Programming 2 (SS 2023)
Saarland University
Faculty MI
Compiler Design Lab

# Sample Solution 6
# C0

Prof. Dr. Sebastian Hack
Pascal Lauer, M. Sc.
Marcel Ullrich, B. Sc.

The calendar indicates which script chapters you should study in conjuction with each lecture. The exercises are designed to enhance your understanding of the lecture material and prepare you for the mini-tests and the final exam. Additional exercises can be found at the end of each chapter in the script.
The difficulty of an exercise on the sheet is determined by the number of annotated 'X' and 'O' marks in the tic-tac-toe field, with four levels (1-4) increasing by one mark per level.

## Exercise 6.1:

Find the syntax errors in the following C0 statements.

1. `if (x < y) r = x else r = y;`
2. `while a < b {r = r + 1; a = a + 1;}`
3. `if (a = 4) b = 42; else b = x;`
4. `if (x > y) abort; else r = y;`
5. `{x = 5; y = 4;; z = 0;}`
6. `if (z < 0) abort() else x = x + y;`

## Solution

1. Semicolon behind the first assignment missing.
   `if (x < y) r = x; else r = y;`
2. While condition needs to be parenthesized.
   `while (a < b) {r = r + 1; a = a + 1;}`
3. Equality comparison with =, instead of ==.
   `if (a == 4) b = 42; else b = x;`
4. Parentheses for `abort` missing.
   `if (x > y) abort(); else r = y;`
5. Too many semicolons.
   `{x = 5; y = 4; z = 0;}`
6. Missing semicolon behind `abort`.
   `if (z < 0) abort(); else x = x + y;`

## Exercise 6.2:

Draw the abstract syntax tree for each of the following C0 programs:

1. Program 1:

```
{
    r = 1;
    while (r <= n) {
        r = r * n;
        n = n - 1;
    }
}
```

2. Program 2:

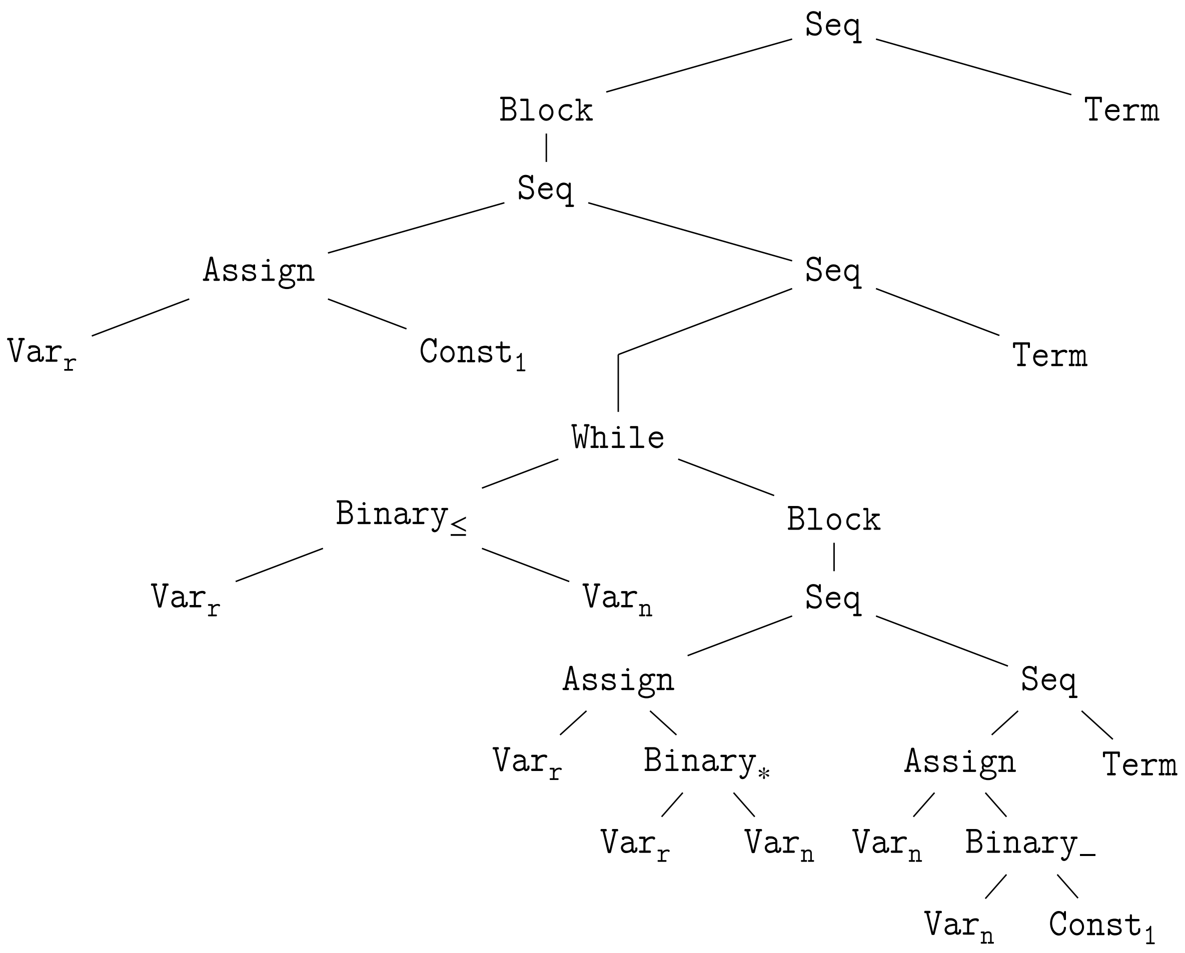
```
if (b < a) {
    a = 3;
    abort();
} else {
    b = 3;
}
```

