PROGRAMACIÓN
Escuela de Informática. UPV. Curso 2001/2002
PROBLEMAS RESUELTOS.
Correspondientes a los temas 2 al 5

Francisco Marqués y Natividad Prieto

Índice General

1 Introducción

Se muestra continuación la resolución de los problemas más significativos que aparecen en los apuntes de la asignatura.

Como no se presenta la resolución de todos los problemas propuestos originalmente (aunque si de la mayoría de los mismos), conviene indicar que no siempre coincide la numeración seguida para los problemas a lo largo de esta resolución con la utilizada originalmente cuando se proponían en los apuntes de cada tema. Se ha respetado, sin embargo, el orden correlativo de presentación de los mismos.

2 Tema 2. CONCEPTOS BÁSICOS

Problemas resueltos:

1. Escribir un programa en Java que pregunte un nombre, dirección y teléfono y escriba en pantalla una ficha con dicha información

   El programa define como variables de tipo cadena de caracteres `nombre`,`apellidos`, `direccion` y `telefono`. Estos datos se leen del teclado, para ello se define la variable `entrada` de tipo `input` y se utiliza la operación `readline`.

Después se presenta en pantalla la información en forma de ficha, para ello se utilizan las primitivas `write` y `writeln` asociadas a la variable `salida` de tipo `output`. Finalmente se cierran los ficheros `entrada` y `salida`.

El programa completo sería:

```
import nsIO.*;
class Ficha {
  public static void main (String param[]) {

   String nombre, apellidos, direccion, telefono;

   input entrada = new input();
   output salida = new output();

   salida.write(‘‘Nombre:’’);
   nombre = entrada.readname();

   salida.write(‘‘Apellidos:’’);
   apellidos = entrada.readname();

   salida.write(‘‘Dirección:’’);
   direccion = entrada.readline();

   salida.write(‘‘Teléfono:’’);
   telefono = entrada.readname();

   salida.writeln(‘‘*** Ficha ***’’);
   salida.writeln(‘‘_____________’’);
   salida.writeln(apellidos+’’, ‘‘+nombre);
   salida.write(‘‘Dirección:’’+direccion+’’\t’’);
   salida.writeln(‘‘Teléfono:’’+telefono);

   entrada.close(); salida.close();
  }
}
```

Un ejemplo de ejecución de este programa sería el siguiente:

```
[....]\$ java EscribeFicha
Nombre:Pedro
Apellidos:Martinez Ferrero
Dirección: c/ Perez Galdós, 156
```

```
Teléfono:963756823
*** Ficha ***
-------------
Martinez Ferrero, Pedro
Dirección: c/ Perez Galdós, 156 Teléfono:963756823
```

2. Escribir un programa en Java que transforme una temperatura dada en grados Fahrenheit a grados Celsius, siendo 1ºC=33.8ºF.

   El programa define como variable de entrada `tempC` y como variable resultado o de salida `tempF`, ambas de tipo real `double`. El dato, `tempF`, se lee del teclado, para ello se define la variable `entrada` de tipo `input` y se utiliza la operación `readdouble`. A continuación se realizan los cálculos para obtener el valor de la temperatura en grados Fahrenheit:

$$^{o}F = \frac{^{o}C * 9}{5} + 32$$

   Posteriormente se presenta en pantalla el resultado obtenido, para ello se utilizan la primitiva `writeln` asociada a la variable `salida` de tipo `output`. Finalmente se cierran los ficheros `entrada` y `salida`.

   El programa completo sería:

```
import nsIO.*;
class Temperatura {
  public static void main (String param[]) {

   double tempC, tempF;

   input entrada = new input();
   output salida = new output();

   salida.write(‘‘Temperatura en ºC:’’);
   tempC = entrada.readdouble();

   tempF = tempC*9/5+32;

   salida.writeln(‘‘Temperatura en ºF=’’+ tempF);

   entrada.close(); salida.close();
  }
}
```

Un ejemplo de ejecución de este programa sería el siguiente:

```
[....]\$ java Temperatura
Temperatura en ºC:100.0
Temperatura en ºF=2012.0
[....]\$ java Temperatura
Temperatura en ºC:0.0
Temperatura en ºF=32.0
```

3. Escribir un programa en Java para calcular la superficie y el volumen de una esfera a partir del valor del radio (supóngase que es un valor positivo)

   El programa define como variable de entrada `radio` y como variables resultado o de salida `superficie` y `volumen`, todas ellas de tipo real, `double`. El dato, `radio`, se lee del teclado, para ello se define la variable `entrada` de tipo `input` y se utiliza la operación `readdouble`. A continuación se realizan los cálculos para obtener la superficie y el volumen de la esfera:

$$superficie = 4 * \pi * radio^2$$

$$volumen = \frac{4}{3} * \pi * radio^3 = \frac{superficie * radio}{3}$$

   Después se presentan en pantalla los resultados obtenidos, para ello se utilizan la primitiva `writeln` asociada a la variable `salida` de tipo `output`. Finalmente se cierran los ficheros `entrada` y `salida`.

