Exam 2, Computer science 356, Fall 2014

Time allowed: 75 minutes. Total points: 75.

Name: Solution

Question 1. (5 points) The fictitious programming language Qava requires identifiers to begin with an uppercase letter, followed by any combination of uppercase letters, lowercase letters, and digits. For example, `Aj32h5`, `T` and `Pabc` are legal Qava identifiers, but `abc` and `56PQR` are not. Give a regular expression (using the regular expression notation from our lecture notes and/or JFLex) for Qava identifiers.

[A-Z] [a-zA-Z0-9]*

Question 2. (5 points) Explain, in your own words, why it is sometimes necessary to specify the associativity of an operator when using a parser generator such as CUP.

Many common operations (such as addition and multiplication) are ambiguous if the associativity is not specified

e.g. 2+3+4.

Question 3. (5 points) In your opinion, what is the most significant difference between imperative languages and functional languages? Briefly justify your answer.

Many answers are possible. Any well-justified answer is accepted.

Question 4. (15 points) Write a Scheme function equivalent to the following C function. Your solution does not need to be tail recursive, and it only needs to work for nonnegative integer arguments.

```
int kangaroo(int m, int n)
{
  int val = 3;
  int j;
  for(j=m; j<n; j++){
    val = (val + m) * j;
  }
  return val;
}
```

```
(define (kangaroo m n)
  (letrec ((kangaroo-helper
            (λ (m n val j)
              (if (>= j n) val
                  (let ((new-j (+ j 1))
                        (new-val (* j (+ val m))))
                    (kangaroo-helper m n new-val new-j))))))
    (kangaroo-helper m n 3 m)))
```

Question 5. (15 points) Convert the following Scheme function into an equivalent tail recursive Scheme function. Your function need only work correctly for nonnegative integer arguments.

```
(define (platypus x)
  (if (= x 0) -1
      (+ (* x 3) (platypus (- x 1)))))
```

```
(define (platypus x)
  (letrec ((platypus-helper
            (λ (x i tot)
              (if (= i x) tot
                  (letrec ((new-i (+ i 1))
                           (new-tot (+ (* new-i 3) tot)))
                    (platypus-helper x new-i new-tot))))))
    (platypus-helper x 0 -1)))
```

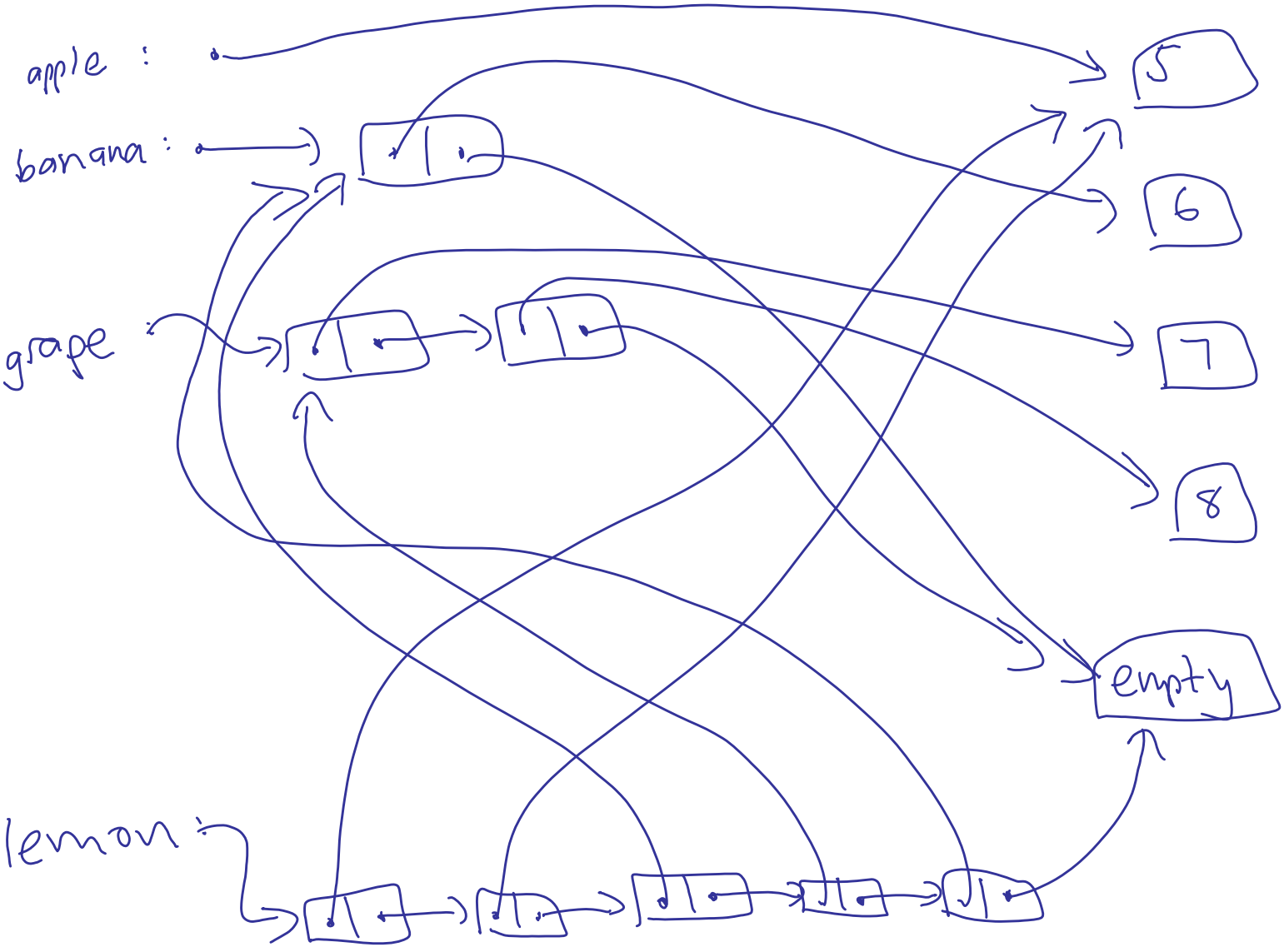
Question 6. (7 points) Curry the Scheme function `p` below into a Scheme function `q`. That is, define `q` such that `(p a b)` evaluates to the same thing as `((q a) b)`. You may not use `p` in your definition of `q`. You must use only elementary arithmetic functions and standard Scheme keywords.

```
(define (p a b)
  (let ((x (+ b (* a a)))
        (y (- a b)))
    (* x y)))
```

```
(define (q a)
  (λ (b)
    (let ((x (+ b (* a a)))
          (y (- a b)))
      (* x y))))
```

