240205 Programación

Exámenes de Prácticas de laboratorio

José Ramón González de Mendívil
mendivil@unavarra.es
Mikel Díaz
mikel.diaz@unavarra.es
Iñigo Ezcurdia
inigofermin.ezcurdia@unavarra.es

Departamento de Estadística, Informática y Matemáticas
Universidad Pública de Navarra
Campus de Arrosadía
31006 Pamplona

8 de diciembre de 2021

En este documento se presentan algunos ejemplos de exámenes de prácticas de la asignatura de Programación. Los estudiantes deben realizar estos ejercicios para poder abordar el examen final de prácticas sin dificultad. Se asume que los estudiantes ya han realizado las sesiones prácticas de la asignatura y por tanto, en algunos de los ejercicios se pide escribir algunos algoritmos, en forma de acciones o funciones, que o bien, ya han sido desarrollados, o bien son lo suficientemente asequibles al conocimiento adquirido por los estudiantes a lo largo del semestre. Los estudiantes deben saber realizar recorridos y bÃºsquedas simples en tablas usando los métodos explicados en la asignatura. También deben saber analizar y comprender correctamente las especificaciones de los algoritmos asi como, identificar los parámetros de dichos algoritmos para convertirlos correctamente a acciones o funciones en los programas resultantes de los ejercicios. Los programas en C resultantes deben seguir las normas de estilo propuestas en la asignatura. Finalmente, los ejercicios están pensados para que se puedan desarrollar en menos de hora y media, incluyendo la realización de algunas pruebas de ejecución para comprobar su funcionalidad.

Nota: Si encuentra algún error en este documento, por favor, escriba a mendivil@unavarra.es indicando dicho error.

0.1. Ejercicio: Ordenación por inserción

Se pide codificar en C un programa `ordbus.c` que lea una secuencia de valores enteros introducidos por el usuario desde teclado, compruebe si la secuencia de números está ordenada y, en caso negativo, ordene dichos valores en orden ascendente. Al final, el programa deberá mostrar en una sola línea el contenido de la secuencia de valores ordenada, junto con un mensaje que indique si los valores originales estaban ya ordenados o no. Para ello, el programa deberá realizar los siguientes pasos:

1) Pide al usuario $N + 1$ valores enteros que forman la secuencia de entrada. Estos valores se guardarán en un tabla f de tipo $tabla0N$, siendo `tabla0N:: tabla [0..N] de enteros`, y N, una constante mayor o igual a cero.

2) Comprueba si la tabla f está ordenada. Para ello, el programador deberá diseñar y codificar en C una función de nombre `esta_ordenada` especificada como sigue:

> **funcion** esta_ordenada(t: tabla0N) **dev** b: *booleano*
> $\{P : cierto\}$
>
> $\{Q : b \equiv (\forall k : 0 \leq k < N : t[k] \leq t[k+1])\}$
> **ffuncion**

3) Si la secuencia no está ordenada, el programa debe ordenar los valores mediante una acción `insercion`. Esta acción ordena, en general, los valores almacenados en una tabla t (de tipo $tabla0N$) por el método de inserción. La cabecera de esta acción debe ajustarse a la siguiente especificación

> **accion** insercion(**e/s** t: tabla0N)

El algoritmo para la acción se presenta al final del texto.

4) Finalmente, el programa debe indicar si los valores de la secuencia inicial estaban ordenados o no, con el mensaje 'La secuencia está ordenada' o 'La secuencia NO está ordenada' y, en cualquier caso, debe mostrar en una sóla línea los valores de la secuencia en orden ascendente.

El programa deberá seguir las normas de estilo dadas en clase y deberá contener la cabecera donde se identifique autor, fecha de realización y descripción del programa. Esta información se mostrará la primera vez que se ejecute el programa. Además, el programa permitirá su ejecución tantas veces como el usuario desee. **Se debe respetar también el nombre de las variables especificadas tanto en el enunciado como en los algoritmos dados**.

Para probar el funcionamiento del programa se considerará $N = 6$. La salida por pantalla de la ejecución del programa deberá ajustarse, aparte de lo especificado por las normas de estilo, al ejemplo que se muestra a continuación. Tenga en cuenta que los valores que se piden al usuario deben introducirse en la misma línea en la que se solicitan.

```
(...)
Introduce 7 valores enteros,
Elemento 1 de la secuencia: 1
Elemento 2 de la secuencia: 2
Elemento 3 de la secuencia: -3
Elemento 4 de la secuencia: 4
Elemento 5 de la secuencia: -6
Elemento 6 de la secuencia: 5
Elemento 7 de la secuencia: 3
La secuencia NO esta ordenada
Los valores en orden ascendente son: -6 -3 1 2 3 4 5
(...)
```

EL algoritmo para la acción es el siguiente:

algoritmo insercion;
const $N \geq 0$ **fcost**

t: tabla[0..N] de *entero*;
$\{Pre : t = T\}$
var
    i, p: *entero*;
fvar
    i:= 0;
    $\{Inv : t \in \text{Perm}(T) \wedge ord(t[0..i]) \wedge 0 \leq i \leq N\}$
    mientras i $\neq$ N **hacer**
        p:= i + 1;
        mientras $1 < p \wedge t[p-1] > t[p]$ **hacer**
            swap($t[p-1]$, $t[p]$); (* intercambio *)
            p:= p - 1
        fmientras;
        si $t[p-1] \leq t[p] \longrightarrow$ **continuar**
          [] $t[p-1] > t[p] \longrightarrow$ **swap**($t[0]$, $t[1]$)
        fsi;
        i:= i + 1
    fmientras
$\{Post : t \in \text{Perm}(T) \wedge ord(t[0..N])\}$
falgoritmo

0.2. Ejercicio: Suma de una secuencia de primos

Se pide codificar en C un programa `primesum.c` que lea una secuencia de n valores enteros desde teclado y que realice la suma de aquellos valores que son primos en dicha secuencia. Al final hay que mostar cuáles eran los números primos en la secuencia y la suma total obtenida. Para ello, el programa deberá realizar los siguientes pasos:

1) Pedir al usuario la longitud n de la secuencia (entre 1 y $L + 1$). A continuación, se le pedirá al usuario uno a uno los elementos (valores enteros mayores que uno) de dicha secuencia. Los valores de la secuencia se almacenarán en una variable a de tipo `vector`, cuya declaración en pseudocódigo es:

tipo vector= tabla[0..L] de enteros ftipo, siendo L una constante $L \geq 0$.

2) A partir de esta primera tabla a, calcular una segunda tabla b del mismo tipo, junto con un tamaño m. La nueva tabla sólo contiene los números primos que hay en la primera tabla a. Para ello, será necesario escribir previamente una acción `primos` que a partir de una tabla t de tipo `vector`, y un tamaño n dados como entrada, calcule como salida otra tabla s de tipo `vector` y una posicíon m, de manera que entre las posiciones de $0..m-1$ de s se encuentran los primos de t que hay entre las posiciones $0..n-1$. Para codificar dicha acción, se utilizará el algoritmo que se proporciona al final del enunciado. La cabecera de la acción tendrá que ajustarse a la siguiente especificación:

accion primos(e/ t: vector; e/ n: *entero*; sal/ s: vector; sal/ m: *entero*);

3) Una vez obtenida la nueva tabla de primos, deberá calcular la suma de todos sus elementos. Para ello, será necesario escribir una función `sumat`. La especificación de esta función es la siguiente:

> **funcion** sumat(s: vector; m: *entero*) **dev** r: *entero*
> $\{P : 0 \leq m \leq L + 1\}$
> $\{Q : r = (\sum k: 0 \leq k < m: s[k])\}$
> **ffuncion**

4) Finalmente, tras realizar los pasos anteriores, el programa mostrará en una sóla línea cuáles eran los números primos que había en la tabla original a (utilizar para ello el contenido de la tabla b) y después mostrará cuál es el resultado de la suma de estos números primos.

