ESCUELA SUPERIOR POLITECNICA DEL LITORAL

ESCUELA DE CIENCIAS DE LA COMPUTACION

ENSEÑANZA INTERACTIVA DE LA SINTAXIS DEL PASCAL

TESIS DE GRADO
Previa a la Obtención del Título de
ANALISTA DE SISTEMAS

Presentada por:

Deydamia Dominga Fernández Zamora

Guayaquil - Ecuador

1984

A G R A D E C I M I E N T O

A todos aquellos que
de una u otra manera
han ayudado para el
éxito de esta obra.

D E D I C A T O R I A

A MI FAMILIA

.....

ING. LUIS RODRIGUEZ
Director de Tesis

DECLARACION EXPRESA

"La responsabilidad por los hechos, ideas y doctrinas expuestos en esta tesis, me corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL".

(Reglamento de Exámenes y Títulos profesionales de la ESPOL).

.....Deydamia D. Fernández Z.....

DEYDAMIA DOMINGA FERNANDEZ ZAMORA

Autora

R E S U M E N

Se ha podido llegar a observar que muchas personas que desean aprender el Pascal no lo hacen, ya sea porque consideran tedioso el tener que traducir un libro para poder llegar a comprenderlo o porque simplemente no disponen del material y/o conocimientos necesarios.

En vista de esto, la presente obra tiene como objetivo principal ensenar la sintaxis del Pascal de una manera que sea fácil de comprender para el usuario.

Además de llegar a conocer la forma de cómo está estructurado el Pascal, el usuario puede evaluar los conocimientos que vaya adquiriendo a través de la lectura, así como mandar a imprimir la unidad en repaso, siempre que lo requiera, pudiendo terminar su estudio en el momento en que lo desee.

I N D I C E G E N E R A L

INTRODUCCION

Debido al interés demostrado tanto de estudiantes como de profesores, así como de personas particulares a la institución de llegar a aprender un lenguaje estructurado como el Pascal, este Sistema se ha desarrollado con la finalidad de enseñar la sintaxis de este lenguaje en una forma conversacional, logrando para ello recopilar y seleccionar información que sirva como base para la realización de este trabajo.

El primer capítulo de esta tesis explica de una manera general sobre el Computador que se utilizó para el desarrollo de este Sistema, así como de su manejo.

El segundo capítulo trata sobre los objetivos del Sistema propiamente dicho, así como de los elementos que lo conforman y su modo de operación, mostrando para ello ilustraciones para una clara y mejor comprensión del mismo.

En la parte final se adjunta una copia del contenido de la sintaxis del Pascal con su respectivo índice para una mejor referencia del mismo; al igual que una copia de los programas que tienen lugar en el Sistema.

GUIA PARA OPERAR EL COMPUTADOR TRS-80 MODELO II DE LA RADIO SHACK

1.1 Breve descripción del Sistema

1.1.1 Computador

El TRS-80 Modelo II de la Radio Shack es un sistema de computador basado en diskettes, que consiste de dos componentes mayores:

- una Consola de despliegue con dispositivo de diskette incorporado
- un teclado cerrado separado

El software del Sistema Operativo es cargado desde el diskette por un programa de "realimentación" incorporado.

1.1.2 Procesador

En el corazón del Computador está un microprocesador Z-80 A, operando en su velocidad máxima de diseño (4 millones de ciclos-máquina por segundo).

El procesador recibe impulso y reposiciona las instrucciones desde memoria de lectura solamente (ROM). Después el programa de inicialización TRSDOS es cargado des

de el diskette, esta ROM es electrónicamente desconectada del sistema y reemplazada con memoria de acceso al azar (RAM).

1.1.3 Memoria de acceso al azar (RAM)

El sistema básico incluye 32K bytes de memoria de acceso al azar (1K = 1024). Un adicional de 32K bytes puede ser agregado, para un total de 64K bytes de RAM direccionable.

1.1.4 Pantalla de visión

La pantalla de visión tiene su propia placa controladora LSI, para liberar el procesador Z-80A desde la pantalla renovando y relacionando tareas.

La pantalla ofrece dos modos: 80 caracteres por 24 líneas y 40 caracteres por 24 líneas. El conjunto de caracteres representables incluye el conjunto completo de ASCII (alfabeto en letras mayúsculas y minúsculas, números y símbolos especiales), mas 32 caracteres gráficos. Cada caracter puede ser representado tanto como blanco sobre negro o negro sobre blanco.

1.1.5 Teclado

El teclado del Modelo II tiene su propio controlador LSI

para linerar el procesador Z-80A desde el teclado buscando y relacionando tareas. El teclado es una caja separada y esta conectado a la Consola de pantalla mediante un cable incorporado en el frente inferior de la Consola.

El Modelo II tiene las teclas de escribir regulares (letras, números y símbolos de puntuación); no obstante, cada una de éstas teclas puede sacar muchos códigos diferentes al Computador, dependiendo del modo en que esté el teclado: Automático, Cambio, Mayúsculas o Control. Además, el teclado ofrece una tecla de repetición y dos teclas de "función" programables.

1.1.b Dispositivo flotante de diskette

El Modelo II incluye un dispositivo de diskette incorporado de 8". Proyectado para que 3 dispositivos más puedan ser anadidos en una unidad de Expansión externa. A causa de una técnica especial de grabación de alta densidad, cada diskette puede contener 509.184 bytes de información, lo cual es más que 5 veces la capacidad de un diskette de 5-1/4".

El "Dispositivo del Sistema" (el que está incorporado) debe siempre contener un diskette del Sistema Operativo. Los otros dispositivos opcionales pueden estar dedicados

exclusivamente al almacenamiento de programas y datos del usuario.

1.1.7 Interconexiones de Periféricos

Hay cuatro conexiones de interconexión en lo posterior de la Consola de pantalla:

- Dos canales en serie (RS-232-C) Entrada/Salida (E/S)
- Un canal de E/S paralelo, por ejemplo, para conexión de impresoras en línea de interconexión paralela para TRS-80 regular
- Canal de E/S de dispositivo flotante para conexión de la unidad de Expansión de diskettes del Modelo II.

La Consola de pantalla también provee conectores y ranuras para futura expansión.

1.2 Operación

1.2.1 Encendiendo y apagando el Computador

1. Asegúrese que todos los dispositivos en el Sistema estén vacíos, y todos los componentes estén apagados.

2. Encienda el Computador (Consola de pantalla). Espere hasta que el mensaje "INSERT DISKETTE" aparezca en la pantalla. Si el mensaje para aparecer dentro de diez

falla, presione RESET. Si el aún falla para aparecer, apague el Computador y cheques todas las consecciones. Espere al menos 15 segundos antes de empezar de nuevo el paso 1.

3. Cuando "INSERT DISKETTE" es presentado, encienda el Sistema de Expansión de Diskette y todos los otros periféricos.

4. Inserte el diskette del Sistema en el dispositivo 0 y cierre la puerta del dispositivo. El Computador cargará el sistema operativo y le impulsa a ingresar la fecha y hora.

5. Antes de apagar el Computador, remueva todos los diskettes desde todos los dispositivos. Luego apague el sistema completo.

6. Después de cualquier pérdida de potencia, espere al menos 15 segundos antes de encender el Sistema otra vez. Empiece en el paso 1.

1.2.2 Usando el interrupto RESET

Si usted en cualquier tiempo perdiera el control sobre el teclado del Sistema, o usted simplemente quiere re-inicializar, presione RESET hacia arriba momentáneamente y libé

relo. El Computador repetirá la secuencia de encendido, pero los contenidos de la memoria del usuario no serán afectados. Usted no necesita remover el diskette durante esta secuencia de reposición.

1.2.3 Insertando un diskette

1. Si la puerta del dispositivo está cerrada, ábrala presionando la barra de liberación hasta que la puerta salte al abrirse.

2. Remueva el diskette del Sistema Operativo desde su envoltura de almacenamiento. Coja el lado de la etiqueta con la etiqueta mirando afuera desde la pantalla e insértelo en la abertura del dispositivo.

3. Suavemente ponga el diskette hasta el fin en la abertura.

4. Cierre la puerta moviéndola hacia la izquierda hasta que suene secamente en el lugar. Alguna presión puede ser requerida.

1.2.4 Removiendo un diskette

Nunca remueva un diskette mientras la luz de selección del dispositivo esté encendida, o mientras un archivo del

diskette está abierto.

Presione la barra de liberación del dispositivo. La puerta se abrirá y el diskette parcialmente será expulsado. Cuidadosamente remuévalo, teniendo cuidado que la superficie brillante del diskette no toque en la salida el armazón o la puerta del dispositivo.

Una vez que un diskette ha sido acomodado en el dispositivo, usted debe cerrar la puerta del dispositivo antes de que pueda remover el diskette.

1.2.5 Cargando el Sistema Operativo

Cuando el Computador le impulsa a INSERT DISKETTE, cuidadosamente inserte el diskette del Sistema Operativo en el dispositivo.

Tan pronto como usted cierre la puerta, el Computador empezará la realimentación del Sistema Operativo.

Si nada sucede cuando usted cierra la puerta, el diskette está probablemente insertado incorrectamente. Remuévalo y re-insértelo correctamente.

El Computador luego ejecutará un programa diagnóstico antes de iniciar el Sistema Operativo. Esto le permitirá

verificar que el sistema completo esté trabajando en orden - antes de que intente cualquier procesamiento de datos.

Después de la terminación del Programa Diagnóstico, el Computador cargará el Sistema Operativo.

Un mensaje INITIALIZING aparece brevemente, y la luz de la barra de liberación del dispositivo del diskette se enciende y luego se apaga. Aproximadamente en 15 segundos el aviso de derecho de copia de la Radio Shack aparece, la última línea dice:

ENTER DATE (MM/DD/YYYY)

Escriba la fecha. Use un número de dos dígitos para el mes y día y un número de cuatro dígitos para el año. Escriba una diagonal para separar el mes, día y año, luego de presionar ENTER, este mensaje aparece:

ENTER TIME (HH.MM.SS)

Escriba la hora. Use un número de dos dígitos para la hora, minutos y segundos. Escriba un punto para separar los números, luego presione ENTER (para saltar la hora justamente presione ENTER).

Si usted contesta la solicitud incorrectamente, el mensaje "BAD Response" aparece, seguido por la solicitud de la fecha o de la hora. Cuando esto sucede, escriba la fecha o la hora otra vez y luego presione ENTER.

1.2.6 Ajustamiento de la Pantalla de visión

Controles de brillo y contraste están localizados en el área de descanso en la parte inferior izquierda de la Consola de Pantalla. Ajuste como sea necesario para una calidad de pantalla confortable.

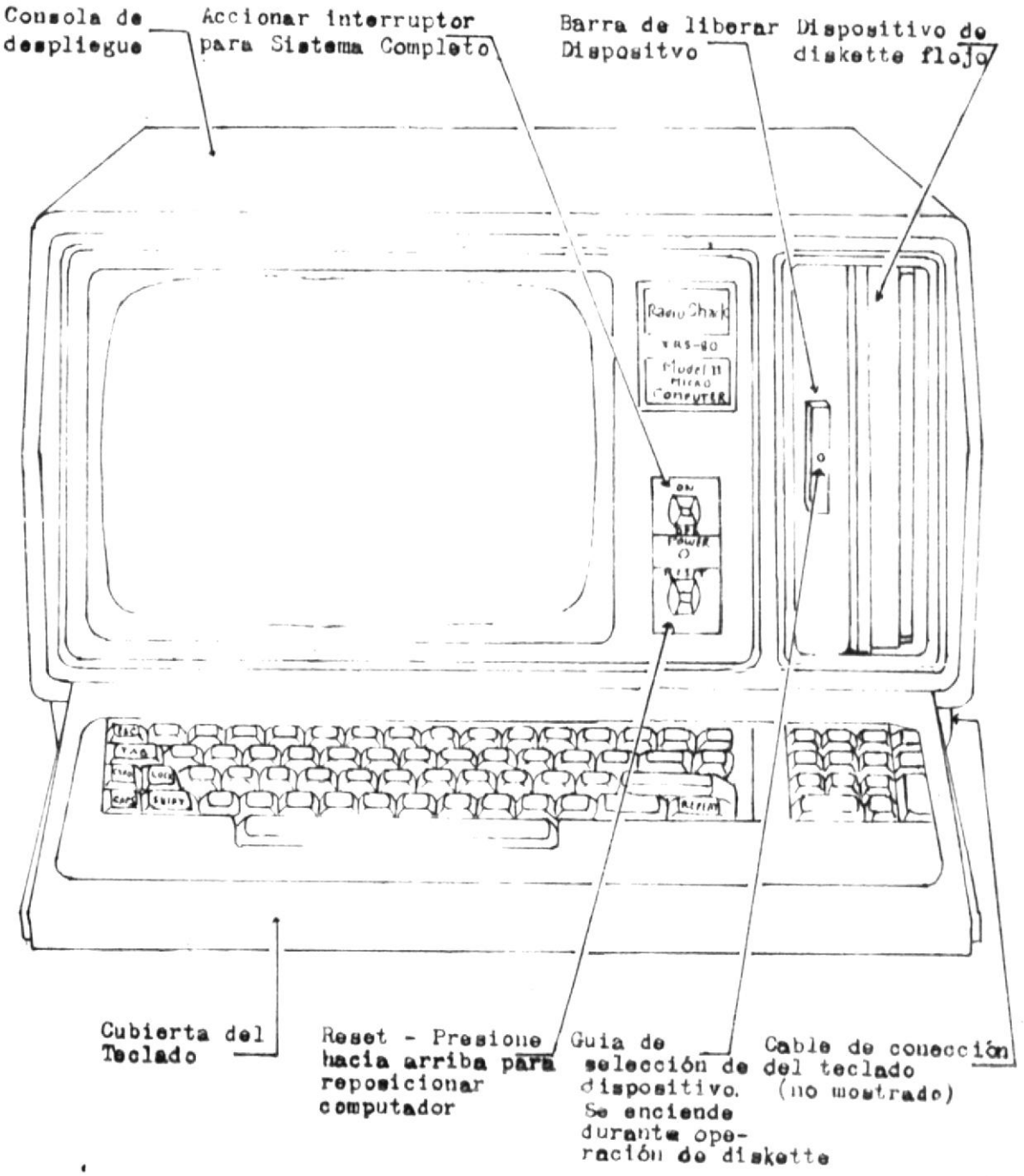


fig. 1 TRS-80 Modelo II

SISTEMA DE ENSEÑANZA INTERACTIVA DE LA SINTAXIS DEL PASCAL

2.1 Descripción general del Sistema

El objetivo principal del Sistema es dar a conocer la forma sintáctica de los puntos más sobresalientes del Lenguaje Pascal, de una manera clara y concisa, mediante el uso de un micro computador TRS-80 Modelo II.

La operación del Sistema se basa en el manejo de dos grandes archivos, los mismos que contienen:

- Información concerniente a la sintaxis del Pascal
- Datos referentes a la evaluación realizada sobre la sintaxis del Pascal

Es mediante la utilización de estos archivos, que el Sistema se encuentra formado por los siguientes procesos:

- Creación y mantenimiento del texto de la sintaxis
- Creación y mantenimiento de la evaluación de la sintaxis
- Impresión del texto que contiene la sintaxis del Pascal
- Presentación de la sintaxis en forma interactiva
- Evaluación de la sintaxis mediante preguntas de respuestas múltiples.

2.2 Definición de Archivos

2.2.1 Archivo Sintf01

La finalidad de este archivo es contener información relacionada con la sintaxis del Lenguaje Pascal.

Su organización es indexada. El nombre de su registro es Reg-01, siendo su longitud de 80 caracteres y teniendo como clave de acceso el campo Keycap

El registro Reg-01 esta constituído por:

DESCRIPCION	NOMBRE	TIPO	LONGITUD
Clave de acceso	Keycap	X	6
Sub-campos	Cap	N	2
	Uni	N	2
	Pant	N	2
	Lin	N	2
Contenido	Conten	X	60

2.2.2 Archivo Sintf02

En este archivo se almacena información sobre la evalua-

ción que se realiza al texto que contiene la sintaxis del lenguaje Pascal.

La organización de este archivo es indexada. El registro se accesa por medio de la clave Keyeva. El nombre del registro es Reg-02 y tiene una longitud de 67 caracteres.

Cada registro está formado por:

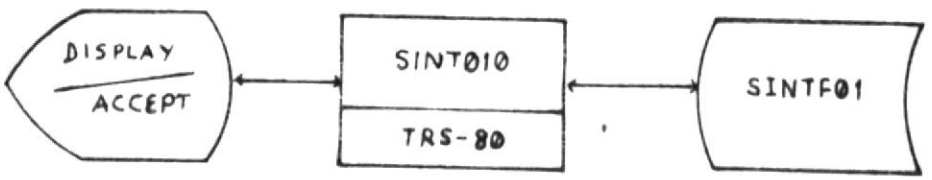
DESCRIPCION	NOMBRE	TIPO	LONGITUD
Clave de acceso	Keyeva	X	6
Sub-campos	Cap	N	2
	Pant	N	2
	Lin	N	2
Tipo de registro	Tipo	X	1
Contenido	Conten	X	60

2.3 Programas

2.3.1 Programa SINTO10

El objetivo de este programa es permitir el ingreso, modificación, eliminación y consulta de los datos del archivo que contiene el texto sobre la sintaxis del Pascal.

El diagrama de bloque es el siguiente:



El programa utiliza el siguiente archivo:

NOMBRE	MODO DE APERTURA	MODO DE ACCESO
SINTF01	I-O	DYNAMIC

El procedimiento a realizarse es el siguiente:

Inicialmente el programa presenta una pantalla en la que se pide digitar la palabra clave que permitirá el acceso a este proceso (ver fig. # 2).

Si la clave está errada el programa termina. Caso contrario, dentro de pocos segundos se despliega un menú en el

que constan cinco alternativas a elegir (ver fig. # 3).

El módulo INGRESO exnibe una pantalla en la que se solicita digitar la clave y contenido del registro a ingresar, con opción a corregir (ver fig. # 4).

Si se elige el módulo MODIFICACION, se presenta una pantalla en la que se requerirá que se escriba la clave mediante la cual se accesará al registro que se desea modificar con la oprtunidad de corregir el requerimiento hecho (ver fig. # 5 y fig. # 6).

En el módulo ELIMINACION se despliega una pantalla en la que se impulsa a escribir la clave por medio de la que se accesará al registro a eliminarse. Este proceso también brinda la oprtunidad de corregir la solicitud realizada, (ver fig. # 7 y fig. # 8).

Para el módulo CONSULTA se ofrece la alternativa de consultar el archivo bien desde el inicio o desde un registro específico del mismo (ver fig. # 9 y fig. # 10). Sea cual fuere la decisión tomada, a continuación se presentarán los registros consultados, brindando la oportunidad de terminar la consulta en cualquier instante (ver fig. # 11).

Si la opción FINALIZAR es la elegida, se procede a cerrar el archivo y el programa termina.

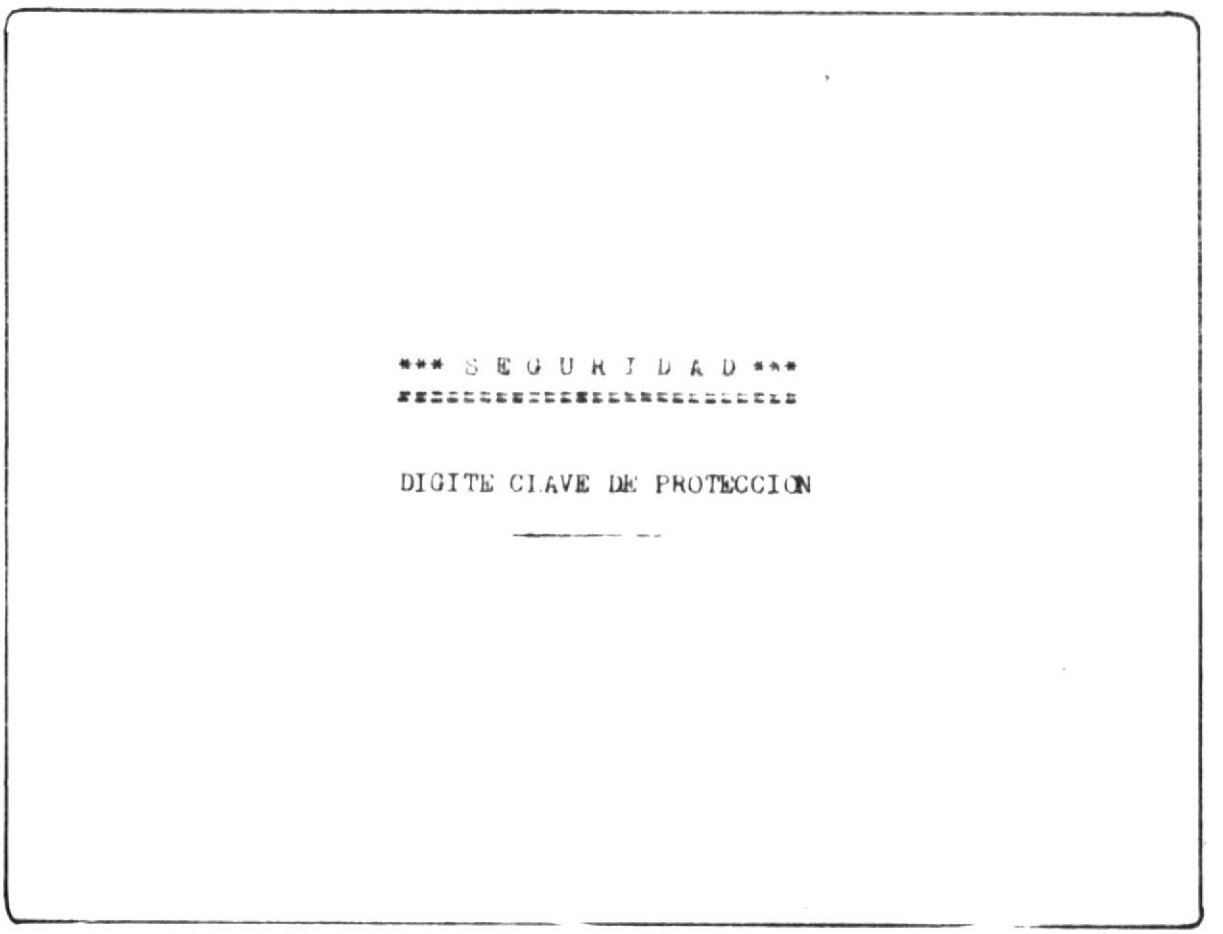


fig. 2 Pantalla donde se solicita digitar palabra clave.

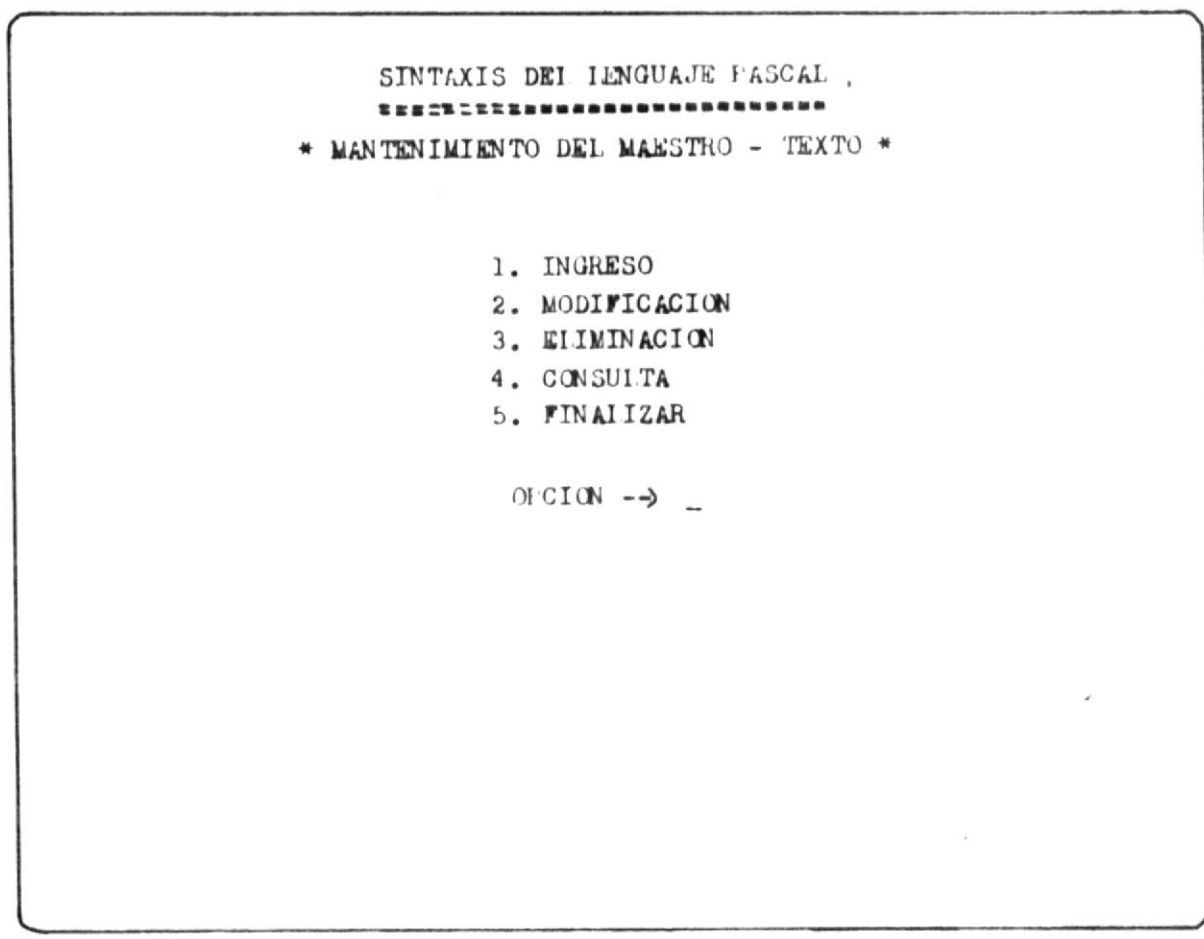


fig. 3 Opciones para el mantenimiento de la sintaxis del Pascal.

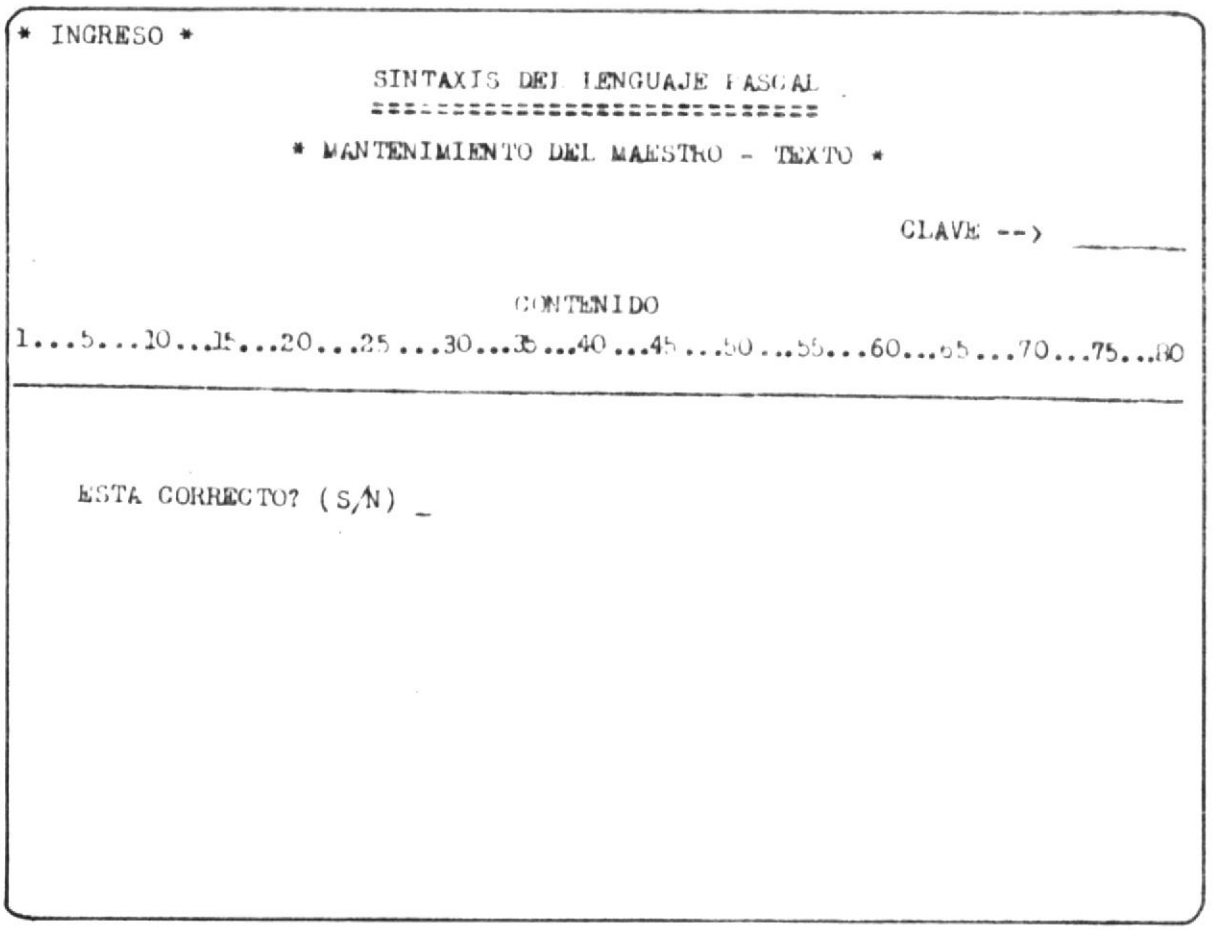


fig. 4 Pantalla donde se solicita digitar la clave y el contenido del registro a ingresar.

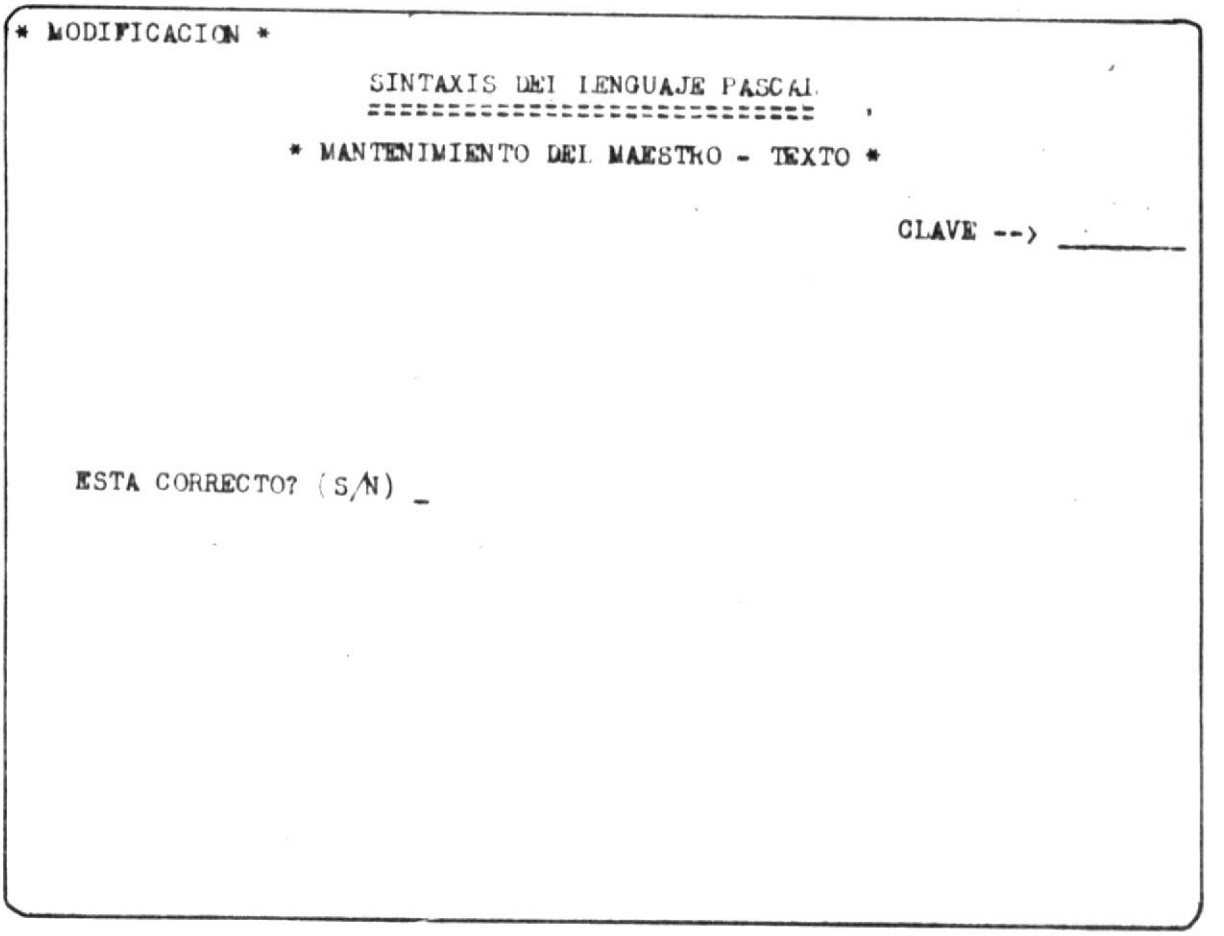


fig. 5 Pantalla donde se solicita la clave del registro a modificar.

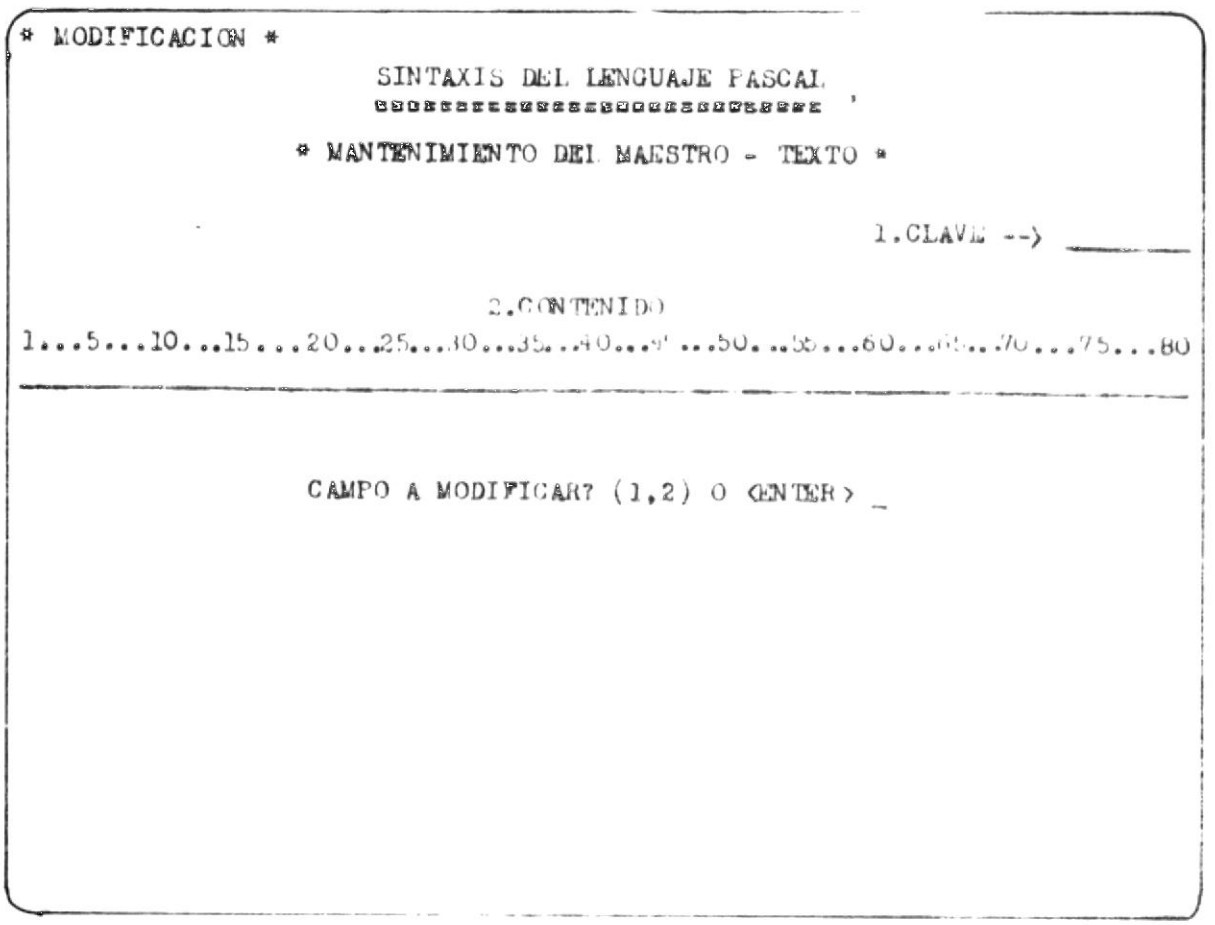


fig. 6 Muestra el contenido del registro a modificar.

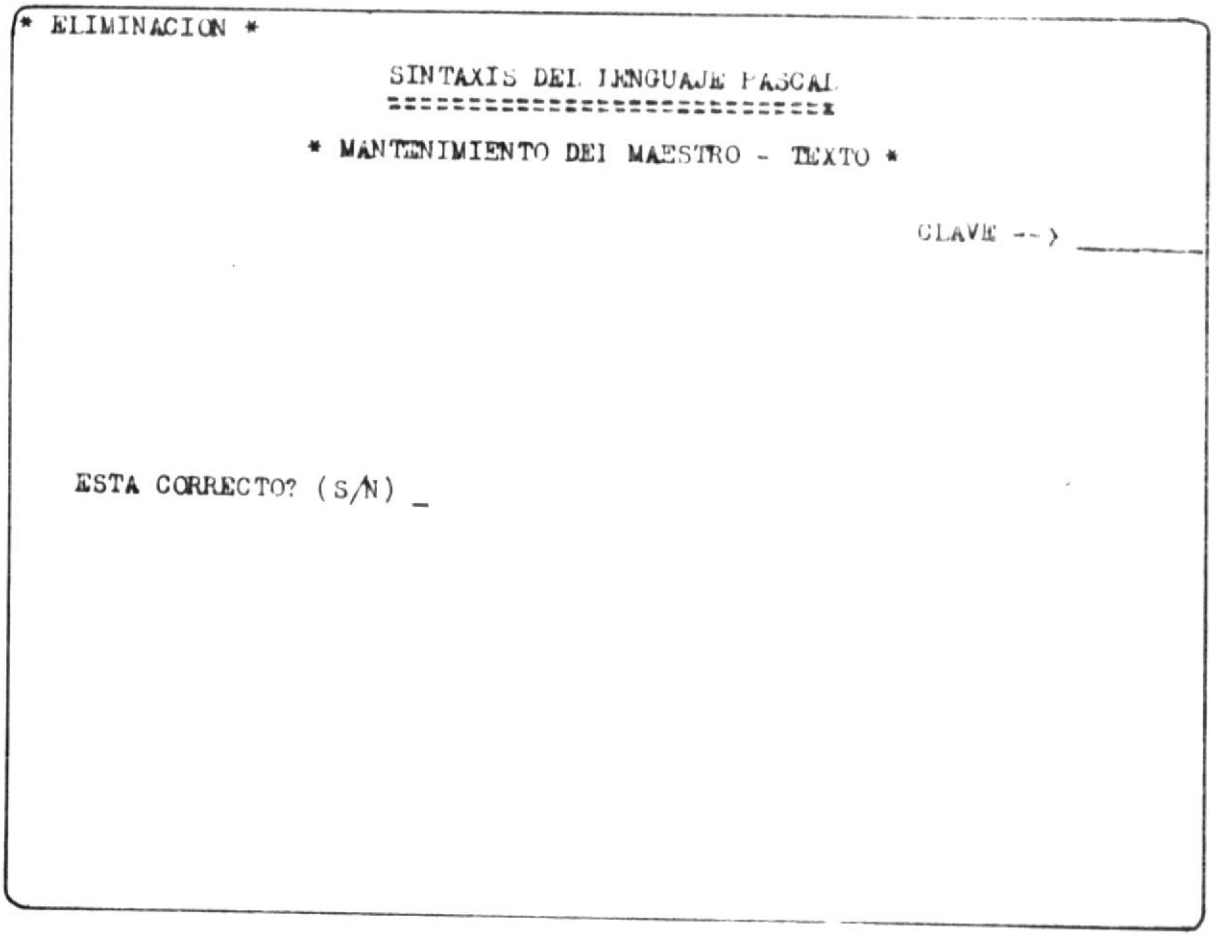


fig. 7 Solicita la clave del registro que se desea eliminar.

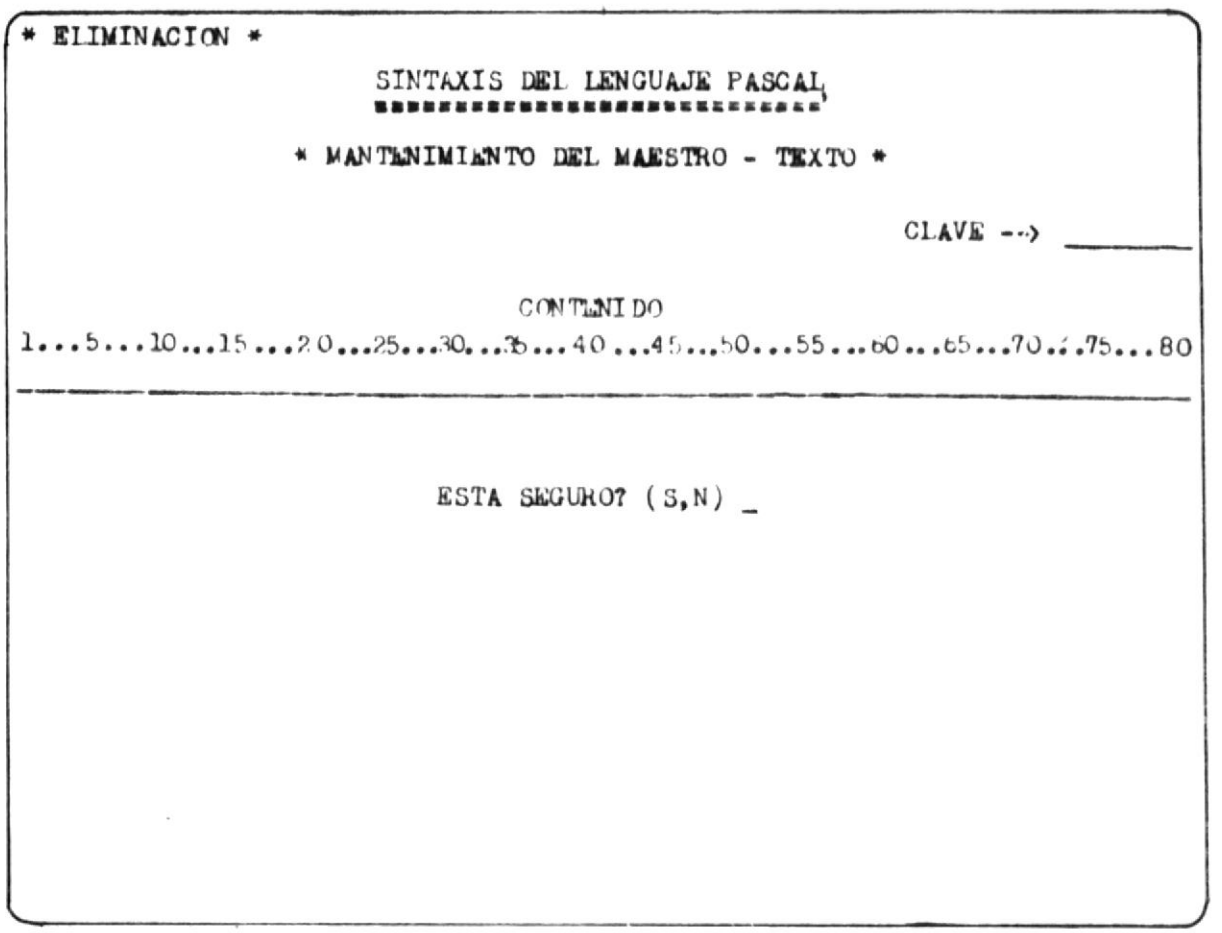


fig. 8 Muestra el contenido del registro a eliminarse

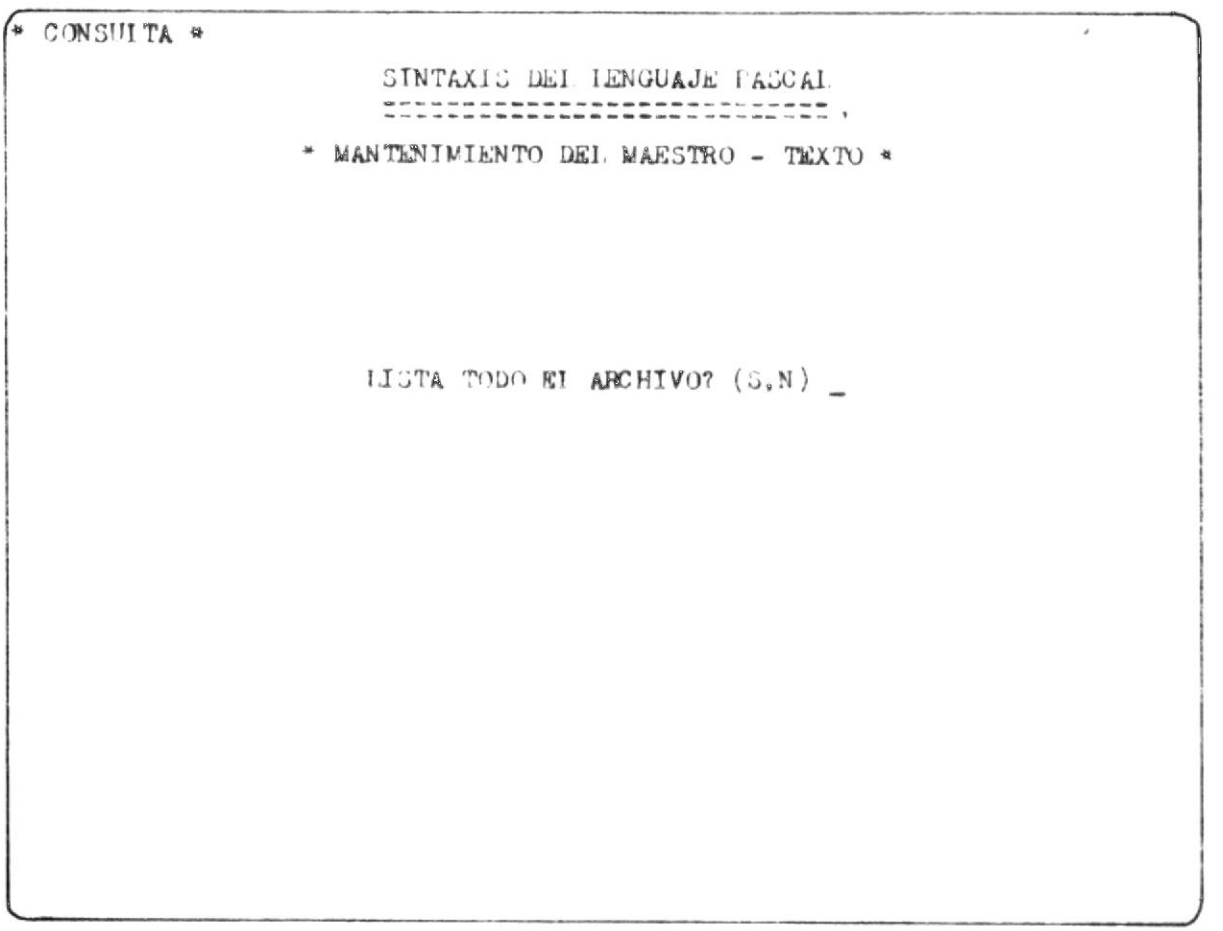


fig. 9 Pregunta si se consulta todo el archivo.

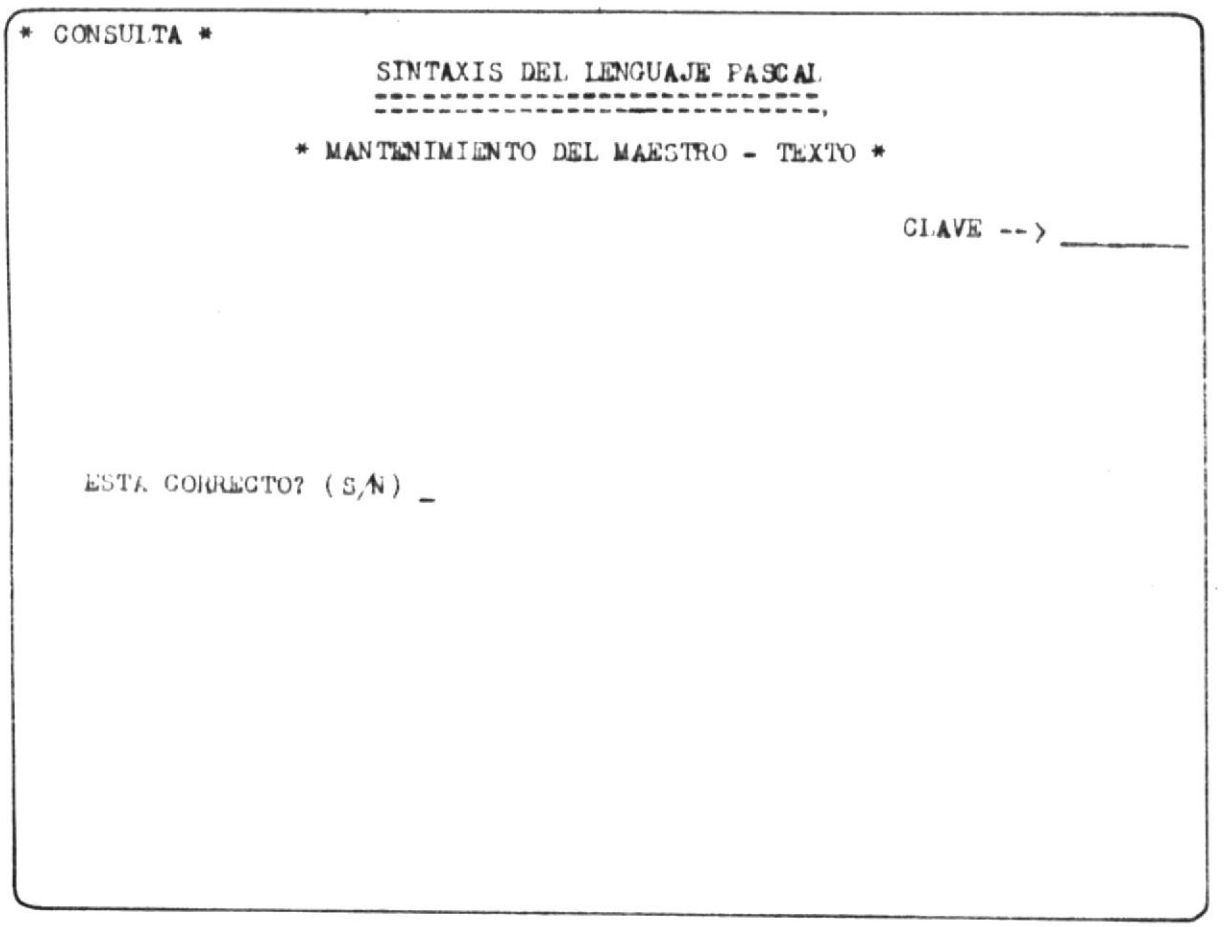


fig. 10 Solicita la clave del registro desde donde se empezará la consulta, si esta no es desde el inicio del archivo.

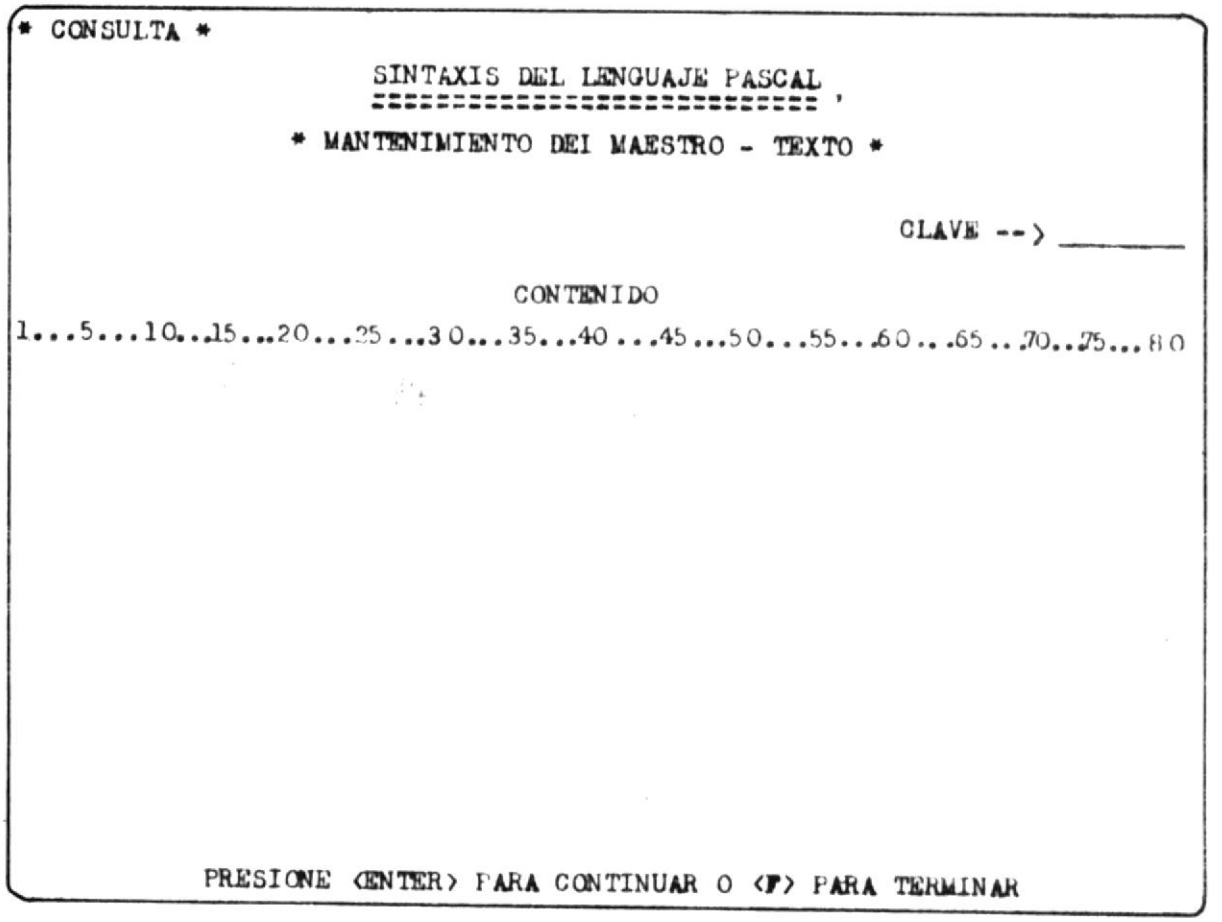
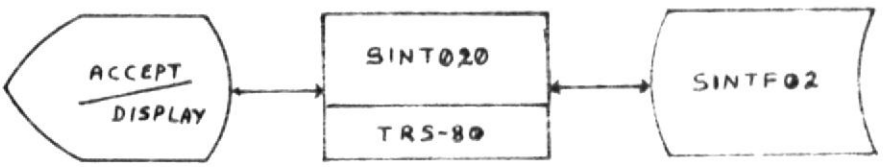


fig. 11 Muestra el contenido de cada registro que se vaya consultando, al igual que su clave.

2.5.2 Programa SINT020

El objetivo de este programa es permitir el ingreso, modificación, eliminación y consulta de la información del archivo que contiene la evaluación correspondiente a la sintaxis del Pascal.

El diagrama de bloque es el siguiente:



El programa utiliza el siguiente archivo:

NOMBRE	MODO DE APERTURA	MODO DE ACCESO
SINTF02	I-O	DYNAMIC

El procedimiento que se lleva a cabo es el siguiente:

Primeramente aparece una pantalla en la que se pide escribir la palabra clave que permita continuar con el proceso (ver fig. # 2). Si la clave es digitada incorrectamente, el programa termina. Caso contrario, en breves instantes se presenta un menú con cinco opciones a elegir (ver fig. # 12).

En rutina INGRESO, se requiere que se escriba la clave y contenido del registro, teniendo opción a corregir la información digitada (ver fig. # 13).

La rutina MODIFICACION impulsa a digitar la clave que accesará al registro a modificarse, contando además con la alternativa de corregir la clave escrita (ver fig. #14 y 15).

Para la rutina ELIMINACION se requerirá que se digite la clave que accesará al registro a ser eliminado. Este proceso también ofrece la oportunidad de corregir la información escrita. (ver fig. # 16 y fig. # 17).

La rutina CONSULTA presenta dos opciones: consultar el archivo desde el inicio o desde un registro específico (ver fig. # 18 y fig. # 19). Para las dos opciones se presentarán los datos consultados, pudiendo terminar el proceso en cualquier momento (ver fig. # 20).

Si se elige la opción FINALIZAR, se cierra el archivo y se procede a terminar el programa.

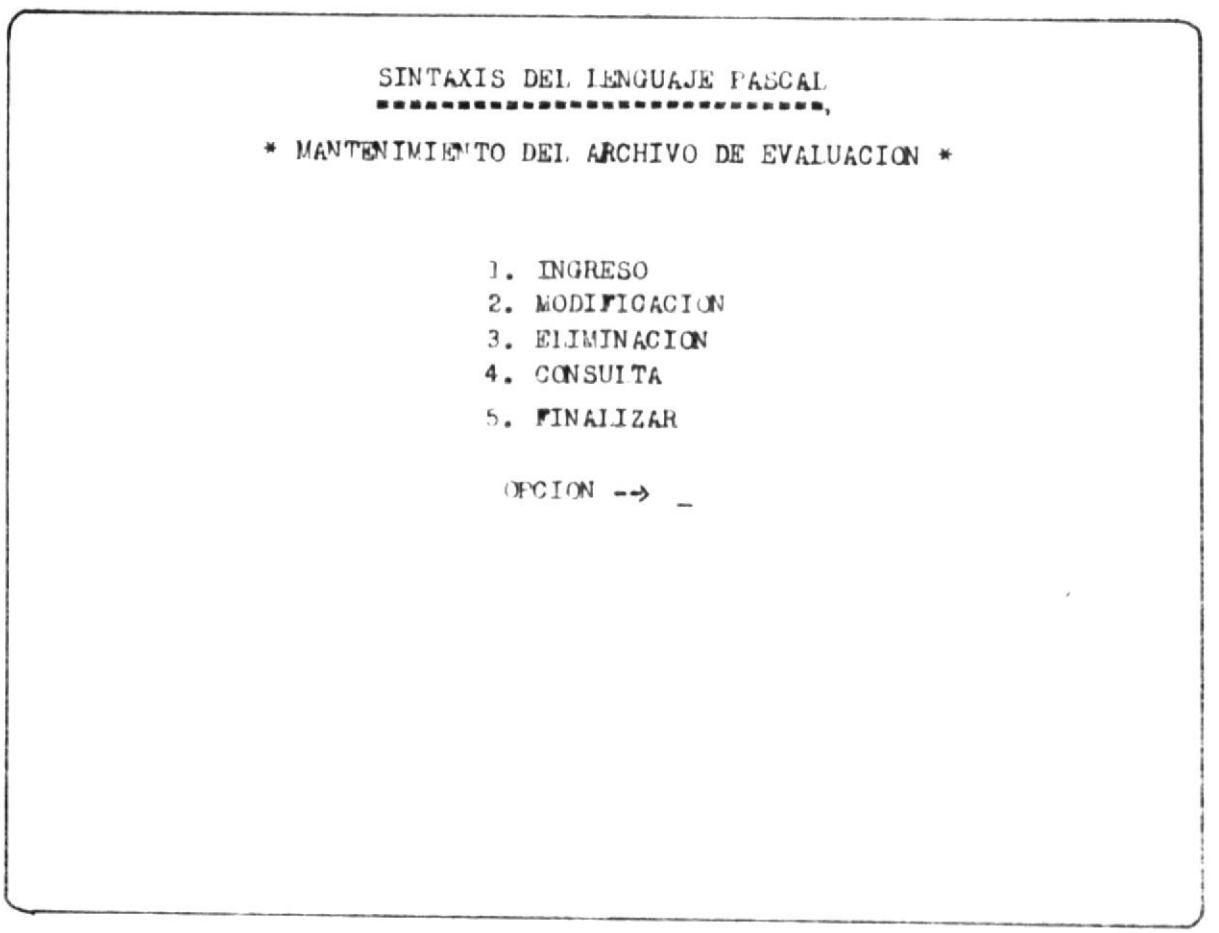


fig. 12 Opciones para el mantenimiento de la evaluación de la sintaxis del Pascal.

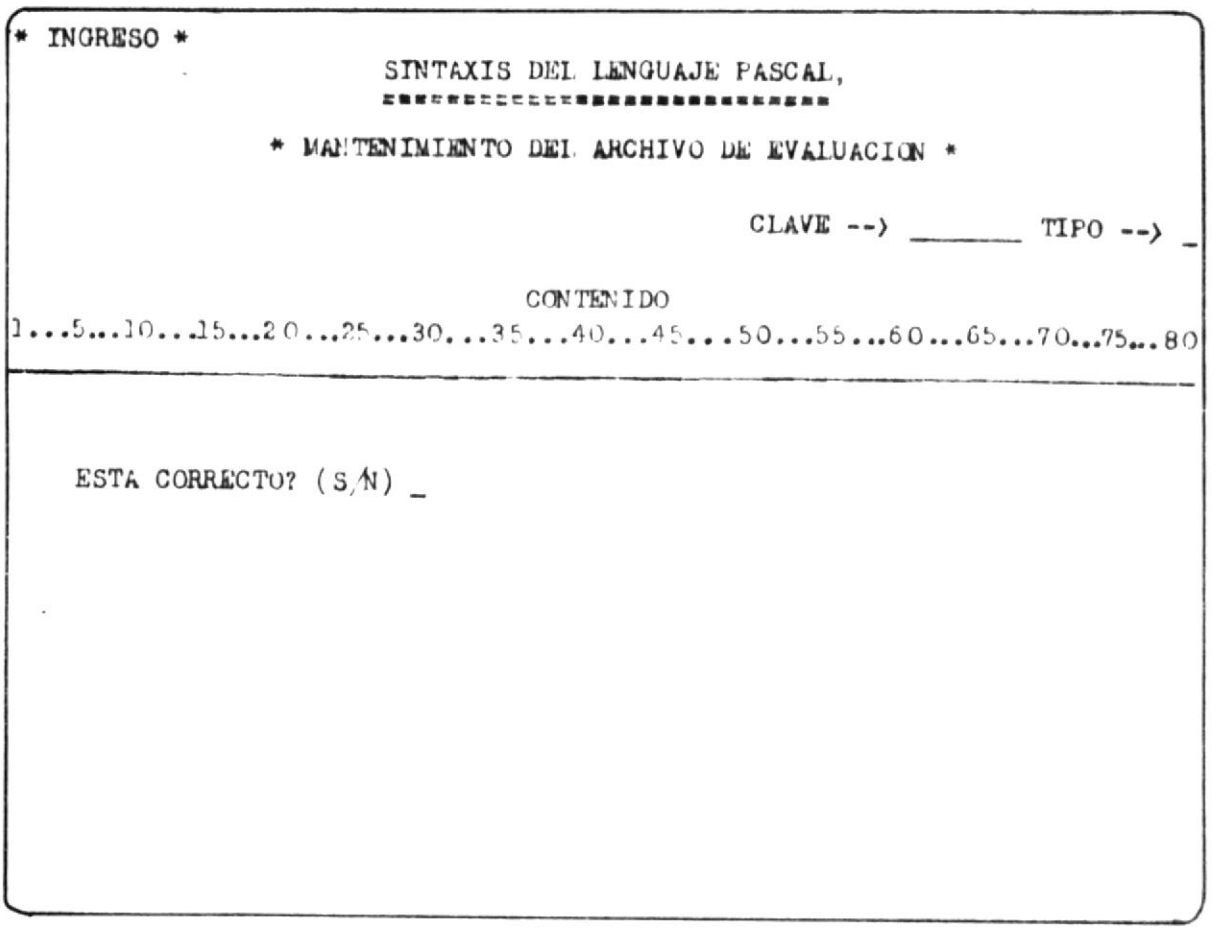


fig. 13 Solicita la información del registro a ingresar.

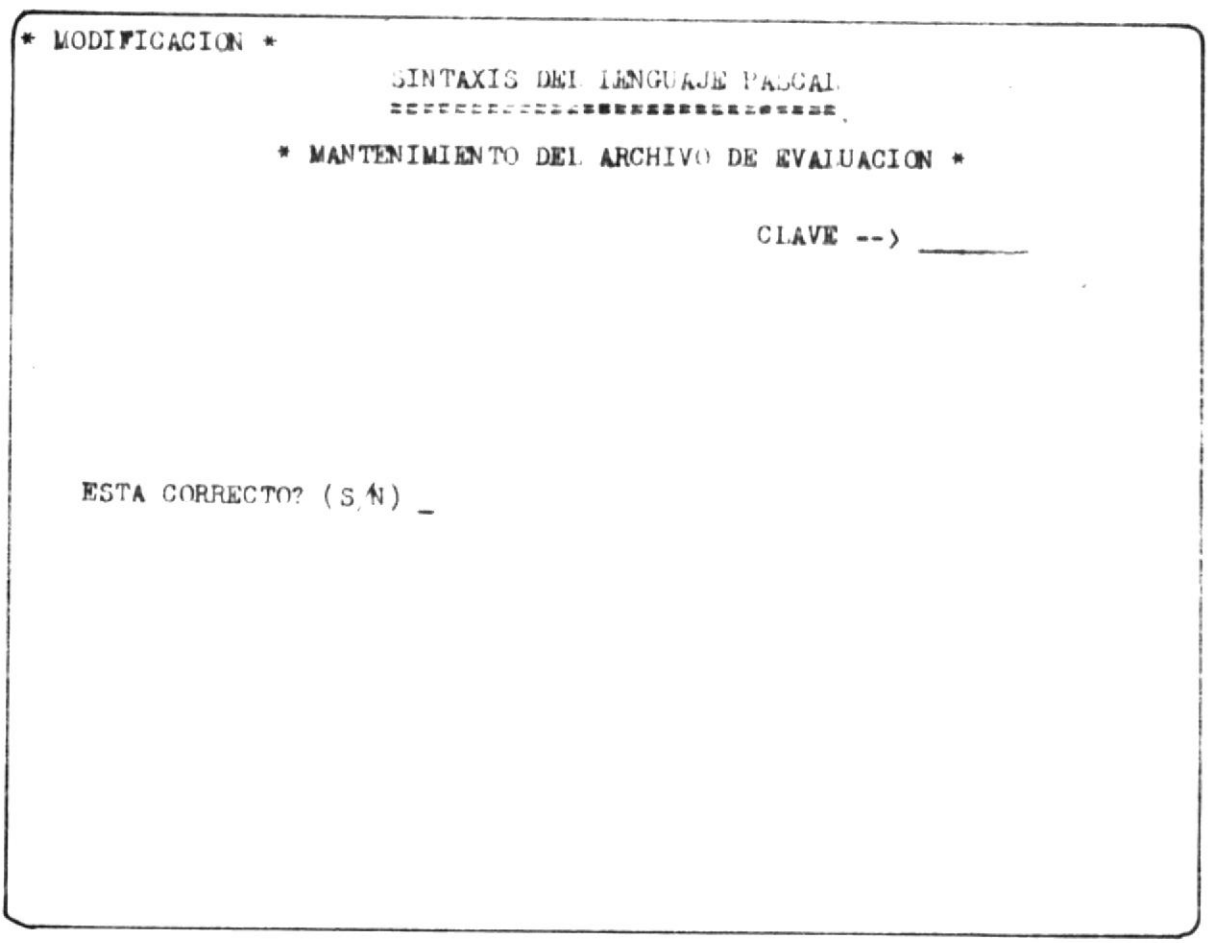


fig. 14 Pide clave del registro que se desea modificar.

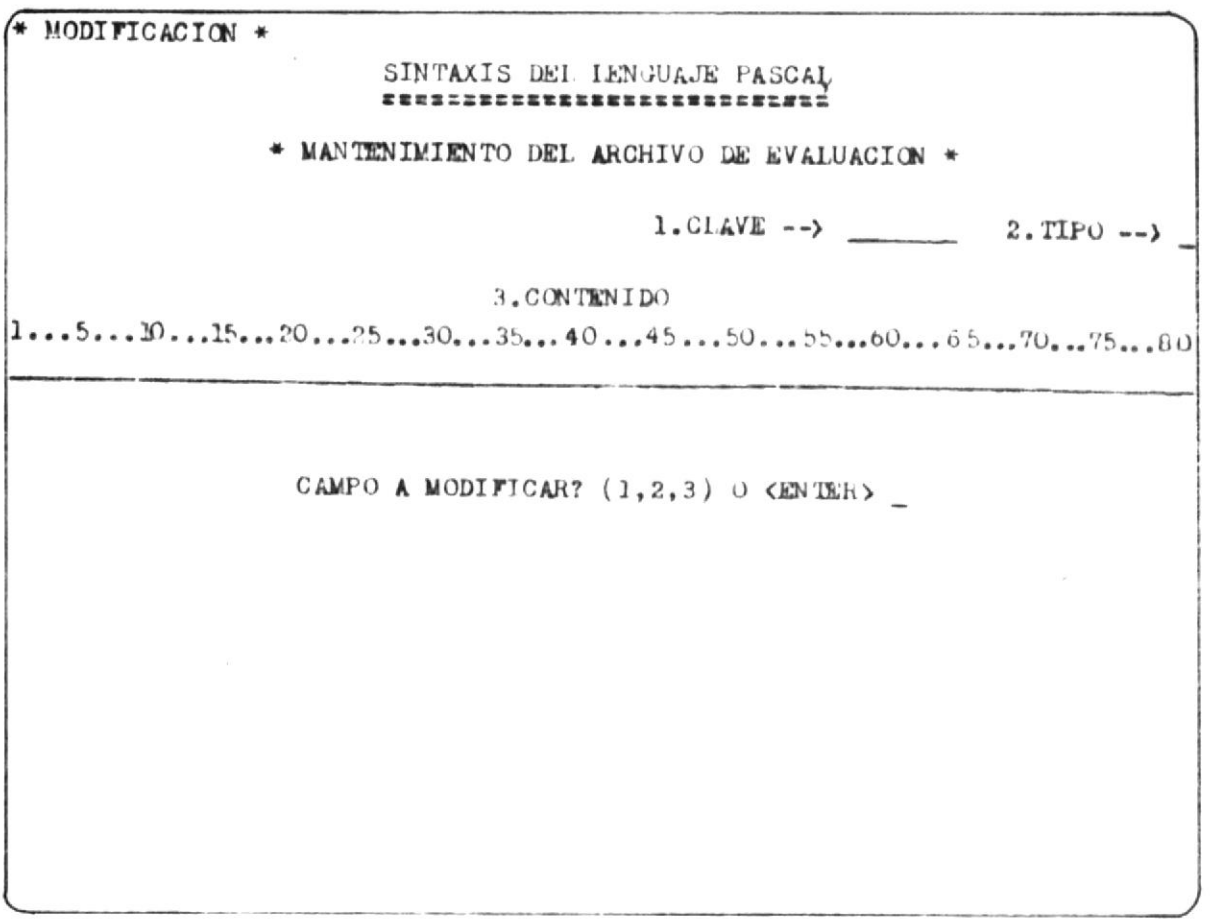


fig. 15 Muestra el contenido del registro a modificarse.

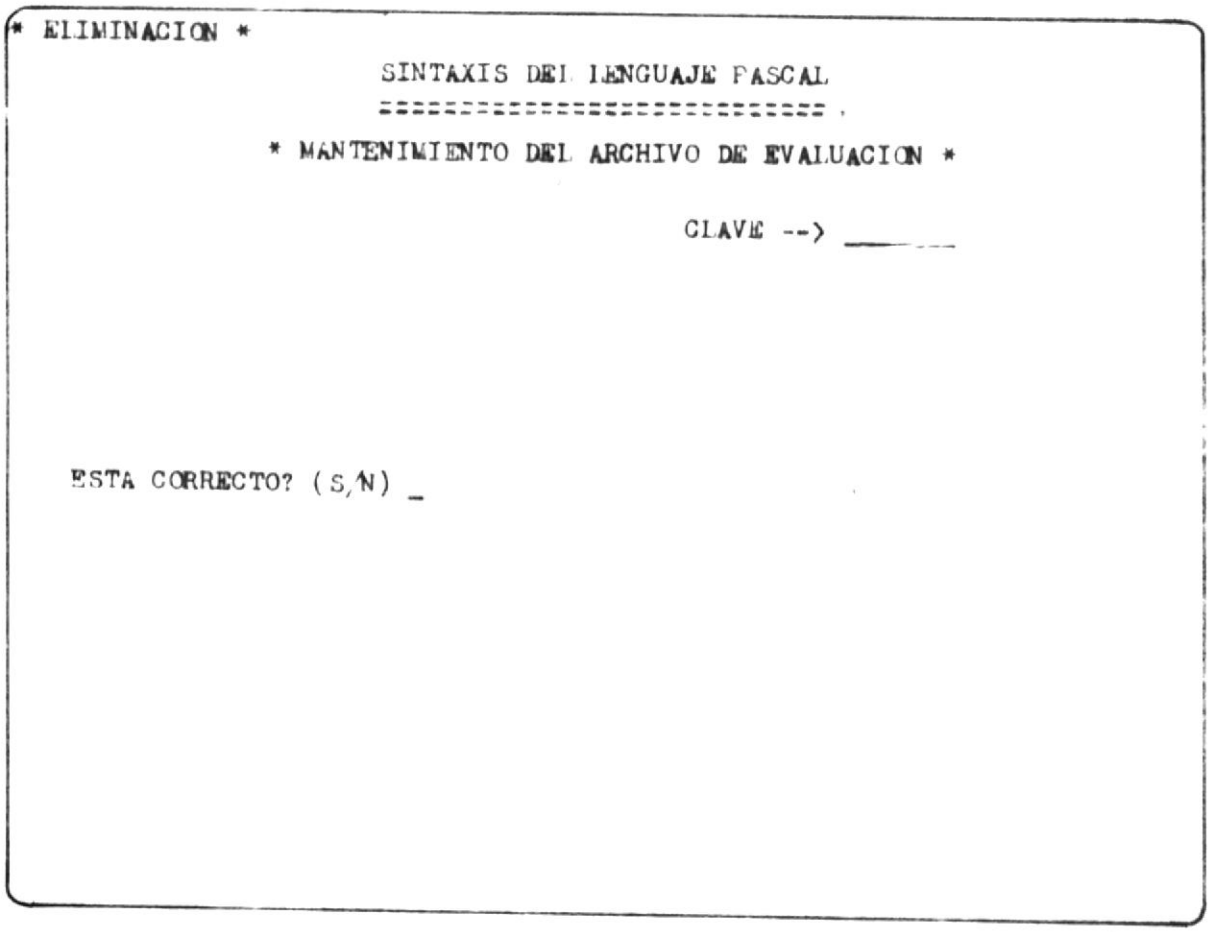


fig. 16 Solicita clave que accesará al registro a eliminarse.

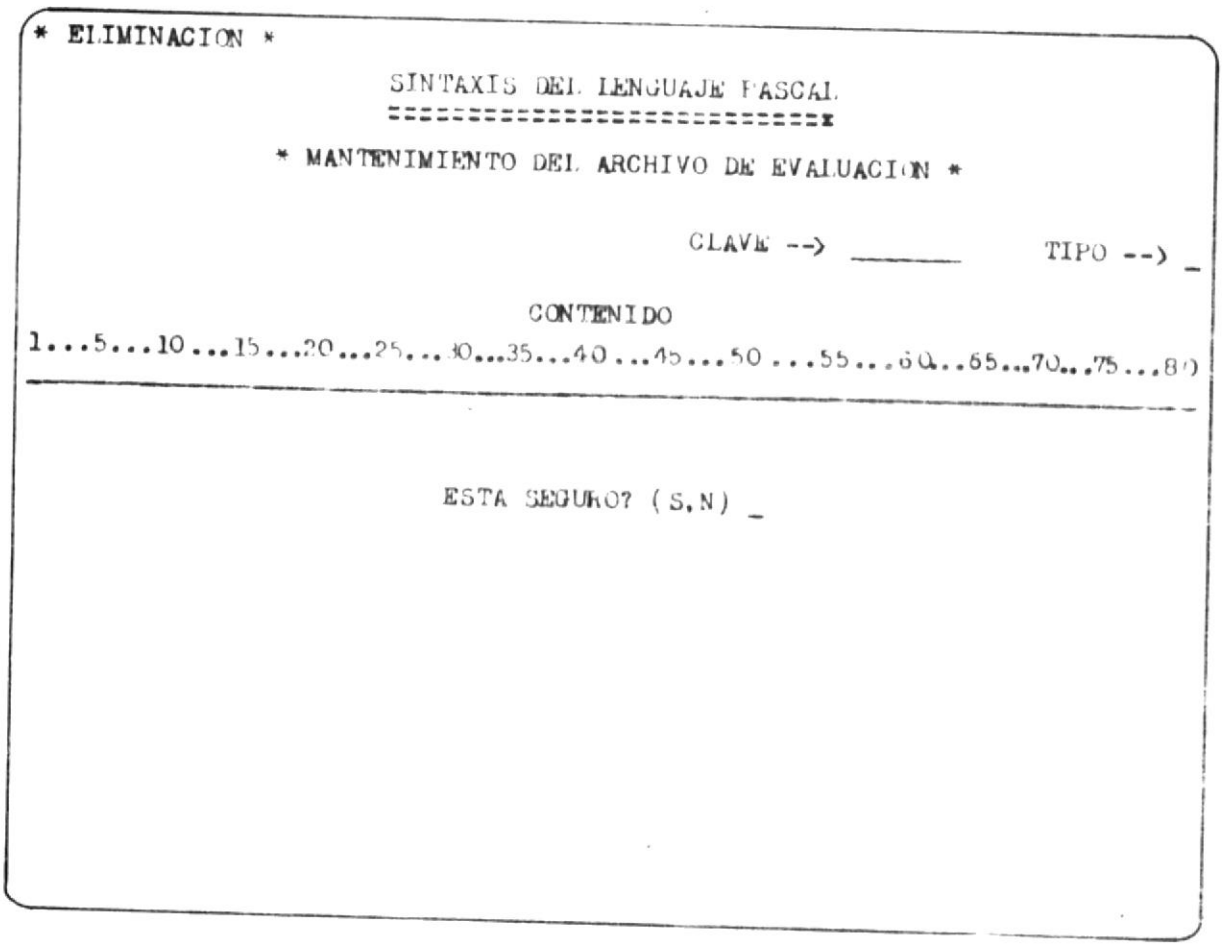


fig. 17 Muestra el contenido del registro a eliminarse.

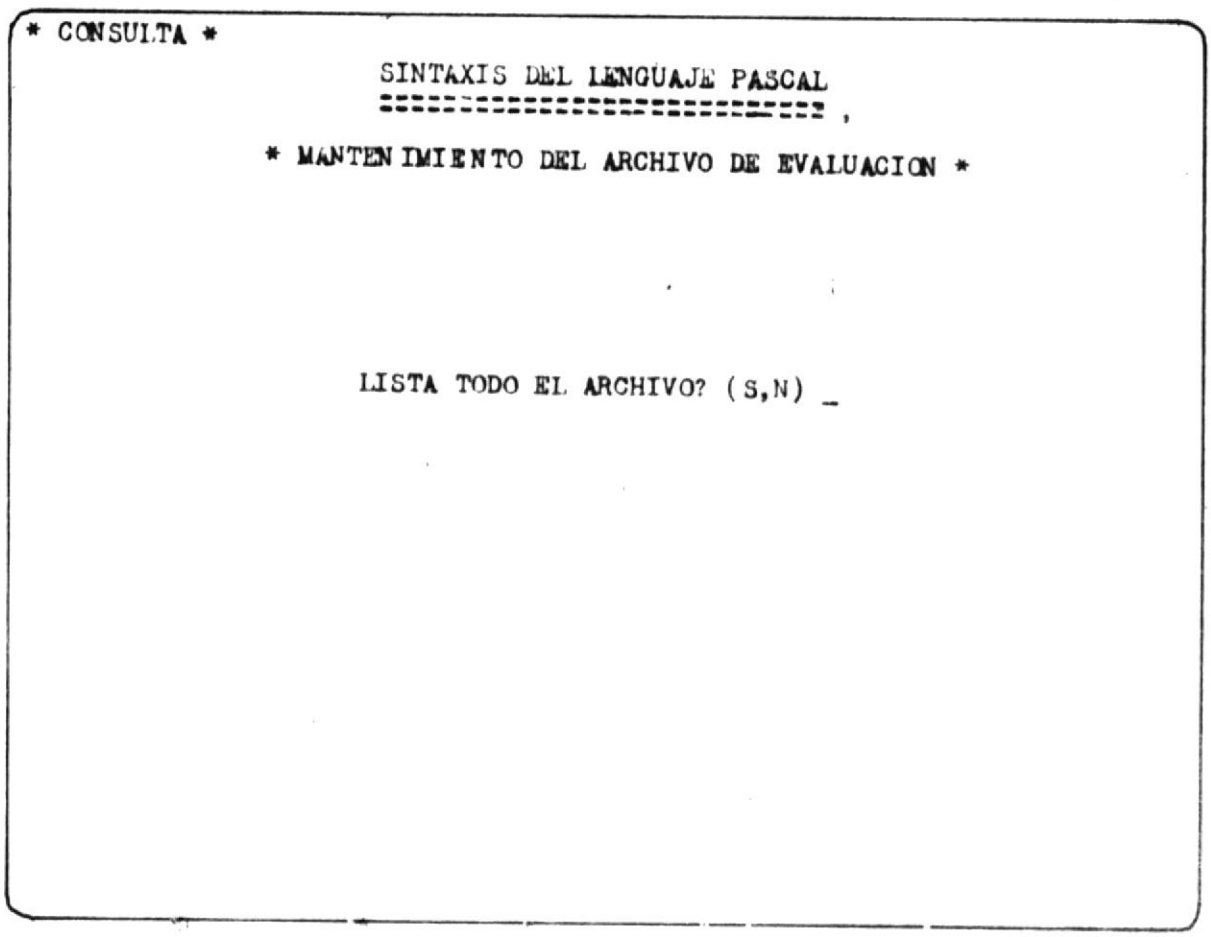


fig. 16 Pregunta si se consulta el archivo desde el inicio.

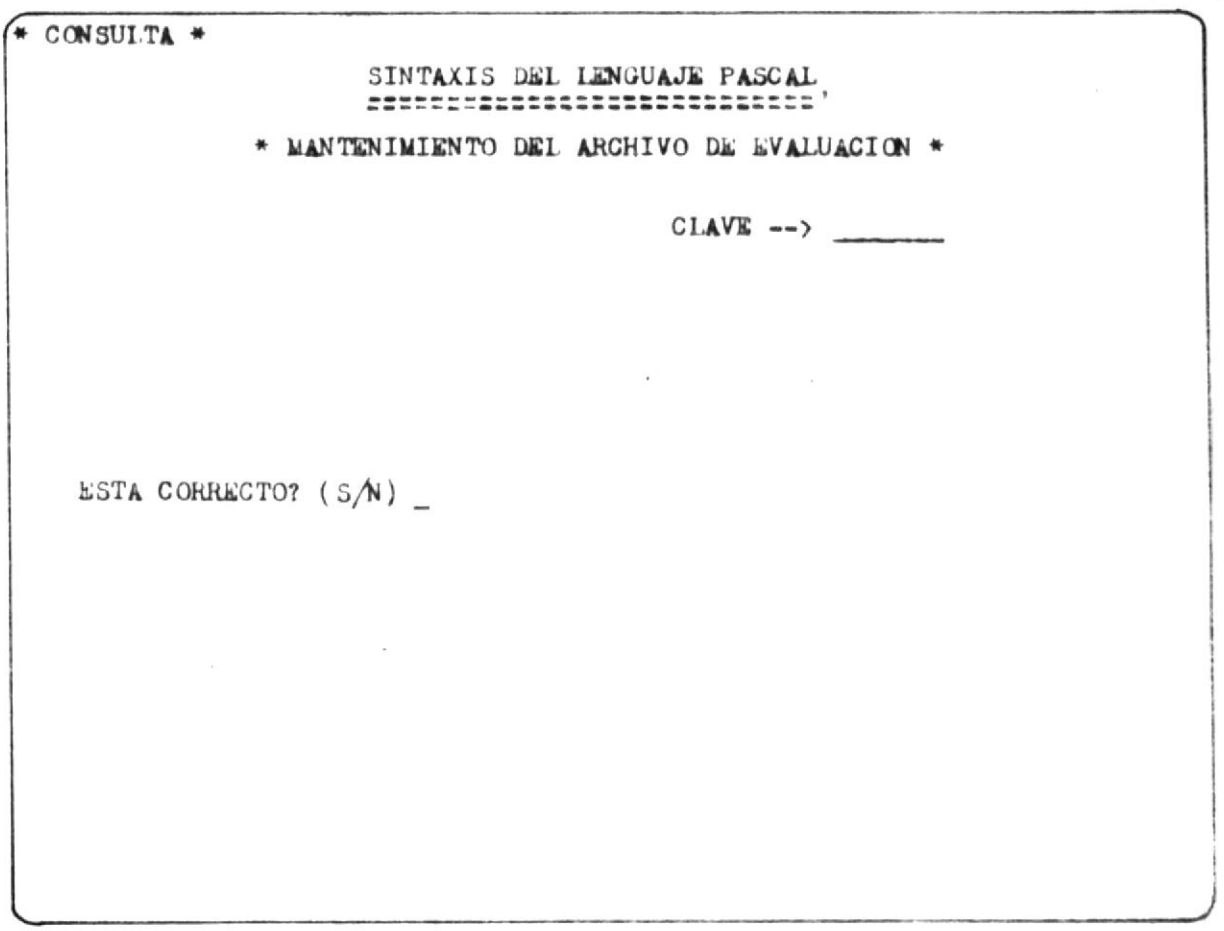


fig. 19 Si el archivo se consulta desde un registro específico, solicita la clave de éste.

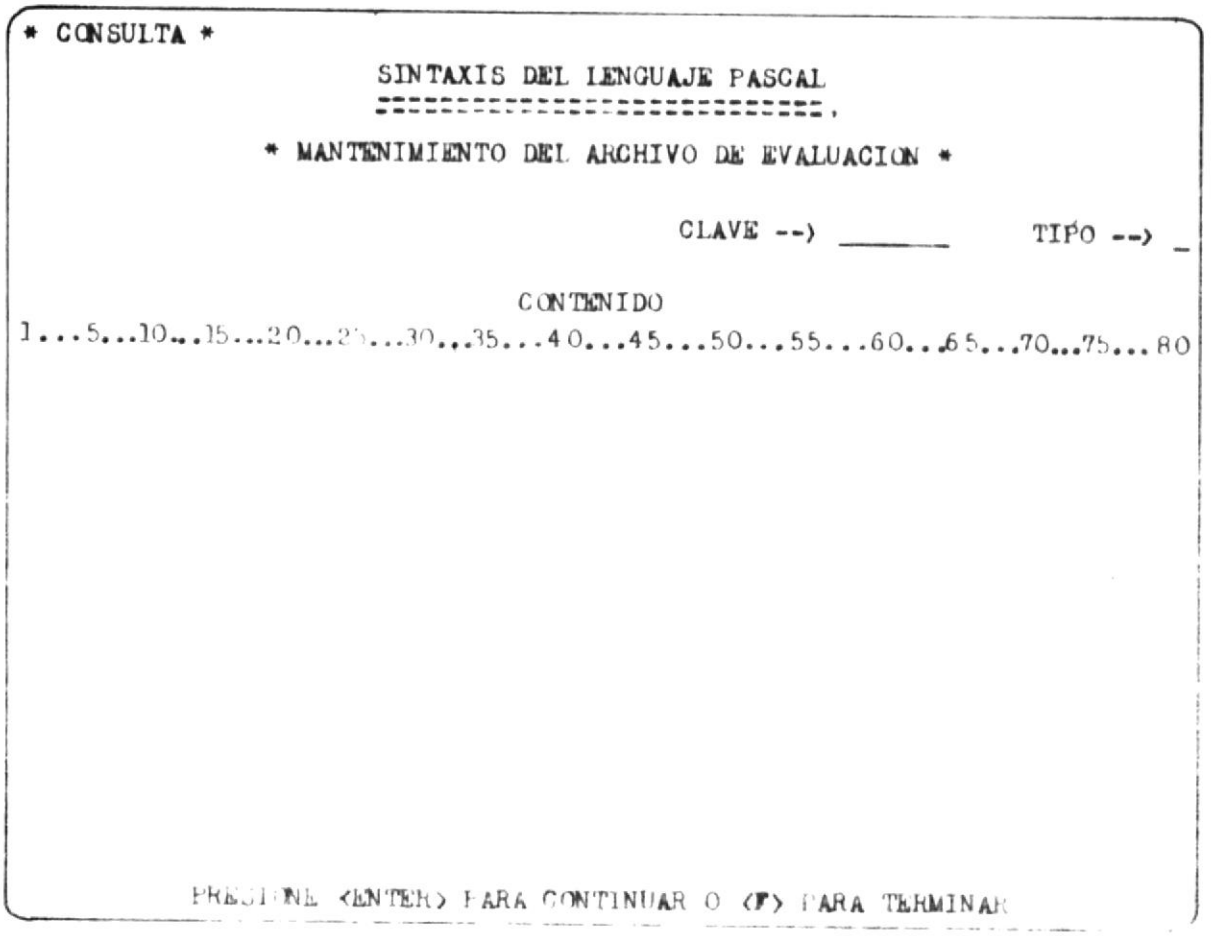
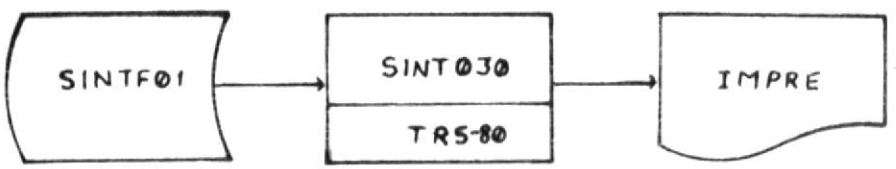


fig. 20 Muestra el contenido de cada registro que se vaya consultando al igual que su clave y tipo.

2.3.3 Programa SINT030

El objetivo de este programa es imprimir la sintaxis del Pascal, desde el archivo SINTF01.

El diagrama de bloque es mostrado a continuación:



El programa utiliza los siguientes archivos:

NOMBRE	MODO DE APERTURA	MODO DE ACCESO
SINTF01	INPUT	DYNAMIC
IMPRE	OUTPUT	

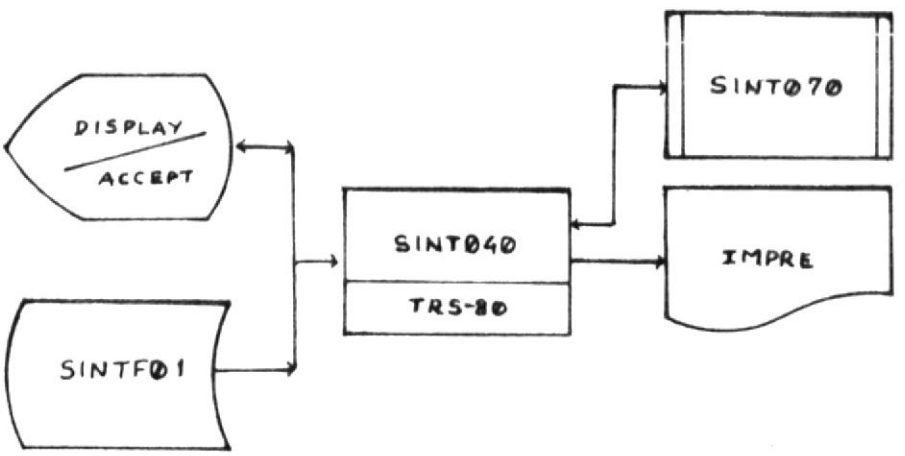
El procedimiento de este programa es leer secuencialmente el archivo SINTF01 e imprimir sólo la parte correspondiente al contenido de la sintaxis de cada registro.

El programa termina cuando se ha detectado el fin de archivo, o cuando la impresora falla.

2.3.4 Programa SINT040

El objetivo de este programa es mostrar la sintaxis del lenguaje Pascal en una forma interactiva, mediante la intervención del usuario.

El diagrama de bloque es el siguiente:



Los archivos que utiliza el programa son:

NOMBRE	MODO DE APERTURA	MODO DE ACCESO
SINTF01	INPUT	DYNAMIC
IMPRE	OUTPUT	

El proceso que se realiza es el siguiente:

Inicialmente se muestra un menú en el que constan los diferentes capítulos que forman el texto de la sintaxis del pascal, además de la opción de terminar el programa (ver fig. # 21).

Si se selecciona cualquiera de los capítulos, a continuación se presentará un menú correspondiente al capítulo seleccionado, el que contendrá las unidades que conforman ese capítulo, y adicionalmente una opción para retornar al menú principal (ver fig. # 22).

Si se elige una de las unidades para repasar, el programa procederá a exhibir la primera pantalla correspondiente a la unidad y capítulo escogidos. Y en la parte inferior de la misma, aparecerán alternativas para poder retroceder, avanzar, evaluar, listar y terminar respecto a la unidad que se está repasando.

Para rutina RETROCEDER, se presentará la pantalla predecesora a la que actualmente se encuentra en exhibición.

En rutina AVANZAR, la pantalla sucesora a la que actualmente se encuentra presentada, será la que se exhiba.

La rutina EVALUAR, enlaza a un subprograma en donde se re

aliza un evaluación sobre el capítulo al cual hace referencia la unidad que se encuentra en exhibición.

Para la rutina LISTAR, el programa generará un listado sobre la unidad que se está repasando.

Si se elige la rutina TERMINAR, se regresará al menú al cual corresponde la unidad de repaso.

Si en el menú del capítulo que se está repasando se escoge la opción cero, se retornará al menú principal.

Cuando la opción TERMINAR SESION es seleccionada en el menú principal, el programa procederá a cerrar los archivos y éste termina.

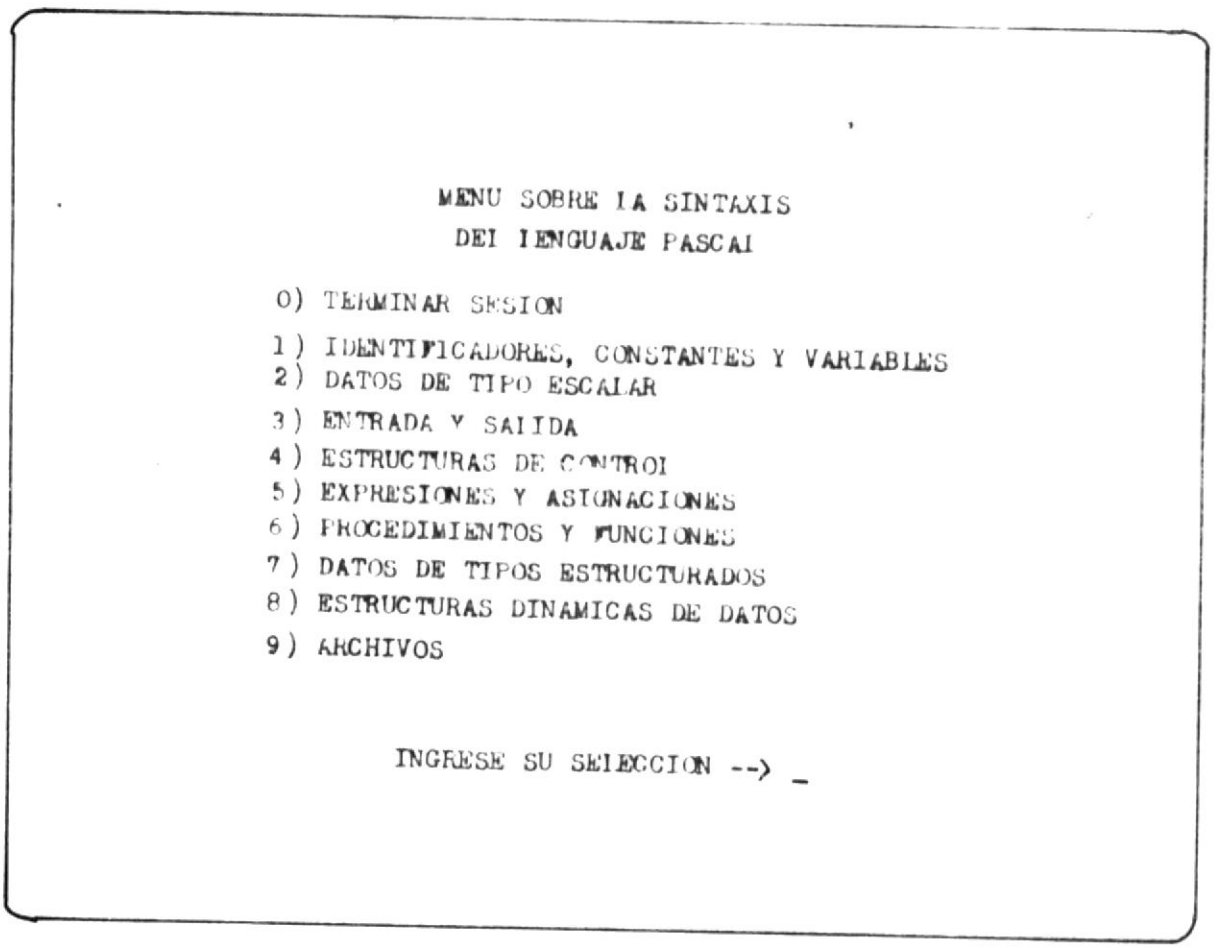


fig. 21 Presenta los capítulos que conforman el texto sobre la sintaxis del Pascal.

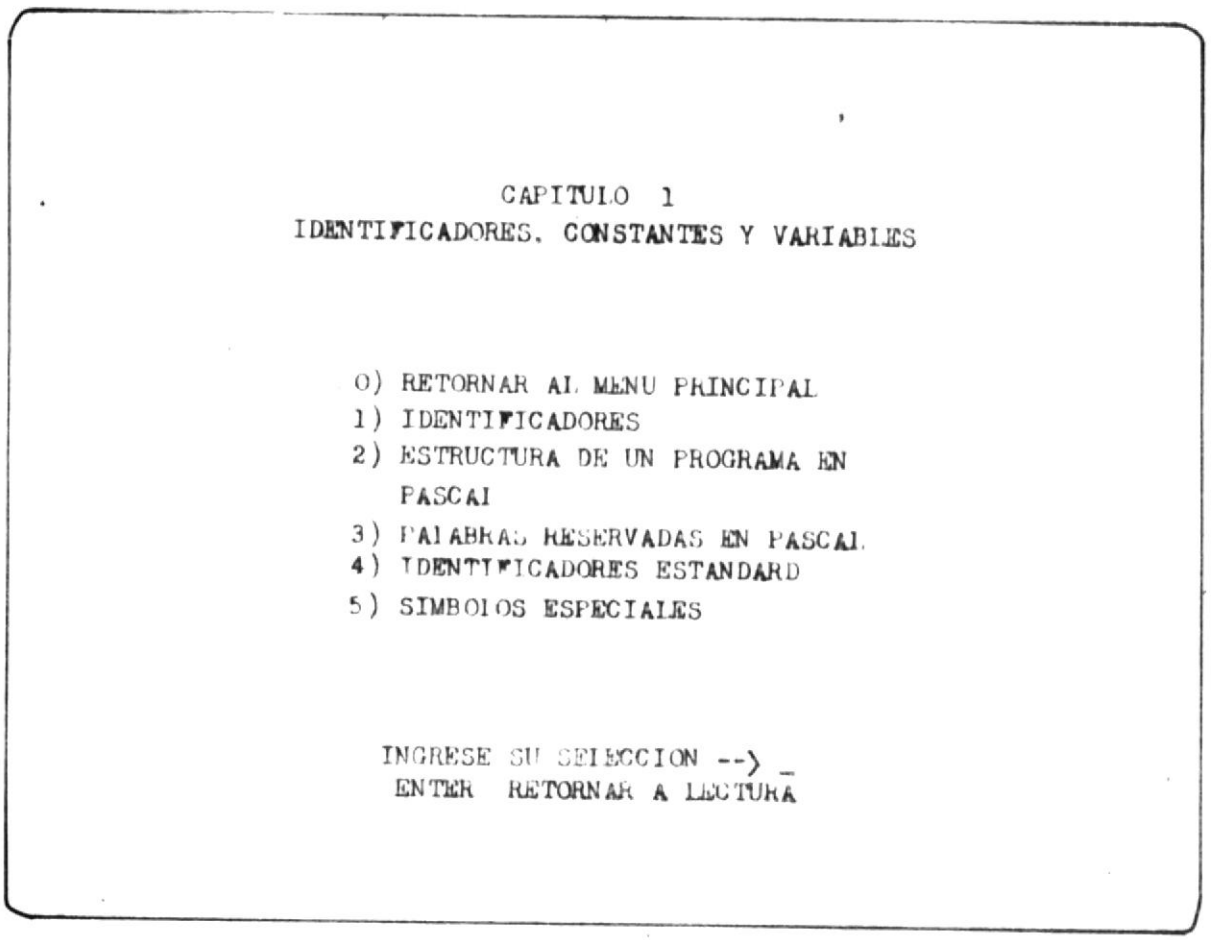
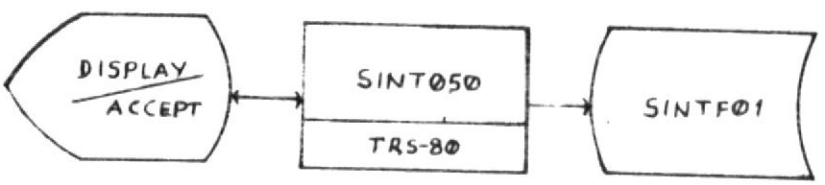


fig. 22 Presenta las unidades que componen el Capítulo 1.

2.3.5 Programa SINT050

El objetivo de este programa es inicializar el área donde se creará el archivo SINTF01, el cual contendrá el texto de la sintaxis del Pascal.

El diagrama de bloque del programa es el siguiente:



El archivo que se utiliza es:

NOMBRE	MODO DE APERTURA	MODO DE ACCESO
SINTF01	OUTPUT	RANDOM

El procedimiento que se lleva a cabo es el siguiente:

En primera instancia se presenta una pantalla en la que

se solicita digitar la palabra clave que permitirá el ingreso a este proceso (ver fig. # 2).

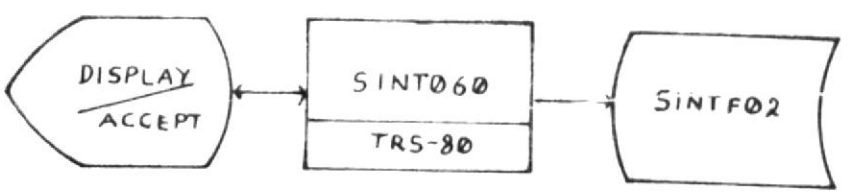
Si la clave está errada, el programa termina. Caso contrario, se procede a inicializar el área del archivo SINTF01 grabando un registro con clave cero.

Al ejecutarse este programa se borra cualquier otro archivo que exista con el mismo nombre.

2.3.6 Programa SINT060

El objetivo de este programa es inicializar el área donde se grabará la información concerniente a la evaluación sobre la sintaxis del Pascal, generándose el archivo SINTF-02.

El diagrama de bloque del programa es el siguiente:



El archivo que utiliza el programa es el siguiente:

NOMBRE	MODO DE APERTURA	MODO DE ACCESO
SINTFO2	OUTPUT	RANDOM

El procedimiento a realizarse es el siguiente:

Primeramente se emite una pantalla en donde se solicita la palabra clave que permitirá continuar con el proceso de inicialización (ver fig. # 2).

Si la clave es errada, el programa termina. Caso contrario, se procede a inicializar el área donde se grabará la información sobre la evaluación que se realiza respecto a la sintaxis del Pascal. Al generarse el archivo SINTFO2, se grabará un registro con clave cero.

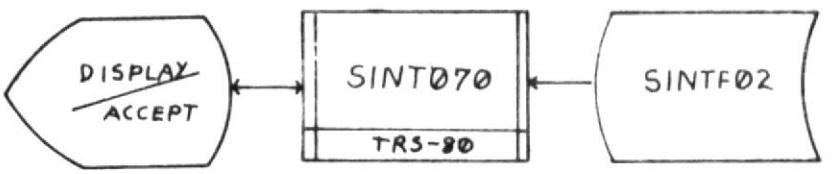
Al ejecutarse este programa se borra cualquier otro archivo que exista con el mismo nombre.

2.4 Subprogramas

2.4.1 Subprograma SINTO70

Este subprograma tiene como función principal emitir preguntas concernientes al capítulo que está repasando el usuario, sobre la sintaxis del Pascal; y además realizar una evaluación sobre las respuestas dadas por éste.

El diagrama de bloque del subprograma es el siguiente:



El archivo que se utiliza en este subprograma es el que a continuación se muestra:

NOMBRE	MODO DE APERTURA	MODO DE ACCESO
SINTF02	INPUT	DYNAMIC

El proceso que se efectúa es como sigue:

Una vez que este subprograma es llamado por el programa que genera la sintaxis del Pascal (ver programa SINT040), recibiendo de este último el número del capítulo que se va a evaluar, se procede a emitir la primera pantalla de preguntas correspondiente a este capítulo, presentando además alternativas para poder retroceder, avanzar, contestar, evaluar y terminar la evaluación.

La opción RETROCEDER, presenta la pantalla anterior a la que se encuentra en exhibición.

La opción AVANZAR, en cambio emitirá la pantalla posterior a la que se encuentra en pantalla.

La opción CONTESTAR, da la oportunidad de responder a la pregunta que se encuentra en pantalla.

En la opción EVALUAR, se emite la evaluación que se realizó a las respuestas dadas por el usuario, presentando además la calificación respectiva, y dando la oportunidad de corregir alguna respuesta (ver fig. # 23 y fig. # 24).

Para la opción TERMINAR, se procede a cerrar el archivo y se termina el proceso pasando el control al programa que hizo referencia a este subprograma.

EVALUACION CORRESPONDIENTE AL CAPITULO 1

C = RESPUESTA CORRECTA I = RESPUESTA INCORRECTA

PREGUNTAS \ RESPUESTAS	A	B	C	D	E	C / I
1	-	X	-	-	-	C
2	X	-	-	-	-	I
3	-	-	-	X	-	I
4	-	-	X	-	-	I
5	-	-	-	-	X	C

PUNTAJE OBTENIDO --> 4/10

DESEA CORREGIR ALGUNA PREGUNTA? (S,N) _

fig. 23 Muestra la evaluación que se realiza al capítulo que se está repasando.

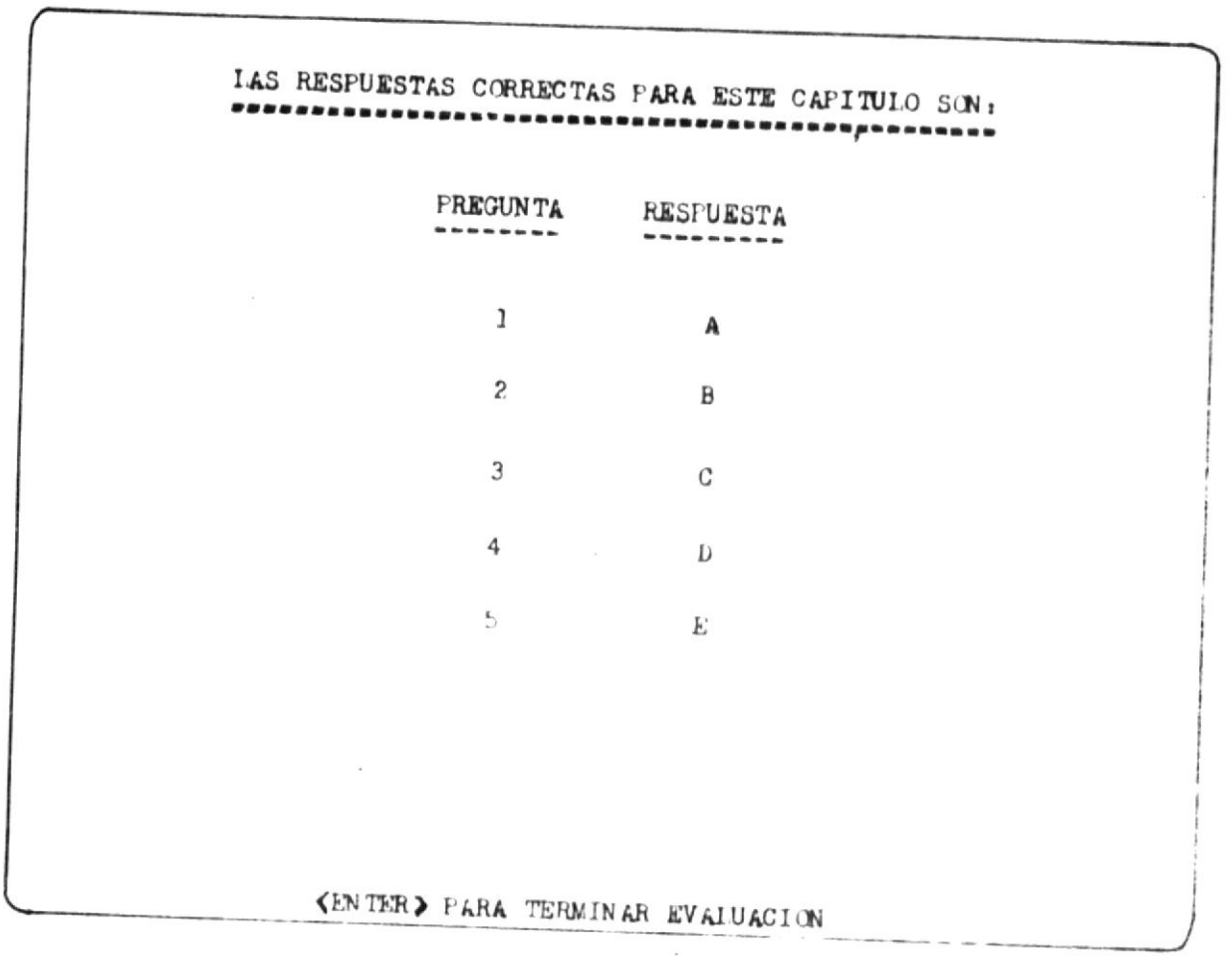


fig. 24 Muestra las respuestas correctas sobre el capítulo que se está repasando, siempre que se lo solicite.

2.5 Operación del Sistema

Antes de llevar a cabo cualquiera de los procesos del Sistema, usted:

1. Necesita tener a disposición los dos diskettes que conforman el Sistema: el diskette que contiene los archivos, nombrado ARCHIVOS; y el diskette que contiene los programas, llamado PROGRAMAS.

2. Encienda el Computador, si está apagado, asegurándose de que no exista ningún diskette en ninguno de los dispositivos y que todos los periféricos estén en orden (ver Encendido y apagado del Computador, Capítulo 1).

3. Si el Computador se encuentra encendido, reposiciónelo presionando el interruptor RESET (ver Utilizando el interruptor RESET, Capítulo 1).

4. Cargue el Sistema Operativo del TRS-80, colocando el diskette de PROGRAMAS en el dispositivo que está incorporado a la consola de pantalla e inicialice la fecha y/o la hora en el Computador (ver Cargando el Sistema Operativo del TRS-80, Capítulo 1).

5. Inserte el diskette de ARCHIVOS en un dispositivo de la unidad de Expansión del Sistema.

6. Coloque el teclado en el modo de mayúsculas, presionando la tecla CAPS, pues todos los comando del Sistema Operativo TRS DOS se deben digitar en letras mayúsculas y desde el inicio de línea.

7. Para ejecutar un programa del Sistema de Enseñanza Interactiva de la Sintaxis del Pascal, usted primaramente debe escribir RUNCOBOL nombre del programa, presionando luego la tecla ENTER.

8. Para actualizar la información de un programa fuente, en la línea de comandos del TRSDOS digite EDIT y luego presione ENTER. Ya en el editor cargue el programa y modifíquelo (para mayor información ver Guía del usuario del TRS80 para lenguaje Cobol, el mismo que se encuentra disponible en la sala de Micros del Instituto de Matemáticas de la ESPOL).

9. Después de modificar un miembro fuente debe compilarlo, para esto digite en la línea de comandos: RSCOBOL nombre del programa y luego presione ENTER (para mayor información ver también Guía del usuario del TRS-80 para lenguaje Cobol). Si en la compilación existen errores, regrese al paso 8.

Los dos últimos pasos sólo son permitidos para aquellas personas autorizadas para modificar un programa del Sistema de Enseñanza Interactiva de la Sintaxis del Pascal.

CONCLUSIONES Y RECOMENDACIONES

De todo lo que se ha expuesto en los capítulos anteriores, podemos concluir que:

1. Al ser éste un Sistema conversacional, mediante el cual se explica la estructura de cada una de las sentencias que forman el Pascal, el usuario obligatoriamente debe intervenir en la ejecución del mismo, dando respuestas correctas a las preguntas que se formulen en el Sistema para su completo éxito.

2. El Sistema ofrece la ventaja de que el archivo que contiene la sintaxis del Pascal puede ser modificado, esto es, que el archivo puede ampliarse agregándosele información que se considere de importancia en el futuro.

3. Al igual que el archivo maestro, el archivo que contiene la evaluación de la sintaxis, también puede ser modificado.

4. El usuario puede listar la sintaxis del Pascal en una forma completa o bien mandando a imprimir su contenido por unidades, las mismas que pueden ser excluyentes entre sí, es decir, pueden o no pertenecer a un mismo capítulo.

Este Sistema va recomendado para aquellas personas que tienen

deseo de aprender el lenguaje Pascal, así como para aquellos programadores que se inician en el estudio de este lenguaje, pues esta tesis es una guía útil y práctica que enfoca las principales características de este importante lenguaje.

Se recomienda además que para la ejecución de aquellos programas que necesitan de una clave para su acceso, ésta sea solicitada al Director de Tesis.

A P E N D I C E S

APENDICE A

T E X T O S O B R E L A
S I N T A X I S D E L P A S C A L

C O N T E N I D O

Cap. 1 IDENTIFICADORES, CONSTANTES Y VARIABLES

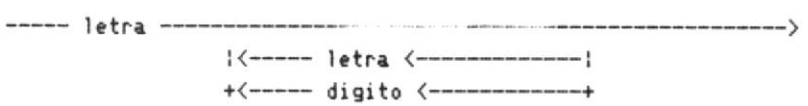
IDENTIFICADORES

Usados para denotar datos y acciones en un programa Pascal. Un identificador Pascal legal:

- Puede contener cualquier combinacion de letras (A - Z), y digitos (0 -9).
- Debe siempre comenzar con un letra (A - Z).

El diagrama de sintaxis de un identificador es:

identificador



Un identificador legal se obtiene siguiendo las flechas de izquierda a derecha del diagrama. Los saltos y lazos pueden ser seguidos si las flechas estan en la direccion correcta.

Identificadores legales en Pascal son: fecha, mt1, sueldo.

Identificadores ilegales en Pascal son: 25letras, dia-actual, linea numero.

El identificador '25letras' es ilegal porque el primer caracter no es una letra. Los otros dos son ilegales porque contienen un caracter el cual no es una letra o un digito. Los identificadores '25letras' y 'dia-actual' seran perfectamente reconocidos por el compilador como ilegales, no asi el identificador 'linea numero' porque el compilador trata un espacio como un separador. El por lo tanto intentara interpretar al identificador como dos identificadores consecutivos 'linea' y 'numero' y en la mayoria de los casos decide que la linea del programa contiene un error de sintaxis, ya que existe un identificador ilegal.

El diagrama de sintaxis no coloca restriccion en el numero de caracteres en el identificador. Esto es una caracteristica deliberada del Pascal y permite que a los datos y acciones les sean dados nombres significativos a fin de que un lector humano comprenda las funciones de una actividad particular o el proposito de algun dato particular.

En la practica la implementacion del Pascal no impone un limite en el numero de caracteres que son significantes. El compilador UCSD Pascal, solamente usa los primeros [illegible] de un identificador para distinguir [illegible] identificadores. Esto significa que tanto 'distancia1' y 'distancia2', por ejemplo, no pueden ser usados como identificadores en el mismo programa.

ESTRUCTURA DE UN PROGRAMA EN PASCAL

La estructura completa de un programa Pascal es definida en el diagrama de sintaxis de un programa:

```
programa
--------

    -----> PROGRAM ---> identificador ---> ; ---> bloque ---> . ----->
```

El bloque es la parte principal del programa y debe contener una declaracion de los datos a ser usados, seguidos por las acciones a ser ejecutadas usando los datos. Por ejemplo:

```
    PROGRAM roldepago;
            | declaracion de datos ej.: horas, tarifa, bruto, neto, etc,
    bloque  | descripcion de acciones ej.: leer las horas trabajadas y la tarifa
            | fa de pago, calcular el sueldo bruto y el sueldo neto, etc.
```

El diagrama de sintaxis de un bloque es mostrado abajo y consiste de las declaraciones de datos seguidos del cuerpo del programa, el cual contiene las acciones que abarca:

```
bloque
------

    --> dec. const. --> dec. tipos --> dec. var. --> dec. proc. --> cuerpo -->
```

DECLARANDO LOS DATOS.- Los datos usados en un programa son declarados usando las declaraciones CONST, TYPE y VAR. Los datos cuyos valores no son cambiados por las acciones del programa son declarados primero usando la declaracion CONST. Esta tiene el siguiente diagrama de sintaxis:

```
declaracion CONST
-----------------

    -----> CONST ------> identificador --> = --> constante --> ; --------->
       |           +---------------------<-----------------------+   |
       +----------------------------->-------------------------------+
```

Las reglas para formar una constante seran dadas mas adelante, pero como un ejemplo del uso de CONST en un programa Pascal, a los identificadores uno, dos y tres les seran asignados valores constantes de 1, 2, y 3 respectivamente por:

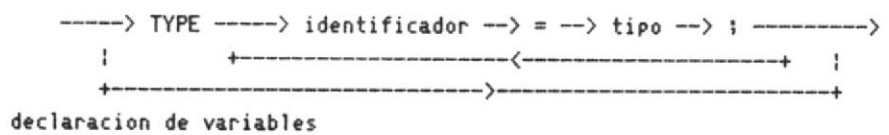
```
        CONST uno  = 1;

              dos  = 2;
              tres = 3;
```

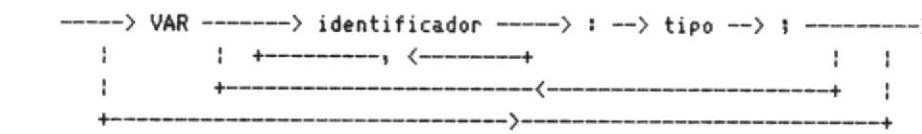
Puede verse, siguiendo el camino mas interno del diagrama de sintaxis previo que la parte de declaracion de constantes del bloque de Pascal puede estar vacio Asi, CONST aparecera solo en un programa Pascal que requiera algun dato constante.

La declaracion VAR, es usada para declarar datos cuyos valores son cambiados por las acciones del programa. Cuando estas variables son declaradas, el rango de valores que ellas pueden tomar deben tambien ser declarados. Estos son definidos por la declaracion TYPE. Los diagramas de sintaxis de declaracion de tipos y variables son:

declaracion de tipos



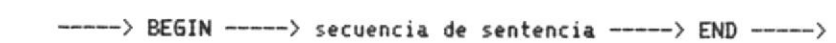
declaracion de variables



Como en el caso de CONST, TYPE y VAR solo aparecen en un programa Pascal cuando sea necesario definir tipos de datos o variables.

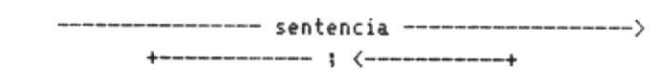
DECLARANDO LAS ACCIONES.- Las acciones son descritas en el cuerpo del programa que tiene el diagrama de sintaxis:

cuerpo



donde la secuencia de sentencia es definida por:

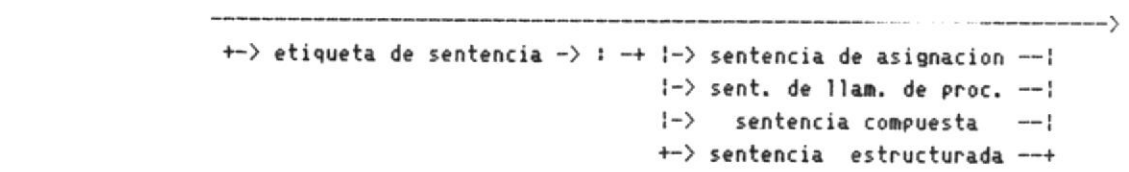
secuencia de sentencia



Debe notarse que el punto y coma es un separador de sentencia y no un terminador de sentencia. Un punto y coma no es requerido entre la ultima sentencia y el END.

Cuatro clases principales de sentencias, las cuales representan alguna forma de accion, existen en Pascal como puede ser visto desde el siguiente diagrama de sintaxis de una sentencia:

sentencia



Programa ejemplo:

```
PROGRAM sumardiez;
{ Suma los diez primeros enteros }

CONST
  numeroelementos = 10;

VAR
  suma, proximo : integer;

BEGIN
  suma := 0;
  FOR proximo TO numeroelementos DO
    suma := suma + proximo;
  writeln('suma =', suma)
END.
```

PALABRAS RESERVADAS EN PASCAL

Algunos identificadores legales aparentemente no son permitidos, puesto que ellos tienen significados especiales dentro de un programa Pascal. Pascal tiene 33 de estas 'palabras reservadas'. Ellas seran dadas en la etapa apropiada en el transcurso de la lectura, pero seran listadas a continuacion como referencia:

AND	END	NOT	SET
ARRAY	FILE	OF	THEN
BEGIN	FOR	OR	TO
CASE	FUNCTION	PACKED	TYPE
CONST	IF	PROCEDURE	UNTIL
DIV	IN	PROGRAM	VAR
DO	MOD	RECORD	WHILE
DOWNTO	NIL	REPEAT	WITH

ELSE

Las palabras reservadas seran escritas en letras mayusculas a traves del curso de la lectura a fin de distinguirlas de los identificadores ordinarios.

Estas deben aparecer en un programa Pascal sin espacios entre los caracteres por ejemplo, 'DOWN TO' sera interpretado por el compilador como dos identificadores 'DOWN' y 'TO'.

IDENTIFICADORES ESTANDARD

Ademas de las palabras reservadas, ciertos identificadores son predefinidos. Estos son llamados 'identificadores estandard' y, asi como las palabras reservadas seran discutidos en la parte relevante del curso.

Los identificadores estandard son:

Constantes:
false, true, maxint

Tipos:
integer, boolean, real, char, string, text, interactive

Archivos del sistema:
input, output, keyboard

Funciones:
abs, atan, chr, concat, copy, cos, eof, eoln, exp, ioresult, length, log, ln, odd, ord, pos, pred, pwroften, round, sin, sqr, succ, trunc

Procedimientos:
delete, get, insert, new, put, read, readln, reset, rewrite, seek, write, writeln

A diferencia de las palabras reservadas, el programador puede redefinir los identificadores y el nuevo significado sera usado en el programa.

SIMBOLOS ESPECIALES

De la sintaxis de un identificador, quedo claro que letras y digitos son caracteres validos en Pascal y por tanto ellos deben ser parte del vocabulario del lenguaje Pascal. El vocabulario completo consiste de las letras A a la Z, de los digitos del 0 al 9, de las palabras reservadas mencionadas anteriormente y de los siguientes simbolos especiales:

;	+	:	(	=
.	-	,	)	>
	*	'	[	>=
	/	↑	]	<>
		..	{	<=
		:=	}	<

El significado y uso de estos simbolos especiales seran descritos cuando sea necesario en el transcurso de la lectura. Como en el caso de identificadores y palabras reservadas, espacios no deben aparecer dentro de un simbolo especial. El simbolo < > por ejemplo, podria ser interpretado por el compilador como dos simbolos < y >. No es necesario tener algun espacio entre los simbolos especiales y los identificadores y palabras reservadas.

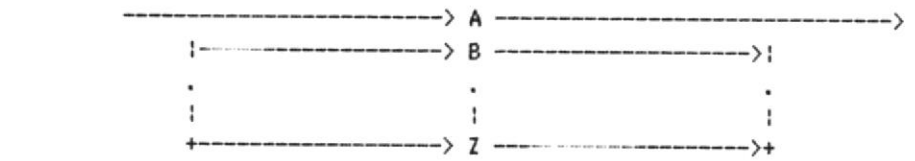
Otra caracteristica deliberada del Pascal, es que no hay restriccion en el numero de espacios (y lineas en blanco) usados entre simbolos especiales, palabras reservadas e identificadores en un programa, lo que permite a un programador escoger una presentacion del programa que sirva de ayuda a alguien que este probando comprender el programa.

Comentarios pueden ser insertados en el texto del programa y deben ser incluidos entre { y }, donde quiera que los espacios lo permitan. Estos simbolos pueden ser reemplazados por los simbolos (* y *) en algunas implementaciones.

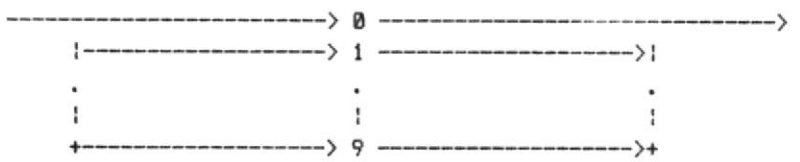
Las letras, digitos, palabras reservadas y simbolos especiales son clasificados como 'terminales' en teoria de lenguajes; [illegible] que son escritas usando letras minusculas en los diagramas de sintaxis son llamadas 'no-terminales'. Estas no aparecen en el programa del computador y cada no terminal sera definida en otro diagrama de sintaxis, el cual mostrara las reglas para la construccion de las no terminales en terminos de otras no terminales y eventualmente terminales.

El diagrama de sintaxis para un identificador fue dado anteriormente en terminos de dos no-terminales: una letra y un digito. Los diagramas de sintaxis para estos son simples y pueden ser dados directamente en terminos de terminales.

letra



digito



En los diagramas de sintaxis previos, las letras minusculas no fueron incluidas en la lista de simbolos especiales, porque en algunas implementaciones no las usan y en aquellas que las usan, el compilador no las distinguira de las letras mayusculas, y ambos tipos de letras solo son usados para representacion del texto.

Una regla adoptada en este curso, es usar letras mayusculas para palabras reservadas y letras minusculas para identificadores.

