Programación III
Informe de la Práctica 1
Verificación Formal con SPARK 2014
Curso 2020/2021

1. Nombre y apellido de los miembros del equipo.
   Aarón Hernández Álvarez
   Juan José Bello Santana

2. Listado enumerado con el nombre y tipo (procedimiento/función) de los procedimientos y funciones verificados.

   1. *function Es_Par (Number : Integer) return Boolean*

   -- Retorna verdadero o falso si el entero pasado por parámetro es par o no. Discrimina indistintamente del signo.

   2. *function Es_Primo (Number : Integer) return Boolean*

   -- Retorna verdadero o falso si el entero pasado por parámetro es primo o no. Verifica además que el entero se encuentre en un rango de valores concreto (positivos).

   3. *procedure cuentaPrimos (Vector : T_Table)*

   -- Cuenta el número de primos en un vector y lo almacena en una variable global.

   4. *procedure primerPrimo (Vector : T_Table, Resultado : Natural)*

   -- Calcula el primer primo que se encuentra en un vector. Modifica para ello la variable "Resultado".

   5. *procedure histoParImpar (Vector :T_Table)*

   -- Calcula los números pares e impares que contiene un vector. Modifica para ello dos variables globales que almacenan los números pares e impares encontrados.

   6. *procedure arrayIguales (Vector1 : T_Table, Vector2 : T_Table)*

   -- Comprueba si dos vectores de enteros son iguales o no. Deposita el resultado en una variable global.

7. *procedure marcaPares (Vector : T_Table)*

-- Comprueba los números pares situados en un vector y modifica dicho valor par por un 2 en la posición donde se encontró.

3. Tabla que muestra qué características de SPARK se han utilizado para verificar formalmente cada uno de los procedimientos y funciones.

El número de cada columna se corresponde con el número asignado a los métodos en el apartado (2). Si necesitas más columnas añade más columnas a las tablas de esta plantilla.

En cada casilla solamente hay que marcar con 'X' (en el centro de la casilla) si la verificación de este método utiliza la característica de SPARK indicada en el margen derecho de esa fila ***y su valor no es NULL ni True****.*

El uso de Contract_Cases es opcional.

Para que tu práctica puntúe debes tener al menos una X en cada una de las filas (excepto en la fila de Contract_Cases, que es opcional)

	1	2	3	4	5	6	7
Global	-	-	X	-	X	X	-
Depends	X	X	X	X	X	X	X
Pre	X	X	X	X	X	X	X
Post	X	X	X	X	X	X	X
Contract_Cases	-	-	-	-	-	-	-
'Result	X	X	-	-	-	-	-
'Old	-	-	X	X	-	-	X

	1	2	3	4	5	6	7
for all	-	X	X	X	X	X	X
for some	-	X	X	X	X	X	-

	1	2	3	4	5	6	7
Loop_Variant	-	-	X	-	-	-	X
Loop_Invariant	-	X	X	X	X	X	X
'Loop_Entry	-	X	X	X	X	X	X

Número de tests unitarios hechos para comprobar cada procedimiento y función. De nuevo, el número de cada columna se corresponde con el número asignado a los métodos en el apartado (2). Si necesitas más columnas añade más columnas a esta tabla.

El número mínimo de tests por cada ejercicio es 3.

	1	2	3	4	5	6	7
Número de tests	6	9	9	10	10	7	9

4. Cabecera completa (con su contrato) y cuerpo de cada uno de los procedimientos y funciones verificados formalmente (incluyendo el comentario que describe su comportamiento, y manteniendo la numeración del apartado (2)).

   *En cada procedimiento/función debes especificar si debe verificarse con un **nivel de verificación mínimo**. En caso de no especificar nada se PRESUPONE que se verifica con nivel 0.*

   Utiliza fuente de letra pequeña para facilitar la lectura del código SPARK. Respeta también el sangrado del código.

