Práctico 7 - Subprogramas (Funciones y Procedimientos)

Programación 1
InCo - Facultad de Ingeniería, Udelar

1. (a) Explique la diferencia entre un parámetro pasado por valor y uno pasado por referencia.

> Consultar los materiales: libro, notas teóricas, clases *openfing*.

(b) Identifique cuáles parámetros están pasados por valor y cuáles están pasados por referencia en los siguientes encabezados de procedimientos y funciones:

I) `function areaRectangulo(largo, ancho : real) : real;`
II) `procedure leerTriangulo(var base, altura : real);`
III) `procedure perimetroAreaCuadrado(lado : real; var perimetro, area : real);`

> **areaRectangulo**
>
> - Por valor: todos
> - Por referencia: ninguno
>
> **leerTriangulo**
>
> - Por valor: ninguno
> - Por referencia: todos
>
> **perimetroAreaCuadrado**
>
> - Por valor: `lado`.
> - Por referencia: `perimetro` y `area`.

(c) Explique la diferencia entre un parámetro de entrada y un parámetro de salida. ¿Cómo se relacionan con los dos tipos de pasaje de parámetros vistos?

> Un parámetro de entrada es aquel que debe necesariamente tener un valor asignado cuando se lo invoca. Ese valor es usado como una entrada del algoritmo.
> Un parámetro de salida debe tener un valor asignado cuando termina el subprograma. Ese valor será utilizado por el programa que invoca al subprograma luego de la ejecución de este. Ese valor es parte de la salida que produce el subprograma. En estos casos, no se requiere que el parámetro reciba un valor inicial cuando es invocado el subprograma.
> Un parámetro puede ser a la vez de entrada y de salida.
> Un parametro de salida siempre debe ser un parámetro por referencia.
> Una parámetro de entrada es generalmente un parámetro por valor salvo en el caso que sea de entrada y salida a la vez.

(d) Indique cuáles de las siguientes invocaciones son correctas. Suponga que las variables `x`,`y`,`z` son reales y que las variables `i`,`j` son enteras. Explique.

√ `leerTriangulo(x, y)`
☐ `leerTriangulo(i, j)`
☐ `leerTriangulo(2.2, 3.5)`
√ `z := areaRectangulo(3, 5)`
√ `areaRectangulo(x, y)`
√ `perimetroAreaCuadrado(2.5, y, z)`
☐ `perimetroAreaCuadrado(x, 3.2, 4.7)`

2. Dado el siguiente fragmento de programa:

```
program Pr7Ej2
...
procedure inicio(tiempo, espacio: real; var dia: real; signo: char);
begin {cuerpo del procedimiento}
   ...
end;
...
begin {programa principal}
   ...
   inicio(3.5, 6.0, hora, 'Z');
   ...
end.
```

(a) Explique la diferencia entre un parámetro nominal (o formal) y un parámetro efectivo (o verdadero). Indique cuáles son los parámetros nominales en el programa.

> Consulte las definiciones de parámetro nominal y parámetro efectivo en el material teórico del curso.
> En este ejemplo, son parámetros formales todos los que aparecen en el cabezal
> `procedure inicio(tiempo, espacio: real; var dia: real; signo: char);`
> Los parámetros efectivos son los que aparecen en la invocación:
> `inicio(3.5, 6.0, hora, 'Z');`

(b) ¿Es necesario que los parámetros efectivos sean siempre variables? Explique.

> No, los parámetros efectivos que corresponden a un parámetro por valor pueden ser cualquier expresión del tipo apropiado.

(c) En caso de que los parámetros efectivos sean variables, ¿es necesario que tengan el mismo nombre que sus correspondientes parámetros nominales? Explique.

> No, no es necesario.

3. Dado el siguiente procedimiento:

```
procedure prueba(x: real; y: integer; var z: real);
```

Indique cuáles de las siguientes invocaciones son correctas. Suponga que en el contexto en el que se hacen las invocaciones, la variable `x` es de tipo `real` y la variable `n` es de tipo `integer`.

- ☐ `prueba(1, 2.0, x)`
- √ `prueba(n, 3, x)`
- ☐ `prueba(n, 3, 2.0)`
- ☐ `prueba(1, 3, n)`
- √ `prueba(5*n, round(7.3), x)`
- √ `prueba(x, 3, x);`
- ☐ `prueba(Prueba(5, 33.8, x), 92, x)`

4. Dado el siguiente programa:

```
program Pr7Ej4;
var tum, num, temp : integer;

procedure proc(a, b : integer; var c : integer);
var aux : integer;
begin
    aux := a * b;
    aux := aux + 1;
    c := aux + a;
    writeLn(a, b, c, aux)
```

```
end;

{ Programa principal }
begin
    tum := 1;
    num := 2;
    proc(tum, num, temp);
    writeLn(temp);
    tum := 0;
    num := 1;
    proc(tum, num, temp);
    writeLn(temp)
end.
```

(a) Indique cuáles son los parámetros formales del procedimiento *proc*. Indique cuáles de ellos son parámetros por valor y cuáles son parámetros por referencia.

