Prof. P. Thiemann, S. Wehr Summer Term 2009

Concepts of Programming Languages

`http://proglang.informatik.uni-freiburg.de/teaching/konzepte/2009ss/`

Exercise Sheet 1

2009-04-23

To following preparations are necessary before commencing with the exercises.

- Install the latest Version of PLT Scheme (Version 4.1.5, `http://www.plt-scheme.org/`). (DrScheme is already installed in the labs of the faculty.)
- Run `drscheme`. (In the labs, you possibly have to execute `setup drscheme` first.)
- Choose *Essentials of Programming Languages* as DrScheme's language (menu Language → Choose Language ...).
- Insert the following line at the beginning of your file:

  ```
  (require (planet schematics/schemeunit:3:4))
  ```

  This line imports SchemeUnit, a unit testing framework for Scheme. If SchemeUnit is not installed on your machine, the `require` installs it automatically.

Follow these steps to solve the exercises:

- Write a type signature for the procedure.
- Write a short description of the procedure.
- Write sensible test cases. To write a test case, you should use the construct `check-equal?`. `check-equal?` takes three arguments: the first argument is the expression under test, the second argument the expected result, and the third argument, which is optional, is a description of the test case.

  The exercises already define one test case. However, to cover all possible cases, further tests are needed.
- Start with the actual implementation only if you have completed the three steps above.

Exercise 1

Write a procedure `nth-element` that takes a list `lst` and a zero-based index `n`; the procedure then returns the `n`-th element of `lst`.

```
(check-equal? (nth-element '(a b c d) 2) 'c)
```

Exercise 2

Write a procedure `occurs-free?` for checking which variables occur free in a lambda expression. The book *Essentials of Programming Languages* contains a definition of free variables.

```
(check-equal? (occurs-free? 'x '(lambda (x) (x y))) #f)
```

Exercise 3

Write a procedure `duple` that takes a natural number `n` and a value `x`; the procedure then returns a list consisting of `n` repetitions of `x`.

```
(check-equal? (duple 3 'a) '(a a a))
```

Exercise 4

Write a procedure `down` that takes a list `lst` and returns another list such that every top-level element of `lst` is wrapped inside parenthesis.

```
(check-equal? (down '(1 2 3)) '((1) (2) (3)))
```

Exercise 5

Write a procedure `count-occurrences` that takes a symbol `s` and a s-list `slist`; the procedure then returns the number of occurrences of `s` in `slist`.

```
(check-equal? (count-occurrences 'x '(x (f x) (g (g y x)))) 3)
```

Exercise 6

Write a procedure `product` that takes two lists of symbols `sos1` und `sos2`; the procedure returns a list of two-element lists, which represents the cartesian product of `sos1` und `sos2`.

```
(check-equal? (product '(a b c) '(x y)) '((a x) (a y) (b x) (b y) (c x) (c y)))
```

Exercise 7

Write a procedure `flatten` that takes a s-list `slist` and returns a list of all symbols contained in `slist`. The ordering of the symbols in `slist` should be retained.

```
(check-equal? (flatten '((a) () (b ()) () (c))) '(a b c))
```

Exercise 8

Write a procedure `exists?` that takes a predicate `pred` (i.e. a one-argument procedure returning a boolean value) and a list `lst`; `exists?` then returns `#t` if, and only if, at least one element of `lst` satisfies `pred`.

```
(check-equal? (exists? number? '(a b 1 c)) #t)
```

Submission: Via email to `wehr@informatik.uni-freiburg.de`. Please submit in pairs of two. Your code must not raise errors when pressing DrScheme's "Run" bottom. The strict submission deadline is **2009-04-30, 2 pm.**