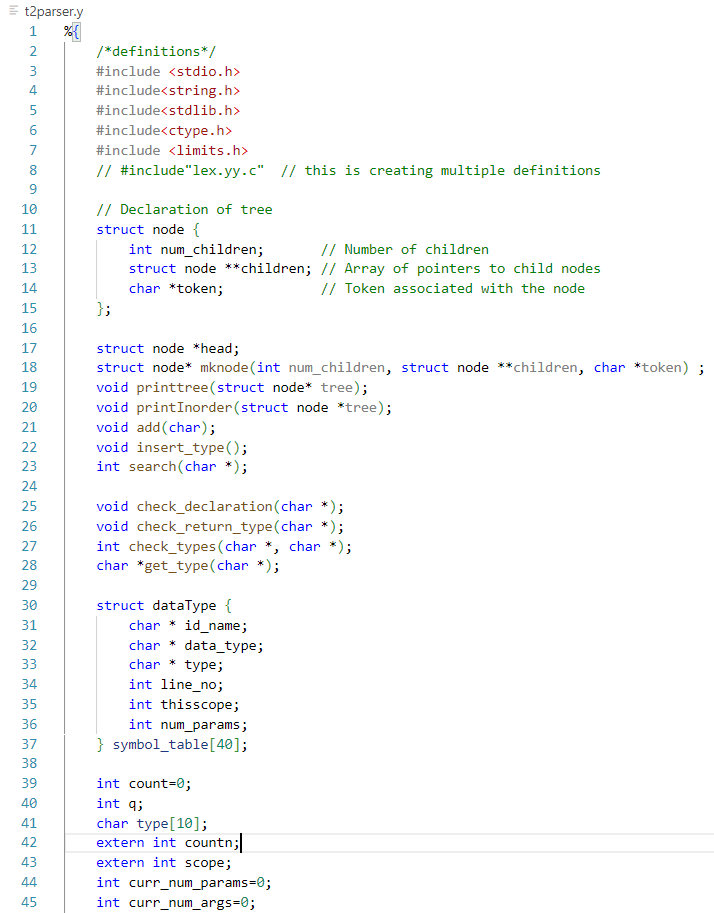
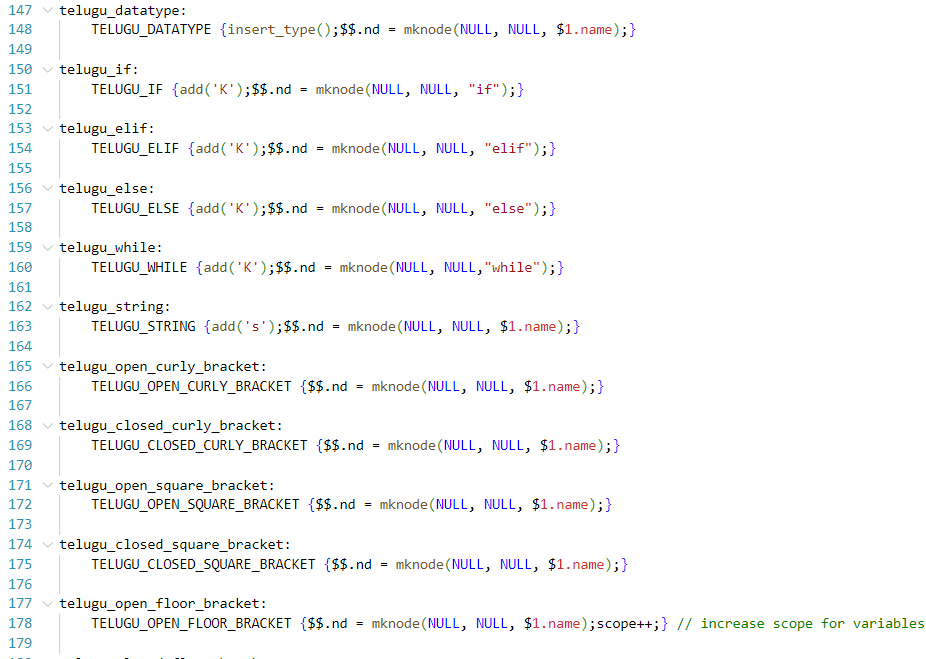
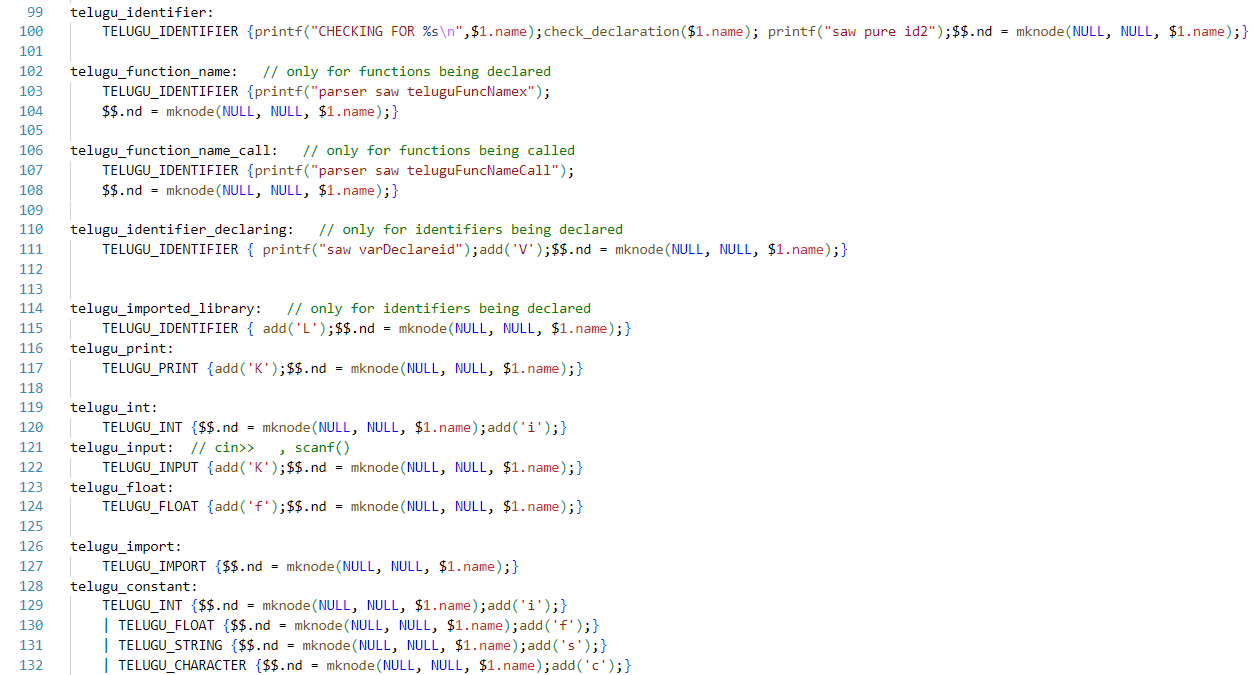
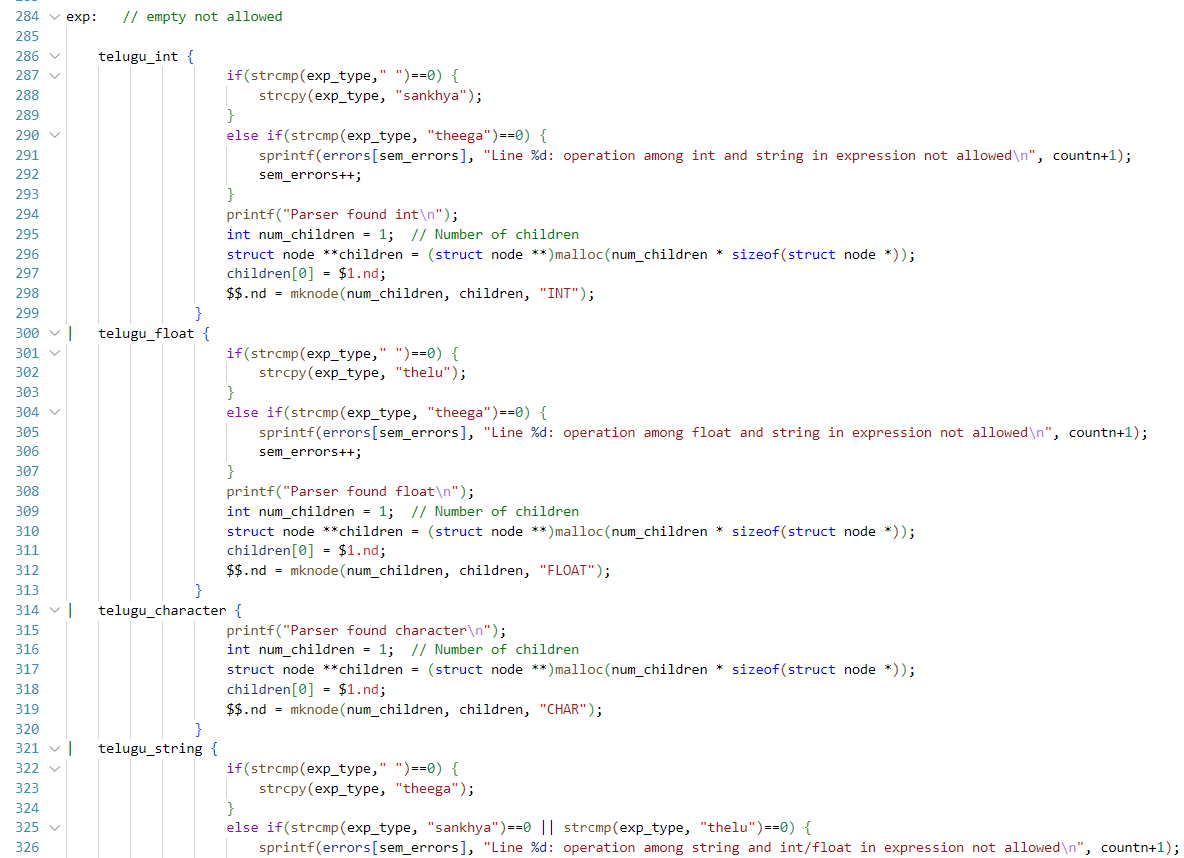
