LTP Exercises

Theme 2: Foundations of Programming Languages

Part I: Questions

1. According to the following BNF rules:

```
<conditional> ::= IF <cond> THEN <exp> ELSE <exp>;
<cond>        ::= X>0 | X<0
<exp>         ::= X:=X+1 | X:=X-1
```

   which of the following sentences is *legal*?

   - `IF X>0 THEN X:=X+1;`
   - `IF X<0 THEN X:=X+1 ELSE X:=X-1;`
   - `IF X>0 THEN X:=X+1 ELSE X<0;`
   - `IF X>0 THEN X:=X-1 ELSE X:=X+1`

2. According to the following BNF rules:

```
<arit>       ::= <num> + <num> | <num> - <num>
<expr>       ::= <var> = <arit> | <arit> = <var> | <expr> ; <expr>
<num>        ::= 1 | 2 | 3 | 4 | 5
<var>        ::= X | Y | Z
```

   which of the following claims is *true*?

   - `1+1` is an `<arit>` expression.
   - `1+2-3` is an `<arit>` expression.
   - `1+2=X` is an `<expr>` expression.
   - `Z=2+3;Y=1-4` is an `<expr>` expression.

3. Check whether the following C program is legal with respect to the grammar which describes the syntax of C-minus, a subset of C (see the appendix below). Provide and answer (yes/no) and a detailed explanation.

```
long factorial(int n) {
  int c = 2;
  long result = 1;

  while (c<=n) {
    result = result*c;
    c++;
    }

  return result;
}
```

4. Write the semantic rules that define the boolean disjunction.

5. Consider the following code P:

```
X:=5;
Y:=X
```

   (a) Write the execution trace that corresponds to the small-step semantics starting from the initial state $s_I = \{\}$.

   (b) Compute the *big-step* semantics for the initial state $s_I = \{\}$. Show the intermediate computations as well.

6. For the following configuration, develop the evaluation of the arithmetic expression by using the appropriate rules:

$$\langle \texttt{X} + \texttt{3}, \{X \mapsto 2\} \rangle \Rightarrow \dots$$

7. For the following configuration, develop the evaluation of the boolean expression by using the appropriate rules:

$$\langle \texttt{X} + \texttt{3} \leq \texttt{Y}, \{X \mapsto 2, Y \mapsto 0\} \rangle \Rightarrow \dots$$

8. Consider the following program P:

```
X:=5;
if X>3 then X:= X-1 else Y:=X
```

   (a) Write the *small-step* execution trace together with the intermediate computations for the initial state $\{X \mapsto 2\}$.

   (b) Compute now the *big-step* semantics for the same initial state $\{X \mapsto 2\}$ (again, show the intermediate computations)

9. We want to extend language SIMP with a new instruction *repeat* with the following syntax:

```
repeat i until b
```

This instruction works as follows: the (possibly compound) instruction `i` is executed until condition `b` is satisfied; then the execution of the instruction terminates.

(a) Provide the rules for the *small-step* semantic description of *repeat*.

(b) Provide the rules for the *big-step* semantic description of *repeat*

10. Consider the following program P:

```
X:=4;
while X>3 do X:= X-1
```

(a) Write the *small-step* execution trace (with the intermediate computations) from the empty state $s = \{\}$.

(b) Write the *big-step* execution tree (with the intermediate computations) from the empty state $s = \{\}$.

11. Consider a new *for* loop for SIMP as follows

```
for V:=a0 to a1 do i
```

where counter `V` takes increasing values from a0 to a1 (both included), as instruction `i` is executed.

(a) Provide the *small-step* semantic description corresponding to instruction *for*.

(b) Provide the *big-step* semantic description corresponding to instruction *for*.

12. We want to extend SIMP with a new instruction *times* with the following syntax:

```
do n times i
```

The instruction works as follows: instruction `i` is executed `n` times if `n` is a positive number; if $\mathtt{n} \leq 0$ then `i` is not executed.

(a) Provide the *small-step* semantic description for instruction *times*.

(b) Provide the *big-step* semantic description for instruction *times*.

13. Consider the following code S that computes the maximum of two numbers:

```
if X>Y then max:=X else max:=Y
```

(a) Obtain the *small-step* computational trace (with the intermediate computation) for the initial state $\{X \mapsto 3, Y \mapsto 5\}$.

(b) Obtain the *big-step* computational tree (with the intermediate computation) for the initial state $\{X \mapsto 3, Y \mapsto 5\}$.

