CvP - Werkcollege 9

Exercise 1 Consider the following C++ skeletal program:

```
class Big {
    int i;
    float f;
    void fun1() throw int {
        . . .
        try {
            . . .
            throw i;
            . . .
            throw f;
            . . .
        }
        catch(float) { . . . }
        . . .
    }
}
class Small {
    int j;
    float g;
    void fun2() throw float {
        . . .
        try {
            . . .
            try {
                Big.fun1();
                . . .
                throw j;
                . . .
                throw g;
                . . .
            }
            catch(int) { . . . }
            . . .
        }
        catch(float) { . . . }
```

```
    }
}
```

In each of the four `throw` statements, where is the exception handled? Note that `fun1` is called from `fun2` in class `Small`.

Exercise 2 Explain why exceptions can lead to memory leaks in a language that is not garbage collected.

Exercise 3 Recall that a function is tail recursive if all recursive calls are tail calls. Consider the Takeuchi function `tak` defined below.

```
(define (tak x y z)
  (if (< y x)
    (tak
      (tak (- x 1) y z)
      (tak (- y 1) z x)
      (tak (- z 1) x y)
    )
    z
  )
)
```

The recursive function `tak` runs for a *very* long time (Try it!). Therefore, this function is often used as a benchmark for languages with optimization for recursion. Is `tak` tail recursive?

Exercise 4 Rewrite the following Scheme functions as a tail-recursive function:

(a)
```
(define (doit n)
  (if (= n 1)
    1
    (* n (doit (- n 1)))
  )
)
```

(b)
```
(define (reverse list)
  (if (null? list)
    '()
    (append
        (reverse (cdr list))
        (cons (car list) '())
    )
  )
)
```

(c)
```
(define (fib n)
     (cond
       ((= n 0) 0)
       ((= n 1) 1)
       (else (+ (fib (- n 1)) (fib (- n 2))))
     )
   )
```