2 semantic checks have been implemented.

In general, all semantics checks are done during the iterations through the parse\_tree.

1. Variables, functions, and parameters have to be defined before being used, no double definitions allowed.

All information about objects used in program is stored in the semdata.stack. When some variable, constant or function definition occur it is checked whether there is an object with such name (if precisely – its name) in semdata.stack or not. If there is – error is raised because redefinition is not allowed in this syntax. Also function parameters are checked not to be a global variable – otherwise the error is raised.

In the same way, when some variable, function (function call occur) or constant is used it is checked whether its name exists in stack or not. If it is not – error is raised.

1. Parameters are only allowed to appear in the function in which they have been defined.  
   There is an additional data structure called local\_vars, which is used as stack. When the function is defined – information about objects is stored in local\_vars (and not stored semdata.stack). The only thing which is stored in semdata.check – name of the defined function. After the return from the function is reached local\_vars stack is cleared.

Before each check of the variable/constant/function definition/using it is checked whether we are inside the function or not and, hence in which stack should their existence be checked.

The semantic check prints all errors found.  
This part is definitely the most creative part as you are allowed to implement your own (in some way) grammatical rules. I think the most difficult part was to understand the idea how to do the semantics checks in general and after that it was quite easy to continue. Also in this stage some problems with the semantic tree were fixed (there wasn’t any real errors, but at some points it was not that useful).