14. We want to extend SIMP with a new multiple assignment operator with syntax

$$\mathtt{x}_1\mathtt{,x}_2\mathtt{,\ldots x}_n\ \mathtt{:=}\ \mathtt{a}_1\mathtt{,a}_2\mathtt{,\ldots a}_n$$

where

- variables $\mathtt{x}_i$ are distinct
- the $\mathtt{a}_i$ are expressions

and the assignment works as follows (see [Gries81], page 121):

- expressions $\mathtt{a}_1, \mathtt{a}_2, \ldots \mathtt{a}_n$ are first evaluated (the order does not matter) to obtain values $\mathtt{v}_1, \mathtt{v}_2, \ldots, \mathtt{v}_n$, respectively;
- for each i, variable $\mathtt{x}_i$ is given value $\mathtt{v}_i$.

(a) Provide the *small-step* semantic description for the *multiple assignment* operator

(b) Provide the *big-step* semantic description for the *multiple assignment* operator

15. Consider the following program S:

```
t:=x;
x:=y;
y:=t;
```

Write the *small-step* execution trace for the initial state $\{x \mapsto 2, y \mapsto 5\}$.

16. Consider the following program P:

```
x:=x+1;
y:=y+x;
x:=x+1;
```

Write the *small-step* execution trace for the initial state $\{x \mapsto 3, y \mapsto 7\}$.

17. Consider the following C code to compute the maximum of two numbers:

```
int maximum (int x, int y)
{
if (x>y)
return x ;
else
return y ;
} ;
```

(a) Write the code of `maximum` using SIMP syntax (where subprograms are not allowed).

(b) Write the *small-step* and *big-step* execution of `maximum(3,5)` (i.e., for the initial state $\{x \mapsto 3, y \mapsto 5\}$).

18. Compute the weakest precondition $wp(S, Q)$ for program S in question 15 and postcondition Q given by $\mathtt{x} = \mathtt{X} \wedge \mathtt{y} = \mathtt{Y}$. According to these results:

    (a) What is the functionality implemented by this program? Is there any difference between program S and the following program S' that uses the multiple assignment introduced in Exercice 14?

    ```
    x, y := y, x
    ```

    (b) When considering an axiomatic semantics, is there any difference between S and S' as above with regard to postcondition Q?

    (c) Are S and S' equivalent from the operational point of view, or there is a way to distinguish them?

19. Consider the following program S:

    ```
    t:=x;
    x:=y;
    y:=t;
    ```

    Given the precondition $P = (\mathtt{x{=}a} \wedge \mathtt{y{=}b} \wedge \mathtt{z{=}c})$ and the postcondition $Q = (\mathtt{x{=}b} \wedge \mathtt{y{=}a})$, is S correct with respect to P and Q (i.e., correctness of $\{P\}\ S\ \{Q\}$)? Use the weakest precondition calculus to prove it and show the steps of the correctness proof.

20. Consider the following program S:

    ```
    X:=X-1
    ```

    Given the precondition $P = (\mathtt{X{=}1})$ and the postcondition $Q = (\mathtt{X} \geq \mathtt{0})$, is S correct with respect to P and Q?. Use the weakest precondition calculus to prove it and show the steps of the correctness proof.

21. Compute the weakest precondition of the following programs S for the given postcondition Q:

    (a) $S = (\mathtt{x{:=}1})$, $Q = (\mathtt{x} = \mathtt{1})$.

    (b) $S = (\mathtt{x{:=}y})$, $Q = (\mathtt{x} = \mathtt{0})$.

    (c) $S = (\mathtt{x{:=}x{-}1})$, $Q = (\mathtt{x} = \mathtt{0})$.

    (d) $S = (\mathtt{x{:=}x{-}1})$, $Q = (\mathtt{y} > \mathtt{0})$.

    (e) $S = (\mathtt{if\ (x>0)\ then\ x{:=}y\ else\ y{:=}x})$, $Q = (\mathtt{x} \geq \mathtt{y} \wedge \mathtt{y} > \mathtt{0})$.

    (f) $S = (\mathtt{if\ (x>0)\ then\ x{:=}y\ else\ x{:=}y})$, $Q = (\mathtt{x} \geq \mathtt{y} \wedge \mathtt{y} > \mathtt{0})$.

    (g) $S = (\mathtt{if\ (x{=}0)\ then\ x{:=}1\ else\ x{:=}x{+}1})$, $Q = (\mathtt{x} > \mathtt{y} \wedge \mathtt{y} \leq \mathtt{0})$.

    (h) $S = (\mathtt{x{:=}y;\ y{:=}5})$, $Q = (\mathtt{x} > \mathtt{0})$.

    (i) $S = (\mathtt{x{:=}x{+}1;\ if\ (x>0)\ then\ x{:=}y\ else\ y{:=}x})$, $Q = (\mathtt{x} > \mathtt{0})$.

Part II: Test

22. According to the compilation scheme discussed in the course, which kind of error would be raised by the following Java statement?

```
int 3 = x;
```

$\boxed{\text{A}}$ A lexical error.

$\boxed{\text{B}}$ A syntactic error.

$\boxed{\text{C}}$ A semantical error.

$\boxed{\text{D}}$ There is no error.

23. Which of the following claims about the static semantics of a programming language is **TRUE**?

$\boxed{\text{A}}$ It considers those syntax restrictions that cannot be expressed using BNF but can be checked during the compilation.

$\boxed{\text{B}}$ During the compilation, the static semantics checkings are accomplished from lexical analysis to semantic analysis.

$\boxed{\text{C}}$ It considers those restrictions that can only be checked during the execution.

$\boxed{\text{D}}$ It considers those syntax restrictions that cannot be expressed using BNF but can be checked during the execution.

24. The outcome of the lexical analysis is:

$\boxed{\text{A}}$ A sequence of characters.

$\boxed{\text{B}}$ A sequence of tokens.

$\boxed{\text{C}}$ A sequence of instructions.

$\boxed{\text{D}}$ A syntactic tree.

25. Which of the following claims is **WRONG**?

$\boxed{\text{A}}$ The dynamic semantics is computed during the compilation.

$\boxed{\text{B}}$ The dynamic semantic can be used to investigate runtime properties or errors.

$\boxed{\text{C}}$ The operational semantics is a kind of dynamic semantics.

$\boxed{\text{D}}$ Static semantics does not suffice to detect, for instance, whether a division-by-zero condition will happen during the execution of the program.

26. According to the following execution using the small-step operational semantics:

$$\langle \textbf{while } X > 0 \textbf{ do } X := X + 1,\ \{X \mapsto 0\}\rangle \quad \to \quad \boxed{?}$$

fill the gap (denoted by $\boxed{?}$) with the appropriate expression:

$\boxed{\text{A}}$ $\{X \mapsto 0\}$.

$\boxed{\text{B}}$ $\{X \mapsto 1\}$.

$\boxed{\text{C}}$ $\langle \surd, \{X \mapsto 0\}\rangle$.

$\boxed{\text{D}}$ $\langle \surd, \{X \mapsto 1\}\rangle$.

27. Assume that the syntax of the Simple IMPerative language considered during the course is extended with a new instruction $\textbf{do } i \textbf{ loop } b$ so that instruction i is executed until condition b is satisfied, thus leaving the loop (note that i is executed at least once). If its *big-step* operational semantics is:

$$\frac{\langle i,\ e\rangle \Downarrow e' \qquad \langle b,\ e'\rangle \Rightarrow true}{\langle \textbf{do } i \textbf{ loop } b,\ e\rangle \ \Downarrow \ e'}$$

$$\frac{\langle i,\ e\rangle \Downarrow e' \qquad \langle b,\ e'\rangle \Rightarrow false \qquad \langle \textbf{do } i \textbf{ loop } b,\ e'\rangle \Downarrow e''}{\langle \textbf{do } i \textbf{ loop } b,\ e\rangle \ \Downarrow \ e''}$$

choose the appropriate value for s in the following step:

$$\langle \textbf{do } X := X + 1 \textbf{ loop } X = Y,\ \{X \mapsto 1, Y \mapsto 3\}\rangle \ \Downarrow \ s$$

$\boxed{\text{A}}$ $\{X \mapsto 3, Y \mapsto 3\}$

$\boxed{\text{B}}$ $\{X \mapsto 2, Y \mapsto 3\}$

$\boxed{\text{C}}$ $\{X \mapsto 1, Y \mapsto 3\}$

$\boxed{\text{D}}$ $\{X \mapsto 0, Y \mapsto 3\}$

28. For the operational semantics of the Simple IMPerative language considered during the course, how do you fill the gap in the following rule defining the subtraction $a_0 - a_1$?

$$\frac{\langle a_0, e\rangle \Rightarrow n_0 \qquad \langle a_1, e\rangle \Rightarrow n_1}{\langle a_0 - a_1, e\rangle \boxed{\quad ? \quad}} \quad n \text{ es la diferencia de } n_0 \text{ y } n_1$$

$\boxed{\text{A}}$ $\to \langle \surd, e[X \mapsto n]\rangle$

$\boxed{\text{B}}$ $\Rightarrow \langle \surd, \{e \mapsto n\}\rangle$

$\boxed{\text{C}}$ $\Downarrow n$

$\boxed{\text{D}}$ $\Rightarrow n$

29. Consider the Hoare triple $\{P\}$ $\texttt{x} := \texttt{x} + \texttt{y}$ $\{Q\}$, with Q the assertion $\texttt{y} = \texttt{0}$. Which of the following assertions P makes the triple *correct*?