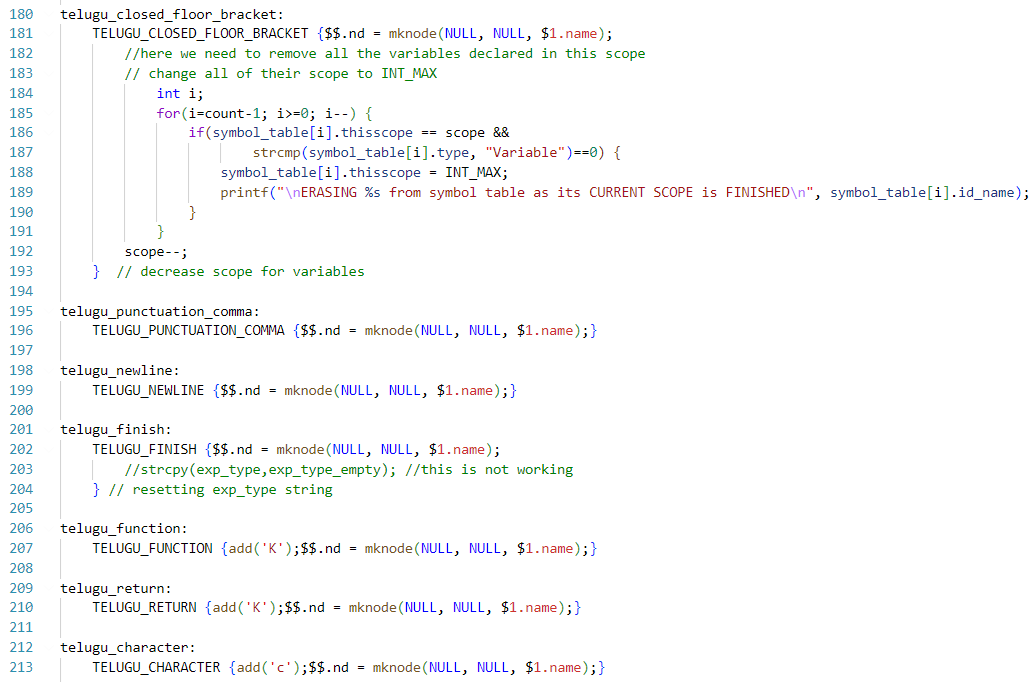
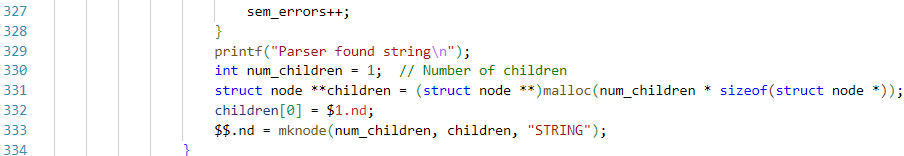
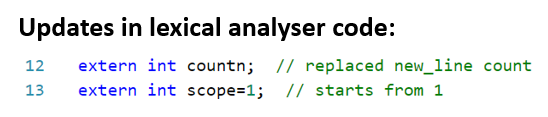
**Updates in parser code:**

