The link to github: https://github.com/eduard-C0/FLCD/tree/Lab8

The parser algorithm is the one for recursive descendent.

The algorithm is running as long as the current state isn't the final state and the current state isn't error state. If the cu rrent state is the normal state the algorithm is checking if the index of the parser is equal with the size of the given se quence and if the input stack is empty. If this condition is respected the success function from the parser will be calle d else if the parser index isn't equal with the size of the given string we will check if the input stack is empty or not. I f it is the insuccess function from the parser will be called.Else if the input stack isn't empty we will take the first ele ment from the input stack and we will check if that element is contained in the list of non terminals, if it is contained the expand function form the parser will be called. If not we will verify if the size of the given string is smaller or eq ual than the current index from the parser, if it is the inssuccess function from the parser will be called if not we will get the element situated at a position equal with the current index from the given string . If this element is equal with the element from the input stack we will call the advance function from the parser. If not we will call the inssuccess f unction. If the current state isn't a normal state but it is a back state we will take the last element from the working st ack and we will verify if our list of non terminals contains that element. If the answer is true than we will call the bac k function from the parser if not we will call the try again function.
At the end we verify if the current state is an error state and if it is than we will write into the output file an error mes sage else we will write the derivation message.

The parser class contains a string for each state, the current index, a todoStack, an inputStack, the grammar, a string i n which we keep the current state of the parser and a list of strings for the derivation.
It also contains a success function which sets the current state of the parser to the final state, a insuccess function whi ch sets the current state of the parser to the back state. The back function is decreasing the current index and it adds i nt the input stack on the first position the removed element from end of todoStack and sets the current state ot the ba ck state.The advance function is increasing the current index and it adds to the todoStack at the end the first element from the begining of the inputStack removing it from the inputStack. The expand function takes and removes the firs t element of the inputStack and sets the current index of the element to 0,it adds to the end of the todoStack that elem ent. After that we take the first production list for our element and for each elemet of that list we are creating a new g rammar element which will be added to the input stack in the reverse order. The tryAgain function takes and remove s the last element from the todoStack. It also takes the list of productions from grammar for this element. If the index of the element is smaller than the size - 1 of the productions list it will increase the index of the element, will add it at the end of the todoStack and will get the list of productions for this element and will remove all the productions co rresponding to the element which were already in the input stack. At the end we will add in the input stack the new p roductions. If the index is equal to 0 and the current element is the starting element we will state the current state to t he error state and we will exit the function else we will set the current state to be the back state and we will remove a ll the productions corresponding to the current element from the input stack.

The parsing tree is represented as a derivation string.The derivation string is build in the expand and the try again fu nction when we add another production.