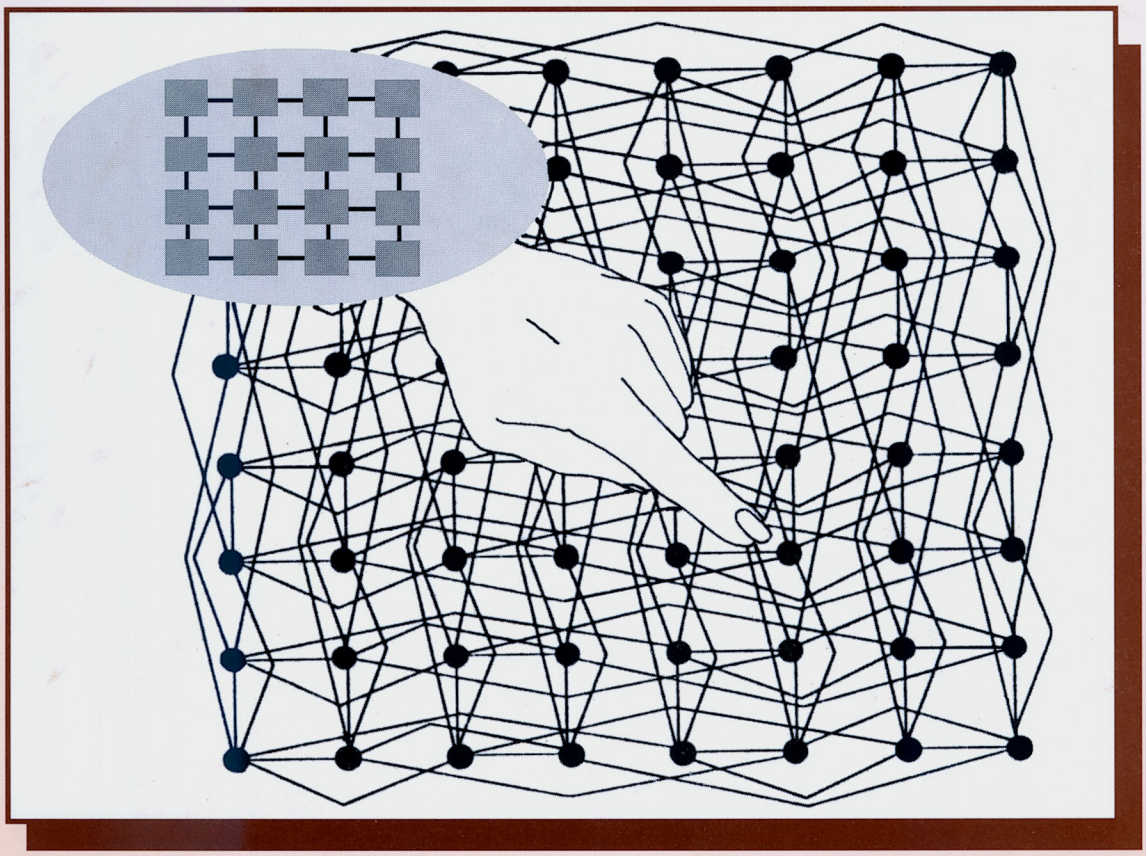
Ingeniería

Revista de la Universidad de Costa Rica
Julio/Diciembre 1996 VOLUMEN 6 Nº 2



INGENIERIA

Revista Semestral de la Universidad de Costa Rica
Volumen 6, Julio/Diciembre 1996 Número 2

CORRESPONDENCIA Y SUSCRIPCIONES

Editorial de la Universidad de Costa Rica
Apartado Postal 75
2060 Ciudad Universitaria Rodrigo Facio
San José, Costa Rica

CANJES

Universidad de Costa Rica
Sistema de Bibliotecas, Documentación e Información
Unidad de Selección y Aquisiciones-CANJE
Ciudad Universitaria Rodrigo Facio
San José, Costa Rica

Suscripción anual:
Costa Rica: ¢ 1 000,00
Otros países: US $ 25,00

Número suelto:
Costa Rica: ¢ 750,00
Otros países: $ 15,00

Edición aprobada por la Comisión Editorial de la Universidad de Costa Rica

Ciudad Universitaria Rodrigo Facio
San José, Costa Rica.

Revisión Filológica: *Lorena Rodríguez*

Diagramación:
José R. Argüello V.