   El programa completo quedaría:

```
import nsIO.*;
class Esfera_1 {
  public static void main (String param[]) throws Exception {
   double radio;
   double superficie, volumen;

   input entrada = new input();
   output salida = new output();

   salida.writeln(‘‘Cálculo de la superficie y volumen de una esfera’’);

   salida.write(‘‘Escribe el radio (real positivo):’’);
```

```
    radio = entrada.readouble();

    superficie = 4.0*Math.PI*radio*radio;
    volumen = superficie*radio/3.0;

    salida.writeln(``Superficie=''+superficie);
    salida.writeln(``Volumen=''+volumen);

    entrada.close(); salida.close();
   }
}
```

4. Una empresa de transporte por carretera ha adquirido vehículos nuevos que viajan más rápido que los antiguos. La empresa desearía conocer cómo afectará ésto a la duración de los viajes. Supóngase que la reducción media que se consigue del tiempo total de viaje es del 15%. Escribid un programa en Java que lea el horario de salida y llegada antiguo, calcule el nuevo horario de llegada y muestre en pantalla el nuevo tiempo de viaje y la nueva hora de llegada.

   Los datos del problema planteado son:

   - hora de salida
   - hora de llegada

   El programa debe realizar los cálculos necesarios para presentar los siguientes resultados:

   - duración del viaje
   - hora de llegada

   En primer lugar se plantea la cuestión de cómo representar los datos de entrada del problema, horas de salida y llegada. Éstas se pueden representar como enteros, por ejemplo:

hora	9	9:30	10:15	11:45	20:10	23:50
representación	900	930	1015	1145	2010	2350

   Para resolver el problema planteado se deben de seguir los siguientes pasos:

   (a) Leer los datos de entrada
   (b) Calcular el tiempo de viaje
   (c) Calcular el nuevo tiempo de viaje con la reducción del 15%

(d) Obtener el nuevo tiempo de llegada

Para realizar los cálculos de tiempos de viaje, resulta cómodo pasar los tiempos a minutos; si hhmm es el tiempo en horas (hh) y minutos (mm), por ejemplo 1015 para representar las 10:15, se calcularán los minutos mediante:

$$minutos = \frac{hhmm * 60}{100} + hhmm\%100$$

El proceso inverso para calcular el tiempo en horas y minutos será:

$$hh = minutos/60; mm = minutos\%60;$$

Con todo ello, el método de resolución del problema sería el siguiente:

(a) Leer los datos de entrada, `salida` y `llegada`

(b) Calcular el tiempo de viaje

- Pasamos las horas de salida y llegada a minutos
- El tiempo de viaje es la diferencia:

$$tiempoviaje = (\frac{llegada * 60}{100} + hhmm\%100) - (\frac{salida * 60}{100} + hhmm\%100)$$

(c) Calcular el nuevo tiempo de viaje con la reducción del 15%:

$$tiempoviajenuevo = tiempoviaje - 0.15 * tiempoviaje = 0.85 * tiempoviaje$$

(d) Obtener el nuevo tiempo de llegada La nueva hora de llegada se calcula sumando `tiempoviajenuevo` a la hora de salida:

$$llegadanueva = salida + tiempoviajenuevo$$

Este cálculo se realiza en minutos y ya sólo queda pasarlo a horas y minutos:

$$llegadanueva = (llegadanueva/60) * 100 + (llegadanueva\%60)$$

El programa sería el siguiente:

```
import nsIO.*;
class Trayectos {
   public static void main (String param[]) {
     input entrada = new input();
     output salida = new output();

     int hllegada, hsalida;
```

```
      int tpoviajeNuevo, llegadaNueva;

      salida.write("Hora de salida:");
      hsalida = entrada.readint();
      salida.write("Hora de llegada:");
      hllegada = entrada.readint();

      int salidaMin  =(hsalida/100)*60 + (hsalida\%100);
      int llegadaMin =(hllegada/100)*60 +(hllegada\%100);
      int tpoviaje   =llegadaMin - salidaMin;

      tpoviajeNuevo = (tpoviaje * 85)/100;

      int llegadaNuevaMin = salidaMin + tpoviajeNuevo;

      llegadaNueva=100*(llegadaNuevaMin/60)+(llegadaNuevaMin\%60);

      salida.writeln("Nueva Hora de llegada:"+llegadaNueva);
      salida.writeln("Duración del viaje:"+tpoviajeNuevo+" minutos");

      salida.close(); entrada.close();
   }

}
```

La compilación y un ejemplo de ejecución del programa sería:

```
[....]$ javac Trayectos.java
[....]$ java Trayectos
Hora de salida:900
Hora de llegada:1015
Nueva Hora de llegada:1003
Duración del viaje:63 minutos
```

3 Tema 3. CONDICIONES Y ELECCIÓN

Problemas resueltos:

1. Dados dos números enteros, `num1` y `num2`, realizar un programa que escriba uno de los dos mensajes: "el producto de los dos números es positivo o nulo" o bien "el producto de los dos números es negativo". No hay que calcular el producto.

