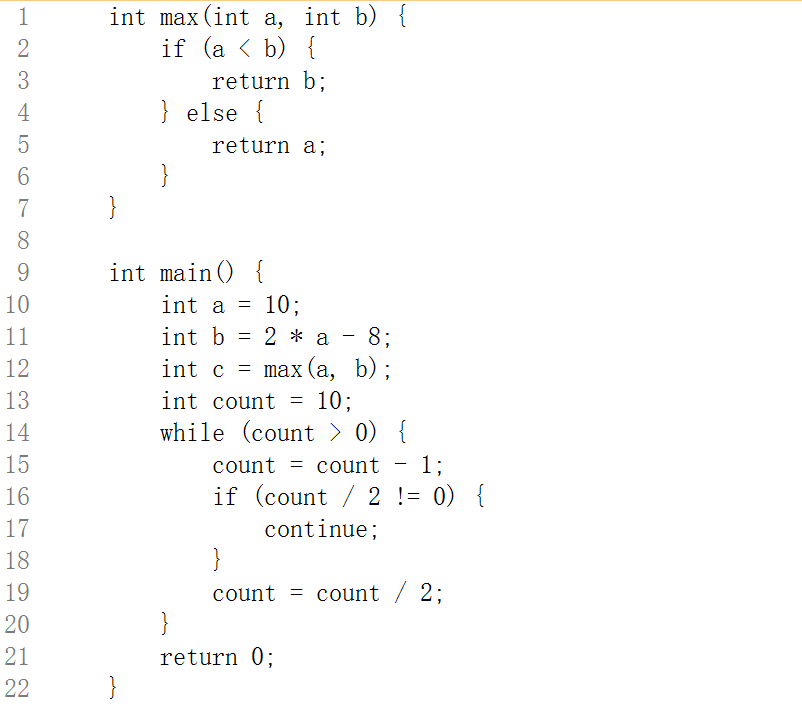
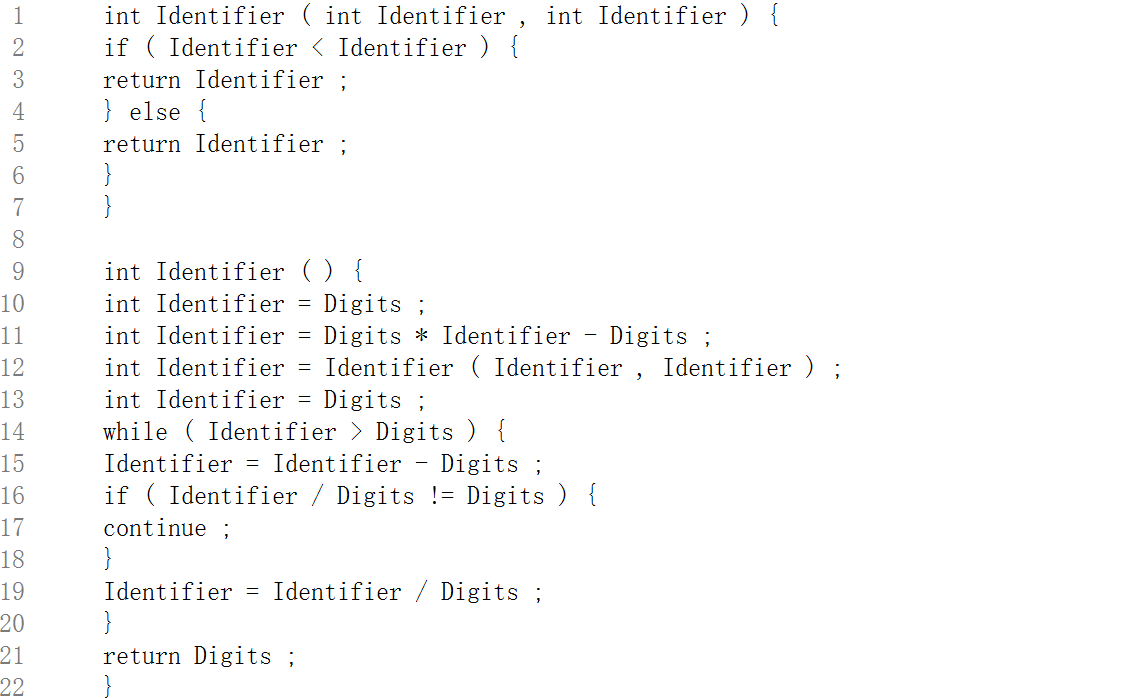
Errors generally consist of four categories, roughly as shown in the figure.



