Compiler Phase 2 : Parser

Requirements:

- GUI for your program and it mustn't be reopened with each new test case

- If you were assigned **Case 1 or 2** then implement **LL(1)** parser you will be given the grammar below you should adjust is to be suitable for LL(1) parser then get the **parsing table** of your parser and store it in your program GUI and show it if asked (note: keep the steps of deriving your Parser table, you might be asked to show it) , the **input** will be any string that follows the grammar you were given, your program output should label the input if it is a correct syntax or not using the parsing stack , if it is correct your program should output **the Parse tree** and **syntax tree** , **Hint:** use the parsing stack table and convert it into nodes of a tree, if not correct syntax then just label it as not ,no need to draw any tree

- If case 3 or 4 implement SLR(1) parser (it hasn't been covered yet in the lecture) you can go through Fathi's playlist starting from 53 to 62 to understand SLR(1) and not waste time until it is covered in lecture : https://www.youtube.com/playlist?list=PLQkyODvJ8ywuGxYwN0BfMSvembIJkNQH1 or you can read the textbook **Compilers: Principles, Techniques, and Tools** , will be given the grammar

below you should adjust is to be suitable for SLR(1) parser then get the **parsing table** of your parser and store it in your program GUI and show it if asked (note: keep the steps of deriving your Parser table, you might be asked to show it) , the **input** will be any string that follows the grammar you were given, your program output should label the input if it is a correct syntax or not using the parsing stack , if it is correct your program should output **the Parse tree** only **, Hint:** use the parsing stack table and convert it into nodes of a tree, if not correct syntax then just label it as not ,no need to draw any tree

- The **terminals** in the grammar are your **tokens** so adjust your token extraction to be exactly like terminals of the grammar (**all in lowercase)** and there is **atleast a space** between each token so adjust for that also

- The **Identifier** token must start with a letter and after it any number of letters or digits may occur , **Number** token is an unsigned int

- Please view your grammar carefully as some cases have had a slight modification in order to balance difficulty of the project for all teams

- You may use any library or module to help you with drawing the tree but not a library that does the parsing (that is for you to implement

- As for questions about corner cases if the test case you have in mind follows the grammar then assume it might be given to you

Case 1 :

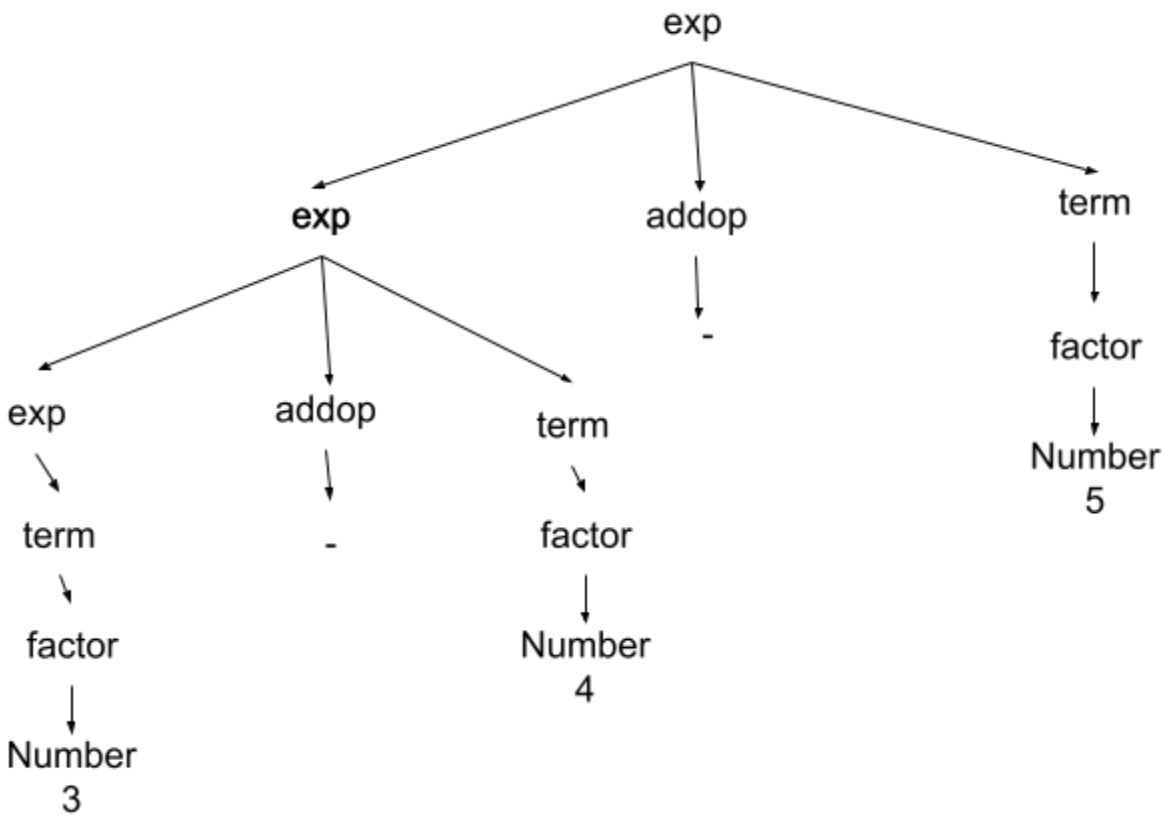
exp → exp addop term | term
addop → **+** | **-**
term → term mullop factor | factor
mullop → ***** | */*
factor → **(** exp **)** | **Identifier** | **Number**

