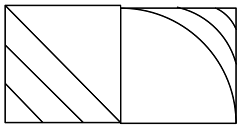
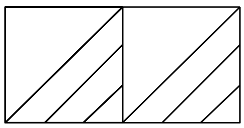
Principles of Programming Languages
Answers for small examination 1

Problem 1 Illustrate the quilts represented by the following expressions (1), (2), and (3) in the language Little Quilt.

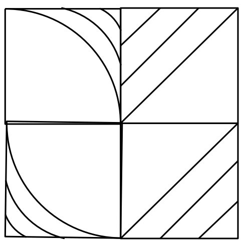
(1) `sew (turn (turn (b)), a)`



(2)
```
let
    val x = turn (b)
in
    sew (x,x)
end
```



(3)
```
let
    fun unturn (x) = turn (turn (turn (x)))
    fun pile (x,y) = unturn (sew (turn (y), turn (x)))
    val aa = pile (a, turn (turn (a)))
    val bb = pile (unturn (b), turn (b))
in
    sew (aa, bb)
end
```



The meaning of `a`, `b`, `turn`, `sew` are as follows. The other constructs of Little Quilt (`let` expressions, `val` declarations, `fun` declarations) have the meaning explained in the lecture.

- Expressions `a` and `b` represent the quilts in Figure 1 and Figure 2 respectively.
- The expression `turn` (e) represents the quilt obtained by rotating 90 degrees to the right the quilt represented by the expression e.
- The expression `sew` $(e_1$, $e_2)$ represents the quilt that is obtained by sewing the two quilts e_1 and e_2, where e_1 is in the left side and e_2 is in the right side, and they must have the same height.

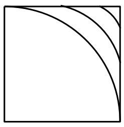


Figure 1: The quilt that `a` represents

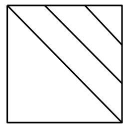
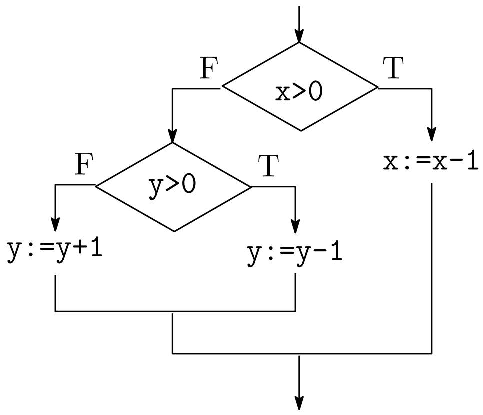


Figure 2: The quilt that `b` represents

Problem 2 Answer the following problems about the control flow in the imperative language presented in the lecture.

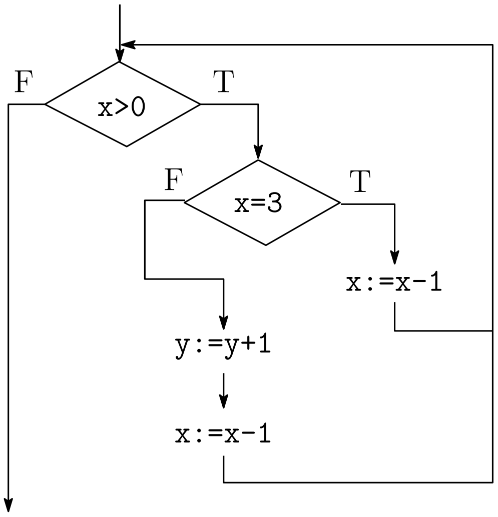
(1) Illustrate the control flow of the following program fragment.

```
if x>0 then x := x - 1
else if y>0 then y := y - 1
     else y := y + 1
```



(2) Illustrate the control flow of the following program fragment.

```
while x>0 do
    begin
        if x=3 then
            begin
                x := x - 1;
                continue
            end;
        y := y + 1;
        x := x - 1
    end
```



Problem 3

Derive the Hoare triples (1), (2), and (3) by using the rules presented in the lecture.

(1) $\{a = 3\}\ a := a + 1\ \{a = 4\}$

$$\dfrac{a = 3 \Rightarrow a + 1 = 4 \qquad \dfrac{}{\{a + 1 = 4\}\ a := a + 1\ \{a = 4\}}\text{(assign)} \qquad a = 4 \Rightarrow a = 4}{\{a = 3\}\ a := a + 1\ \{a = 4\}}\text{(conseq)}$$

As I said in the lecture, the logical expression $a = 4 \Rightarrow a = 4$ in the above proof tree may be omitted in this class as follows.

$$\dfrac{a = 3 \Rightarrow a + 1 = 4 \qquad \dfrac{}{\{a + 1 = 4\}\ a := a + 1\ \{a = 4\}}\text{(assign)}}{\{a = 3\}\ a := a + 1\ \{a = 4\}}\text{(conseq)}$$