El programa deberá seguir las normas de estilo dadas en clase y deberá contener la cabecera donde se identifique autor, fecha de realización y función del programa. Esta información se mostrará cada vez que se ejecute el programa. Además, el programa permitirá su ejecución tantas veces como el usuario desee. Se debe respetar también el nombre de las variables especificadas tanto en el enunciado como en los algoritmos dados. Para probar el funcionamiento del programa se considerará $L = 9$. La salida por pantalla de la ejecución del programa deberá ajustarse, además de lo especificado por las normas de estilo, al ejemplo que se muestra a continuación.

```
(...)
```

```
Introduce la longitud de la secuencia (1..10): 5
Introduce 5 valores enteros mayores que 1:
Elemento 1 de la secuencia: 11
Elemento 2 de la secuencia: 25
Elemento 3 de la secuencia: 37
Elemento 4 de la secuencia: 51
Elemento 5 de la secuencia: 19
Los numeros primos son: 11 37 19
La suma de los numeros primos es: 67
(...)
```

algoritmo primos;
const L ≥ 0 **fconst**
tipo vector = tabla[0..L] de entero **ftipo**

funcion es_primo(z: *entero*) **dev** b: *booleano*
$\{P : z > 1\}$
var
j: *entero*;
fvar
j := 2;
$\{Inv_2 : (\forall l : 1 < l < j : z \ mod \ l \neq 0) \wedge 2 \leq j \leq z\}$
mientras z mod j $\neq 0$ **hacer** j := j + 1 **fmientras**;
b := (j = z);
$\{Q : b = primo(z)\}$ (* $primo(z) \equiv \forall l : 1 < l < z : z \ mod \ l \neq 0$ *)
dev(b)
ffuncion

t, s: vector;
n, m: *entero*;
$\{Pre : t = T \ \wedge \ n = N \wedge \ 1 \leq N \leq L + 1 \ \wedge \ (\forall k{:}\, 0 \leq k < n{:}\, t[k] > 1)\}$
var
i: *entero*;
b: *boleano*;
fvar
m := 0;
i := 1;
$\{Inv_1 : t = T \wedge n = N \wedge \forall k : 0 \leq k < i : primo(t[k]) \Rightarrow s[\#l : 0 \leq l < k : primo(t[l])] = t[k]$
$\wedge m = \#l : 0 \leq l < i : primo(t[l]) \wedge \ 0 \leq i \leq n\}$
mientras i $\leq$ n **hacer**
b := es_primo(t[i]);
si $\neg$b $\rightarrow$ **continuar**
[] b $\rightarrow$ s[m] := t[i]; m := m + 1;
fsi;
i := i + 1
fmientras
$\{Post : t = T \wedge n = N \wedge \forall k : 0 \leq k < n : primo(t[k]) \Rightarrow s[\#l : 1 \leq l < k : primo(t[l])] = t[k]$
$\wedge m = \#l : 0 \leq l < n : primo(t[l])\}$
falgoritmo

0.3. Ejercicio: Parejas con diferente signo

Se solicita codificar en C un programa de nombre `sumpar.c` que realice las siguientes operaciones:

1. Rellenar una tabla de N posiciones, solicitando al usuario cada uno de los N elementos. Los valores de la tabla se almacenarán en una variable t de tipo `tablaN`, cuya definición en pseudocódigo es:
   tipo `tablaN :: tabla[0..N-1] de enteros` **ftipo**, siendo N una constante, $N > 0$.

2. Mediante el algoritmo `Sumas_parciales_inversas` se calcula una nueva tabla q del mismo tipo que la anterior. Cada elemento de la nueva tabla q se calcula de forma indirecta mediante ciertas sumas de la tabla t a partir de una determinada posición, tal y como indica la postcondición de dicho algoritmo. Este algoritmo tiene que codificarse en forma de una acción cuya declaración es la siguiente:
   accion `sumasParcialesInv (ent t: tablaN, sal q: tablaN)`

3. Entre las dos tablas se cuentan las parejas de distinto signo siguiendo el algoritmo `Parejas_distinto_sig`. Este algoritmo se codifica en forma de una función cuya declaración es la siguiente:
   funcion `parejasDisSig (t,q: tablaN)` **dev** `r: entero`

4. Se debe mostrar el contenido de cada tabla y el número de parejas con distinto signo tal y como se indica en el ejemplo que acompaña este enunciado.