3. Program 3:

```
{
    b = 3 * (2 + 1);
    c = a != b;
    while (c)
        c = 1;
}
```

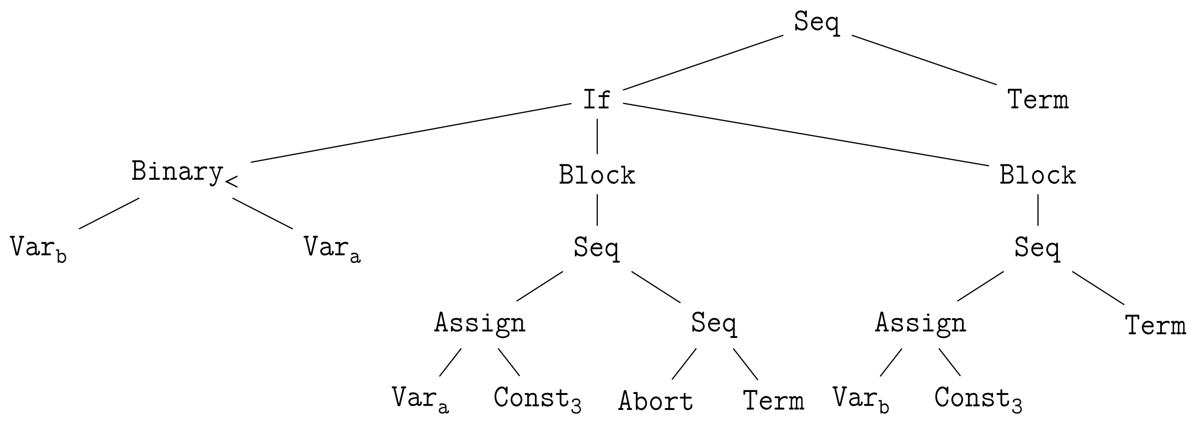
## Solution

Program 1:



Seq
Block
Term
Seq
Assign
Seq
$\mathtt{Var}_\mathtt{r}$
$\mathtt{Const}_1$
While
Term
$\mathtt{Binary}_\le$
Block
$\mathtt{Var}_\mathtt{r}$
$\mathtt{Var}_\mathtt{n}$
Seq
Assign
Seq
$\mathtt{Var}_\mathtt{r}$
$\mathtt{Binary}_*$
Assign
Term
$\mathtt{Var}_\mathtt{r}$
$\mathtt{Var}_\mathtt{n}$
$\mathtt{Var}_\mathtt{n}$
$\mathtt{Binary}_-$
$\mathtt{Var}_\mathtt{n}$
$\mathtt{Const}_1$

Program 2:



Program 3:

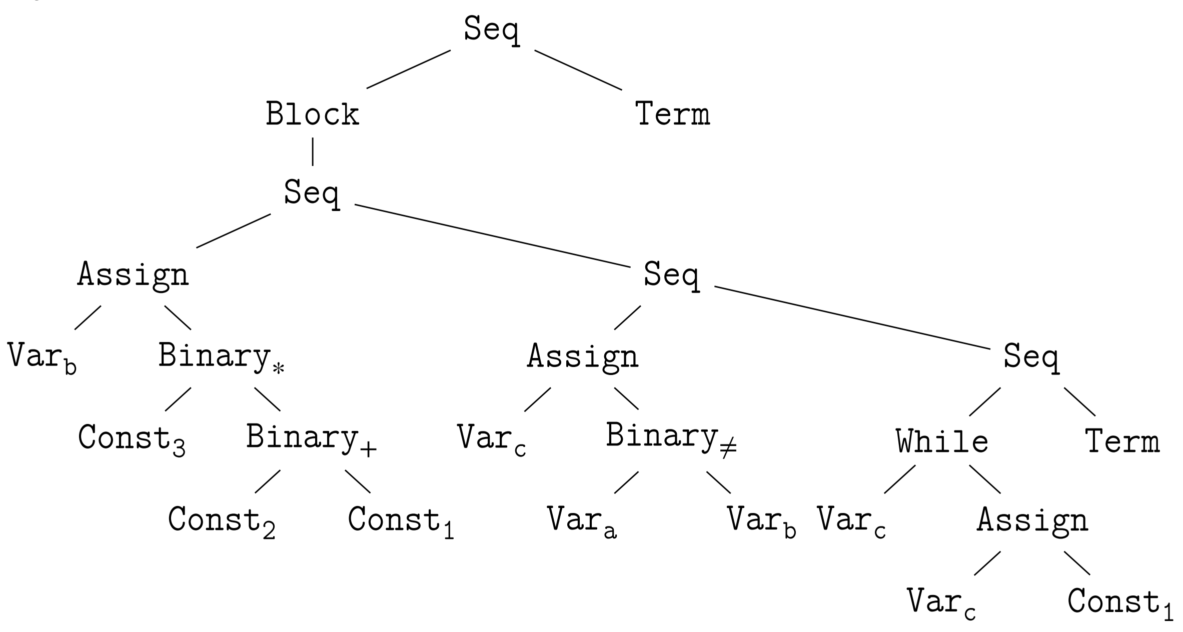


Figure 1

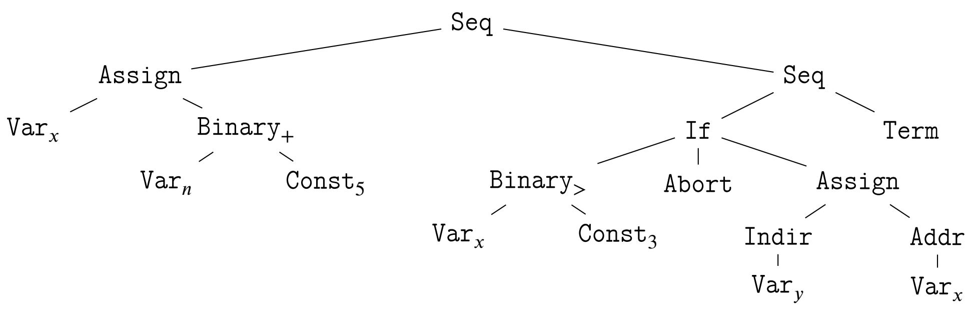
## Exercise 6.3: C0 AST

Draw the AST for the following C0 program.

```
x = n + 5;
if (x > 3)
    abort();
else
    *y = &x;
```

## Solution

At the highest level, the program has a sequence of an assignment and the if-statement. We need a new Seq node for every statement in the sequence. Hence, the root of the AST is Seq, the left child the assignment and the right one another Seq with the if-statement and empty program as the children. For unary and binary operators, the operator itself is the node and the arguments are the children. However, notice that the assignment and the $*$ and & are written a bit differently than e.g. the plus operator. The if-statement has three children (the condition and the then- and else-case).



## Exercise 6.4:

Define the abstract syntax of formulas in propositional logic. These consist of the binary operators $\vee, \wedge, \Rightarrow, \Leftrightarrow$ and the unary operator $\neg$.

## Solution

| | | Abstract Syntax | Concrete Syntax | Description |
|---|---|---|---|---|
| $LExpr \ni l$ | $::=$ | $\texttt{Var}_x$ | $x$ | Variable |
| $Expr \ni e$ | $::=$ | $l$ | | L-Value |
| | $\mid$ | $\texttt{Binary}_o[e_1, e_2]$ | $e_1\, o\, e_2$ | Binary Expr. |
| | $\mid$ | $\texttt{Neg}[e]$ | $\neg e$ | Unary Expression |
| $BOp \ni o$ | $::=$ | | $\wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow$ | Binary Operators |

## Exercise 6.5: For Loop

Extend the syntax and semantics of the C0-language by the following constructs:

1. The do-while loop (see the Control Flow chapter in the book).
2. A limited version of the for-loop. It should have the concrete syntax $\langle \texttt{for}\ (x\ =\ e_1;\ e_2;\ x{+}{+})\ s \rangle$.

