```
for(i = deadline; deadline<October9; deadline++){
public void createParticipant(Participant participant){
this.Amer_Biro = paritipant;
this.Sulaiman_Kasas = paritipant;
this.Bashar_Bedwai = paritipant;
}
Assignment1 assignment1 = new Assignment1(createParticipant){
Array tasks[6] = new Array[tasks];
tasks[0] = tasks.add(calculator);
tasks[1] = tasks.add(while_loop);
tasks[2] = tasks.add(if_statment);
tasks[3] = tasks.add(if_else_statment);
tasks[4] = tasks.add(for_loop);
tasks[5] = tasks.add(array);
return grades;
}
return assignment1();
```

s176356

s195462

s183356

Assignment 1, Compiler Teknik F2020

WHAT TO RECEIVE

- Source files
  Part1, Calculator
  A link to Github
  https://github.com/AmerBiro/Assignment1_Compiler_Teknik_F2020_Part1.git
  A parser tree with a test case
  input.txt with the used test case

  Part2, while, if statement, if else statement, for loop, array
  A link to Github
  https://github.com/AmerBiro/Assignment1_Compiler_Teknik_F2020_Part2.git
  A parser tree with a test case
  impl_input.txt with the used test case

- A zip file containing the following
  All the source files for part1 and part2
  Rapport in pdf format
  Parser tree for part1 and part2

  IT IS IMPORTANT THAT THE VRSION OF ANTLR IN THE MAKE FILE MUST BE CHANGED DEPENDING ON WHICH ANTLR VERSION YOU HAVE ON YOUR MACHINE

Documentation of Compiling implementation

- Part1

Calculator

RegEx: we have worked on the calculator by splitting the project into mini projects. We have started by writing the RegEx. For example, what it is allowed/not allowed to type, numbers or letters, float or int. Then we could use input as digits from 0-9. Float is optional.
Afterwards, we have defined the WHITESPACES such as \t\r\n , single comment and multiple comments.

```
NUMBER                : [0-9]+ ('.' [0-9]+)? ;
WHITESPACES           : [ \t\n\r]+ -> skip ;
COMMENT               : ('//' (~[\n])*) -> skip ;
MULTILINECOMMENTS     :  ( '/*'  (( '*'~[/] | ~[*]  )*) '*/') -> skip;
```

Grammar: we found out which rules we have to include to get a well working calculator. We have first added the basic operations as addition, subtraction, division and multiplication.

```
| e1=expr op=('*' | '/') e2=expr                       # MULTI_DEVI
| e1=expr op=('+' | '-') e2=expr                       # ADD_SUB
```

Then we added two extra operations as the power of and sqrt().

```
| e1=expr '^'  e2=expr                                 # POWER
| 'sqrt('  e1=expr   ')'                               # SQRT
```

We have also added a possibility to type a positive or negative number.

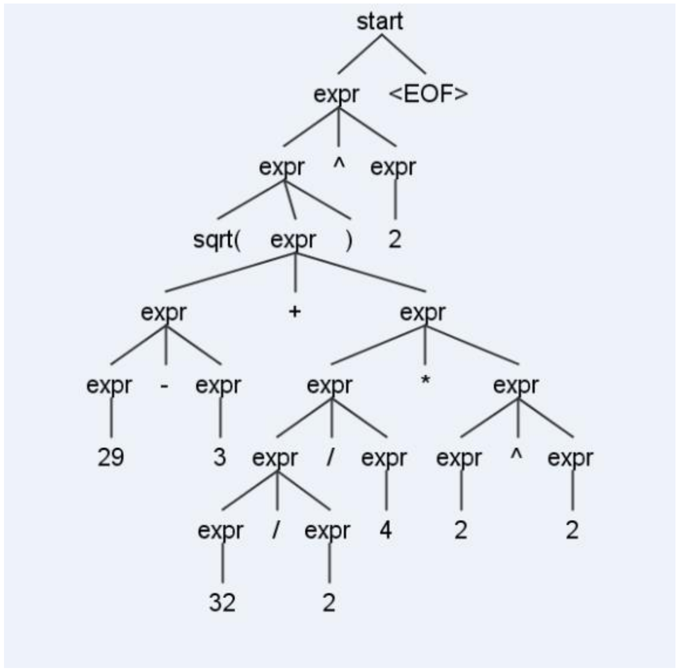
```
| op=('+' | '-') e=expr                                # Minus
```

