Introduction to Computer Science (CSL102) Minor II October 13, 2012

Name: **Entry:** **Grp:**

Note: Maximum marks : 60. All questions carry equal marks. All notations standard (as done in class). Answer only in the space provided. 4 pages in all

---

1. The *Bisection method* is applicable when we wish to solve the equation $f(x) = 0$ for a real variable x, where f is a continuous function defined on an interval $[a, b]$ and $f(a)$ and $f(b)$ have opposite signs. In this case a and b are said to bracket a root since, by the *intermediate value theorem*, f must have at least one root in the interval (a, b). At each step the method divides the interval in two by computing the midpoint $c = (a+b)/2$ of the interval and the value of the function $f(c)$ at that point. Then, either $f(c)$ is a root in which case an infitesimal interval around c is output, or the search for a root continues in either $[a, c]$ or $[c, b]$ depending on the sign of $f(c)$ (remember that the search interval needs to bracket a root).

   User the following higher order `ML` procedure

```
fun iter(f,acceptable,update,guess,epsilon) =
        if acceptable(guess,f,epsilon) then guess
        else iter(f,acceptable,update,update(guess,f),epsilon);

val iter = fn : 'a * ('b * 'a * 'c -> bool) * ('b * 'a -> 'b) * 'b * 'c -> 'b
```

   to define the following function to implement the *Bisection method* method. The output should be an infitesimal interval of size less than `eps` that brackets the root.

   Solution: (See http://www.cse.iitd.ac.in/∼suban/CSL102/programs/iter.ml)

```
fun bisection(f,interv,eps) =
        let
                fun acceptable(interv,f,eps) =
                        let
                                val (a,b) = interv;
                        in
                                ((b - a) <= eps)
                        end;
                fun update(interv,f) =
                        let
                                val (a,b) = interv;
                                val c = (a+b)/2.0
                        in
                                if abs(f(c)) < eps then (c-eps/2.0,c+eps/2.0)
                                else if (f(a)*f(c) < 0.0) then (a,c)
                                else (c,b)
                        end
        in
            iter(f,acceptable,update,interv,eps)
        end;
```

2. Consider the sequence $q = 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, \ldots$ which includes 1 and all numbers divisible by no primes other than 2, 3 and 5. Complete the following `Python` function that stores the first 1000 values of the sequence in an array `q[0..999]`.

   Solution: (See http://www.cse.iitd.ac.in/∼suban/CSL102/programs/hamming.py. It may also be instructive to check out Dijkstra's notes on this problem.)

```
def hamming(q,n):
        q[0] = 1
        # complete the rest of the initialization

        i,x2,x3,x5,j2,j3,j5 = 1,2,3,5,0,0,0

        #INV: q[0..i-1] contains the first i values of the sequence; 1<=i<= n.
        #    x2 = 2*q[j2] is the minimum value > q[i-1] with the form 2*x for x
        #         in q[0..i-1]
        #    x3 = 3*q[j3] is the minimum value > q[i-1] with the form 3*x for x
        #         in q[0..i-1]
        #    x5 = 5*q[j5] is the minimum value > q[i-1] with the form 5*x for x
        #         in q[0..i-1]
        while (i < n):
                q[i] = min(x2,x3,x5)
                i = i+1
                #INV: 0 <= j2 <= i-1; x2 = 2*q[j2]
                while (x2 <= q[i-1]):
                        j2,x2 = j2+1,2*q[j2]
                #assert: x2 = 2*q[j2] is minimum value greater than q[i-1] with
                #        the form 2*x for x in q[0..i-1]
                #INV and assesrtion similar to above
                while (x3 <= q[i-1]):
                        j3,x3 = j3+1,3*q[j3]
                #ditto
                while (x5 <= q[i-1]):
                        j5,x5 = j5+1,5*q[j5]
        #assert: q[0..n-1] contains the sequence
```

3. Consider two n digit arrays, $a[0..n-1]$ and $b[0..n-1]$ representing two n digit non-negative decimal numbers, with $a[0]$ and $b[0]$ storing the least significant digits and $a[n-1]$ and $b[n-1]$ storing the most significant digits. Complete the following function to add the two numbers in their array representations and storing the result in an array c.

   Solution: (See http://www.cse.iitd.ac.in/∼suban/CSL102/programs/add.py)

```
def add(a,b,c,n):
        i = 0
        carry = 0
        #INV: 0 <= i <= n; c[0..i-1] contains the sum of a[0..i-1] and
        #     b[0..i-1], and carry is carry on to the i_th position
        while (i < n):
                sum = a[i]+b[i]+carry
                c[i] = sum%10
                carry = sum/10
                i = i+1
        c[i] = carry
        #assert: c[0..n] contains the sum of a[0..n-1] and  b[0..n-1]
```

Bonus (10 marks): In a language like `ML` which can manipulate functions, we can get by without numbers (at least insofar as non-negative integers are concerned) by implementing 0 and the operation of adding 1 as

```
val zero = fn f => fn x => x;
val succ = fn n => fn f => fn x => f ((n f) x);
```

This representation is known as Church numerals, after the logician Alonzo Church who invented λ-calculus.

1. Explain how the above definitions work.
2. Define `one` and `two` directly (not in terms of `zero` and `succ`).
3. Define `add` directly

You may submit your solution in the CSL102 section of https://sakai.iitd.ac.in latest by 23:59:59 today.
Solution: It is still a good idea to try to figure it out for yourself.

Rough work: