Imperative Programming (Sheet #2)

Marius Gavrilescu

5.1
```
PROCEDURE printlr (x : INTEGER);
BEGIN
    IF x # 0 THEN
        printlr(x DIV 10);
        Out.Int(x MOD 10, 0);
    END
END printlr;
```

5.3 When x and y are different variables and $P = Q$, or x and y are the same variables and P is $Q[x := 0]$

$tricky(y)$ has no effect. $tricky(x)$ sets x to 0.

6.1
```
PROCEDURE fib(x : INTEGER) : INTEGER;
VAR i, ans : INTEGER;
BEGIN
    i := 0;
    WHILE i < depth DO
        Out.String("|  ");
i := i + 1;
    END;
    Out.String("fib(");
Out.Int(x, 0);
Out.String(")");
Out.Ln;

    depth := depth + 1;
    IF x = 0 THEN ans := 0
ELSIF x = 1 THEN ans := 1
ELSE ans := fib(x-1) + fib(x-2) END;
    depth := depth - 1;

    i := 0;
    WHILE i < depth DO
    i := i + 1;
        Out.String("|  ")
    END;
    Out.String("= ");
Out.Int(ans, 0);
Out.Ln;

    RETURN ans;
END fib;
```

7.1 Call f with powers of two until it returns true. The last argument to f is an upper bound that on x that is not bigger than $2x$.

Now that we know the bound, we can determine the bits of x in turn. For the ith bit we can call f with the first $i-1$ bits, an one on the ith position, and zeros afterwards. If this returns true then we know the ith bit is 0, otherwise we know the ith bit is 1.

For the IntSqrt procedure, this idea would take $O(\log y)$ time to reduce the time to $O(\log \sqrt{y}) = O(\frac{1}{2} \log y) = O(\log y)$. It is not worth doing this.

7.2 The procedure will most likely not give the right result as it still assumes the previous invariant is true. It will keep reducing the interval based on the relation between its middle and t, and when the interval reaches 1 element it will just return that element's index.

8.1
```
i := 0;
WHILE i < N DO
    j := i;
    WHILE (j > 0) & (a[j-1] > a[j]) DO
        x := a[j];
        a[j] := a[j-1];
        a[j-1] := x;
        j := j - 1;
    END;
    i := i + 1;
END;
```

Insertion sort is $O(n^2)$ both in number of comparisons and in execution time.

8.2 The order of calls does not affect the size of the call stack.

```
PROCEDURE P(l, r : INTEGER);
BEGIN
    WHILE <expression> DO
        <commands>;
        l := l_1;
        r := r_1;
    END;
END P;

PROCEDURE sort(l, r : INTEGER);
BEGIN
    WHILE r - l > 1 DO
        i := partition(l, r);
        IF i - l < r - i THEN
            sort(l, i);
            l := i + 1;
        ELSE
            sort(i + 1, r);
            r := i;
        END
    END;
END sort;
```