(2) $\{a = 3\}\ a := a + 1; a := a + 2\ \{a = 6\}$

$$\dfrac{\dfrac{a = 3 \Rightarrow a + 1 = 4 \qquad \dfrac{}{\{a + 1 = 4\}\ a := a + 1\ \{a = 4\}}\text{(assign)}}{\{a = 3\}\ a := a + 1\ \{a = 4\}}\text{(conseq)} \qquad \dfrac{a = 4 \Rightarrow a + 2 = 6 \qquad \dfrac{}{\{a + 2 = 6\}\ a := a + 2\ \{a = 6\}}\text{(assign)}}{\{a = 4\}\ a := a + 2\ \{a = 6\}}\text{(conseq)}}{\{a = 3\}\ a := a + 1; a := a + 2\ \{a = 6\}}\text{(composition)}$$

In this proof, I omitted $a = 4 \Rightarrow a = 4$ and $a = 6 \Rightarrow a = 6$ in the applications of the consequence rule.

(3) $\{x = 5\}$ **while** $x > 0$ **do** $x := x - 1\ \{x = 0\}$

Due to space restriction, I write the proof tree by separating it into two parts.

$$\dfrac{x = 5 \Rightarrow x \geq 0 \qquad \dfrac{\text{(I write this part below.)}}{\{x \geq 0\}\ \textbf{while}\ x > 0\ \textbf{do}\ x := x - 1\ \{x \geq 0 \land \neg\, x > 0\}} \qquad x \geq 0 \land \neg\, x > 0 \Rightarrow x = 0}{\{x = 5\}\ \textbf{while}\ x > 0\ \textbf{do}\ x := x - 1\ \{x = 0\}}\text{(conseq)}$$

$$\dfrac{\dfrac{x \geq 0 \land x > 0 \Rightarrow x - 1 \geq 0 \qquad \dfrac{}{\{x - 1 \geq 0\}\ x := x - 1\{x \geq 0\}}\text{(assign)} \qquad x \geq 0 \Rightarrow x \geq 0}{\{x \geq 0 \land x > 0\}\ x := x - 1\{x \geq 0\}}\text{(conseq)}}{\{x \geq 0\}\ \textbf{while}\ x > 0\ \textbf{do}\ x := x - 1\ \{x \geq 0 \land \neg\, x > 0\}}\text{(while)}$$

In the above proof tree, the logical expression $x \geq 0 \Rightarrow x \geq 0$ may be omitted as follows.

$$\dfrac{\dfrac{x \geq 0 \wedge x > 0 \Rightarrow x - 1 \geq 0 \quad \dfrac{}{\{x - 1 \geq 0\}\ x := x - 1 \{x \geq 0\}}\text{(assign)}}{\{x \geq 0 \wedge x > 0\}\ x := x - 1 \{x \geq 0\}}\text{(conseq)}}{\{x \geq 0\}\ \textbf{while}\ x > 0\ \textbf{do}\ x := x - 1\ \{x \geq 0 \wedge \neg\, x > 0\}}\text{(while)}$$

I abbreviated the assignment axiom as assign, the consequence rule as conseq, the while rule as while, and the composition rule as composition.

Problem 4

Show the output produced by executing the following Pascal program. When the keyword `var` is attached to a formal parameter, it designates the parameter as call-by-reference. The procedure `writeln` writes out to the standard output the value of the parameter and a new line character.

```
program test;
var x : integer;
var y : integer;
procedure swap
  (var x: integer;
   var y : integer);
var z : integer;
begin
   z := x; x := y; y := z
end;
begin
   x := 3;
   y := 4;
   swap (x,y);
   writeln (x);
   writeln (y)
end.
```

```
4
3
```

Problem 5

Show the output produced by executing the following Pascal program. Note that Pascal is statically (lexically) scoped.

```
program P;
var n : char;
procedure W;
begin
   writeln(n)
end;
procedure D;
var n : char;
begin
   n := 'D';
   W
end;
begin
   n := 'L';
   W;
   D
end.
```

```
L
L
```

Problem 6

Show the meaning of the following programs (1) and (2) by using the rules presented in the lecture. Note that the programs are in the small subset of C presented in the lecture. Let the states before executing the programs both to be $\sigma = \{(\mathtt{X}, 3), (\mathtt{Y}, 1), (\mathtt{Z}, 0)\}$.

(1) `2`

$$< \mathtt{2}, \sigma > \longrightarrow 2$$

So in the state σ, executing the program `2` results in 2.

(2) `((2+3)*X)`

$$\dfrac{\dfrac{< \mathtt{2}, \sigma > \longrightarrow 2 \quad < \mathtt{3}, \sigma > \longrightarrow 3}{< (\mathtt{2} + \mathtt{3}), \sigma > \longrightarrow 5} \quad < \mathtt{X}, \sigma > \longrightarrow 3}{< ((\mathtt{2} + \mathtt{3}) * \mathtt{X}), \sigma > \longrightarrow 15}$$

So in the state σ, executing the program `((2+3)*X)` results in 15.