CS109A Assignment #2
Due 3:15 PM, 1/26/96

Written Homework (Don't Forget Programming on Page Two!)

H1. (10 points) Numbers of the form $t_n = n(n+1)/2$ are called *triangular numbers,* because marbles arranged in an equilateral triangle, n on a side, will total $\sum_{i=1}^{n} i$ marbles, which we saw in Exercise 2.3.1(a) is t_n marbles. For example, bowling pins are arranged in a triangle 4 on a side and there are $t_4 = 4 \times 5/2 = 10$ pins. Show by induction on n that $\sum_{j=1}^{n} t_j = n(n+1)(n+2)/6$. (Exercise 2.3.2, p. 41, FCS.)

Aside: Numbers of the form $n(n+1)(n+2)/6$ are sometimes called *pyramidal numbers.* They count the number of cannonballs, e.g., arranged in a pyramid. For example, a pyramid with a side of 3 has $3 \times 4 \times 5/6 = 10$ cannonballs.

H2. (15 points) The formula for the sum of n terms of a geometric series

$$a + ar + ar^2 + \cdots + ar^{n-1}$$

is

$$\sum_{i=0}^{n-1} ar^i = \frac{(ar^n - a)}{(r-1)}$$

Prove this formula by induction on n. Note that you must assume $r \neq 1$ for the formula to hold. Where do you use that assumption in your proof? (Exercise 2.3.9, p. 42, FCS.)

H3. (15 points) Consider an expression E whose operators are all binary. The *length* of E is the number of symbols in E, counting an operator or a left or right parenthesis as one symbol, and also counting any operand such as 123 or `abc` as one symbol. Prove that E must have an odd length. *Hint*: Prove the claim by complete induction on the length of the expression E. (Exercise 2.4.6, p. 51, FCS.) *Note*: Some people may be disturbed by the fact that we are proving a statement that says "if E is of length n then n is odd." What if E is of even length? Aren't we proving that an even integer is odd? Yes, we would, but the whole point of the problem is that there *are* no E's of even length. This is a good example of a statement that holds vacuously; when n is even (i.e., there aren't any E's of even length).

H4. (10 points) Show that for any $n \geq 1$ there is an error-detecting set of strings of length n, using only the digits 0, 1, and 2, that has 3^{n-1} strings. (Exercise 2.3.5, p. 42, FCS.) *Hint*: You need to read Example 2.5 and learn about error-detecting codes; this material will *not* be covered in class. Then, generalize the "parity-check" idea of that example. The proof does *not* involve induction. Rather, simply show how to construct the code by describing its member strings, just as was done in Example 2.5.

Programming Problems

Just to set the stage, over the next few weeks we're going to write two multiplication algorithms that work for binary numbers of arbitrary length, and we're going to measure and compare their running times. Since weekly assignments will build upon previous ones, it is important that you **save the code you write each week**.

In this assignment, you will write some simple ML functions that perform operations that your multiplication algorithms will need. All functions take bit-string arguments of the form described in the second programming problem of HW1. That is, the type of an integer represented as a bit string is `int * (int list)`. The intention is that the first component (the integer by itself) tells how long the list in the second component is, and that list is the bits of the represented integer, with the low-order bit first. We shall call this representation *binary integer*.

For example, the binary-integer representation of (decimal) 13 is `(4, [1,0,1,1])`, because the binary representation of 13 is 1101. The reverse of this list of bits is `[1,0,1,1]`, and the length of this list is 4.

As a special case, the binary integer `(0,nil)` represents integer 0. However, the integer 0 has other representations such as `(1,[0])` or `(2,[0,0])`. In fact, one can append a 0 onto any list, add one to the length component and get another representation of the same integer.

P1. (10 points) Write a function `genOnes(n)` that produces a list of n `1`'s. Show the result of `genOnes(0)` and `genOnes(5)`.

P2. (5 points) Write a function `genBinInt(n)` that produces a representation of the integer $2^n - 1$ as a binary integer. That is, you should produce `(n,[1,1,...,1])`, where there are n `1`'s in the list. *Hint*: Use your function from P1. Show the result of `genBinInt(0)` and `genBinInt(5)`.

P3. (15 points) Write a function `shift(n,x)`, where n is an integer and x is a binary integer (i.e., its type is `int * (int list)`. The result of this function is a binary integer that that has 2^n times the value of x. You produce this binary integer by adding n to the length and placing n `0`'s in front of the list. *Hint*: As (almost) always, think recursively. Show the result of `shift(0,genBinInt(5))` and `shift(4,genBinInt(5))`.

P4. (20 points) Write a function `break(n,x)` that takes an integer n and a binary integer x and returns a pair of binary integers (y, z). Here, y must be the binary integer formed from the first n bits of x and z is the binary integer formed from the remaining bits of x. For example, if x is `(7,[0,1,1,1,1,0,1])` then `break(4,x)` produces the pair

```
((4,[0,1,1,1]), (3,[1,0,1]))
```

Don't worry about what happens if the list component of x is of length less than n. Show the result of `break(0,genBinInt(5))` and `break(4,(7,[0,1,1,1,1,0,1]))`.