Control de Calidad:
Unidad Diseño Revistas. Oficina de Publicaciones

Impreso en la Oficina de Publicaciones
de la Universidad de Costa Rica

Revista
620.005
I-46i Ingeniería / Universidad de Costa Rica. ——
Vol. I, no. 1 (ene./jun. 1991)— . —— San José, C. R. : Editorial de la Universidad de Costa Rica, 1991— (Oficina de Publicaciones de la Universidad de Costa Rica)
v. : il

Semestral.

1. Ingeniería - Publicaciones periódicas.

CCC/BUCR—250

Ingeniería 6 (2): 45-54 1996 San José, Costa Rica

USO DE CONVENCIONES DE PROGRAMACIÓN PARA DESCIFRAR PROGRAMAS

Adolfo Di Mare[1]

Resumen

Mediante un sencillo ejemplo Pascal se explica cómo puede el programador usar convenciones de programación para lograr entender el código escrito por otros.

Summary

With the use of a Pascal example it is shown how to use programming conventions to understand code written by others.

1. INTRODUCCIÓN.

Uno de los primeros trabajos de investigación que hice fue mi artículo sobre convenciones de programación [DiM-88] para el lenguaje Pascal [CC-85]. Con el tiempo, mis estudiantes de CI-1201 Programación II en la Universidad de Costa Rica y de Algorítmica en el Stvdivm Generale de la Universidad Autónoma de Centro América, se han acostumbrado a usar estas convenciones en sus programas, y esto ha generado que ahora los programas sean más legibles y fáciles de mantener. Con sólo aplicar esas convenciones, hemos logrado hasta que sea natural reutilizar programas (labor en la que mis estudiantes son poco menos que expertos).

Aunque en teoría todos los programadores debemos hacer bien nuestro trabajo, en la práctica encontramos programas pésimos.

Mediante un pequeño ejemplo, quiero exponer la técnica que he usado en muchas ocasiones para averiguar qué hace el código que otro programador escribió. Para eso desarrollo la respuesta a una pregunta que les hice a mis estudiantes en un examen en que evalué el conocimiento que tenían de las convenciones de programación y del tipo abstracto de datos Lista. La pregunta del examen es la siguiente:

Considere el código Turbo Pascal adjunto.

```
PROCEDURE KaWabunga(VAR L :TList);
var cantuca,gula,trimulta_salbaca,banana,
old_last:pnode_replist;begin
if l.rep._last=nil then begin exit;end;
banana:=l.rep._last;old_last:=banana^.next;
cantuca:=l.rep._last;gula:=l.rep.
_last^.next;while gula<>banana
do begin trimulta_salbaca:=gula^.next;gula^.
next:=cantuca;cantuca:=gula;
gula:=trimulta_salbaca;end;banana^.
next:=cantuca;l.rep._last:=old_last;end;
```

Reescriba este código de acuerdo con las convenciones de programación que usted conoce, y diga qué es lo que hace.

Para llegar a saber qué hace este procedimiento he seguido los siguiente pasos:

Paso 1: Indentar el código
Paso 2: Cambiar los identificadores

[1] Profesor Escuela de Ciencias de la Computación e Informática.
Facultad de Ingeniería. Universidad de Costa Rica

Paso 3: Usar el conocimiento sobre el programa para mejorar los nombres de las variables.
Paso 4: Encontrar la función del procedimiento
Paso 5: Escribir la versión final del procedimiento.

2. PASO 1: INDENTAR EL CÓDIGO

Sólo un compilador puede entender la mezcolanza del código original. Como el orden nos ayuda mucho a entender los programas, lo primero que debemos hacer es reescribir el procedimiento KaWabunga() indentando adecuadamente cada una de las instrucciones, para descubrir cuál es la estructura real del programa.

Aunque mis estudiantes debieron realizar su trabajo con papel y lápiz, en la práctica el programador profesional debe usar su editor de textos preferido para realizar este trabajo.