GENERALIDADES

Cada complemento de dato usado dentro de un programa Pascal tiene un tipo unico. Este tipo especifica el conjunto de valores que objetos de ese tipo pueden asumir y cualquier intento de dar a un objeto de datos un valor fuera de este conjunto de valores es tratado como un error de programacion. Por ejemplo:

```
PROGRAM ejemplodecolor;
TYPE politico = .....;
     partido = .....;
     color = .....;
VAR  mp : politico;
     partidodemp : partido;
     colordemp : color;
BEGIN
     leerenpolitico(mp);
     encuentrepartido(mp, partidodemp);
     seleccionarcolor(partidodemp, colordemp);
     imprimircolor(colordemp);
END.
```

En este ejemplo, el procedimiento 'encuentrepartido' es asumido para calcular un valor de tipo 'partido' y asigna ese valor a su segundo parametro. Si lo siguiente fue escrito por error

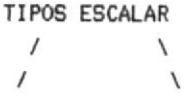
```
encuentrepartido(mp, colordemp);
```

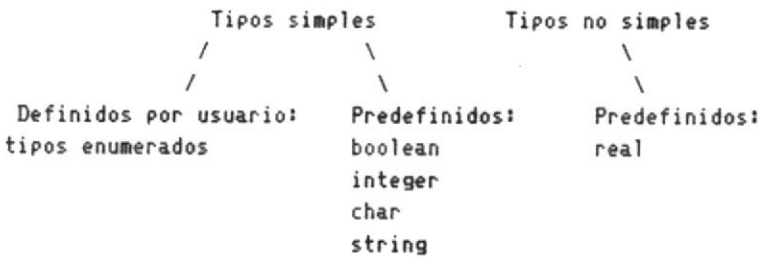
luego el compilador Pascal reportara un error ya que el tipo del segundo parametro no emparejo al tipo de dato que 'encuentrepartido' esperaba.

En comun con la mayoria de los lenguajes de programacion, Pascal provee un numero de tipos predefinidos y operadores, para permitir al programador representar valores aritmeticos, logicos y caracteres. Sin embargo, Pascal tambien provee facilidades para que el usuario pueda definir sus propios tipos apropiados a la tarea en mano y operadores para ese tipo en la forma de procedimientos.

TIPOS DE DATOS DEL PASCAL.- El Pascal provee dos tipos de datos: tipos escalar y estructurado, el primero de los cuales sera descrito en este capitulo.

El tipo escalar puede ser completamente clasificado como se muestra en el siguiente diagrama:

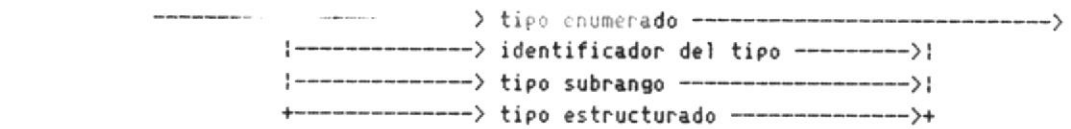




Todos los objetos de datos pueden tener un valor escalar simple. Los tipos simples tienen ademas la propiedad de ser capaces de constituir un conjunto ordenado de valores donde el numero de distintos valores en el conjunto es llamado el fundamental del tipo.

El diagrama de sintaxis de un tipo es:

tipo



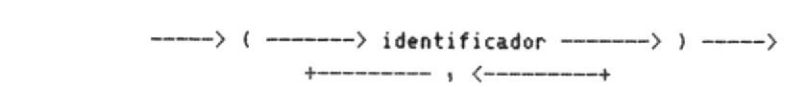
donde solo las tres primeras opciones se refieren al tipo escalar. La primera opcion denota la sintaxis usada para definir tipos enumerados y es descrita en la siguiente unidad. Todos los otros tipos escalar son predefinidos y son denotados en un programa simplemente escribiendo el identificador del tipo. Esto corresponde a la segunda opcion en el diagrama de sintaxis. La tercera opcion refiere a la sintaxis de una forma de tipo derivado llamado subrango y es descrito mas tarde.

TIPO ENUMERADO

Un tipo simple puede ser definido justamente enumerando los valores de ese tipo de tal forma que cada valor es denotado por un unico identificador.

El diagrama de sintaxis para un tipo enumerado es:

tipo enumerado



Como un ejemplo, los tipos 'politico', 'partido' y 'color' usados en el programa 'miembrodecolor', pudieron ser definidos como tipos enumerados asi:

```
TYPE politico = (dennis, david, jim, maggie);
     partido = (laboral, conservador, liberal, snp);
     color = (rojo, verde, amarillo, azul, purpura);
```

dando esta definicion, la variable 'mp' pudo asumir uno de los cuatro valores 'dennis', 'david', 'jim' o 'maggie' y no otro.

Dos funciones estandard son definidas para todos los tipos simples: la funcion sucesor 'succ' y la funcion predecesor 'pred', de tal forma que dado x de tipo simple T, luego

```
succ(x) denota el sucesor de x en T
pred(x) denota el predecesor de x en T
```

Como un ejemplo aplicando estas funciones a tipos enumerados

```
succ(jim)          es maggie
pred(conservador)  es laboral
```

El sucesor del ultimo valor y el predecesor del primer valor de un tipo simple es indefinido, por lo tanto

```
succ(maggie) es indefinido
```

Asi como 'succ' y 'pred', un conjunto de operadores relacionales es definido para todos los tipos escalar.

=	igual	>	mayor que
<>	no igual	<=	menor o igual que
<	menor que	>=	mayor o igual que

En el caso de los tipos enumerados anteriormente

```
jim < maggie   es verdad
rojo > azul    es falso
```

Note que `rojo <= conservador`

no es valido ya que los dos tipos de datos objetos que estan siendo comparados no son iguales. Escribiendo esto en un programa Pascal, conducira a un error de compilacion.

Los operadores relacionales dados anteriormente son ejemplos de operadores didacticos implatados, ellos son usados con dos operandos escritos uno a cada lado del operador. La secuencia de simbolos

```
jim < maggie
```

es llamada una expresion y tiene el valor en este caso de 'true', el cual es uno de los del tipo Boolean.

TIPO BOOLEAN

El tipo Boolean es un tipo predefinido del Pascal. El es un conjunto ordenado de los valores 'true' y 'false' y puede ser descrito como un tipo enumerado asi:

```
TYPE boolean = (false, true);
```

Pascal provee un conjunto de operadores logicos los cuales toman operandos boolean y producen un resultdo boolean. Estos incluyen

```
AND  Y logico
OR   O logico
NOT  No logico (negacion)
```

AND y OR son operadores didacticos, sus efectos pueden ser resumidos por una tabla de verdad donde se asume que las variables a y b han sido declaradas por

```
VAR a, b : boolean;
```

a	b	a AND b	a OR b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

NOT es un operador prefijado, usado con un operando simple escrito despues del operador. La tabla de verdad es

a	NOT a
true	false
false	true

Los operadores relacionales pueden ser usados con operandos boolean para extender el conjunto de operadores logicos, por ejemplo

```
=    denota equivalencia
<>   denota O exclusivo
<=   denota implicacion
```

La tabla de verdad para estos operadores puede ser calculada desde la definicion de boolean como un tipo enumerado. Por ejemplo, si a es false y b es true, luego

a <= b es verdad

Finalmente, note que 'succ' y 'pred' pueden ser aplicados a operadores boolean

succ(false) es true
pred(true) es false

Cuando se este escribiendo expresiones boolean es buena idea encerrar cada sub-expresion boolean entre parentesis. A falta de parentesis, la prioridad del operador mostrada a continuacion es seguidad:

OPERADOR	PRIORIDAD
NOT	mayor (hecha primero)
*, /, DIV, MOD, AND	↑
+, -, OR	!
<, <=, <>, >, >=	menor (hecha al final)

Donde se muestra que el operador NOT tiene la prioridad mas alta y que los operadores relacionales tienen la prioridad menor. Los operadores aritmeticos junto con AND y OR, concuerdan entre si. Por lo tanto los operadores relacionales tienen la menor prioridad, ellos pueden ser usados con parentesis para prevenir errores de sintaxis.

Por ejemplo, la expresion

x < y + z

que involucra las variables reales x, y i z, pudo ser interpretada correctamente como

x < (y + z)

ya que + tiene mayor prioridad que <. Sin embargo, la expresion

x < y OR z < y

pudo ser erroneamente interpretada como

x < (y OR z) < y

ya que OR tiene mayor prioridad que <. Obviamente, (y OR z) es una expresion boolean ilegal cundo y i z son variables reales. Use parentesis como se muestra

abajo, para prevenir un error de sintaxis.

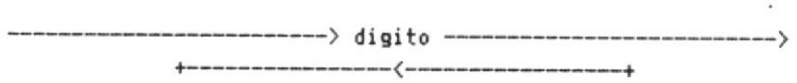
(x < y) OR (z < y)

TIPO INTEGER

El tipo integer consiste del conjunto ordenado de numeros enteros. Sin embargo, como este conjunto tiene cardinales infinitos, en la practica el tipo integer consiste unicamente de un subconjunto de los numeros enteros. El rango de valores permitidos depende de la implementacion.

A diferencia del tipo boolean, el tipo integer no es facilmente dado a una definicion de tipo enumerado. Por lo tanto valores de tipo integer son escritos con una sintaxis especial.

entero sin signo



asi 0, 123, 5493 son valores de tipo entero.

Pascal provee cinco operadores didacticos, los cuales toman operandos enteros y producen resultados enteros.

+	suma
-	resta
*	multiplicacion
DIV	division entera
MOD	residuo despues de la division entera

Ademas los operadores + y - pueden ser usados como operadores prefijados unitarios, para denotar identidad del signo e inversion del signo respectivamente No hay definiciones estandard para DIV y MOD, aparte de las convenciones que para cualquier entero x i y

(x DIV y) * y + (x MOD y) = x

Con tal que x i y sean positivos, no hay problema, ejemplo

7 DIV 3 es 2 7 MOD 3 es 1

pero si x es negativo, luego el resultado depende en todo caso de si la truncacion se acerca a cero o al infinito negativo, ejemplo

-7 DIV 3 es ? o -3

Es por lo tanto aconsejable no usar los operadores DIV y MOD con operandos negativos.

Algunos ejemplos del uso de los operadores aritmeticos son:

```
15   +   27   es   42
-9   +    4   es   -5
 4   -    9   es   -5
16  DIV   5   es    3
 9  MOD   2   es    1
 7   *    3   es   21
```

Finalmente, como en todos los tipos simples, los operadores relacionales y 'succ' y 'pred', pueden ser aplicados a operandos enteros. Algunos ejemplos son

```
5 < 4      es   falso
29 >- 3    es   verdad
succ(16)   es   17
pred(-4)   es   -5
```

TIPO CHAR

El tipo char es usado para representar el conjunto de valores que constituyen el conjunto de caracteres de la implementacion particular de Pascal que esta siendo usada. El es de este modo, desafortunadamente dependiente de la implementacion.

Un valor de tipo char es escrito como un caracter simple dentro de apostrofes, por ejemplo

```
'A',   'B',   '0'   ':'
```

el caracter ' mismo, es escrito como sigue

```
''''
```

Al igual que en todos los tipos simples, los operadores relacionales y 'succ' y 'pred' pueden ser aplicados a operandos de tipo char. No obstante los resultados de estas operaciones dependeran del conjunto de caracteres fundamentales que esta siendo usado. Es usualmente seguro asumir que los digitos y letras estan ordenados y que los digitos son contiguos, asi

```
'A'  <  'B'  es  verdad
'3'  <  '4'  es verdad
```

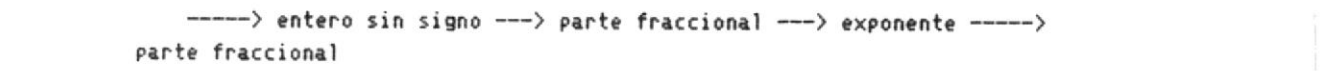
succ('B') es '9'

TIPO REAL

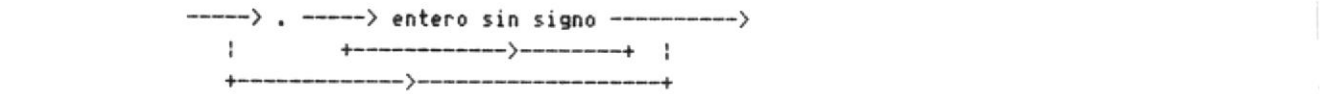
El tipo real consiste del conjunto de los numeros reales. Al igual que los enteros, el rango actual permitido de los numeros reales depende de la implementacion. Sin embargo, a diferencia [illegible] ro real dentro de un computador es solo aproximado. Si bien los numeros reales son tipo escalar, ellos no son tipo simple, porque son un intento de representar una funcion continua y no deben ser considerados como un conjunto ordenado de valores diferentes. Por esta razon 'succ' y 'pred' no son definidos para tipos reales.

Un valor de tipo real es escrito con la siguiente sintaxis

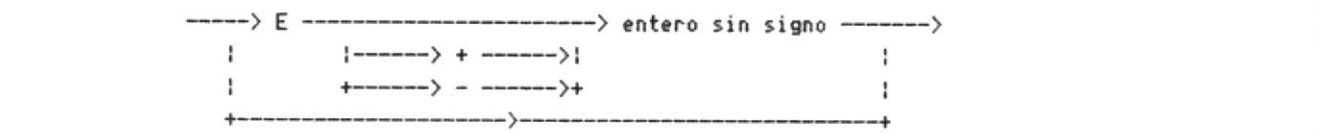
real sin signo



parte fraccional



exponente



por lo tanto, numeros reales validos son

34.968
27E-49
2.7E4

donde 2.7E4 es equivalente al valor de 27000.0.

Pascal provee cuatro operadores prefijados que pueden ser usados con operndos reales y producen un resultado real.

\+ suma

- resta
* multiplicacion
/ division

Al igual que con los enteros, + y - pueden ser usados como operadores unitarios, para representar la identidad e inversion del signo.

Los operadores relacionales pueden ser aplicados a operandos reales pero debe tenerse cuidado cuando se esta probando para igualdad, como por ejemplo, la relacion

$$(x / y) * y = x$$

no puede necesariamente ser verdadera. Cuando probamos para igualdad de numeros reales es usualmente mas seguro probar con una relacion de la forma

$$abs(x - y) <= e$$

donde abs(z) denota el valor absoluto de z y e es un numero muy pequenno.

TIPOS SUBRANGO

Cuando se conoce que un objeto de dato va a tomar valores que se encuentran dentro de un subrango del conjunto de valores denotados por algun tipo existente es buena practica definir el tipo de un objeto tal, como un tipo subrango.

La sintaxis de una declaracion de tipo subrango es

tipo subrango

constantes, son definidas formalmente en la siguiente unidad.

Ejemplos de declaraciones de tipo subrango son:

```
TYPE liderdelpartido = david .. maggie;
     digitooctal = 0 .. 7;
     letras = 'a' .. 'z';
```

El tipo desde el cual un tipo subrango se deriva es llamado el tipo base. Asi, en el ejemplo anterior, el tipo base de 'liderdelpartido' es el tipo enumerado 'politico'.

Un valor de un tipo subrango puede ser usado dondequiera que un valor del tipoo base es esperado; y todos los operadores, funciones y procedimientos defi-

nidos en el tipo base son igualmente aplicables al tipo subrango derivado.

Es digno de notar de paso, que el tipo entero es realmente un subrango del conjunto de todos los numeros enteros

```
TYPE integer = -mexint .. maxint;
```

La constante 'maxint' es un identificador predefinido del Pascal, cuyo valor es dependiente de la implementacion.

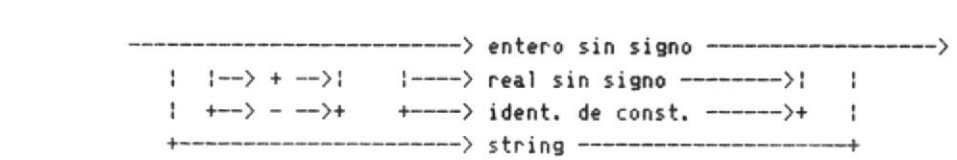
Es buena practica de programacion hacer el mayor uso posible de subrangos. Ello improvisa la claridad de un programa y el compilador es capaz de ejecutar chequeo de rango adicional sobre los valores tomados por las variables de tipos subrango, resultando asi programas mas seguros.

DECLARACIONES DE DATOS

Todos los objetos de datos en Pascal, son bien constantes o variables.

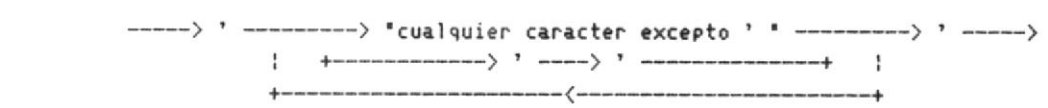
La sintaxis de una constante es

constante



donde

string



La forma de la constante de caracter dado anteriormente es un caso especial de un string de un caracter. String (cadenas de caracteres), son descritos mas adelante.

Un ejemplo tipico de una completa especificaicon de datos del Pascal, puede ser

```
CONST
```

```
     asterisco = '*';
     pi = 3.1415926;
     anchodelinea = 80;
TYPE
     frecuencia = real;
     indicedecolumna = 1 .. anchodelinea;
VAR
     i, j : indicedecolumna;
     w, f : frecuencia;
     ch : char;
```

ENTRADA ESTANDARD

Mediante la sentencia read, el lenguaje Pascal, permite la lectura de un dato y su asignacion a una variable del programa.

Ejemplo:

Supongamos que tenemos declaradas las variables:

```
largo, ancho, alto, area : integer
```

Ahora escribimos las tres sentencias de lectura (read) y la sentencia de asignacion correspondiente para calcular el area de un ortoedro determinado:

```
read(largo);
read(ancho);
read(alto);
area := 2 * (largo * ancho + largo * alto + ancho * alto);
```

si utilizamos como entrada los datos siguientes:

```
10   5   3
```

equivale a ejecutar las asignaciones siguientes:

```
largo := 10;
ancho := 5;
alto := 3;
```

con la posibilidad de cambiarse, modificando los datos de lectura. La sentencia read(variable-entera) lo que realiza es el seguir (o comenzar) la exploracion de caracteres de entrada, ignorando los blancos y el siguiente numero entero que se encuentre lo asigna a la variable-entera.

Se pueden leer varias variables con una sentencia read, colocandolas en forma de lista entre parentesis. Entonces los datos deben ir separados entre si por uno o mas blancos. En el ejemplo anterior se pueden pues fundir los tres read's quedando:

```
read(largo, ancho, alto);
```

SALIDA ESTANDARD

Con la sentencia write el lenguaje Pascal permite que el valor de una varia-

ble o expresion se imprima o represente en las salida estandard del ordenador.

Ejemplo: (continuacion de la unidad anterior)

Ahora hacemos que se imprima el resultado del calculo del area del ortoedro con la sentencia:

```
write(area)
```

el segmento de programa quedara pues asi:

```
read(largo, ancho, alto);
read(largo, ancho, alto);
area := 2 * (largo * ancho + largo * alto + ancho * alto);
write(area);
```

El papel de una instruccion de salida es sacar resultados del programa en forma legible, bien sobre papel de impresora, pantalla o cualquier otro dispositivo de salida.

En Pascal este cometido se realiza mediante la sentencia write.

Ejemplo:

Continuando con el mismo ejemplo anterior, podriamos escribir el valor del area mediante la sentencia:

```
write(area)
```

Si desearamos escribir tambien los datos leidos, largo, ancho, alto, la sentencia seria:

```
write(largo, ancho, alto, area);
```

y los valores de cada una de las variables saldrian impresos sobre una forma.

Lo mismo que en el caso de la sentencia read, la sentencia write puede tener cualquier numero de parametros. Sin embargo, cada parametro de una sentencia 'write', puede ser una 'expresion' y no necesariamente una variable simple, y el valor de esta expresion, una vez evaluado saldra escrito.

Ejemplo:

Si supongamos que m y n son variables enteras con valores asignados, la sentencia:

```
write(m, n, m + n, m - n, m * n);
```

imprimira los valores de m, n, de la suma, diferencia y producto, respectivamente.

ENTRADA/SALIDA

La salida expuesta en la unidad anterior, nos permite unicamente escribir los valores numericos de las variables. Si deseáramos que cualquiera de los valores [illegible] encerrado entre comillas aparezca como parametro en la sentencia write.

En la salida se reproduce dicho texto sin comillas.

Ejemplo:

La sentencia:

```
write('neto es =', neto);
```

producira la siguiente salida:

```
neto es =       50475
```

Hemos supuesto que la variable neto tiene el valor 50475.

Si el texto (string o tira de caracteres) tiene algun caracter comilla, esta debera escribirse duplicada.

En el ejemplo anterior se supone que la salida estandard de cualquier numero entero ocupa 12 posiciones.

Esta es la razon por la que despues del signo = aparecen varios blancos o espacios (siete).

Si conocieramos a priori que el valor de neto no excede de cinco cifras, la sentencia anterior podria escribirse:

```
write('neto es =', neto : 5);
```

El sufijo "5" de la variable neto:5, especifica que el valor de neto se escribira en un campo de 5 espacios.

```
neto es =50475
```

Ejemplo:

La siguiente sentencia:

```
write('bruto =',br:5,'desc =',des:4,'neto =',nt);
```

producira la salida:

```
bruto =56300    desc =5630    neto = 50670
```

Se habra observado hasta aqui que los valores numericos se escriben sobre una unica linea. Si desearamos escribir sobre varias lineas, utilizariamos writeln en lugar de write.

Utilizando el mismo ejemplo anterior, si escribieramos las siguientes sentencias:

```
writeln('bruto pts =',br:5);
writeln('desc pts =',des:5);
writeln('neto pts =',nt:5);
```

la salida que se produciria seria:

```
bruto pts =56300
desc pts = 5630
neto pts =50670
```

La utilizacion de una writeln sin parametros, produciria una linea en blanco

Podemos considerar igualmente que la entrada de datos consista de varias lineas.

En el caso de utilizar la sentencia read, se pasara automaticamente a una nueva linea de entrada de datos si no hay mas datos en la linea actual.

Para algunos propositos se desea forzar a que la entrada de datos sea una linea cada vez. La sentencia que [illegible] readln.

Despues de que una sentencia readln haya sido ejecutada, la parte no leida de la linea actual de entrada de datos es saltada, u omitida, y la siguiente entrada sera tomada de la siguiente linea.

Ejemplo:

Consideremos las sentencias:

```
readln(nmr);
readln(pqr);
```

donde nmr y pqr son variables enteras. Si la entrada de datos es:

```
50 num1
 8 num2
```

entonces los valores 50 y 8 respectivamente seran asignados a las dos variables; los textos de ambas lineas son ignorados.

Si las sentencias readln fueran reemplazadas por las sentencias read, la segunda sentencia read hubiera tomado como valor entero el texto num1, dando un error.

En la entrada de datos se leen estos siempre de izquierda a derecha dentro de cada linea, y desde una linea.

Como resumen de lo expuesto hasta ahora, vamos a describir las distintas formas mas generales de las sentencias de entrada y salida. La sentencia read tiene las cuatro formas siguientes:

```
read(input,v1,v2, ... ,vn);
read(v1,v2, ... ,vn);
readln(input,v1,v2, ... ,vn);

readln(v1,v2, ... ,vn);
```

donde las Vi representan las variables de entrada e input el archivo o fuente de los datos de entrada.

La sentencia de salida write tiene las cuatro formas siguientes:

```
write(output,e1,e2, ... ,en);
write(e1,e2, ... en);
writeln(output,e1,e2, ... en);
writeln(e1,e2, ... en);
```

donde las ei pueden tener una de las tres formas siguientes:

```
f
f:f1
f:f1:f2
```

siendo f, f1, y f2 expresiones:

f es el valor que se escribe y puede ser de cualquier tipo.

f1 es un control opcional y sirve para definir la longitud del campo de salida, es decir, el numero de caracteres que se desea salgan impresos; f1 debe ser un numero natural.

f2 se le llama longitud de fraccion y es un control tambien opcional. Es aplicable solo en el caso que f tenga un valor decimal. Debe ser un numero [illegible] y especifica el numero de [illegible]

La lista de variables o expresiones encerradas entre los parentesis de las sentencias de entrada y salida se las denominara tambien lista de parametros como se vera mas adelante en el capitulo de procedimientos.

SENTENCIA IF

Un programa ha sido tratado a menudo como una secuencia de declaraciones de datos seguida por una secuencia de acciones (sentencias de Pascal). En muchos casos puede ser necesario, bajo algunas condiciones, bien ignorar algunas de las acciones o ejecutar algunas adicionales. Este capitulo se refiere a las estructuras que estan disponibles en Pascal para controlar la secuencia de acciones.

A veces es necesario especificar dos o mas cursos de accion de tal forma que durante la ejecucion del programa uno de ellos sea seleccionado.

Para seleccionar una de dos acciones se usa la sentencia IF. El diagrama de sintaxis es el siguiente:

sentencia IF

Es importante notar que no hay punto y coma entre la sentencia del THEN y la del ELSE. El punto y coma separa sentencias de Pascal; y no solamente separara el ELSE de la parte del THEN de la sentencia IF, sino que tambien causa que el compilador pruebe y encuentre un comienzo de sentencia con un ELSE. Como no hay tal diagrama de sintaxis, un error sera reportado.

La frase 'expresion', es usada bastante como condicion, por lo tanto la parte condicional de una sentencia de Pascal puede ser cualquier expresion aritmetica o logica complicada, que produzca un valor booleano.

Ejemplo del uso de la sentencia IF

```
VAR  numero, base : integer;
     lado, area : real;
BEGIN
     ....
     IF numero < base - 1
          THEN numero := numero + 1
          ELSE numero := 0;
     ....
END
```

La parte ELSE pu[illegible]

```
VAR  pequenno, grande : real;
```

```
....
IF grande > pequenno
     THEN
          BEGIN
               grande := pequenno;
               pequenno := 0
          END
```

este grupo de sentencias tiene un efecto diferente que las siguientes:

```
IF grande > pequenno
     THEN grande := pequenno;
pequenno := 0;
```

En algunas situaciones una ambiguedad puede ocurrir con sentencias IF necesitadas.

Considere por ejemplo, la interpretacion de

```
IF condicion1 THEN
     IF condicion2 THEN sentencia1
ELSE sentencia2
```

En este caso el compilador estara inconsciente de la discusion que se trata y no sabra que IF asociar con el ELSE. La regla en Pascal es que un ELSE es asociado con el IF mas cercano. En este caso la sentencia IF sera tratada como:

```
IF condicion1 THEN
     IF condicion2 THEN sentencia1
     ELSE sentencia2
```

lo que significa que el camino es indeterminado si la condicion1 no es verdad. A fin de tomar el siginificado correcto, la sentencia en este caso puede ser realizada en dos sentencias IF separadas

```
IF condicion1 THEN
     IF condicion2 THEN sentencia1;
IF condicion1 THEN sentencia2;
```

En algunos casos la condicion puede tener mas de dos valores, verdadero/falso o si/no. Considere el ejemplo de una persona que va a trabajar de lunes a viernes, va al futbol el sabado y lava el carro el domingo. Su accion dependera del dia de la semana:

```
IF diaeslunes THEN iraltrabajo
ELSE
   IF diaesmartes THEN iraltrabajo
```

```
ELSE
   IF ...
            .
            .
      ELSE
         IF diaessabado THEN iralfutbol
            ELSE lavarcarro
```

La sentencia IF es muy compleja en esta situacion y alguna forma de estructura de conmutador puede ser preferible. Este tipo de estructura es implementada en Pascal y es llamada [illegible]

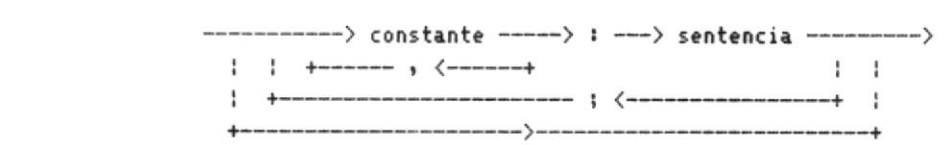
SENTENCIA CASE

Hay muchas decisiones en las cuales hay multiples alternativas a considerar (mas de dos). La sintaxis de la sentencia case es:

sentencia case

donde la lista del case es definida como:

lista del case



Debe notar que un punto y coma no es requerido despues de la ultima sentencia en la lista del case; y el compilador de algunas implementaciones reportara un error si este punto y coma extra es incluido en el programa. Esto contrasta con la secuencia de sentencia del Pascal, donde un punto y coma colocado despues de la ultima sentencia es ignorado y un error no es reportado.

La expresion es evaluada y comparada con cada una de las constantes. Solo una sentencia sera ejecutada; si el valor de la expresion es listada en constante(i), luego la sentencia(i) es ejecutada. El control es despues pasado a la primera sentencia siguiente al END del case.

Si el valor de la expresion no corresponde a alguna de las constantes en la lista del case, la accion resultante depende de la implementacion particular del Pascal. En UCSD Pascal por ejemplo, el computador sale de la sentencia case y

continua con la siguiente sentencia en el programa.

Un valor de la expresion puede aparecer en una constante, al menos.

El tipo de cada valor debe corresponder al tipo de la expresion.

Cualquier dato de tipo escalar o subrango es permitido (incluyendo integer, boolean y char, pero no real o string).

El ejemplo de la unidad anterior puede aparecer en un programa Pascal como:

```
PROGRAM diario;
TYPE dia = (lunes, martes, miercoles, jueves, viernes,
            sabado, domingo);
VAR diadelasemana : dia;
BEGIN
          .
          .
     consigadiadelasemana;
     CASE diadelasemana OF
          lunes, martes, miercoles,
          jueves, viernes          : iraltrabajo;
          sabado                   : iralfutbol;
          domingo                  : lavarcarro
     END;
          .
          .
END.
```

En este ejemplo la accion 'iraltrabajo' es ejecutada si la variable 'diadelasemana' tiene un valor 'martes', mientras el carro sera lavado si 'diadelasemana' tiene un valor 'domingo'. No hay forma de abreviar:

```
lunes .. viernes : iraltrabajo;
```

esto no es permitido.

Si ninguna accion toma lugar en un dia particular, digamos 'miercoles', luego una sentencia en blanco es usada:

```
CASE diadelasemana OF
     lunes, martes, jueves, viernes : iraltrabajao;
     miercoles                      : ( nada a hacer );
     sabado                         : iralfutbol;
     domingo                        : lavarcarro
END;
```

Sentencias en blanco son usadas por aquellas implementaciones de Pascal que difieren del UCSD Pascal. Asi en UCSD Pascal los vlaores que no requieren alguna accion pueden ser sacadas de la lista del case y el ejemplo puede ser escrito

```
CASE diadelasemana OF
     lunes, martes, jueves, viernes : iraltrabajo;
     sabado                         : iralfutbol;
     domingo                        : lavarcarro
END;
```

ACCIONES MULTIPLES

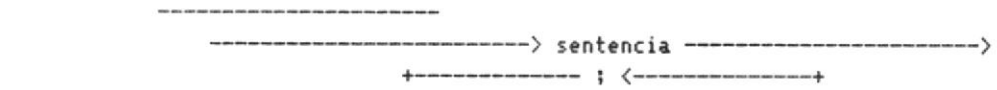
El diagrama de sintaxis para la sentencia IF solo permite una sentencia despues del THEN y una sentencia despues del ELSE. Similarmente, solo una sentencia es permitida seguida por un punto y coma en la sentencia case. En muchas ocasiones mas de una accion pueden ser requeridas.

Pascal permite sentencias a ser agrupadas juntas para formar una 'sentencia compuesta', encerrando la sentencia entre un BEGIN - END. La sintaxis de una sentencia compuesta es:

sentencia compuesta

donde la secuencia de sentencia fue dada anteriormente (Cap. 1, Uni. 2), como:

secuencia de sentencia



Una sentenica compuesta puede reemplazar la parte de sentencia en cualquiera de los diagramas de sintaxis. Ejemplo:

```
IF area >= 0
     THEN lado := sqrt(area)
     ELSE BEGIN
               lado := 0
               write ('area negativa')
          END
```

para la sentencia case:

```
domingo : BEGIN
               iralaiglesia;
```

```
            lavarcarro
        END
```

Una regla util al usar sentencias compuestas es indentar la secuencia de sentencia por 2 o mas caracteres como en los ejemplos anteriores. Esto es particularmente util donde sentencias compuestas sean necesitadas, asi la indentacion permite que las sentencias incluidas por cada sentencia compuesta sean claramente visibles.

La sentencia compuesta puede ser usada para segurar que cualquier ambiguedad en sentencias IF necesitadas, sea evitadas. El ejemplo

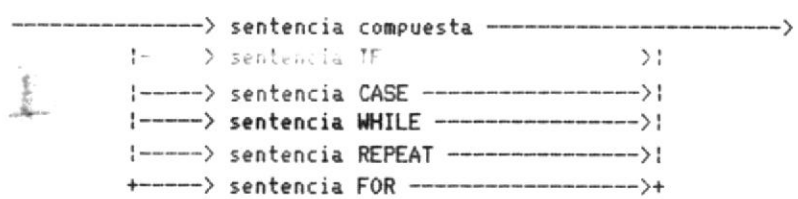
```
IF condicion1
    THEN
        BEGIN
            IF condicion2
                THEN sentencia1
            END
        ELSE sentencia2
```

es interpretada en la forma deseada, por el computador.

ITERACIONES EN PASCAL

Ademas de ordenar y seleccionar un programa, generalmente requerira de alguna iteracion, donde ciertas acciones sean ejecutadas un numero de veces. Un programa bien estructurado debe estar compuesto completamente de estos tres tipos de operacion. Estos pueden ser agrupados juntos en un diagrama mostrando una sentencia estructurada:

sentencia estructurada



donde la sentencia compuesta corresponde a secuencia, la sentencia if y la sentencia case representan las dos formas de seleccion en Pascal y las tres sentencias finales representan iteracion.

SENTENCIA FOR

Cuando el numero de veces que se debe repetir un lazo es conocido de antemano, la sentencia mas apropiada es la sentencia FOR.

La sintaxis de la sentencia FOR es dada por:

sentencia for

```
---> FOR -> identificador -> := -> rango del for -> DO -> sentencia -->
```

donde el rango del for es definido por:

rango del for

```
---> expresion -------> TO ------------> expresion --->
                  +--> DOWNTO -----+
```

La sentencia despues del DO es ejecutada una vez por cada uno de los valores de la variable de control en el rango del for. La expresion es simplemente los valores inicial y final que toma el identificador variable. Estos valores pueden ser constantes, variables o expresiones; no obstante, identificador, valor inicial y valor final deben ser todos del mismo tipo. Cualquier tipo escalar es permitido incluyendo integer, char y boolean.

El valor del identificador no puede ser modificado en la sentencia. El valor final es calculado justo antes de la ejecucion de la sentenica. Cualquier cambio subsecuente en las variables que forman la expresion final no afectara el numero de veces que la sentencia es ejecutada.

Despues de salir desde la sentencia for, el valor del identificador es considerado indefinido. En el caso de usar TO en el for, la sentencia no sera ejecutada si el valor inicial es mayor que el valor final. Si se usa DOWNTO, la sentencia no sera ejecutada si el valor inicial es menor que el valor final.

El siguiente ejemplo [illegible] version de [illegible] grados Celsius [illegible] grados Fahrenheit por [illegible] 2 grados C. y decrementando a -2 grados C. Un ejemplo de ejecucion es tambien mostrado.

```
PROGRAM temperaturas;
(Conversion de grados Celsius a grados Fahrenheit)
CONST
     maxcelsius = 2;
     mincelsius = -maxcelsius;
VAR
     celsius : integer;
```

```
     fahrenheit : real;
BEGIN
     writeln('celsius   fahrenheit');
     FOR celsius := maxcelsius TO DOWNTO mincelsius DO
          BEGIN
               fahrenheit := 1.8 * celsius + 32.0;
               write('celsius:4, fahrenheit:12:1)
          END
END.
celsius   fahrenheit
   2         35.6
   1         33.8
   0         32.0
  -1         30.2
  -2         28.4
```

SENTENCIA REPEAT

La sentencia repeat es usada cuando al escribir un programa se desconoce el numero de iteraciones que seran necesarias para ejecutar una sentencia. La sintaxis de la sentencia repeat es:

sentencia repeat

---> REPEAT ---> secuencia de sentencia ---> UNTIL ---> expresion --->

La expresion tendra un valor booleano y despues de cada ejecucion de la secuencia de sentencia, la expresion es evaluada. Si la expresion es falsa, la secuencia de sentencia es ejecutada otra vez. Si la expresion evaluada es verdadera, una salida del lazo ocurre y la siguiente sentencia del programa es ejecutada. La secuencia de sentencia es siempre ejecutada por lo menos una vez.

Para escribir lazos usando la sentencia REPEAT, se debe tener en cuenta los siguientes aspectos:

a) La expresiones usadas deben ser correctas.

b) Las sentencias que forman la secuencia de sentencia, deben ser escritas en la secuencia correcta, debe haber por lo menos una sentencia que tenga efecto la expresion de terminacion, pues de otro modo el lazo se ejecutara infinitamente.

c) La expresion de terminacion debe eventualmente ser satisfecha.

Las palabras reservada REPEAT - UNTIL, actuan de la misma manera que [illegible]

labras BEGIN y END
TIL es colocada exactamente despues de la palabra REPEAT, todas las sentencias que se encuentran entre ellas se indentan.

Por ejemplo, si queremos saber cuantos terminos de la serie armonica seran necesarios para satisfacer la desigualdad

```
1 + 1/2 + 1/3 + .... + 1/n  >  limite
```

podemos escribir un segmento de programa que use la sentencia REPEAT:

```
....
contadorterminos := 0;
suma := 0;

read(limite);
REPEAT
     contadorterminos := contadorterminos + 1;
     suma := suma + 1 / contadorterminos
UNTIL suma > limite;
```

SENTENCIA WHILE

Otra forma de ejecutar sentencias repetidamente es usando la sentencia WHILE La sentencia WHILE es similar a la sentenica REPEAT, la excepcion es que la expresion de terminacion del lazo se evalua al inicio y no al final. La sintaxis de la sentencia WHILE ES:

sentencia while

Por supuesto que la sentencia sobre la cual la sentencia while tiene efecto, puede ser una sentencia compuesta.

```
WHILE condicion DO
     BEGIN
          sentencias
     END
```

La sentencia dentro del lazo debe tener algun efecto sobre la expresion de terminacion del lazo, pues de otra forma el lazo se ejecutara infinitamente.

En el caso de la sentencia WHILE, se tiene el requerimiento adicional de que al momento de iniciar el lazo la expresion debe estar bien definida.

Que imprimira el siguiente programa?

```
PROGRAM desconocido(output);
VAR
     numero : integer;
BEGIN
     numero := 0;
     WHILE numero <= 10 DO
          BEGIN
               numero := numero + 1;
               write(numero)
          END
END.
```

El archivo de salida contendra:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

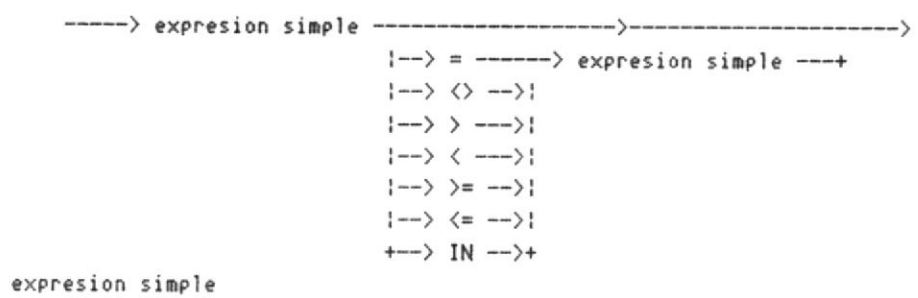
Sin embargo, muy comunmente los programadores inexpertos asumen que la sentencia while chequea la expresion de terminacion de una manera [illegible] el lazo se termina tan pronto como el numero sea igual a 10, lo cual no es cierto. La expresion de terminacion es evaluada al principio del lazo solamente.

En la practica de programacion, la sentencia while es mas util que la sentencia repeat. Esto se debe a que en muchos casos existe la posibilidad de que el lazo no sea ejecutado y esta posibilidad solo la ofrece la sentencia while, pues en la construccion repeat, el lazo se ejecuta por lo menos una vez. Si en alguna instancia se tiene duda entre usar la sentencia while o la sentenica repeat, use la sentencia while primero.

JERARQUIA EN EXPRESIONES

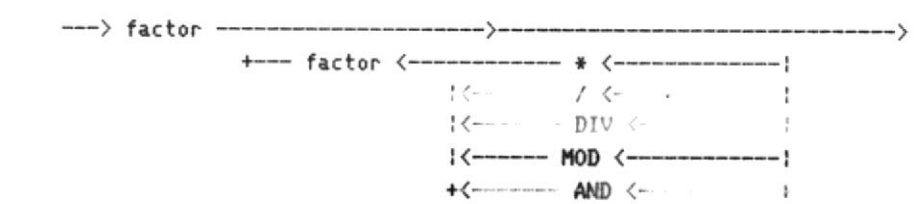
Pascal tiene cuatro niveles de jerarquia para determinar el valor de una expresion. Ademas de las operaciones matematicas, las cuales tienen la misma jerarquia como en Matematicas, las operaciones logicas AND, OR y NOT han sido incluidas. Las frases 'expresion', 'expresion simple', 'termino' y 'factor' son usadas en Pascal para indicar los diferentes niveles de jerarquia, como se muestra en los diagramas de sintaxis:

expresion



expresion simple

termino

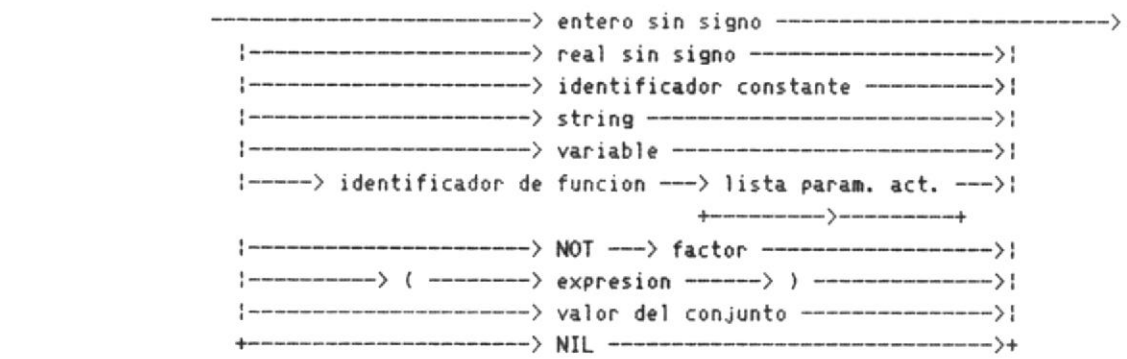


El uso [illegible] es como sigue:

= igual a <> no igual a > mayor que < menor que
>= mayor o igual que <= menor o igual que
DIV division entera MOD residuo despues de la division entera
OR O logico AND Y logico NOT valor logico opuesto

IN miembro del conjunto NIL puntero nulo

factor



Las diferentes operaicones de multiplicacion y division seran referidas mas tarde en este capitulo, mientras que el siginificado y uso de los simbolos IN y NIL seran explicados en los cpaitulos sobre conjuntos y datos dinamicos respectivamente. En muchos casos un simbolo tiene diferentes valores dependiendo del tipo de dato involucrado. Los operadores aritmeticos +, -, =, etc., tienen sus valores usuales cuando son usados con variables de tipo entero, pero tienen significados diferentes en el tratamiento con conjuntos. En este caso los significados son:

+ union de conjunto - diferencia de conjunto * interseccion de conjunto
= igualdad de conjunto <> no igualdad de conjunto <=, >= inclusion de conjunto

Estos seran discutidos en detalle en el capitulo sobre conjuntos.

