Informe 1ª Práctica Programación II

Jordi Rafael Lazo Florensa

Programació II

Universitat de Lleida

Grau en Enginyeria Informàtica

Las funciones zero y one
public int[] zero() { }
public int[] one() { }

Estas son las 2 funciones más sencillas de esta práctica.

En mi caso, para la función *zero* inicializo un array llamado *array_zero* de 1 posición con un 0 en dicha posición y finalmente retorno ese mismo array.

Para la función *one* inicializo un array llamado *array_one* de 1 posición con un 1 en dicha posición y finalmente retorno ese mismo array.

La función de comparación equals
public boolean equals (int[] number1, int[] number2) { }

La función *equals* compara 2 vectores de enteros y debe devolver una variable *boolean* que retornará *true* or *false* en función si las arrays son iguales o no, respectivamente.

Para resolver este problema empiezo inicializando una variable *boolean* llamada *array_equals=true* y un contador *i=0*. A continuación creo un gran *if* que determinará si el largo de las 2 arrays introducidas son iguales o no. En caso que no lo sean saltará el *else* determinando que las arrays son diferentes (si el largo de las arrays es diferente estamos 100% seguros que no serán iguales) mientras que si son iguales de largo las arrays y además se cumple *boolean = true* las iremos comparando posición por posición (por eso inicializamos la *i=0*). El bucle *while* recorrerá el array *number1.length* (también podría recorrer el *number2.length*) y devolverá *arrays_equals=true* si las posiciones de *number1[i]==number2[i]* son exactamente iguales o *array_equals=false* si no lo son. De este modo evitamos que, aunque las 2 arrays introducidas contengan el mismo número de posiciones, puedan ser distintas.

Finalmente, la función equals devolverá el resultado de la *boolean array_equals*.

La suma de dos números
public int[] add(int[] number1, int[] number2) { }

La función *add* es la función más importante y la columna vertebral de esta práctica y la que más tiempo he necesitado para desarrollar.

La función *add* empieza recibiendo 2 arrays de enteros: *num1* y *num2*. Empiezo inicializando una variable *positions_sum=0* que más a delante determinará el largo del array del resultado de la suma de las 2 arrays (*array_add*), una variable *array_length* que determinará cuál de las 2 es la más larga y un array auxiliar llamada *array_num* (más adelante la usaremos para sumar).

Empiezo creando un *if/else* que determinará cual de las 2 arrays introducidas será la más larga, añadiendo ese valor +1 a la variable creada anteriormente *positions_sum*, determinando el largo de *array_length* y los valores de *array_num*. Soy totalmente consciente que este paso se podría haber implementado en una función auxiliar pero debido a mi falta de experiencia y a mi falta de conocimiento en como implementar esta función no lo he hecho (miedo a estropear el código) aunque espero en un futuro poder hacerlo.

Una vez ya hemos determinados dichos valores, creo un array llamada *array_add* que será donde se guardaran todos los valores que se vayan sumando de las arrays *num1* y *num2*, además declaro las variables *carry=0* (las unidades extras que nos llevaremos en cada suma), *i=0* (para

recorrer la array principal y sumar los valores), *j* (recorrerá los valores del array más largo solo cuando un array sea más largo que el otro) y *k_aux=0* (variable auxiliar para eliminar ceros).

A continuación, creo el bucle *while* principal que realizará la suma de cada array (*num1* y *num2*) guardando dicha suma en *array_add*, además dentro de este bucle, al sumar cada elemento de cada posición de los vectores se puede crear un error de fuera de rango (por ejemplo 9+9=18) por eso he añadido un condicional *if* si el resultado de la suma es >= ya que en este caso se deberán restar 10 unidades y llevar 1 unidad de *carry*, seguidamente incremento la *i* para que se siga repitiendo el bucle. El problema de esta suma es que solo se realiza mientras el largo de las arrays sea igual, en el momento en que una sea más larga que la otra se parará de sumar.

Para solucionar el problema, a continuación del bucle *while* creo un bucle *for* aparte, empezando este por la última posición que leyó el bucle *while*. Dentro del bucle *for* sumo las posiciones que faltan por sumar del *array_num* (contenido total de la array más larga) en la *array_add* junto al *carry* añadiendo también el mismo condicional *if* que añadí más arriba en el bucle *while* (>= *10…-10…carry=10*). Fuera del bucle *for* si el *carry* es diferente a 0 lo sumamos a la última *posición* obtenida.

Esta es la parte más difícil de mi código para que lo pueda entender.

Si la última posición del array*_add = = 0* y es *>1* recorreremos el array de derecha a izquierda siempre que la posición sea un 0 y aumentaremos *k_axu++* por cada cero.

Posteriormente declararemos la *array_add_aux* (cuya función será eliminar los ceros que se creen en las sumas de las arrays que tengan el mismo largo) que tendrá las mismas posiciones que el array_add restando los ceros que habían a la derecha del array.

Finalmente devolveremos *array_add_aux* si la suma contenía ceros y se tenían que eliminar o *array_add* si se tenían que devolver la suma de arrays con diferentes longitudes.