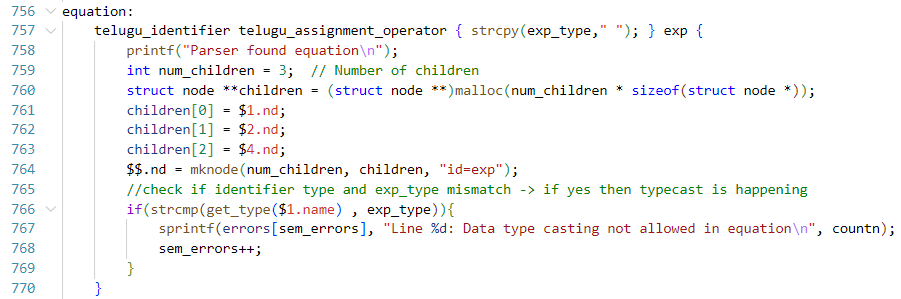
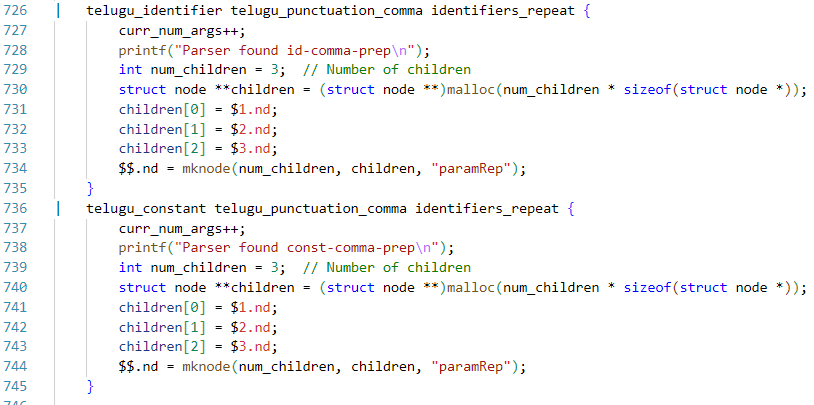
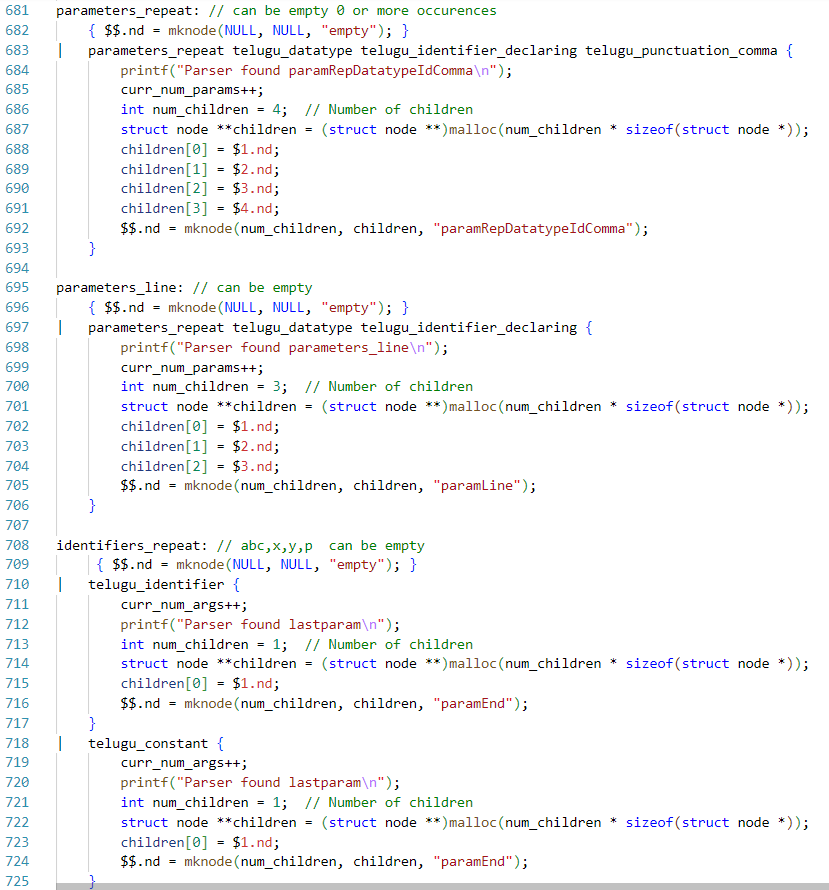
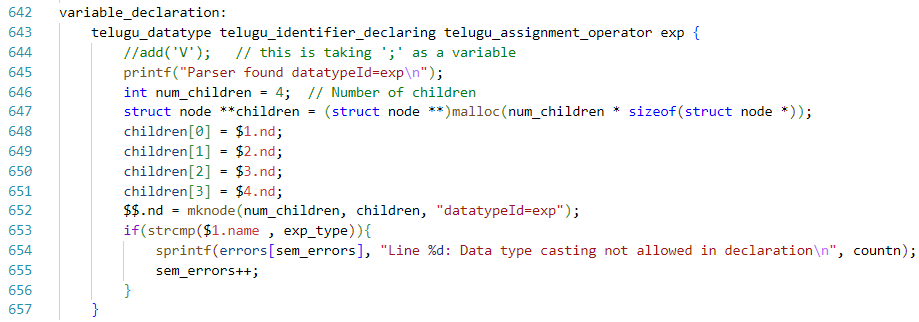


