ESCUELA UNIVERSITARIA POLITECNICA DE LAS PALMAS DE GRAN CANARIA

TITULO: TECNICAS DE INGENIERIA DEL SOFTWARE. USO DEL PASCAL. APLICACION A UN PROGRAMA DE GESTION DE BIBLIOTECA.

Autor: Roberto Domínguez Rodríguez

Tutor: Sebastián Suárez Gil

PARTE I

TECNICA DE LA INGENIERIA DEL SOFTWARE

PAGINA

1 CONSIDERACIONES DEL SOFTWARE..... 1
1.1 COMPONENTES DEL SOFTWARE..... 1
1.2 APLICACIONES DEL SOFTWARE..... 6
1.2.1 SISTEMA SOFTWARE..... 6
1.2.2 SOFTWARE EN TIEMPO REAL..... 6
1.2.3 EMPLEO DEL SOFTWARE..... 6
1.2.4 INGENIERIA Y SOFTWARE CIENTIFICO..... 7
1.2.5 SOFTWARE COMBINACIONAL..... 7
1.3 LA IMPORTANCIA DEL ESTILO..... 8
1.4 LA CALIDAD DEL PROGRAMA..... 9
1.4.1 EL PROGRAMA DEBE FUNCIONAR..... 10
1.4.2 EL PROGRAMA NO DEBE TENER DIFICULTA - DES..... 11
1.4.3 EL PROGRAMA DEBE ESTAR BIEN DOCUMEN- TADO..... 12
1.4.4 EL PROGRAMA DEBE SER EFICIENTE..... 13
1.5 LAS FASES DEL PROCESO DE PROGRAMACION..... 15
1.6 EL DISEÑO TOP-DOW DE PROGRAMAS..... 19
1.7 ELEMENTOS DEL ESTILO DE PROGRAMACION..... 28
1.7.1 DISEÑO DE PROGRAMAS..... 28
1.7.2 CONSTRUCCION DE PROGRAMAS..... 33
1.7.3 PRESENTACION DE PROGRAMAS..... 36
2 ALGORITMOS DE FASES DE PROGRAMACION..... 39
- FASE DE PLANIFICACION..... 41
- FASE DE DESARROLLO..... 42
- FASE DE MANTENIMIENTO..... 43
3 MODELO DE FLUJO DE DATOS..... 44
1 DESARROLLO Y FLUJO DE INFORMACION..... 44
1.1 CONTRIBUIDORES..... 44
1.2 AREAS DE APLICACION..... 45
2 CONSIDERACIONES DEL DESARROLLO..... 47

PAGINA

2.1 TRANSFORMACION DE FLUJO..... 47
2.2 TRAMITACION DE FLUJO..... 49
2.3 UN PROCESO ABSTRACTO..... 50
4 MODELO DE ESTRUCTURA DE DATOS..... 51
1 MODELO Y ESTRUCTURA DE DATOS..... 51
1.1 CONTRIBUIDORES..... 51
1.2 AREAS DE APLICACION 52
1.3 ESTRUCTURA DE DATOS CONTRA EL FLUJO DE DATOS..... 53
2 DESARROLLO DE LOS DATOS..... 54
5 TESTEO Y DEPURADO DEL SOFTWARE..... 58
1 CARACTERISTICAS DEL TESTEO..... 58
1.1 OBJETIVOS DEL TESTEO..... 58
1.2 TEST EN EL FLUJO DE INFORMACION..... 59
1.3 PASOS EN EL TESTEO DEL SOFTWARE..... 61
1.3.1 UNIDAD DE TESTEO..... 62
1.3.2 TESTEO DE INTEGRACION..... 63
1.3.3 TESTEO DE VALIDACION..... 63
1.3.4 TESTEO DE SISTEMA..... 64
2 EL ARTE DEL DEPURADO..... 64
2.1 CONSIDERACIONES PSICOLOGICAS..... 64
2.2 ACERCANDONOS AL DEPURADO..... 65

PARTE II

PASCAL

PAGINA

0 INTRODUCCION..... 68
1 ESTRUCTURA DEL PROGRAMA PASCAL..... 70
1.1 ENCABEZAMIENTO..... 71
1.2 DECLARACION DE ETIQUETAS..... 72
1.3 DEFINICION DE CONSTANTES..... 72
1.4 DEFINICION DE TIPOS..... 73
1.4.1 TIPO SUBRANGO..... 73
1.5 DECLARACION DE VARIABLES..... 74
1.6 DECLARACION DE FUNCIONES Y PROCEDIMIENTOS.. 74
2 CONCEPTOS BASICOS..... 79
1 JERARQUIA DE LAS OPERACIONES EN PASCAL..... 81
2 ASIGNACION DE VARIABLES..... 81
3 ENTRADAS Y SALIDAS SIMPLES..... 83
3 TIPO DE DATOS E INSTRUCCIONES..... 87
1 TIPO DE DATOS (GRAFICA)..... 87
1.1 ESCALARES NORMALIZADOS..... 88
1.1.1 INTEGER (ENTEROS)..... 88
1.1.2 REAL (REALES)..... 88
1.1.3 BOOLEAN..... 88
1.1.4 CHAR (CARACTERES)..... 89
1.2 ESCALARES DEFINIDOS..... 89
1.2.1 ENUMERADOS..... 89
1.2.2 SUBRANGO..... 90
1.3 ESTRUCTURAS ESTATICAS..... 91
1.3.1 ARRAYS..... 91
1.3.2 RECORD..... 93
1.3.3 STRING..... 94
1.3.4 SET..... 95
1.4 ESTRUCTURAS DINAMICAS..... 96
1.4.1 FICHERO..... 96
1.4.2 PUNTERO..... 98

PAGINA

2 INSTRUCCIONES (GRAFICA)..... 100
2.1 INSTRUCCIONES SIMPLES..... 101
2.2 INSTRUCCIONES ESTRUCTURADAS..... 102
2.2.1 ESTRUCTURA SECUENCIAL..... 103
2.2.2 ESTRUCTURA ITERATIVA..... 103
2.2.3 ESTRUCTURA SELECTIVA..... 103
3 INDICE DE LOS PROGRAMAS..... 106
4 PROCEDIMIENTOS PREDECLARADOS..... 107
1 PROCEDIMIENTO DE MANIPULACION DE FICHEROS.... 107
1.1 RESET (f)..... 107
1.2 RESET (f, string)..... 107
1.3 REUNITE (f, string)..... 107
1.4 CLOSE (f)..... 108
1.5 PUT (f)..... 108
1.6 GET (f)..... 108
1.7 SEEK (f, integer)..... 108
1.8 READ (f, v1,...,vj) y READLN (f,v1,..,vj) 108
1.9 WRITE (") y WRITELN (") 109
1.10 PAGE (f)..... 110
2 FUNCIONES DE MANIPULACION DE FICHEROS..... 110
2.1 BUFFERREAD Y BUFFERWRITE..... 110
2.2 BLOCKREAD Y BLOCKWRITE..... 110
2.3 EOF (f) Y EOFLN (f)..... 111
2.4 IORESULT..... 111
3 PROCEDIMIENTOS DINAMICOS DE COLOCACION..... 112
3.1 NEW (p)..... 112
3.2 NEW (p, t1,...., tj)..... 112
3.3 MARK (p)..... 112
3.4 RELEASE (p)..... 112
3.5 MEMAVAIL..... 112
4 FUNCIONES ARITMETICAS..... 113
4.1 ABS..... 113
4.2 SQR..... 113
4.3 SEN, COS, LOG, EXP, LN, SQRT, ARCTAN, TRUNC, ROUND..... 113
5 ATRIBUTOS..... 114

PAGINA

6 FUNCIONES DE TRANSFERENCIA..... 114
6.1 ORD (x)..... 114
6.2 CHR (x)..... 114
7 RUTINAS DIVERSAS..... 114
7.1 SUCC (x)..... 114
7.2 PRED (x)..... 114
7.3 SIZEOF..... 115
7.4 EXIT..... 115
7.5 GOTOXY..... 115
7.6 SETPOINTER..... 115
7.7 PORTINPUT..... 115
7.8 PORTOUT..... 116
8 RUTINAS DE MANIPULACION DE CADENAS..... 116
8.1 LENGHT..... 116
8.2 POS..... 116
8.3 CONCAT..... 116
8.4 COPY..... 116
8.5 INSERT..... 116
8.6 DELETE..... 117
9 RUTINAS DE MANIPULACION DE ARRAY DE CARACTERES..... 117
9.1 SCAN..... 117
9.2 MOVELEFT Y MOVERIGHT..... 118
9.3 FILLCHAR..... 118
5 INSTRUCCIONES DE OPERACION..... 129
1 MODO DE EMPEZAR A TRABAJAR CON EL PASCAL 80
1.1 PASOS INICIALES..... 130
1.2 COMPILADOR..... 131
1.3 RUN-TIME-SISTEM..... 131
1.4 LIBRERIA..... 132
1.5 PROGRAMAS FIGURAS..... 132
1.6 DISTRIBUCION DE LA MEMORIA PARA EL RUN-TIME-SISTEM..... 132
2 OPERACIONES CON EL SISTEMA "PASCAL-80"..... 134
2.1 LLAMANDO AL PASCAL-80..... 134
2.2 COMANDO DE SINTAXIS DE LINEA PARA EL

PAGINA

"PASCAL-80"..... 135
2.2.1 ESPECIFICACIONES EN CUANTO AL NOMBRE DEL FICHERO CODIGO..... 136
2.3 VENTAJAS PARA PROGRAMAS QUE EMPLEA TRAZO..... 137
2.3.1 EL FLAG TRACE..... 137
2.3.2 INSTRUCCIONES DE TRAZO..... 137
2.4 VISUALIZACION DEL RUN-TIME..... 138
2.5 VISUALIZACION DEL STATS..... 140
2.6 EJECUCION AUTOMATICA DE UN PROGRAMA... 141
2.7 INTERRUPCION DE UN PROGRAMA EN EJECUCION..... 141
3 OPERANDO CON EL COMPILADOR DEL PASCAL-80.. 142
3.1 SINTAXIS DE LOS COMANDOS DE LINEA PARA EL COMPILADOR..... 142
3.2 DIRECTIVOS DE COMPILACION..... 143
3.3 DIRECTIVOS DE COMANDOS DE LINEA PARA EL COMPILADOR..... 143
3.3.1 COMBINACIONES EXCLUIDAS..... 145
3.4 DIRECTIVOS ENCAJADOS..... 145
3.5 SUMARIO DE INFORMACION SOBRE LA PANTALLA..... 147
3.6 FORMATO DEL LISTADO DEL COMPILADOR.... 148
3.7 COMPILACION INICIAL..... 149
6 COMPILACION SEPARADA, UNION DE PROGRAMAS, RELOCALIZACION Y EJECUCION..... 150
1 PARTICION DE UN PROGRAMA PASCAL..... 150
1.1 ESTRUCTURA DE UN PROGRAMA PARTIDO..... 151
1.2 CONSTRUCCION DE UN FICHERO CODIGO..... 155
2 UNION CON MODULOS NO PASCAL..... 157
2.1 EXTENSION DEL PASCAL-80..... 158
2.2 LLAMANDO A UNA FUNCION O PROCEDIMIENTO EXTERNO..... 162
2.3 TABLA DE REFERENCIA EXTERNA..... 164
3 EJECUCION CON OBJETOS EXTERNOS..... 165
3.1 COMPILANDO UN PROGRAMA PASCAL..... 166

