APÉNDICE A, APARTADO 2

METODOLOGÍA PARA LA ELABORACIÓN DE PROGRAMAS ENTENDIBLES

Definición de estándares y control del proyecto

La experiencia en la programación dicta que un programa es leído más veces de lo que se escribe, y es responsabilidad del programador hacer legibles sus programas.

La creación de un programa legible y confiable es un proceso creativo, por lo cual resulta complicado poner reglas estrictas que imperen sobre el estilo de programación; sin embargo, es posible establecer principios generales que mejoren la legibilidad de los programas, así como de sus nombres para su posterior mantenimiento. ¿Por qué razón es importante pensar en el mantenimiento del sistema? Bien, la razón es que no siempre los miembros del equipo de trabajo que desarrolla un sistema sean los mismos que le darán mantenimiento; por lo que se requerirá que las personas que se encarguen de esa tarea puedan comprender: la programación, el orden en que son llamados los programas y el nombre de las variables.

En un grupo de trabajo deben establecerse normas que definan la metodología a la que debe someterse todo el personal involucrado en el desarrollo de sistemas, con objeto de obtener productos de alta calidad que resulten fáciles de mantener por cualquier miembro del equipo de trabajo. Los programadores que trabajan en la empresa o institución deberán apegarse a los estándares convenidos para el desarrollo de sistemas.

En el diseño de un sistema se definen las especificaciones necesarias para su desarrollo e implantación conforme a los resultados del análisis. Para poder llevar a cabo su desarrollo es necesario diseñar las estructuras de las bases de datos y archivos necesarios para la integración de la información que utilizará, así como los procedimientos de entradas, salidas y procesos que permitirán satisfacer los requerimientos identificados en la etapa de análisis; con ello se logrará optimizar el uso de los recursos de cómputo involucrados y se garantizará la facilidad de operación, la integridad de la información y la confiabilidad del sistema.

Calidad del software

Para asegurar en mayor porcentaje el éxito de un sistema, tanto en la etapa de desarrollo como en la de mantenimiento y corrección de errores, se pueden tomar ciertas medidas al momento de desarrollar los programas que lo conformarán.

Existen técnicas para mejorar la calidad del software y a la vez evitar la existencia de errores en el código, o en su caso, hacer más fácil su localización. Estas técnicas darán mayor legibilidad al código escrito, tanto para el programador como para el personal que se ocupe de proporcionar el mantenimiento.

1. **Insertar espacios en blanco en el código**
   Esto es algo muy simple que se puede hacer para mejorar la productividad del código. Los espacios en blanco permiten que el programador identifique los bloques lógicos del código, lo que lo hace más comprensible. Esta costumbre no representa diferencia en la velocidad de ejecución del programa, por lo que no hay un costo con esta mejora, excepto el papel al imprimirlos.

2. **Insertar sangrías en el código**
   Hacer esto dentro de las estructuras de control tiene una gran influencia en la productividad al hacer más fácil la identificación de los diferentes ciclos del programa. Lo cual tampoco afecta la velocidad de ejecución del programa.

3. **Comentar el código**
   Esta práctica resulta útil, tanto para el programador como para la persona que dará mantenimiento al sistema. Las líneas con comentarios son esenciales si el programador usa algún tipo de artificio ("truco") que piense que puede ser poco comprensible para otras personas, e incluso para la misma persona que elaboró el programa transcurrido unos meses. Los comentarios pueden servir también para "marcar" bloques dentro del código, como cuando se lee un archivo, cuando se efectúan operaciones, cuando se manda a escribir a un archivo de salida, etc. Esta, al igual que las dos técnicas anteriores no afectan la velocidad de ejecución; sólo añaden claridad al programa al momento de leerlo. La experiencia determina que los comentarios al final de una línea son menos claros que los que se escriben en una línea completa.

   Para ilustrar los tres puntos anteriores, a continuación se muestra una parte del código sin el apoyo de estas técnicas, y otro con ellas. La diferencia no es lo que hace el código, sino la forma de su presentación.

```
PROCEDURE Init
LPARAMETERS toOrdEntryForm
LOCAL lnNumParms, ;
lcFilter
thisform.Left = 11
thisform.Top = 2
thisform.cOriginalFormCaption = thisform.Caption
lnNumParms = PARAMETERS()
IF lnNumParms = 0
thisform.cOriginalFormName = thisform.Name
thisform.Name=thisform.Name+ALLT(STR(oApp.AddInstance (thisform)))
thisform.Caption = thisform.Caption + ":" + ;
RIGHT(ALLT(thisform.Name), 1)
ENDIF
OrderEntry::Init()
```

Note lo difícil que es leer el código; en él hay varias secciones diferentes que no se distinguen entre sí. Verifique que es más fácil leerlo si se insertan espacios en blanco y sangrías:

```
PROCEDURE Init

LPARAMETERS toOrdEntryForm
LOCAL lnNumParms, ;
    lcFilter

thisform.Left = 11
thisform.Top = 2
thisform.cOriginalFormCaption = thisform.Caption

lnNumParms = PARAMETERS()

IF lnNumParms = 0
   thisform.cOriginalFormName = thisform.Name
   thisform.Name = thisform.Name +
   ALLT(STR(oApp.AddInstance(thisform)))
   thisform.Caption = thisform.Caption + ":" + ;
                      RIGHT(ALLT(thisform.Name), 1)
ENDIF

OrderEntry::Init()
```