We have added functions to the operations in the g4 file then we can implement them in main.java by importing them from simpleCalcVisitor classe. It is absolut necessary to do this to get all the operations to work.
When we added different function in g4 file we the compiler automatically defined functions in the interface classe called simpleCalcVisitor. Then we could implement these functions in main.java classe. Functions have different code depending on that task of each function.

```
T visitMULTI_DEVI(simpleCalcParser.MULTI_DEVIContext ctx);
T visitADD_SUB(simpleCalcParser.ADD_SUBContext ctx);
T visitMinus(simpleCalcParser.MinusContext ctx);
```

`visitMULTI_DEVI`	Stands for multiplication and division
`visitADD_SUB`	Stands for addition and multiplication
`visitMinus`	Stands for negative

parser tree



The parser tree above shows how the different operations work together depending on binding strongness

- Part2

Extending the while loop

We have used the same strategy to do this task. Excepting that this time we have got a bit support which is the a starting point. We worked against extending the while loop. The while loop worked well but it had just “!=” not equal condition. It needed the following conditions.==, >=, >, <, <= . It had also just multiplication and addition. We should also implement other operations such as division and subtraction.
We could reuse what we have done in the calculator such as the missing operation. We just needed to import them into the new project.
Afterwards, we started implementing the missing conditions. We did it in the same way as we did in the first part of the project. Writing the rules in the g4, having functions’ names to the rules, defining the functions in the interface classe implVisitor.java and finally implementing the functions in the main.java classe.

g4

```
condition
    : expr '>' expr                                                    # Greater
    | expr '>=' expr                                                   # GreaterThan
    | expr '<'  expr
```

implVisitor.java

```
T visitGreaterThan(implParser.GreaterThanContext ctx);
T visitGreater(implParser.GreaterContext ctx);
T visitLess(implParser.LessContext ctx);
```

main.java

```
   public Double visitLess(implParser.LessContext ctx){
   Double left = visit(ctx.expr(0));
   Double right = visit(ctx.expr(1));
   if (left < right)  return 1.0;
   else return 0.0;
}

   public Double visitGreaterThan(implParser.GreaterThanContext ctx){
   Double left = visit(ctx.expr(0));
   Double right = visit(ctx.expr(1));
   if (left >= right)  return 1.0;
   else return 0.0;
}

   public Double visitGreater(implParser.GreaterContext ctx){
   Double left = visit(ctx.expr(0));
   Double right = visit(ctx.expr(1));
   if (left > right)  return 1.0;
   else return 0.0;
}
```

if statement and if else statement

We started working on the simple if statement and when we were done then we worked on the if else statement.

g4

```
| 'if'     '('c=condition')' p=program                                      # IfStatment
| 'if'     '('c=condition')' p1=program  ('else' p2=program)?               # IfElseStatment
```

main.java

```
@Override
public Double visitIfStatment(implParser.IfStatmentContext ctx) {
   if(visit(ctx.c).equals(1.0)){
      visit(ctx.p);
   }return null;
}

@Override
public Double visitIfElseStatment(implParser.IfElseStatmentContext ctx) {
   if(visit(ctx.c).equals(1.0)){
      visit(ctx.p1);
   }else visit(ctx.p2);
   return null;
}
```

for loop

a simple for loop

g4

```
| 'for'  '('x=ID '=' min=expr '..' max=expr')' p=program                 # ForLoop
```

main.java

```
@Override
public Double visitForLoop(implParser.ForLoopContext ctx) {
       Double min = visit(ctx.min);
       Double max = visit(ctx.max);
   for (double i = min; i<max; i++){
      visit(ctx.p);
   }
   return null;
   }
```

array

g4

```
| x=ID '[' expr']' '=' expr ';'                                          #
ArrayAssignment
```