1. Es_Par

Cabecera

```
-- Verificado en niveles 0-4
package Pkg_esPar with SPARK_Mode is

   -- Retorna verdadero o falso si el módulo de la división por dos es 0 o no
   -- (si es par o impar)
   function Es_Par
     (Number : Integer) return boolean
   with
   --      Global  => ...
     Depends => (Es_Par'Result => Number),
     Pre     => Number > Integer'First,
     Post => ((Es_Par'Result
               and then
                 (Number rem 2 = 0))
              or else
                (not Es_Par'Result
                 and then
                   (Number rem 2 /= 0)));

end Pkg_esPar;
```

Cuerpo

```
package body Pkg_esPar with SPARK_Mode is

   function Es_Par
     (Number : Integer) return boolean is
   begin
      if(Number mod 2 = 0) then
         return true;
      else
         return false;
      end if;
   end Es_Par;

end Pkg_esPar;
```

2. Es_Primo

Cabecera

```
-- Verificado en niveles 0-4
package Pkg_esPrimo with SPARK_Mode is

   -- Retorna verdadero o falso si alguno de los números que hay entre 2 y A-1
   -- divide a A. Si A es divisible por algún n entre 2 y A-1, no es primo

   function Es_Primo
     (Number : Integer) return Boolean
   with
   --   Global  => null,
     Depends => (Es_Primo'Result=> Number),
     Pre     => Number > 0,
   -- Si es primo entonces para todo k entre 2..N-1 el resto es /= 0
   -- Si no es primo, es que existe al menos un N en 2..N-1 que lo divide
     Post => ((if Es_Primo'Result then
                  (for all k in 2 .. Number-1 => Number rem k /= 0)
               or else
                 (for some k in 2 .. Number-1 => Number rem k = 0))
             );

end Pkg_esPrimo;
```

Cuerpo

```
package body Pkg_esPrimo with SPARK_Mode is
   function Es_Primo
     (Number : Integer) return Boolean is
      A: Integer;
   begin
      A := Number;
      for i in 2 .. A-1 loop
         if (A mod i = 0) then
            return false;
         end if;

         -- ESTRUCTURAL: Se comprueba que la i nunca pueda superar el valor del número
         pragma Loop_Invariant(i <= A);

         -- LOCAL: Se comprueba que si el número es primo la función devuelve True y en caso
         -- contrario false
         pragma Loop_Invariant ((for all k in 2 .. A-1 =>
                                   (if A mod k/=0 then
                                       True)));
      end loop;

      return true;
   end Es_Primo;

end Pkg_esPrimo;
```

3. cuentaPrimos

Cabecera

```
-- Verificado en niveles 0-4
with Pkg_esPrimo;        use Pkg_esPrimo;
package Pkg_cuentaPrimos with SPARK_Mode is
   NPrimos : Integer;
   type T_Table is array (Natural range <>) of Integer;

   procedure cuentaPrimos
     (Vector : T_Table)
     with
     Global  => (In_Out => NPrimos),
     Depends => (NPrimos => (Vector),
                 null => (NPrimos)),
     Pre  =>(Vector'Length > 0
             and then Vector'First = 1
             and then Vector'Last < Natural'Last
             and then (for all k in Vector'Range => Vector(k) > 0)
             and then (NPrimos = 0)),
     Post => ((if NPrimos > Nprimos'Old
                  then
                (for some k in Vector'Range => Es_Primo(Vector(k)))
              and then NPrimos <= Vector'Length)
              or else
                (if NPrimos = NPrimos'Old then
                   (for all k in Vector'Range => not Es_Primo(Vector(k))))
             );
end Pkg_cuentaPrimos;
```

Cuerpo

```
package body Pkg_cuentaPrimos with SPARK_Mode is
   procedure cuentaPrimos
     (Vector : T_Table) is
      i : Natural := Vector'First;
   begin
      NPrimos := 0;
      while (i <= Vector'Last) loop
         if(Es_Primo(Vector(i))) then
            NPrimos := NPrimos+1;
         end if;

         -- ESTRUCTURAL: i se incrementa a cada vuelta
         pragma Loop_Variant(Increases => i);

         -- ESTRUCTURAL / LOCAL: Se comprueba que i está en el rango.
         pragma Loop_Invariant(i in Vector'Range);

         -- LOCAL: Se verifica que si el valor de Nprimos cambia respecto a su
         -- entrada en la iteracion N, es que hay algún número que es primo. Se
         -- justifica ademñas que NPrimos siempre es <= que el recorrido actual
         pragma Loop_Invariant(if NPrimos > NPrimos'Loop_Entry then
                                  (for some k in Vector'First .. i =>
                                     Es_Primo(Vector(k))) and then NPrimos <= i);
         i := i+1;
      end Loop;
   end cuentaPrimos;
end Pkg_cuentaPrimos;
```