Hasta este punto, el código se lee más fácilmente, tal como si se tratara de instrucciones simples en inglés. Si ahora se añaden comentarios, se convierte en su propia documentación.

```
PROCEDURE Init

LPARAMETERS toOrdEntryForm
LOCAL lnNumParms, ;
    lcFilter

thisform.Left = 11
thisform.Top = 2
thisform.cOriginalFormCaption = thisform.Caption

lnNumParms = PARAMETERS()
*-- Cambiar el título y el nombre del formulario antes de llamar
*-- OrderEntry::Init() para asegurarse de que se ha agregado
*-- el sufijo adecuado tanto al nombre como al título
*-- para permitir múltiples instancias de este formulario.
*-- Sólo se permite una instancia por formulario de entrada de pedidos,
*-- por lo que esta instancia está asociada a un formulario Entrada de
*-- pedidos, no nos preocupamos de cambiar el título.
IF lnNumParms = 0
   thisform.cOriginalFormName = thisform.Name
   thisform.Name = thisform.Name +
                       ALLT(STR(oApp.AddInstance(thisform)))
   thisform.Caption = thisform.Caption + ":" + ;
                         RIGHT(ALLT(thisform.Name), 1)
ENDIF

OrderEntry::Init()
```

4. Alinear el código

Se han realizado estudios que demuestran que al alinear el código se aumenta la legibilidad de éste. Por ejemplo, si se asignan valores a un conjunto de variables de memoria, deben alinearse los signos de igualdad para aumentar su legibilidad. Como se muestra, en los siguientes ejemplos:

Ejemplo 1.

```
m.lclname=company.lname
m.lcfirstname=company.firstname
m.lcaddr1=company.addr1
m.lcaddr2=company.addr2
m.lccity=company.city
m.lcstate=company.state
m.lczipcode=company.zipcode
```

Ejemplo 2.

```
m.lclname       = company.lname
m.lcfirstname   = company.firstname
m.lcaddr1       = company.addr1
m.lcaddr2       = company.addr2
m.lccity        = company.city
m.lcstate       = company.state
m.lczipcode     = company.zipcode
```

5. Detallar los comandos compuestos

Como una consecuencia lógica de la alineación, se deben detallar los comandos compuestos, haciendo una "lista" que sea más fácil de leer, tal como se muestra en los siguientes dos ejemplos:

Ejemplo 1.

```
DEFINE WINDOW myindow FROM 00,00 SIZE 20,40 ;
IN WINDOW myparent FONT 'MS Sans Serif', 10 ;
STYLE 'B' TITLE 'MyWindow' SYSTEM CLOSE ;
FLOAT GROW SHADOW ZOOM
```

Ejemplo 2.

```
DEFINE WINDOW myindow ;
   FROM 00,00 ;
   SIZE 20,40 ;
   IN WINDOW myparent ;
   FONT 'MS Sans Serif', 10 ;
   STYLE 'B' ;
   TITLE 'MyWindow' ;
   SYSTEM ;
   CLOSE ;
   FLOAT ;
   GROW ;
   SHADOW ;
   ZOOM ;
```

6. Adoptar convenciones al nombrar los objetos de los programas

Una convención en los nombres consiste en usar un conjunto uniforme de palabras que sirvan para describir las variables. Éstas deben ser fácilmente comprendidas por la persona que lea el código, no sólo para quien las ideó. Los objetos de un programa son las constantes, variables, procedimientos, funciones y tipos; cada uno de ellos representa una entidad del mundo real, y su función se refleja en la función del mundo real a la que representan. De acuerdo con esta concepción, los nombres de los objetos de un programa deben estar estrechamente relacionados con los nombres de las entidades del mundo real que modelan.

Por ejemplo, si un programa obtiene un reporte de los alumnos titulados en la carrera de ingeniería en computación, y se relaciona con las entidades nombre del alumno, año de ingreso y carrera, éstas deben representarse en el programa por medio de objetos que pueden tener los nombres de: *nombre_alumno*, *anio_ingreso_alumno* y *carrera_alumno*. Esta sería una opción, ya que no es recomendable que sólo se les llamara: *nombre, anio y carrera*, porque no será evidente para el lector que estos nombres pudieran referirse a los datos de titulación del alumno. Peor que esto resulta elegir nombres que no tienen relación con las entidades que modelan, como: abreviaturas criptográficas, identificadores de una sola letra o nombres de personas.

Una mala costumbre es elegir nombres cortos que son fáciles de introducir en el teclado, ya que esto puede originar que los programas sean incomprensibles. La variable *f_esi* significaría algo para el programa y para el programador, pero no para el lector. Observe los dos programas que se presentan a continuación; el primero usa nombres cortos no significativos:

```
Program CF(input, output);
    var t,f : real;
    begin
read(t);
f := t *9/5 + 32;
write(f);
    end.
```

El segundo programa no contiene esta ilegibilidad; al leerlo se piensa en la función del programa, a diferencia del primero:

```
Program Convierte_Celsius_a_Fahrenheit(input, output);
    var Fahrenheit, Celsius : real;
    begin
        read(Celsius);
        Fahrenheit:= Celsius *9/5 + 32;
        write(Fahrenheit);
    end.
```

Si el lector no conoce la fórmula para convertir la temperatura de grados Celsius a Fahrenheit, es poco probable que pueda deducir lo que hace el primer programa. En el segundo ejemplo, el nombre del programa ya explica lo que hace, y la ayuda de identificadores significativos evidencia el procedimiento usado para lograr la conversión.