PAGINA

3.2 COMPILAR Y/O ENSAMBLAR LOS MODULOS EXTERNOS..... 167
3.3 LINCAJE DE TODOS LOS MODULOS..... 167
3.4 COLOCACION DE LOS MODULOS EXTERNOS... 168
3.5 CARGA DEL MODULO EXTERNO EN MEMORIA Y LLAMADA AL PASCAL-80 RTS..... 170
4 GENERACION DE UN PROGRAMA EN LA FORMA CARGA Y EJECUCION (LOAD AND GO)..... 171
4.1 GENERACION DE UN MODULO OBJETO RELOCALIZABLE..... 172
4.2 LINCADO CON MODULOS OBJETOS..... 174
4.3 LOCALIZANDO A UN MODULO OBJETO..... 175
4.4 EJECUCION DE UN PROGRAMA..... 176

PARTE III

APLICACION DEL PASCAL A UN PROGRAMA PARA LA GESTION DE UNA BIBLIOTECA

PAGINA

1 ANALISIS DEL PROGRAMA..... 177
2 LISTADO DEL PROGRAMA..... 185

PAGINA

APENDICE A

LISTADOS DE PROGRAMAS..... 197

APENDICE B

MENSAJES DE ERROR DEL COMPILADOR..... 223

APENDICE C

MENSAJES DE ERROR DEL RUN-TIME..... 226

APENDICE D

MENSAJES DE ERROR DEL ISIS-II..... 228

OBJETIVO

Con la aparición de los microprocesadores la labor del ingeniero en el diseño de sistemas se ha ampliado a la totalidad de las ramas de la industria, la automatización de procesos, es un ejemplo claro y evidente de este hecho.

El diseño de sistemas basados en micropocesadores tiene dos caminos distintos entre si, pero intimamente ligados, el 'hardware' y el 'software'. Y es a este último al que dedicaremos la parte primordial de este proyecto. En un principio para el software de los sistemas con microprocesadores se empleaban lenguajes de bajo nivel conocidos como 'Assembly', los cuales se consideran de bajo nivel por estar cerca del lenguaje máquina. Pero para todo aquel que ha trabajado con estos lenguajes, no se esconde, que el proceso de desarrollo del software con este tipo de lenguajes es lento y muy laborioso.

Hoy en día se pretende en todo tipo de investigaciones y creaciones, que estas sean rápidas y precisas. El desarrollo de sistemas basados en microprocesadores, naturaleente debe cumplir estas premisas. Y aparecieron entonces los lenguajes de alto nivel aplicados a los microprocesadores.

El objeto de este proyecto consiste en hacer un estudio de las técnicas de la ingenieria del software como son: componentes del software, aplicaciones del softwa-

re, la importancia del estilo, la calidad del programa, las fases del proceso de programación, el diseño 'top-dow' de programas, elementos del estilo de programación, modelo de flujo de datos y de estructura de datos, y testeo y depurado del software.

Seguidamente se rezliza un estudio del Pascal-80 dada la necesidad de crear una especie de manual, donde se encuentre contenida toda la informaciónm existente sobre este lenguaje, de forma que cualquier interesado, pueda facilmente aprender este lenguaje y al mismo tiempo conocer todas sus caracteristidas ante de hacer ejecutar un programa.De esta forma, este proyecto podría servir tanto para ayudar a la realización de proyectos, como para estudiar este tipo de lenguaje y ver sus características.

Por último se hace una aplicación del Pascal-80 a un programa para llevar la gestión de una biblioteca.

PARTE I

CONSIDERACIONES DEL SOFTWARE

1 CONSIDERACIONES DEL SOFTWARE

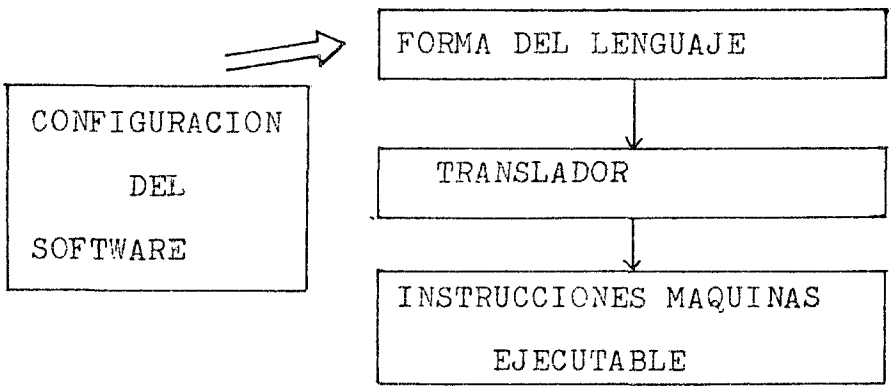
El software de una computadora es más bien un sistema lógico que un sistema físico. Sin embargo, el software tiene caracteristicas que son considerablemente diferentes del hardware. Estas son por ejemplo:

- No hay fase de manufacturación para el software; todos los costos hacen referencia a planificación y desarrollo.
- El software no consume; hay pocas partes de ahorro en el mundo del software.
- El software incluye frecuentemente modificaciones y encarecimientos.

1.1 COMPONENTES DEL SOFTWARE

El software de las computadoras es una información que existe en dos formas básicas: componentes máquina ejecutable y componentes máquina no ejecutable.

En cada componente máquina hay encerrados tres clases de software que son: una forma de lenguaje que especifica la estructura de datos, un 'traslador' que se encarga de procesar el lenguaje anterior para convertirlo en instrucciones máquinas ejecutables que sería la tercera clase del software.



COMPONENTES DEL SOFTWARE

Idealmente, el mundo humano se comunica con computadoras por el uso natural del lenguaje (por ejemplo Inglés Español, Ruso). Desafortunadamente, los largos vocabularios y la sofisticada gramática, así como nuestro uso del contexto para el entendimiento, impide que se utilize el lenguaje natural en las computadoras. Durante la década de los 70 se intentó utilizar un lenguaje natural como comunicación para las computadoras pero no dió resultado, por lo que actualmente todas las especificaciones de programas están limitadas a lenguajes artificiales.

Todos los lenguajes de programación son lenguajes artificiales. Cada uno tiene un vocabulario limitado, gramática explícitamente definida y rutas bien formadas para la sintáxis y la semántica. Estos atributos son

necesarios para la translación al lenguaje máquina o instrucciones máquinas. Las formas de lenguajes, que son un componente del software, están caracterizadas por lenguajes máquinas y lenguajes de alto nivel.

El lenguaje máquina es una representación simbólica del grupo de instrucciones de la CPU. Cuando se desarrolla un buen software da lugar a un programa bien documentado y sostenible de tal manera que se puede hacer un uso eficiente de la memoria y, por lo tanto, optimizar la velocidad de ejecución de un programa. Pero cuando un programa cuenta con un software pobre y poco documentado, el lenguaje máquina tiende a dar bastantes problemas.

Aún cuando el lenguaje máquina tenga una velocidad de ejecución atractiva y carecterísticas de memoria inmejorables, presenta unas ciertas desventajas como son:

- La implementación en el tiempo está protegida.
- Los resultados del programa son difícil de leer.
- El testeo resulta complicado.
- El mantenimiento es extremadamente difícil.

La productividad del software se ve seriamente alterada cuando se usa el nivel del lenguaje máquina. Todas las desventajas apuntadas anteriormente hacen que la forma de este lenguaje desaparezca muy probablemente durante la próxima década.

Los lenguajes de alto nivel permiten que el software se desarrolle de tal manera que sea independiente de la máquina. Cuanto más sofisticado es el 'traslador' usado, el vocabulario, la gramática, la sintaxis y la semántica, sirven para que el lenguaje de alto nivel pueda ser más eficiente que el lenguaje máquina. De hecho, el compilador y intérprete (traslador del lenguaje de alto nivel) produce un código máquina de salida.

En la actualidad existen alrededor de unos 200 lenguajes de programación de alto nivel, pero unos pocos, alrededor de 10, son los que se usan en la industria. Estos lenguajes pueden ser divididos en tres categorias.

a) Lenguajes iniciales. Desarrollados al final de la década de los 50 y principio de los 60, estos lenguajes forman el inicio del desarrollo general científico y comercial. FORTRAN y COBOL son representaciones de esta categoría. ALGOL también puede ser considerado un lenguaje de principio aunque por su forma de estructura está en la siguiente categoría.

b) Lenguajes estructurados. Estos lenguajes surgieron como caída de los lenguajes iniciales. Las caracteristicas principales de estos lenguajes son:

- Estructura de datos sofisticadas.
- Definición de subprogramas.

- Definición del grupo de sentencias(bloques estructurales).

- Construcción lógica.

Los principales lenguajes que podemos considerar en esta categoría son: ADA,ALGOL,PL/1,PL/M,PASCAL y C.

c) Lenguajes especializados. Estos lenguajes proporcionanunas características especiales para la aplicación de un software específico y son especiales para valorar lo unusual o inconveniente de una forma de lenguaje. Los principales lenguajes que podemos considerar dentro de esta caracteristica son: APL,BLISS, LISP,RPG y SNOBOL.

Ya hemos visto que la función del 'traslador' es transformar la forma del lenguaje en instrucciones máquina ejecutables. Un 'ensamblador' es el 'translador' para el nivel del código máquina,ejecutando tareas simples de convertir las instrucciones máquinas en instrucciones máquinas ejecutables. Un 'intérprete' es un translador que se usa para transformar un lenguaje de alto nivel en sentencias básicas que pueda entender el ordenador. Cuando se encuentra cada sentencia la convierte en un código máquina ejecutable y la ejecuta. El APL y el BASIC están entre los lenguajes que usualmente son ejecutados con un intérprete. El translador más común de un lenguaje de alto nivel es el 'compilador'.

1.2 APLICACIONES DEL SOFTWARE

1.2.1 SISTEMA SOFTWARE

Un sistema software es una colección de programas escrita al servicio de otros programas. Algunos sistemas software son: compiladores, editores, y ficheros útiles de manejo,los cuales procesan determinadas estructuras de información. Otros sistemas de aplicación como son: componentes de sistemas operativos,drivers.. procesan largas estructuras de datos.