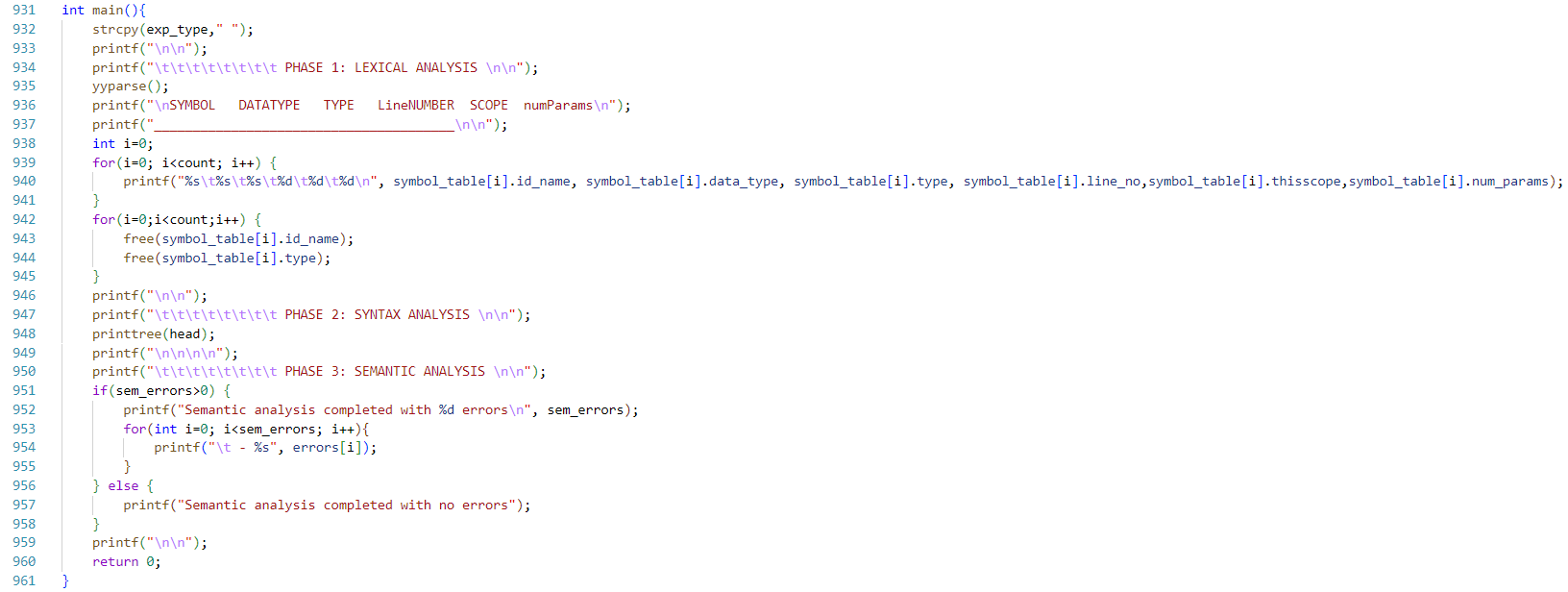


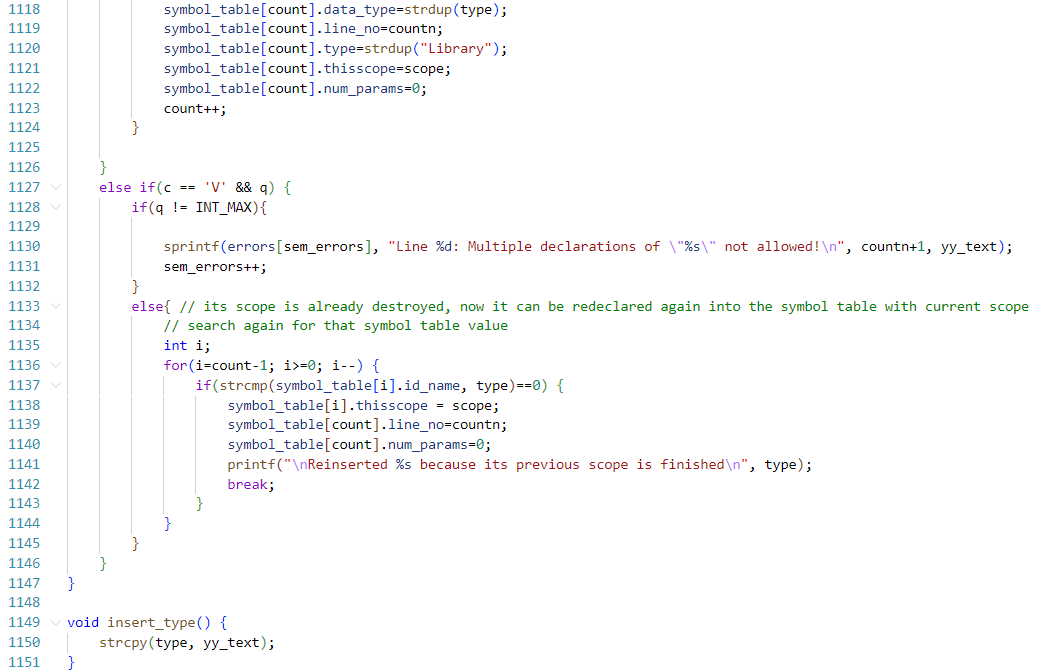
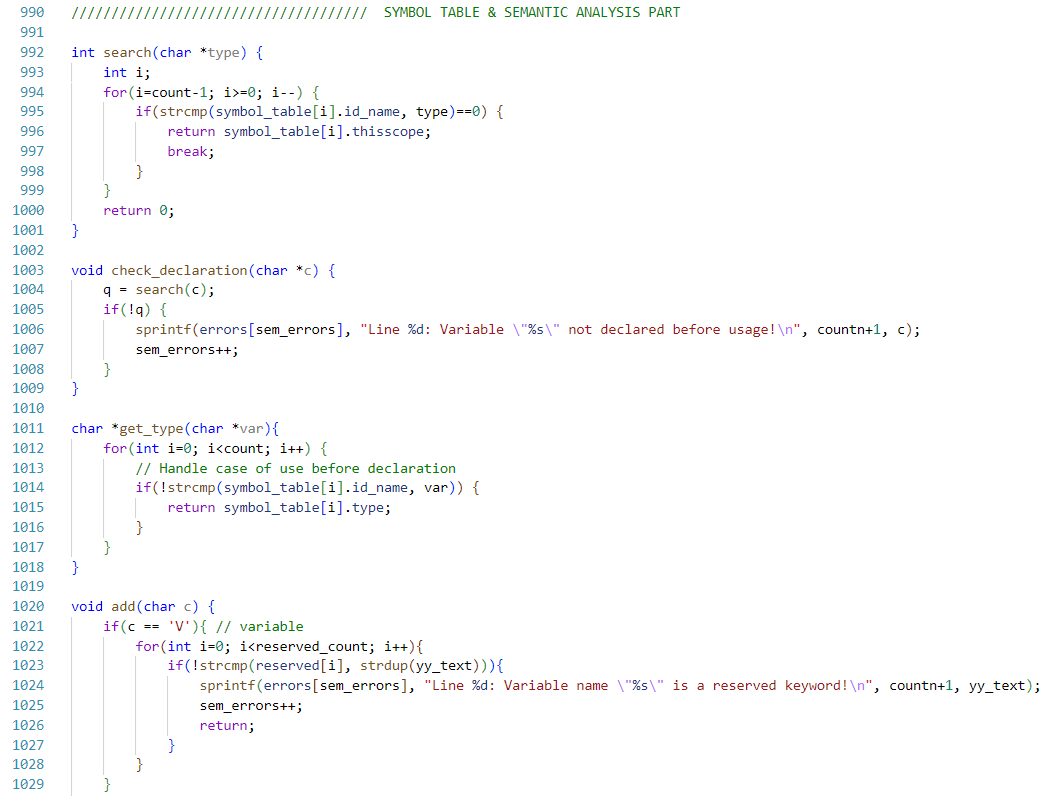