EVALUACION DE EXPRESIONES

Al evaluar una expresion, el computador determina los valores de los factores primarios, y luego aplica algunas operaciones y finalmente evalua la expresion. Recuerde que NOT tiene la jerarquia mas alta y las operaciones de relacion la mas baja. Secuencia de operadores de la misma jerarquia son ejecutados de izquierda a derecha.

Algunos ejemplos son:

factor:
57
3.141
siempre

```
'cadena'
maggie
succ(maggie)
NOT tanquelleno
(numerodegalones + 3)
[maggie, jim]

termino:
     tanquevacio AND dinerolisto
     numerodegalones DIV 4

expresion simple:
     tanquelleno OR garajecerrado
     numerodegalones + 3

expresion:
     numerodegalones = 8
     contador <= capacidaddeltanque
```

En los ejemplos dados hasta ahora la expresion habia producido un resultado booleano. El resultado de una expresion no esta restringido a un valor booleano y puede ser de cualquier tipo. La expresion en la sentencia case es un ejemplo tipico de como la sentencia que es ejecutada depende de los valores de la expresion. En los ejemplos dados en capitulos anteriores:

```
conseguirdiadelasemana;
CASE diadelasemana OF
     lunes, martes, miercoles,
     jueves, viernes          : iraltrabajo;

     sabado                   : iralfutbol;
     domingo                  : lavarcarro
END
```

la expresion consistia completamente de un factor, la variable 'diadelasemana' y el valor particular de 'diadelasemana' determinaba que sentencia era ejecutada Una version alternativa del ejemplo anterior la cual usa una expresion mas complicada es dada abajo. En este ejemplo una variable 'diadelmes' es usada y el residuo despues de una division entera para 7, 'diadelmes MOD 7', da el dia de la semana en el rango 0 (dia lunes) al 6 (domingo). Si el primero del mes es un dia miercoles (dia 2 de la semana), luego el ejemplo se convierte en:

```
CASE (diadelmes + 1) MOD 7 OF
     0,1,2,3,4 : iraltrabajo;
         5     : iralfutbol;
         6     : lavarcarro
END
```

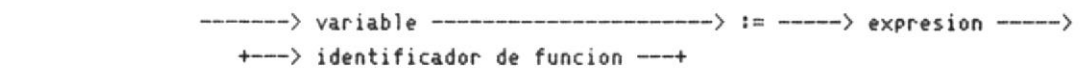
ASIGNACION DE VALORES A DATOS

El control en el ejemplo de la unidad anterior pudo haber sido claro si el valor de [illegible] cual seria luego usada en la sentencia case:

```
asignar (fechadelmes + 1) MOD 7 a 'numerodeldia'
CASE numerodeldia OF
     0,1,2,3,4 : iraltrabajo;
         5     : iralfutbol;
         6     : lavarcarro
END
```

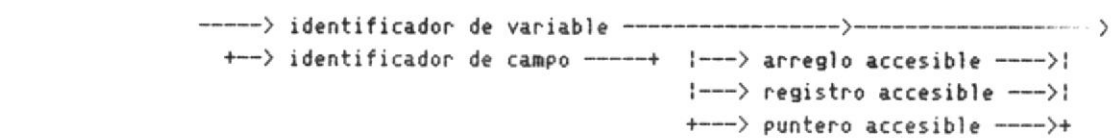
Una sentencia de asignacion es usada en Pascal para asignar el valor de una expresion a una variable y tiene la sintaxis:

sentencia de asignacion



donde una variable tiene la sintaxis:

variable



La unica variable que ya ha sido mencionada es el identificador de variable, el cual es declarado usando una declaracion VAR. Las otras clases seran mencionadas en el capitulo sobre datos de tipos estructurados.

El simbolo de asignacion :=, ya ha sido visto en la sentencia for donde a la variable se le asignaron valores desde la expresion inicial TO o DOWNTO a la expresion final. El ejemplo anterior sera escrito en Pascal como:

```
numerodeldia := (fechadelmes + 1) MOD 7;
CASE numerodeldia OF
     0,1,2,3,4 : iraltrabajo;
         5     : iralfutbol;

         6     : lavarcarro
END
```

IMPLICACIONES DE REPRESENTACIONES PODEROSAS

En Pascal el tipo resultante de una expresion debe ser del mismo tipo que el de la variable a la cual es asignado. Esta 'representacion poderosa' es una caracteristica deliberada del Pascal y mejora considerablemente la seguridad de un programa.

Algunos ejemplos de asignaciones validas son:

```
diadelasemana := viernes
tanquelleno := true
contador := 5
mp := maggie
color := verde
```

Ejemplo de expresiones de tipo enumerado:

```
ultimomp := pred(maggie)          {asigna a 'ultimomp' el valor precedente
                                   a 'maggie' en la declaracion del tipo
                                   'politico'}

tanquevacio := NOT tanquelleno    {si 'tanquelleno' es verdadero, luego
                                   'tanquevacio' se le asigna el valor fal-
                                   so y viceversa}

numero := tres * dos + uno        {numero := 7}

numero := tres * (dos + uno)      {numero := 9}

numerodeldia := (fechadelmes + 1) MOD 7

valorbooleano := (entero1 > entero2) OR (real1 < real2)
```

El ultimo ejemplo es legal no obstante que datos enteros y reales estan presentes en la expresion, porque los factores:

```
entero1 > entero2        y        real1 > real2
```

son evaluados primero para dar resultados booleanos. El resultado booleano de estos dos valores intermedios es luego asignado a la variable 'valorbooleano'.

Las siguientes asignaciones son invalidas porque hay conlicto de tipos:

```
diadelasemana := verde          tanquelleno := 8
mp := maggie + 5                color := rojo * 2
```

CONVERSION DE DATOS DE UN TIPO A OTRO

En muchos problemas es necesario convertir un numero entero a un numero real Por ejemplo, la conversion de un numero entero de millas a kilometros necesitara tener un resultado real, si algunos grados de precision son requeridos. En Pascal es posible transferir datos de un tipo a otro. El mecanismo usado depende de los tipos de datos involucrados en la transferencia.

DATOS REAL Y ENTERO.- La representacion poderosa es cedida en el caso de numeros reales y es posible usar un entero si un numero real es esperado. Cuando esto ocurre Pascal automaticamente convierte el valor entero a su equivalente numero real. La conversion es sin embargo, no permitida. Un numero real no puede ser usado donde un entero es esperado ya que la parte fraccional del numero podria ser perdida y la conversion [illegible] precision del numero. Estas reglas tambien se aplican a asignaciones asi como expresiones. Ejemplos validos son:

```
numeroreal := numeroreal + 26       {el 26 es convertido a real antes de ser
                                     sumado}
numeroreal := numeroreal * 3        {el 3 es convertido a real antes de ser
                                     multiplicado}

numeroreal := numeroentero + 26     {la expresion entera es convertida a real
                                     para la asignacion}
```

Ejemplos invalidos son:

```
numeroentero:= numeroentero + 26.3 {conversion de 26.3 a entero puede causar
                                    una perdida en la precision}

numeroentero:= numeroentero * 3.7  {conversion de la expresion a entero pue-
                                    de causar una perdida en la precision}
```

La restriccion de que numeros reales no pueden ser automaticamente asignados a enteros en Pascal, se debe a que una perdida de precision significa que muchas de las operaciones matematicas y funciones que toman un argumento entero, tienen que producir un resultado real.

Pascal provee un numero de funciones matematicas estandares que pueden ser usadas por el programador. Los nombres y descripciones de estas funciones son dadas en la siguiente tabla. El nombre de la funcion es siempre seguida por su argumento encerrado entre parentesis.

Cualquier expresion aritmetica legal puede ser usada como un argumento para estas funciones.

Nombre	Descripcion del calculo	Argumento	Resultado
abs	El valor absoluto del argumento	real/entero	real/entero
atan	El arco tangente del argumento	real/entero (positivo)	real
chr	El caracter cuya representacion interna es dada como argumento	entero	char
cos	El coseno del argumento	real/entero	real
exp	El valor de e(2.71828) elevado a la potencia del argumento	real/entero	real
ln	El logaritmo (en la base e) del argumento	real/entero	real
log	El logaritmo (en la base 10) del argumento	real	real
odd	El valor booleano del argumento	entero	booleano
ord	El numero ordinal que corresponde a la representacion interna del argumento	boolean/char entero/escalar	entero
pred	El valor cuyo numero ordinal es uno menos que el numero ordinal de su argumento	boolean/char entero/escalar	boolean/char entero/escal.
pwroften	El valor de 10 elevado a la potencia del argumento	entero	real
round	El valor entero mas cercano al argumento	real	entero
sin	El seno del argumento	real/entero (radianes)	real
sqr	El cuadrado del argumento	real/entero	real/entero
sqrt	La raiz cuadrada positiva del argumento	real/entero (radianes)	real
succ	El numero ordinal del argumento mas uno	boolean/char entero/escalar	boolean/char entero escal.
trunc	La parte integral del argumento	real	entero

DECLARACION DE PROCEDIMIENTOS

Un procedimiento define un grupo de acciones llamadas. Estas acciones son invocadas simplemente nombrando al procedimiento en una sentencia de llamada del procedimiento. La sintaxis de una declaracion de procedimiento es:

declaracion de procedimiento

declaracion del procedimiento

cabecera

Ignorando parametros por el momento, puede ser visto que un procedimiento es declarado escribiendo la palabra reservada PROCEDURE seguida por un identificador que denota el nombre del procedimiento seguido por un bloque. La forma de un bloque es exactamente igual al del programa principal, el conjunto de acciones definidas por las sentencias entre el BEGIN - END del bloque, denota las acciones a ser ejecutadas por el procedimiento cuando el es invocado o llamado por una sentencia de llamada de procedimiento. Ejemplo:

```
PROCEDURE alinearsalida;
...
BEGIN
     writeln;
     FOR column := 0 TO maxfila DO
          write(' ':anchodecampo DIV 2)
END;
```

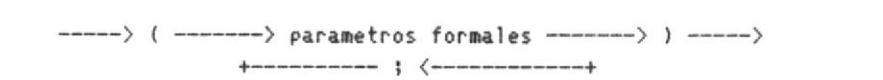
Llamada al procedimiento 'alinearsalida'

```
...
alinearsalida;
...
```

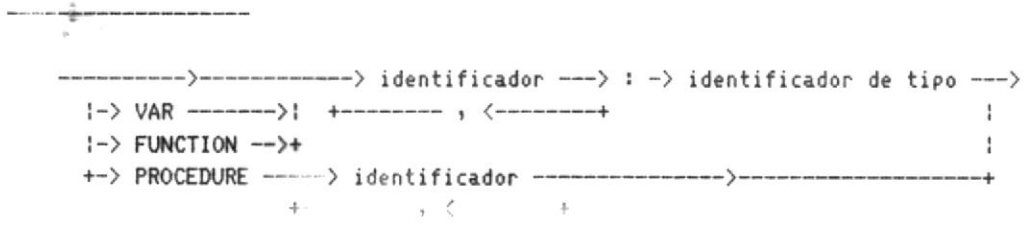
PARAMETROS DEL PROCEDIMIENTO

La sintaxis de una declaracion de procedimiento muestra que la cabecera del procedimiento puede incluir una lista de parametros formales cuya sintaxis es la siguiente:

lista de parametros formales



parametros formales



Asi una lista de parametros formales consiste de una lista de secciones de parametros formales separados por punto y coma y encerrados con parentesis. Cada seccion denota una lista de objetos del mismo tipo y clase de parametro. La omision de la clase del parametro es la clase del valor. La palabra reservada VAR denota la clase de la variable, las palabras reservadas FUNCTION y PROCEDURE denotan las clases de la funcion y del procedimiento respectivamente. Como un ejemplo, la cabecera del procedimiento

```
PROCEDURE xyz(VAR a,b:char; i:integer; VAR x,y:real);
```

declara el procedimiento 'xyz' a tener dos parametros variables: a y b de tipo char, un solo parametro por valor i de tipo integer y dos parametros variables x i y de tipo real.

Cuando un procedimiento es llamado, los parametros actuales suministrados por la sentencia de llamada del procedimiento deben coincidir con los correspondientes parametros formales en la declaracion del procedimiento en numero, orden y tipo.

PARAMETROS POR VALOR

Parametros por valor son usados para pasar un valor a un procedimiento y no pueden ser usados para retornar resultados. El mecanismo de pasar parametros por valor es explicado mejor por un ejemplo. Considere el procedimiento definido como

```
PROCEDURE imprimirasteriscos(n:integer);
BEGIN
     WHILE n > 0 DO
          BEGIN
               imprimirasteriscosimple;
               n := n - 1
          END
END;
```

y la llamada

```
imprimirasteriscos(contador DIV 2);
```

El parametro formal por valor 'n' es tratado dentro de 'imprimirasteriscos' como una variable local, pero a diferencia de las variables locales normales, el es inicializado con el valor denotado por el correspondiente parametro actual en el punto de llamada. Este parametro actual puede ser cualquier variable o expresion del mismo tipo que el del parametro formal. Asi, en el ejemplo, cuando 'imprimirasteriscos' es llamado, el valor de la expresion 'count DIV 2' es calculado y el resultado [illegible] riscos', 'n' es tratada como una variable local inicializada a [illegible]. Puede ser modificada sin afectar el parametro actual suministrado, aun si este ha sido una variable. Por ejemplo, si la llamada ha sido

```
m := 3;
imprimirasteriscos(m);
```

luego, despues de la llamada, el valor de m todavia sera 3, aun si el parametro formal n es cambiado a 0 durante la operacion de 'imprimirasteriscos'.

PARAMETROS VARIABLES

Parametros variables pueden ser usados para transmitir valores a un procedimiento y retornar resultados desde un procedimiento. El parametro actual suministrado en una llamada del procedimiento, para enlazar un parametro formal VAR, debe ser una variable. Dentro de un procedimiento, todas las referencias a un parametro VAR, seran tratadas como referencias a la variable suministrada como parametro actual. Asi por ejemplo, la accion del siguiente programa

```
PROGRAM ejemplo;
     VAR
          x:integer;
     PROCEDURE pasar(VAR z:integer);
     BEGIN
          z := z + 10
     END;
```

```
BEGIN
     x := 20;
     pasar(x)
END.
```

es que luego de la llamada 'pasar(x)', todas las referencias hechas al parametro formal z en el procedimiento 'pasar', se convierten en referencias al parametro actual x. Asi x es incrementado por 10 a un nuevo valor de 30.

NECESIDAD DE PROCEDIMIENTOS Y EXTENSION DE REGLAS DEL PASCAL

El cuerpo de un procedimiento es un bloque y un bloque puede contener declaraciones de procedimientos. En otras palabras, declaraciones de procedimientos pueden ser necesitadas y esta necesidad puede dar lugar a cualquier complejidad.

Un procedimiento declarado dentro de un procedimiento se dice a ser local a ese procedimiento y no puede ser referenciado desde fuera de el. Por ejemplo:

```
...
PROCEDURE imprimirtriangulo;
     ...
     PROCEDURE alinearsalida;
     ...
     BEGIN
          ...
     END;
BEGIN
     ...
     alinearsalida;
     ...
END;
```

Hay [illegible] casionan que el texto de los procedimientos incluidos se extienda y fragmente, y por lo tanto se dificulta [illegible]. Segundo, ellos fuerzan al programador a escribir el programa en orden inverso al que el habia disennado. Un ordenamiento libre es posible, si procedimientos necesitados son usados, haciendo uso de la facilidad de la declaracion forward descrita mas tarde. Tercero, el acceso de variables no locales desde dentro de un procedimiento necesitado, es menos eficiente que desde un procedimiento no necesitado en muchas maquinas.

No obstante, los procedimientos necesitados pueden jugar una regla util en construccion de programas. A fin de usarlas apropiadamente, es necesario comprender las reglas extendidas del Pascal.

LA REGLA EXTENDIDA DEL PASCAL.- Un bloque x (por ejemplo), define la region

de texto sobre la cual un identificador declarado dentro de ese bloque puede ser referenciado. Esta region incluye todos los bloques necesitados, excepto que si un identificador es redeclarado en un bloque necesitado Y luego, la extension de ese bloque necesitado Y y todos los bloques dentro de el, son excluidos desde la extension de la declaracion de identificador en el bloque X.

Puesto que Pascal permite una estructura de bloque necesitado, el es dicho a ser un 'lenguaje de bloque estructurado'. Debe ser notado con interes, que al extenderse este, los identificadores estandares del Pascal son considerados a ser declarados en un bloque, incluyendo el programa entero. Asi, ellos pueden ser redefinidos dentro del cuerpo del programa principal si es requerido.

El siguiente bosquejo de programa, ayudara a aclarar las reglas extendidas del Pascal.

```
PROGRAM ejemplo;
     VAR i,j : index;
     PROCEDURE uno;
          CONST j = 15;
          VAR x : real;
          PROCEDURE dos;
               VAR b : boolean;
          BEGIN
               {usa: b:boolean;
                     x:real;
                     j = 15;
                     i:index;
                     PROCEDURE ... ;
                     PROCEDURE dos}
          END; {dos}

     BEGIN
          {usa: j = 15;
                x:real;
                i:index;
                PROCEDURE ...
                PROCEDURE dos}
     END; {uno}
BEGIN
     {usa: i,j: index;
           PROCEDURE uno}
END.
```

RECURSION Y DIRECTIVA FORWARD

El bosquejo de programa de la unidad anterior, muestra que el nombre de un

procedimiento esta al alcance dentro del mismo procedimiento, es decir, el puede llamarse a si mismo. Un procedimiento el cual se llama a si mismo se dice a ser recursivo y tiene la forma

```
PROCEDURE a(x:t);
BEGIN
     ...
     a(p);
     ...
END;
```

La recursion puede tambien ser indirecta, por ejemplo 'a' llama a 'b' y 'b' llama a 'a'. Esto presenta un problema, puesto que un procedimiento puede ser llamado luego de haber sido declarado. Pascal provee una directiva FORWARD, la cual permite a la cabecera del procedimiento a ser separada del cuerpo del procedimiento. Por ejemplo:

```
PROCEDURE b(x:t); FORWARD;
PROCEDURE a(x:t);
BEGIN
     ...
     b(q)
END;
PROCEDURE b;
BEGIN
     ...
     a(p);
     ...
END;
```

Aqui el procedimiento 'a' llama a 'b' antes de que el sea declarado. Sin embargo, el compilador conoce acerca de 'b' porque la cabecera del procedimiento de 'b' ha sido declarada anterior a 'a'. Note que luego que 'b' es actualmente definido, sus parametros no son dados otra vez. Es buena practica no obstante, incluir los parametros, no importa como, dentro de las llaves de comentarios para ahorrar al lector tener que buscar hacia atras por medio de listados para encontrar la declaracion FORWARD.

La directiva FORWARD permite un mejor ordenamiento libre de procedimientos dentro de un programa. Toma como limite, todos los procedimientos que pudieron ser declarados FORWARD al inicio de un programa y luego la definicion de sus parametros actuales sera dada en algun orden. Una declaracion FORWARD y la actual definicion del procedimiento deben ser del mismo nivel logico y de la misma ex
tensión. [illegible]
una declaracion FORWARD y luego declarar realmente los procedimientos necesitados fuera del pro[illegible]uyente.

FUNCIONES

Un procedimiento puede ser usado para retornar cualquier numero de resultados via sus parametros VAR. Pascal tambien provee una clase de llamada de procedimiento nombrada FUNCTION, la cual retorna un resultado simple no via parametros sino como el valor de la funcion misma. Funciones han sido encontradas en la forma de funciones estandares del Pascal, por ejemplo

```
b := odd(i)
```

'odd' es una funcion que toma un parametro simple de tipo entero y retorna un resultado booleano. Una funcion es llamada escribiendo el nombre de la funcion como un componente de una expresion.

En programador puede declarar sus porpias funciones de la misma manera que declara sus propios procedimientos. La sintaxis de una declaracion de funcion es

declaracion de funcion

---> cabecera --> : --> identificador de tipo --> ; --> bloque --> ; -->

Una declaracion de funcion es similar a una declaracion de procedimiento excepto que un tipo debe ser especificado para el valor retornado por la funcion.

Dentro de una declaracion de funcion, un valor debe ser dado al nombre de la funcion, usualmente por una sentencia de asignacion. Como un ejemplo, la siguiente funcion pudo ser definida para retornar el valor maximo de dos enteros

```
FUNCTION max(i,j:integer):integer;
BEGIN
     IF i > j THEN max := i ELSE max := j
END;
```

Una sentencia de la forma

```
k := max(3,5) + max(2,9)
```

resultara en la funcion 'max' siendo llamada dos veces retornando valores 5 y 9, respectivamente, asi da a k el valor de 14. Es importante recalcar que dentro de una declaracion de funcion, el nombre de la funcion debe normalmente solo ser usado en el lado izquierdo de las sentencias de asignacion. Escribiendo el nombre como [illegible] una expresion constituye una llamada recursiva de la [illegible]

Si en el proceso de evaluacion de una funcion, esa funcion altera valores

fuera de su propio ambiente, luego la función [illegible] efectos por todos lados.

Efectos por todos lados pueden ocurrir por permitir funciones que tengan parametros VAR o mas insidiosamente, al permitir que una funcion asigne valores a variables no locales. Por ejemplo, considere el siguiente programa

```
PROGRAM efectosmultiples;
VAR a,b:integer;
FUNCTION sum(x,y:integer):integer;
BEGIN
     a := a + 1;
     sum := x + y
BEGIN
     a := 1;
     b := 1;
     b := sum(b,4) + a
END.
```

donde el valor asignado a 'b' puede ser 6 o 7, dependiendo del orden en el cual el compilador decida evaluar los terminos de la expresion

```
sum(b,4) + a
```

efectos multiples son peligrosos y deben ser evitados.

FUNCIONES Y PROCEDIMIENTOS COMO PARAMETROS

Si bien no es una facilidad usada comunmente, desde luego en muchos compiladores aun no lo implementan, Pascal estandard permite que nombres de procedimientos y funciones sean pasados como parametros a un procedimiento o funcion. Estos procedimientos y funciones pasados deben sin embargo, solo tomar parametros por valor ellos mismos.

Si una funcion va a ser pasada como un parametro, luego el tipo del resultado de la funcion debe ser especificado. Sin embargo, los parametros y sus tipos no son explicitamente declarados para las funciones y procedimientos parametros.

Como un ejemplo, la siguiente funcion retornara el valor de alguna funcion real f(x), integrado sobre los intervalos x = a hasta x = b

```
FUNCTION integral(FUNCTION f:real; a,b:real):real;
CONST numerodeintervalos = 12;
VAR tamannointervalo, sum:real;
    indice:integer;
BEGIN
```

```
        tamannointervalo := (b - a) / numerodeintervalos;
        sum := (f(a) + f(b)) / 2;

        FOR indice := 1 TO numerodeintervalos - 1 DO
             sum := sum + f(a + indice * tamannointervalo);
        integral := sum * tamannointervalo
    END;
```

Procedimientos y funciones como parametros solo son realmente de gran uso en [illegible]. Estos constituyen una caracteristica peligrosa del lenguaje ya que la mayoria de los compiladores no chequean que la funcion o procedimiento actual suministrado en el punto de llamado tenga los apropiados tipos de parametros.

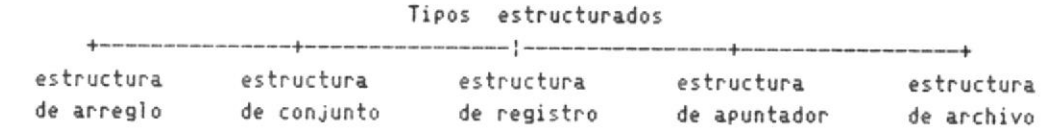
Otro problema con procedimientos y funciones como parametros, es que ellos hacen que la operacion de un programa oculte especialmente cuando los procedimientos y funciones actuales, suministrados en la ejecucion, manipulan variables globales.

En conclusion, procedimientos y funciones como parametros son mejor evitarlos en lo posible.

Cap. 7 DATOS DE TIPOS ESTRUCTURADOS

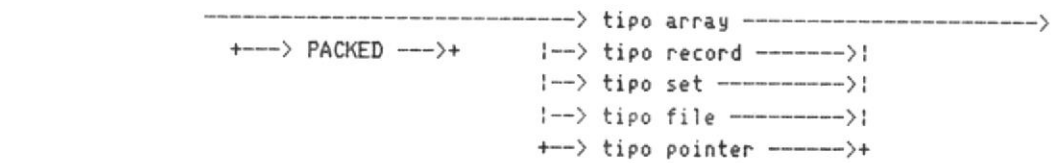
TIPOS ESTRUCTURADOS

Asi como las sentencias estructuradas son composiciones de otras sentencias, los tipos estructurados son composiciones de otros tipos. Un tipo estructurado es caracterizado por un metodo de estructuracion y los tipos de sus componentes. Pascal provee cinco metodos de estructuracion como se muestra en el siguiente diagrama



La sintaxis general de tipos estructurados es mostrada mas adelante, y en ella se puede observar que el uso de la palabra reservada PACKED, para prefijar un tipo estructurado, indica que alguna optimizacion de espacio de memoria, dependiente de la implementacion, va a ser ejecutada para todas las variables de ese tipo.

tipo estructurado



TIPO SET

Es a menudo util agrupar juntos un numero de objetos y poder referenciar a ellos [illegible] es llamada un conjunto (SET). El tipo set de Pascal, implementa esta fundamental idea matematica. El diagrama de sintaxis para un [illegible]:

tipo set

Como un ejemplo, el tipo 'color' puede ser definido como un conjunto de colores primarios como sigue:

```
TYPE colorprimario = (rojo, verde, azul);
     color = SET OF colorprimario;
```

Variables de tipo 'color' pueden ser declaradas y se les pueden asignar valores, los cuales pueden ser definidos por 'valores set'. Por lo tanto, dando las declaraciones:

```
VAR colorrojo, uncolor, colorblanco: color;
```

los valores posibles para estas variables 'color' son descritos por los siguientes valores set:

```
colorrojo := [rojo];
uncolor := [verde, azul];
colorblanco := [azul, verde, rojo];
```

La sintaxis de un valor set es como sigue:

valor set

El tipo de los componentes de un tipo set es llamado su tipo base y debe ser un tipo simple (por ejemplo, el tipo base de 'color' es 'colorprimario', el cual es un tipo enumerado simple). Solo los valores que son elementos del tipo base pueden ocurrir en un valor set. Aquellos valores que ocurren son llamados miembros de esa variable. Por lo tanto 'rojo' es un miembro de 'colorrojo' como de 'colorblanco'. Una variable set puede no tener miembros en ella. Su valor es luego representado por el valor de set vacio, escrito [].

Ejemplo: colornegro := [];

La forma de valor set

```
[expresion .. expresion]
```

permite la especificacion de un rango de valores como miembros de un set. El rango debe ser un subrango propio del tipo base.

Ejemplo: colorblanco := [rojo .. azul];

Los operadores relacionales estandares definidos para conjuntos son:

= igual

<> no igual

<= menor o igual que

>= mayor o igual que

IN miembro del set

Estos operadores solo son definidos entre conjuntos con el mismo tipo base.

Dos conjuntos son iguales si ambos tienen los mismos miembros. Por lo tanto

```
uncolor = [verde, azul]  es verdadero
uncolor = [azul, verde]  es verdadero
     [] = colornegro     es verdadero
uncolor <= colorblanco   es verdadero
 [rojo] >= colorblanco   es falso
```

El primer operando de IN debe ser un valor del tipo base del segundo operando de tipo set.

```
rojo IN [rojo, azul] es verdadero
rojo IN uncolor      es falso
```

Los otros operadores estandares definidos para conjuntos son:

```
+ union de conjunto
* interseccion de conjunto
- diferencia de conjunto
```

La union de dos conjuntos es un nuevo conjunto cuyos miembros son los miembros de ambos conjuntos originales.

```
[rojo, verde] + [azul] = [rojo, verde, azul] es verdadero
uncolor + [rojo] = colorblanco               es verdadero
```

La interseccion de dos conjuntos es un nuevo conjunto cuyos miembros son solo aquellos valores los cuales fueron miembros de ambos conjuntos originales.

```
[rojo, verde] * [azul, rojo] = [rojo] es verdadero
uncolor * [] = []                     es verdadero
```

La diferencia de dos conjuntos es un nuevo conjunto cuyos miembros son miembros del primer operando, pero no miembros del segundo.

```
[rojo, verde, azul] - [verde, azul] = [rojo] es verdadero
colorblanco - [] = colorblanco               es verdadero
```

Desde un punto de vista practico, es importante notar que la mayoria de las implementaciones de Pascal restringen el numero maximo de miembros que un conjunto pueda tener. Tipos tales como

```
SET OF integer        o        SET OF char
```

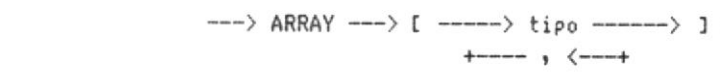
no es permitido.

No hay un orden asociado con los elementos de un conjunto, asi no es conve-
niente pregu[illegible] algo como 'cual es el segundo [illegible]'.
Sin embargo, es a menudo deseable asociar algunos ordenamientos con un grupo de objetos de datos y Pascal provee el tipo estructurado ARRAY para este proposito.

TIPO ARRAY

Al igual que los tipos set, los tipos ARRAY permiten un numero de objetos de datos a ser agrupados juntos bajo un nombre colectivo simple. A diferencia de los tipos set sin embargo, los objetos en una agrupacion array son considerados a estar ordenados de alguna forma. Asi podemos preguntar algo como 'cual es el septimo numero del arreglo'. La sintaxis para un tipo array es:

tipo array



Como un ejemplo, el tipo 'espectro' el cual agrupa las intensidades de los colores primarios escritos como alguna fuente de luz en orden ascendente por frecuencia, pudo ser definido como sigue:

```
TYPE intensidad = (cero, baja, media, alta);
     colorprimario = (rojo, verde, azul);
     espectro = ARRAY [colorprimario] OF intensidad;
```

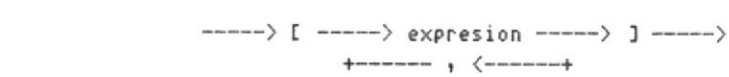
En este caso el ordenamiento de las intensidades en el espectro es el mismo que el de los valores del tipo indice 'colorprimario'. El tipo indice debe ser un tipo simple, pero el tipo componente ('intensidad' en el ejemplo de 'espectro') puede generalmente ser cualquier tipo del Pascal.

Variables de un tipo array pueden ser declaradas de la manera usual

```
VAR luzdelsol : espectro;
```

Un mapeo particular (o busqueda de informacion en una tabla representada por una variable array) es indicado en Pascal agregando un acceso del arreglo, con la sintaxis dada abajo, al nombre de una variable array.

acceso de arreglo



La expresion debe producir un valor el cual es un elemento del tipo indice del arreglo a ser accesado. Por lo tanto:

```
luzdelsol[verde]
```

es un

```
luzdelsol[2]
```

no lo es. El resultado de un acceso de arreglo es una variable con el mismo tipo asociado al tipo componente del arreglo. Todas las operaciones definidas para el tipo componente del arreglo pueden ser usadas con un componente del arreglo accesado en esta forma. En particular una variable de arreglo puede ser inicializada por asignamiento a sus componentes individuales.

```
luzdelsol[rojo] := media;
luzdelsol[verde] := alta;
luzdelsol[azul] := baja;
```

Despues de estas asignaciones las siguientes relaciones son todas verdaderas

```
luzdelsol[rojo] < luzdelsol[verde]
luzdelsol[verde] > luezdelsol[azul]
ord( luzdelsol[verde] )  =  1
```

Variables de arreglo del mismo tipo pueden ser asignadas a entre si. Dando otra variable del tipo 'espectro' llamada luzdelsol1 la siguiente asignacion

a) `luzdelsol1 := luzdelsol`

es equivalente a las tres asignaciones de siguientes:

b)
```
luzdelsol1[rojo] := luezdelsol[rojo];
luzdelsol1[verde] := luzdelsol[verde];
luzdelsol1[azul] := luzdelsol[azul];
```

No obstante, otra forma de conseguir las mismas series de asignamientos es usar una sentencia FOR como sigue:

c)
```
VAR i: colorprimario;
    .
    .
FOR i := rojo TO azul DO
     luzdelsol1[i] := luzdelsol[i];
```

Esto ilustra un uso comun de la sentencia for en manipulacion de variables de arreglo. Si bien a, b y c logran el mismo efecto, en este caso las tres tec-

nicas generalmente seran usadas bajo diferentes circunstancias.

El metodo a) es la manera mas facil de inicializar todos los componentes de un arreglo a los valores de los componentes correspondientes de otro.

El metodo b) es generalmente la manera mas lenta de lograr el mismo fin, pero es util si solo un subconjunto casual de valores de componentes van a ser copiados desde un arreglo a otro.

El metodo c) es el mas util cuando un subconjunto ordenado de valores de componentes van a ser copiados, como

```
VAR a, b: ARRAY [1..100] OF integer;
    i: 3..25;
     .
     .
FOR i := 3 TO 25 DO a[i] := b[i];
```

PROCEDIMIENTOS Y FUNCIONES USANDO ARREGLOS Y CONJUNTOS

Variables de tipo set y array pueden ser usadas libremente dentro de procedimientos y funciones. Ellas tambien pueden ser pasadas bien como parametros por valor o parametros variables. La unica restriccion sobre su uso es que una funcion no puede retornar un resultado el cual es bien un valor tipo set o array

La funcion ejemplo dada abajo puede ser usada para encontrar el valor maximo en un conjunto de enteros con valores entre 0 y 100.

```
TYPE enteros = 0 .. 100;
     setenteros = SET OF enteros;
FUNCTION maxenteros(s: setenteros): enteros;
VAR max, i: enteros;
BEGIN
     max := 0;
     IF s = [] THEN (resultado indefinido)
     ELSE FOR i := 0 TO 100 DO
          IF (i IN s) AND (i > max) THEN max := i;
     maxenteros := max
END;
```

Note que el resultado de 'maxenteros([])' es indefinido. Un programa usando esta funcion debe chequear que un conjunto no este vacio antes de aplicarle la funcion.

El siguiente ejemplo ejecuta una funcion similar para un arreglo de 25 enteros en el rango de 0 a 100.

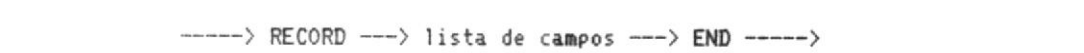
```
TYPE rangoenteros = 0 .. 100;
     arregloenteros = ARRAY [1 .. 25] OF rangoenteros;
FUNCTION maxenarreglo(a: arregloenteros): rangoenteros;
VAR max: rangoenteros;
    i: 1 .. 25;
BEGIN
     max := 0;
     FOR i := 1 TO 25 DO
          IF a[i] > max THEN max := a[i];
     maxenarreglo := max
END;
```

Note que el resultado de la funcion 'maxenarreglo' es siempre definido si todos los componentes del arreglo a los cuales es aplicado estan definidos. Esto es porque a diferencia de un conjunto, un arreglo nunca puede estar vacio.

TIPO RECORD

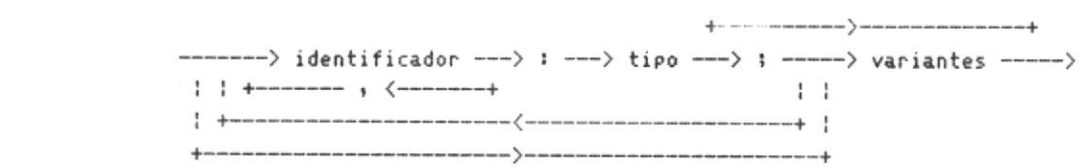
Los tipos array y set permiten objetos del mismo tipo a ser agrupados juntos y referenciados por un simple nombre colectivo. A menudo es deseable ser capaz de agrupar objetos juntos, de diferentes tipos, en una forma similar. Los tipos

tipo record



donde una lista de campos esta descrita por:

lista de campos



El diagrama de variantes sera dado mas tarde.

Como un ejemplo, el tipo persona puede ser definido como el siguiente tipo registro.

```
TYPE persona = RECORD
                    edad: integer;
```

```
                    casado: boolean;
                    sexo: (masculino, femenino)
                END;
```

Los identificadores 'edad', 'casado' y 'sexo', denotan los campos del registro persona y pueden generalmente ser de cualquier tipo. Variables de tipo persona pueden ser declaradas y se les pueden asignar los valores de otras variables registro, del mismo tipo. Por lo tanto, dando las declaraciones:

```
VAR adan, eva: persona;
```

la siguiente sentencia de asignacion esta en orden

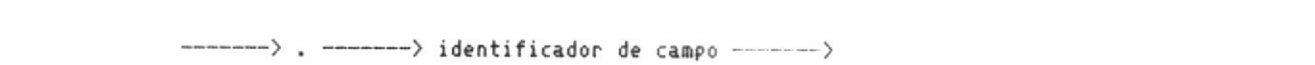
```
adan := eva
```

Asi como se les asignan valores a registros enteros, los campos individuales de un registro pueden ser accesados separadamente. Una variable representando un campo simple de un registro, es denotado al usar el operador de seleccion de campo, escrito como un punto. Ejemplo:

```
adan.sexo := masculino;
eva.sexo := femenino;
adan.edad := 25;
adan.casado := falso;
```

El diagrama de un campo esta dado por

acceso de campo



Si una variable de algun tipo de registro tiene un gran numero de campos puede ser tedioso tener que escribir el nombre de esa variable cada vez que un campo es accesado.

Ejemplo:

```
(inicializar adan)
adan.edad := 25;
adan.sexo := masculino;
```

La sentencia WITH es provista en Pascal para simplificar esta clasificacion de inicializacion.

SENTENCIA WITH

La sintaxis completa de la sentencia WITH esta dada por

sentencia with

La extension de una variable denotada por un nombre de campo es la sentencia mas interna siguiente al DO de una sentencia WITH, mencionando una variable registro con un campo del mismo nombre.

```
WITH adan DO
BEGIN
     adan := 25;
     WITH eva DO edad := 17
END;
```

Cuando mas de una variable aparece en una simple 'lista with', la sentencia with simple es equivalente a una serie de sentencias with necesitadas como sigue

```
WITH r1, r2, r3 .... , rn DO sentencia
```

es equivalente a

```
WITH r1 DO
     WITH r2 DO
          ...
          WITH rn DO sentencia
```

Al usar esta clase de sentencia with gran cuidado debe tenerse para asegurar que cualquier campo que esta siendo accesado es en efecto el correcto. Como los nombres de los campos no son mencionados en lista with, es muy facil forzar a que la variable mencionada en una sentenica with interna tenga un campo con el mismo nombre, como a la que un acceso fue intentado.

TIPO DE REGISTRO VARIANTE

A menudo es necesario para un tipo de registro tener algunos campos extras, el numero y tipos de los cuales depende el valor de algun otro campo dentro del registro. Estos campos extras y el campo sobre cuyo valor depende su estructura pueden ser definidos como los variantes de un tipo record. La sintaxis de una parte variante como se define abajo, completa la sintaxis de una lista de campos dada anteriormente.

variante

lista

Como un ejemplo del uso de registros variantes, considere la siguiente definicion extendida del tipo persona:

```
TYPE genero = (masculino, femenino);
     persona = RECORD
                    edad: integer;
                    casado: boolean;
                    CASE sexo: genero OF
                         masculino: (pecho, cintura, pierna: integer);
                         femenino: (busto, cintura, cadera: integer)
               END;
```

En este ejemplo, el campo original 'sexo' se ha convertido en lo que es llamado el campo etiqueta del registro variante 'persona'. Es el valor de este campo el que determina la estructura de una variable de tipo persona. El tipo de un campo etiqueta debe ser un tipo enumerado o subrango.

Solo los campos asociados con el valor del campo etiqueta actual de un registro variante puede ser accesado a la vez. Una restriccion mayor es que todos los nombres de campos en un registro, deben ser unicos, aun si ellos ocurren en diferentes variantes. Cuando el valor de un campo etiqueta es cambiado debe ser asumido que toda la informacion mantenida por los campos de la variante previa es perdida.

REPRESENTACION EMPAQUETADA DE ESTRUCTURAS DE DATOS (PACKED)

Cada variable declarada en un programa Pascal toma hasta una cierta cantidad de espacio en la memoria del computador que ejecuta el programa. La cantidad exacta de espacio ocupado por cualquier variable, depende de su tipo y del computador usado. En general tipos estructurados ocupan mas espacio que los tipos escalar; y tipos estructurados con muchos componentes, toman hasta mas espacio que los tipos estructurados con pocos componentes.

En algunas situaciones es importante para una variable, tomar el menor espa-

cio como sea posible. Esto puede ser comunicado a la implementacion prefijando a [illegible]

```
TYPE palabra = PACKED ARRA[illegible]
```

La cantidad exacta de espacio de memoria salvada es dependiente de la implementacion.

Hay arrays [illegible]

```
TYPE nstring = PACKED ARRAY [1 .. n] OF char
```

Variables de este tipo son llamadas variables string y pueden tener asignados valores indicados por constantes, de la siguiente forma:

```
'c1 c2 c3 ..... ci ..... cn'
```

donde cada ci es un caracter imprimible.

Ejemplo:

```
VAR s : PACKED ARRAY [1..6] OF char;
BEGIN
     s := 'monday';
       .
       .
     write(s, 'morning')
END
```

Este programa ejemplo escribira la cadena de caracteres 'monday morning'.

El tipo de una constante string es PACKED ARRAY [1..n] OF char. Donde n es el numero de caracteres en el string (cadena). Note que s:= 'junio' sera ilegal en el ejemplo previo, ya que 'junio' es un PACKED ARRAY [1..5] OF char y s es un PACKED ARRAY [1..6] OF char. Sin embargo, s:= 'junio ' sera correcto.