1.2.2 SOFTWARE EN TIEMPO REAL

Al software que analiza y controla el mundo real, incluso lo que en él ocurre se le llama tiempo-real. El software en tiempo real incluye una colección de datos que recoge del medio externo lo siguiente: un análisis de los datos recogidos para transformalos en información, según sea requerido para una determinada aplicación, un control de salida que sirve para responder al medio externo, y un componente que coordine todo lo anterior para que la respuesta en tiempo real (el rango típico va desde 1milisegundo a 1 minuto) pueda ser mantenida. Un sistema en tiempo real debe responder sin tiempo de contracción.

1.2.3 EMPLEO DEL SOFTWARE

En el procesamiento de información es don

de más aplicación tiene el software. Los sistemas discretos (inventarios, facturas de pago/cobro) están envueltos en un software de manejo de información que accede a una o más bases de datos donde está contenida la información.

1.2.4 INGENIERIA Y SOFTWARE CIENTIFICO

La ingeniería y el software científico está caracterizado por los algoritmos numéricos. Las aplicaciones las encontramos en la astronomía, física nuclear, biología molecular, manufacturaciones automáticas,.. Sin embargo, nuevas aplicaciones en el área científica y de la ingeniería están alejando los convencionales algoritmos numericos.

1.2.5 SOFTWARE COMBINACIONAL

El software combinacional hace uso de algoritmos no números para poder acercanos a problemas que requieren una inteligencia artificial. Poder imitar la imagen y la voz, representa algunos de los problemas que se resuelven con este software.

1.3. LA IMPORTANCIA DEL ESTILO

Las personas creativas como los artistas, escritores y arquitectos, trabajan con mucha intensidad durante el período de estudios para llegar a manejar los secretos de su oficio. Al mismo tiempo, desarrollan un estilo que es único e identificable para cada uno de ellos. Este estilo no es accidental para llegar a triunfar en su trabajo; para que tengan éxito, es preciso que su estilo sea preferido en el mercado por encima de sus demás colegas.

En cualquier campo, ciertos estilos tienen ventajas definidas. Por ejemplo, ciertos estilos de música o de arte tienen más aceptación que otros. Existen estilos definidos de escribir que pueden comunicar las ideas con más efectividad. Otros pueden ser mejores para comunicar detalles técnicos. Ciertos estilos de arquitectura son más apropiados que otros para soportar determinadas condiciones climáticas.

El estilo también tiene importantes consecuencias en programación, algunas de las cuales demostraremos. Además de las cuestiones relativas a los estándares profesionales, algunas consideraciones sobre el estilo pueden ayudar a mejorar la calidad de los programas; por ejemplo, la investigación ha demostrado que ciertas prácticas estilísticas pueden servir para reducir el número de errores que se presentan durante el desarrollo del programa. Al mismo tiempo el programa mismo resulta más fácil de leer y de compren-

der por otros programadores,quienes pueden en algún momento ser llamados para realizar modificaciones del mismo.El mantenimiento del programa,es decir,él "ajuste" de los programas que existen para reunir requerimientos siempre cambiantes ,consume gran parte del tiempo del trabajo de los programadores profesionales;es muy común,de hecho,que se emplee más tiempo en el mantenimiento de un programa que en su desarrollo original.No debe sorprender ,entonces,que tanto los programadores como quienes los dirigen estén muy interesados en que la actividad de mantenimiento consuma menos tiempo y que ,por tanto,el programador quede libre para realizar un trabajo más original y creativo.

El buen estilo de programación puede ser una importante contribución al éxito del programador.Las recompensas son muy tangibles,pero se requiere un esfuerzo consciente por parte del programador.

1.4. LA CALIDAD DEL PROGRAMA

Antes de seguir adelante en las consideraciones de los métodos que existen para mejorar la calidad de los programas de computación,sería conveniente tratar de definir qué es lo que se está buscando.Esto es,sin embargo,evasivo.Si se interrogara a muchos programadores cuáles son las características de un buen programa,probablemente se recibiría una gran variedad de respuestas,según el gusto personal y la experiencia de las personas entrevistadas

Sin embargo,cierta clase de respuestas pueden presentarse con mayor frecuencia.Analizaremos las respuestas más comunes.

1.4.1. EL PROGRAMA DEBE FUNCIONAR

Nunca debe olvidarse que la característica más simple e importante de un programa es que funcione . Esto puede parecer obvio,pero es difícil de asegurar en los programas de tamaño considerable.Circulan en el medio muchas historias de errores catastróficos y muy costosos de programas de cómputo.Tales incidentes no sólo reflejan las fallas de la industria como un todo,sino que también contribuyen en gran medida a robustecer el recelo y la desconfianza que tiene el público en las computadoras.

Los programadores deben ser muy cuidadosos en el sentido de que el programa instalado en el sistema sea efectivamente el que se necesita.Es muy fácil sumergirse de tal manera en los detalles que llegue a perderse el concepto original de las especificaciones.La solución del problema plantado puede originar incidentes desafortunados y clientes insatisfechos,en el caso de que se de una solución casi completa.Es muy importante que las especificaciones(que pueden,de hecho,variar con el tiempo)sean continuamente revisadas durante las fases de diseño e instalación del programa.Las especificaciones mismas pueden ser erróneas o incompletas,o simplemente pueden ser malinterpretadas. Debe

tenerse cuidado de no hermosear el programa con características no pedidas en forma específica(pero divertidas) porque esto significa una fuente adicional de error.

1.4.2.EL PROGRAMA NO DEBE TENER DIFICULTADES

Muchos programadores aceptan las pequeñas dificultades (o errores) de los programas como una consecuencia natural de la condición humana,y consideran su corrección como una situación inevitable de la vida.Sin embargo,no hay ninguna razón para que esto suceda.Las imágenes de diabólicos duendes que en forma muy secreta introducen errores en los programas asaltan con frecuencia la imaginación cuando,de hecho,son los mismos programadores los responsables frecuentes de ellos por descuido,falta de comprensión de las especificaciones o de anticipar alguna situación particular en la cual va a emplearse el programa.Los programadores nunca hacen esto en forma deliberada (nadie disfruta rastreando errores)y,de hecho,están en condiciones de evitar la mayor parte de ellos.Se ha hablado de un estilo de desarrollar los programas cuya filosofía consiste en tratar de evitar los errores desde el comienzo.

A pesar de que esto puede hacerse,es evidente que es responsabilidad del programador asegurar que el programa está libre de errores.Buena parte de la investigación en ciencia de la computación ha estado dirigida hacia la búsqueda de pruebas formales,matemáticas,de la validez de los pro-

gramas.Se ha establecido que estas pruebas son posibles; sin embargo,los procedimientos son largos y dificultosos, y pocas veces prácticos en las aplicaciones reales.Cuando éste sea el caso,el programador debe recurrir a otros métodos,como las pruebas para establecer la validez de sus programas.

1.4.3. EL PROGRAMA DEBE ESTAR BIEN DOCUMENTADO

Es muy importante que el programa de computadora esté bien documentado.La documentación existe para ayudar a comprender o usar un programa.Esto no sólo es importante para los encargados de dar mantenimiento o modificar los programas;también puede ser de gran valor para el programador mismo.La mayoría de los programadores se ven forzados a prestar atención simultánea a diferentes asuntos,ya sean programas distintos,diversas partes de un mismo programa e,incluso,diferentes tareas de su trabajo. Los detalles de los programas en particular,o algunas partes especiales de los mismos,pueden olvidarse fácilmente o confundirse si no se tiene la documentación apropiada.

La documentación puede tenerse en dos formas:la "documentación externa" ,que incluye cosas como manuales de referencia,descripciones de los algoritmos,diagramas de flujo, proyectos de trabajo y aspectos similares;y la "documentación interna " que aparece en la lista de instrucciones misma del programa(esencialmente,el código del programa ,

más algunos comentarios).El valor de la documentación interna no debe sobrevaluarse.Para un programa cualquiera, la lista mínima de instrucciones constituye la primera línea de la documentación ,por lo cual se ha hecho incapié en la importancia de que las listas de programas sean fácilmente legibles.La documentación externa está dirigida con más frecuencia a los usuarios del programa,quienes no necesitan ni quieren saber nada del código del programa,y sólo desean conocer qué hace el programa y cómo trabaja. La documentación externa proporciona una importante descripción complementaria.

1.4.4. EL PROGRAMA DEBE SER EFICIENTE

El asunto de la eficiencia es muy espinoso. En los primeros días de la computación,las máquinas eran lentas y pequeñas en comparación con los estándares actuales.Los programas tenían que ser diseñados con mucho cuidado para aprovechar al máximo los escasos recursos disponibles de tiempo y espacio de almacenamiento.Los programadores debían gastar horas tratando de eliminar secciones y ahorrar con ello tiempo de ejecución de sus programas o debían comprimir los programas en un pequeño espacio de memoria de la máquina.La eficiencia de un programa,determinada las más de las veces por medio de un"producto espacio-tiempo" ,constituía el mérito primero.

Hoy la situación ha cambiado en forma drástica.Los recur

sos del "hardware" han ido disminuyendo continuamente,mientras que los recursos humanos se han elevado.El tiempo de ejecución y el espacio disponible de memoria ya no son recursos tan escasos como antes.Se debe estar siempre atento a realizar ahorros que podrían derivarse de la inclusión de una técnica diferente de solución,como la elección hecha para reemplazar la técnica de búsqueda lineal por la más eficiente de búsqueda binaria,aunque no siempre significa un ahorro apreciable para el programador tratar de mejorar cada detalle en aras de la eficiencia de su programa. No obstante que esto podría ser de mucha utilidad en algunos casos muy especiales,por lo general un esfuerzo de esta naturaleza no siempre se justifica.

A pesar de ello,aún hay muchos programadores que siguen cultivando la numeración del producto espacio-tiempo,y como resultado producen un código de programación innecesariamente complejo.Un programa que no trabaje o que sea difícil de mantener debido a este código tan rebuscado es sin duda alguna un programa de baja calidad,independientemente de su producto espacio-tiempo.

Como puede verse,la calidad de un programa tiene muchas facetas.Sin duda es importante que un porgrama trabaje correctamente y que sea confiable,es decir,que reúna todos los requisitos exigidos y que los errores inesperados ocurran muy rara vez.Sin embargo,el asunto no concluye aquí. La evolución de los programas parece ser un fenómeno real.