   Resolución: El problema puede resolverse directamente en Java, utilizando las operaciones predefinidas de entrada/salida, del modo siguiente:

```
import nsIO.*;
class neg_pos {
   public  static void main(String args[]) {

     input fent = new input();
     output fsal = new output();

     //pedir los dos numeros
     fsal.writeln("Introduce los dos numeros: ");

     int num1 = fent.readint();
     int num2 = fent.readint();

     //decidimos como es
     if (num1==0 || num2==0 || (num1>0 && num2>0) || (num1<0 && num2<0))
        fsal.writeln("el producto de los dos numeros es positivo o nulo");
     else fsal.writeln("el producto de los dos numeros es negativo");
   }
}
```

   Es posible modificar la expresión condicional utilizando cualquiera de las siguientes (entre otras):

```
.....
     if ((num1<0 && num2>0) || (num1>0 && num2<0))
       fsal.writeln("el producto de los dos numeros es negativo");
     else fsal.writeln ("el producto de los dos numeros es positivo o nulo");
.....

.....
     if (num1!=0 && num2!=0  && (num1<0 ^ num2<0))
       fsal.writeln("el producto de los dos numeros es negativo");
     else fsal.writeln("el producto de los dos numeros es positivo o nulo");
.....
```

2. Considérese el texto siguiente: "En una empresa el cálculo de las vacaciones pagadas se efectúa de la manera siguiente: si una persona lleva menos de un año en la empresa, tiene derecho a dos días por cada mes de presencia, si no, al menos a 28 días. Si es un directivo y si tiene menos de 35 años y si su antigüedad es superior a 3 años, obtiene 2 días sumplementarios. Si tiene menos de 45 años y si su antigüedad es superior a los 5 años, se le conceden 4 días sumplementarios.

   - Reformular el texto suprimiendo toda ambigüedad.
   - Escribir el programa correspondiente, considerando que los datos son: la antigüedad expresada en meses `ant`, la edad en años `edad`, y la condición de ser o no directivo `dir`. El resultado es el número de días de vacaciones pagadas.
   - Completar el programa introduciendo verificaciones sobre la coherencia de los datos. Por ejemplo, la edad ha de ser inferior a 65 años y superior a 18 años, etc.

   Resolución: El texto no es ambiguo, ya que se determina en el mismo, con precisión, que debe hacerse en cada caso. Sin embargo el texto puede ser confuso debido al hecho de que puede haber directivos que cumplan al mismo tiempo la última condición. Ello lleva a una resolución, expresada directamente en el lenguaje java de la forma siguiente:

```
import nsIO.*;
class antigüedad {
   public  static void main(String args[]) {

      int ant, años; boolean dir;   // datos de entrada
      int diasPag;                        // salida
      input fent = new input();
      output fsal = new output();

      fsal.writeln("Antigüedad en meses?");
      ant = fent.readint();
      fsal.writeln("Edad en años?");
      años = fent.readint();
      fsal.writeln("Es un directivo (s/n?)");
      dir = ('s'==fent.readchar());
      //están todos los datos, efectuar los cálculos

      if (ant<12) diasPag=2*ant;
      else {
          diasPag=28;
          if (dir && años < 35 && ant > 36) diasPag += 2;
```

```
            if (años < 45 && ant > 60) diasPag += 4;
         }
         fent.writeln("Tiene derecho a: " + diasPag + "dias");
      }
   }
```

Una modificación como la propuesta, para validar la edad, podría efectuarse modificando el programa anterior de la forma siguiente:

```
import nsIO.*;
class antigüedad {
   public  static void main(String args[]) {

      int ant, años; boolean dir;   // datos de entrada
      int diasPag;                          // salida
      input fent = new input();
      output fsal = new output();

      fsal.writeln("Antigüedad en meses?");
      ant = fent.readint();
      fsal.writeln("Edad en años?");
      años = fent.readint();
      if (años < 18 || años > 65) out.writeln("Demasiado joven o mayor");
      else {
          fsal.writeln("Es un directivo (s/n?)");
          dir = ('s'==fent.readchar());
          //están todos los datos, efectuar los cálculos

         if (ant<12) diasPag=2*ant;
         else {
             diasPag=28;
             if (dir && años < 35 && ant > 36) diasPag += 2;
             if (años < 45 && ant > 60) diasPag += 4;
         }
         fent.writeln("Tiene derecho a: " + diasPag + "dias");
      }
   }
}
```

3. Considérese el juego siguiente: un jugador (jug. A) elige un número entre 1 y 16. Otro jugador (jug. B) puede hacer cuatro preguntas de la forma: ¿es "13" el número?, a lo que el primer jugador (jug. A) responderá diciendo: `igual`, `mayor`, o `menor`, según el número propuesto sea igual menor o mayor que el elegido.

- Estudiar una estrategia ganadora para el juego.
- Realizar un programa en que el usuario sea el jugador A y el ordenador el B.

