report for syntax analyzer

This is my report for syntax analyzer, I will show my design proposal here.

■According to grammar, teacher has already give us the syntax parsing table, so what should I consider is how to represent it in my program. Of course, I choose a two-dimensional table for what I must number the non-terminal symbols (To distinguish from terminal symbols, I start to number from 60):

ID	non-terminal symbols
0	ε
60	program
61	stmt
62	compoundstmt
63	stmts
64	ifstmt
65	whilestmt
66	assgstmt
67	boolexpr
68	boolop
69	arithexpr
70	arithexprprime
71	multexpr
72	multexprprime
73	simpleexpr

Grammer:

1:program → compoundstmt

2 3 4 5:stmt → ifstmt | whilestmt | assgstmt | compoundstmt

6:compoundstmt → { stmts }

7 8:stmts → stmt stmts | ε

9;ifstmt → if (boolexpr) then stmt else stmt

10:whilestmt → while (boolexpr) stmt

11:assgstmt → ID = arithexpr ;

12:boolexpr → arithexpr boolop arithexpr

13 14 15 16 17:boolop → < | > | <= | >= | ==

18:arithexpr → multexpr arithexprprime

19 20 21:arithexprprime → + multexpr arithexprprime | - multexpr arithexprprime | ε

22:multexpr → simpleexpr multexprprime

23 24 25multexprprime → * simpleexpr multexprprime | / simpleexpr multexprprime | ε

26 27 28 29 30 31:simpleexpr → ID | NUM | (arithexpr)

(**note**:No.27-30 because there are four kinds of NUM according to my lexical analyzer)

The item in this two-dimensional table is an index of another table----production table. And I also encode the production for convenient with the number of terminal and non-terminal. For example, number of production 1 is 6062 and the number of production 9 is 643013671431613261.

■Pseudocode:

```
//Token[] is the output of the lexical analyzer
Push grammar terminator on stack
Push grammar start symbol on stack
i=0
a=Token[i]
while (stack is not empty && there is still input symbol)
{
   Pop up top item from stack to S
   if(S is terminal symbol)
   {
      if(S==a)
            i++
      else
            error handing
   }
   else
   {
       if (S==’ $’ )
       {
           if (S==a)
                break;
           else
               error handing
       }
       else
       {
           if(T[S,a] is the Candidate production of S)
               push the right symbols of production in reversed order(except ε)
           else
               error handling
       }
    }
    if (stack is not empty || there is still input symbols)
         error handling
}
```

■error handling and output

Usually when stack operations occur, I will put related information about the parse tree to message queue. And when errors occur, I put the error message to the message queue. At last when the whole input is analysed, program will output the message.