$\boxed{\text{A}}$ $\texttt{x} = \texttt{x}$.

$\boxed{\text{B}}$ $\texttt{x} + \texttt{y} = \texttt{0}$.

$\boxed{\text{C}}$ $\texttt{y} = \texttt{0}$.

$\boxed{\text{D}}$ $\texttt{x} = \texttt{x} + \texttt{0}$.

30. Which is the weakest precondition of the following program `S`

```
x:= x+1;
y:= x+y
```

with respect to postcondition `{Q}={y>5}`

$\boxed{\text{A}}$ `pmd(S,Q)={x+y>4}`.

$\boxed{\text{B}}$ `pmd(S,Q)={x+y>=4}`.

$\boxed{\text{C}}$ `pmd(S,Q)={x>5}`.

$\boxed{\text{D}}$ `pmd(S,Q)={x+y>5}`.

31. Which of the following claims is $\boxed{\text{WRONG}}$?

$\boxed{\text{A}}$ The dynamic semantics is computed during the compilation.

$\boxed{\text{B}}$ The dynamic semantics allows us to deal with runtime properties and errors.

$\boxed{\text{C}}$ Operational semantics is a particular style to describe a dynamic semantics.

$\boxed{\text{D}}$ Static semantics does not suffice to, for instance, detect whether a division by zero will be attempted during the program execution.

32. Which of the following claims about the small-step operational semantics is $\boxed{\textbf{TRUE}}$?

$\boxed{\text{A}}$ The description of the evaluation of boolean and arithmetic expressions is the same as for the big-step semantics.

$\boxed{\text{B}}$ In every transition step the state is always modified.

$\boxed{\text{C}}$ The final state cannot be obtained from the last configuration in the trace.

$\boxed{\text{D}}$ Configurations consist of an assertion and a program instruction.

33. When using the big-step operational semantics, what should we write instead of the question mark in the following expression?

$$\langle \texttt{while x} > \texttt{0 do x} := \texttt{x} - 1, \{\texttt{x} \mapsto 1, \texttt{y} \mapsto 1\}\rangle \Downarrow \boxed{?}$$

$\boxed{\text{A}}$ $\langle \texttt{x} := \texttt{x} - 1; \texttt{while x} > \texttt{0 do x} := \texttt{x} - 1, \{\texttt{x} \mapsto 1, \texttt{y} \mapsto 1\}\rangle$.

$\boxed{\text{B}}$ $\{\texttt{x} \mapsto 0, \texttt{y} \mapsto 1\}$.

$\boxed{\text{C}}$ $\langle skip, \{\texttt{x} \mapsto 0, \texttt{y} \mapsto 1\}\rangle$.

$\boxed{\text{D}}$ $\langle \texttt{while x} > \texttt{0 do x} := \texttt{x} - 1, \{\texttt{x} \mapsto 0, \texttt{y} \mapsto 1\}\rangle$.

34. Which is the outcome of $\mathsf{wp}(\texttt{x} := \texttt{x} + \texttt{y}, \{\texttt{y} = \texttt{0}\})$?

$\boxed{\text{A}}$ $\{\texttt{x} = \texttt{x}\}$

$\boxed{\text{B}}$ $\{\texttt{x} + \texttt{y} = \texttt{0}\}$

$\boxed{\text{C}}$ $\{\texttt{y} = \texttt{0}\}$

$\boxed{\text{D}}$ $\{\texttt{x} = \texttt{x} + \texttt{0}\}$

35. Assume the following definition of the predicate transformer *wp* that associates its *weakest precondition* to a multiple assignment instruction and a predicate Q so that variables $\mathsf{x_i}$ are *simultaneously* replaced by the corresponding expressions $\mathsf{expr_i}$:

$$\mathsf{wp}(\text{“}\mathsf{x_0}, \mathsf{x_1}, \ldots, \mathsf{x_n} ::= \mathsf{exp_0}, \mathsf{exp_1}, \ldots, \mathsf{exp_n}\text{”}, \mathsf{Q}) = \mathsf{Q}[\mathsf{x_i} \mapsto \mathsf{exp_i}]_{\mathsf{i=0}}^{\mathsf{n}}$$

Which is the outcome of $\mathsf{wp}(\texttt{x}, \texttt{y} ::= \texttt{1}, \texttt{x} + \texttt{1}, \{\texttt{x} > 0, \texttt{y} \geq 1\})$?