5. El programa permitirá la ejecución del mismo tantas veces como el usuario desee.

6. Recuerda que la codificación del programa en C debe atenerse a las **normas de estilo** que se han fijado en la asignatura y, por tanto, entre otras normas, deberá contener una cabecera con la información correspondiente. Esta información también se deberá mostrar **una única vez** al comenzar la ejecución del programa.

7. La salida resultante de la ejecución del programa para $N = 5$ deberá **corresponderse** con el texto de la siguiente traza:

```
.................
Introduce 5 valores enteros:
Valor pos 0 de la tabla t: 32
Valor pos 1 de la tabla t: -5
Valor pos 2 de la tabla t: 25
Valor pos 3 de la tabla t: -15
Valor pos 4 de la tabla t: 90

La primera tabla es:
32 -5 25 -15 90
la segunda tabla es:
127 -32 5 -25 15
El numero de parejas con distinto signo es: 3

Deseas continuar (s/n): s
```

Los algoritmos a codificar, según el enunciado, se presentan a continuación.

algoritmo Sumas_parciales_inversas
constante N $>$ 0 **fconstante**
tipo tablaN :: tabla [0 .. N-1] de entero **ftipo**

t, q: tablaN;
{Pre : $t = T$}
var
x: entero
fvar
    q[0]:= 0;
    para x= 0 **hasta** N-1 **hacer**
        q[0]:= q[0] + t[x]
    fpara;
    x:= 0;
    {Inv : $t = T \wedge (\forall k : 0 \leq k \leq x : \sum_{k \leq i \leq N-1} t[i] = \sum_{0 \leq i \leq k} q[i]) \wedge 0 \leq x \leq N-1$}
    mientras x $\neq$ N-1 **hacer**
        q[x+1]:= -t[x];
        x:= x + 1
    fmientras
{Post : $t = T \wedge (\forall k : 0 \leq k \leq N-1 : \sum_{k \leq i \leq N-1} t[i] = \sum_{0 \leq i \leq k} q[i])$}
falgoritmo

algoritmo Parejas_distinto_sig
constante N $>$ 0 **fconstante**
tipo tablaN :: tabla [0 .. N-1] de entero **ftipo**

t, q: tablaN;
r: entero;
{Pre : $t = T \wedge q = Q$}
var
x, neg: entero
fvar
    x:= 0;
    r:= 0;
    neg:= 0;
    {Inv : $t = T \wedge q = Q \wedge 0 \leq x \leq N-1 \wedge neg = (\#i : 0 \leq i \leq x-1 : t[i] < 0)$ ...
    ... $r = (\#(i,j) : 0 \leq i < j \leq x : t[i] < 0 \wedge q[j] > 0)$}
    mientras x $\neq$ N-1 **hacer**
        si t[x] $<$ 0 $\rightarrow$ neg:= neg + 1
        [] t[x] $\geq$ 0 $\rightarrow$ continuar
        fsi;
        si q[x+1] $>$ 0 $\rightarrow$ r:= r + neg
        [] q[x+1] $\geq$ 0 $\rightarrow$ continuar
        fsi;
        x:= x + 1
    fmientras
{Post : $t = T \wedge q = Q \wedge r = (\#(i,j) : 0 \leq i < j \leq N-1 : t[i] < 0 \wedge q[j] > 0)$}
falgoritmo

0.4. Ejercicio: maximo segmento ordenado

Se pide codificar en C un programa `maxsegmento.c` que calcule el segmento que contiene la máxima suma de una tabla de $N+1$ elementos e indique si dicho segmento está o no ordenado. Para ello deberá realizar los siguientes apartados:

(1). Leer una tabla de $N+1$ elementos de números enteros. N es una constante entera, $N \geq 0$. Definir el tipo `tabla0N` que en pseudocódigo se define como
`tabla0N :: tabla [0..N] de enteros`.
Para las pruebas suponer que la constante `N`= 5.
(2). Calcular el segmento cuya suma es máxima en la tabla construida en el paso (1). El segmento queda especificado por su posición inicial y posición final; y el valor de la suma.
(3). Indique si los valores del segmento desde su posición inicial hasta la final están en orden ascendente.
(4). El programa deberá seguir las normas de estilo dadas en clase. El programa permitirá su ejecución tantas veces como el usuario desee y mostrará, además de lo indicado por las normas, lo siguiente:

```
......
Introduce 6 valores enteros:
Valor para la posicion 0: -1
Valor para la posicion 1: 2
Valor para la posicion 2: 3
Valor para la posicion 3: -2
Valor para la posicion 4: 4
Valor para la posicion 5: 0
La tabla es: -1 2 3 -2 4 0
La posicion inicial es: 1
La posicion final es: 4
La suma maxima es: 7
El segmento maximo es: 2 3 -2 4
El segmento NO esta ordenado
......
```