All in bold are terminals

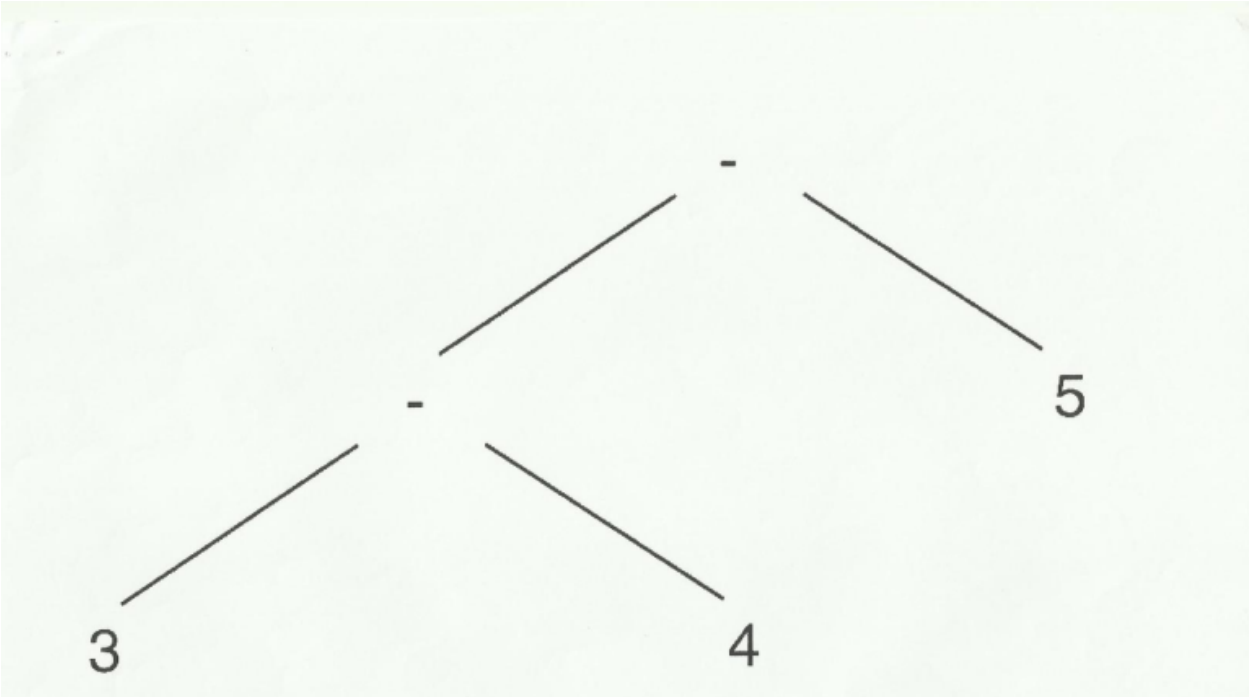
Example:

