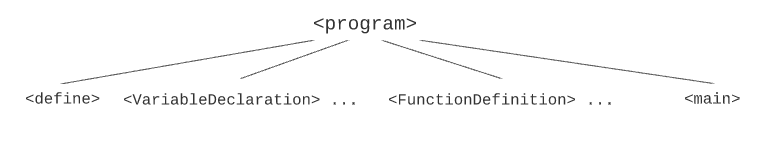
**Parser Documentation**

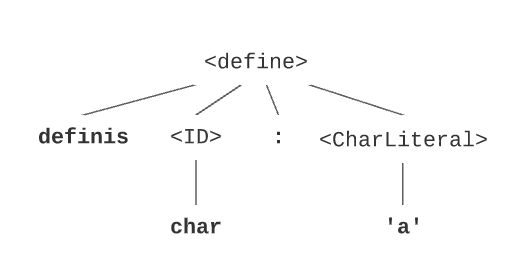
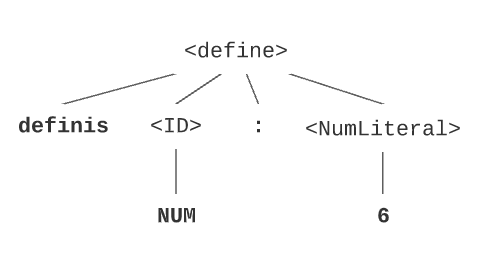
1. **Acceptor:**

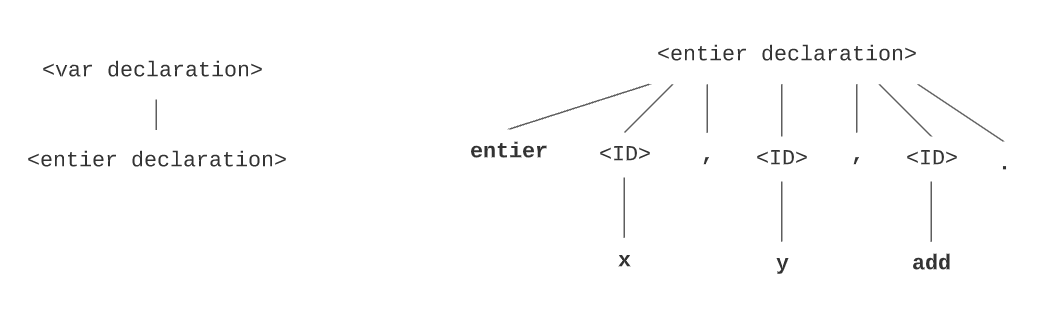
The acceptor takes as input the token stream that was generated by the lexer. It reads it token by token and checks if the sequence of tokens adheres to the rules of the grammar. While reading the tokens, the subprograms that correspond to each rule also generate their corresponding parse tree.

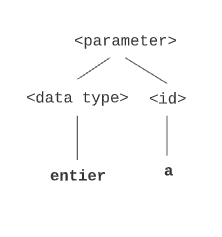
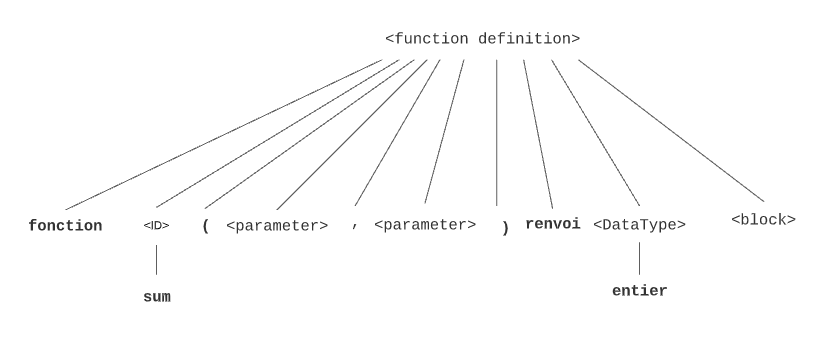
Here are some examples of how parse trees look like for each grammar rule:

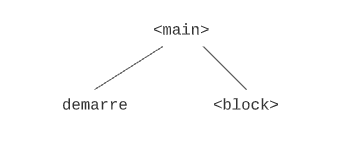
*N.B., these are just examples, because for each example there might be many variations.*

