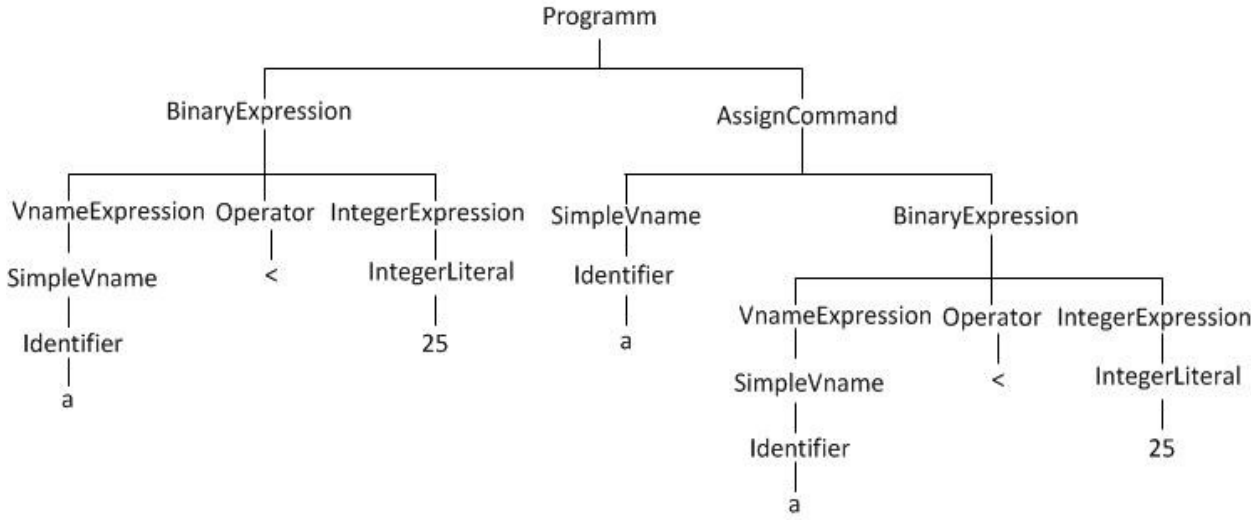
Compiler 1: Übung 1

Gruppe:

1. Michael Scholz (matr.: 1576630, rbg: mi48azih)
2. Ulf Gebhardt (matr.: 1574373, rbg: hu56nifa)

Aufgabe 1.1:



Aufgabe 1.2:

```
i := 0
while (i < count) do
        foo := count – i – 2
        j := 0
        while (j <= foo) do
                if(arr[j] > arr[j+1)
                        tmp := arr[j]
                        arr[j] := arr[j+1]
                        arr[j+1] := tmp
                ;
                j = j+1
        ;
        i := i+1
;
```

Aufgabe 1.3:

a) → Durch LL(1) Parser verarbeitbar.

b) → Durch LL(1) Parser nicht verarbeitbar.

Umformung:

G=({S,B},{x,y,z,0,1},P,S),P:

S ::= B (x | y) S | zS**

B ::= 0 | 1

c) → Durch LL(1) Parser nicht verarbeitbar. Die Grammatik kann nicht umgeformt werden. Sie produziert Wörter mit genauso vielen a wie b, bzw. genauso vielen a wie c. Deshalb kann mit einem festen k in LL(k) die Sprache nicht verarbeitet werden.

Aufgabe 1.4:

Die minimale Größe von k ist 2. Somit benötigt man für die gegebene Grammatik einen LL(2)-Parser.

Aufgabe 1.5:

Java Klassen: Command.java, WhileCommand.java, TypeDenoter.java, BoolTypeDenoter.java, Identifier.java

```
private void parseCommand(){
      if (currentToken.kind == Token.WHILE){
            parseWhileCommand();
      }else{
            parseID();
      }
}

private void parseID(){
      if(currentToken.KIND == Token.IDENTIFIER){
            acceptIt();
      }else{
            //report a syntatic error
      }
}

private void parseWhileCommand(){
      //wir erwarten, dass accept im Fehlerfall eine Exception wirft.
      accept(Token.WHILE);
      accept(Token.OpenBraket);
```

```
      parseBool();
      accept(Token.CloseBraket);
      parseCommand();
      accept(Token.END);
}
private void parseBool(){
      if(currentToken == „true“ ||
            currentToken == „false“){
                  acceptIt();
      }else{
            //report a syntatic error
      }
}
```

Aufgabe 1.6:

HexLet :: = A | B | C | D | E | F

Hex ::= 0x(Digit | HexLet) (Digit | HexLet)*

```
private byte scanToken (){
      switch (currentchar){
            case '0':
                  takeIt();
                  //Hex
                  if(currentchar == 'x'){
                              takeit();
                              while(isHexLet(currentChar) ||
                                    isDigit(currentChar))
                              takeit();
                              return Toke.HEX;
                  }
            case 'a':
            …
            case 'z' :
            …
      }
}
```

Das wesentliche Problem dieser Grammatik ist „0x“. Es müssen somit zwei Zeichen gelesen werden, um eine Hexadezimalzahl von einer Dezimalzahl zu unterscheiden.