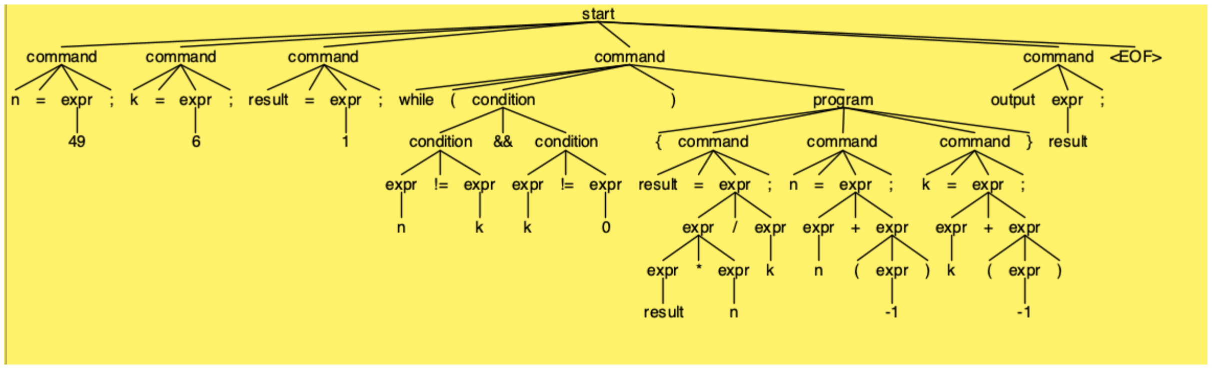
main.java

```java
@Override
public Double visitArrayAssignment(implParser.ArrayAssignmentContext ctx) {
       Double left = visit(ctx.expr(0));
       Double right = visit(ctx.expr(1));
       env.setVariable(ctx.x.getText()+"#"+left.toString(), right);
   return null;
}
```

TEST CASES

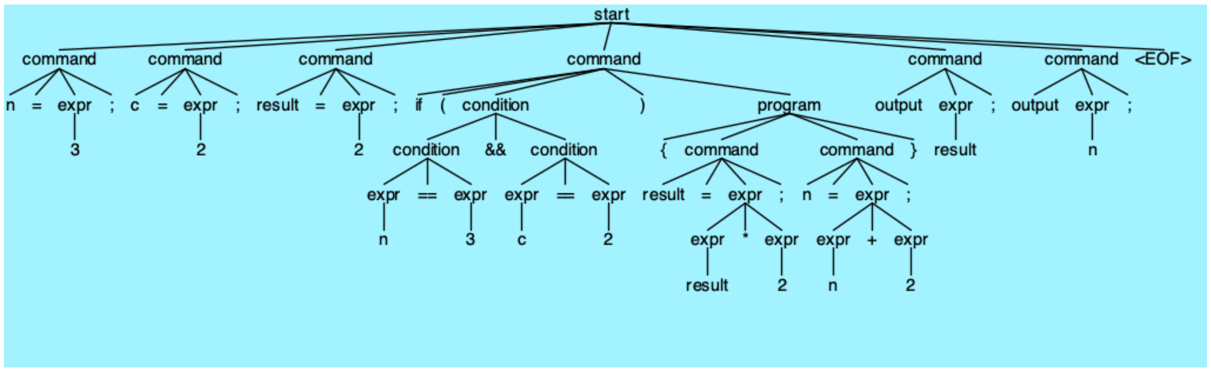
while loop with "&&" condition

```
n=49;
k=6;
result=1;

while(n!=k && k!=0){
  result=result*n/k;
  n=n+(-1);
  k=k+(-1);
}

output result;
```


start
command command command command command <EOF>
n = expr ; k = expr ; result = expr ; while (condition) program output expr ;
49 6 1 condition && condition { command command command } result
expr != expr expr != expr result = expr ; n = expr ; k = expr ;
n k k 0 expr / expr expr + expr expr + expr
expr * expr k n (expr) k (expr)
result n -1 -1

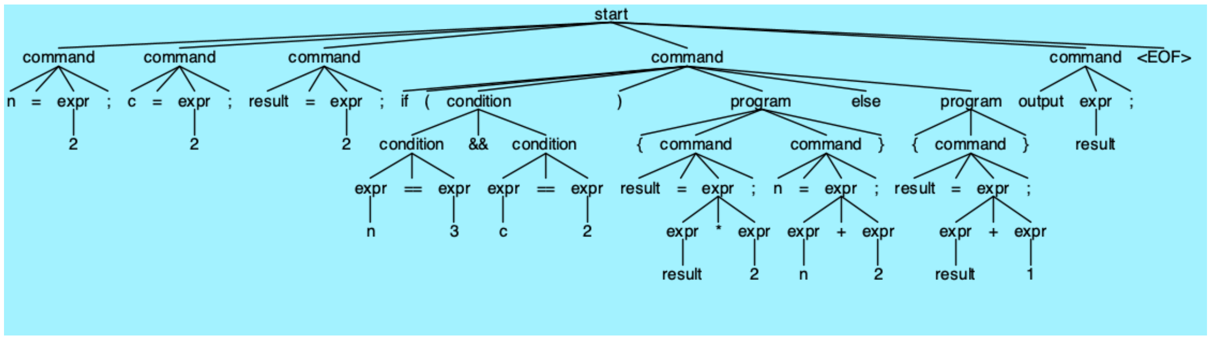
If Statement with “&&” condition

```
n=3;
c=2;
result=2;

if(n==3 && c==2){
result = result*2;
n = n+2;
}

output result;
output n;
```


start
command command command command command command <EOF>
n = expr ; c = expr ; result = expr ; if (condition) program output expr ; output expr ;
3 2 2 condition && condition { command command } result n
expr == expr expr == expr result = expr ; n = expr ;
n 3 c 2 expr * expr expr + expr
result 2 n 2