Resolución: Para simplificar la solución se supondrá que el intervalo de números entre los que se puede elegir son los del rango comprendido entre 1 y 7, lo que facilitará la discusión. Más adelante se verá una solución general al problema planteado.
La estrategia más razonable consiste en dividir cada intervalo en dos subintervalos de tamaño lo más parecido posible. Con ello conseguimos ante cualquier pregunta descartar el número más elevado posible de valores. Por ejemplo, si el rango de valores en un momento dado está comprendido entre 1 y 7, se puede preguntar: ¿es "4" el número?. Si no se ha acertado los dos posibles rangos de valores son los números entre 1 y 3, por una parte; y los números entre 5 y 7, por otra.

Siguiendo textualmente la estrategia anterior, se puede realizar un programa como el que sigue:

```
import nsIO.*;
class adivina {
   public  static void main(String args[]) {

     input fent = new input();
     output fsal = new output();
     String resp;

     // Preguntamos
     fsal.writeln("es 4 el número?");
     resp = fent.readline();
     if (resp.equals("igual")) fsal.writeln("He acertado");
     else if (resp.equals("menor")) {
              fsal.writeln("es 2 el número?");
              resp = fent.readline();
              if (resp.equals("igual")) fsal.writeln("He acertado");
                    else if(resp.equals("menor"))
                                fsal.writeln("Es el número 1");
                           else fsal.writeln("Es el número 2");
          }
     else {fsal.writeln("es 6 el número?");
              resp = fent.readline();
              if (resp.equals("igual")) fsal.writeln("He acertado");
                    else if(resp.equals("menor"))
                                fsal.writeln("Es el número 5");
                           else fsal.writeln("Es el número 7");
          }
```

```
   }
}
```

4. Se desea escribir un programa para calcular la raíz cuadrada real de un número real cualquiera pedido inicialmente al usuario. Como dicha operación no está definida para los números negativos es necesario tratar, de algún modo, dicho posible error sin que el programa detenga su ejecución. Escríbanse distintas soluciones al problema anterior, haciendo uso de: operadores cortocircuitados, instrucciones condicionales, el operador ternario y mediante tratamiento de excecpciones.

5. Escríbase un programa para simular una calculadora. Considérese que los cálculos posibles son del tipo `num1 operador num2`, donde `num1` y `num2` son dos números reales cualesquiera y `operador` es uno de entre: `+, -, *, /`. El programa pedirá al usuario en primer lugar el valor `num1`, a continuación el `operador` y finalmente el valor `num2`. Resuélvase utilizando tanto instrucciones `"if...then...else"`, como `"switch"`.

   Resolución: El siguiente programa resuelve el problema anterior suponiendo que la entrada de datos se efectúa libre de errores. Hace uso de la instrucción `"switch"`.

```
import nsIO.*;
class calculadora {
   public  static void main(String args[]) {

     input fent = new input();
     output fsal = new output();

     double num1=0, num2=0, result=0;   // num entrada y resultado,
     char op;                   // operador implicado

     fsal.writeln("num1?");
     num1 = fent.readdouble();
     fent.readln();
     fsal.writeln("operador?");
     op = fent.read();
     fsal.writeln("num2?");
     num2 = fent.readdouble();

     switch (op) {
        case '+': result = num1+num2; break;
        case '-': result = num1-num2; break;
        case '*': result = num1*num2; break;
        case '/': result = num1/num2; break;
     }
     fsal.writeln("El resultado es: " + result);
```

```
   }
}
```

La versión siguiente trata la situación en la que uno de las entradas numéricas se ha escrito incorrectamente, originándose una excepción:

```
import nsIO.*;
class calculadora {
   public  static void main(String args[]) {

     input fent = new input();
     output fsal = new output();

     double num1=0, num2=0, result=0;   // num entrada y resultado,
     char op;                  // operador implicado

     try {
         fsal.writeln("num1?");
         num1 = fent.readdouble();
         fent.readln();
         fsal.writeln("operador?");
         op = fent.read();
         fsal.writeln("num2?");
         num2 = fent.readdouble();
     }
       catch (NumberFormatException e) {
          fsal.writeln("ERROR EN LA ENTRADA:");
          fsal.writeln("uno de los numeros introducidos es incorrecto");
          return;
     }

     switch (op) {
        case '+': result = num1+num2; break;
        case '-': result = num1-num2; break;
        case '*': result = num1*num2; break;
        case '/': result = num1/num2; break;
     }
     fsal.writeln("El resultado es: " + result);
   }
}
```

4 Tema 4. LA ITERACIÓN

Problemas resueltos:

1. Escribir un programa en Java que calcule el cuadrado del número que se introduce como dato utilizando únicamente sumas

   Resolución: El cuadrado de un número, `x`, se puede expresar como la suma de `x`, `x` veces. Por lo tanto, si `x` es el número que se introduce por teclado, y `x2` es la variable sobre la que se va a realizar el cálculo, la estrategia o método para realizar este cálculo sería:

   ```
   x2=x;
   repetir (x-1) vez añadir x a x2
   ```

   El código en Java sería:

   ```
   import nsIO.*;
   class cuadrado {
     public void static main (String args)
     { input fent = new input();
       output fsal= new otput();

       fsal.writeln(‘‘Introduce el número:’’);
       int x=fent.readint();

       int x2=x;
       for (int i=1; i<x; i++) x2+=x;

       fsal.writeln(‘‘El cuadrado de ‘‘+x+’’ es:’’+x2);
       fent.close();
       fsal.close();
     }
   }
   ```

   (a) ¿Cuántas variables intervienen y de qué tipo son? `x` que es el dato y `x2` que es el resultado, ambas de tipo entero, y la variable de control del bucle `i`

   (b) ¿Cuál es la variable de control del bucle?. ¿Qué valores toma?. La variable `i` toma valores entre 1 y x-1.