Los programas parecen necesitar un procesamiento continuo de mantenimiento y modificación para mantenerse al día con los requerimientos cambiantes de la tecnología y con la instalación de ellos en las máquinas.Las capacidades de modificación y mantenimiento son características esenciales de los programas reales.Que un programa sea fácil de leer y de comprender son prerrequisitos importantes para lograr su mantenimiento y su modificación apropiados.En resumen, se desean programas correctos,confiables,fáciles de mantener,modificables,legibles y comprensibles.

1.5. LAS FASES DEL PROCESO DE PROGRAMACION

Cualquier consideración del proceso de programación mismo debe comenzar aislando cada una de sus fases componentes.Podemos identificar las siguientes cinco fases:

1. Análisis del problema
2. Desarrollo de la solución
3. Construcción de la solución en forma de programa
4. Prueba
5. Mantenimiento

El "análisis del problema" se refiere a la etapa del proceso en la que el programador toma conocimiento del problema antes de proceder a desarrollar una solución.Es un proceso de "introducción" ,de naturaleza cognoscitiva y

muy difícil de describir.Son demasiados los programadores que recorren esta etapa muy rápidamente,lo que hace que entiendan mal o malinterpreten las especificaciones.Algunos programadores prefieren devolver las especificaciones del problema al analista o diseñador,para reducir la posibilidad de malentendido.Los errores que se cometen en esta etapa son con mucha frecuencia difíciles de detectar y consumen mucho tiempo cuando se les trata de remediar en las etapas posteriores.

La segunda etapa ,el"desarrollo de la solución" ,es eminentemente creativa.A lo largo del tiempo se ha dado mucha importancia a la separación de esta etapa de la de instalación o construcción;hay una desafortunada tendencia por parte de muchos programadores a sucumbir al engañoso atractivo de la máquina,iniciando la fase de construcción e instalación antes de que el problema haya sido resuelto realmente.

La tercera etapa identificada es la "construcción de la solución" desarrollada en el paso 2 "en forma de un programa real(o código)" .Considerando que la solución ha sido bien definida ,este proceso es completamente mecánico,pues es un proceso mental directo.Mediante varias reglas del lenguaje de programación ,el programa mismo puede ensamblarse de acuerdo con determinados estándares de estilo y estructura.El estilo y la estructura deben verse como una ayuda en la producción de un programa correcto,en lugar de

considerarlos como ideas nuevas que deben agregarse a un programa que trabaja ya.

La cuarta fase se refiere a la demostración de la corrección del programa instalado en la máquina.Es inevitable que algunas pruebas deban realizarse como parte de las etapas 2 y 3. Todo programador experto prueba mentalmente cada instrucción cuando la está escribiendo,y simula mentalmente la ejecución de cualquier módulo o sección de su programa,ántes de proceder a una prueba real de la etapa. La prueba de los programas nunca es sencilla.Aun cuando las pruebas muestran la presencia de errores,nunca pueden demostrar la ausencia de ellos.Una prueba con éxito sólo significa que no se detectaron errores bajo las circunstancias especiales de dicha prueba;esto no significa nada frente a otras circunstancias. En teoría,la única manera en que las pruebas pueden demostrar que un programa es correcto es que se examinen todos los casos posibles(lo cual se conoce como una prueba exhaustiva),situación que es imposible técnicamente,incluso para los programas más simples. Supóngase,por ejemplo,que se está escribiendo un programa para calcular la calificación promedio de un examen.Una prueba exhaustiva debería tomar en cuenta todas las combinaciones de calificaciones y el tamaño de los grupos;esto requeriría varios años para que la prueba estuviese completa.

Esto no significa que las prueban sean inútiles.El pro-

gramador puede hacer mucho por reducir el número de casos a probar a partir del número requerido por una prueba exhaustiva. Tomando con mucho cuidado y seleccionando apropiadamente el diseño de los casos de prueba,puede reducirse el número de los casos,haciendo posible una prueba con un número relativamente pequeño de ellos.

La prueba de un programa es una tarea tan creativa como su mismo desarrollo, por lo que debe considerarse con la misma diligencia y entusiasmo.Algunos principios de las pruebas son claros: trátese de iniciar las pruebas de un programa con una mentalidad de saboteador, casi disfrutando de la tarea de buscar un error. Hay que sospechar de todo.Los casos de prueba deberían diseñarse a partir de las especificaciones originales, en lugar del programa mismo; si se efectúan a partir del programa,algunos aspectos del problema que han sido pasados por alto durante su construcción también lo serán cuando se le pruebe. Para reducir las posibilidades de que esto ocurra en las compañías profesionales de programación, los encargados suelen insistir en que sean personas diferentes a los programadores originales quienes tengan a su cargo la prueba de los programas. Los usuarios de los programas disponen,con frecuencia,de sus propios datos de prueba desarrollados, independientemente,para usarlos cuando el programa llegue a sus manos. Tómese en cuenta que los encargados conceptúan muy mal a los programadores cuyos programas no son

capaces de pasar las pruebas de los clientes, ya que esto es un mal reflejo de la organización completa y puede afectar su reputación en el mercado. De cualquier forma que se haga, una prueba completa es una parte esencial de cualquier proyecto de programación.

Cuando los principiantes programan, muy rara vez por desgracia, se ven implicados en la quinta etapa del proceso de programación,el "mantenimiento del programa".Sin embargo,su importancia en el trabajo real nunca debe despreciarse.Al contrario de lo que sucede con el mantenimiento del hardware, el mantenimiento de los programas no se refiere a la reparación o cambio de partes deterioradas, sino a las modificaciones que deben hacerse a los defectos del diseño, lo cual puede incluir el desarrollo de funciones adicionales para cubrir nuevas necesidades. La habilidad de los programadores para producir nuevos programas se ve afectada claramente por el tiempo que deben dedicar al mantenimiento de los programas viejos. La inevitabilidad del mantenimiento debe reconocerse y, en consecuencia,deben realizarse las acciones que sean necesarias para reducir el tiempo que ello implica.

1.6. EL DISEÑO "TOP-DOWN" DE PROGRAMAS

La programación es sin duda una actividad compleja,pues en ella se combinan muchos procesos mentales. Deben reunirse muchos factores en la producción de

un programa final. Tal vez, la tarea no sea muy diferente de la de un malabarista;si éste trata de mantener demasiadas pelotas en el aire a un tiempo, antes de lo esperado se estrellarán contra el suelo.

La solución de cualquier problema puede darse en varias formas o,como se les llamará,"niveles de abstracción".

Se comenzará el proceso de solución con un enunciado muy general o abstracto de la solución del problema, expresado en términos del problema mismo. A continuación , se procederá a refinar esta solución elaborando los detalles que se habían ignorado previamente, de lo que resulta una solución nueva que es mucho menos abstracta. Este proceso continúa a través de un cierto número de etapas cada vez más refinadas, hasta que se ha logrado un nivel de detalles apropiado. Esta es la esencia del diseño top-down. Se trabaja a partir de una solución muy abstracta, (el nivel inicial -top) hasta llegar a una construcción final, mediante una serie de refinamientos sucesivos. Este enfoque es independiente de cualquier lenguaje de programación;de hecho,se está programando"dentro"de un lenguaje de programación,más que en"uno".

Este diseño top-down es una técnica que han aplicado durante muchos años los buenos programadores. Sólo en fechas recientes se le ha dado nombre.De hecho,la misma idea la han denominado de diferentes formas desde los años sesenta: "refinamiento por pasos","modelado iterativo

multinivel" , "programación jerárquica" . Este enfoque es atrayente porque puede definir una estructura para el proceso no estructurado del desarrollo de programas. La atención se concentra en el diseño ,en vez de hacerlo en los detalles de la construcción y, por tanto,limita el número de "pelotas" que será necesario tratar a un mismo tiempo. Como toda herramienta,su uso es más eficiente cuando se tiene alguna práctica con ella. El sentido común ,la intuición y la creatividad continúan siendo atributos valiosos del programador.

Vamos a ejemplificar la aplicación del diseño top-down en la solución de un problema.Este problema consiste en leer un número N y dar una lista de los cuadrados perfectos que hay entre 1 y N (se supone que todos los números son enteros positivos) .Por ejemplo, si N es 30, se obtendrá la siguiente lista de cuadrados: 1,4,9,16,25.

Se comenzará con una solución muy abstracta de este problema, la que se enuncia simplemente así :

Inprímase una lista de los cuadrados perfectos entre 1 y N.

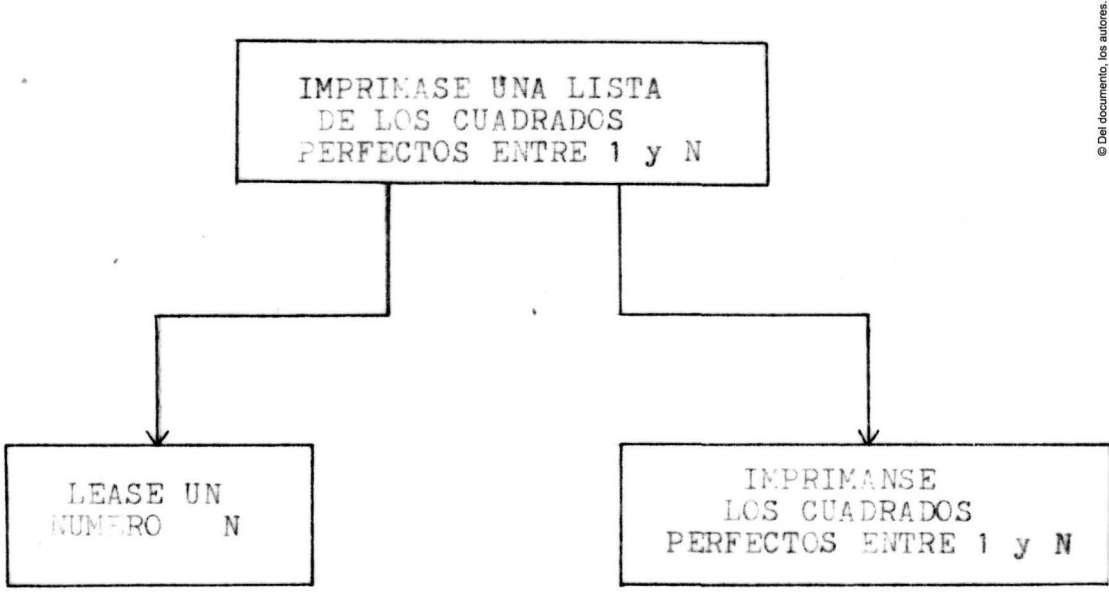
Esto describe qué es lo que se desea hacer, pero no dice nada de cómo debe realizarse.En este nivel de diseño no se está tratando de resolver estos detalles;se resolverán cuando la solución básica se refine, que será lo que a continuación se haga.

Como primera medida de la refinación, se procederá a dividir la solución en dos pasos o módulos:

Léase un número N

Imprímanse los cuadrados perfectos entre 1 y N.