$\boxed{\text{A}}$ $\{\texttt{x} > 1, \texttt{y} \geq 2\}$

$\boxed{\text{B}}$ $\{\texttt{x} > 0, 2 \geq 2\}$

$\boxed{\text{C}}$ $\{1 > 0, \texttt{x} \geq 0\}$ o también $\{1 > 0, \texttt{x} + 1 \geq 1\}$ o $\{\texttt{x} \geq 0\}$

$\boxed{\text{D}}$ $\{\texttt{x} > 1, \texttt{y} \geq 2\}$ o también $\{\texttt{x} > 0\}$

36. Consider the programs P_1 and P_2:

P_1:

```
x:=0;
if x>0 then y:=10
       else y:=5
```

P_2:

```
x:=0;
x:=x+5;
y:=x
```

We can say that:

$\boxed{\text{A}}$ P_1 and P_2 are equivalent, disregarding the considered semantics.

$\boxed{\text{B}}$ P_1 and P_2 are equivalent with respect to the big-step semantics.

$\boxed{\text{C}}$ P_1 and P_2 are equivalent with respect to the small-step semantics.

$\boxed{\text{D}}$ P_1 and P_2 are not equivalent, disregarding the considered big-step or small-step semantics.

37. Are the following programs equivalent?

```
x := 1;                     x := 3;
x := 2;                     while x > 5 do
x := 3;                        x := 1; x := 2;
```

A According to the big-step semantics, the answer is NO.

B According to the small-step semantics, the answer is NO.

C If the initial state is $\{\texttt{x} \mapsto \texttt{0}\}$, they are equivalent with respect to the small-step semantics.

D They cannot be equivalent.

38. Mark the **WRONG** claim:

A A compiler takes a source program and returns an object program.

B A object program receives input data and returns output data.

C An interpreter receives a source program and input data and returns output data.

D A source program receives input data and returns an object program.

39. How does a mixed implementation of a programming language work?

A The source code is first translated into an intermediate code, which is then interpreted.

B Some parts of the program are translated, whereas other (more difficult) parts are interpreted.

C The source code is first interpreted and then translated into machine language.

D The program is always translated into machine language, and then interpreted by the virtual machine of the processor.

A C-Minus BNF grammar

```
<ID>                    ::= <letter><letter>*
<NUM>                   ::= <digit><digit>*
<letter>                ::= a | ...| z | A | ...| Z
<digit>                 ::= 0 | ...| 9
<program>               ::= <declaration-list>
<declaration-list>      ::= <declaration-list> <declaration> | <declaration>
<declaration>           ::= <var-declaration> | <fun-declaration>
<var-declaration>       ::= <type-specifier> <ID> ; | <type-specifier> <ID> [ <NUM> ] ;
<type-specifier>        ::= int | void
<fun-declaration>       ::= <type-specifier> <ID> ( <params> ) <compound-stmt>
<params>                ::= <param-list> | void
<param-list>            ::= <param-list> , <param> | <param>
<param>                 ::= <type-specifier> <ID> | <type-specifier> <ID> [ ]
<compound-stmt>         ::= { <local-declarations> <statement-list> }
<local-declarations>    ::= <local-declarations> <var-declaration> | empty
<statement-list>        ::= <statement-list> <statement> | empty
<statement>             ::= <expression-stmt> | <compound-stmt> | <selection-stmt>
                            | <iteration-stmt> | <return-stmt>
<expression-stmt>       ::= <expression> ; | ;
<selection-stmt>        ::= if ( <expression> ) <statement>
                            | if ( <expression> ) <statement> else <statement>
<iteration-stmt>        ::= while ( <expression> ) <statement>
<return-stmt>           ::= return ; | return <expression> ;
<expression>            ::= <var> = <expression> | <simple-expression>
<var>                   ::= <ID> | <ID> [ <expression> ]
<simple-expression>     ::= <additive-expression> <relop> <additive-expression>
                            | <additive-expression>
<relop>                 ::= <= | < | > | >= | == | !=
<additive-expression>::= <additive-expression> <addop> <term> | <term>
<addop>                 ::= + | -
<term>                  ::= <term> <mulop> <factor> | <factor>
<mulop>                 ::= * | /
<factor>                ::= ( <expression> ) | <var> | <call> | <NUM>
<call>                  ::= <ID> ( <args> )
<args>                  ::= <arg-list> | empty
<arg-list>              ::= <arg-list> , <expression> | <expression>
```