   (c) ¿Cuál es la condición de repetición del bucle?. ¿Cuántas veces se evalúa?. ¿Deja alguna vez de cumplirse?. La condición de repetición del bucle es `i<x`, que se evalúa `x` veces. La variable `i` comienza valiendo 1 y se incrementa de uno en uno por lo que seguro que llega un momento en el que vale `x` y el bucle termina.

(d) ¿Que instrucciones forman el cuerpo del bucle?. ¿Cuántas veces se realizan?. El cuerpo del bucle está formado por la instrucción de incremento de la variable resultado `x2`. Se realiza `x-1` veces.

(e) ¿Qué relación guardan las variables que se utilizan en el bucle?. En la variable `x2` siempre se tiene la suma acumulada de `x`, `i+1` vez, esto es:

$$x2 = \sum_{k=0}^{i} x$$

(f) Una traza del programa para `x=4` sería:

i	x2
–	4
1	8
2	12
3	16
4	

El programa anterior pero utilizando la estructura `while` en lugar del `for` sería:

```
import nsIO*;
class cuadrado {
  public void static main (String args) throws Exception
  { input fent = new input();
    output fsal= new otput();

    fsal.writeln(‘‘Introduce el número:’’);
    int x=fent.readint();

    int x2=x; int i=1;
    while (i<x) {
       x2+=x;
       i++;
    }

    fsal.writeln(‘‘El cuadrado de ‘‘+x+’’ es:’’+x2);
    fent.close();
    fsal.close();
  }
}
```

2. Describe qué hace el siguiente segmento de código en Java. Supongamos que `n` es una variable de tipo `int` que ya tiene asignado un determinado valor

```
{  double resultado=0.0;
   int i;

   if (n<0) i=-n;
   else i=n;

   while (i>=1) {
      resultado+=(1/i);
      i--;
   }
}
```

Resolución: Este segmento de programa utiliza dos variables, `i` que es la variable de control del bucle y `resultado` que es de tipo real, además del dato `n` que se supone conocido.

Inicialmente la variable `i` toma el valor absoluto de `n`. Esta variable va decreciendo de uno en uno hasta valer 0. La condición de repetición del bucle es `i>=1`, y por lo tanto se evalúa una vez más que el valor absoluto de `n`. El cuerpo del bucle es el incremento de la variable `resultado` en la cantidad `1/i` y la modificación de la variable de control del bucle. El segmento de programa calcula la suma de la serie armónica:

$$resultado = \sum_{k=1}^{|n|} 1/k$$

Y la relación que cumplen las variables del bucle en cada iteración es la siguiente:

$$resultado = \sum_{k=i+1}^{|n|} 1/k$$

Este fragmento de programa se podría haber escrito utilizando un bucle for:

```
if (i<0) i=-n else i=n;
for (; i>=1;i--) resultado+=(1/i);
```

3. Escribir un programa en Java para calcular el término n-ésimo de la serie:

$$\begin{aligned} a_n &= 3 * a_{n-1} + 2 \\ a_0 &= 1 \end{aligned}$$

Algunos términos de esta serie son los siguientes: 1, 5, 17, 53, 161, ...
Resolución: El cálculo del n-ésimo término de la serie se puede hacer mediante un bucle en el que en cada iteración se calcula un término de la serie. El valor del término anterior es el contenido de la variable en la iteración anterior.

El programa en Java sería:

```
import nsIO.*;
class serie {
  public void static main (String args)
  { input fent = new input();
    output fsal= new otput();

    fsal.writeln(‘‘Introduce el número:’’);
    int n=fent.readint();
    int a=1;

    for (int i=0; i<n; i++) a=3*a+2;

    fsal.writeln(‘‘El término es:’’+a);
    fent.close();
    fsal.close();
  }
}
```

4. Escribir un programa en Java para calcular el término n-ésimo de la serie de Fibonacci:

$$\begin{aligned} a_n &= a_{n-1} + a_{n-2} \\ a_1 &= 1 \\ a_0 &= 0 \end{aligned}$$

Resolución: Los primeros términos de esta serie son: 0, 1, 1, 2, 3, 5, 8, 13, ... Cada término de esta serie se calcula a partir de los dos anteriores. Por ello necesitaremos dos variables para poder calcular cada término:

```
import nsIO.*;
class serie {
  public void static main (String args)
  { input fent = new input();
    output fsal= new output();
```

```
    fsal.writeln(‘‘Introduce el número:’’);
    int n=fent.readint();

    int anterior=0;
    int actual=1;

    for (int i=0; i<n; i++) {
       actual+=anterior;
       anterior=actual-anterior;
    }

    fsal.writeln(‘‘El término es:’’+actual);
    fent.close();
    fsal.close();
  }
}
```