> Los parámetros formales de proc son: `a` , `b` y `c`.
> Por valor: `a` y `b`. Por referencia: `c`.

(b) Indique cuáles son los parámetros efectivos que aparecen en el programa.

> Son `tum`, `num`, `temp`.

(c) Indique qué se exhibirá al ejecutar el programa.

> Compile y ejecute el programa para conocer la salida correcta.

5. (a) Escriba un procedimiento llamado *corrimiento* con tres parámetros enteros: `a`, `b` y `c`. El procedimiento debe hacer un corrimiento a la derecha de los valores de los parámetros de manera que, después de la invocación, el valor que originalmente estaba en `a` quede en `b`, el que estaba en `b` quede en `c` y el que estaba en `c` quede en `a`.

(b) Escriba un programa principal que lea tres valores y exhiba el resultado de invocar al procedimiento *corrimiento*.

Ejemplo:

```
Ingrese tres números: 4 1 7
El corrimiento es: 7 4 1
```

```
program Pr7Ej5;
var
   uno, dos, tres : integer;

procedure corrimiento(var a, b, c : integer);
var foo : integer;
begin
   foo := c;
   c   := b;
   b   := a;
   a   := foo;
end;

begin
   write('Ingrese tres números: ');
   readln(uno, dos, tres);

   corrimiento(uno,dos,tres);

   writeln('El corrimiento es :', uno, dos, tres)

end.
```

6. Sea el siguiente encabezado del procedimiento `raices` que calcula las raíces reales de un polinomio de segundo grado, de la forma $ax^2 + bx + c$.

   ```
   procedure raices(a,b,c : integer; cant : integer; raiz1, raiz2 : real)
   ```

   donde:

   - `a`, `b` y `c` son los coeficientes del polinomio.
   - `cant` indica la cantidad de raices reales diferentes(0, 1 ó 2).
   - `raiz1` y `raiz2` son las raíces reales del polinomio. Si el polínomio tiene **una** raíz real, se almacena en `raiz1`. Si el polinomio tiene **dos** raíces reales diferentes, se almacenan en `raiz1` y `raiz2`.

Ejemplos:
Entrada: a = 2, b = -3, c = -2 Resultado: cant = 2, raiz1 = 2, raiz2 = -0.5
Entrada: a = 1, b = -6, c = 9 Resultado: cant = 1, raiz1 = 3, raiz2 = ?
Entrada: a = 5, b = 4, c = 1 Resultado: cant = 0, raiz1 = ?, raiz2 = ?

   (a) Determine para cada parámetro del encabezado del procedimiento si debe ser pasado por *valor* o *referencia*.

   Ver solución de parte b)

   Modifique el encabezado anterior según su respuesta.

   (b) Escriba el cuerpo del procedimiento `raices` (análogo a los casos de raíces reales del ejercicio 12, práctico 3).

```
procedure raices(a,b,c : integer; var cant : integer; var raiz1, raiz2 : real);
var
   disc, raizdisc : real;
begin
   disc := sqr(b)-4*a*c;

   if (disc = 0) then {única solución}
   begin
      cant:= 1;
      raiz1 := -b/(2*a);
   end
   else if (disc > 0) then {dos raíces reales distintas}
   begin
      cant:= 2;
      raizdisc := sqrt(disc);
      raiz1 := (-b + raizdisc)/(2*a);
      raiz2 := (-b - raizdisc)/(2*a);
   end
   else {Raices imaginarias}
   begin
      cant:= 0;
   end
end;
```

   (c) Escriba un programa principal que lea los tres coeficientes e invoque al procedimiento para calcular las raíces del polinomio. El programa exhibe las raíces devueltas por el procedimiento.

   Solución a cargo del estudiante.

7. (a) Escriba una función llamada *distancia* que tenga cuatro parámetros de entrada (reales) llamados `x1`, `y1`, `x2`, `y2`, que representan las coordenadas en el plano de los puntos $(x1, y1)$ y $(x2, y2)$. La función debe calcular y retornar la distancia entre ambos puntos.

```
function distancia(x1,y1,x2,y2 : real) : real;
begin
   distancia:= sqrt(sqr(x1 - x2) + sqr(y1 - y2))
end;
```

(b) Escriba un programa principal que lea las coordenadas de dos puntos y exhiba la distancia entre ambos. El programa debe invocar a la función *distancia*.

Ejemplo:

```
Ingrese las coordenadas del primer punto: 1 2
Ingrese las coordenadas del segundo punto: 5 7
La distancia entre los puntos es: 6.40
```

Solución a cargo del estudiante.

8. (a) Escriba una función llamada *esPrimo* que, dado un entero mayor o igual que 0, devuelva `true` si es primo y `false` en caso contrario. En el ejercicio 7 del práctico 5 se escribió un programa principal que resolvía el mismo algoritmo.