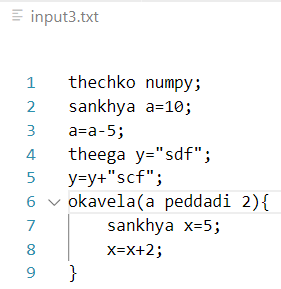




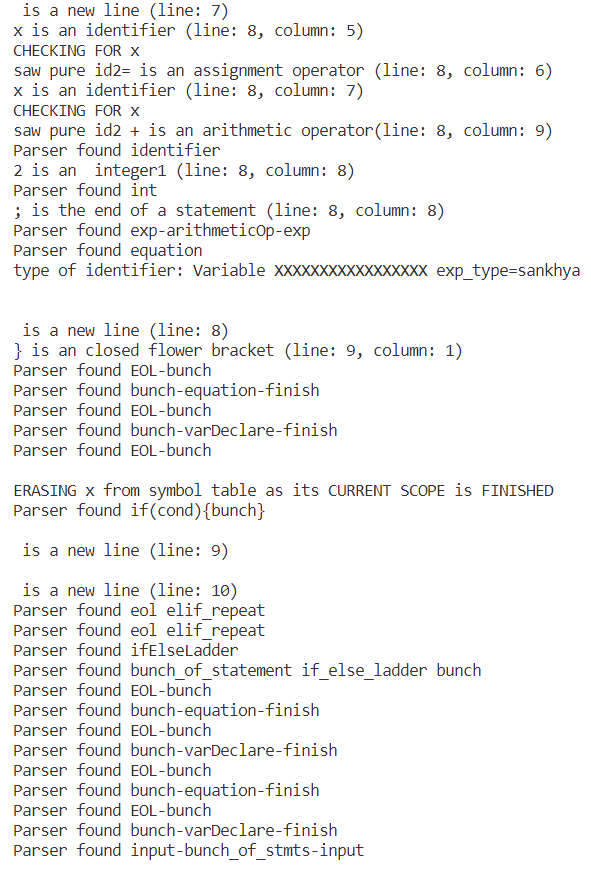
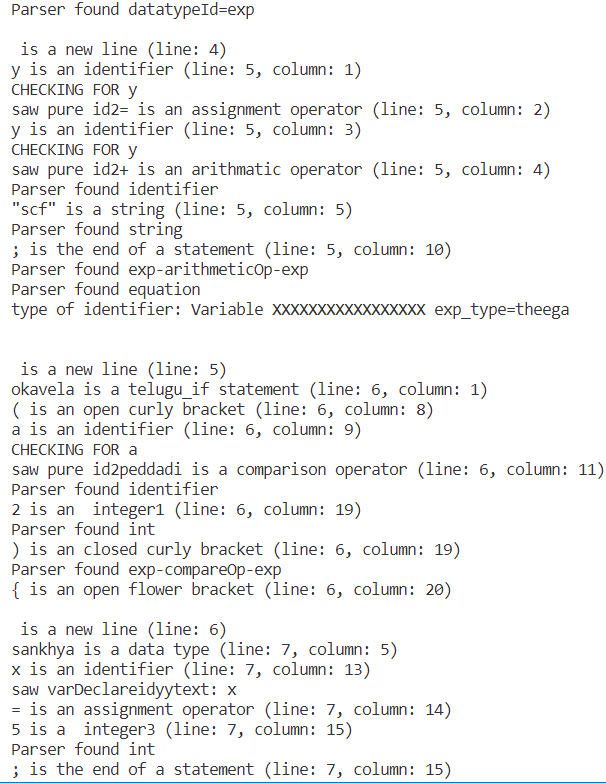
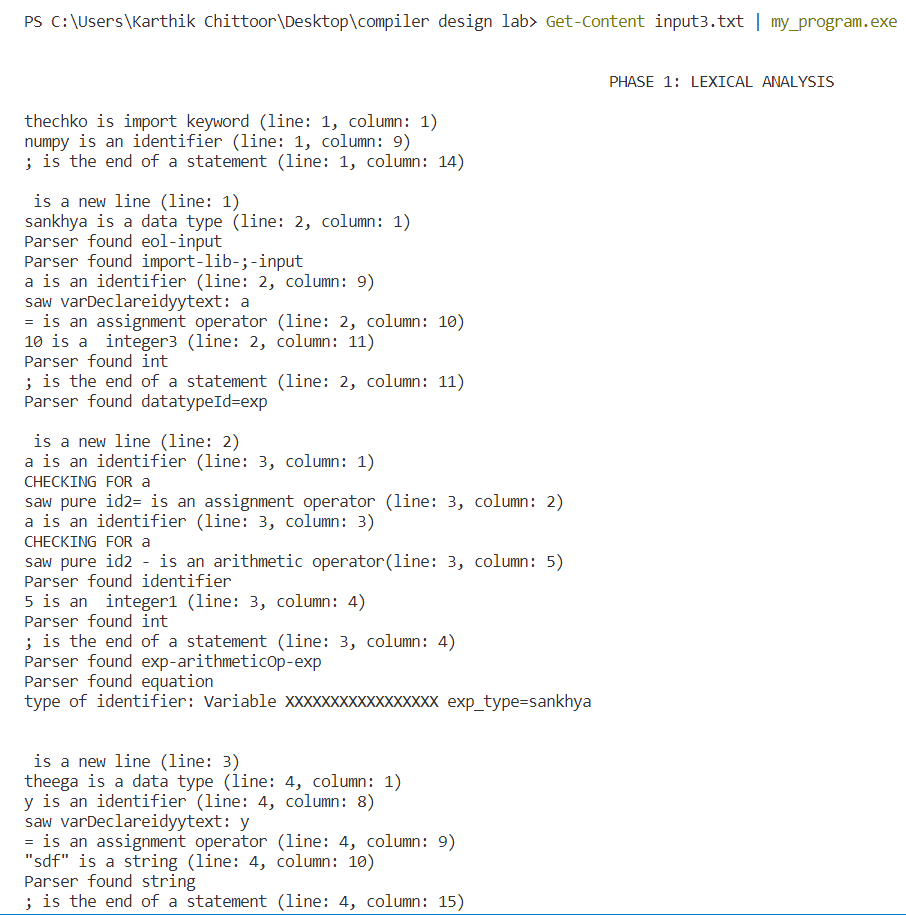