Después de indentar el procedimiento KaWabunga() obtenemos el siguiente código:

```
PROCEDURE KaWabunga(VAR L : TList);

var
  cantuca,
  gula,
  trimulta_salbaca,
  banana,
  old_last      : pnode_replist;

begin { KaWabunga }
  if l.rep._last=nil then begin
    exit;
  end;
  banana   := l.rep._last;
  old_last:= banana^.next;
  cantuca := l.rep._last;
  gula     := l.rep._last^.next;

  while gula<>banana do begin
    trimulta_salbaca := gula^.next;
    gula^.next := cantuca;
    cantuca    := gula;
    gula       := trimulta_salbaca;
  end;

  banana^.next := cantuca;
  l.rep._last:=old_last;
end;  { KaWabunga }
```

3. PASO 2: CAMBIAR LOS IDENTIFICADORES

A diferencia de los compiladores o los autómatas finitos, los seres humanos necesitamos que los identificadores usados en nuestros programas tengan algún sentido. Si los nombres de las variables no tienen nada que ver con su función, es mucho más difícil saber para qué se usan en un algoritmo.

Para cambiarles los nombres a las variables lo lógico es usar la opción del editor que permite cambiar sólo palabras; en el caso del editor incorporado en Turbo Pascal, esa opción aparece en la ventana de sustitución (Find & Replace) bajo el encabezado "Whole words only".

Es importante recordar que siempre debemos realizar las sustituciones de los nombres siguiendo un orden que evite que dos identificadores diferentes terminen llamándose igual. Por ejemplo, si debemos cambiar el Identificador "x" por "y", y a "y" por "x", no basta con hacer una de esas dos sustituciones, pues el resultado sería que terminaríamos con todas las variables llamándose "x", o "y". Uso la notación x->y para indicar que usamos al editor para sustituir la hilera "x" por la hilera "y".

Antes	Sustitución	Después
VAR	x->y	VAR
x, y: INTEGER;		y, y: INTEGER;

Veamos el siguiente ejemplo, en donde al aplicar la sustitución x->y terminamos con todas las variables llamándose "y":

La forma correcta de hacer esta sustitución es usar un nombre temporal (z) para alguna de las variables, así:

```
Antes          Sustitución        Después
                  x->z
x, y: INTEGER;              z, y: INTEGER;

Antes          Sustitución        Después
                  y->x
z, y: INTEGER;              z, x: INTEGER;

Antes          Sustitución        Después
                  z->y
z, y: INTEGER;              y, x: INTEGER;
```

Aunque tomó un poco más de esfuerzo, el resultado final es que hemos intercambiado los nombres de "x" y "y".

Para escoger cuáles son los nombres que usaremos en este paso, lo que hacemos es aplicar el sentido común. En el encabezado del procedimiento aparace el tipo TList, y a lo largo del código se usa la notación Pascal para punteros "^.next". Lo natural entonces es llamar a cada variable con los nombres que usualmente usamos para identificar a los punteros: "p" y "q". Apliquemos entonces las siguientes sustituciones al código que obtuvimos del paso anterior:

```
cantuca            -> p
gula               -> q
trimulta_salbaca   -> xP
banana             -> yP
old_last           -> old_last
```

¿Por qué he escogido los nombres "xP" y "yP"? Bueno, como usualmente usamos "x" y "y" para nombrar cualquier variable, lo que he hecho es agregarles una "P" al final porque sé que estas dos variables son punteros. ¿Por qué no le he cambiado el nombre a "old_last"?. Porque lo más seguro es que ese nombre esté bien puesto, pues ningún programador sería tan torpe de usar un identificador que es tan significativo para nombrar algo diferente a lo que el identificador significa.

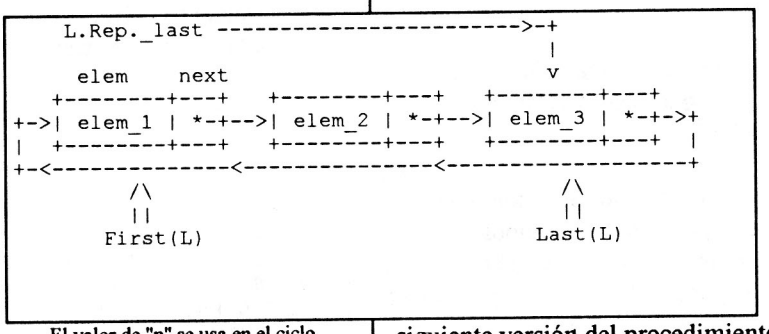
Es importante aclarar el uso de nombres en inglés, y no en español. La principal razón es que los nombres en inglés casi siempre son más cortos. Además, es mucho más fácil nombrar procedimientos en inglés que en español, pues en español muchas veces es necesario calificar los verbos que se usan para nombrar procedimientos con palabras que eliminen ambigüedades. Por ejemplo, la traducción exacta de DISPOSE() es Retorne_Memoria_Dinámica(), que además de ser un identificador muy largo, es incómodo de manipular y de recordar.
El resultado de aplicar estas sustituciones, con el debido cuidado, es el siguiente código:

```
PROCEDURE Incognita(VAR L : TList);
VAR
  p,
  q,
  xP,
  yP,
  old_last : PNode_RepList;
BEGIN { Incognita }
  IF L.Rep._last = NIL THEN BEGIN
    EXIT;
  END;

  yP := L.Rep._last;
  old_last := yP^.next;
  p  := L.Rep._last;
  q  := L.Rep._last^.next;

  WHILE q<>yP DO BEGIN
    xP := q^.next;
    q^.next := p;
    p := q;
    q := xP;
  END;

  yP^.next := p;
  L.Rep._last := old_last;
END;  { Incognita }
```