Para acentuar la relación jerárquica que existe entre estos dos módulos y el módulo original, se ha escogido representarlos como se muestra en la figura. Cada nivel de la figura representa uno de los niveles de abstracción ya mencionados, donde el nivel de mayor abstracción se encuentra arriba. Las líneas de conexión acentúan los refinamientos realizados; el módulo superior ha sido refinado en dos módulos en el siguiente nivel.

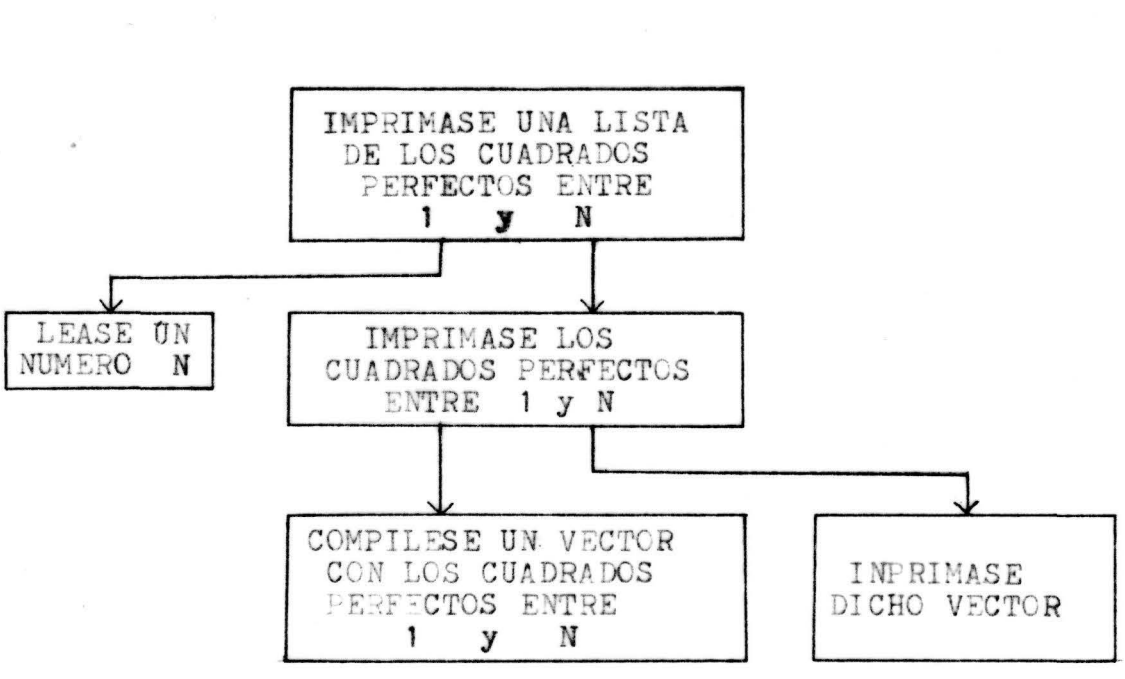


Ahora se procederá a refinar los módulos separados del segundo nivel de abstracción, de nuevo mediante la especificación de cómo deben realizarse. El primero de ellos (o,en el diagrama, el de la extrema izquierda) ya está suficientemente detallado como para programarlo de inmediato (en forma de un Read(N) del lenguaje algorítmico), por lo que ya no es necesario continuar refinándolo. Luego se considera el segundo módulo(el de la extrema derecha), el cual se dividirá de la siguiente forma:

Compílese un vector con los cuadrados perfectos entre 1 y N

Imprímase dicho vector

La figura resultante en forma de diagrama es:



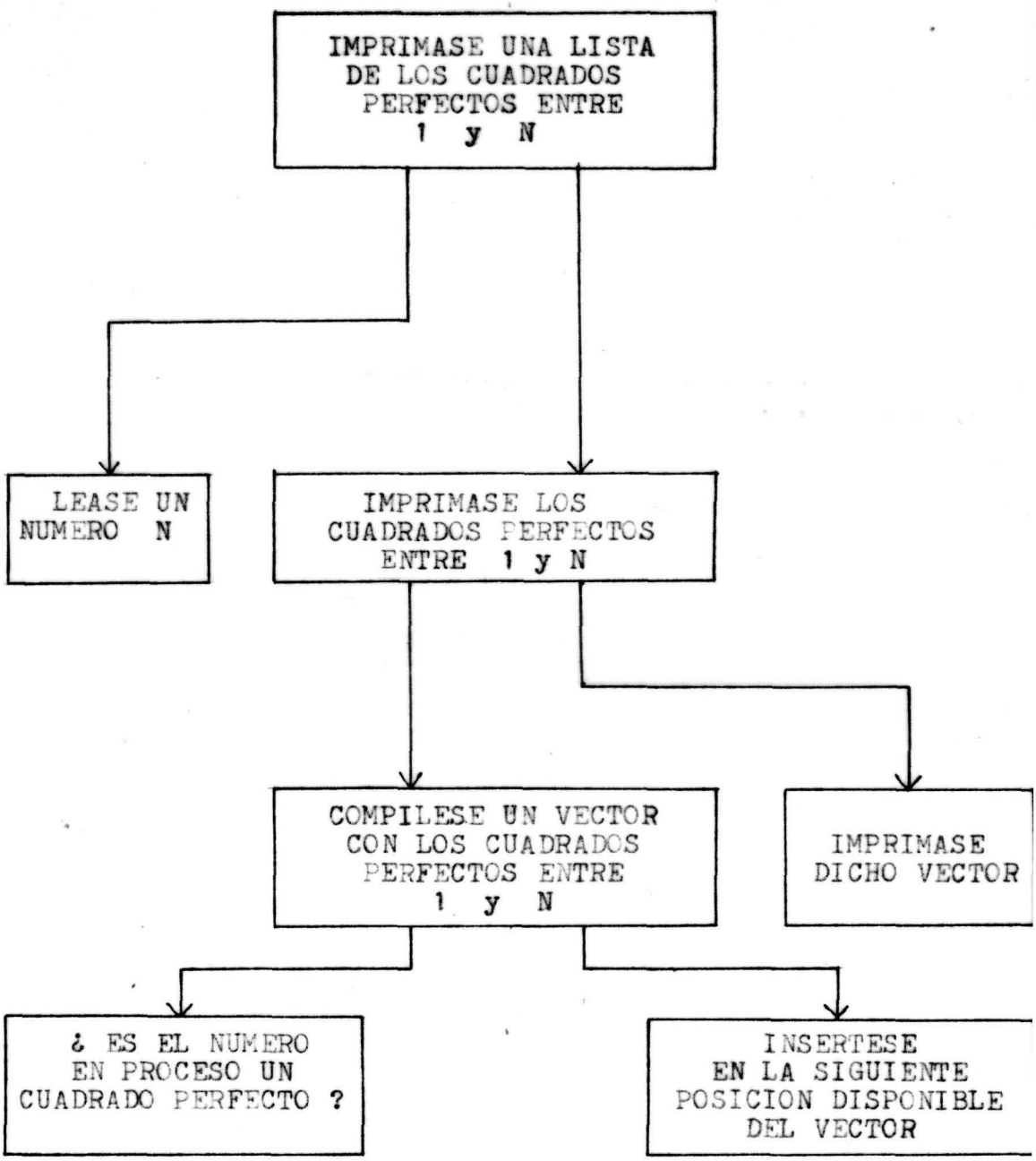
Este último refinamiento parece no agregar mucho a la solución,pero de hecho ha introducido un vector para representar la lista de cuadrados perfectos. Este es un detalle de la construcción que no forma parte de la solución abstracta. Antes de entrar en el diseño de un algoritmo real, se introducirán dos módulos adicionales de la solución que formarán parte de la compilación del vector de cuadrados perfectos. Fstos son:

¿Es el número en proceso un cuadrado perfecto?

Insértese en la siguiente posición disponible del vector

Esto completa la estructura para el diseño, el cual puede realizarse en forma de algoritmo. La forma resultante será mostrada al final y sirve de guía para la construcción final. Los cuatro rectángulos, a partir de los cuales no salen líneas inferiores (algunas veces denominados terminales),pueden considerarse como puntos de partida;son los que corresponden más exactamente con los pasos reales del algoritmo. Los otros rectángulos (algunas veces denominados no terminales) definen estructuras de decisión de más alto nivel que describen cómo se ha desarrollado el trabajo.

Nota: en la página siguiente se muestra la figura anteriormente citada.



La forma de figura vista en la página anterior, sirve de guía a la construcción del algoritmo en forma de comentario. El resultado es el algoritmo PERFECT_SQUARES. Este algoritmo lee un entero positivo N e imprime una lista de ellos entre 1 y N cuando son cuadrados perfectos. El vector SQUARES (con el índice VEC_PTR) se usa para almacenar los cuadrados mientras se espera la impresión de ellos. T es una variable entera.

```
1. [Entrada]
   Read(N)
2. [Se compila el vector de cuadrados perfectos]
   VEC_PTR←0
   Repetir hasta el paso 4 para I= 1,2,...,N
3. [Se clacula el valor truncado de la raiz cuadrada
    del número en proceso
   T←TRUNC(SQRT(I))
4. [¿ Es I un cuadrado perfecto ?]
   If T*T=I
   then VEC_PTR←VEC_PTR+1
        SQUARES(VEC_PTR)←I
5. [Se imprimen los números contenidos en el vector]
   Repetir para I=1,2,...,VEC_PTR
      Escribir (SQUARES(VEC_PTR))
6. [Terminar]
   Salida
```

Este ejemplo ilustra una aplicación de la técnica de diseño top-down. La solución del problema ha sido encontrada tras de una sistemática descomposición en subproblemas cada vez más simples. En cada nivel de abstracción se ha centrado la atención en qué es lo que se quiere hacer; en seguida se ha procedido a definir los módulos que realizan tal tarea, de lo cual resulta una colección de módulos que definen el siguiente nivel de abstracción. Este proceso se repite hasta que se tiene un conjunto de módulos que puede ser codificado con relativa facilidad.

Este método tiene algunas otras ventajas. Los módulos individuales son lo bastante pequeños (en términos de su función) como para ser comprendidos con facilidad. El peligro de complicaciones por efectos laterales se ha reducido, lo cual disminuye también la probabilidad de error. El diagrama final es una estructura que sugiere un patrón organizado de prueba(véase figura última). Como el propósito de cada módulo se especifica claramente, cada uno puede probarse en forma separada, con lo cual se tiene que la prueba resultará más simple que probar el programa completo. Las interfaces, o relaciones que existen entre los módulos individuales, también son definidas por la estructura del diagrama y pueden probarse una vez que cada módulo ha sido probado concienzudamente. Tal vez una ventaja adicional del método top-down es que sugiere una

estructura para los comentarios a través de la estructura del diagrama. Esto puede permitir mejorar la legibilidad del programa resultante.