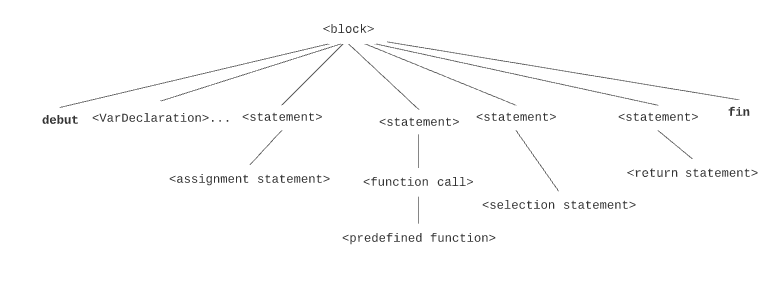


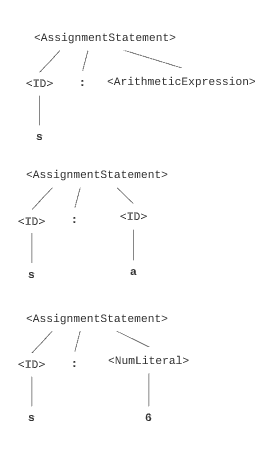
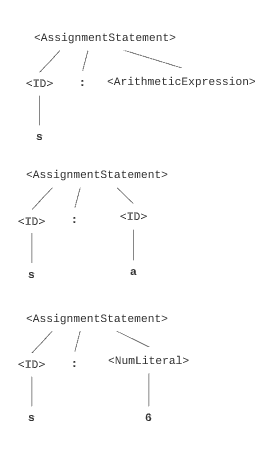


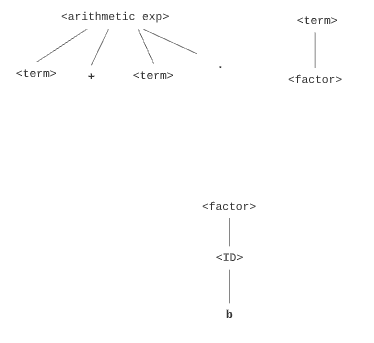
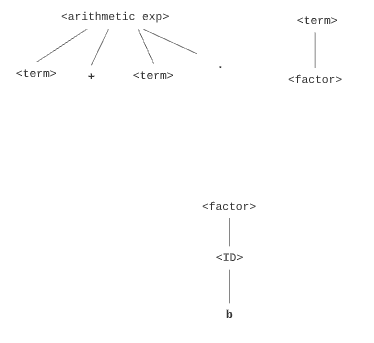






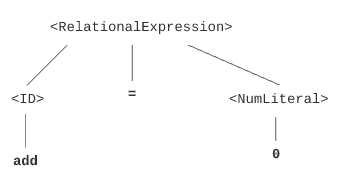


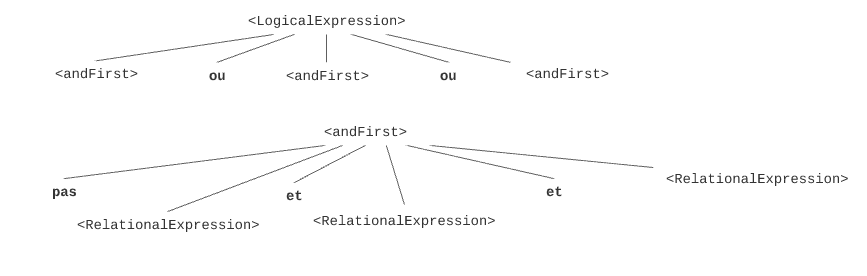


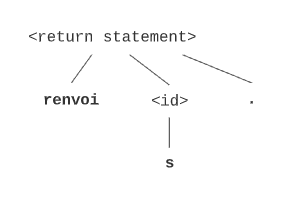
 











To build the parse tree, since the number of children of each node is unknown and cannot be identifier since a block for example can have “infinitely” many statements, we used the first-child-next-sibling method. To display the tree, we used levels to easily know which children belong to which parent.

If a subprogram finds an unexpected token, it generates an error that specifies the token that caused it, and it stops the execution of the program. Otherwise, if no error was caught, the parse tree gets fully generated and printed to an output file that will be the input to the next phase.

*N.B., This phase works for all the four input files (from the simple to the complex) and covers all the grammar rules.*

1. **Static Semantics:**

Static semantics program takes as input the parse tree generated by the acceptor and the symbol table generated by the lexer.

*N.B., Please refer to the modifications done on part 3 to know what the symbol table contains.*

The static semantics uses the symbol table to check if the variables and the functions that appear in the parse tree are declared and defined before being used. It first reads the symbol table to an array to simplify the search, and then it goes through the parse tree and does the following:

* If an identifier that refers to a variable (not a function) is read from the parse tree, the static semantics goes through the symbol table array and reaches out to the scope in which that variable was found, then it scans all the variables in that scope until it encounters the desired variable. If it doesn’t encounter it, it scans the global scope. If the variable was not found in neither the local scope nor the global scope, and error is reported.
* If an identifier that refers to a function is encountered in the parse tree, static semantics does through the symbol table array and searches for the scope of that function. If no scope for the function was found, an error is reported.

When an error is found, the program does not stop, because we want to report all errors and not only the first one caught.

The output of the static semantics is either a sentence denoting the success of the correctness of the semantics, or the list of errors found.

*N.B., This phase works for all the four input files (from the simple to the complex).*

1. **Generator:**

The generator takes as input the parse tree generated by the acceptor and generates from it the corresponding assembly code. It reads through the parse tree, takes into consideration the necessary non-terminals, and skips the unnecessary ones when they are known by default (by us). The assembly code generated, can then be the input to the assembler that we developed in the first deliverable.

*N.B., This phase works only for the simple program.*