(5). Para el paso (2) será necesario escribir un algoritmo, codificado en forma de acción, que realice el cálculo del segmento que continene la máxima suma y además, devuelva la posición inicial y final que ocupa el segmento en la tabla. Su declaración es:

accion segmento_maximo(e/ t: tabla0N, sal/ r: entero, sal/ a: entero, sal/ b: entero)

(6). Para el paso (3) se requiere codificar una función que devuelva un booleano indicando si el segmento está ordenado o no. Su especificación es:

funcion ordenado(t: tabla0N, a: entero, b: entero) **dev** c: booleano
$\{P : 0 \leq a \leq b \leq N\}$
$\{Q : c \equiv (\forall k : a \leq k < b : t[k] \leq t[k+1])\}$
ffuncion

El algoritmo para la acción es el siguiente:

algoritmo segmento_maximo;
const $N \geq 0$ **fconst**
t: **tabla** [0..N] **de** *entero*;
r: *entero*;
a, b: *entero*;
$\{Pre : t = T\}$
var
    x, z: *entero*
fvar
    x:= 0;
    a:= 0;
    b:= 0;
    r:= t[0];
    z:= t[0];
$\{Inv_1 ....\}$ $\{cota\, N - x\}$
    mientras x < N **hacer**
        si (t[x+1] > z + t[x+1]) $\wedge$ (t[x+1] > r) $\rightarrow$ a:= x+1; b:= a
        [] $\neg$ ((t[x+1] > z + t[x+1]) $\wedge$ (t[x+1] > r)) $\rightarrow$
            si z + t[x+1] > r $\rightarrow$ b:= x+1
            [] $\neg$ (z + t[x+1] > r) $\rightarrow$ **continuar**
            fsi
        fsi;
        z:= max(t[x+1], z+t[x+1]);
        r:= max(r, z);
        x:= x + 1
    fmientras
$\{Post : t = T \wedge\ r = \max\left\{\sum_{i\leq k\leq j} t[k] : 0 \leq i \leq j \leq N\right\}\ \wedge\ r = \sum_{a\leq k\leq b} t[k]\ \wedge 0 \leq a \leq b \leq N\}$
falgoritmo

El invariante que ha conducido su diseño es el siguiente:

$$Inv_1 \equiv t = T \wedge\ r = \max\left\{\sum_{i\leq k\leq j} t[k] : 0 \leq i \leq j \leq x\right\} \wedge 0 \leq x \leq N \wedge$$
$$z = \max\left\{\sum_{i\leq k\leq x} t[k] : 0 \leq i \leq x\right\} \wedge r = \sum_{a\leq k\leq b} t[k]\ \wedge 0 \leq a \leq b \leq N$$

0.5. Ejercicio: Tartaglia

Se pide codificar en C un programa `tartaglia.c` que dibuje el Triángulo de Tartaglia que contiene los números combinatorios, para calcular un número combinatorio concreto. En cada fila x-ésima encontramos los número combinatorios C_y^x, con $0 \leq y \leq x$. Las dimensiones del triángulo completo vienen dadas por el mayor valor para x. El rango de x es $0 \leq x \leq N$ siendo N una constante mayor o igual a cero. Seguir los siguiente pasos:

(1) Definir una tabla de N+1 elementos con tipo tabla0N :: tabla[0..N] de enteros. N es una constante, $N \geq 0$. Para los ejemplos, considere que N es 10.
(2) El programa solicita primero un entero n entre $0 \leq n \leq N$. Luego solicita un entero m entre $0 \leq m \leq n$. En caso de que estos dos números no cumplan dichas condiciones el programa termina indicando las condiciones que deben cumplir n y m mediante un mensaje 'error: se debe cumplir $0 <= m <= n <= N$'.
(3) A continuación, si n y m cumplen sus condiones se dibuja el Triángulo de Tartaglia en la pantalla. Para realizar este proceso se debe utilizar una acción de nombre `filaC` cuya declaración es

```
accion filaC(e/s t: tabla0N, e/ x: entero)
```

(4) Finalmente, el programa escribe el número combinatorio C_m^n que se ha solicitado para los dos números dados. (5) La codificación en C debe atenerse a las normas de estilo fijadas en la asignatura. El programa pertmitirá la ejecución del mismo tantas veces como el usuario desee y mostrará, además de los indicado por las normas, lo siguiente

```
......
Introduce un entero n entre 0 y 10: 5
Introduce un entero m entre 0 y 5: 3
Triangulo de Tartaglia para filas 0 hasta 5:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
--------
El numero combinatorio 5 sobre 3 es: 10
desea continuar? s/n: s
Introduce un entero n entre 0 y 10: 5
Introduce un entero m entre 0 y 5: 6
error: se debe cumplir 0 <= m <= n <= 10
desea continuar? s/n: s
```

El algoritmo de la acción filaC(...) se da a continuación. Debe comprender bien su especificación. Básicamente calcula la nueva fila de números combinatorios a partir de la fila anterior.

accion filaC(e/s t: tabla0N, e/ x: entero)
$\{P : 0 \leq x \leq N \wedge (\forall q : 0 \leq q \leq x - 1 : t[q] = C_q^{x-1})\}$
var
y, ant: entero
fvar
    y := 0;
    mientras y < x div 2 + 1 **hacer**
        si y = 0 → t[y] := 1; ant := 1;
        [] y > 0 →
                t[y] := ant + t[y];
                ant := t[y] - ant
        fsi;
        t[x - y] := t[y];
        y := y + 1
    fmientras
$\{Q : (\forall q : 0 \leq q \leq x : t[q] = C_q^x)\}$
faccion

La derivación del algoritmo anterior se basa en que Q es equivalente al predicado,

$$Q : (\forall q : 0 \leq q \leq x \operatorname{div} 2 : t[q] = t[x - q] = C_q^x)$$

usando la propiedad de que $C_q^x = C_q^{x-q}$. El invariante del bucle necesita una variable auxiliar para guardar un valor anterior ya que el cálculo de la nueva fila de números combinatorios se hace sobre la misma tabla que contiene al principio los valores de la fila anterior.

$I1 : (\forall q : 0 \leq q < y \operatorname{div} 2 : t[q] = t[x - q] = C_q^x)$
$I2 : 0 \leq y \leq x \operatorname{div} 2 + 1$
$I3 : y > 0 \Rightarrow ant = C_{y-1}^{x-1}$
$I4 : t[y] = C_y^{x-1}$

0.6. Ejercicio: Números amigos

Los pitagóricos observaron una rara relación entre los números 220 y 284: la suma de los divisores de cada uno de ellos, salvo el propio número, es el otro. Los denominaron números amigos. Durante muchos siglos, la pareja 220 y 284 fueron los únicos amigos conocidos, hasta que en 1636 Fermat descubrió que 17.296 y 18.416 también lo son. En 1638 Descartes, colega y competidor de Fermat, encontró la tercera pareja: 9.363.584 y 9.437.056. (referencia Wikipedia). Podeís comprobar que la pareja 1184 y 1210 también son números amigos.

Escribir un programa en C, `amigos.c`, que busque las parejas de números amigos incluidos en un intervalo de números $[A, B]$ siendo A y B dos números positivos distintos. Seguir los pasos siguientes.

(1) Define una tabla con dos filas y N columnas, del tipo,
$lista :: tabla[0.,1][0..N-1]\, de\, enteros$. N es una constante positiva, por ejemplo, $N = 1000$. En una variable t: $lista$, encontraremos en la primera fila números del intervalo $[A, B]$ y en la segunda fila la suma de divisores propios de dichos números. Los divisores propios de un número para este programa incluyen al 1 pero no al propio número

(2) Escribe una función suma_divisores() siguiendo la especificación

> **funcion** suma_divisores(n: entero) **dev** s: entero
> $\{P : n > 1\}$
> $\{Q : s = \sum d : 1 \leq d < n \wedge n \bmod d = 0 : d\}$
> **ffuncion**

Esta función ya ha sido programada en al menos dos sesiones de prácticas. Tenga en cuenta que no se suma el propio número en el resultado final.