4. primerPrimo

Cabecera

```
-- Verificado en niveles 0-4
with Pkg_esPrimo;  use Pkg_esPrimo;
package Pkg_primerPrimo with SPARK_Mode is

   -- Inserta en la variable resultado, el primer primo existente en el vector.
   type T_Table is array (Natural range <>) of Integer;

   procedure primerPrimo (Vector : T_Table; Resultado :in out Natural)

   with
       Global  => null,
       Depends => (Resultado => (Vector),
                   null => (Resultado)),
       Pre     => ((Resultado = 0)
                   and then
                     (Vector'Length > 0 and Vector'Last < Natural'Last)
                   and then
                     (for all k in Vector'Range => Vector(k) > 0)),
       Post    => ((Resultado > Resultado'Old and then
                     (for some k in Vector'Range => Es_Primo(Vector(k))
                      and then
                        Resultado = Vector(k)))
                 or else
                   (Resultado = 0 and then
                        (for all k in Vector'Range => not Es_Primo(Vector(k)))
                     ));

end Pkg_primerPrimo;
```

Cuerpo

```
package body Pkg_primerPrimo with SPARK_Mode is
   procedure primerPrimo(Vector : T_Table; Resultado :in out Natural) is
   begin
      Resultado := 0;
      for i in Vector'Range loop
         if(Es_Primo(Vector(i))) then
            Resultado := Vector(i);
            exit;
         end if;

         -- LOCAL: Se verifica que R=>S ya que:
         -- La variable resultado, al depender del vector, se mantendrá similar
         -- o mayor que su mayor índice (en su rango)
         pragma Loop_Invariant((for all j in Vector'First .. i =>
                                   (if Es_Primo(Vector(j)) then
                                   Resultado = Vector(j))));
      end loop;
   end primerPrimo;

end Pkg_primerPrimo;
```

5. histoParImpar

Cabecera

```
with Pkg_esPar; use Pkg_esPar;
package pkg_HistoParImpar with SPARK_Mode is

   -- Calcula los números pares e impares que contiene un vector.
   -- Modifica para ello dos variables globales que almacenan los números
   -- pares e impares encontrados.
   Pares: Integer;
   Impares: Integer;
   type T_Table is array (Positive range <>) of Integer;

   procedure histoParImpar (Vector : T_Table)
     with
       Global  => (Output => (Pares,Impares)),
       Depends => (Pares => (Vector),
                   Impares => (Vector)),
       Pre     => (Vector'Length > 0
                   and then Vector'Last < Positive'Last
                   and then (for all k in Vector'Range =>
                                Vector(k) > Integer'First)),
       Post    => (Pares + Impares = Vector'Length
                   and then
                     ((Pares > 0 and then
                          (for some j in Vector'Range => Es_Par(Vector(j))))
                      or else
                        (Pares = 0 and then
                             (for all j in Vector'Range =>
                                  not Es_Par(Vector(j))))))
                  );

end pkg_HistoParImpar;
```

Cuerpo

```
package body pkg_HistoParImpar with SPARK_Mode is
   procedure histoParImpar (Vector : T_Table) is
      nPares : Natural := 0;
   begin
      for i in Vector'Range loop
         if(Es_Par(Vector(i))) then
            nPares := nPares+1;
         end if;

         -- LOCAL: Si hay Pares registrados, es que al menos uno de los
         -- anteriores es par y, aún así Npares <= i.
         pragma Loop_Invariant(if nPares > nPares'Loop_Entry then
                                  (for some j in Vector'First .. i =>
                                     Es_Par(Vector(j)))
                               and then nPares <= i);

         -- LOCAL: Si no hay Pares registrados es que todos los anteriores son
         -- impares y por tanto Npares <= i.
         pragma Loop_Invariant(if nPares = nPares'Loop_Entry then
                                  (for all j in Vector'First .. i =>
                                     not Es_Par(Vector(j)))
                               and then nPares <= i);
      end loop;
      Pares := nPares;
      Impares := Vector'Length - nPares;
   end histoParImpar;
 end pkg_HistoParImpar;
```