3-4-5

Parse tree , note: your parse tree will change because modification to grammar



Syntax tree



Case 2:

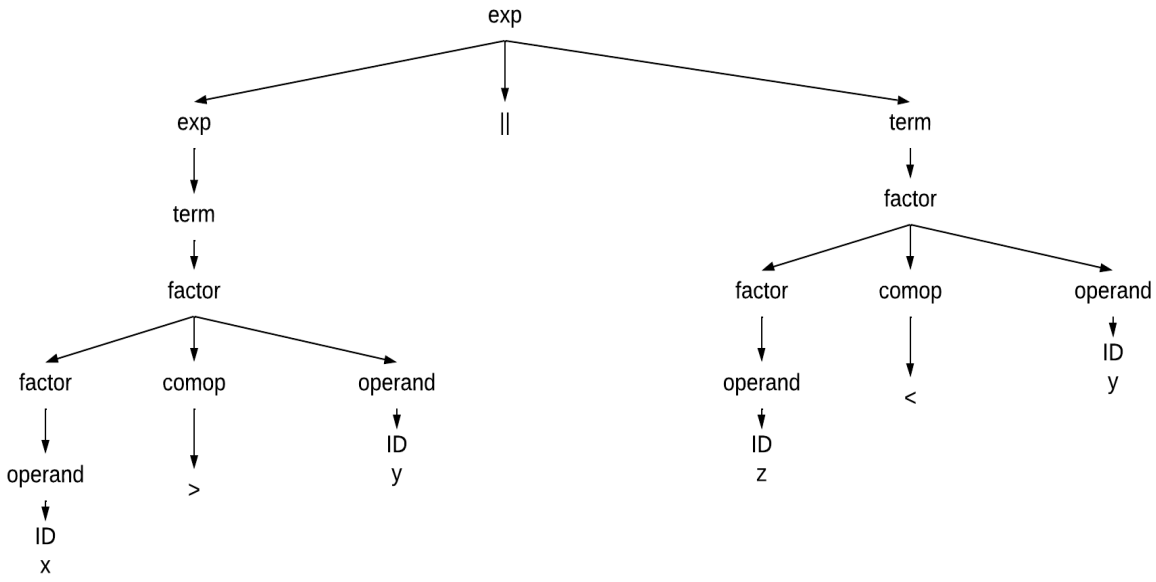
exp → exp **||** term | term
term → term **&&** factor | factor
factor → factor comop operand | operand
comop → **>** | **=** | **<**
operand → **!** operand | **Identifier**

Notice the statement changed from before we only have 3 comparison operators and we removed brackets and removed Number and not can be repeated any number of times

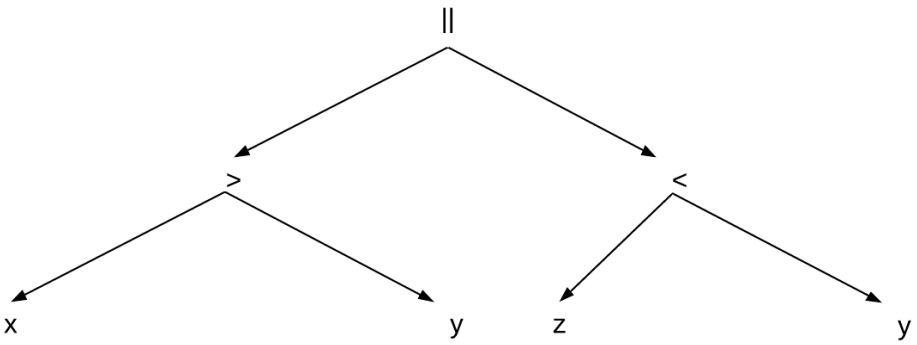
Example:

x > y || z < y

Parse tree , note: your parse tree will change because modification to grammar



Syntax tree:

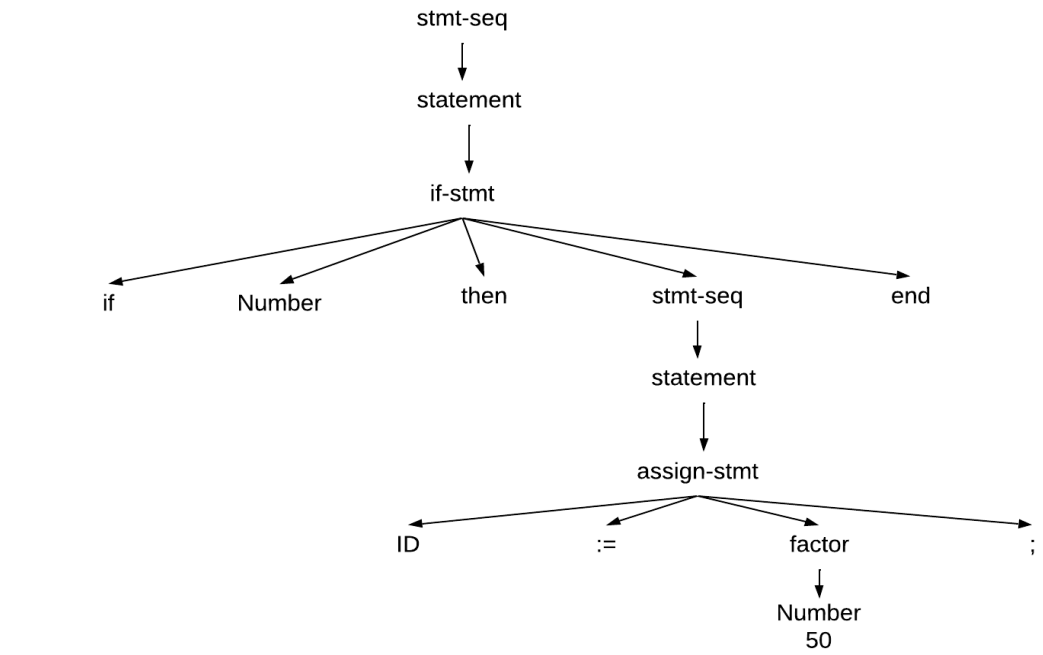


Case 3 :

stmt-seq → stmt-seq statement | statement
statement → if-stmt | assign-stmt
if-stmt → **if Number then** stmt-seq **end**
assign-stmt → **Identifier :=** factor ;
factor → **Identifier** | **Number**

Notice the statement changed from before we don't have an else part this time but we can have multiple ifs or nested ifs and we can have many standalone assign statement (not inside and if) and each if must have at least one assign statement or an if statement inside of it

Example:
if 5 then x := 50 ; end
Parse tree , note: your parse tree will change because modification to grammar



Case 4

stmt-seq → stmt-seq statement | statement
statement → repeat-stmt | assign-stmt
repeat-stmt → **repeat** stmt-seq **until Identifier**
assign-stmt → **Identifier :=** factor ;
factor → **Identifier** | **Number**

Notice the statement changed from before we don't have a condition part this time we have only an Identifier as the condition , we can have multiple repeats or nested repeats and we can have many standalone assign statement (not inside a repeat) and each repeat must have at least one assign statement or a repeat statement inside of it
Example:

repeat x := 50 ; until x

Parse tree , note: your parse tree will change because modification to grammar