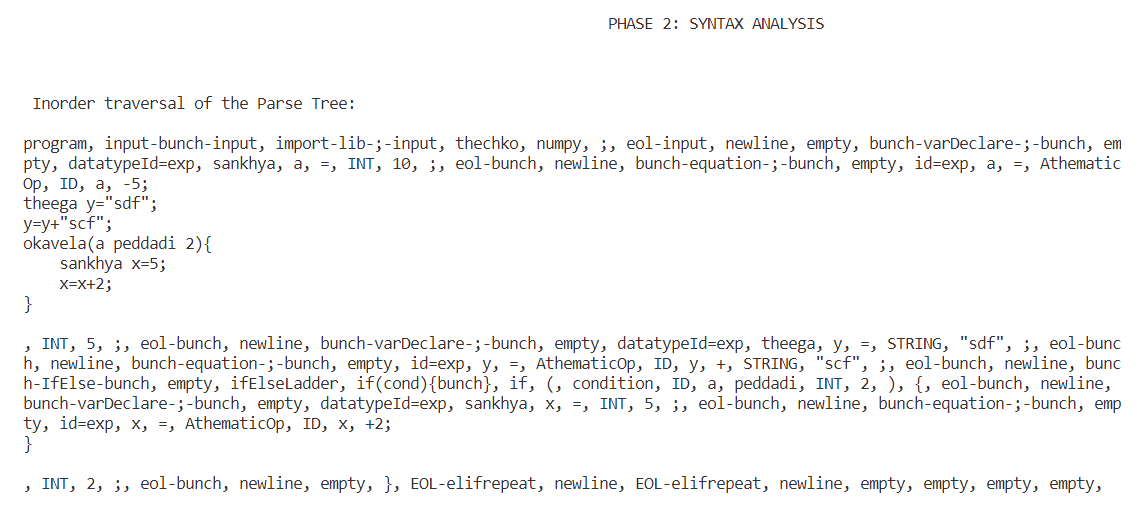
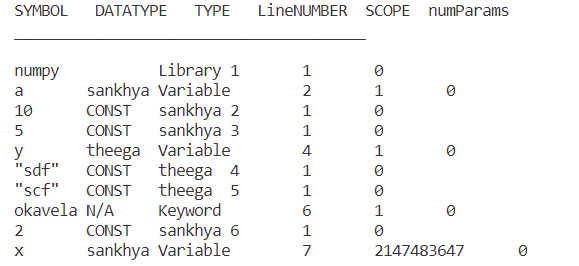
**SAMPLE INPUT:**