## Solution

1. Extension of the syntax:

| | | Abstract Syntax | Concrete Syntax | Description |
|---|---|---|---|---|
| $Stms \ni s$ | $:=$ | $\texttt{DoWhile}[s, e]$ | $\texttt{do}\ s\ \texttt{while}(e)$ | Do-While Loop |

Extension of the semantics:

$$\langle \textbf{do}\ s\ \textbf{while}(e) | \sigma \rangle \to \langle s \llcorner\ \textbf{while}(e)\ s | \sigma \rangle \qquad \text{[DoWhile]}$$

2. Extension of the syntax:

| | | Abstract Syntax | Concrete Syntax | Description |
|---|---|---|---|---|
| $Stms \ni s$ | $:=$ | $\texttt{For}[x, e_1, e_2, s]$ | $\texttt{for}\ (x\ =\ e_1;\ e_2;\ x{+}{+})\ s$ | Do-While Loop |

Extension of the semantics:

$$\langle \textbf{for}\ (x\ =\ e_1;\ e_2;\ x{+}{+})\ s | \sigma \rangle \to \langle \{ x\ =\ e_1; \llcorner\ \textbf{while}(e_2)\{\ s \llcorner\ x = x + 1;\ \}\} | \sigma \rangle \qquad \text{[For]}$$

## Exercise 6.6:

Consider the following state in shorthand notation as defined in the lecture script:

$$\sigma = \{ \mathtt{x} \mapsto 1, \mathtt{y} \mapsto 2 \}$$

Write down $\rho$ and $\mu$. Use $\triangle$, ⬠ … do depict addresses.

## Solution

$$\begin{aligned} \sigma &= (\rho, \mu) \text{ where} \\ \rho &= \{\mathtt{x} \mapsto \triangle, \mathtt{y} \mapsto \text{⬠}\}, \\ \mu &= \{\triangle \mapsto 1, \text{⬠} \mapsto 2\} \end{aligned}$$

## Exercise 6.7: C0 execution protocol

Execute the following program according to C0 semantics on the state $\sigma = \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto ?\}$

```
r = 1;
while (r <= n) {
    r = r * n;
    n = n - 1;
}
```

## Solution

To improve readability, we use the abbreviated notation $\sigma$ and the shorthand S for loop body.

$$\begin{array}{lr}
\phantom{\to}\ \langle \mathtt{r} = 1; \mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto ?\}\rangle & \\
\to \langle \mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto 1\}\rangle & \text{[Assign]} \\
\to \langle \{\mathtt{r} = \mathtt{r} * \mathtt{n}; \mathtt{n} = \mathtt{n} - 1;\}\mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto 1\}\rangle & \text{[WhileTrue]} \\
\to \langle \mathtt{r} = \mathtt{r} * \mathtt{n}; \mathtt{n} = \mathtt{n} - 1; \mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto 1\}\rangle & \text{[Block]} \\
\to \langle \mathtt{n} = \mathtt{n} - 1; \mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 3, \mathtt{r} \mapsto 3\}\rangle & \text{[Assign]} \\
\to \langle \mathtt{while}\,(\mathtt{r} <= \mathtt{n})\, S \mid \{\mathtt{n} \mapsto 2, \mathtt{r} \mapsto 3\}\rangle & \text{[Assign]} \\
\to \langle \varepsilon \mid \{\mathtt{n} \mapsto 2, \mathtt{r} \mapsto 3\}\rangle & \text{[WhileFalse]}
\end{array}$$

## Exercise 6.8:

Which kind of termination can you observe? In case of proper termination also give the state after execution. You may use short-hand notation.

1. $\sigma = \{\}$

```
x = 1;
```

2. $\sigma = \{\mathtt{x} \mapsto 1, \mathtt{y} \mapsto 2\}$

```
x = 5;
{
   y = 6;
}
```

3. $\sigma = \{\mathtt{b} \mapsto 5, \mathtt{z} \mapsto 1, \mathtt{c} \mapsto 3\}$

```
    z = b + 3;
    if (b == 5) {
      abort();
    }
```

4. $\sigma = \{\mathtt{c} \mapsto 5, \mathtt{x} \mapsto 1, \mathtt{z} \mapsto 3\}$

```
    x = 8;
    z = 2;
    while (c < 7) {
      z = z + 1;
      c = c + 1;
      x = x + z;
    }
    if (x == 16) {
      abort();
    }
```

## Solution

1. Since the container of the variable `x` does not exist, the program gets stuck (`Assign` cannot be applied).
2. The program terminates properly. State: $\sigma = \{\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 6\}$
3. The program aborts, since b equals 5 in the given state. The `abort();` statement is thus executed.
4. The program terminates properly. State: $\sigma = \{\mathtt{c} \mapsto 7, \mathtt{x} \mapsto 15, \mathtt{z} \mapsto 4\}$