1.7. ELEMENTOS DEL ESTILO DE PROGRAMACION

1.7.1 DISEÑO DE PROGRAMAS

En cierto sentido, aquí es donde se pueden obtener las mayores ventajas en la calidad de los programas. La calidad no es una característica "adicional", muchas características deseables de los programas (la facilidad para modificarlo es un buen ejemplo) son difíciles de agregar a los programas ya desarrollados; deben incorporarse a ellos durante la fase de diseño. La apariencia de un programa es difícil de modificar en cualquier momento; la manera en que trabaja es todavía más difícil.

Los programas que se construyen sin errores no necesitan una fase separada de corrección. Esto no sólo es un ahorro precioso de tiempo para el programador; el evitar errores es también la clave para preparar programas confiables.

El pasarse de listo ha sido la ruina de más de un programador. Quizá los programadores se enorgullecen de sus habilidades para resolver rompecabezas y consideran cada programa como otro problema más. Tal actitud de programación se revela por el uso excesivo de trucos de programación, trucos que con frecuencia sacan provecho de las su

tilezas del lenguaje o de la construcción. Estos trucos pueden servir para reducir el producto espacio-tiempo de un programa, pero a costa de una considerable pérdida de claridad, pérdida esta última que normalmente no puede ser sacrificada. Como regla general, nunca debe sacrificarse la claridad por tratar de mostrar la destreza personal.

Uno de los beneficios más atractivos del enfoque top-down para diseñar programas es la oprtunidad que brinda de separar las funciones. Se trata de una decisión de diseño que puede tener impacto significativo en la facilidad con que los programas resultantes pueden modificarse cuando se presenta la oportunidad. La separación funcional se basa en la premisa de que la esfera de influencia de cualquier decisión particular es razonablemente pequeña. Entonces, su impacto en el programa mismo será localizado muy pronto. La metodología de diseño top-down trata de reforzar esta disciplina al controlar los tipos de interacciones permitidas entre los módulos. En esta sección se ofrece un ejemplo que aclara el uso y los defectos de la separación de las funciones.

Este ejemplo de la idea de separación de las funciones comprende el síndrome del"número mágico". Los números mágicos son constantes numéricas misteriosas que aparecen entre los cálculos del programa, casi siempre con poca o ninguna explicación. Con frecuencia se utilizan como pará

metros de un lazo, para definir el tamaño de un vector o formación, o simplemente como constantes de alguna fórmula. Aunque sus efectos pueden ser inocuos, el uso excesivo de números mágicos puede comprometer seriamente la capacidad de modificación de un programa.

Supóngase,por ejemplo, que se va a diseñar un modesto sistema de recuperación de información para mantener información (o una base de datos) de 37 estudiantes de un curso en particular sobre ciencia de la computación. Para cada estudiante se guarda la siguiente información: nombre,número de problemas de laboratorio terminados, calificaciones de laboratorio al día. Se elige mantener esta información en tres vectores paralelos, cada uno de los cuales tiene 37 elementos; un vector tipo hilera de caracteres NAMES , un vector entero LABS_DONE y un vector real LAB_MARK. Se han escrito rutinas para insertar nuevas calificaciones de laboratorio, para corregir las calificaciones erróneas que se han registrado, o sólo para desplegar determinados segmentos de la información. Cada una de estas operaciones requiere de una o más exploraciones de estos vectores, mediante un lazo de un tipo parecido al siguiente:

```
Repetir para I= 1,2,...,37
```

En este programa en particular, el número 37 es, sin duda, un número mágico. De hecho, la ejcución de este programa se halla estrechamente ligada con este número;

puede aparecer en una docena de lugares diferentes dentro del código, incluyendo declaraciones, lazos de cálculo y lazos de impresión. El uso de este número enlaza las diferentes rutinas en forma muy estrecha, de forma no totalmente evidente.

Supóngase que se desea utilizar este programa para un curso distinto, como uno compuesto por 212 estudiantes. En primer lugar, deberá recorrerse dicho programa por completo, cambiando todas las veces que aparece el número 37 por el número 212. Deberá localizarse todas las veces que aparezca, y cualquiera de las apariciones no detectada producirá con seguridad (aunque tal vez no en forma inmediata) un error. Además, las apariciones de 36 o 38 pueden necesitar ser modificadas a 211 o 213 y así sucesivamente. El uso de números diferentes de estudiantes ha hecho que este programa sea innecesariamente difícil de modificar.

Este caso en particular es fácil de corregir. En vez de usar un número específico para el número de estudiantes de un curso, se introduce una variable, llamada 'TAMAÑO_CLASE', que se usará donde se necesite especificar el número de estudiantes, De hecho, el nombre elegido constituye una medida de la documentación interna del programa. Cada vez que se use este programa para un curso diferente, todo lo que tiene que hacerse es dar el valor adecuado a la variable TAMAÑO_CLASE, ya sea por asigna-

ción (en el tiempo de compilación) o mediante la entrada (en el tiempo de ejecución). Este es un ejemplo de la separación de funciones. Se han diseñado las rutinas de este nuevo programa para que sean independientes del número real de estudiantes del curso.

La separación de funciones se logra con frecuencia mediamte el uso de subprogramas, es decir, módulos de trabajo que se construyen como subprogramas y que son llamados por los módulos de orden superior cuando se necesita que hagan algo determinado. La separación de funciones se mejora si todos los valores de los datos se pasan mediante la lista de parámetros, en lugar de las variables globales; además, el poder de los efectos laterales se disminuye. En el ejemplo del sistema de recuperación de información, se supone que cada una de las operaciones requeridas por la base de datos se encuentra codificada como un programa separado y, además, que toda la información intercambiada entre los subprogramas y el punto de llamada se produce por medio de la lista de parámetros.

1.7.2 CONSTRUCCION DE PROGRAMAS

A veces se juzga que la construcción real de los programas es la fase más importante e interesante de la tarea. Aunque la solución del problema y el diseño del programa pueden ser en realidad conocidos, la construcción del programa nunca parece ser tan directa como se cree. La última década ha visto aparecer un nuevo método de programación, conocido como 'programación estructurada', que algunas personas suponen intuitivamente que elimina todos los problemas de la construcción. Por desgracia, no es así. Existe muchísima sobrevaluación y confusión sobre la programación estructurada, debido básicamente a que una comunidad inexperta en procesamiento de datos está ansiosa de encontrar respuestas simples a los problemas complejos.

La programación estructurada no es otra cosa que un método de construcción de los programas en el cual el rigor y la estructura reemplazan la programación intuitiva y de gusto personal. Los buenos programadores han empleado esta forma desde antes que a este método se le diera nombre.

La 'estructura' de un programa está determinada en cierta forma por las construcciones que se han usado para dirigir el flujo del control. Es importante recodar que mientras se está leyendo la lista del programa desde arriba hasta abajo, la ejecución del mismo se lleva a cabo de una manera muy distinta. Uno de los principales logros

de la programación estructurada es que consigue que el flujo de control se distribuya en forma tal que la secuencia de ejecución sea similar a la secuencia de lectura. Esto impone una disciplina al programador en términos de las estructuras que puede usar y, además, en la forma en que éstas pueden ser usadas.

Cualquier programa escrito que utiliza las estructuras IF-THEN-ELSE y REPEAT son por definición programas estructurados. Por desgracia, los malos programas pueden escribirse usando cualquier técnica.

Hay algunos programas que según la estricta definición de programación estructurada se pueden considerar como tales. Sin embargo, su legibilidad puede mejorarse sacrificando un poco la eficiencia, al 'desenrrollar' algunos de los nidos. La mente humana tiene dificultad para comprender las estructrras anidadas complejas; esto requiere la retención de varios estados diferentes del programa en forma simultánea. Las estructuras profundamente anidadas son fuentes de errores y normalmente deben ser evitadas.

Hay varias formas de evitar las estructuras muy anidadas. Un método, tal vez apropiado para este caso en particular, es usar condiciones compuestas en un enunciado 'if' para definir las alternativas especificas en forma más precisa. Es necesario asegurarse, sin embargo, de que las condiciones mismas sean fáciles de comprender,

porque de lo contrario el cambio será de muy poco valor.

A continuación veremos el algoritmo de un programa que lee tres números, A,B, y C, e imprime el más grande y el más pequeño de los tres. Se supone que los valores son diferentes.

```
1. [Entran los valores de los datos]
   Read(A,B,C)
2. [Se determina el valor mayor]
   MAX←A
   if B>MAX then MAX←B
   if C>MAX then MAX←C
3. [Se determina el valor menor]
   MIN←A
   if B<MIN then MIN←B
   if C<MIN then MIN←C
4. [Se imprimen los resultados]
   Write('EL MAYOR ES',MAX,',EL MENOR ES',MIN)
5. [Termino]
   Exit
```

El hecho de que un programa parezca estar estructurado no debe ser motivo para no hacerle modificaciones tendientes a mejorarlo.

1.7.3 PRESENTACION DE PROGRAMAS

El formato y la apariencia de la lista del programa no son un incidente en la calidad del programa. Aquí es donde más puede hacerse por mejorar la legibilidad de un programa mejor, que en casi cualquier otro punto. A continuación vamos a considerar dos facetas de esta cuestión: los comentarios y los párrafos.

Los comentarios constituyen el principal componente de la documentación interna de los programas. Estos sirven de ayuda al lector para comprender las intenciones o los propósitos de las porciones del código, y también pueden servir para explicar las secciones lógicas o difíciles. A los programadores novatos rara vez se les dan instrucciones de que escriban comentarios, aunque la escritura de buenos comentarios es quizá tan importante, y tal vez tan difícil de aprender, como la escritura de buenos programas. Los buenos comentarios no pueden hacer mucho para mejorar una mala modificación, pero los malos comentarios pueden deteriorar muchísimo un buen código.

Muchas personas caen en una de dos categorias extremas: los que escriben pocos comentarios o ninguno y los que sobrevaloran los comentarios. Cada uno de estos extremos disminuye la legibilidad del programa en forma diferente; la escasez o ausencia fallan porque no proporcionan la adecuada información de apoyo, y la abundancia de comentarios aumenta la confusión. Los comentarios no de

berían salpicar el código, sino explicarlo y apoyarlo. Esto requiere, por supuesto, que el programador enfoque su programa en forma simultánea desde dos puntos de vista: el programador y el del que lo documenta, y que tome en cuenta los objetivos de ambos.