(3). El programa comienza solicitando al usuario dos números A y B, siendo $1 < A < B$ y además $B - A + 1 \leq N$. Si la entrada es incorrecta se escribe por pantalla 'error en la entrada de datos' y a continuación se indica al usuario si 'desea continuar?s/n: '. Mire los ejemplos de ejecución. Recuerde que el programa debe ejecutarse tantas veces como el usuario quiera sin salir del propio programa (normas de estilo). Si la entrada de datos es correcta, el programa debe comportarse como indican los ejemplos, siguiendo el mismo formato de salida de datos.

```
(...)
Introduce un numero A mayor que 1: 234
Introduce un numero B en el intervalo (234, 1233]: 2343
error en la entrada de datos
Deseas continuar (s/n): s
Introduce un numero A mayor que 1: 220
Introduce un numero B en el intervalo (220, 1219]: 290
(candidato,sumadiv):
220 284
222 234
224 280
```

```
246 258
248 232
256 255
258 270
272 286
284 220
290 250
numero de candidatos: 10
pareja de amigos:(220, 284)
Deseas continuar (s/n): s

(...)
Introduce un numero A mayor que 1: 300
Introduce un numero B en el intervalo (300, 1299]: 330
(candidato,sumadiv):
304 316
315 309
318 330
328 302
numero de candidatos: 4
Deseas continuar (s/n): s
```

(4) Para generar la lista de posibles candidatos a 'números amigos' entre los números dados A y B se utilizará el algoritmo que se dá a continuación. En dicho algoritmo se asume la definición de la constante N y del tipo *lista :: tabla* $[0.,1, 0..N-1]$ *de entero*:

algoritmo candidatos;
t: **lista**;
p: *entero*;
a,b: *entero*;
$\{Pre : a = A \wedge b = B \wedge 1 < A < B \wedge B - A + 1 \leq N\}$
var
x, sdv: *entero*;
fvar
$x :=\ a$;
$p :=\ 0$;
mientras $x < (b+1)$ **hacer**
$sdv :=$ suma_divisores(x);
si $(a \leq sdv)$ **and** $(sdv \leq b)$ $\longrightarrow$
$t[0][p] := x$;
$t[1][p] := sdv$;
$p := p + 1$
[] $(a > sdv)$ **or** $(sdv > b)$ $\longrightarrow$
continuar
fsi;
$x :=\ x + 1$
fmientras
$\{Post : a = A \wedge b = B \wedge p = C(a,b) \wedge$
$(\forall j : a \leq j \leq b \wedge S(j) \in [a,b] : t[0][C(a,j)-1] = j \wedge t[1][C(a,j)-1] = S(j))\}$
falgoritmo

En la postcondición de este algoritmo $S(j)$ representa la suma de los divisiores propios de j, y $C(l,r) = (\#j : l \leq j \leq r : S(j) \in [l,r])$ cuenta los números cuya suma de divisores está entre $[l,r]$. El invariante se obtiene por sustitución de $b{+}1$ en la postcondición.

Nota de ayuda: el algoritmo *candidatos* devuelve en p el número de posibles candidatos (puede ser 0) y en la lista t, cada posición $0 \leq i \leq p-1$, $t[0][i]$ es un candidato y $t[1][i]$ es la suma de los divisores propios de ese candidato.

Se debe programar el algoritmo anterior como una acción cuya declaración es:

accion candidatos(e/s t: lista, e/ a, b: entero, sal/ p: entero)

(5) Finalmente, en la tabla obtenida t, como resultado de la acción anterior, se deben buscar las parejas de amigos (deberá escribir el código para realizar lo indicado!!). Se imprime cada pareja que se encuentre o en caso contrario no se imprime nada. Recuerde que se deben seguir las normas de estilo a la hora de escribir el programa en C.