Ver solución de la siguiente parte.

(b) Escriba un programa principal que resuelva el ejercicio 8 del práctico 6 invocando a la función *esPrimo*. El programa lee dos enteros positivos `m` y `n` y exhibe todos los números primos en el rango. En caso de que `m` sea mayor que `n`, no se exhibirá ninguno.

```
program Pr6Ej8;
var
   n, m, num : integer;

function esPrimo(numero : integer) : boolean;

   function tieneDivisores(numero : integer) : boolean;
   var
      fin, divisor : integer;

   begin
      fin := trunc(sqrt(numero));
      divisor:= 2;
      { busco divisores desde 2 hasta la raíz del numero }
      while (divisor <= fin) and (numero mod divisor <> 0) do
         divisor:= divisor + 1;

      tieneDivisores := divisor <= fin
   end;

begin
   esPrimo :=  (numero > 1) and not tieneDivisores(numero)
end;

begin

   write('Ingrese dos numeros enteros positivos: ');
   readln(m, n);

   write('Los numeros primos entre ', m, ' y ', n, ' son: ');
   for num := m to n do
      if esPrimo(num) then
         write(num, ' ')
end.
```

9. (a) Escriba una función llamada *multiplo* que, dados dos enteros positivos `m` y `n`, devuelva `true` si `m` es múltiplo de `n` o si `n` es múltiplo de `m`. En otro caso contrario, devuelve `false`.

(b) Escriba un programa principal, que lea, en una misma línea, parejas de enteros positivos e invoque a la función *multiplo* para indicar si uno de los dos números es multiplo del otro. La secuencia de números terminar con el valor `-1`.

Ejemplo:

```
4 7
No
4 8
Si
16 8
Si
-1
```

```
program Pr7Ej9;
const
   CENTINELA = -1;
var
   a, b : integer;

function multiplo(m,n : integer) : boolean;
begin
   multiplo:= (n mod m = 0) or (m mod n = 0)
end;

begin
   read(a);
   while a <> CENTINELA do
   begin
      readln(b);
      if multiplo(a,b) then
         writeln('Si')
      else
         writeln('No');
      read(a)
   end;
end.
```

10. (a) Escriba el siguiente procedimiento llamado *multiplicidadFactor* que, dado un entero positivo `numero` y un entero positivo `factor`, devuelve en `multiplicidad` la cantidad máxima de veces que `factor` divide a `n` y en `residuo` el resto.

```
procedure multiplicidadFactor(numero, factor : integer;
                              var multiplicidad, residuo : integer);
```

Ver solución de la siguiente parte.

Ejemplos:
Entrada: numero = 39, factor = 2 Salida: multiplicidad = 0, residuo = 39
Entrada: numero = 39, factor = 3 Salida: multiplicidad = 1, residuo = 13
Entrada: numero = 42, factor = 2 Salida: multiplicidad = 1, residuo = 21
Entrada: numero = 12, factor = 2 Salida: multiplicidad = 2, residuo = 3

(b) Escriba un programa principal que lea una secuencia de enteros positivos mayores a 1 y exhiba la descomposición en factores primos de cada número. La secuencia de números termina con el valor `-1`. El programa debe invocar a la función *multiplicidadFactor*.

Ejemplo:

```
39 17 1517 42 12 18 -1
39 = 3 * 13
17 = 17
1517 = 37 * 41
42 = 2 * 3 * 7
12 = 2 * 2 * 3
18 = 2 * 3 * 3
```

```
program Pr7Ej10;
const
    CENTINELA = -1;
var
   numero, factor, residuo, nuevoResiduo, multiplicidad, i : integer;

function divide(a,b : integer) : boolean;
begin
   divide:= b mod a = 0
end;

procedure multiplicidadFactor(numero, factor : integer;
                              var multiplicidad, residuo : integer);
begin
   multiplicidad:= 0;
   residuo:= numero;
   while divide(factor, residuo) do
   begin
      multiplicidad:= multiplicidad + 1;
      residuo:= residuo div factor
   end;
end;

begin
   read(numero);
   while numero <> CENTINELA do
   begin
      write(numero,' = ');

      residuo := numero;
      factor:= 2;

      { busco primer factor }
      while not divide(factor,numero) do
         factor:= factor + 1;
      { el primer factor va sin '*'
        por eso se procesa primero    }
      write(factor:4);
      residuo := numero div factor;

      { busco siguientes factores }
      while residuo <> 1 do
      begin
         multiplicidadFactor(residuo,factor, multiplicidad, nuevoResiduo);
         residuo:= nuevoResiduo;
         { repito el factor de acuerdo con su multiplicidad}
         for i:= 1 to multiplicidad do
            write(' * ', factor:4);
         factor:= factor + 1
      end;
      writeln;
     read(numero)
   end;
end.
```