4. PASO 3: USAR EL CONOCIMIENTO SOBRE EL PROGRAMA PARA MEJORAR LOS NOMBRES DE LAS VARIABLES

Recordemos que este código aparece en un examen en que yo evalué el conocimiento de mis estudiantes sobre la lista [DiM-94b]. Por eso ellos debían saber que la lista con la que trabajarían era una lista circular, en donde el campo "L.Rep._last" siempre apunta al último

nodo de la lista, lo que muestro en el siguiente diagrama:



Con base en el conocimiento sobre la lista es posible escoger un mejor nombre para cada uno de los identificadores del procedimiento Incognita(). A continuación defino cada una de las sustituciones que resultan de usar el nombre que aparentemente más se ajusta a la función que desempeña cada variable en el programa.

p --> p	El valor de "p" se usa en el ciclo WHILE, y como lo usual es llamar a los punteros "p" o "q", parece que el nombre "p" es el adecuado para esta variable.
q --> first	El valor inicial de esta variable es el siguiente a "L.Rep._last", o sea, que es "first".
xP --> firstNext	En el ciclo WHILE, esta variable siempre sigue a "q", por lo tiene sentido llamarla "el siguiente a q".
yP --> old_last	Esta variable toma el valor "L.Rep._last" desde el principio y su valor nunca cambia, por lo que se mantiene apuntando al último nodo de la lista. Por eso debe llamarse "old_last".
old_last --> old_first	El valor que toma esta variable es el puntero al nodo que sigue al nodo último de la lista. Eso quiere decir que este valor es realmente el primero de la lista, por lo que debe llamarse "old_first", y no "old_last".

Vemos ahora que el programador que escribió este código usó un pésimo nombre para la variable "old_last". Como yo escribí el procedimiento KaWabunga(), justifico mi acción argumentando que los profesores, a veces, les ponemos "zancadillas" a los estudiantes en los exámenes. Pero eso no es importante; lo que sí lo es, es que muchos programadores usan nombres "clave" para evitar que otros entiendan el código que escriben, y de esa manera protegen su puesto. En forma alguna apruebo esta mala práctica; pero sí es importante saber que no es necesario entender mucho sobre un programa para saber cuándo un identificador está denominado incorrectamente.

Como en muchas otras actividades, encontramos de nuevo que el sentido común resulta ser un arma eficaz para resolver problemas.

Al aplicar las sustituciones obtenemos la siguiente versión del procedimiento:

```
PROCEDURE Incognita(VAR L : TList);
VAR
  p,
  first,
  firstNext,
  old_last,
  old_first : PNode_RepList;
BEGIN { Incognita }
  IF L.Rep._last = NIL THEN BEGIN
    EXIT;
  END;

  old_last  := L.Rep._last;
  old_first := old_last^.next;
  p         := L.Rep._last;
  first     := L.Rep._last^.next;

  WHILE first<>old_last DO BEGIN
    firstNext   := first^.next;
    first^.next := p;
    p           := first;
    first       := firstNext;
  END;

  old_last^.next := p;
  L.Rep._last    := old_first;
END;  { Incognita }
```

5. PASO 4: ENCONTRAR LA FUNCIÓN DEL PROCEDIMIENTO

Ahora ya el programa no está tan mal escrito como al principio, por lo que podemos dedicarnos ahora a pensar, para adivinar qué hace Incognita(). Tal vez todavía no tenemos suficiente información para adivinar qué hace el programa, pero ya no podemos hacerle cambios puramente superficiales al procedimiento para hacerlo más claro: los cambios puramente cosméticos ya no nos ayudan más.