Question 7. (15 points) Draw a diagram, similar to the diagrams in the lecture notes, showing the memory layout of all pairs and pointers to pairs that results from the following Scheme definitions:

```
(define apple 5)
(define banana (list 6))
(define grape (list 7 8))
(define lemon (list 5 apple banana grape banana))
```



Question 8. (8 points) What is the output of the following Scheme code? For each function that produces output, write the output directly to the right of the function that produces the output. The number of points for each output value are listed as comments in the code.

```
(define w (list 3 4 5))
(define x (list 6 7 8))
(define y (list 9 10 11))
(define z (list w x y))

; 1 point
(car x)        6

; 1 point
(cdr x)        (list 7 8)

; 1 point
(car z)        (list 3 4 5)

; 1 point
(cadr z)       (list 6 7 8)

; 1 point
(caadr z)      6

(define (alpha f i j k) (apply f (list i j k)))

; 1 point
(alpha * 2 3 4)    24

(define (beta f i j k) (map f (list i j k)))

; 1 point
(beta add1 2 3 4)    (list 3 4 5)

; 1 point
(apply * (beta add1 2 3 4))
                              60
```

Question 9. *[This question is worth only three points of extra credit. Do not attempt it until you have completed and checked your answers to all previous questions.]* What is the output of the following Scheme code? Write your answers next to the functions that produce output.

```
(define (zuchini f init-val lst)
  (letrec ((zuchini-helper
            (lambda (f val lst)
              (if (null? lst) val
                  (if (list? lst) (f (zuchini-helper f init-val (car lst))
                                     (zuchini-helper f val (cdr lst)))
                      (f val lst 1))))))
    (zuchini-helper f init-val lst)))
```

flattens any lists in the input

calls f on an atom (1st) with accumulated val and constant 1

```
; 1 point of extra credit
(zuchini = 5 empty)
```

5

```
; 1 point of extra credit
(zuchini * 1 (list 2 3 (list 3 4)))
```

72 [= (2×1×1)×(3×1)×(3×1)×(4×1)]

```
; 1 point of extra credit
(zuchini + 0 (list 2 3 (list 3 (list 8 1) 4)))
```

27 [= (2+0+1)+(3+1)+(3+1)+(8+1)+(1+1)+(4+1)]

Total points: 75