5. Escribir un programa para convertir números en base decimal a base 2
   Resolución: Si el número en decimal se divide sucesivas veces por 2 y se escriben los restos de derecha a izquierda se obtiene la representación en binario del número. Un algoritmo para hacer esto seria:

```
// n es el numero en decimal

System.out.println(‘‘La representacion en binario del numero ’’+n+’’ es...’’);
System.out.println(‘‘Leelo al reves!!’’);

int m=n;
while (m!=1) {
   System.out.print(m$%$2);
   m=m/2;
}
System.out.println();
```

   Si se deseara devolver el número en binario en una variable de tipo entera podríamos hacer:

```
// n es el numero en decimal

System.out.print(‘‘La representacion en binario del numero ’’+n+’’ es...’’);

int m=n;   //copia del dato
```

```
int pot=1; //guardar la potencia de 10
int res=n$%$2; // representación entera del número binario

while (m!=0) {
   m/=2;
   pot*=10;
   res+=(m$%$2)*pot;
}

System.out.println(res);
```

(a) ¿Cuántas variables intervienen y de qué tipo son? `n` que es el dato, `res` que es el resultado, ambas de tipo entero, la variable de control del bucle `m` y la auxiliar `pot`.

(b) ¿Cuál es la variable de control del bucle?. ¿Qué valores toma?. La variable `m` toma valores entre n y 1, y se va reduciendo a la mitad.

(c) ¿Cuál es la condición de repetición del bucle?. ¿Cuántas veces se evalúa?. ¿Deja alguna vez de cumplirse?. La condición de repetición del bucle es `m!=1`, que se evalúa `log n +1` veces. La variable `m` comienza valiendo n y se va dividiendo por la mitad por lo que seguro que llega un momento en el que vale `1` y el bucle termina.

(d) ¿Que instrucciones forman el cuerpo del bucle?. ¿Cuántas veces se realizan?. El cuerpo del bucle está formado por la instrucción de actualización de la variable resultado `res`, y de las variables `m` y `pot`. Se realiza `log n` veces.

(e) ¿Qué relación guardan las variables que se utilizan en el bucle?. Si estamos en la iteración i-ésima:

$$m = \frac{n}{2^i}; pot = pot * 10^i; res = \sum_{k=0}^{i} \frac{n}{2^k} mod 2 * 10^k$$

(f) Una traza del programa para `n=115` sería:

m	res	pot
115	1	1
57	11	10
28	110	100
14	1100	1000
7	11001	10000
3	110011	100000
1	1100111	1000000

5 Tema 5. VECTORES. RECORRIDO Y BÚSQUEDA

Problemas resueltos:

1. Sea `v` un vector de `num` elementos de cierto tipo base `tbase`. Constrúyase un programa para invertir los elementos del vector, esto es: al finalizar la ejecución del mismo el vector contendrá en su posición 0 el elemento que inicialmente contenía la posición `num-1`, en su posición 1 el elemento que inicialmente contenía la posición `num-2` y así sucesivamente.

   Resolución: Las dos estrategias siguientes permiten resolver el problema:

   (a) Recorrer el vector hasta su elemento central (`(N-1)/2`), utilizando una variable contador `i`, e intercambiar sucesivamente los elementos simétricos. Siendo el elemento simétrico del que tiene posición `i` el de posición `N-i-1`.

   (b) Recorrer el vector hasta su elemento central utilizando dos variables contador: `i` y `j`, de forma que una se incremente en sentido ascendente y la otra se decremente en sentido descendente. Finalizando cuando ambas se crucen.

   Las instrucciones siguientes se corresponden a cada una de las dos estrategias:

   (a)
   ```
   // v es el vector de dimensión N
   tbase aux;
   for (int i=0; i<=(N-1)/2; i++) {
     aux=v[i]; v[i]=v[N-i-1]; v[N-i-1]=aux;
   }
   ....
   ```

   (b)
   ```
   // v es el vector de dimensión N
   tbase aux;
   for (int i=0, int j=N-1; i<j; i++, j--) {
     aux=v[i]; v[i]=v[j]; v[j]=aux;
   }
   ....
   ```

2. Dado el fichero `"pluvio.dat"`, cuyo contenido y organización es ya conocido, realícese un programa para: "Obtener el mes que ha llovido más, de forma acumulada, a lo largo de todo el año, así como la cantidad llovida en dicho mes."

   Resolución: Como los datos se encuentran no ordenados temporalmente en un fichero, es necesario leerlos inicialmente reorganizándolos para facilitar la posterior determinación del máximo. Una posible estrategia consiste en leer los datos del fichero `"pluvio.dat"` en las matrices `lluvia` y `diasM`, tal y como se ha efectuado en la resolución de problemas similares en los apuntes de teoría. Hecho esto, es

necesario recorrer todos los meses del año, y para cada mes recorrer todos sus días para determinar la cantidad llovida a lo largo de dicho mes. Cada vez que se dispone de la cantidad llovida en un mes será necesario determinar si es o no una cantidad máyor que cualquiera de las anteriores.