Al examinar la lógica del ciclo WHILE encontramos que los identificadores "first" y "firstNext" están mal denominados, pues el valor del puntero "first" camina desde el principio hasta el final de la lista, y no permanece estático apuntando al principio de la lista como parecía a primera vista. En la práctica también ocurre que el primer nombre que usamos para un identificador no es el más apropiado. Como nos ha ocurrido aquí, con frecuencia nos encontramos en la necesidad de renombrar algunos identificadores para expresar mejor el algoritmo. En el caso que nos ocupa, hagamos entonces las siguientes sustituciones para que el programa sea más claro:

```
first     ---> q
firstNext ---> qN  (por q-Next)
```

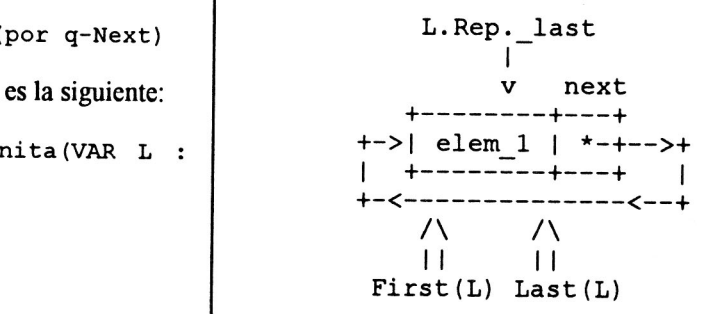
La nueva versión del programa es la siguiente:

```
{ 1}   PROCEDURE Incognita(VAR L :
TList);
{ 2}
{ 3}    VAR
{ 4}    p,
{ 5}    q,
{ 6}    qN,
{ 7}    old_last,
{ 8}    old_first : PNode_RepList;
{ 9}    BEGIN { Incognita }
{10}  IF L.Rep._last = NIL THEN
BEGIN
{11}  EXIT;
{12}  END;
{13}
{14}  old_last  := L.Rep._last;
{15}  old_first := old_last^.next;
{16}
{17}  p := L.Rep._last;
{18}  q := L.Rep._last^.next;
{19}
{20}  WHILE q<>old_last DO BEGIN
{21}  qN      := q^.next;
{22}  q^.next := p;
{23}  p  := q;
{24}  q  := qN;
{25}  END;
{26}
{27}  old_last^.next := p;
{28}  L.Rep._last    := old_first;
{29}  END;  { Incognita }
```

Cuando trabajamos con estructuras de datos complicadas porque utilizan punteros, conviene mucho usar gráficos para entender qué hace el código. Como sabemos que el programa trabaja con listas circulares, conviene correrlo viendo el efecto de cada instrucción en un dibujo. Lo primero que debemos hacer es correrlo con la lista vacía. Como una lista vacía se representa con el valor NIL en "L.Rep._last", entonces la condición del IF de la instrucción {10} es verdadera. Podemos concluir entonces que:

Si la lista está vacía, entonces Incognita() no la cambia. (1)

Ahora usemos una lista que contiene un solo elemento [DiM-94a]:

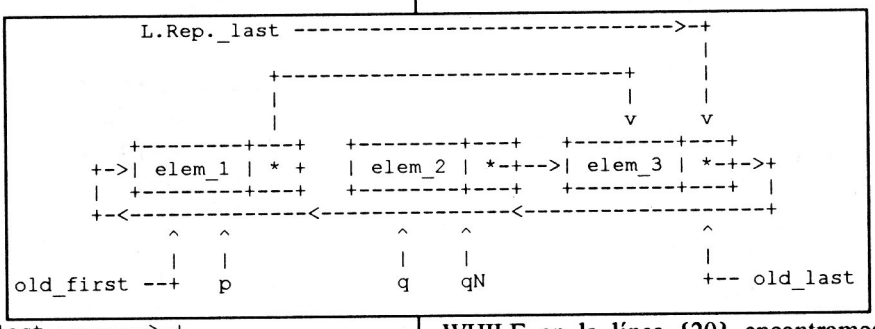


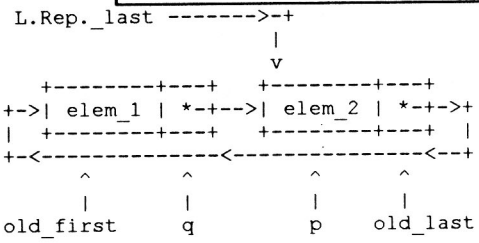
En este caso, vemos que el valor de las variables "old_last", "old_first", "p" y "q" va a ser el mismo, pues el campo ".next" del nodo apunta hacia sí mismo. Esto implica que la condición del WHILE en la instrucción {20} es falsa. El efecto de la instrucción {21} será el de copiar en el nodo el valor que ya tenía, y lo mismo

ocurrirá al cambiar el valor de "L.Rep._last". Concluimos entonces que:

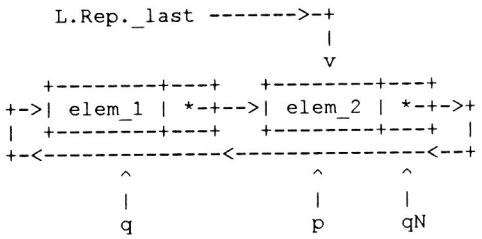
Si la lista tiene sólo un elemento, entonces Incognita() no la cambia. (2)

Veamos ahora qué sucede con una lista de dos elementos. Al ejecutar el procedimiento hasta la instrucción {20} obtenemos lo siguiente:

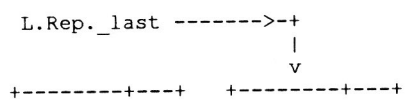


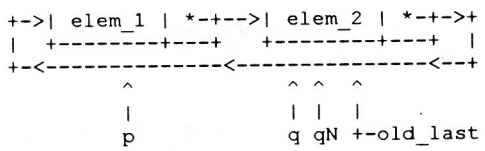


Como "q" no apunta a "elem_2", el resultado de ejecutar las instrucciones {21} y {22} es el siguiente:



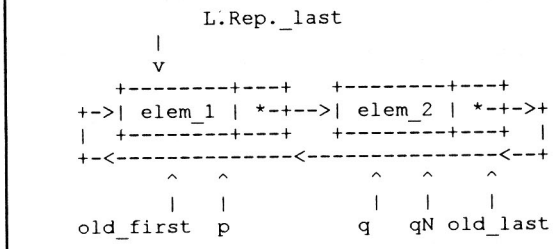
Parece extraño, pero la instrucción {22} deja en el campo ".next" del nodo "elem_1" el mismo valor. Al ejecutar las instrucciones {23} y {24} obtenemos lo siguiente:



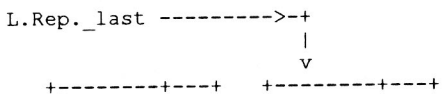


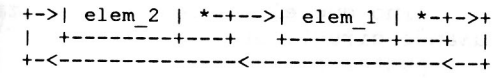
Todo parece indicar que el programador escribió un programa que sobreescribe los valores en el campo ".next" de cada nodo, pero sin cambiarlos, pues al ejecutar la instrucción WHILE en la línea {20} encontramos que la condición es verdadera pues ya "q" apunta al nodo "elem_2" y en el ciclo nunca cambia el valor de "old_last". También ocurre que el valor de "old_last^.next" no cambia en la instrucción {27}.

Sin embargo, al ejecutar la instrucciones {28} sí cambia el valor de "L.Rep._last", pues toma el valor de la variable "old_first":



Todo el efecto de este trabajo ha sido cambiarle el valor a "L.Rep._last". Recordemos que "L" siempre apunta al final de la lista. Redibujemos el diagrama para ver qué ha ocurrido:

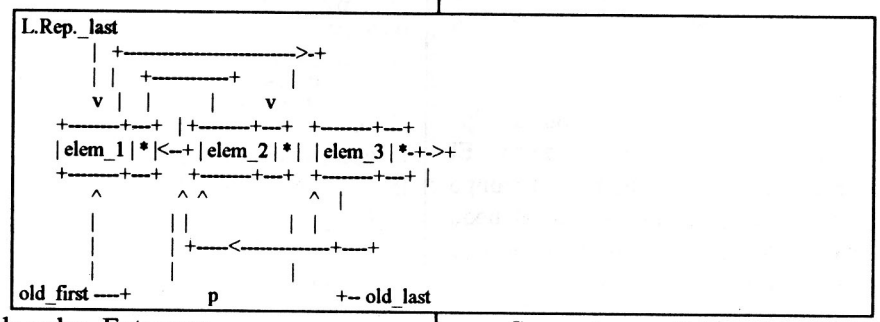




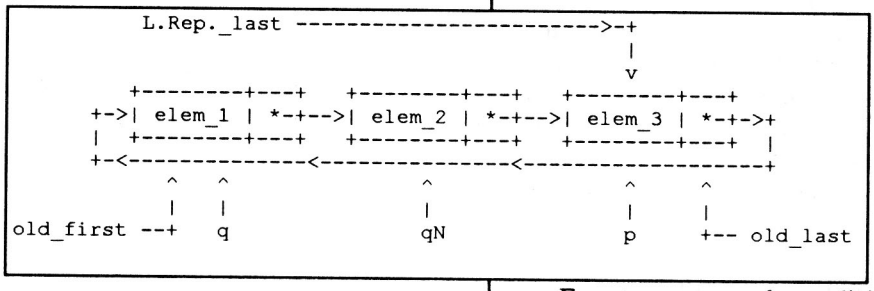
Es evidente que los nodos de la lista han cambiado de orden. Entonces:

Si la lista tiene dos elementos, Incognita() les cambia el orden a los nodos. (3)

El resultado de ejecutar las instrucciones {22}, {23} y {24} es el siguiente:



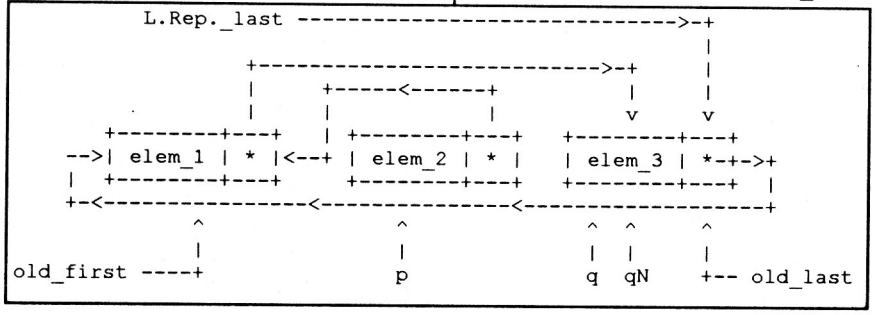
Como la condición del WHILE no es verdadera, después de ejecutar una iteración completa del ciclo obtenemos lo siguiente:



Ahora podemos avanzar un poco más rápido, pues resulta más clara la función de cada variable en el algoritmo. Veamos qué ocurre cuando la lista tiene tres elementos. Después de ejecutar la instrucción {21} tenemos la siguiente situación:

En este momento, la condición del WHILE de la instrucción {20} es verdadera, pues ya "q" le dio vuelta a la lista. Después de ejecutar las instrucciones {27} y {28} obtenemos lo siguiente:

Como "L" apunta al último nodo de la lista, ahora el último es "elem_1". Además, el



primero de la lista siempre es aquél al que el último apunta, por lo que el último de la lista ahora es "elem_3". Como el campo ".next" del nodo "elem_3" apunta a "elem_2", entonces el segundo nodo de la lista es "elem_2". Esto quiere decir que Incognita() ha vuelto al revés la lista, pues al principio el orden que tenía era (1,2,3) y ahora es (3,2,1).

Si ahora examinamos la función de "p" vemos que siempre está un nodo atrás de "q". El ciclo WHILE lo que hace es cambiar el campo ".next" de cada nodo para que apunte al nodo anterior, esto efectivamente ocurre cuando se ejecuta la instrucción {22}.

Entonces, lo que el código hace es volver al revés los punteros de los campos ".next" en cada nodo. Esto quiere decir que este procedimiento vuelve al revés la lista "L".

A primera vista no era obvia la función del procedimiento, pero después de pensar un rato, y de recurrir a unos cuantos dibujos, pudimos descifrar la función del procedimiento Incognita(), cuyo nombre más apropiado por supuesto es Reverse().

6. PASO 5: ESCRIBIR LA VERSIÓN FINAL DEL PROCEDIMIENTO

Cuando escribí este procedimiento usé otros nombres de variables, pero nunca se me ocurrió usar el nombre "q". Transcribo aquí la versión original del procedimiento Reverse(L) que usé para crear a "trimulta_salbaca", y a todos los demás:

```
PROCEDURE Reverse(          { EXPORT
}      { ADH }
  {?} VAR L : TList         { SELF }
);
{ RESULTADO
  Invierte la lista "L" de forma
que sus últimos
  elementos pasen a ser los
primeros.
  - La implementación no copia los
elementos de "L",
    sino que el trabajo se realiza
usando cirugía de
    punteros. }
{ EJEMPLO
  L = (1,2,3,4) ==> Reverse(L) ==>
L = (4,3,2,1).  }
VAR
  p,
  pN,
  pNN,
  old_last,
  old_first: PNode_RepList;

BEGIN { Reverse }
  { se sale si la lista está vacía
}
  IF L.Rep._last = NIL THEN BEGIN
    EXIT;
  END;

  { recuerda cuáles son los nodos
en los extremos }
  old_last  := L.Rep._last;
  old_first := old_last^.next;

  { ciclo para volver todos los
nodos de la lista }
  p  := L.Rep._last;
  pN := L.Rep._last^.next;
  WHILE pN <> old_last DO BEGIN
    { apunte hacia atrás }
    pNN       := pN^.next;
    pN^.next := p;

    { avance al siguiente nodo }
    p  := pN;
    pN := pNN;
  END;

  { vuelve el puntero del último
nodo }
  old_last^.next := p;

  { pone el nuevo último nodo }
  L.Rep._last := old_first;
END;  { Reverse }
```

De la discusión anterior se puede concluir que la función de "q", que yo llamé "pN" en la versión original de Reverse(), es la de servir de pivote para cambiar el valor del puntero en el campo ".next" del siguiente nodo. Es discutible que los nombres de variable "p", "pN" y "pNN" sean los identificadores que mejor ayudan a expresar el algoritmo. Cuando escribí este algoritmo me parecieron los mejores, pero sé que algunos programadores disienten.

7. CONCLUSIÓN

Un programa mal escrito es prácticamente imposible de mantener [LH-89]. Pero con un esfuerzo relativamente pequeño, es posible modificarlo cosméticamente para que el resultado sea un algoritmo más claro. Para esto es importante seguir alguna convención de programación.

Ya existen muchas herramientas que le pueden ayudar al programador a entender el código de un programa [KP-78]. Para el lenguaje C desde hace mucho existe el comando "bc" que ayuda a indentar un programa. En Turbo Pascal, la casa Turbo Power vende un producto llamado Turbo Analyst que, entre otras cosas, incluye herramientas para reformatear el código fuente.

Pero, independientemente de la cantidad de ayuda mecánica, el programador siempre deberá usar el más poderoso de todos sus recursos: el cerebro [Dij-75]. Nunca dejará de ser necesario pensar para arreglar entuertos en los programas.

8. RECONOCIMIENTOS

Esta investigación se realizó dentro del proyecto de investigación 326-93-256 "DBgen: generación automática de programas a partir de su base de datos", inscrito ante la Vicerrectoría de Investigación de la Universidad de Costa Rica. La Escuela de Ciencias de la Computación e Informática también ha aportado fondos para este trabajo.

9. BIBLIOGRAFÍA

[CC-85] Cooper, Doug & Clancy, Michael: Oh! Pascal! 2nd edition, W.W. Norton & Company, New York and London, 1985.

[Dij-75] Dijkstra, Edsger W.: Selected Writings on Computing: A personal perspective, Springer-Verlag, 1975.

[DiM-88] Di Mare, Adolfo: Convenciones de Programación para Pascal; Reporte Técnico ECCI-01-88; Proyecto 326-86-053; Escuela de Ciencias de la Computación e Informática; Universidad de Costa Rica; http:// www.di-mare.com /adolfo/p/ convpas.htm; 1988.

[DiM-94a] Di Mare, Adolfo: La Implementación de Elem.pas; Reporte Técnico ECCI-94-02; Proyecto 326-93-256; Escuela de Ciencias de la Computación e Informática; Universidad de Costa Rica; http:// www.di-mare.com /adolfo/adt/ elem.htm; Mayo 1994.

[DiM-94b] Di Mare, Adolfo: La Implementación de List.pas; Reporte Técnico ECCI-94-06; Proyecto 326-93-256; Escuela de Ciencias de la Computación e Informática; Universidad de Costa Rica; http:// www.di-mare.com /adolfo/adt/ list.htm; Mayo 1994.

[KP-78] Kernighan, Brian W. & Plauger, P. J.: The Elements of Programming Style, second edition, McGraw-Hill Book Company, New York, 1978.

[LH-89] Lieberherr, Karl & Holland, Ian.: Assuring Good Style for Object-Oriented Programs, IEEE Software, pp [38-48], Septiembre 1989.