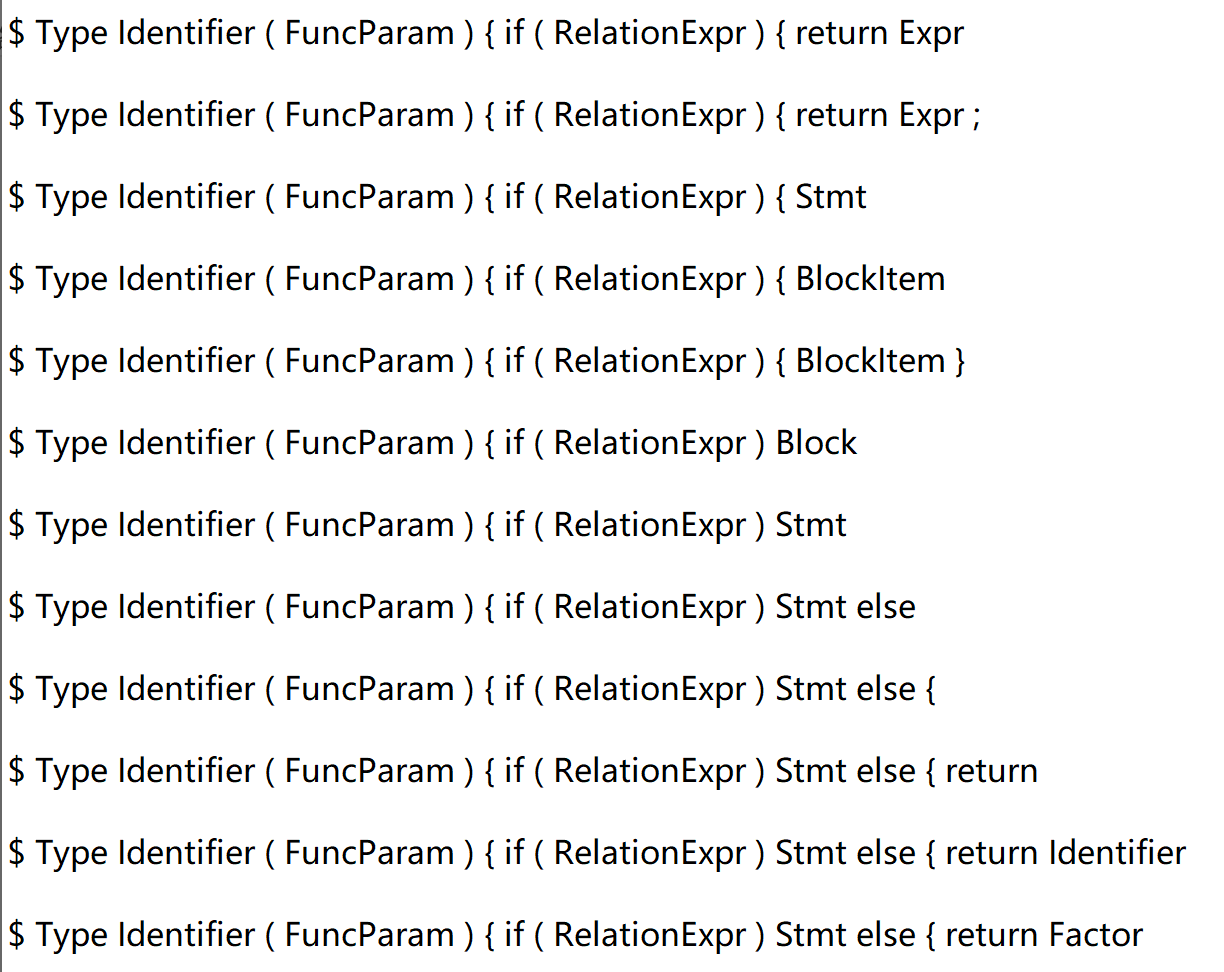
The first category is because we wrote some extra symbols. The second category is because the program is incomplete, which is rare. The third category is because we are missing the id, making the operator read directly. The fourth category is because we are missing operators, making the ids read in directly.

1. Use cases on running

We process the source file using a simplified version of the lexical analyzer to get the lex file.