La mayor parte de lo que puede o no hacerse con los comentarios es función dellenguaje de programación utilizado. Por desgracia, algunos lenguajes para los cuales los comentarios son más necesarios disponen de pocas formas en las que pueden expresarse buenos comentarios. No obstante, la mayor parte de ellos permiten al programador utilizar una línea completa como comentario. Esta característica hace posible el uso de la longitud precisa en un comentario de varios renglones para explicar el propósito y la interacción con un determinado componente del programa o módulo. Por ejemplo, cada subprograma debería comenzar con comentarios de este tipo para esplicar su propósito, la manera en que se llama (incluyendo el significado de los parámetros) y cualquier situación especial que pueda presentarse.

Además de la facilidad para escribir comentarios en renglones separados, algunos lenguajes (como PL/I) permiten que los comentarios se sitúen en el mismo renglón en que está el código del programa. Esto puede ser muy útil para escribir comentarios breves a fin de explicar un reglón simple o una sola operación, sin romper el formato visual completo del programa mismo. Tales comentarios

deberán estar tan separados del texto como sea posible, quizá desplazándonos hacia el lado derecho de la lista de instrucciones.

Como un punto final sobre los comentarios, debe asegurarse siempre que los comentarios y el código estén de acuerdo. Cuando se realiza una modificación en el código debe tenerse cuidado de que se realice un cambio similar en cualquier comentario que se le relacione. Esto se pasa por alto con mucha frecuencia.

El valor de los párrafos o 'sangrías' controladas de secciones del listado de un programa es tambien una forma de mejorar la legibilidad. En cualquier texto escrito tienen dos propósitos principales: identificar las unidades de la estructura del texto y mitigar el tedio del proceso de lectura. Ambos objetivos se aplican perfectamente a los programas.

ALGORITMOS DE FASES DE PROGRAMACION

ALGORITMOS DE LAS FASES DE LOS PROCESOS DE PROGRAMACION

Los presentes algoritmos están divididos en fase de planificación, fase de desarrollo, y fase de mantenimiento.

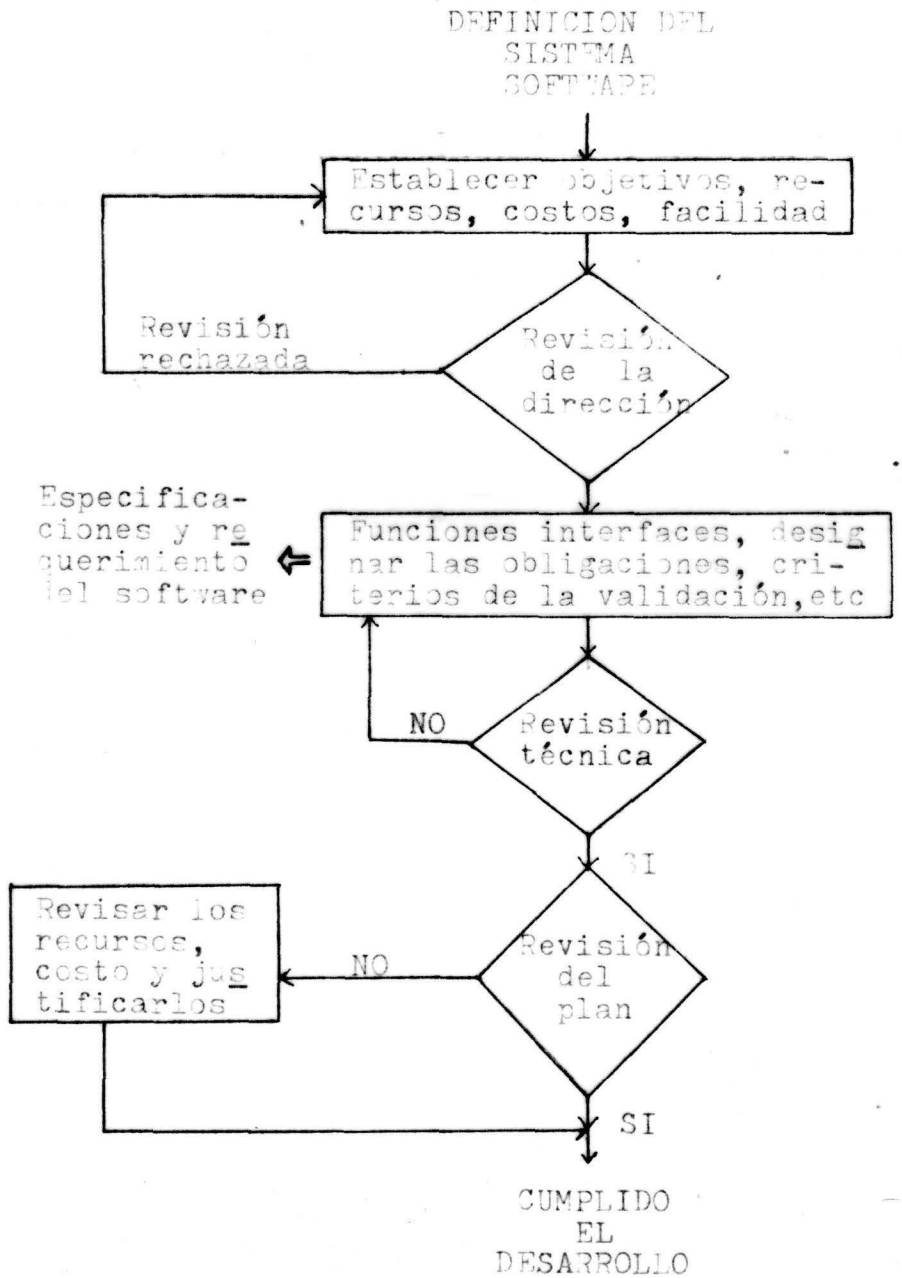
- Fase de planificación. En esta fase lo primero que se hace es trazar un plan software con las especificaciones que nos ha dado el empresario, de tal manera que quede todo bien detallado. Una vez que este plan sea revisado por el empresario pasaremos a lo que se denomina 'especificaciones y requerimientos del software' que consiste en ver todas las posibilidadws que presenta la computadora como son: definición de funciones, interfaces, criterios de validación, etc. Después de aquí se pasa a una revisión técnica y a una revisión del plan de tal manera que si procede estamos preparados para empezar el plan de desarrollo.

- Fase de desarrollo. Partiendo de los requerimientos hechos en el plan de planificación lo primero que hacemos en esta fase de dasarrollo es: definir las estructuras de datos, establecer módulos, identificar las obligaciones,etc. Todo esto da lugar a un desarrollo preliminar. En el próximo paso se consideran los aspectos de cada módulo,desarrollando una descripción detallada de cada módulo, que una vez que es revisada, se añade al documento preliminar. Finalmente, después de los dos

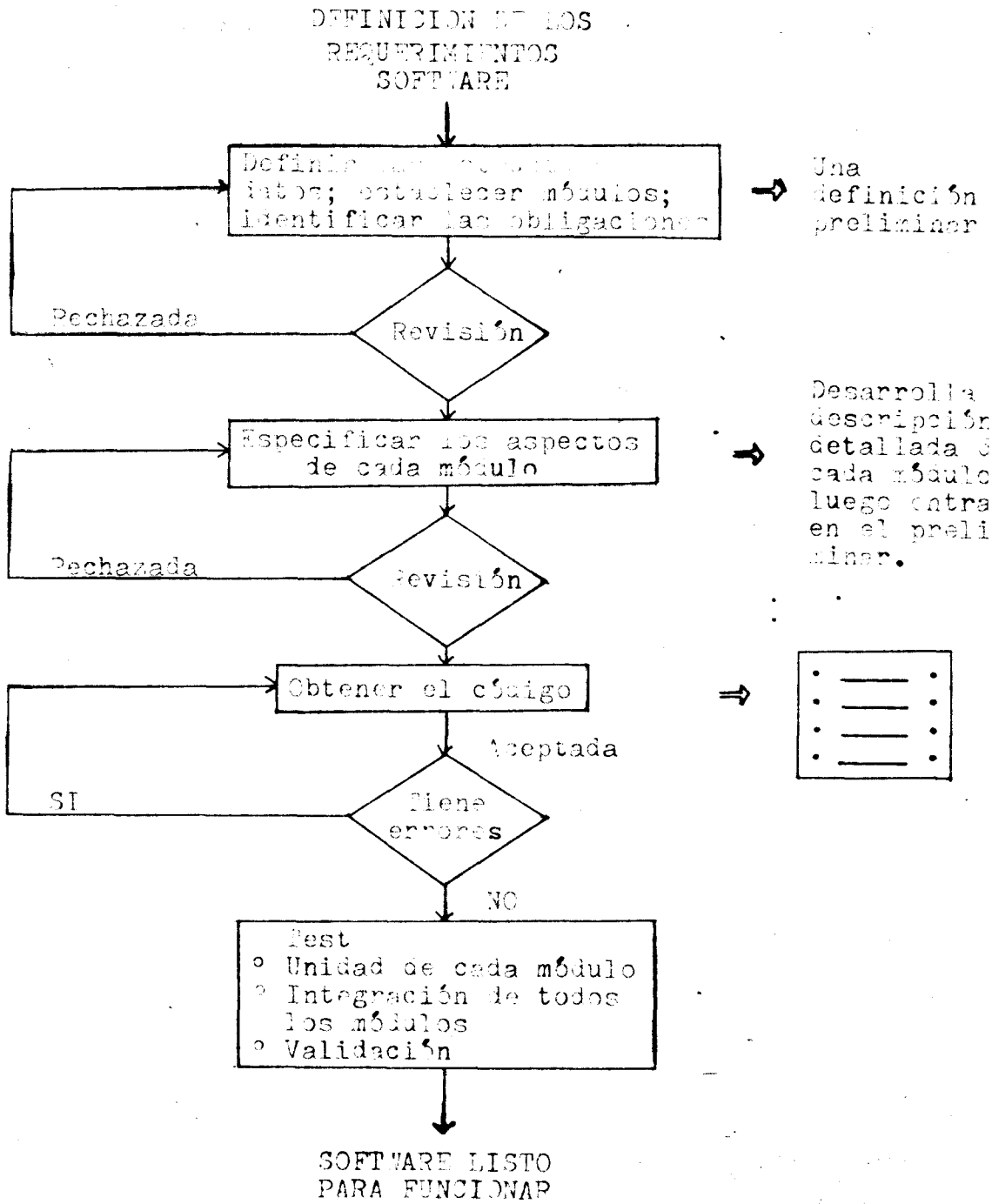
desarrollos, viene el 'código', que es la generación de un programa con el uso apropiado de un lenguaje de programación. Los últimos tres pasos del desarrollo están asociados con el testeo del software. La unidad de testeo mira como funciona cada función y cada componente individual del software. El test de integración proporciona un significado para ensamblar todas las estructuras modulares mientras hay un testeo de funciones y interfaces. El test de validación verifica que todos los requirimientos del software han sido encontrados.

