1 Week 6

1. Transform an arithmetic expression from postfix to infix, without execution. There is one expression per line.

   ```
   2 3 + should return 2+3 or (2+3)
   4 2 3 *+  should return 4+2*3
   22 3 + 5 6 + * should return (22+3) * (5+6)
   ```

   Hints:

   a) use $\%union\{char * s;\}$ in order to pass from lex a string for numbers

   b) use strdup, strcat, strcpy from string.h in order to create a new string, concatenate two string.

2. Create a **parser** for If-then-else statement. The program should just recognize whether a sentence is correct or not (without execution and for the moment without returning the parse trees).

   Allowed structures:

   ```
   if (a>2) then a=2
   if (3>a) then if (b>=2) then b=2
   if (3>a) then if (b>=2) then b=2 else c=10
   2
   c=10
   ```

   Forbidden:

   ```
   if (a=3) then a=2;
   if (a>3) then a<2;
   ```

   - In if condition you should have a relational or equality operator($<, >, <=, >=, ==, !=$).
   - As body of *then* branch you can have only an assignment or another if statement.
   - Any if statement ends with ';'.

   Hints: possible rule for Yacc

   ```
   if_stmt : IF '(' cond ') THEN stmt ';'
           ;
   %PREC changes the precedence level associated with a particular grammar rule
   %NONASSOC allows to define precedence for operators which are not associative
   ```

3. Create a parser for the previous problem which creates and prints the parse tree. Add functions from $if\_functions.c$ as actions in the grammar described in the previous problem.

   First, compile and analyse the functions defined in $if\_functions.c$.

   ```
   gcc  if_functions.c
   ```