Soy consciente que esta función *add* se podría haber hecho de una manera más simple, llamando a funciones auxiliares creadas por mi y probablemente eliminado algunas redundancias, sobre todo en el primer *if/else* para determinar el array mas largo y en el bucle *while*.

El desplazamiento hacia la izquierda

public int[] shiftLeft(int[] number, int positions) { }

La función *shiftleft* consiste en desplazar el array *number* tantas posiciones a la izquierda como la variable *positions* tiene añadiendo zeros en dicha array.

Para que la función funcione perfectamente he creado un *if* que, mediante la función programada anteriormente *equals*, compara las posiciones del array *number* con la función *zero* de manera que retornará el mismo array siempre que este contenga 1 posición y sea cero.

A continuación, creo *array_shiftleft* que contendrá un largo equivalente a la suma de la variable *positions* más el largo de la array *number*.

Seguidamente creo un bucle *for* que recorrerá la nueva array creada de manera que en esta se añadirán las posiciones del el array anterior (*number*) empezando por el final y añadiendo 0 a la izquierda en funcione del largo de *array_shiftleft*.

Finalmente retornará *array_shiftleft* con los ceros según *positons*.

Multiplicación de un número por un dígito
public int[] multiplyByDigit(int[] number, int digit) { }

Esta función la empiezo creando un *if* muy simple con la condición que, siempre que el digito a multiplicar sea 0, la función retornará la función anteriormente programada *zero*.

Seguidamente inicializo la array *array_multiplybydigit* con el tamaño de la función *zero*.

A continuación, recorro un bucle *while* con un contador inverso, que empezará con la condición que el *digit* sea superior a 0 e irá restando 1 al valor de digit. En la primera vuelta del bucle la array *array_multiplybydigit* tendrá el valor de la suma de *array_multiplybydigit + number* (es decir de una vez *number*) de manera que conseguiré que la array *number* se sume tantas veces como sea necesaria hasta que el valor de *digit* llegue a 0. De esta manera obtendré la multiplicación de dígitos (sumando el valor de *number* tantas veces como *digit* quiera).

Finalmente retornará el valor de *array_multiplybydigit*.

Como apunte remarcar que el *while* se puede cambiar por un *for*.

Multiplicación de dos números
public int[] multiply(int[] a, int[] b]) { }

Para esta función he usado algunas de las funciones anteriores. Cuando desarrollas el *add* y el *multiplybydigit* las funciones posteriores son cada vez mas fáciles.

Empiezo inicializando 2 arrays, *array_multiply* y *array_multiply_aux*. La primera la inicializo con el tamaño de la función *zero* y la segunda no hará falta. Será un array auxiliar que su función será multiplicar las arrays y desplazar el resultado de estas multiplicaciones para posteriormente sumarlas.

A continuación, inicializo el bucle *for* para generar las multiplicaciones.

Primero en la *array_multiply_aux* se guardará el resultado de multiplicar una posición del array *number2* (empezando por la posición *i=0*) por todo el array *number1*.

Seguidamente desplazaré una posición del resultado obtenido (en función de *i*, de manera que en la primera multiplicación al *i* ser 0 no se moverá ninguna posición pero cuando *i* sea 1 se moverá 1 ,i=2 2…y así sucesivamente) con la función *shiftleft* y se guardará en *array_multiply_aux*.

Posteriormente, en el *array_multiply* se guardará la suma de *array_multiply* junto a la *array_multiply_aux* y así hasta recorrer el largo del array *number2*.

Finalmente se retornará el resultado obtenido de *array_multiply*.

El factorial
public int[] factorial(int[] number) { }

El factorial es la ultima función a implementar y requiere de prácticamente todas las funciones anteriormente programadas. Si se programó laguna función mal, en esta función los errores saldrán a la luz y como me paso a mi tuve que revisar el código varias veces antes de que me compilara con éxito, sobre todo es importante que la suma *add* este bien construida.

Esta función la implemento dentro de un gran *if/else*. Empiezo definiendo el condicional *if* y comparo la array *number* con la función *zero* mediante la función *equals*. De manera que si son iguales la función factorial devolverá la función *one* ya que el factorial de 0 es 1.

Dentro del *else* declaro *array_factorial* y un array auxiliar que le llamo *factor* que empieza en el 1 (función *one*). Posteriormente inicializo un bucle que comparará el array *number* con el *factor* y mientras sea falso se repetirá el bucle hasta que los arrays sean iguales. Dentro del bucle guardamos la multiplicación del *array_factorial* con el *factor* y este lo vamos aumentando en 1 su valor sumándole 1 en cada vuelta del bucle.

Finalmente retornaremos el *array_factorial*.

Como pequeña consideración se podría haber cambiado el bucle *while* por un *for*.

Anexo

En este anexo intento mostrar la explicación y razonamiento a varias funciones (*add* y *multiplybydigit*, principalmente) de una manera muy simple y breve. Son parte de los apuntes en los que me centre para resolver la práctica.