6. arrayIguales

Cabecera

```
-- Verificado en niveles 0-4
package Pkg_arrayIguales with SPARK_Mode is

   -- Comprueba si dos vectores de enteros son iguales o no.
   -- Deposita el resultado en una variable global.
   type T_Table is array (Natural range <>) of Integer;
   resultado : Boolean;

   procedure arrayIguales (Vector1 : in T_Table; Vector2 : in T_table)
     with
     Global => (Output => resultado),
     Depends => (resultado => (Vector1, Vector2)),
     Pre => ((Vector1'Length = Vector2'Length)
             and then
               (Vector1'Length > 0)
             and then
               (Vector1'First = Vector2'First)),
     Post => ((if not resultado then
                  (for some i in Vector1'Range =>
                     Vector1(i) /= Vector2(i))
              or else
                (for all j in Vector1'Range =>
                   Vector1(j) = Vector2(j))));

end Pkg_arrayIguales;
```

Cuerpo

```
package body Pkg_arrayIguales with SPARK_Mode is

   procedure arrayIguales(Vector1 : in T_Table; Vector2 : in
                                T_table) is
   begin
      resultado := True;

      for i in Vector1'First .. Vector1'Last loop
         if Vector1(i) /= Vector2(i) then
            resultado := false;
            exit;
         end if;

         -- LOCAL: se verifica la postcondición a nivel local
         pragma Loop_Invariant(if resultado /= resultado'Loop_Entry then
                                  (for some k in Vector1'First .. i =>
                                     Vector1(k) /= Vector2(k))
                               else
                                  (for all k in Vector1'First .. i =>
                                     Vector1(k) = Vector2(k)));
      end loop;

   end arrayIguales;

end Pkg_arrayIguales;
```

7. marcaPares

Cabecera

```
-- Verificado en niveles 0-4
with Pkg_esPar;        use Pkg_esPar;
package Pkg_marcaPares with SPARK_Mode is
   -- Comprueba los números pares situados en un vector y modifica dicho valor
   -- par por un 2 en la posición donde se encontró.

   type T_Table is array (Natural range <>) of Integer;
   procedure marcaPares
     (Vector : in out T_Table)
     with
     Global  => null,
     Depends => (Vector => Vector),
     Pre  => (Vector'Length > 0
              and then Vector'Last < Natural'Last
              and then (for all k in Vector'Range =>
                           Vector(k) > Integer'First)),
     Post => (for all k in Vector'Range =>
                (if Vector(k) = 2 then
                     Es_Par(Vector'Old(k))
                   else
                     Vector(k)=Vector'Old(k)));
end Pkg_marcaPares;
```

Cuerpo

```
package body Pkg_marcaPares with SPARK_Mode is

   procedure marcaPares
     (Vector : in out T_Table) is
      i : Natural := Vector'First;
   begin
      while (i<=Vector'Last) loop
         if Es_Par(Vector(i)) then
            Vector(i) := 2;
         end if;

         -- LOCAL: Se comprueba que el valor de vector cambia si hemos pasado
         -- por él y si es necesario
         pragma Loop_Invariant(for all k in Vector'First .. i =>
                                  (if Es_Par(Vector'Loop_Entry(k)) then
                                      Vector(k) = 2
                                   else
                                      Vector(k)=Vector'Loop_Entry(k)
                                  ));
         -- LOCAL: se comprueba si en el lado restante del vector sigue igual.
         pragma Loop_Invariant(for all k in i+1..Vector'Last =>
                                  Vector(k)=Vector'Loop_Entry(k));

         -- ESTRUCTURAL: i se incrementa a cada vuelta
         pragma Loop_Variant(Increases => i);

         -- ESTRUCTURAL: Se comprueba que i está en el rango.
         pragma Loop_Invariant(i in Vector'Range);
         i := i+1;
      end loop;
   end marcaPares;

end Pkg_marcaPares;
```

Entrega adicional

5. (Video) Cada equipo debes subir un video de un máximo de 10 minutos explicando la verificación formal de su código: precondiciones, postcondiciones, invariantes.