Las siguientes instrucciones resuelven el problema siguiendo la estrategia planteada en el párrafo anterior:

```
int maxMes=1;  // Contendrá el mes de pluviosidad máxima
int maxCant=0; // Contendrá la pluviosidad del mes maxMes
int acum=0;    // Contendrá la pluviosidad acumulada del mes en curso

for (mes=1; mes<=12; mes++) {
  acum = 0;
  for (dia=1; dia<=diasM[mes]; dia++)
    acum += lluvia[mes][dia];
  if (acum>maxCant) {maxCant=acum; maxMes=mes;}
}
System.out.println(maxMes+" : "+maxCant);
....
```

Una segunda estrategia posible habría consistido en la lectura de los valores día a día del fichero `"pluvio.dat"`, acumulando las pluviosidades de cada mes en una variable al efecto (un elemento de un vector de 12 componentes). Hecho esto, la determinación del máximo de dicho vector, y por lo tanto del mes del año con más pluviosidad, es trivial. Resuélvase el problema de nuevo siguiendo dicha estrategia.

3. La moda de un conjunto de valores es el valor que se repite mayor número de veces en el conjunto. Escríbase un programa para leer desde cierto fichero (que puede ser la entrada estándard) los valores de las edades de un grupo de personas y que presente, a continuación, la moda de ese conjunto de valores (basta con escribir una si hay varias).

   Resolución: Para resolver el problema es necesario acumular el número de personas que tienen las distintas edades posibles. Como se puede considerar que dichas edades están limitadas al rango 0..125, se puede utilizar un vector de enteros, de tamaño 125, para acumular en cada una de sus componentes el número de personas que tienen dicha edad. Así, por ejemplo, la componente 30 del vector contendrá el número de personas que tengan 30 años.

   Una vez se hayan leído y acumulado en el vector todos los valores, un recorrido del mismo bastará para determinar un máximo (valor más frecuente, o moda).

   El siguiente programa en Java resuelve, siguiendo la estrategia anterior, el problema:

```
import nsIO.*;
class moda {
  public static main(String args[]) {
    input fent = new input();
    output fsal = new output();
    int edades[] = new int[125];  // vector acumulador
    // lectura del fichero
    while (fent.more())
      edades[fent.readint()]++;
    // búsqueda del máximo
    int posMax=0;
    for (int i=1; i<=125; i++)
      if (edades[i]>edades[posMax]) posMax=i;
    // resultados
    fsal.writeln("la moda es..."+posMax);
  }
}
```

4. A partir de la información del fichero "pluvio.dat" es posible subdividir los días del año, en función de la cantidad llovida, teniendo en cuenta los intervalos siguientes: [0..10[, [10..20[, ... [90..100[, ... [190 a 200[y, por último, más de 200 litros. Escríbase un programa que escriba en una lista cada intervalo junto con el número total de días del año en que la cantidad llovida pertenezca a dicho intervalo.

   Resolución: Se puede seguir una estrategia similar a la utilizada en el problema anterior, haciendo uso de un vector con 21 componentes (de la 0 a la 20), y almacenando en cada una de ellas la cantidad llovida en el intervalo correspondiente. Por ejemplo, la componente de índice 9 contendrá la pluviosidad comprendida en el rango [90..100[. La última componente del vector (num 20) contendrá la pluviosidad que exceda los 200 litros.

   El siguiente segmento en Java resuelve el problema partiendo de la suposición de que inicialmente se han almacenado en las variables matriz `lluvia` y `ultM` la información correspondiente a la pluviosidad del año, de forma similar a como se ha efectuado en la resolución de problemas en los apuntes de teoría:

```
....
int acumI[] = new int[21];  // vector acumulador;
int aux;
// recorrido de la matriz lluvia
for (int mes=1; mes<=12; mes++)
  for (int dia=1; dia <=ultM[mes]; dia++)
    if (lluvia[mes][dia]>=200) acumI[20]++;
      else { aux = (int)lluvia[mes][dia];
```

```
            acum[aux/10]++;
      }
// escritura de resultados
for (int i=0; i<21; i++)
  System.out.println(i*10+" "+(i*10+10)+" lluvia:"+acum[i]);
....
```

5. Escríbase un programa para determinar si un vector de palabras (`String`) es cap-i-cua, esto es, para determinar si la primera y última palabras del vector son la misma, la segunda y la penúltima palabras del vector también lo son, y así sucesivamente.

   Resolución: Supóngase que el vector `vPal` con `N` elementos de tipo `String` ha sido definido e inicializado. Entonces la siguiente secuencia de instrucciones en Java permiten determinar si dicho vector es o no cap-i-cua:

   ```
   ....
   String vPal[] = new String[N];
   ....
   // El vector vPal ha sido inicializado con las palabras
   ....
   int i=0; int j=N-1;   // para recorrer el vector
   while ((i<j) && !(v[i].equals(v[j]))) {i++; j--;}
   // i>=j o v[i] distinta v[j]
   if (i>=j) System.out.println("Es cap-i-cua");
     else System.out.println("No es cap-i-cua");
   ....
   ```

   Nótese que se trata de de un problema de búsqueda: el de determinar la existencia de posiciones simétricas (`i` y `j`) en el vector que contengan palabras distintas. Si en el recorrido efectuado con la iteración anterior se llega al centro del vector, finalizándose porque los índices señalan la misma posición, o porque se cruzan entonces el vector es cap-i-cua.