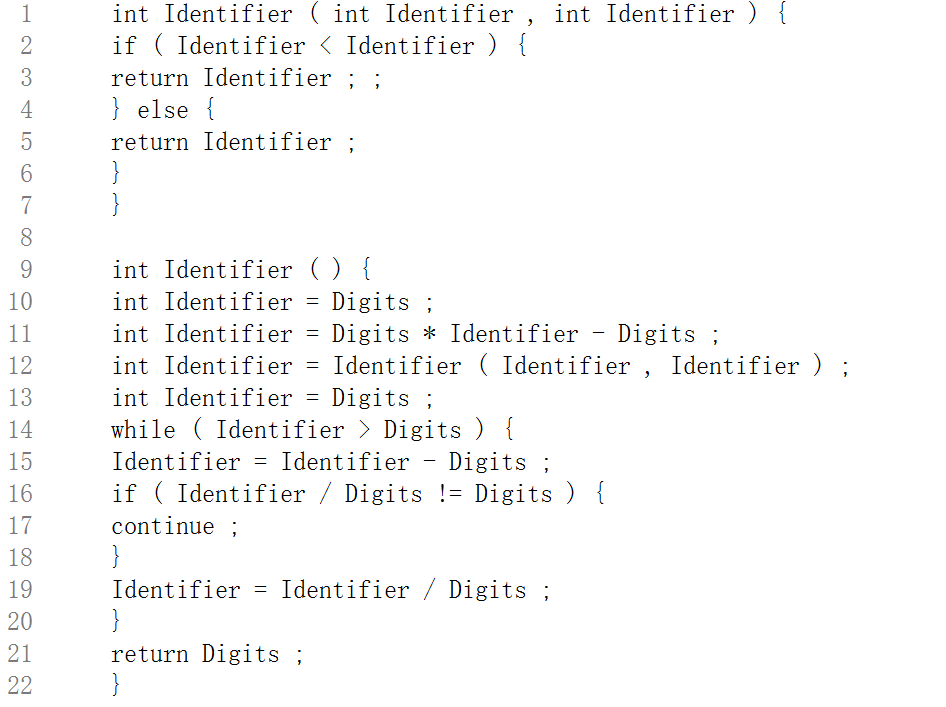
 

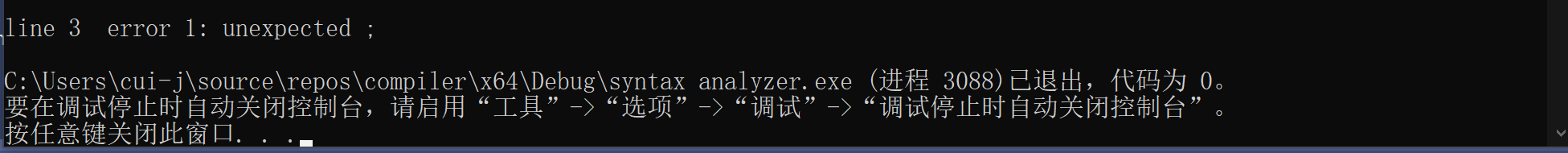
Then we use the production set to process the program. We put him in the appendix of part 7, since there are many productions. With this production set, we get a state machine with 219 states. We also get an LR1 table. All documents can be found in the file. Here we show part of the process.



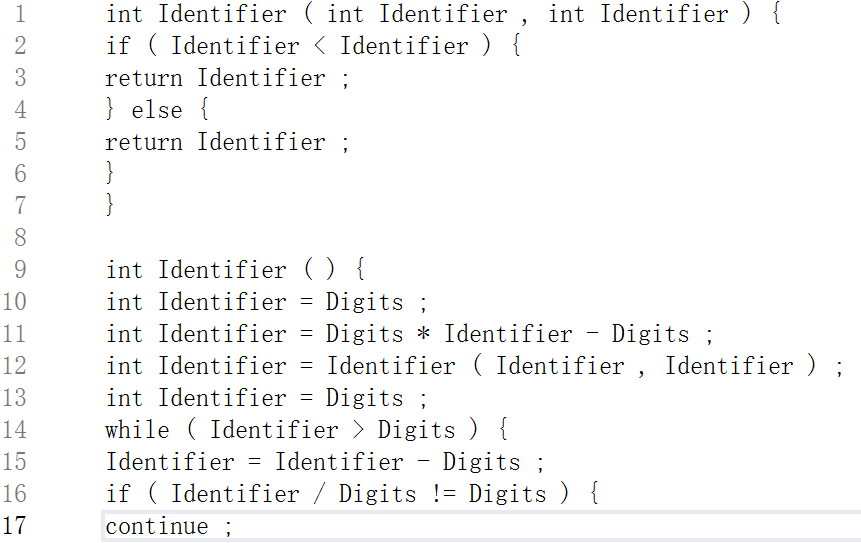
We have also designed four different types of errors which is stored in files named errors 1-4. The error is shown below. When you need to use a different file, you only need to change the code in lines 109-113 of the code.