**SAMPLE OUTPUT:**



**SYMBOL TABLE:**





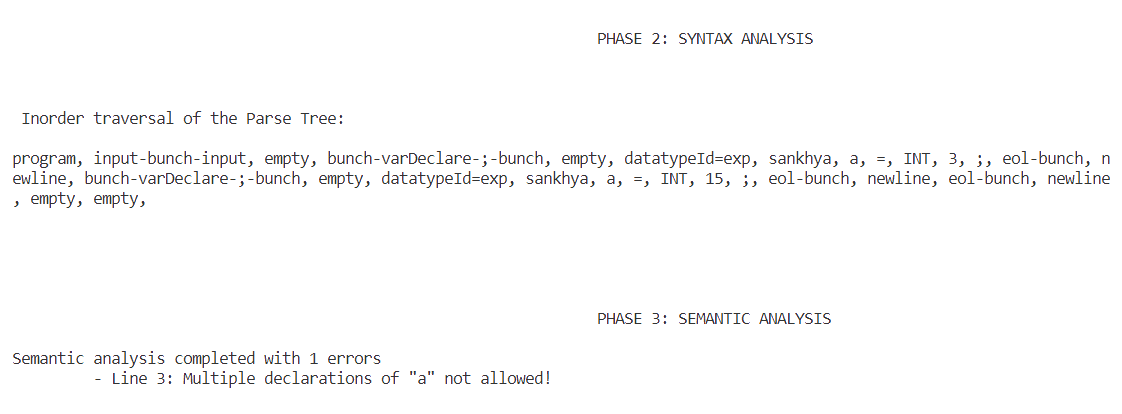
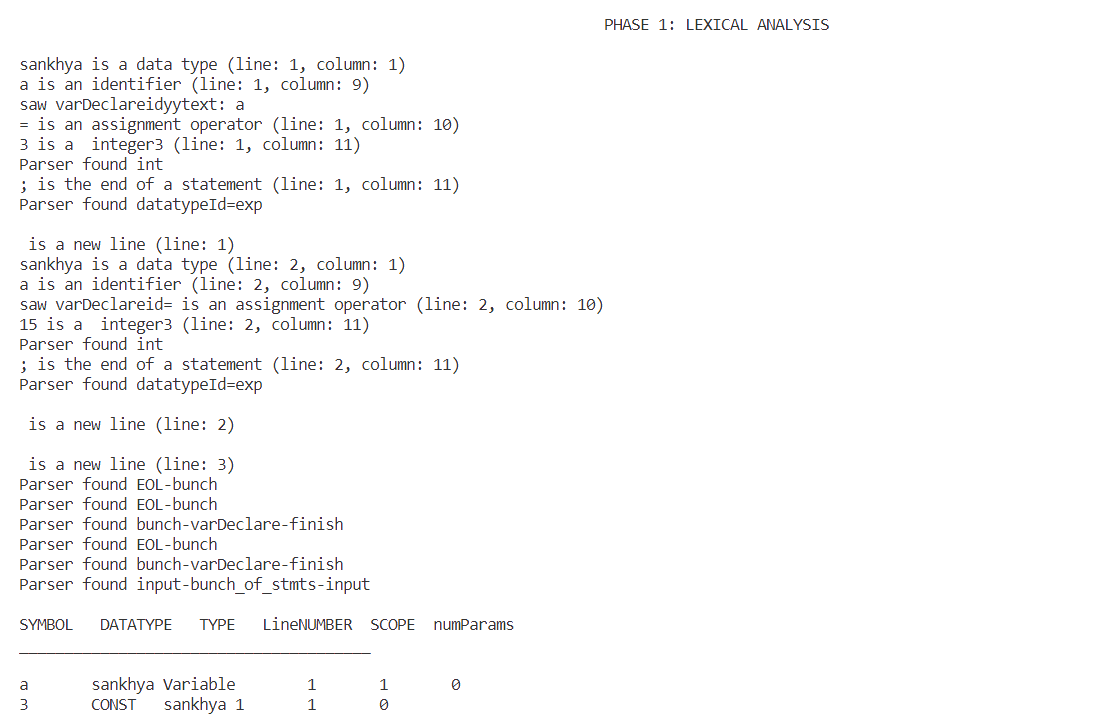
**SEMANTIC ERRORS:**

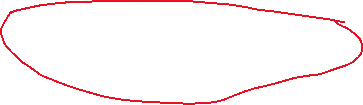
**1.Multiple Declarations of Variables/Functions:**

On encountering an identifier in **variable\_declaration** production, the semantic analyser looks up in the symbol table entries to check if any symbol is already declared with **symbol\_table[i].id\_name = $$.name**

Input:



Output: 



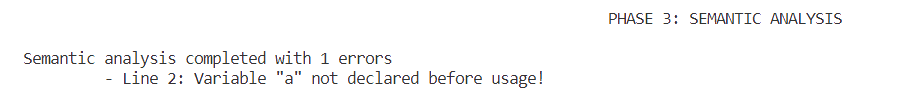
**2.UNDECLARED VARIABLES:**

On encountering an identifier in **exp** or **equation** or **bunch\_of\_statements** productions, the semantic analyser looks up in the symbol table entries to check if this symbol is already declared with **symbol\_table[i].id\_name = $$.name**

Input:



Output:

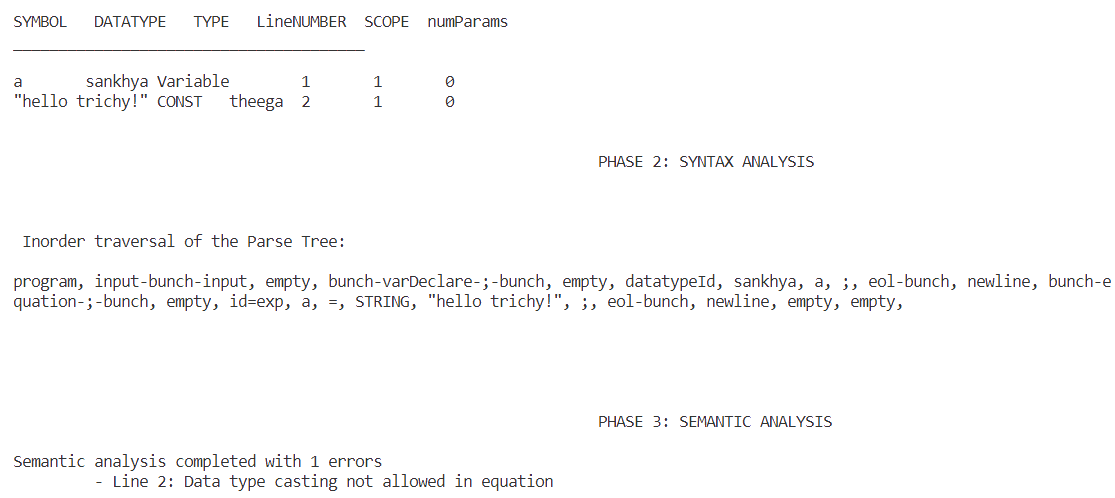


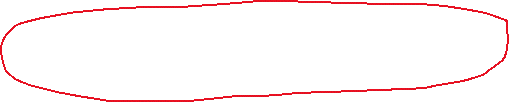
**3. TYPECASTING:**

When parsing **equation**/**variable\_declaration** , the semantic analyser evaluates the data type of the resulting expression in RHS of the equation. If RHS’s datatypes does not match the datatype of the LHS (looks up in the symbol table for this), then semantic error is reported.