6. Sea `v` un vector de valores enteros. Realícese un programa para determinar la posición, si existe, de la primera subsecuencia del vector que comprenda, al menos, tres números enteros consecutivos en posiciones consecutivas del vector.

   Resolución: Se trata de un problema de búsqueda: el de la posición de la primera subsecuencia que satisfaga el enunciado.

   Si el tamaño de la subsecuencia es pequeño (por ejemplo 3, como es el caso) entonces una solución posible consiste en recorrer el vector ascendentemente, comprobando en cada elemento si es o no el inicio de una subsecuencia como la deseada. Así, las siguientes instrucciones siguen la estrategia indicada:

```
....
int v[] = new int[N];
....
boolean subSec = false;    // true si se ha hallado la secuencia
int i = 0;                 // para recorrer el vector

while (i <= N-3 && !subSec) {
  if (v[i]+1==v[i+1] && v[i+1]+1==v[i+2]) subSec=true;
  else i++;
}
// i > N-3 o subSec
if (subSec) System.out.println("Empieza en "+i);
  else System.out.println("No se encuentra");
....
```

Si el tamaño de la subsecuencia fuera grande (mayor que 3 por ejemplo), la estrategia anterior podría ser farragosa e ineficiente. Se podría, por lo contrario, contar cuantas apariciones como las deseadas han habido (con una sola comparación en cada iteración) y, a continuación determinar si es o no el número deseado. Las siguientes instrucciones siguen la estrategia señalada:

```
....
int v[] = new int[N];
int numRep = 3;            // tamaño de la secuencia buscada
....
int conta=1;               // contador de apariciones
int i = 1;                 // para recorrer el vector

while (i < N && conta<numRep) {
  if (v[i]-1==v[i]) conta++;
  else conta=1;
  i++;
}
// i = N o conta=numRep
if (conta==numRep) System.out.println("Empieza en "+ (i-numRep));
  else System.out.println("No se encuentra");
....
```

7. Las instrucciones siguientes permiten encontrar la posición del elemento más pequeño en un vector v de n enteros ($n \geq 1$):

```
....
   int v[] = new int[n];
```

```
   ....
   ....
   int pos, min;
   pos=0; min=v[pos];
   for (int i=1; i<n; i++)
      if v[i]<min { pos=i; min=v[pos];}
   // posición del mínimo en pos;
}
```

Resolución:

(a) ¿De qué va a depender el coste temporal del algoritmo anterior?. Y por lo tanto, ¿cuál es el tamaño del problema?. El coste de este algoritmo dependerá del número de elementos del vector, n; éste es el tamaño del problema.

(b) ¿Existen instancias significativas? No existen instancias significativas ya que se trata de un algoritmo de recorrido en el que se efectúan n iteraciones y cada una de ellas con coste constante.

(c) ¿Cuál podría ser la instrucción crítica en este algoritmo?. ¿Cuántas veces se repite?. Una instrucción crítica es la comparación $v[i] < min$; se repite $n-1$ veces.

(d) ¿Cuál es la complejidad temporal del algoritmo expresada en términos del número de veces que se repite la instrucción crítica?. La complejidad del algoritmo en términos del número de veces que se repite la instrucción crítica es $n-1$. El coste del algoritmo es por lo tanto lineal.

8. Estúdiese la complejidad temporal de los algoritmos iterativos obtenidos de la implementación directa (sin mejoras especiales) de las definiciones de las siguientes operaciones:

   Resolución:

   (a) Producto escalar de dos vectores de dimensión n. El algoritmo correspondiente será un recorrido por los dos vectores implicados, en cada iteración k, se multiplican las correspondientes componentes k de los vectores y se acumulan sobre el resultado. Se trata de un algoritmo de coste lineal con la dimensión de los vectores.

   (b) Suma de dos matrices cuadradas de dimensión nxn. El algoritmo consiste en dos bucles anidados para el recorrido de la matriz por filas y columnas, respectivamente; en cada iteración más interna se calcula cada una de las componentes de la matriz resultante de la suma. El coste del algoritmo es cuadrático con el número de filas o columnas.

   (c) Producto de matrices cuadradas de dimensión nxn. También en este caso se tiene dos bucles anidados; en cada iteración se calcula cada una de las componentes de la matriz resultante; en este caso, este cálculo implica un

recorrido adicional de fila y columna. El algoritmo para calcular la matriz c como producto de las matrices a y b sería:

```
for (int i=0; i<n; i++)
   for (int j=0; j<n; j++) {
      aux=0;
      for (int k=0; k<n; k++) aux=aux+a[i,k]*b[k,j];
      c[i,j]=aux;
   }
```

Si tomamos como instrucción crítica de este algoritmo el producto de componentes de las matrices $aux = aux + a[i,k] * b[k,j]$; el coste del mismo sería cúbico con el número de filas o columnas.