Los operadores relacionales estandard definidos para cadenas de caracteres es como sigue:

```
 = cadenas iguales
<> cadenas diferentes
 < Relaciones de ordenamiento
 > lexicografico. El orden es
<= determinado por el ordenamiento
>= del tipo char.
```

Ejemplos:

```
'abcd'    =    'abcd'    es verdadero
'abcd'    <>   'defg'    es verdadero
'abcd'    <    'abce'    es verdadero
'abcd'    >    'abcc'    es verdadero
```

Cap. 8 ESTRUCTURAS DINAMICAS DE DATOS

CONCEPTOS BASICOS

Estructuras dinamicas de datos, son estructuras de datos que 'surgen' cuando un programa se ejecuta. Una estructura dinamica de datos es una coleccion de elementos (llamados nodos) que normalmente son registros. A diferencia de un arreglo que siempre contiene un espacio de almacenamiento para un numero fijo de elementos, una estructura dinamica de datos se expande y contrae durante la ejecucion de un programa, basada en los requerimientos de almacenamiento de datos del programa.

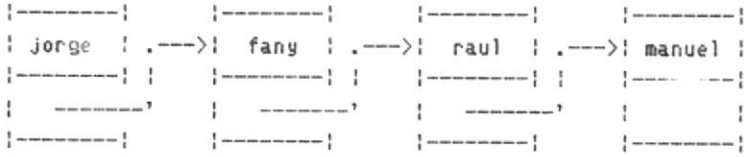
palabra real, que estan constantemente cambiando.

Estructuras dinamicas de datos son extremadamente flexibles. Es extremadamente facil annadir nueva informacion al crear un nuevo nodo e insertarlo entre dos nodos existentes. Es tambien relativamente facil modificar estructuras dinamicas de datos removiendo o eliminando un nodo existente. Es mas conveniente que modificar un arreglo de registros, donde cada registro esta en una posicion relativa, fijado a los otros, ya que es determinado por su suscrito.

Como un ejemplo suponga que se requiere suministrar una lista desconocida de nombres de personas y almacenarla de tal forma que sea facil agregar nombres nuevos a la lista y eliminar los nombres viejos. Cada nombre puede ser almacenado en un registro declarado como

```
TYPE item = RECORD
              nombre: string;
              proximo: apuntadordelitem
            END;
```

Una estructura de dato de tipo 'item' es un registro con dos componentes. El primer componente tiene un nombre de persona y el segundo componente puede tener una referencia a otro item. Asi, una lista de nombres de personas puede ser construida enlazando registros de tipo 'item' a una cadena. Esta puede ser representada graficamente como sigue



El tipo 'apuntadordelitem' usado para enlazar los registros es un tipo apuntador del Pascal. Las caracteristicas del lenguaje concernientes a apuntadores es descrita en la siguiente unidad.

APUNTADORES Y DATOS DE OBJETOS DINAMICOS

El tipo pointer (apuntador) es declarado con la siguiente sintaxis

tipo pointer

apuntador sennala. Asi, en el ejemplo de la unidad anterior, el tipo 'apuntadordelitem' pudo ser declarado como

```
TYPE apuntadordelitem = ↑item;
     item = RECORD
               nombre: string;
               proximo: apuntadordelitem
            END;
```

Note que el identificador 'item' es usado antes de que sea declarado. Este es el unico caso en Pascal donde esto es permitido, sin embargo, una definicion de 'item' debe ser dada en la misma declaracion de TYPE.

Debe ser enfatizado que el tipo 'apuntadordelitem' como se declaro anteriormente, puede apuntar a objetos de tipo 'item' y solo objetos de tipo 'item'. Asi, aun cuando objetos referenciados via variables pointer esten involucrados en asignaciones y algo semejante, las firmes reglas de representacion de Pascal sin embargo, se aplicaran.

A fin de referenciar un objeto de dato creado dinamicamente, al menos una variable apuntador localizada estaticamente es necesitada. Tales variables son declaradas en la forma usual, ejemplo:

```
VAR head, este: apuntadordelitem;
```

Un dato de objeto dinamico es creado por una llamada al procedimiento regular 'new' del Pascal, el cual toma un parametro simple que debe ser un tipo apuntador. La operacion de 'new' es mejor explicada con ejemplo. La llamada

```
new(head);
```

tiene dos efectos:

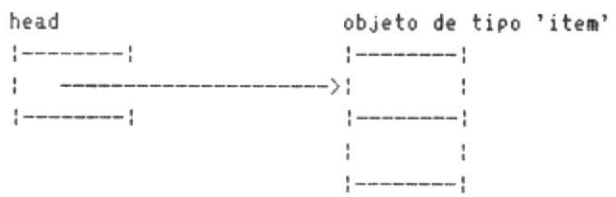
(a) Un ejemplo del tipo de dato apuntado por 'head' es creado.

(b) La variable 'head' es luego inicializada para apuntar a la estructura de dato creada recientemente.

Graficamente esto pudo ser representado como sigue

Antes de la llamada a 'new'

Despues de la llamada a 'new'

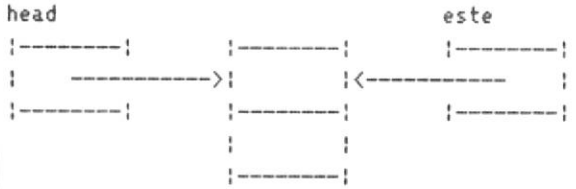


El dato [illegible] solo ser referenciado via un apuntador. Si en cualquier momento una condicion fue extendida donde ningun apuntador sennalaba a este objeto, luego el pudo ser totalmente inaccesible para la restante duracion del programa.

Tipos apuntadores pueden ser asignados como cualquier otra variable. Por ejemplo, el valor de 'head' pudo ser asignado a 'este' por

```
este := head;
```

Luego, dos apuntadores pueden estar apuntando al objeto 'item'



Accesar al dato del objeto mismo es denotado por una 'flecha hacia arriba' siguiente al apuntador, como muestra el siguiente diagrama de sintaxis

acceso del apuntador

asi head↑ se refiere al objeto de tipo 'item'

mientras que head se refiere al apuntador mismo

Tecnicamente, el '↑' es dicho a ser un 'operador de referenciacion'.

Procediendo con el ejemplo, el campo 'nombre' del objeto de dato puede ahora ser inicializado por

```
head↑.nombre := 'jorge';
```

Como ya no hay otro objeto siguiente para que el campo 'proximo' lo apunte, Pascal provee un identificador estandard 'NIL' el cual representa una referencia a ningun objeto en absoluto. NIL es tipo compatible con todos los tipos apuntadores. Asi, el campo 'proximo' del objeto de dato puede ser inicializado por

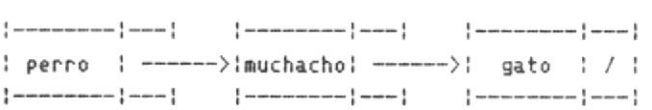
```
head↑.proximo := NIL;
```

La estructura de dato construida hasta aqui, esta ahora completamente inicializada y se parece a

de nombres' es seguido para ilustrar algunas de las tecnicas asociadas con el uso de estructuras dinamicas de datos.

LISTAS

Listas enlazadas son la forma basica comun de una estructura dinamica de datos. Una lista enlazada o simplemente lista, es una secuencia de nodos, en la cual cada nodo esta enlazado o conectado al nodo precedente a el. Un ejemplo de una lista enlazada con tres nodos es dada a continuacion.



Cada nodo en la lista tiene dos campos: el primer campo contiene dato y el segundo campo es un apuntador (representado por una flecha) al siguiente elemento de la lista.

El campo apuntador del ultimo elemento de la lista contiene el simbolo slash (/). Este simbolo es usado para denotar el fin de una lista. Un valor especial de apuntador NIL, es normalmente almacenado en el campo del apuntador del ulti-

mo elemento de la lista. NIL es una constante predefinida en Pascal.

A fin de representar una lista enlazada en memoria, es necesario definir un tipo registro con por lo menos un campo que sea apuntador.

Un nuevo elemento puede ser agregado al extremo de la estructura de lista desarrollada en la unidad anterior, por

```
new(este);
este↑.proximo := head;
este↑.nombre := 'susana';
head := este;
```

donde

Note que 'este' esta siendo usado como una variable apuntador general manteniendo referencias temporalmente mientras que la estructura de datos esta siendo manipulada. La variable 'head' sin embargo, esta siendo usada para un proposito

lista

le el nue

```
new(este↑.proximo);
este↑.proximo↑.nombre := 'simon';
este↑.proximo↑.proximo := NIL;
```

Note el uso sucesivo del operador de referenciacion en las dos ultimas sentencias. No hay restriccion sobre esto, ejemplo

```
head↑.nombre := 'susana';
head↑.proximo↑.nombre := 'jorge';
head↑.proximo↑.proximo↑.nombre := 'simon';
```

todas son validas. Sin embargo, el programador debe estar consciente que cada operacion de referenciacion incurre un tiempo de ejecucion superior (de acceso

de arreglo) y que en el ejemplo anterior

```
este := este↑.proximo;
este↑.nombre := 'simon';
este↑.proximo := NIL;
```

puede ser mejor, pero el mejor de todos es

```
WITH este↑.proximo↑ DO
BEGIN
     nombre := 'simon';
     proximo := NIL
END;
```

Donde el uso explicito de la sentencia WITH mejora la claridad de las operaciones de asignacion.

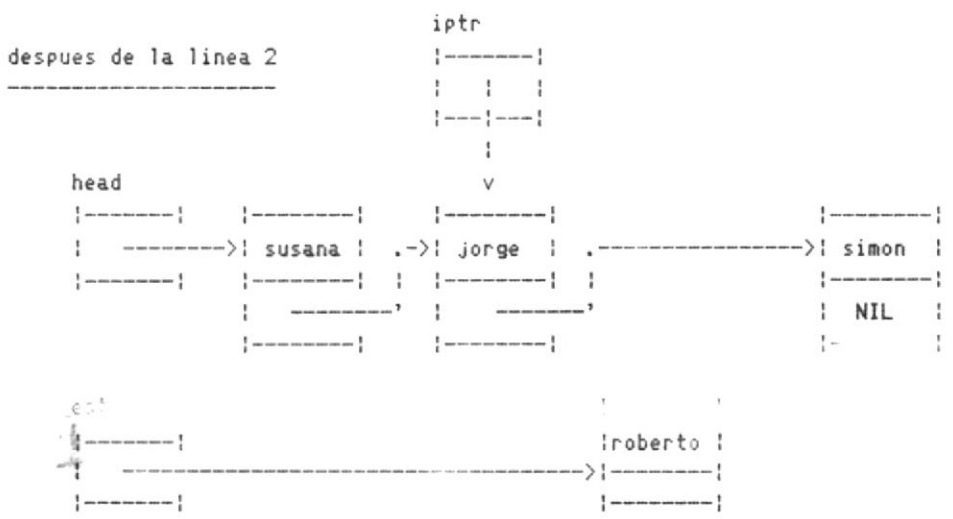
Un elemento puede ser insertado dentro de la lista como sigue. Suponga que una variable 'iptr' de tipo 'apuntadordelitem' ha sido puesta para apuntar al elemento que contiene el nombre 'jorge'.

Luego un nuevo nombre 'roberto' puede ser insertado despues de 'jorge' por

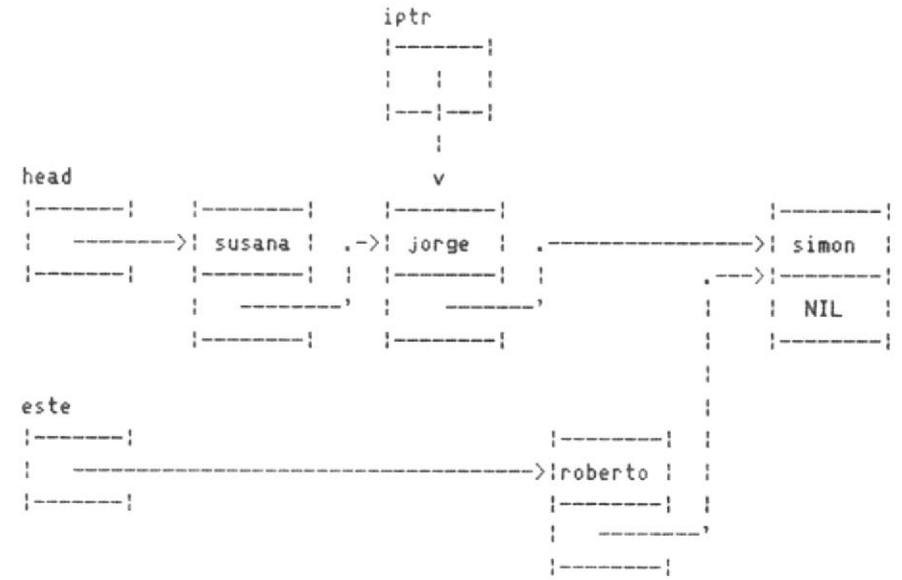
```
1.   new(este);

2.   este↑.nombre := 'roberto';
3.   este↑.proximo := iptr↑.proximo;
4.   iptr↑.proximo := este;
```

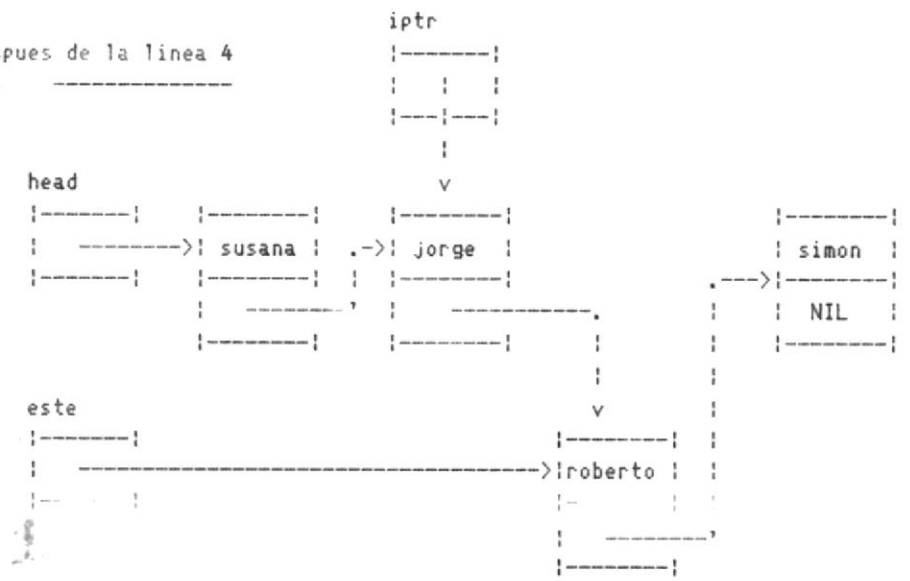
Graficamente las operaciones anteriores pueden ser representadas por



despues de la linea 3



despues de la linea 4



Este ultimo ejemplo ilustra la facilidad con que las estructuras dinamicas de datos pueden ser manipuladas, simplemente por reordenamiento de apuntadores.

ARBOLES BINARIOS

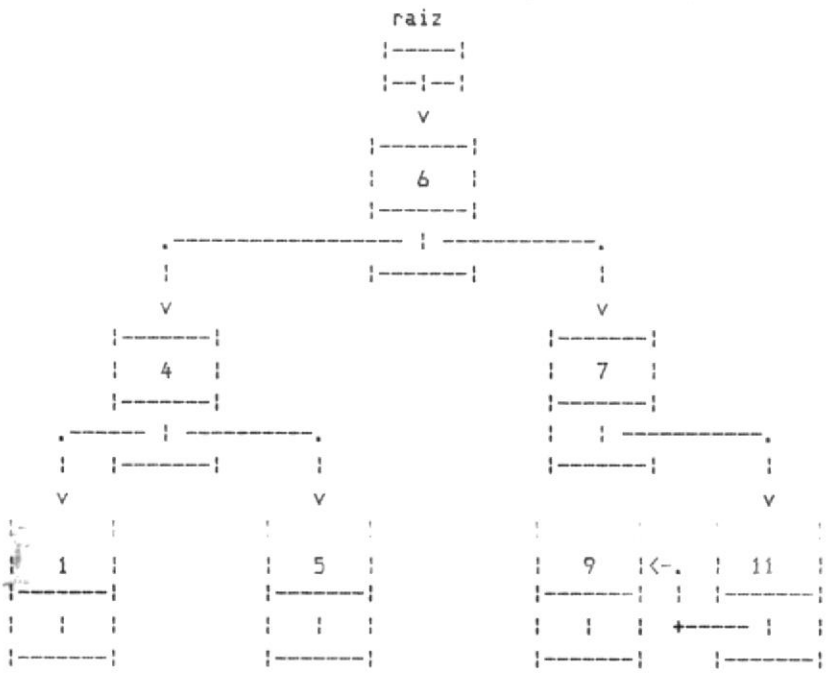
Otra estructura dinamica de datos usada comunmente es el arbol binario. Un bol binario consiste de un numero de nodos donde cada nodo puede tener por lo menos dos extensiones: una extension izquierda y una extension derecha. Un arbol binario es util para almacenar elementos para los cuales una relacion de ordenamiento existe, de tal forma que el numero de operaciones de comparacion que deben ser hechas para agregar un nuevo elemento a la estructura sea minimizado. Como un ejemplo, suponga que un conjunto de enteros van a ser almacenados en un arbol binario. Un nodo del arbol es definido por

```
TYPE enlacenodo = ↑nodo;
     nodo = RECORD
               entrada: integer;
               izquierdo, derecho: enlacenodo;
            END;
```

Cada nodo del arbol tiene la propiedad que si p es un apuntador al nodo luego

```
p↑.izquierdo↑.entrada <= p↑.entrada < p↑.derecho↑.entrada
```

Un tipico arbol binario es representado graficamente por



Como la propiedad de ordenamiento del arbol es esencialmente una definicion recursiva, los elementos pueden ser introducidos al arbol por una llamada a un procedimiento recursivo, asi

```
insertar(raiz,x)
```

doonde x es el entero a ser introducido e 'insertar' es definido como

```
PROCEDURE insertar(VAR p: enlacenodo; i: integer);
BEGIN
     IF p = NIL THEN
     BEGIN
          new(p);
          WITH p↑ DO
          BEGIN
               entrada := i; izquierdo := NIL; derecho := NIL
          END
     END
     ELSE IF p↑.entrada >= i
          THEN insertar(p↑.izquierdo, i)
          ELSE insertar(p↑.derecho, i)
END;
```

Estructuras de arbol tales como la anterior, pueden ser generalizadaa a configuraciones de muchas direcciones (es decir, mas de dos extensiones desde cada nodo) y tener un gran rango de aplicaciones en calculos no-numericos. Ademas de listas y arboles, las estructuras dinamicas de datos son comunmente usadas para representar varias clases de estructuras de grabados ordenados.

REGISTROS VARIANTES

Suponga que una estructura dinamica es declarada como sigue

```
TYPE enlace = ↑vreg;
     vreg = RECORD
               CASE tamanno: (largo, corto) OF
               corto: (campocorto: boolean);
               largo: (campolargo: ARRAY [1..1000] OF char)
            END;
VAR raiz: enlace;
BEGIN
     new(raiz);
      .
      .
      .   etc.
END.
```

En este caso, los datos de objetos dinamicos creados por llamadas a 'new', son de tamanno variante. Dependiendo del valor del campo etiqueta 'tamanno', el registro puede tener bien un simple

res.

A fin de ordenar uno u otro variante, la anterior llamada a 'new' causa que el espacio de almacenamiento a ser creado sea suficiente para mantener la mayor variante del registro. En el ejemplo anterior, esto pudo ser mucho derroche de espacio de almacenamiento, si la variante mayor fue solamente usada rara vez.

Si la variante actual requerida de una estructura de datos del registro es conocida cuando un ejemplo dinamico de ella es creado por una llamada a 'new', luego Pascal permite que los campos etiquetas sean especificados, ejemplo

```
new(p, t1, t2, t3, ....)
```

donde p es un apuntador a un registro variante con valores de campos de etiquetas t1, t2, t3, etc, declarados en ese orden. La especificacion de campos de etiquetas causa que 'new' distribuya solo almacenamiento suficiente para tener la variante particular especificada. De este modo por ejemplo,

```
new(raiz, corto)    =>      ¦  tamanno ¦       :(largo, corto)
                            ¦----------¦
                            ¦campocorto¦       :boolean
                            ¦----------¦

                            ¦----------¦
new(raiz, largo)    =>      ¦  tamanno ¦       :(largo, corto)
                            ¦----------¦
                            ¦   campo  ¦       :ARRAY [1..1000]
                            ¦   largo  ¦            OF char
                            ¦          ¦
                            ¦----------¦
```

Note que la especificacion de los valores del campo etiqueta no resultan en los correspondientes campos etiquetas de la estructura creada que esta siendo inicializada.

A fin de que los datos de objetos creados por esta forma extendida de 'new' sean usados apropiadamente, las siguientes reglas deben ser agregadas:

(i) Una vez especificada en la llamada a 'new' una variante, no debe subsecuentemente ser cambiada durante la ejecucion del programa.

(ii) A diferencia de los objetos creados por llamadas normales a 'new', los objetos [illegible] sentencias de asignacion completa, solo asignaciones a sus componentes son permitidas.

Asi por ejemplo, dado

```
VAR p, p: enlace;
```

luego

```
new(p); new(q); ....; p↑ := q↑;
```

es legal, mientras que

```
new(p, corto); new(q, corto); ....; p↑ := q↑;
```

no es legal aunque sea aparentemente significativo.

Estructuras dinamicas de datos variantes creadas por la segunda forma de new son muy peligrosas usar. La mayoria de las implementaciones hacen poca verificacion sobre el uso de tales objetos, ya que por su naturaleza son completamente inseguros.

Aqui los programadores estan advertidos para evitar esta caracteristica del lenguaje, al menos que sea absolutamente necesario.

Cap. 9 ARCHIVOS

GENERALIDADES

Algunos programas de un centro de proceso de datos estan relacionados entre si y esta relacion casi siempre consiste en archivos: creados por unos programas y usados (leidos y/o modificados) por otros programas.

Los archivos o conjuntos de datos tienen pues una existencia independiente de sus programas de procesoo.

Ejemplo:

Un programa editor crea y modifica un archivo cuyo contenido es el texto de un documento preparado para su edicion.

El programa debe reunir las siguientes caracteristicas:

- ser capaz de comenzar a formar el archivo desde el principio;
- annadir y borrar lineas, parrafos, palabras;
- introducir caracteres especiales (salto de pagina, espaciado, sangrado, etc.);
- listado de salida del documento.

El archivo usado es de entrada y de salida. En este ejemplo seria recomendable un proceso o funcionamiento interactivo con un terminal.

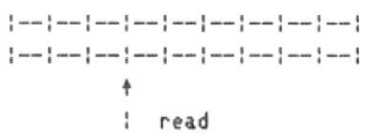
El archivo usado mas simple es el secuencial que esta formado por varios componentes del mismo tipo, uno despues de otro.

El archivo secuencial tiene como caracteristica esencial de su proceso, el que hay que recorrer componente a componente en secuencia. Si estamos al principio y queremos el componente 500, tenemos que pasar por todos los intermedios hasta llegar a el.

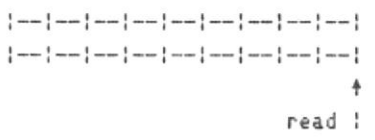
Un archivo secuencial es parecido a un arreglo, puesto que todos los componentes de un archivo son del mismo tipo, pero nada mas. En un arreglo, hacemos v[500] y tenemos sin ningun problema, el elemento 500, es decir, todos sus elementos son igualmente accesibles. Pero un archivo secuencial presupone un almacenamiento exterior y una manera secuencial de recorrerlo buscando el elemneto deseado.

La forma mas sencilla de representar un archivo secuencial es considerar el caso de una cinta magnetica en la que sus componentes van uno despues de otro y

mediante lecturas se llega a un cierto sitio del archivo en un instante dado:



en este esquema el archivo es de entrada y consta de 9 componentes y la flecha indica que ya se han leido 3 componentes y el siguiente read leeria el 4to componente. Si llegamos a leer el noveno componente (el ultimo), hemos agotado todo el archivo, se tiene entonces la condicion de fin de archivo (end of file) que hay que detectar para no tener un error.



Si el archivo fuera de salida, la situacion es parecida.

En el esquema se han escrito cinco componentes

si se escribe un componente mas, queda asi

ARCHIVOS

Un archivo [illegible] rada en el programa indicando que es un archivo y cual es su componente. Las variables de archivo se pueden usar como parametros de procedimientos, pero no de ninguna otra forma (p. ej. en la sentencia de asignacion). La sintaxis del tipo archivo es la siguiente:

tipo archivo

-----> FILE -----> OF -----> tipo ------>

Su definicion general es:

```
TYPE t = FILE OF to;
```

la cual expresa que cualquier archivo de tipo t consta de cero o mas componentes de tipo to.

Ejemplo:

```
TYPE item = RECORD
              nombre: PACKED ARRAY [1..longnom] OF char;
              descripcion: PACKED ARRAY [1..longdesc] OF char;
              codigo: char;
              existencia: integer
            END;
VAR
     maestro, detalle, nmaestro: FILE OF item;
```

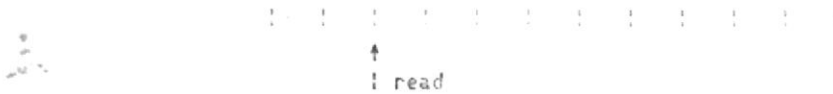
Un archivo se lee con un programa. Para ayudar al usuario, en el lenguaje hay dos procedimientos previos de entrada (es decir, ya definidos por el compilador Pascal):

reset(archivo): prepara el archivo para comenzar la lectura, posicionandose en el primer componente.

read(archivo, v): entrega el siguiente componente en la variable v y avanza la posicion del fichero. El tipo de v debe ser compatible con el tipo de componente del archivo.

Es decir, despues de ejecutarse el procedimiento reset el fichero queda:

La ejecucion del procedimiento read(archivo, v) se indica en el esquema siguiente (se han leido ya dos componentes):



Para leer un archivo se le debe posicionar llamando a reset. Solo en el caso del archivo estandard de entrada INPUT no debe darse el reset(INPUT), pues ya lo hace automaticamente el programa.

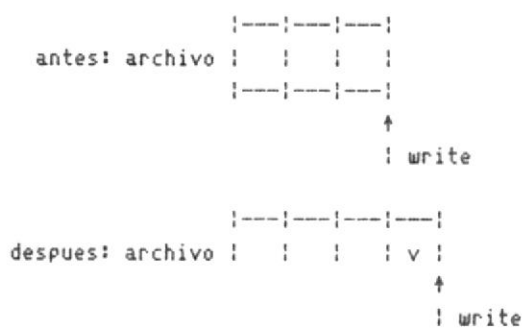
El procedimiento read fallara si el archivo ya se ha terminado de leer. Para prevenirlo existe la funcion logica EOF(archivo) que sera TRUE al agotarse el archivo.

Un archivo se escribe o crea escribiendo componente a componente. Tambien el Pascal tiene dos procedimientos previos de salida, ya definidos.

rewrite(archivo): prepara el archivo para comenzar a escribir, apuntando al principio; el archivo tiene ahora cero componentes.

write(archivo, v): annade al archivo el componente v, cuyo tipo debe ser compatible con el tipo de componente del archivo.

La accion del procedimiento write(archivo, v) es la siguiente:



Para escribir en un archivo se le debe preparar llamando al procedimiento rewrite. Solo en el caso del archivo estandard de salida OUTPUT no debe darse el rewrite(OUTPUT), pues ya lo hace automaticamente el programa.

Ejemplo:

Una empresa quiere obtener un archivo de las existencias que esten por debajo de un cierto umbral leido de un registro. Escribir un programa que extraiga del archivo de existencias aquellos elementos indicados, listando simultaneamente el valor calculado de los mismos.

```
PROGRAM extraer(vexis, nexis, input, output);
```

```
CONST
     longnom = 12;
     longdesc = 15;

TYPE item = RECORD
               nombre: PACKED ARRAY [1..longnom] OF char;
               desc: PACKED ARRAY [1..longdesc] OF char;
               costo: real; {unitario}
               existencia: integer
            END;

VAR
     vexis, nexis: FILE OF item;
     elemento: item;
     conta, umbral: integer;

BEGIN
     reset(vexis); {para comenzar lectura}
     rewrite(nexis); {para comenzar escritura}
     writeln('nombre descripcion valor exist');
     writeln('------------------------------');
     writeln;
     read(umbral);
     conta := 0;

     WHILE NOT eof(vexis) DO
          BEGIN
               read(vexis, elemento);
               WITH elemento DO
               IF existencia < umbral THEN
                    BEGIN
                         writeln(nombre, desc:8, (costo *
                                 existencia):12:2, existencia);
                         conta := conta + 1;
                         write(nexis, elemento)
                    END
          END; {de while}
     writeln('registros de salida =', conta)
END. {de extraer}
```

ARCHIVOS DE TEXTO

El Pascal tiene un tipo especial de archivos, cuyos componentes son caracteres, y que facilitan la comunicacion hombre-ordenador. Hay un tipo de archivo predefinido por el Pascal:

```
text = FILE OF char;
```

que se puede emplear como atributo de archivos de texto. Los archivos estandard del sistema son de este tipo y no hay que declararlos (ni inicializarlos con reset ni rewrite), pues ya estan declarados asi: input, output: text.

Los archivos de texto se pueden considerar como una tira de caracteres, en la que aparecen los finales de linea (que indicamos con @). Asi, este archivo de texto

```
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|'a'|'h'|'o'|'r'|'a'| @ |'y'|' '|'a'|'n'|'t'|'e'|'s'|
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
```

al imprimirlo daria

```
ahora
y antes
```

El caracter de fin de line (@) no es un caracter normal, pues no se puede le

El efecto de read(texto, c) en un final de linea es:

```
                  |---|---|---|---|---|---|     |---|
       antes: texto |   |   |   | @ |'a'|   |   c |   | eoln = true
                  |---|---|---|---|---|---|     |---|
                              ↑
                              | read

                  |---|---|---|---|---|---|     |---|
     despues: texto |   |   |   | @ |'a'|   |   c |' '| eoln = false
                  |---|---|---|---|---|---|     |---|
                                  ↑
                                  | read
```

El procedimiento readln(texto) mueve la posicion de lectura inmediatamente despues del siguiente caracter de fin de linea:

```
                |---|---|---|---|---|---|---|---|---|---|---|
     antes: texto |   |   |   |   |   |   |   | @ |   |   |
                |---|---|---|---|---|---|---|---|---|---|---|
                              ↑
                              | read

                |---|---|---|---|---|---|---|---|---|---|---|
   despues: texto |   |   |   |   |   |   |   | @ |   |   |
                |---|---|---|---|---|---|---|---|---|---|---|
```

```
                                                    ↑
                                                    ¦ read
```

Ejemplo:

Escribir un programa que cuente el numero de blancos y el de no-blancos en un archivo de texto.

```
PROGRAM cuenta(texto, output);

CONST
     blanco = ' ';

VAR
     texto: text;
     caracter: char;
     nb, nnb: integer;

BEGIN
     nb := 0;  nnb := 0;
     reset(texto);
     WHILE NOT eof(texto) DO
          BEGIN
               WHILE NOT eoln(texto) DO
                    BEGIN
                         read(texto, caracter);
                         IF caracter = blanco
                              THEN nb := nb + 1
                              ELSE nnb  nnb + 1
                    END;
               readln(texto) (ignora final de linea)
          END;
     writeln('numero de blancos: ',nb);
     w
END.
```

AREAS DE LECTURA Y ESCRITURA

El Pascal lee y escribe desde un area de memoria o 'buffer' (en la que cabe un componente) y a la que nosotros podemos acceder ya que el identificador es el nombre del archivo seguido de una flecha archivo↑. La operacion descrita anteriormente con la sentencia read(archivo, v), hacia estas dos cosas:

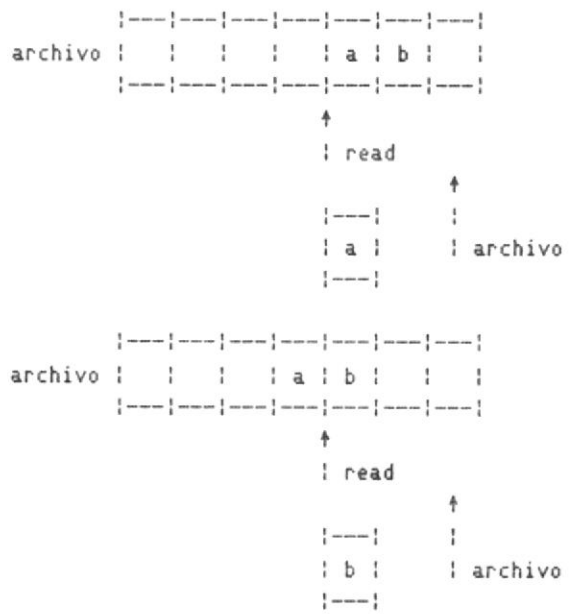
- copiaba el siguiente componente en v

- avanzaba la posicion de lectura

En algunos procesos, conviene separar el segundo paso del primero. Para tal proposito se utiliza el concepto de buffer.

En un archivo que se esta leyendo el buffer(archivo↑) contiene una copia del componente de archivo inmediatamente a la derecha de la posicion del archivo, se lo demuestra en el primer esquema de la siguiente pantalla (pagina).

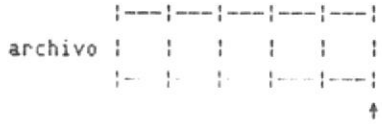
Si se quiere se puede usar el buffer directamente sin copiarlo en ningun sitio. Hay otro procedimiento auxiliar get(archivo) que avanza la posicion de lectura y actualiza el buffer, cuyo efecto sobre el archivo se muestra en el segundo esquema de la siguiente pantalla (pagina).

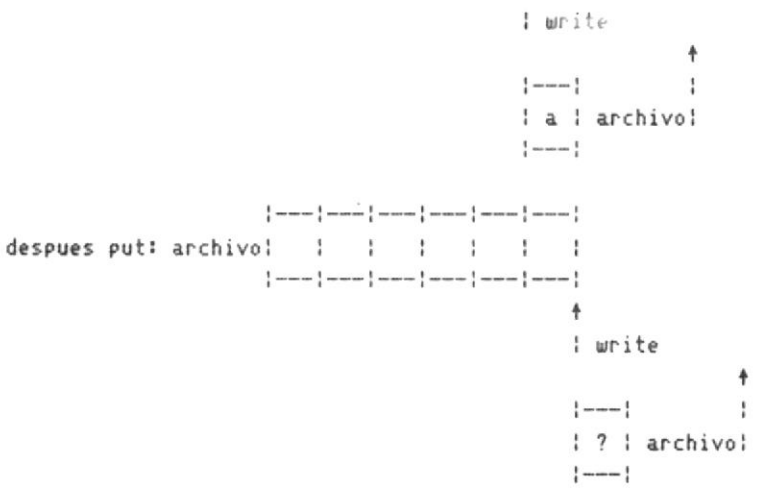


Es decir que read(archivo, v) equivale a:

```
BEGIN
     v := archivo↑;
     get(archivo)
END;
```

En un archivo de salida tambien se tiene un buffer en que se guarda el componente hasta que se pase al archivo o memoria auxiliar y se actualice la posicion, cosa que hace put(archivo)





El procedimiento write(archivo, v) equivale pues a:

```
BEGIN
     archivo↑ := v;
     put(archivo)
END;
```

APENDICE B

L I S T A D O S D E P R O G R A M A S

```
000100 IDENTIFICATION DIVISION.
000110 PROGRAM-ID. SINT010.
000120 AUTHOR.   FERNANDEZ Z. DEYDAMIA. D.
000130 INSTALLATION. RADIO SHACK.
000140
000150*    EL OBJETIVO DE ESTE PROGRAMA ES EL INGRESO DE DATOS Y EL
000160*    MANTENIMIENTO DEL ARCHIVO QUE CONTIENE EL TEXTO SOBRE LA
000170*    SINTAXIS DEL PASCAL.
000180
000190 ENVIRONMENT DIVISION.
000200 CONFIGURATION SECTION.
000210 SOURCE-COMPUTER. MODELII.
000220 OBJECT-COMPUTER. MODELII-64K.
000230 INPUT-OUTPUT SECTION.
000240 FILE-CONTROL.
000250     SELECT SINTF01 ASSIGN TO RANDOM "SINTF01"
000260                    ORGANIZATION IS INDEXED
000270                    ACCESS MODE IS DYNAMIC
000280                    RECORD KEY IS KEYCAP
000290                    STATUS IS ESTADO.
000300
000310 DATA DIVISION.
000320 FILE SECTION.
000330 FD  SINTF01
000340         RECORD CONTAINS 88 CHARACTERS
000350         BLOCK CONTAINS 2 RECORDS
000360         LABEL RECORDS ARE STANDARD.
000370 01  REG-01.
000380     03 KEYCAP.
000390        05 CAP           PIC 99.
000400        05 UNI           PIC 99.
000410        05 PANT          PIC 99.
                 III           PIC 99.
[illegible]
[illegible]
[illegible]
000460 77 LINEA                PIC 99.
000470 [illegible]              PIC XX.
000480 77 RES                  PIC X.
000490 77 OPC                  PIC 9 VALUE ZERO.
000500 77 AUX-CLAVE            PIC X(8) VALUE SPACES.
000510 77 SW                   PIC 9 VALUE ZERO.
000520 77 EOF                  PIC [illegible]
000530 77 CLAVE                PIC X(10).
000540
000550     COPY "CLAVE01/CBL.DF".
000560
```

```
000570 PROCEDURE DIVISION.
000580 DECLARATIVES.
000590 ERROR-SINTF01 SECTION.
000600     USE AFTER STANDARD EXCEPTION PROCEDURE ON SINTF01.
000610 END DECLARATIVES.
000620
000630 PRINCIPAL SECTION.
000640 1-INICIO.
000650     DISPLAY "*** S E G U R I D A D ***"
000660             LINE 10 POSITION 29 BEEP ERASE
000670             "========================="
000680             LINE 11 POSITION 29 SIZE 25
000690             "DIGITE CLAVE DE PROTECCION"
000700             LINE 13 POSITION 29.
000710     ACCEPT CLAVE OFF LINE 14 POSITION 36.
000720     IF CLAVE NOT = CLAVEF01
000730        DISPLAY "*** CLAVE ERRADA ***"
000740                LINE 13 POSITION 31 ERASE REVERSE BEEP
000750        GO TO 1-FIN.
000760
000770     OPEN I-O SINTF01.
000780     PERFORM 2-DISPLAY THRU 2-EXIT UNTIL OPC = 5.
000790     CLOSE SINTF01.
000800     DISPLAY "* FIN DEL PROGRAMA *"  ERASE BEEP
000810             LINE 15 POSITION 31 HIGH.
000820
000830 1-FIN.
000840     STOP RUN.
000850
000860 2-DISPLAY.
000870     DISPLAY "SINTAXIS DEL LENGUAJE PASCAL" LINE 2
000880              POSITION 27  ERASE BEEP
000890             "==========================" LINE 3
000900              POSITION 27
000910             "* MANTENIMIENTO DEL MAESTRO - TEXTO *"
000920              LINE 4 POSITION 22.
000930
000940 2-MENU.
000950     DISPLAY "1. INGRESO"      LINE 7 POSITION 34
000960             "2. MODIFICACION" LINE 8 POSITION 34
000970             "3. ELIMINACION"  LINE 9 POSITION 34
000980             "4. CONSULTA"     LINE 10 POSITION 34
000990             "5. FINALIZAR"    LINE 11 POSITION 34
001000             "OPCION -->"      LINE 13 POSITION 35.
001010
001020 2-ACEPTAR.
001030     ACCEPT OPC LINE 13 POSITION 46 PROMPT ECHO LOW.
```

```
001040     IF OPC = 5 GO TO 2-EXIT
001050       ELSE IF OPC < 1 OR OPC > 4 GO TO 2-ACEPTAR.
001060
001070 210.
001080     MOVE SPACES TO REG-01.
001090     IF OPC = 1 PERFORM 3-INGRESO THRU 3-EXIT
001100              UNTIL CAP = ZEROS
001110       ELSE IF OPC = 2 PERFORM 4-MODIFICACION THRU 4-EXIT
001120                       UNTIL CAP = ZEROS
001130         ELSE IF OPC = 3 PERFORM 5-ELIMINACION THRU 5-EXIT
001140                         UNTIL CAP = ZEROS
001150           ELSE PERFORM 6-CONSULTA THRU 6-EXIT
001160                UNTIL CAP = ZEROS.
001170     MOVE 99 TO CAP.
001180
001190 2-EXIT. EXIT.
001200
001210 3-INGRESO.
001220     PERFORM 2-DISPLAY.
001230     DISPLAY "* INGRESO *" LINE 1 POSITION 1 BEEP.
001240
001250 3-CLAVE.
001260     DISPLAY "CLAVE -->" LINE 6 POSITION 62.
001270
001280 3-RESTO.
001290     DISPLAY  "CONTENIDO" LINE 8 POSITION 36
001300              "1...5...10...15...20...25...30...35...40"
001310              LINE 9 POSITION 1
001320              "...45...50...55...60...65...70...75...80"
001330              LINE 9 POSITION 41.
001340
001350 310.
001360     DISPLAY "E
001370
001380 3-ACEPTAR.
001390     ACCEPT KEYCAP LINE 6 POSITION 72 PROMPT ECHO LOW.
001400
001410 320.
001420     IF CAP
001430     ELSE IF KEYCAP = SPACES OR KEYCAP IS
001440             GO TO 3-ACEPTAR.
001450
001460 330.
001470     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
001480     IF RES = "N" GO TO 3-ACEPTAR
001490       ELSE IF RES NOT = "S" GO TO 330.
001500
```

```
001510 340.
001520     ACCEPT CONTEN LINE 10 POSITION 1 PROMPT ECHO LOW.
001530
001540 350.
001550     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
001560     IF RES = "N" GO TO 340
001570       ELSE IF RES NOT = "S" GO TO 350.
001580
001590 3-GRABACION.
001600     WRITE REG-01.
001610     DISPLAY SPACES LINE 13 POSITION 5 SIZE 22.
001620     IF ESTADO = "22" DISPLAY "* CLAVE YA EXISTE *"
001630                              LINE 23 POSITION 31 BEEP
001640     ELSE IF ESTADO = "98" DISPLAY "* CLAVE INVALIDA *"
001650                                   LINE 23 POSITION 31 BEEP
001660     ELSE IF ESTADO NOT = "00" DISPLAY "* ERROR EN OP. WRITE *"
001670                                       LINE 23 POSITION 28.
001680     IF ESTADO NOT = "00" ACCEPT RES OFF LINE 23 POSITION 50.
001690     MOVE SPACES TO REG-01.
001700
001710 3-EXIT. EXIT.
001720
001730 4-MODIFICACION.
001740     PERFORM 2-DISPLAY.
001750     DISPLAY "* MODIFICACION *" LINE 1 POSITION 1 BEEP.
001760     PERFORM 3-CLAVE.
001770
001780 4-DISPLAY.
001790     PERFORM 310.
001800
001810 4-ACEPTAR.
001820     PERFORM 3-ACEPTAR.
001830
001840 410.
001850     IF CAP = ZEROS GO TO 4-EXIT
001860     ELSE IF KEYCAP = SPACES OR KEYCAP IS NOT NUMERIC
001870             GO TO 4-ACEPTAR.
001880
001890 420.
001900     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
001910     IF RES = "N" GO TO 4-ACEPTAR
001920       ELSE IF RES NOT = "S" GO TO 420.
001930
001940 4-LECTURA.
001950     READ SINTF01 INVALID KEY
001960                  DISPLAY SPACES LINE 13 POSITION 5 SIZE 22
001970                          "* REGISTRO NO EXISTE *"
```

```
001980                          LINE 23 POSITION 31 BEEP
001990                  ACCEPT RES OFF LINE 23 POSITION 53
002000                              GO TO 4-ENCERE.
002010
002020 4-RESTO.
002030     DISPLAY "1." LINE 6 POSITION 60
002040             "2." LINE 8 POSITION 34.
002050     PERFORM 3-RESTO.
002060
002070 430.
002080     DISPLAY CONTEN LINE 10 POSITION 1 LOW.
002090
002100 4-CORRECCION.
002110     DISPLAY SPACES LINE 13 POSITION 1 SIZE 70
002120             "CAMPO A MODIFICAR? (1,2) 0 <ENTER>"
002130             LINE 13 POSITION 23 BEEP.
002140     ACCEPT RES LINE 13 POSITION 58 PROMPT ECHO LOW.
002150     DISPLAY SPACES LINE 13 POSITION 23 SIZE 38.
002160     IF  RES = SPACE GO TO 4-GRABACION
002170       ELSE IF RES = "1" PERFORM 4-MOD-CLAVE
002180         ELSE IF RES NOT = "2" GO TO 4-CORRECCION
002190           ELSE PERFORM 4-MOD-CONTEN THRU 450.
002200     GO TO 4-CORRECCION.
002210
002220 4-GRABACION.
002230     DISPLAY "REGRABA O CORRIGE? (R,C) O <ENTER>"
002240             LINE 13 POSITION 23 BEEP.
002250
002260 440.
002270     ACCEPT RES LINE 13 POSITION 58 [illegible]
                                          [illegible]
[illegible]
002300         ELSE IF RES NOT = "R" GO TO 440.
002310     DISPLAY SPACES LINE 13 POSITION 23 SIZE 36.
002320     IF AUX-CLAVE NOT = KEYCAP
002330        IF AUX-CLAVE = SPACES NEXT SENTENCE
002340          ELSE PERFORM 4-REGRABAR THRU 460-EXIT [illegible]
002350     REWRITE REG-01 INVALID KEY
002360                    DISPLAY "* ERROR AL MODIFICAR REGISTRO *"
002370                            LINE 23 POSITION 25 BEEP
002380                    ACCEPT RES OFF LINE 23 POSITION 56
002390                    GO TO 4-ENCERE.
002400     DISPLAY "* REGISTRO MODIFICADO *" LINE 23 POSITION 29 BEEP.
002410     ACCEPT RES OFF LINE 23 POSITION 52.
002420
002430 4-ENCERE.
002440     MOVE SPACES TO AUX-CLAVE.
```

```
002450     MOVE ZERO TO SW.
002460
002470 4-EXIT. EXIT.
002480
002490 4-MOD-CLAVE.
002500     IF SW = ZERO MOVE KEYCAP TO AUX-CLAVE.
002510     PERFORM 4-DISPLAY THRU 420.
002520     MOVE 1 TO SW.
002530
002540 4-MOD-CONTEN.
002550     PERFORM 310.
002560     PERFORM 340.
002570
002580 450.
002590     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
002600     IF RES = "N" GO TO 4-MOD-CONTEN
002610       ELSE IF RES NOT = "S" GO TO 450.
002620
002630 4-REGRABAR.
002640     WRITE REG-01 INVALID KEY
002650          DISPLAY "* CLAVE YA EXISTE Y REGISTRO NO MODIFICACO *"
002660                  LINE 23 POSITION 19 BEEP
002670          ACCEPT RES OFF LINE 23 POSITION 63 GO TO 460-EXIT.
002680     MOVE AUX-CLAVE TO KEYCAP.
002690     DELETE SINTF01 RECORD.
002700     DISPLAY "* REGISTRO MODIFICADO *" LINE 23 POSITION 29 BEEP.
002710     ACCEPT RES OFF LINE 23 POSITION 52.
002720
002730 460-EXIT. EXIT.
002740
002750 5-ELIMINACION.
002760     PERFORM 2-DISPLAY.
002770     DISPLAY "* ELIMINACION *" LINE 1 POSITION 1 BEEP.
002780     PERFORM 3-CLAVE.
002790
002800 5-DISPLAY.
002810     PERFORM 310.
002820
002830 5-ACEPTAR.
002840     PERFORM 3-ACEPTAR.
002850
002860 510.
002870     IF CAP = ZEROS GO TO 5-EXIT
002880     ELSE IF KEYCAP = SPACES OR KEYCAP IS NOT NUMERIC
002890             GO TO 5-ACEPTAR.
002900
002910 520.
```

```
002920     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
002930     IF RES = "N" GO TO 5-ACEPTAR
002940       ELSE IF RES NOT = "S" GO TO 520.
002950     READ SINTF01 INVALID KEY
002960                  DISPLAY SPACES LINE 13 POSITION 5 SIZE 22
002970                          "* REGISTRO NO EXISTE *"
002980                          LINE 23 POSITION 30 BEEP
002990                  ACCEPT RES OFF LINE 23 POSITION 52
003000                  GO TO 5-EXIT.
003010     PERFORM 3-RESTO.
003020     DISPLAY CONTEN LINE 10 POSITION 1 LOW
003030             SPACES LINE 13 POSITION 1 SIZE 30
003040             "ESTA SEGURO? (S,N)" LINE 13 POSITION 31 BEEP.
003050
003060 530.
003070     ACCEPT RES LINE 13 POSITION 50 PROMPT ECHO LOW.
003080     IF RES = "N" GO TO 5-EXIT
003090       ELSE IF RES NOT = "S" GO TO 530.
003100     DISPLAY SPACES LINE 13 POSITION 31 SIZE 20.
003110     DELETE SINTF01 INVALID KEY
003120                    DISPLAY "* ERROR AL ELIMINAR REGISTRO *"
003130                            LINE 23 POSITION 26 BEEP
003140                            RES OFF LINE 23 POSITION 56
003150
003160     DISPLAY "* REGISTRO ELIMINADO *" LINE 23 POSITION 30 BEEP.
003170     ACCEPT RES OFF LINE 23 POSITION 52.
003180
003190 5-EXIT. EXIT.
003200
003210 6-CONSULTA.
003220     PERFORM 2-DISPLAY.
003230     DISPLAY "* CONSULTA *" LINE 1 POSITION 1 BEEP.
003240
003250 6-ACEPTAR.
003260     DISPLAY "LISTA TODO EL ARCHIVO? (S,N)"
003270             LINE 10 POSITION  26 BEEP.
003280     ACCEPT RES LINE 10 POSITION 55 PROMPT ECHO LOW.
003290     DISPLAY SPACES LINE 10 POSITION 26 SIZE 35.
003300     IF RES = "N" PERFORM 6-CLAVE THRU 620  GO TO 600
003310       ELSE IF RES NOT = "S" GO TO 6-ACEPTAR.
003320     MOVE ZEROS TO KEYCAP.
003330     START SINTF01 KEY > KEYCAP
003340                   INVALID KEY GO TO 6-EXIT.
003350
003360 600.
003370     PERFORM 3-CLAVE.
003380     PERFORM 3-RESTO.
```

```
003390     DISPLAY "PRESIONE <ENTER> PARA CONTINUAR O <F> PARA TERMINAR"
003400             LINE 24 POSITION 16.
003410     PERFORM 6-DISPLAY THRU 630-EXIT
003420             VARYING LINEA FROM 10 BY 1
003430             UNTIL LINEA > 23 OR EOF = "S" OR RES = "F".
003440     IF EOF NOT = "S"
003450        IF RES NOT = "F" PERFORM 6-CONSULTA
003460                         GO TO 600
003470        ELSE DISPLAY SPACES LINE 24 POSITION  16 SIZE 51
003480                     "* CONSULTA TERMINADA *"
003490                     LINE 24 POSITION 30
003500             ACCEPT RES OFF LINE 24 POSITION 52
003510     ELSE DISPLAY SPACES LINE 24 POSITION 16 SIZE 51
003520                  "* FIN DEL LISTADO DEL ARCHIVO *"
003530                  LINE 24 POSITION 25
003540         ACCEPT RES OFF LINE 24 POSITION 56.
003550     MOVE SPACES TO REG-01.
003560     MOVE ZEROS TO CAP.
003570     MOVE SPACE TO EOF.
003580
003590 6-EXIT. EXIT.
003600
003610 6-CLAVE.
003620     PERFORM 3-CLAVE.
003630
003640 610.
003650     PERFORM 310.
003660     PERFORM 3-ACEPTAR.
003670     IF CAP = ZEROS GO TO 6-EXIT
003680     ELSE IF KEYCAP = SPACES OR KEYCAP IS NOT NUMERIC
003690             GO TO 610.
003700
003710 620.
003720     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO.
003730     IF RES = "N" GO TO 610
003740        ELSE IF RES NOT = "S" GO TO 620.
003750     DISPLAY SPACES LINE 13 POSITION 1 SIZE 30.
003760     START SINTF01 KEY = KEYCAP
003770                   INVALID KEY DISPLAY "* REGISTRO INEXISTENTE *"
003780                                       LINE 23 POSITION 29 BEEP
003790                               ACCEPT RES OFF LINE 23 POSITION 53
003800                               DISPLAY SPACES LINE 23 POSITION 29
003810                                       SIZE 24
003820                               GO TO 610.
003830
003840 6-DISPLAY.
003850     READ SINTF01 NEXT AT END MOVE "S" TO EOF
```

```
003860                              GO TO 630-EXIT.
003870     DISPLAY KEYCAP LINE 6 POSITION 72 LOW BEEP
003880             CONTEN LINE LINEA POSITION 1 LOW.
003890
003900 6-RES.
003910     ACCEPT RES OFF LINE 24 POSITION 67.
003920     IF RES NOT = SPACE AND RES NOT = "F" GO TO 6-RES.
003930
003940 630-EXIT. EXIT.
003950 END PROGRAM.
```

```
000100 IDENTIFICATION DIVISION.
000110 PROGRAM-ID. SINT020.
000120 AUTHOR.   FERNANDEZ Z. DEYDAMIA. D.
000130 INSTALLATION. RADIO SHACK.
000140
000150*    EL OBJETIVO DE ESTE PROGRAMA ES LA GRABACION Y MANTENIMIENTO
000160*    DEL ARCHIVO QUE CONTIENE LA  EVALUACION  DE  LA SINTAXIS DEL
000170*    PASCAL.
000180
000190 ENVIRONMENT DIVISION.
000200 CONFIGURATION SECTION.
000210 SOURCE-COMPUTER. MODELII.
000220 OBJECT-COMPUTER. MODELII-64K.
000230 INPUT-OUTPUT SECTION.
000240 FILE-CONTROL.
000250     SELECT SINTF02 ASSIGN TO RANDOM "SINTF02"
000260                    ORGANIZATION IS INDEXED
000270                    ACCESS MODE IS DYNAMIC
000280                    RECORD KEY IS KEYEVA.
000290
000300 DATA DIVISION.
000310 FILE SECTION.
000320 FD  SINTF02
000330         RECORD CONTAINS 87 CHARACTERS
000340         BLOCK CONTAINS 2 RECORDS
000350         LABEL RECORDS ARE STANDARD.
000360 01  REG-02.
000370     03 KEYEVA.
000380        05 CAP           PIC 99.
000390        05 PANT          PIC 99.
000400        05 LIN           PIC 99.
000410     03 TIPO             PIC X.
000420     03 CONTEN           PIC X(80).
000430
000440 WORKING-STORAGE SECTION.
000450 77 LINEA                 PIC 99.
000460 77 RES                   PIC X.
000470 77 OPC                   PIC 9 VALUE ZERO.
000480 77 AUX-CLAVE             PIC X(6) VALUE SPACES.
000490 77 SW                    PIC 9 VALUE ZERO.
000500 77 EOF                   PIC X VALUE "N".
000510 77 CLAVE                 PIC X(10).
000520
000530     COPY "CLAVE02/CBL.DF".
000540
000550 PROCEDURE DIVISION.
000560 1-INICIO.
```

```
000570     DISPLAY "*** S E G U R I D A D ***"
000580             LINE 10 POSITION 29 BEEP ERASE
000590             "========================="
000600             LINE 11 POSITION 29 SIZE 25
000610             "DIGITE CLAVE DE PROTECCION"
000620             LINE 13 POSITION 29.
000630     ACCEPT CLAVE LINE 14 POSITION 36 OFF.
000640     IF CLAVE NOT = CLAVEF02
000650        DISPLAY "*** CLAVE ERRADA ***"
000660                LINE 13 POSITION 31 ERASE REVERSE BEEP
000670        GO TO 1-FIN.
000680
000690     OPEN I-O SINTF02.
000700     PERFORM 2-DISPLAY THRU 2-EXIT UNTIL OPC = 5.
000710     CLOSE SINTF02.
000720     DISPLAY "* FIN DEL PROGRAMA *"  ERASE BEEP
000730             LINE 15 POSITION 31 HIGH.
000740
000750 1-FIN.
000760     STOP RUN.
000770
000780 2-DISPLAY.
000790     DISPLAY "SINTAXIS DEL LENGUAJE PASCAL" LINE 2
000800              POSITION 27  ERASE BEEP
000810             "============================" LINE 3
000820              POSITION 27
000830             "* FIN"
000840              LINE 4 POSITION 20.
000850
000860 2-MENU.
000870     DISPLAY "1. INGRESO"      LINE 7 POSITION 34
000880             "2. MODIFICACION" LINE 8 POSITION 34
000890             "3. ELIMINACION"  LINE 9 POSITION 34
000900             "4. CONSULTA"     LINE 10 POSITION 34
000910             "5. FINALIZAR"    LINE 11 POSITION 34
000920             "OPCION -->"      LINE 13 POSITION 35.
000930
000940 2-ACEPTAR.
000950     ACCEPT OPC LINE 13 POSITION 46 PROMPT ECHO LOW.
000960     IF OPC = 5 GO TO 2-EXIT
000970       ELSE IF OPC < 1 OR OPC > 4 GO TO 2-ACEPTAR.
000980
000990 210.
001000     MOVE SPACES TO REG-02.
001010     IF OPC = 1 PERFORM 3-INGRESO THRU 3-EXIT
001020              UNTIL CAP = ZEROS
001030       ELSE IF OPC = 2 PERFORM 4-MODIFICACION THRU 4-EXIT
```

```
001040                      UNTIL CAP = ZEROS
001050         ELSE IF OPC = 3 PERFORM 5-ELIMINACION THRU 5-EXIT
001060                        UNTIL CAP = ZEROS
001070           ELSE PERFORM 6-CONSULTA THRU 6-EXIT
001080                UNTIL CAP = ZEROS.
001090     MOVE 99 TO CAP.
001100
001110 2-EXIT. EXIT.
001120
001130 3-INGRESO.
001140     PERFORM 2-DISPLAY.
001150     DISPLAY "* INGRESO *" LINE 1 POSITION 1 BEEP.
001160
001170 3-CLAVE.
001180     DISPLAY "CLAVE -->" LINE 6 POSITION 49.
001190
001200 3-RESTO.
001210     DISPLAY "TIPO -->"  LINE 6 POSITION 71
001220             "CONTENIDO" LINE 8 POSITION 36
001230             "1...5...10...15...20...25...30...35...40"
001240              LINE 9 POSITION 1
001250             "...45...50...55...60...65...70...75...80"
001260              LINE 9 POSITION 41.
001270
001280 310.
001290     DISPLAY "ESTA CORRECTO? (S/N)" LINE 13 POSITION 5.
001300
001310 3-ACEPTAR.
001320     ACCEPT KEYEVA LINE 6 POSITION 59 PROMPT ECHO LOW.
001330
001340 320.
001350     IF CAP = ZEROS GO TO 3-EXIT
001360     ELSE IF KEYEVA = SPACES OR KEYEVA IS NOT NUMERIC
001370             GO TO 3-ACEPTAR.
001380
001390 330.
001400     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
001410     IF RES = "N" GO TO 3-ACEPTAR
001420       ELSE IF RES NOT = "S" GO TO 330.
001430
001440 340.
001450     ACCEPT TIPO LINE 6 POSITION 80 PROMPT ECHO LOW.
001460
001470 350.
001480     IF TIPO NOT = "*" AND TIPO NOT = SPACE GO TO 340.
001490
001500 360.
```

```
001510     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
001520     IF RES = "N" GO TO 340
001530       ELSE IF RES NOT = "S" GO TO 360.
001540
001550 370.
001560     ACCEPT CONTEN LINE 10 POSITION 1 PROMPT ECHO LOW.
001570
001580 380.
001590     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
001600     IF RES = "N" GO TO 370
001610       ELSE IF RES NOT = "S" GO TO 380.
001620
001630 3-GRABACION.
001640     WRITE REG-02 INVALID KEY
001650                  DISPLAY SPACES LINE 13 POSITION 5 SIZE 22
001660                          "* CLAVE YA EXISTE *"
001670                           LINE 23 POSITION 31 BEEP
001680                  ACCEPT REC-OFF LINE 23 POSITION 50.
001690
001700
001710 3-EXIT. EXIT.
001720
001730 4-MODIFICACION.
001740     PERFORM 2-DISPLAY.
001750     DISPLAY "* MODIFICACION *" LINE 1 POSITION 1 BEEP.
001760           M 3-CLAVE.
001770
001780 4-DISPLAY.
001790     PERFORM 310.
001800
001810 4-ACEPTAR.
001820     PERFORM 3-ACEPTAR.
001830
001840 410.
001850     IF CAP = ZEROS GO TO 4-EXIT
001860     ELSE IF KEYEVA = SPACES OR KEYEVA IS NOT NUMERIC
001870             GO TO 4-ACEPTAR.
001880
001890 420.
001900     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
001910     IF RES = "N" GO TO 4-ACEPTAR
001920       ELSE IF RES NOT = "S" GO TO 420.
001930
001940 4-LECTURA.
001950     READ SINTF02 INVALID KEY
001960                  DISPLAY SPACES LINE 13 POSITION 5 SIZE 22
001970                          "* REGISTRO NO EXISTE *"
```

```
001980                          LINE 23 POSITION 31 BEEP
001990                  ACCEPT RES OFF LINE 23 POSITION 53
002000                  GO TO 4-ENCERE.
002010
002020 4-RESTO.
002030     DISPLAY "1." LINE 6 POSITION 47
002040             "2." LINE 6 POSITION 69
002050             "3." LINE 8 POSITION 34.
002060     PERFORM 3-RESTO.
002070
002080 430.
002090     DISPLAY TIPO LINE 6 POSITION 80 LOW
002100             CONTEN LINE 10 POSITION 1 LOW.
002110
002120 4-CORRECCION.
002130     DISPLAY SPACES LINE 13 POSITION 1 SIZE 70
002140             "CAMPO A MODIFICAR? (1,2,3) 0 <ENTER>"
002150             LINE 13 POSITION 22 BEEP.
002160     ACCEPT RES LINE 13 POSITION 59 PROMPT ECHO LOW.
002170     DISPLAY SPACES LINE 13 POSITION 22 SIZE 38.
002180     IF  RES = SPACE GO TO 4-GRABACION
002190       ELSE IF RES = "1" PERFORM 4-MOD-CLAVE
002200         ELSE IF RES = "2" PERFORM 4-MOD-TIPO THRU 450
002210           ELSE IF RES NOT = "3" GO TO 4-CORRECCION
002220             ELSE PERFORM 4-MOD-CONTEN THRU 460.
002230     GO TO 4-CORRECCION.
002240
002250 4-GRABACION.
002260     DISPLAY "REGRABA O CORRIGE? (R,C) O <ENTER>"
002270             LINE 13 POSITION 23 BEEP.
002280
002290 440.
002300     ACCEPT RES LINE 13 POSITION 58 PROMPT ECHO LOW.
002310     IF RES = SPACE GO TO 4-ENCERE
002320       ELSE IF RES = "C" GO TO 4-CORRECCION
002330         ELSE IF RES NOT = "R" GO TO 440.
002340     DISPLAY SPACES LINE 13 POSITION 23 SIZE 36.
002350     IF AUX-CLAVE NOT = KEYEVA
002360        IF AUX-CLAVE = SPACES NEXT SENTENCE
002370        ELSE PERFORM 4-REGRABAR THRU 470-EXIT GO TO 4-ENCERE.
002380     REWRITE REG-02 INVALID KEY
002390                    DISPLAY "* ERROR AL MODIFICAR REGISTRO *"
002400                            LINE 23 POSITION 25 BEEP
002410                    ACCEPT RES OFF LINE 23 POSITION 56
002420                    GO TO 4-ENCERE.
002430     DISPLAY "* REGISTRO MODIFICADO *" LINE 23 POSITION 29 BEEP.
002440     ACCEPT RES OFF LINE 23 POSITION 52.
```

```
002450
002460 4-ENCERE.
002470     MOVE SPACES TO AUX-CLAVE.
002480     MOVE ZERO TO SW.
002490
002500 4-EXIT. EXIT.
002510
002520 4-MOD-CLAVE.
002530     IF SW = ZERO MOVE KEYEVA TO AUX-CLAVE.
002540     PERFORM 4-DISPLAY THRU 420.
002550     MOVE 1 TO SW.
002560
002570 4-MOD-TIPO.
002580     PERFORM 340.
002590     PERFORM 310.
002600     IF TIPO NOT = "*" AND TIPO NOT = SPACE GO TO 4-MOD-TIPO.
002610
002620
002630     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
002640     IF RES = "N" GO TO 4-MOD-TIPO
002650       ELSE IF RES NOT = "S" GO TO 450.
002660
002670 4-MOD-CONTEN.
002680     PERFORM 310.
002690     PERFORM 370.
002700
002710 460.
002720     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
002730     IF RES = "N" GO TO 4-MOD-CONTEN
002740       ELSE IF RES NOT = "S" GO TO 460.
002750
002760 4-REGRABAR.
002770     WRITE REG-02 INVALID KEY
002780          DISPLAY "* CLAVE YA EXISTE Y REGISTRO NO MODIFICADO *"
002790                  LINE 23 POSITION 19 BEEP
002800          ACCEPT RES OFF LINE 23 POSITION 63 GO TO 470-EXIT.
002810     MOVE AUX-CLAVE TO KEYEVA.
002820     DELETE SINTF02 RECORD.
002830     DISPLAY "* REGISTRO MODIFICADO *" LINE 23 POSITION 29 BEEP.
002840     ACCEPT RES OFF LINE 23 POSITION 52.
002850
002860 470-EXIT. EXIT.
002870
002880 5-ELIMINACION.
002890     PERFORM 2-DISPLAY.
002900     DISPLAY "* ELIMINACION *" LINE 1 POSITION 1 BEEP.
002910     PERFORM 3-CLAVE.
```

```
002920
002930 5-DISPLAY.
002940     PERFORM 310.
002950
002960 5-ACEPTAR.
002970     PERFORM 3-ACEPTAR.
002980
002990 510.
003000     IF CAP = ZEROS GO TO 5-EXIT
003010     ELSE IF KEYEVA = SPACES OR KEYEVA IS NOT NUMERIC
003020             GO TO 5-ACEPTAR.
003030
003040 520.
003050     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO LOW.
003060     IF RES = "N" GO TO 5-ACEPTAR
003070       ELSE IF RES NOT = "S" GO TO 520.
003080     READ SINTF02 INVALID KEY
003090                  DISPLAY SPACES LINE 13 POSITION 5 SIZE 22
003100                          "* REGISTRO NO EXISTE *"
003110                          LINE 23 POSITION 30 BEEP
003120                  ACCEPT RES OFF LINE 23 POSITION 52
003130                  GO TO 5-EXIT.
003140     PERFORM 3-RESTO.
003150     DISPLAY TIPO LINE 6 POSITION 80
003160             CONTEN LINE 10 POSITION 1 LOW.
003170     DISPLAY SPACES LINE 13 POSITION 1 SIZE 30
003180             "ESTA SEGURO? (S,N)" LINE 13 POSITION 31 BEEP.
003190
003200 530.
003210     ACCEPT RES LINE 13 POSITION 50 PROMPT ECHO LOW.
003220     IF RES = "N" GO TO 5-EXIT
003230       ELSE IF RES NOT = "S" GO TO 530.
003240     DISPLAY SPACES LINE 13 POSITION 31 SIZE 30.
003250     DELETE SINTF02 INVALID KEY
003260                    DISPLAY "* ERROR AL ELIMINAR REGISTRO *"
003270                            LINE 23 POSITION 26 BEEP
003280                    ACCEPT RES OFF LINE 23 POSITION 56
003290                    GO TO 5-EXIT.
003300     DISPLAY "* REGISTRO ELIMINADO *" LINE 23 POSITION 30 BEEP.
003310     ACCEPT RES OFF LINE 23 POSITION 52.
003320
003330 5-EXIT. EXIT.
003340
003350 6-CONSULTA.
003360     PERFORM 2-DISPLAY.
003370     DISPLAY "* CONSULTA *" LINE 1 POSITION 1 BEEP.
003380
```

```
003390 6-ACEPTAR.
003400     DISPLAY "LISTA TODO EL ARCHIVO? (S,N)"
003410             LINE 10 POSITION  26 BEEP.
003420     ACCEPT RES LINE 10 POSITION 55 PROMPT ECHO LOW.
003430     DISPLAY SPACES LINE 10 POSITION 26 SIZE 35.
003440     IF RES = "N" PERFORM 6-CLAVE THRU 620  GO TO 600
003450       ELSE IF RES NOT = "S" GO TO 6-ACEPTAR.
003460     MOVE ZEROS TO KEYEVA.
003470     START SINTF02 KEY > KEYEVA
003480                   INVALID KEY GO TO 6-EXIT.
003490
00350[illegible]
003510     PERFORM 3-CLAVE.
003520     PERFORM 3-RESTO.
00353[illegible]
003540             LINE 24 POSITION 16.
003550     PERFORM 6-DISPLAY THRU 630-EXIT
003560             VARYING LINEA FROM 10 BY 1
003570             UNTIL LINEA > 23 OR EOF = "S" OR RES = "F".
003580     IF EOF N[illegible]
003590       IF RES NOT = "F" PERFORM 6-CONSULTA
003600                        GO TO 600
003610         ELSE DISPLAY SPACES LINE 24 POSITION 16 SIZE 51
003620                             "* CONSULTA TERMINADA *"
003630                             LINE 24 POSITION 30
003640           ACCEPT RES OFF LINE 24 POSITION 52
003650     ELSE DISPLAY SPACES LINE 24 POSITION 16 SIZE 51
003660                  "* FIN DEL LISTADO DEL ARCHIVO *"
003670                  LINE 24 POSITION 25
003680           ACCEPT RES OFF LINE 24 POSITION 56.
003690     MOVE SPACES TO REG-02.
003700     MOVE ZEROS TO CAP.
003710     MOVE SPACE TO EOF.
003720
003730 6-EXIT. EXIT.
003740
003750 6-CLAVE.
003760     PERFORM 3-CLAVE.
003770
003780 610.
003790     PERFORM 310.
003800     PERFORM 3-ACEPTAR.
003810     IF CAP = ZEROS GO TO 6-EXIT
003820     ELSE IF KEYEVA = SPACES OR KEYEVA IS NOT NUMERIC
003830             GO TO 610.
003840
003850 620.
```

```
003860     ACCEPT RES LINE 13 POSITION 26 PROMPT ECHO.
003870     IF RES = "N" GO TO 610
003880        ELSE IF RES NOT = "S" GO TO 620.
003890     DISPLAY SPACES LINE 13 POSITION 1 SIZE 30.
003900     START SINTF02 KEY = KEYEVA
003910                   INVALID KEY DISPLAY "* REGISTRO INEXISTENTE *"
003920                                       LINE 23 POSITION 29 BEEP
003930                               ACCEPT RES OFF LINE 23 POSITION 53
003940                               DISPLAY SPACES LINE 23 POSITION 29
003950                                       SIZE 24
003960                               GO TO 610.
003970
003980 6-DISPLAY.
003990     READ SINTF02 NEXT AT END MOVE "S" TO EOF
004000                              GO TO 630-EXIT.
004010     DISPLAY KEYEVA LINE 6 POSITION 59 LOW BEEP
004020             TIPO LINE 6 POSITION 80 LOW
004030             CONTEN LINE LINEA POSITION 1 LOW.
004040
004050 6-RES.
004060     ACCEPT RES OFF LINE 24 POSITION 67.
004070     IF RES NOT = SPACE AND RES NOT = "F" GO TO 6-RES.
004080
004090 630-EXIT. EXIT.
004100 END PROGRAM.
```

```
000100 IDENTIFICATION DIVISION.
000110 PROGRAM-ID. SINT030.
000120 AUTHOR. DEYDAMIA FERNANDEZ Z.
000130
000140*    EL OBJETIVO DE ESTE PROGRAMA ES LISTAR EL ARCHIVO QUE
000150*    CONTIENE LA SINTAXIS DEL LENGUAJE "PASCAL".
000160
000170 ENVIRONMENT DIVISION.
000180 CONFIGURATION SECTION.
000190 SOURCE-COMPUTER. MODELL-II.
000200 OBJECT COMPUTER. MODELL-II-64K.
000210 INPUT-OUTPUT SECTION.
000220 FILE-CONTROL.
000230     SELECT SINTF01 ASSIGN TO RANDOM "SINTF01"
                          ORGANIZATION INDEXED
000250                    ACCESS MODE DYNAMIC
000260                    RECORD KEY IS KEYCAP.
000270
000280     SELECT IMPRE ASSIGN TO PRINT "PRINTER"
000290                  STATUS IS ESTADO.
000300
000310 DATA DIVISION.
000320 FILE SECTION.
000330 FD SINTF01 RECORD CONTAINS 88 CHARACTERS
000340           BLOCK CONTAINS 2 RECORDS
000350           LABEL RECORDS ARE STANDARD.
000360 01 REG-01.
000370    03 KEYCAP.
000380       05 CAP                 PIC 99.
000390       05 UNI                 PIC 99.
000400       05 PANT                PIC 99.
000410       05 LIN                 PIC 99.
000420    03 CONTEN                 PIC X(80).
000430    03 T REDEFINES CONTEN.
000440       05 T-C                 PIC X OCCURS 80 TIMES.
000450
000460 FD IMPRE RECORD CONTAINS 80 CHARACTERS
000470         LABEL RECORD IS OMITTED.
000480 01 REG-IMP                   PIC X(80).
000490
000500 WORKING-STORAGE SECTION.
000510 77 RES                       PIC X.
000520 77 I                         PIC 99.
000530 77 ESTADO                    PIC XX.
000540 77 LINEA                     PIC 99.
000550 77 CON                       PIC 99 VALUE ZEROS.
000560 01 W-KEYCAP.
```

```
000570     03 W-CAP                  PIC 99.
000580     03 W-UNI                  PIC 99.
000590     03 W-PANT                 PIC 99.
000600     03 W-LIN                  PIC 99.
000610
000620 PROCEDURE DIVISION.
000630 DECLARATIVES.
000640 ERROR-IMPRE SECTION.
000650     USE AFTER STANDARD EXCEPTION PROCEDURE ON IMPRE.
000660 END DECLARATIVES.
000670
000680 PRINCIPAL SECTION.
000690 INICIO.
000700     OPEN OUTPUT IMPRE I-O SINTF01.
000710     PERFORM PROCESO THRU SALIR.
000720     DISPLAY ' FIN DEL PROGRAMA *'
000730             LINE 15 POSITION 31 ERASE BEEP.
000740
000750 CERRAR.
000760     CLOSE SINTF01 IMPRE.
000770
000780 FIN.
000790     STOP RUN.
000800
000810 PROCESO.
000820     MOVE ZEROS TO KEYCAP  W-KEYCAP.
000830     START SINTF01 KEY > KEYCAP INVALID KEY
000840                   DISPLAY '*** ERROR AL LISTAR ARCHIVO ***'
000850                           LINE 13 POSITION 20 ERASE REVERSE BEEP
000860                           AT    REG OFF LINE 13 POSITION 57
000870                           GO TO SALIR.
000880
000890 LEO-ESC.
000900     MOVE LIN TO LINEA.
000910     READ SINTF01 NEXT AT END GO TO SALIR.
000920     IF LIN = 1 MOVE LIN TO LINEA
000930     ELSE COMPUTE LINEA = LIN - LINEA.
000940     IF CAP NOT = W-CAP
000950        PERFORM SALTAR
000960     ELSE IF UNI NOT = W-UNI
000970          ADD 2 TO LINEA
000980          ADD LINEA TO CON
000990          IF CON > 57
001000             PERFORM SALTAR
001010          ELSE WRITE REG-IMP FROM CONTEN AFTER LINEA LINE
001020     ELSE MOVE "S" TO RED
001030          PERFORM BUSCAR VARYING I FROM 1 BY 1
```

```
001040                          UNTIL I > 4 OR RES = "N"
001050          ADD LINEA TO CON
001060          PERFORM CONTINUA
001070          IF CON > 57
001080             PERFORM SALTAR
001090          ELSE WRITE REG-IMP FROM CONTEN AFTER LINEA LINE.
001100     MOVE KEYCAP TO W-K
001110     GO TO LEC-ESC.
001120
001130 SALIR.  EXIT.
001140
001150 CONTINUA.
001160     IF LIN = 1 AND RES = "S"
001170         ADD 1 TO LINEA
001180         ADD 1 TO CON.
001190
001200 SALTAR.
001210     WRITE REG-IMP FROM SPACES AFTER PAGE.
001220     IF ESTADO NOT = ZERO
001230         DISPLAY "*** IMPRESORA NO LISTA ***"
001240                 LINE 15 POSITION 28 ERASE REVERSE BEEP
001250         GO TO CERRAR.
001260     IF CON = ZEROS
001270        DISPLAY "*** LISTANDO SINTAXIS DEL PASCAL ***"
001280                LINE 13 POSITION 23 ERASE BEEP.
001290     WRITE REG-IMP FROM CONTEN AFTER 10 LINES.
001300     MOVE 11 TO CON.
001310
001320 BUSCAR.
001330     IF T-C(I) NOT = SPACE
001340        MOVE "N" TO RES.
001350
001380 END PROGRAM.
```

```
000100 IDENTIFICATION DIVISION.
000110 PROGRAM-ID. SINT040.
000120 AUTHOR. DEYDAMIA FERNANDEZ.
000130
000140*        ESTE PROGRAM[illegible]                    UARIO.
000150
000170 ENVIRONMENT DIVISION.
000180 CONFIGURATION SECTION.
000190 SOURCE COMPUTER. MODELL-II.
000200 OBJECT-COMPUTER. MODELL-II-64K.
000210 INPUT-OUTPUT SECTION.
000220 FILE-CONTROL.
000230     SELECT SINTF01 ASSIGN TO RANDOM "SINTF01"
000240                    [illegible]TION IS INDEXED
000250                    ACCESS MODE IS DYNAMIC
000260                    RECORD KEY IS KEYCAP.
000270
000280     SELECT IMPRE   ASSIGN TO PRINT "PRINTER"
000290                    STATUS IS ESTADO.
000300
000310 DATA DIVISION.
000320 FILE SECTION.
000330 FD SINTF01 RECORD CONTAINS 88 CHARACTERS
000340            LABEL RECORDS ARE STANDARD
000350            BLOCK CONTAINS 2 RECORDS.
000360 01 REG-01.
000370     03 KEYCAP.
000380         05 CAP                  PIC 99.
000390         05 UNI                  PIC 99.
000400         05 PANT                 PIC 99.
000410         05 LIN                  PIC 99.
000420     03 CONTEN                   PIC X(80).
000430     03 T1 REDEFINES CONTEN.
000440         05 TC                   PIC X OCCURS 80 TIMES.
000450
000460 FD IMPRE RECORD CONTAINS 80 CHARACTERS
000470          LABEL RECORD IS OMITTED.
000480 01 REG-IMP                      PIC X(80).
000490
000500 WORKING-STORAGE SECTION.
000510 77 OPC                          PIC X VALUE SPACE JUST.
000520 77 OPC1                         PIC X VALUE SPACE.
000530 77 OPC2                         PIC 9 VALUE ZERO.
000540 77 EOF                          PIC X.
000550 77 I-KEYCAP                     PIC 9(8).
000560 77 F-KEYCAP                     PIC 9(8).
```

```
000570 77 RES                          PIC X.
000580 77 I                           PIC 99.
000590 77 GUION                        PIC X(80) VALUE ALL "-".
000600 77 OK                          PIC X.
000610 77 CONT                        PIC 99.
000620 77 LINEA                       PIC 99.
000630 77 E                           PIC Z9.
000640 77 ESTADO                      PIC XX.
000650 77 I-CAP                       PIC 99 VALUE ZEROS.
000660 77 SW                          PIC 9 VALUE ZERO.
000670 01 TABLA1                       VALUE "050803070508080504".
000680     03 T-C                      PIC 99 OCCURS 9 TIMES.
000690 01 TABLA2                  VALUE "030602020400000000040305030203020 20
000700-"002040700000000000005040301030303000004030302040000000003020202 0
000710-"030402000306050204020203000306070404000000000407040400000000000".
000720     03 T-U                OCCURS 9 TIMES.
000730         05 T-P            OCCURS 9 TIMES PIC 99.
000740 01 TABLA3.
000750     03 FILLER                   PIC X(39) VALUE "IDENTIFICADORES,
000760-" CONSTANTES Y VARIABLES".
000770     03 FILLER                   PIC X(39) VALUE "DATOS DE TIPO ES
000780-"CALAR                ".
000790     03 FILLER                   PIC X(39) VALUE "ENTRADA Y SALIDA
000800-"                     ".
000810     03 FILLER                   PIC X(39) VALUE "ESTRUCTURAS DE C
000820-"ONTROL               ".
000830     03 FILLER                   PIC X(39) VALUE "EXPRESIONES Y AS
000840-"IGNACIONES           ".
000850     03 FILLER                   PIC X(39) VALUE "PROCEDIMIENTOS Y
000860-" FUNCIONES           ".
000870     03 FILLER                   PIC X(39) VALUE "DATOS DE TIPOS E
000880-"STRUCTURADOS         ".
000890     03 FILLER                   PIC X(39) VALUE "ESTRUCTURAS DINA
000900-"MICAS DE DATOS       ".
000910     03 FILLER                   PIC X(39) VALUE "ARCHIVOS
000920-"                     ".
000930 01 [illegible]
000940     03 FILLER                   PIC X(39) OCCURS 9 TIMES.
000950 01 TIT2.
000960     03 FILLER                   PIC X(35) VALUE SPACES.
000970     03 FILLER                   PIC X(8) VALUE "CAPITULO".
000980     03 C                        PIC Z9.
000990     03 FILLER                   PIC X(35) [illegible]
001000 01 TIT3.
001010     03 FILLER                   PIC X(36) VALUE SPACES.
001020     03 FILLER                   PIC X(6) VALUE "UNIDAD".
001030     03 U                        PIC Z9.
```

```
001040      03 FILLER                    PIC X(36) VALUE SPACES.
001050 01 W-KEYCAP.
001060      03 W-CAP                      PIC 99.
001070      03 W-UNI                      PIC 99.
001080      03 W-PANT                     PIC 99.
001090      03 W-LIN                      PIC 99.
001100 01 OPCIONES.
001110      03 FILLER                     PIC X(79) VALUE "-->      1.RETROC
001120-"EDER    2.AVANZAR    3.EVALUAR    4.LISTAR    5.TERMINAR".
001130
001140 01 T1.
001150      03 FILLER                     PIC X(21) VALUE SPACES.
001160      03 C1                         PIC X(39).
001170      03 FILLER                     PIC X(20) VALUE SPACES.
001180 01 T2.
001190      03 FILLER                     PIC X(30) VALUE SPACES.
001200      03 C2                         PIC X(39).
001210      03 FILLER                     PIC X(11) VALUE SPACES.
001220 01 T3.
001230      03 FILLER                     PIC X(32) VALUE SPACES.
001240      03 C3                         PIC X(39).
001250      03 FILLER                     PIC X(9) VALUE SPACES.
001260 01 T4.
001270      03 FILLER                     PIC X(29) VALUE SPACES.
001280      03 C4                         PIC X(39).
001290      03 FILLER                     PIC X(12) VALUE SPACES.
001300 01 T5.
001310      03 FILLER                     PIC X(27) VALUE SPACES.
001320      03 C5                         PIC X(39).
001330      03 FILLER                     PIC X(14) VALUE SPACES.
001340 01 T6.
001350      03 FILLER                     PIC X(27) VALUE SPACES.
001360      03 C6                         PIC X(39).
001370      03 FILLER                     PIC X(14) VALUE SPACES.
001380 01 T7.
001390      03 FILLER                     PIC X(26) VALUE SPACES.
001400      03 C7                         PIC X(39).
001410      03 FILLER                     PIC X(15) VALUE SPACES.
001420 01 T8.
001430      03 FILLER                     PIC X(25) VALUE SPACES.
001440      03 C8                         PIC X(39).
001450      03 FILLER                     PIC X(16) VALUE SPACES.
001460 01 T9.
001470      03 FILLER                     PIC X(36) VALUE SPACES.
001480      03 C9                         PIC X(39).
001490      03 FILLER                     PIC X(5) VALUE SPACES.
001500
```

```
001510 PROCEDURE DIVISION.
001520 DECLARATIVES.
001530 ERROR-IMPRE SECTION.
001540     USE AFTER EXCEPTION PROCEDURE ON IMPRE.
001550 END DECLARATIVES.
001560
001570 PRINCIPAL SECTION.
001580 INICIO.
001590     OPEN INPUT SINTF01.
001600     PERFORM PROCESO THRU FIN UNTIL OPC = ZERO.
001610     DISPLAY "* FIN DEL PROGRAMA *"
001620             LINE 15 POSITION 31 ERASE BEEP.
001630     CLOSE SINTF01.
001640     STOP RUN.
001650
001660 PROCESO.
001670     IF OPC2 = ZERO PERFORM MENU THRU F-OPCIONES
001680     ELSE IF OPC2 = 1 PERFORM RETROCEDER THRU R-EXIT
001690     ELSE IF OPC2 = 2 PERFORM SEGUIR
001700     ELSE IF OPC2 = 3 PERFORM EVALUAR
001710     ELSE IF OPC2 = 4 PERFORM LISTAR THRU L-EXIT
          ELSE IF OPC1 = ZERO PERFORM MENU THRU F-OPCIONES
001730     ELSE IF OPC
001740             MOVE W-CAP TO OPC
001750             PERFORM CAPITULOS THRU F-OPCIONES
001760     ELSE PERFORM CAPITULOS THRU F-OPCIONES.
001770
001780 FIN. EXIT.
001790
001800 MENU.
001810     DISPLAY "MENU SOBRE LA SINTAXIS"
001820             LINE 5 POSITION
001830             "DEL LENGUAJE PASCAL"
001840             LINE 6 POSITION 31
001850             "0) TERMINAR SESION"
001860             LINE 8 POSITION 20
001870             "INGRESE SU SELECCION -->"
001880             LINE 20 POSITION 28.
001890     MOVE 9 TO LINEA.
001900     PERFORM M1 VARYING I FROM 1 BY 1 UNTIL I > 9.
001910
001920 ACEP-CAP.
001930     ACCEPT OPC LINE 20 POSITION 53 PROMPT ECHO.
001940
001950 CAPITULOS.
001960     MOVE OPC TO CAP.
001970     MOVE CAP TO C.
```

```
001980     IF OPC = ZERO GO TO F-OPCIONES
001990     ELSE IF OPC = 1 PERFORM M-CAP1
002000     ELSE IF OPC = 2 PERFORM M-CAP2
002010     ELSE IF OPC = 3 PERFORM M-CAP3
002020     ELSE IF OPC = 4 PERFORM M-CAP4
002030     ELSE IF OPC = 5 PERFORM M-CAP5
002040     ELSE IF OPC = 6 PERFORM M-CAP6
002050     ELSE IF OPC = 7 PERFORM M-CAP7
002060     ELSE IF OPC = 8 PERFORM M-CAP8
002070     ELSE IF OPC = 9 PERFORM M-CAP9
002080     ELSE GO TO ACEP-CAP.
002090     DISPLAY "INGRESE SU SELECCION -->"
002100             LINE 20 POSITION 28
002110             "<ENTER> RETORNAR A LECTURA"
002120             LINE 21 POSITION 28.
002130     MOVE SPACES TO KEYCAP.
002140
002150 ACEP-UNI.
002160     ACCEPT OPC1 LINE 20 POSITION 53 PROMPT ECHO.
002170     IF OPC1 = SPACE
002180         IF KEYCAP = SPACES GO TO ACEP-UNI
002190         ELSE MOVE ZEROS TO W-LIN
002200              MOVE W-KEYCAP TO KEYCAP
002210              GO TO CONTINUAR
002220     ELSE IF OPC1 = ZERO
002230             GO TO F-OPCIONES
002240     ELSE MOVE OPC TO I
002250          MOVE OPC1 TO CONT
002260          IF CONT > T-C(I) GO TO ACEP-UNI
002270          ELSE IF CONT < 1 GO TO ACEP-UNI.
002280     MOVE OPC TO CAP.
002290     MOVE OPC1 TO UNI.
002300     MOVE ZEROS TO PANT LIN.
002310
002320 CONTINUAR.
002330     MOVE "N" TO EOF.
002340     START SINTF01 KEY > KEYCAP
002350                   INVALID KEY GO TO F-OPCIONES.
002360
002370 SGTE-UNIDAD.
002380     READ SINTF01 NEXT.
002390     DISPLAY SPACES ERASE BEEP.
002400     MOVE KEYCAP TO W-KEYCAP.
002410     PERFORM DISPLAYAR UNTIL (W-PANT NOT = PANT)
002420                          OR (W-UNI NOT = UNI)
002430                          OR (W-CAP NOT = CAP)
002440                          OR (EOF = "S").
```

```
002450
002460 DISP-OPCIONES.
002470     DISPLAY GUION LINE 23 POSITION 1 SIZE 80
002480             OPCIONES LINE 24 POSITION 1 SIZE 79.
002490
002500 ACEP-OPC2.
002510     ACCEPT OPC2 LINE 24 POSITION 5 PROMPT ECHO.
002520     IF OPC2 < 1 OR OPC2 > 5 GO TO ACEP-OPC2.
002530
002540 F-OPCIONES. EXIT.
002550
002560 M1.
002570     MOVE I TO E.
002580     DISPLAY E LINE LINEA POSITION 19
002590             ")" LINE LINEA POSITION 21
002600             TIT2(I) LINE LINEA POSITION 23.
002610     ADD 1 TO LINEA.
002620
002630 M-CAP1.
002640     DISPLAY TIT1 LINE 5 POSITION 1 ERASE BEEP
002650             TIT2(CAP) LINE 6 POSITION 22
002660             "0) RETORNAR AL MENU PRINCIPAL"
002670             LINE 10 POSITION 26
002680             "1) IDENTIFICADORES"
002690             LINE 11 POSITION 26
002700             "2) ESTRUCTURA DE UN PROGRAMA EN"
002710             LINE 12 POSITION 26
002720             "PAS
002730             LINE 13 POSITION 29
002740             "3) PALABRAS RESERVADAS EN PASCAL"
002750             LINE 14 POSITION 26
002760             "4) IDENTIFICADORES ESTANDARD"
002770             LINE 15 POSITION 2
002780             "5) SIMBOLOS ESPECIALES"
002790             LINE 16 POSITION 26.
002800
002810 M-CAP2.
002820     DISPLAY TIT1 LINE 5 POSITION 1 ERASE BEEP
002830             TIT2(CAP) LINE 6 POSITION 31
002840             "0) RETORNAR AL MENU PRINCIPAL"
002850             LINE 9 POSITION 27
002860             "1) GENERALIDADES"
002870             LINE 10 POSITION 27
002880             "2) TIPOS ENUMERADOS"
002890             LINE 11 POSITION 27
002900             "3) TIPO BOOLEAN"
002910             LINE 12 POSITION 27
```

```
002920              "4) TIPO INTEGER"
002930              LINE 13 POSITION 27
002940              "5) TIPO CHAR"
002950              LINE 14 POSITION 27
002960              "6) TIPO REAL"
002970              LINE 15 POSITION 27
002980              "7) TIPOS SUBRANGO"
002990              LINE 16 POSITION 27
003000              "8) DECLARACIONES DE DATOS"
003010              LINE 17 POSITION 27.
003020
003030 M-CAP3.
003040     DISPLAY TIT1 LINE 5 POSITION 1 ERASE BEEP
003050             TIT2(CAP) LINE 6 POSITION 33
003060             "0) RETORNAR AL MENU PRINCIPAL"
003070             LINE 12 POSITION 27
003080             "1) ENTRADA ESTANDARD"
003090             LINE 13 POSITION 27
003100             "2) SALIDA ESTANDARD"
003110             LINE 14 POSITION 27
003120             "3) ENTRADA/SALIDA"
003130             LINE 15 POSITION 27.
003140
003150 M-CAP4.
003160     DISPLAY TIT1 LINE 5 POSITION 1 ERASE BEEP
003170             TIT2(CAP) LINE 6 POSITION 30
003180             "0) RETORNAR EL MENU PRINCIPAL"
003190             LINE 10 POSITION 27
003200             "1) SENTENCIA IF"
003210             LINE 11 POSITION 27
003220             "2) SENTENCIA CASE"
003230             LINE 12 POSITION 27
003240             "3) ACCIONES MULTIPLES"
003250             LINE 13 POSITION 27
003260             "4) ITERACIONES EN PASCAL"
003270             LINE 14 POSITION 27
003280             "5) SENTENCIA FOR"
003290             LINE 15 POSITION 27
003300             "6) SENTENCIA REPEAT"
003310             LINE 16 POSITION 27
003320             "7) SENTENCIA WHILE"
003330             LINE 17 POSITION 27.
003340
003350 M-CAP5.
003360     DISPLAY TIT1 LINE 5 POSITION 1 ERASE BEEP
003370             TIT2(CAP) LINE 6 POSITION 28
003380             "0) RETORNAR AL MENU PRINCIPAL"
```

```
003390              LINE 10 POSITION 24
003400              "1) JERARQUIA EN EXPRESIONES"
003410              LINE 11 POSITION 24
003420              "2) EVALUACION DE EXPRESIONES"
003430              LINE 12 POSITION 24
003440              "3) ASIGNACION DE VALORES A DATOS"
003450              LINE 13 POSITION 24
003460              "4) IMPLICACION DE REPRESENTACIONES"
003470              LINE 14 POSITION 24
003480              "PODEROSAS"
003490              LINE 15 POSITION 27
003500              "5) CONVERSION DE DATOS DE UN TIPO"
003510              LINE 16 POSITION 24
003520              "A OTRO"
003530              LINE 17 POSITION 27.
003540
003550 M-CAP6.
003560      DISPLAY TIT1 LINE 5 POSITION 1 ERASE BEEP
003570              TIT2(CAP) LINE 6 POSITION 28
003580              "0) RETORNAR AL MENU PRINCIPAL"
003590              LINE 8 POSITION 24
003600              "1) DECLARACION DE PROCEDIMIENTOS"
003610              LINE 9 POSITION 24
003620              "2) PARAMETROS DEL PROCEDIMIENTO"
003630              LINE 10 POSITION 24
003640              "3) PARAMETROS POR VALOR"
003650              LINE 11 POSITION 24
003660              "4) PARAMETROS VARIABLES"
003670              LINE 12 POSITION 24
003680              "5) RECURSION"
003690              LINE 13 POSITION 24
003700              "RESTRICCION DE REGLAS DEL PASCAL"
003710              LINE 14 POSITION 27
003720              "6) RECURSION Y DIRECTIVA FORWARD"
003730              LINE 15 POSITION 24
003740              "7) FUNCIONES"
003750              LINE 16 POSITION 24
003760              "8) FUNCIONES Y PROCEDIMIENTOS COMO"
003770              LINE 17 POSITION 24
003780              "PARAMETROS"
003790              LINE 18 POSITION 27.
003800
003810 M-CAP7.
003820      DISPLAY TIT1 LINE 5 POSITION 1 ERASE BEEP
003830              TIT2(CAP) LINE 6 POSITION 27
003840              "0) RETORNAR AL MENU PRINCIPAL"
003850              LINE 8 POSITION 25
```

```
003860              "1) TIPOS ESTRUCTURADOS"
003870              LINE 9 POSITION 25
003880              "2) TIPO SET"
003890              LINE 10 POSITION 25
003900              "3) TIPO ARRAY"
003910              LINE 11 POSITION 25
003920              "4) PROCEDIMIENTOS Y FUNCIONES"
003930              LINE 12 POSITION 25
003940              "USANDO ARREGLOS Y CONJUNTOS"
003950              LINE 13 POSITION 28
003960              "5) TIPO RECORD"
003970              LINE 14 POSITION 25
003980              "6) SENTENCIA WITH"
003990              LINE 15 POSITION 25
004000              "7) TIPO DE REGISTRO VARIANTE"
004010              LINE 16 POSITION 25
004020              "8) REPRESENTACION EMPAQUETADA DE"
004030              LINE 17 POSITION 25
004040              "ESTRUCTURAS DE DATOS (PACKED)"
004050              LINE 18 POSITION 28.
004060
004070 M-CAP8.
004080     DISPLAY TIT1 LINE 5 POSITION 1 ERASE BEEP
004090             TIT2(CAP) LINE 6 POSITION 26
004100             "0) RETORNAR AL MENU PRINCIPAL"
004110             LINE 10 POSITION 25
004120             "1) CONCEPTOS BASICOS"
004130             LINE 11 POSITION 25
004140             "2) APUNTADORES Y DATOS DE OBJETOS"
004150             LINE 12 POSITION 25
004160             "DINAMICOS"
004170             LINE 13 POSITION 28
004180             "3) LISTAS"
004190             LINE 14 POSITION 25
004200             "4) ARBOLES BINARIOS"
004210             LINE 15 POSITION 25
004220             "5) REGISTROS VARIANTES"
004230             LINE 16 POSITION 25.
004240
004250 M-CAP9.
004260     DISPLAY TIT1 LINE 5 POSITION 1 ERASE BEEP
004270             TIT2(CAP) LINE 6 POSITION 37
004280             "0) RETORNAR AL MENU PRINCIPAL"
004290             LINE 11 POSITION 26
004300             "1) GENERALIDADES"
004310             LINE 12 POSITION 26
004320             "2) ARCHIVOS"
```

```
004330             LINE 13 POSITION 26
004340             "3) ARCHIVOS DE TEXTO"
004350             LINE 14 POSITION 26
004360             "4) AREAS DE LECTURA Y ESCRITURA"
004370             LINE 15 POSITION 26.
004380
004390 DISPLAYAR.
004400     DISPLAY CONTEN LINE LIN POSITION 1.
004410     READ SINTF01 NEXT AT END MOVE "S" TO EOF.
004420
004430 LISTAR.
004440     OPEN OUTPUT IMPRE.
004450     MOVE ZEROS TO LINEA  CONT.
004460     MOVE KEYCAP TO F-KEYCAP.
004470     MOVE W-KEYCAP TO I-KEYCAP  KEYCAP.
004480     MOVE W-CAP TO C.
004490     IF SW = ZERO WRITE REG-IMP FROM TIT1 AFTER 7 LINES
004500                  PERFORM VERIFICAR
004510     ELSE IF I-CAP = W-CAP WRITE REG-IMP FROM SPACES AFTER PAGE
004520                           PERFORM VERIFICAR
004530                           ADD 7 TO CONT
004540                           ADD 6 TO LINEA
004550                           GO TO L1
004560     ELSE WRITE REG-IMP FROM SPACES AFTER PAGE
004570          PERFORM VERIFICAR
004580          WRITE REG-IMP FROM TIT1 AFTER 6 LINES.
004590     IF W-CAP = 1 MOVE TIT2(W-CAP) TO C1
004600                  WRITE REG-IMP FROM T1 AFTER 2 LINES
004610     ELSE IF W-CAP = 2 MOVE TIT2(W-CAP) TO C2
004620                       WRITE REG-IMP FROM T2 AFTER 2 LINES
     9     ELSE IF W-CAP = 3 MOVE TIT2(W-CAP) TO C3

004650     ELSE IF W-CAP = 4 MOVE TIT2(W-CAP) TO C4
004660                       WRITE REG-IMP FROM T4 AFTER 2 LINES
004670     ELSE IF W-CAP = 5 MOVE TIT2(W-CAP) TO C5
004680                       WRITE REG-IMP FROM T5 AFTER 2 LINES
004690     ELSE IF W-CAP = 6 MOVE TIT2(W-CAP) TO C6
004700                       WRITE REG-IMP FROM T6 AFTER 2 LINES
004710     ELSE IF W-CAP = 7 MOVE TIT2(W-CAP) TO C7
004720                       WRITE REG-IMP FROM T7 AFTER 2 LINES
004730     ELSE IF W-CAP = 8 MOVE TIT2(W-CAP) TO C8
004740                       WRITE REG-IMP FROM T8 AFTER 2 LINES
004750     ELSE IF W-CAP = 0 MOVE TIT2(W-CAP) TO C9
004760                       WRITE REG-IMP FROM T9 AFTER 2 LINES.
004770     ADD 12 TO CONT.
004780     ADD 3 TO LINEA.
004790
```

```
004800 L1.
004810     MOVE W-UNI TO U.
004820     WRITE REG-IMP FROM TIT3 AFTER LINEA LINE.
004830     IF W-UNI = 1 MOVE 3 TO LIN  W-LIN
004840        MOVE 2 TO LINEA
004850     ELSE MOVE 1 TO LIN W-LIN
004860         MOVE ZEROS TO LINEA.
004870     MOVE 1 TO PANT  W-PANT.
004880     START SINTF01 KEY = KEYCAP INVALID KEY GO TO L-EXIT.
004890
004900 LEC-ESC.
004910     READ SINTF01 NEXT AT END GO TO L-CONT.
004920     IF W-CAP NOT = CAP OR W-UNI NOT = UNI GO TO L-CONT.
004930     COMPUTE LINEA = LIN - LINEA.
004940     IF LIN = 1 AND PANT NOT = 1
004950        MOVE "S" TO RES
004960        PERFORM BUSCAR VARYING I FROM 1 BY 1
004970                UNTIL I > 4 OR RES = "N"
004980        IF RES = "S" MOVE 2 TO LINEA
004990        ELSE MOVE 1 TO LINEA.
005000     ADD LINEA TO CONT.
005010     IF CONT > 58 WRITE REG-IMP FROM SPACES AFTER PAGE
005020                  PERFORM VERIFICAR
005030                  WRITE REG-IMP FROM CONTEN AFTER 6 LINES
005040                  MOVE 7 TO CONT
005050     ELSE WRITE REG-IMP FROM CONTEN AFTER LINEA LINE.
005060     MOVE LIN TO LINEA.
005070     MOVE KEYCAP TO W-KEYCAP.
005080     GO TO LEC-ESC.
005090
005100 L-CONT.
005110     MOVE F-KEYCAP TO KEYCAP.
005120     MOVE I-KEYCAP TO W-KEYCAP.
005130     CLOSE IMPRE.
005140     IF SW = ZERO AND ESTADO = ZERO
005150        MOVE 1 TO SW.
005160     PERFORM ACEP-OPC2 THRU F-OPCIONES.
005170
005180 L-EXIT. EXIT.
005190
005200 VERIFICAR.
005210     IF ESTADO NOT = ZERO
005220         DISPLAY SPACES LINE 24 POSITION 1 SIZE 79
005230                 "<C> CONTINUAR LECTURA"
005240                 LINE 24 POSITION 6
005250                 "*** IMPRESORA NO LISTA ***"
005260                 LINE 24 POSITION 28 REVERSE
```

```
005270                 "<F> TERMINAR LECTURA"
005280                 LINE 24 POSITION 55
005290         PERFORM ACEP-LISTAR
005300         IF RES = "C" PERFORM DISP-OPCIONES
005310                 GO TO L-CONT
005320         ELSE IF RES = "F" MOVE ZERO TO OPC
005330                 GO TO L-EXIT.
005340
005350 ACEP-LISTAR.
005360     ACCEPT RES OFF LINE 24 POSITION 75.
005370     IF RES NOT = "C" AND RES NOT = "F" GO TO ACEP-LISTAR.
005380
005390 BUSCAR.
005400     IF TC(I) NOT = SPACE MOVE "N" TO RES.
005410
005420 SEGUIR.
005430     IF EOF NOT = "S" START SINTF01 KEY = KEYCAP
005440                      PERFORM SGTE-UNIDAD THRU F-OPCIONES
005450     ELSE DISPLAY SPACES LINE 24 POSITION 1 SIZE 79
005460                  "*** FIN DEL TEXTO ***"
005470                  LINE 24 POSITION 9 [illegible]
[illegible]                  "PRESIONE CUALQUIER TECLA PARA CONTINUAR"
005490                  [illegible]
005500         ACCEPT RES OFF LINE 24 POSITION 71
005510         PERFORM DISP-OPCIONES THRU F-OPCIONES.
005520
005530 RETROCEDER.
005540     [illegible]          -LIN.
005550     IF W-PANT > 1 SUBTRACT 1 FROM W-PANT
005560     ELSE IF W-UNI > 1 SUBTRACT 1 FROM W-UNI
005570                       MOVE T-P(W-CAP, W-UNI) TO W-PANT
005580     ELSE IF W-CAP > 1 SUBTRACT 1 FROM W-CAP
005590                       MOVE [illegible]
005600                       MOVE T-P(W-CAP, W-UNI) TO W-PANT
005610     ELSE PERFORM ACEP-OPC2 THRU F-OPCIONES
005620          GO TO R-EXIT.
005630     MOVE W-KEYCAP TO KEYCAP.
005640     PERFORM CONTINUAR THRU F-OPCIONES.
005650
005660 R-EXIT. EXIT.
005670
005680 EVALUAR.
005690     CALL "SINT070" USING W-CAP.
005700     MOVE W-KEYCAP TO KEYCAP.
005710     MOVE ZEROS TO LIN.
005720     PERFORM CONTINUAR THRU F-OPCIONES.
005730
```

```
005740 END PROGRAM.
```

```
000100 IDENTIFICATION DIVISION.
000110 PROGRAM-ID. SINT050.
000120 AUTHOR. DEYDAMIA FERNANDEZ.
000130
000140*    EL OBJETIVO DE ESTE PROGRAMA ES INICIALIZAR EL AREA DEL
000150*    ARCHIVO QUE CONTENDRA LA SINTAXIS DEL LENGUAJE PASCAL.
000160
000170 ENVIRONMENT DIVISION.
000180 CONFIGURATION SECTION.
000190 SOURCE-COMPUTER. MODELL-II.
000200 OBJECT-COMPUTER. MODELL-II-64K.
000210
000220 INPUT-OUTPUT SECTION.
000230 FILE-CONTROL.
000240     SELECT SINTF01 ASSIGN TO RANDOM "SINTF01"
000250                    ORGANIZATION INDEXED
000260                    ACCESS MODE RANDOM
000270                    RECORD KEY IS KEYCAP.
000280
000290 DATA DIVISION.
000300 FILE SECTION.
000310 FD  SINTF01  BLOCK  CONTAINS  80 CHARACTERS
000320             RECORD  CONTAINS  1 RECORDS
000330             LABEL  RECORDS ARE STANDARD.
000340 01 REG-01.
000350    03 KEYCAP.
000360       05 CAP        PIC 99.
000370       05 UNI        PIC 99.
000380       05 [illegible]
000390       05 LIN        PIC 99.
000400    03 CONTEN        PIC X(80).
000410
000420 WORKING-STORAGE SECTION.
000430 77 CLAVE            PIC X(10).
000440
000450     COPY "CLAVE01/CBL.DF".
000460
000470 PROCEDURE DIVISION.
000480 INICIO.
000490     DISPLAY "*** S E G U R I D A D ***"
000500             LINE 10 POSITION 29 ERASE BEEP
000510             "========================"
000520             LINE 11 POSITION 29
000530             "DIGITE CLAVE DE PROTECCION"
000540             LINE 13 POSITION 29.
000550     ACCEPT CLAVE LINE 14 POSITION 36 OFF.
000560     IF CLAVE NOT = CLAVEF01
```

```
000570         DISPLAY '*** CLAVE ERRADA ***'
000580              LINE 13 POSITION 31 ERASE REVERSE BEEP
000590              GO TO FIN.
000600
000610     OPEN OUTPUT SINTF01.
000620     PERFORM PROCESO.
000630     CLOSE SINTF01.
000640     DISPLAY '*** FIN DEL PROGRAMA ***'
000650             LINE 20 POSITION 29 ERASE BEEP.
000660
000670 FIN.
000680     STOP RUN.
000690
000700 PROCESO.
000710     MOVE SPACES TO REG-01.
000720     DISPLAY
000730     '** INICIALIZANDO AREA DEL ARCHIVO INDEXADO SINTF01 **'
000740     LINE 11 POSITION 15 ERASE BEEP HIGH.
000750     MOVE ZEROS TO KEYCAP.
000760     WRITE REG-01 INVALID KEY
000770                  DISPLAY '* ERROR EN INICIALIZACION DE AREA *'
000780                          LINE 15 POSITION 24 ERASE BEEP.
000790
000800 END PROGRAM.
```

```
000100 IDENTIFICATION DIVISION.
000110 PROGRAM-ID. SINT060.
000120 AUTHOR. DEYDAMIA FERNANDEZ.
000130
000140*    EL OBJETIVO DE ESTE PROGRAMA ES INICIALIZAR EL AREA DEL
000150*    ARCHIVO QUE CONTENDRA LA EVALUACION DE LA SINTAXIS  DEL
000160*    LENGUAJE PASCAL.
000170
000180 ENVIRONMENT DIVISION.
000190 CONFIGURATION SECTION.
000200 SOURCE-COMPUTER. MODELL-II.
000210 OBJECT-COMPUTER. MODELL-II-64K.
000220
000230 INPUT-OUTPUT SECTION.
000240 FILE-CONTROL.
000250     SELECT SINTF02 ASSIGN TO RANDOM "SINTF02"
000260                    ORGANIZATION INDEXED
000270                    ACCESS MODE RANDOM
000280                    RECORD KEY IS KEYEVA.
000290
000300 DATA DIVISION.
000310 FILE SECTION.
000320 FD SINTF02  RECORD CONTAINS 87 CHARACTERS
000330             LABEL RECORDS ARE STANDARD.
000340 01 REG-02.
000350    03 KEYEVA.
000360      05 CAP       PIC 99.
000370      05 PANT      PIC 99.
000380      05 LIN       PIC 99.
000390    03 TIPO        PIC X.
000400    03 CONTEN      PIC X(80).
000410
000420 WORKING-STORAGE SECTION.
000430 77 CLAVE            PIC X(10).
000440
000450     COPY "CLAVE02/CBL.DF".
000460
000470 PROCEDURE DIVISION.
000480 INICIO.
000490     DISPLAY "*** S E G U R I D A D ***"
000500             LINE 10 POSITION 29 ERASE BEEP
000510             "========================"
000520             LINE 11 POSITION 29
000530             "DIGITE CLAVE DE PROTECCION"
000540             LINE 13 POSITION 29.
000550     ACCEPT CLAVE OFF LINE 14 POSITION 36.
000560     IF CLAVE NOT = CLAVEF02
```

```
000570         DISPLAY '*** CLAVE ERRADA ***'
000580                 LINE 12 POSITION 31 ERASE BEEP REVERSE
000590                 GO TO FIN.
000600
000610     OPEN OUTPUT SINTF02.
000620     PERFORM PROCESO.
000630     CLOSE SINTF02.
000640     DISPLAY '*** FIN DEL PROGRAMA ***'
000650             LINE 20 POSITION 29 ERASE BEEP.
000660
000670 FIN.
000680     STOP RUN.
000690
000700 PROCESO.
000710     MOVE SPACES TO REG-02.
000720     DISPLAY
000730     '** INICIALIZANDO AREA DEL ARCHIVO INDEXADO SINTF02 **'
000740     LINE 11 POSITION 15 ERASE BEEP HIGH.
000750     MOVE ZEROS TO KEYEVA.
000760     WRITE REG-02 INVALID KEY
000770                  DISPLAY '* ERROR EN INICIALIZACION DE AREA *'
000780                          LINE 15 POSITION 24 ERASE BEEP.
000790
000800 END PROGRAM.
```

```
000100 IDENTIFICATION DIVISION.
000110 PROGRAM-ID. SINT070.
000120 AUTHOR. DEYDAMIA FERNANDEZ Z.
000130
000140*        EL OBJETIVO DE ESTE SUBPROGRAMA ES PRESENTAR LAS PREGUNTAS
000150*        CORRESPONDIENTES AL CAPITULO QUE EL USUARIO ESTA LEYENDO
000160*        ADEMAS DE EVALUAR LAS RESPUESTAS DADAS POR ESTE.
000170
000180 ENVIRONMENT DIVISION.
000190 CONFIGURATION SECTION.
000200 SOURCE-COMPUTER. MODEL-II.
000210 OBJECT-COMPUTER. MODEL-II-64K.
000220 INPUT-OUTPUT SECTION.
000230 FILE-CONTROL.
000240     SELECT SINTF02 ASSIGN TO RANDOM "SINTF02"
000250                    ORGANIZATION INDEXED
000260                    ACCESS MODE DYNAMIC
000270                    RECORD KEY KEYEVA.
000280
000290 DATA DIVISION.
000300 FILE SECTION.
000310 FD  SINTF02 LABEL RECORDS ARE STANDARD
000320             RECORD CONTAINS 87 CHARACTERS
000330             BLOCK CONTAINS 2 RECORDS.
000340 01  REG-02.
000350     03 KEYEVA.
000360        05 CAP           PIC 99.
000370        05 PANT          PIC 99.
000380        05 LIN           PIC 99.
000390     03 TIPO             PIC  X.
000400     03 CONTEN           PIC X(80).
000410
000420 WORKING-STORAGE SECTION.
000430 01 W-KEYEVA.
000440     03 W-CAP            PIC 99.
000450     03 W-PANT           PIC 99.
000460     03 W-LIN            PIC 99.
000470 01 TABLA1               VALUE "050102020405050102030405060102030
000480-"050401020203040501020304050601020304060801030406080501020304050
000490-"0102040506".
000500     03 T                OCCURS 9 TIMES.
000510         05 PG           PIC 99.
000520         05 R-PG         PIC 99 OCCURS 5 TIMES.
000530 01 TABLA2.
000540     03 T-CAP            OCCURS 5 TIMES.
000550        05 T-R           PIC X.
000560        05 T-P           PIC 99.
```

```
000570         05 T-L              PIC 99.
000580         05 T-C              PIC 9.
000590 01 TABLA3.
000600      03 T-RESP              PIC X OCCURS 5 TIMES.
000610 01 OPCIONES.
000620      03 FILLER              PIC X(79) VALUE "--> 1. RETROCEDER  2.
000630-" AVANZAR  3. CONSTESTAR  4. EVALUACION".
000640 77 RES                     PIC X.
000650 77 OK                      PIC X VALUE "N".
000660 77 I                       PIC 9 VALUE ZERO.
000670 77 E                       PIC Z9.
000680 77 J                       PIC 9 VALUE ZERO.
000690 77 K                       PIC 99 VALUE ZEROS.
000700 77 L                       PIC 99 VALUE ZEROS.
000710 77 CONT                    PIC 99 VALUE ZEROS.
000720 77 GUION                   PIC X(80) VALUE ALL "-".
000730 77 V                       PIC X VALUE "!".
000740 77 S1                      PIC X VALUE "\".
000750 77 AUX                     PIC X.
000760 77 S2                      PIC X VALUE "/".
000770 77 EOF                     PIC X VALUE "N".
000780 77 OPC                     PIC 9 VALUE ZERO.
000790
000800 LINKAGE SECTION.
000810 77 CAPITULO                PIC 99.
000820
000830 PROCEDURE DIVISION USING CAPITULO.
000840 INICIO.
000850     OPEN INPUT SINTF02.
000860     MOVE SPACES TO REG-02  TABLA2  TABLA3.
000870     MOVE ZEROS TO I  OPC  KEYEVA.
000880     PERFORM PROCESO UNTIL OPC = 5.
000890
000900 CERRAR.
000910     CLOSE SINTF02.
000920
000930 FIN.
000940     EXIT PROGRAM.
000950
000960 PROCESO.
000970     IF OPC = ZERO PERFORM BUSQUEDA THRU ACEPTAR
000980     ELSE IF OPC = 1 PERFORM RETROCEDER
000990     ELSE IF OPC = 2 PERFORM SEGUIR
001000     ELSE IF OPC = 3 PERFORM CONTESTAR THRU C-ACEPTAR
001010     ELSE PERFORM EVALUACION THRU 180.
001020
001030 BUSQUEDA.
```

```
001040     MOVE CAPITULO TO CAP.
001050     READ SINTF02 RECORD INVALID KEY GO TO CERRAR.
001060     MOVE CONTEN TO TABLA3.
001070
001080 CONTINUAR.
001090     START SINTF02 KEY > KEYEVA
001100           INVALID KEY GO TO CERRAR.
001110
001120 SGTE-PANTALLA.
001130     READ SINTF02 NEXT.
001140     DISPLAY SPACES ERASE BEEP.
001150     MOVE KEYEVA TO W-KEYEVA.
001160     MOVE "N" TO EOF.
001170     PERFORM DISPLAYAR UNTIL (W-PANT NOT = PANT) OR
001180                             (W-CAP NOT = CAP) OR
001190                             (EOF = "S").
001200
001210 OPCION.
001220     DISPLAY GUION LINE 23 POSITION 1
001230             OPCIONES LINE 24 POSITION 1 SIZE 79.
001240
001250 ACEPTAR.
001260     ACCEPT OPC LINE 24 POSITION 5 PROMPT ECHO.
001270     IF OPC < 1 OR OPC > 5 GO TO ACEPTAR.
001280
001290 DISPLAYAR.
001300     DISPLAY CONTEN LINE LIN POSITION 1.
001310     IF TIPO = "*" MOVE "N" TO OK
001320                   PERFORM BUSCAR VARYING J FROM 1 BY 1
001330                                  UNTIL J > 5 OR OK = "S"
001340                   SUBTRACT 1 FROM J
001350                   IF (OK = "S") AND (T-L(J) NOT = LIN)
001360                      OR (OK = "N")
001370                      ADD 1 TO I
001380                      MOVE LIN TO T-L(I)
001390                      MOVE PANT TO T-P(I)
001400                   ELSE DISPLAY T-R(J) LINE T-L(J) POSITION 47.
001410     READ SINTF02 NEXT AT END MOVE "S" TO EOF.
001420
001430 RETROCEDER.
001440     IF EOF = "S" MOVE "N" TO EOF.
001450     IF W-PANT > 1 SUBTRACT 1 FROM W-PANT
001460                   MOVE ZEROS TO W-LIN
001470                   MOVE W-KEYEVA TO KEYEVA
001480                   PERFORM CONTINUAR THRU ACEPTAR
001490     ELSE PERFORM ACEPTAR.
001500
```

```
001510 SEGUIR.
001520     IF W-PANT < PG(W CAP
001530                                MOVE ZEROS TO W-LIN
001540                                MOVE
001550                                PERFORM MOSTRAR THRU ACEPTAR
001560     ELSE DISPLA                 ION 1 SIZE 79
001570                  "* NO HAY MAS PREGUNTAS PARA ESTE CAPITULO *"
001580                  LINE 24 POSITION 19 HIGH
           ACCEPT RES OFF LINE 24 POSITION 62
001600     PERFORM OPCION THRU ACEPTAR.
001610
001620 CONTESTAR.
001630     MOVE "N" TO OK.
001640     PERFORM BUSCAR VARYING J FROM 1 BY 1
001650                    UNTIL (J > 5) OR (OK = "S").
001660     IF OK = "N" GO TO C-ACEPTAR.
001670     SUBTRACT 1 FROM J.
001680     IF T-C(J) NOT = SPACE
001690        IF T-C(J) NOT < 3  DISPLAY
001700        SPACES LINE 24 POSITION 1 SIZE 79
001710      "NO ES POSIBLE.  SOLO TIENE TRES OPORTUNIDADES POR PREGUNTA"
001720        LINE 24 POSITION 10
001730        ACCEPT RES OFF LINE 24 POSITION 71
001740        GO TO C-ACEPTAR
001750        ELSE ADD 1 TO T-C(J)
001760     ELSE MOVE 1 TO T-C(J).
001770     DISPLAY SPACES LINE 24 POSITION 1 SIZE 79
001780             "ESTA SEGURO? (S,N) -->" LINE 24 POSITION 29.
001790
001800 VOLVER.
001810     ACCEPT T-R(J) LINE T-L(J) POSITION 47 PROMPT ECHO.
001820     IF T-R(J) < "A" OR T-R(J) > "E" GO TO VOLVER.
001830
001840 RESPUESTA.
001850     ACCEPT RES LINE 24 POSITION 53 PROMPT ECHO.
001860     IF RES = "N" GO TO VOLVER
001870     ELSE IF RES NOT = "S" GO TO RESPUESTA.
001880
001890 C-ACEPTAR.
001900     PERFORM OPCION.
001910     PERFORM ACEPTAR.
001920
001930 BUSCAR.
001940     IF T-P(J) = W-PANT  MOVE "S" TO OK.
001950
001960 EVALUACION.
001970     MOVE CAPITULO TO E.
```

```
001980     DISPLAY "EVALUACION CORRESPONDIENTE AL CAPITULO "
001990             LINE 1 POSITION 20 ERASE BEEP
002000             E LINE 1 POSITION 60
002010             "C = RESPUESTA CORRECTA  I = RESPUESTA INCORRECTA"
002020             LINE 3 POSITION 17 REVERSE
002030             GUION LINE 5 POSITION 17 SIZE 48
002040             S1 LINE 6 POSITION 18
002050             "R  E  S  P  U  E  S  T  A  S" LINE 6 POSITION 28
002060             "C" LINE 6 POSITION 61
002070             S2 LINE 6 POSITION 63
002080             S1 LINE 7 POSITION 20
002090             GUION LINE 7 POSITION 21 SIZE 39
002100             S2 LINE 7 POSITION 62
002110             S1 LINE 8 POSITION 22
002120             "A     B     C     D     E" LINE 8 POSITION 30
002130             S2 LINE 8 POSITION 61
002140             "I" LINE 8 POSITION 63
002150             S1 LINE 9 POSITION 24
002160             GUION LINE 9 POSITION 25 SIZE 39
002170             GUION LINE 21 POSITION 17 SIZE 48
002180             "P" LINE 10 POSITION 19
002190             "R" LINE 11 POSITION 19
002200             "E" LINE 12 POSITION 19
002210             "G" LINE 13 POSITION 19
002220             "U" LINE 14 POSITION 19
002230             "N" LINE 15 POSITION 19
002240             "T" LINE 16 POSITION 19
002250             "A" LINE 17 POSITION 19
002260             "S" LINE 18 POSITION 19.
002270     PERFORM VERTICAL VARYING K FROM 6 BY 1 UNTIL K > 20.
002280     MOVE 1 TO J.
002290     PERFORM RAYA VARYING L FROM 12 BY 2 UNTIL L > 20
002300             AFTER K FROM 30 BY 6 UNTIL K > 54.
002310
002320 100.
002330     MOVE ZEROS TO CONT.
002340     PERFORM PREGUNTA VARYING J FROM 1 BY 1 UNTIL J > 5.
002350     MOVE CONT TO E.
002360     DISPLAY "PUNTAJE OBTENIDO -->" LINE 23 POSITION 28
002370             E LINE 23 POSITION 49
002380             "/10" LINE 23 POSITION 51.
002390     IF CONT = 10 GO TO 160.
002400     DISPLAY "DESEA CORREGIR ALGUNA PREGUNTA (S/N)"
002410             LINE 24 POSITION 21.
002420
002430 110.
002440     ACCEPT RES LINE 24 POSITION 60 PROMPT ECHO.
```

```
002450     IF RES = "N" GO TO 140
002460     ELSE IF RES NOT = "S" GO TO 110.
002470     DISPLAY SPACES LINE 24 POSITION 1 SIZE 80
002480             SPACES LINE 24 POSITION 1 SIZE 79
002490             "NUMERO DE PREGUNTA A CORREGIR? (1 - 5)"
002500             LINE 23 POSITION 21
002510             "ESTA SEGURO? (S,N)"
002520             LINE 24 POSITION 1.
002530
002540 120.
002550     ACCEPT J LINE 23 POSITION 60 PROMPT ECHO.
002560     IF J < 1 OR J > 5 GO TO 120.
002570
002580 130.
002590     ACCEPT RES LINE 24 POSITION 50 PROMPT ECHO.
002600     IF RES = "N" GO TO 120
002610     ELSE IF RES NOT = "S" GO TO 130.
002620     MOVE CAPITULO TO W-CAP.
002630     MOVE R-PG(CAPITULO, J) TO W-PANT.
002640     MOVE ZEROS TO W-LIN.
002650     MOVE W-KEYEVA TO KEYEVA.
002660     GO TO 180.
002670
002680 140.
002690     DISPLAY SPACES LINE 24 POSITION 1 SIZE 79
002700             "DESEA CONOCER LAS RESPUESTAS CORRECTAS? (S,N)"
002710             LINE 24 POSITION 17.
002720
002730 150.
002740     ACCEPT RES LINE 24 POSITION 63 PROMPT ECHO.
002750     IF RES = "N" GO TO 160
002760     ELSE IF RES NOT = "S" GO TO 150.
002770     DISPLAY "LAS RESPUESTAS CORRECTAS PARA ESTE CAPITULO SON:"
002780             LINE 2 POSITION 17 ERASE BEEP
002790             "================================================"
002800             LINE 3 POSITION 17
002810             "PREGUNTA     RESPUESTA"
002820             LINE 5 POSITION 30
002830             "--------     ----------"
002840             LINE 6 POSITION 30.
002850     PERFORM DISP-PREGUNTA VARYING J FROM 1 BY 1 UNTIL J > 5.
002860
002870 160.
002880     DISPLAY SPACES LINE 24 POSITION 1 SIZE 79
002890             "<ENTER> PARA TERMINAR EVALUACION"
002900             LINE 24 POSITION 24.
002910
```

```
002920 170.
002930     ACCEPT RES OFF LINE 24 POSITION 57.
002940     IF RES NOT = SPACE GO TO 170.
002950     MOVE 5 TO OPC.
002960
002970 180.
002980     IF OPC NOT = 5
002990        PERFORM CONTINUAR THRU ACEPTAR.
003000
003010 DISP-PREGUNTA.
003020     COMPUTE K = 6 + J * 2.
003030     DISPLAY J LINE K POSITION 34
003040             T-RESP(J) LINE K POSITION 47.
003050
003060 PREGUNTA.
003070     IF T-R(J) = T-RESP(J) MOVE "C" TO RES
003080                           ADD 2 TO CONT
003090     ELSE MOVE "I" TO RES.
003100     COMPUTE L = 9 + J * 2.
003110     IF T-R(J) = "A" MOVE 1 TO K
003120     ELSE IF T-R(J) = "B" MOVE 2 TO K
003130     ELSE IF T-R(J) = "C" MOVE 3 TO K
003140     ELSE IF T-R(J) = "D" MOVE 4 TO K
003150     ELSE IF T-R(J) = "E" MOVE 5 TO K.
003160     IF T-R(J) NOT = SPACE COMPUTE K = K * 6 + 24
003170                           DISPLAY "X" LINE L POSITION K
003180                                   RES LINE L POSITION 62.
003190
003200 VERTICAL.
003210     IF K > 9 DISPLAY V LINE K POSITION 21
003220                      V LINE K POSITION 25
003230     ELSE IF K > 7 DISPLAY V LINE K POSITION 21.
003240     DISPLAY V LINE K POSITION 17
003250             V LINE K POSITION 60
003260             V LINE K POSITION 64.
003270
003280 RAYA.
003290     IF K = 30 COMPUTE CONT = L - 1
003300               DISPLAY J LINE CONT POSITION 23
003310               ADD 1 TO J.
003320     DISPLAY GUION LINE L POSITION K SIZE 1.
003330
003340 END PROGRAM.
```

BIBLIOGRAFIA

1. ANONIMO, Curso de Pascal, Escuela Superior Politécnica del Litoral, Guayaquil, 1981.

2. GROGONO, PETER, Programación en Pascal, Fondo Educativo Interamericano.

3. KOFFMAN, ELLIOT B., Pascal, a problem solving approach, Addison-Wesley Publishing Company, Massachusetts, 1982.