Input:



Output:



**4.INVALID OPERATIONS ON DATATYPES:**

The semantic analyser is designed to report invalid operations on certain datatypes associated with typecasting.

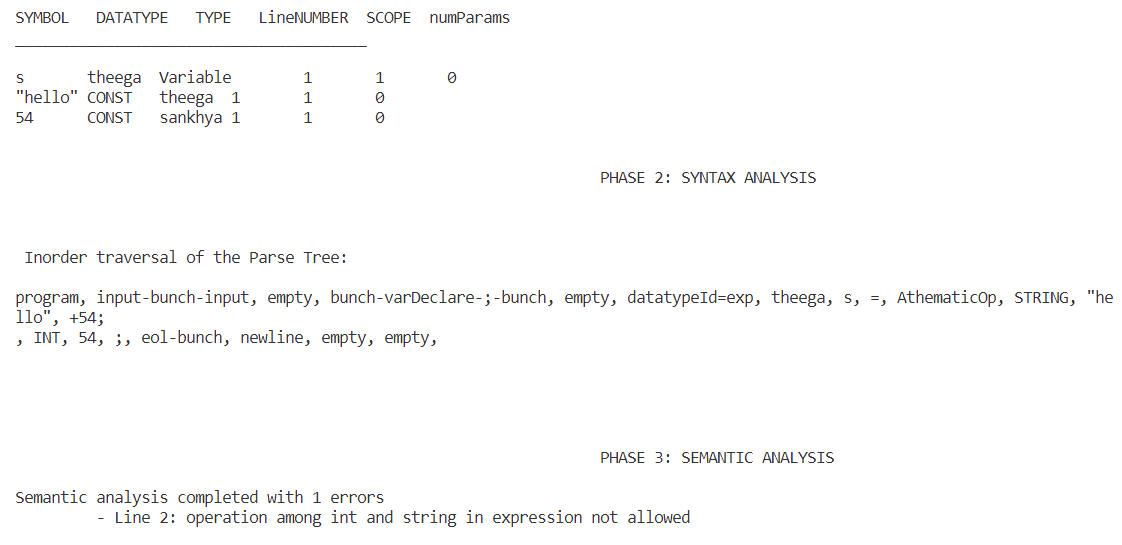
Examples: string + string //should be allowed because its concatenation

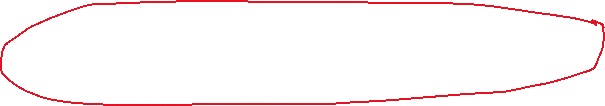
string – string // not allowed as ‘-‘ is not defined for strings

Input:



Output:





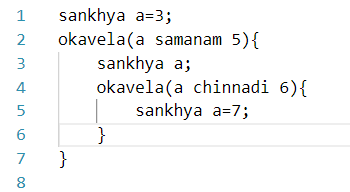
**5. SCOPE OF VARIABLES:**

The symbol table stores the scope of each variable as an integer. Starting from 1, the scope increases by 1 on seeing a ‘{‘ and decreases by 1 on seeing a ‘}’.

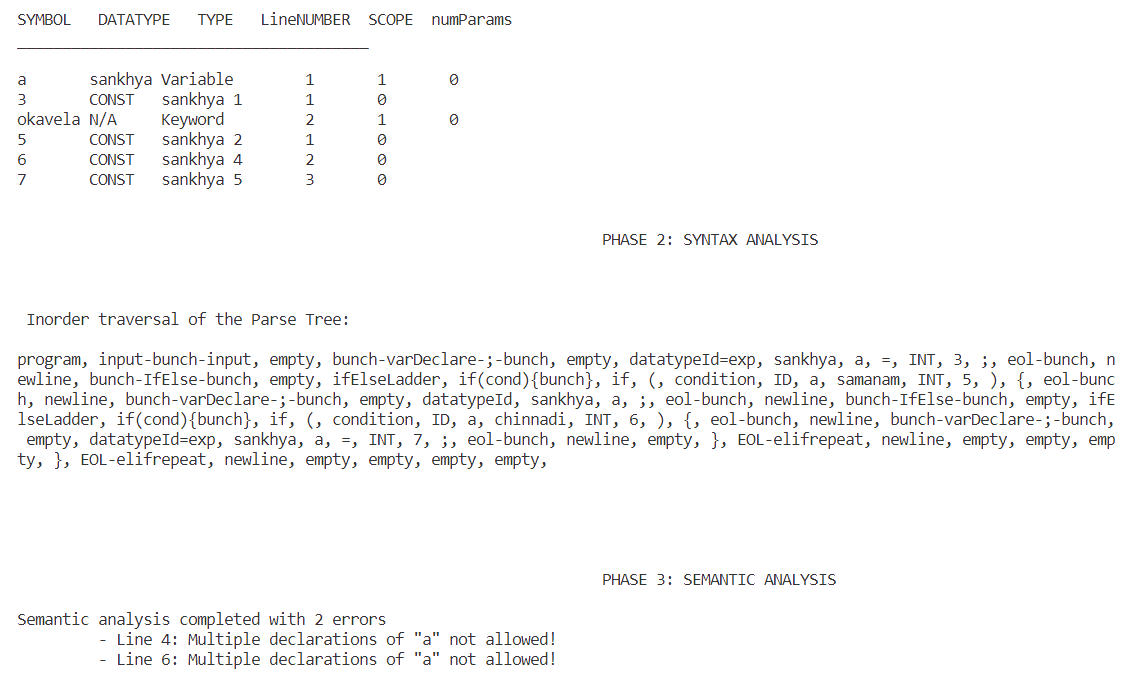
Lower scope value -> higher the globality of that variable

All the variables within {} will be marked as out of scope/destroyed by changing their scope to +infinity. Now any variable which is already defined can be defined again in another scope after verifying its scope=infinity.

Input 1:

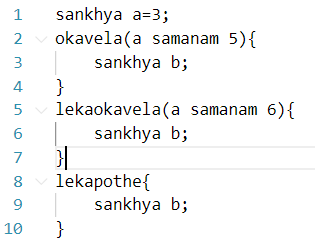


Output 1:





Input 2: // in this input scope of the variable is only within the ‘{}’ . so we are allowed to declare the same sankhya multiple times without triggering semantic errors.



Output2:

