EXERCISES UNIT 6

1. The specification of a function for calculating the scalar product of two vectors of real numbers is the following:

```
func prod(v1, v2: vector[1..N] de Real) dev p: Real
{Pre: cierto}
{Post:  p = Σα: 1 ≤ α ≤ N: (v1[α] * v2[α]) }
```

a) Do an embedding of the specification **weakening the postcondition**. Implement the function.
b) Do an embedding of the specification **strengthening the precondition**. Implement the function.

2. Repeat the last exercise (only section a) for a function that decides if a vector of natural numbers is palindrome.

3. We have a recursive function that calculates the power of two natural numbers (a^b):

```
unsigned elev(unsigned a, unsigned b)
{
    unsigned e;

    if(b == 0)
        e = 1;
    else
        e = elev(a, b – 1) * a;
    return e;
}
```

a) Transform the previous function to a tail recursive function (**g_elev**) using the method of unfold/fold transformation.

b) Write a function (**elev2**) that calculates the power of two natural numbers doing a call to **g_elev** with the appropriate parameters.

c) Transform the previous functions (**elev2** + **g_elev**) to an iterative function (**elev3**).

4. We have a recursive function that calculates how many times the element **e** is inside the stack **p**:

```
func contar(p: pila; e: elem) dev n: nat
si nula(p)
    n ← 0;
sino
    si cima(p) = e
        n ← contar(desapilar(p),e);
        n ← n + 1;
    sino
        n ← contar(desapilar(p),e);
    fsi
fsi
```

a) Transform the previous function to a tail recursive function (**g_contar**) using the method of unfold/fold transformation.

b) Write a function (**contar2**) that calculates the same result as **contar** doing a call to **g_contar** with the appropriate parameters.

c) Transform the previous functions (**contar2 + g_contar**) to an iterative function (**contar3**).

5. Given the following recursive function:

```
func f(v: vector[1..N] de Real; i: Nat) dev p: Real
  si i = N+1
      p ← 2
  sino
      p ← f(v, i+1);
      p ← p * v[i]
  fsi
{Post:    p = 2 * (Πα: i ≤ α ≤ N: v[α]) }
```

a) Transform the previous function to a tail recursive function (**f1**) using the method of unfold/fold transformation. Write all the intermediate calculations.

b) Write a function (**f2**) that calculates the same result as **f** doing a call to **f1** with the appropriate parameters.

c) Transform the previous functions (**f1 + f2**) to an iterative function (**f3**).

6. The Warshall algorithm is very similar to the Floyd algorithm, but in this case we do not want to know the shortest path but only if there is a path. Write the algorithm modifying as appropriate the Floyd algorithm. The adjacency matrix is of bool type.

7. Modify the algorithm "*viaje*" shown in the slides so it also returns the path, and not only the price. Write a function that has as input parameters 2 trading post and the 2 matrices from the previous algorithm and shows the whole path between 2 trading posts and its cost.

8. We want to calculate the matrix (M) which is a **chained matrix multiplication** of n matrices: $M = M_1M_2\ldots M_n$, but minimizing the total number of scalar multiplications. Remember that matrix multiplication is associative but not commutative, so we can choose which multiplications we want to do first, but we cannot change the order of the matrices.
Suppose M_i is of dimension $d_{i-1}xd_i$ and so the M_iM_{i+1} multiplication requires a total of $d_{i-1}d_id_{i+1}$ scalar operations.
We call M(i,j) to the minimum number of scalar multiplications needed for the calculation of the chained matrix multiplication of $M_iM_{i+1}\ldots M_j$. Therefore, the solution to the problem is M(1,n).

a) Write M(i,j) with a recursive formula.

b) From the previous formula, write a program for calculating M(1,n) with dynamic programming.

c) Modify the previous program so it also returns the order in which the multiplications have to be done.