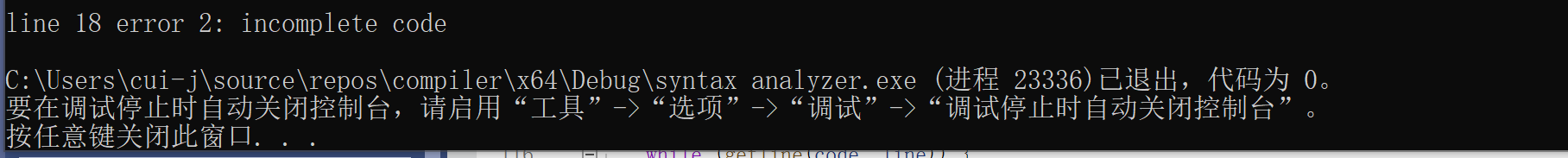
The first mistake was an extra “;”.



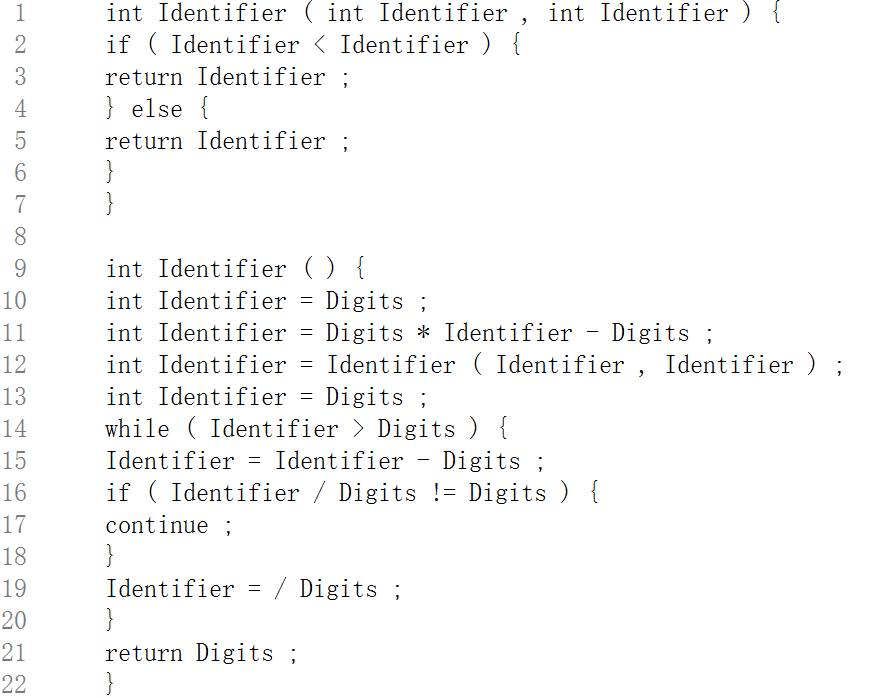


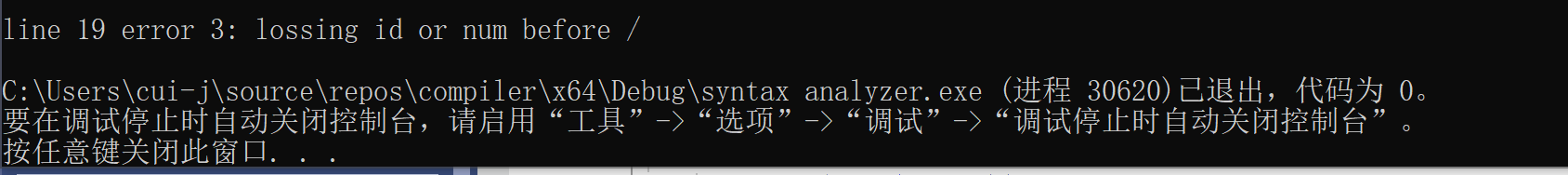
The second error is in complete code.



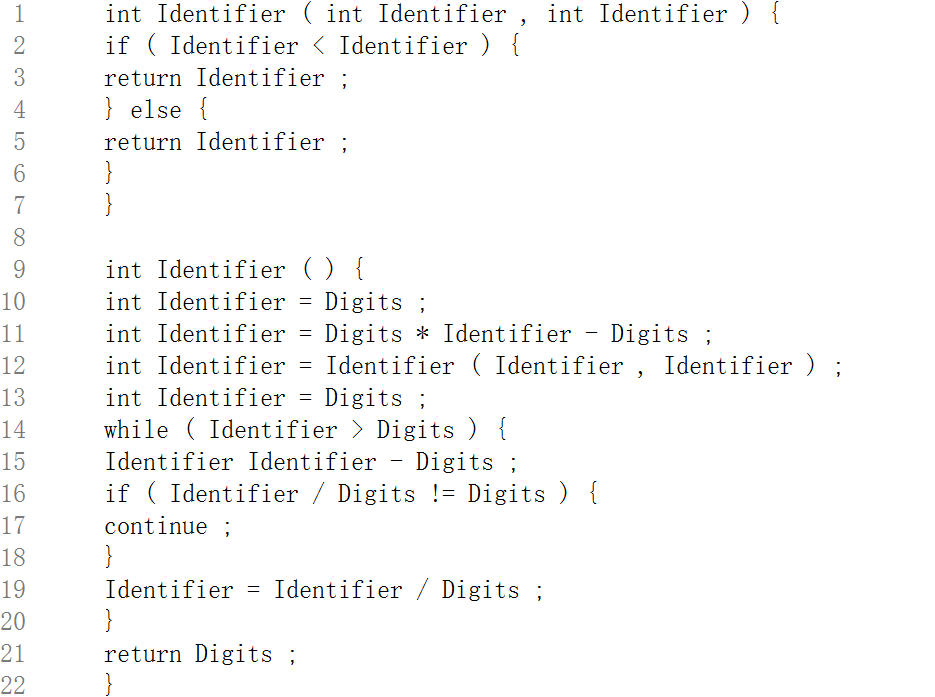


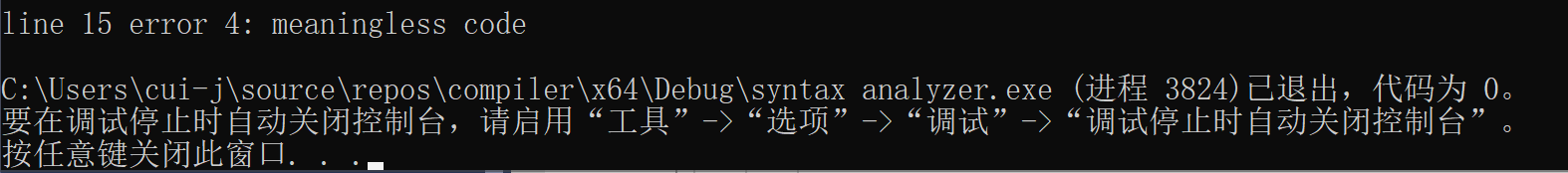
The third error is losing id or num.





The fourth error meaningless code. Because the missing reserved word or id makes the code meaningless.





1. Problems occurred and related solutions

Like the lexical analyzer program exits when it encounters an error. Because there is no way to find a new state when an error is encountered, subsequent code cannot be analyzed.

The second problem that exists is because of the production itself. Errors are not handled very accurately, mainly error 1. The categorization of error types is based on common sense alone. When symbols such as parenthesized semicolons, brackets, etc. are encountered under these productions, there may be a transfer of state as well. This ultimately makes the location where the error was produced change and the type of error change.

This can be done by using a more sophisticated error analyzer. Combine this with the state transfers or reductions that are already in place for error analysis. This might improve the accuracy of the errors.

1. Your feelings and comments

This experiment gave me a deeper understanding of the creation of the LR1 table. At the same time, it allowed me to use LR1 table to solve some syntactic analysis problems. Error analysis has always been a hard problem that has not been realized. This needs to be strengthened.

1. Appendices: Production set