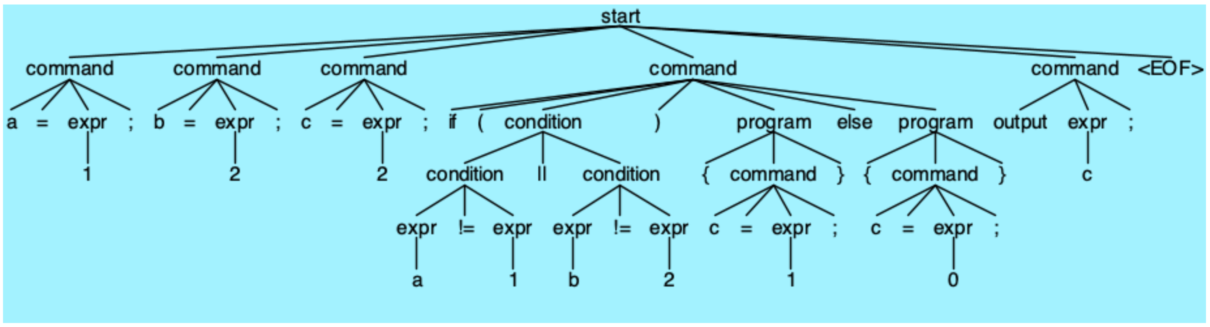
If else Statement with ”&&” condition

```
n=2;
c=2;
result=2;

if(n==3 && c==2){
result = result*2;
n = n+2;
}else{
result = result+1;
}

output result;
```


start
command command command command command <EOF>
n = expr ; c = expr ; result = expr ; if (condition) program else program output expr ;
2 2 2 condition && condition { command command } { command } result
expr == expr expr == expr result = expr ; n = expr ; result = expr ;
n 3 c 2 expr * expr expr + expr expr + expr
result 2 n 2 result 1

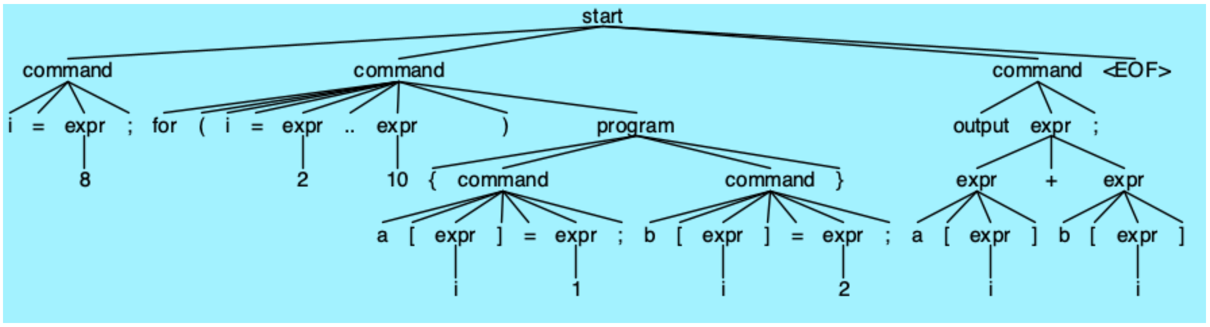
IF NOT with ”||” condition

```
a=1;
b=2;
c=2;

   if(a!=1 || b!=2) {
      c=1;
   }else{
      c=0;
    }
output c;
```



For loop with array

```
   i=8;
   for (i=2..10){
   a[i]=1;
   b[i]=2;

   }

output a[i] + b[i];
```


start
command
command
command
<EOF>
i
=
expr
;
for
(
i
=
expr
..
expr
)
program
output
expr
;
8
2
10
{
command
command
}
expr
+
expr
a
[
expr
]
=
expr
;
b
[
expr
]
=
expr
;
a
[
expr
]
b
[
expr
]
i
1
i
2
i
i