- Fase de mantenimiento. Esta fase está enfocada hacia la revisión del software para cuando tengamos que hacer mejoras. Según el algoritmo en esta fase podemos corregir, adaptar, y perfeccionar.

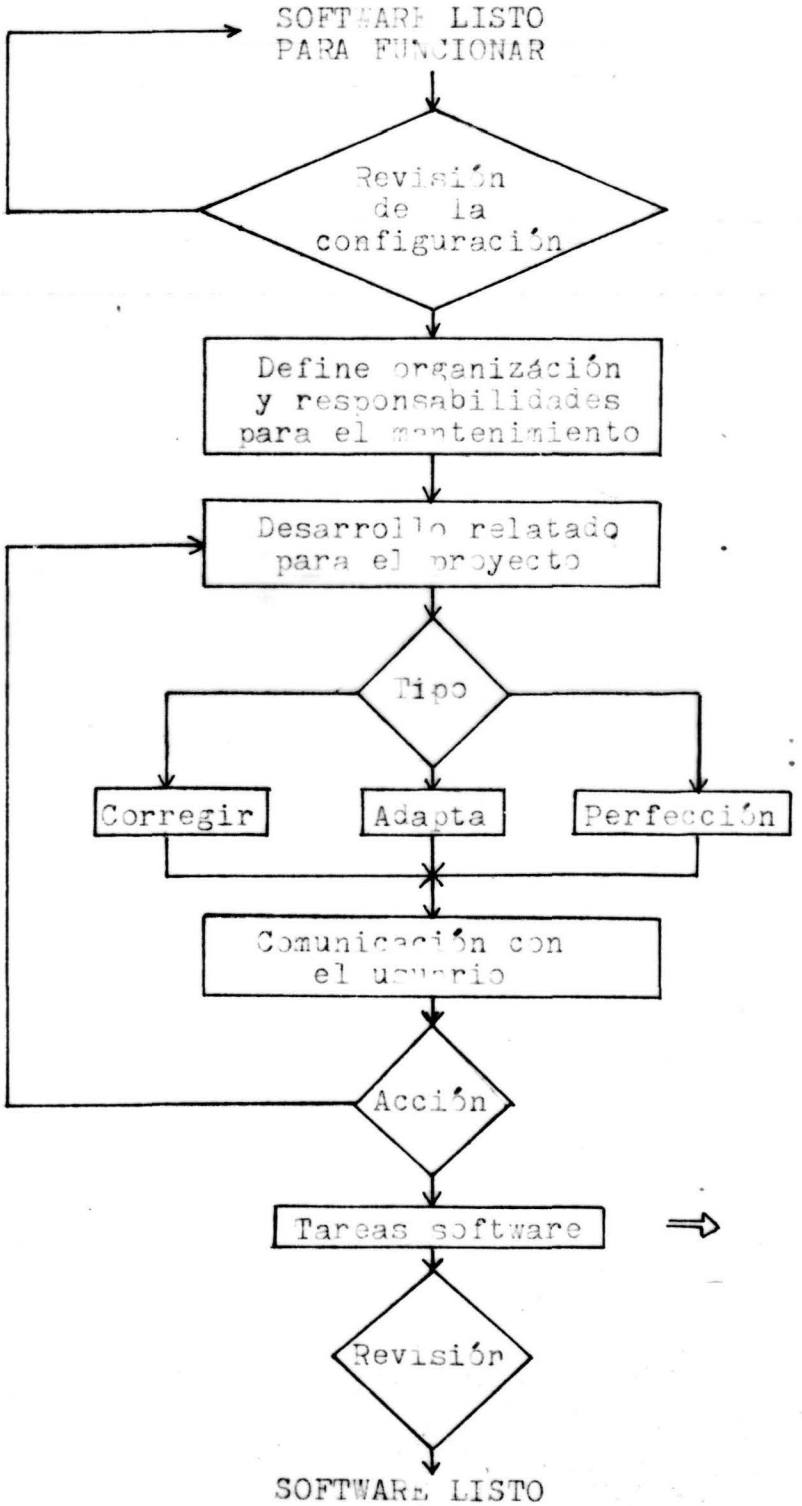
- FASE DE PLANIFICACION



- FASE DE DESARROLLO



- FASE DE MANTENIMIENTO



MODELO DE FLUJO DE DATOS

MODELO DE FLUJO DE DATOS

1 DESARROLLO Y FLUJO DE INFORMACION

El flujo de información es una situación que hay que tener en cuenta durante la fase de análisis y requerimientos de la ingeniería del software. Al principio, con un sistema de modelo fundamental, la información puede ser representada como un continuo 'fluir' el cual sufre una serie de transformaciones como pueden ser los desarrollos de entradas y sadidas. El diagrama de flujo de datos (DFD) se usa como una herramienta gráfica para dibujar el flujo de información. El modelo de flujo de datos define un número de diferentes mapeados que transforma el flujo de información en una estructura software.

1.1 CONTRIBUIDORES

El modelo de flujo de datos (y el desarrollo del software en general) tiene sus origenes en el desarrollo de los conceptos que da importancia a lo modular, al desarrollo 'top-dow', y a las estructuras de programación. Sin embargo, el modelo de flujo de datos acerca estas técnicas por la integración explícita del flujo de información en el proceso de desarrollo.

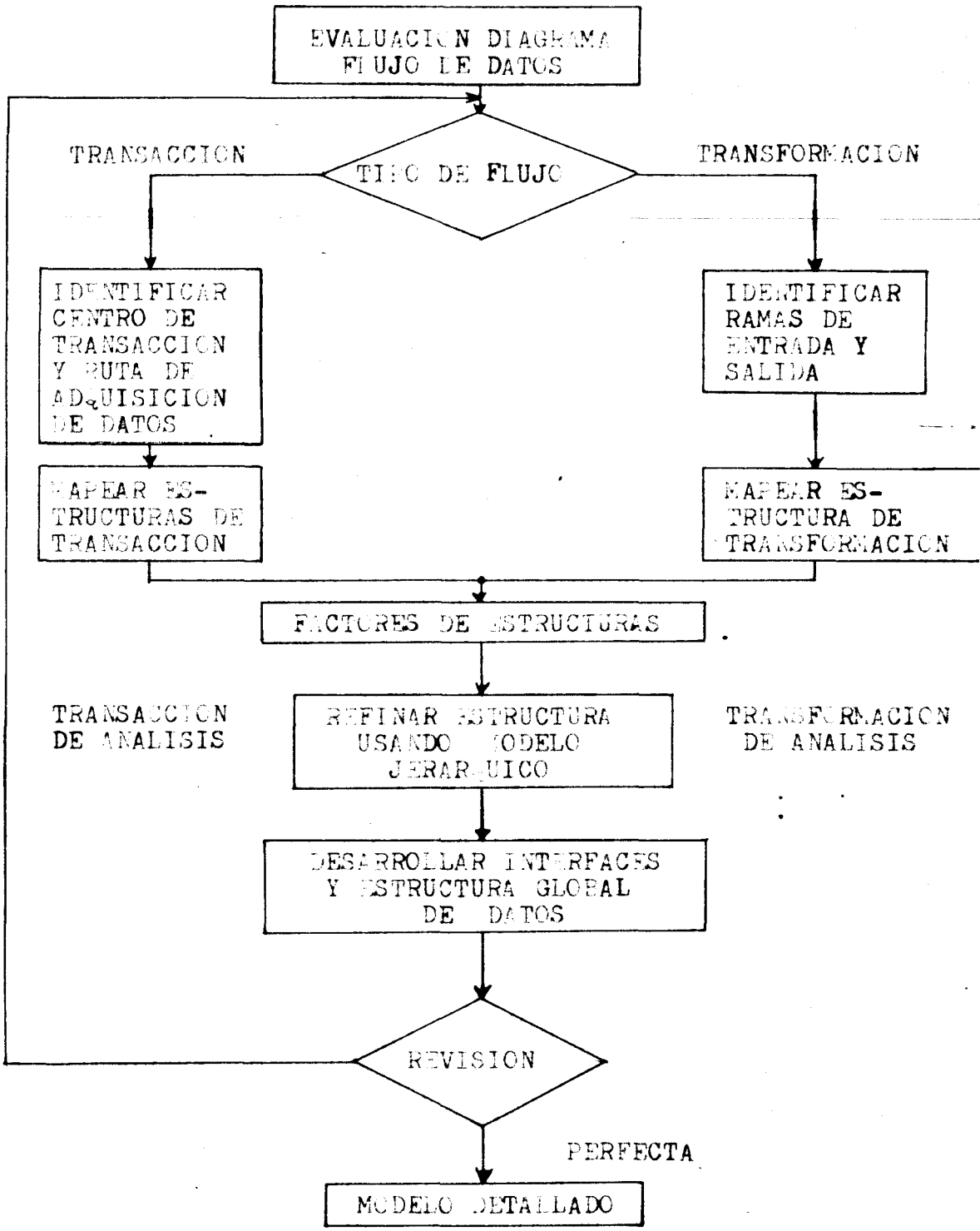
1.2 AREAS DE APLICACION

Cada metodología de desarrollo software tiene sus ventajas y sus inconvenientes. Un importante factor de juicio para un desarrollo metodológico es la extensión de aplicaciones a las cuales puede ser aplicada. El modelo de flujo de datos es responsable de una aplicación en áreas muy diversas.

El modelo de flujo de datos es particularmente poderoso cuando no existe una estructura de datos. Por ejemplo, aplicaciones de control de un microprocesador, complejos análisis de procedimientos, procesos de control, y muchas aplicaciones en ingeniería y en el campo científico en donde no se requieren sofisticadas estructuras de datos, Todo esto hace que sea difícil emplear el modelo de estructura de datos; sin embargo, un modelo de flujo de datos resolvería las anteriores dificultades de una manera sutil.

El modelo de flujo de datos también puede ser aplicado en procesos convencionales de datos y otras aplicaciones donde existen distintas estructuras de datos.

Una extensión del modelo de flujo de datos, llamada MASCOT, adapta el acercamiento a el tiempo-real, tiene aplicaciones para la interrupción y para la conducción. Usando una representación de flujo de información en procesos concurrentes (que se unen), MASCOT provee áreas de intercomunicación de datos que permiten la definición



de otros procesos de comunicación. MASCOT permite el desarrollo de interfaces específico y coordina la comunicación con una sincronización original.

2 CONSIDERACIONES DEL DESARROLLO EN PROCESO

El modelo de flujo de datos permite la translación de información contenida en los 'Requerimientos y especificaciones del software' a un desarrollo preliminar de estructura del software. La translación del flujo de información para la estructura, está acompañada en parte de los siguientes procesos:

1) Establecimiento de la categoría del flujo de información.

2) Indicación de los límites del flujo.

3) Mapeado del diagrama de flujo de datos (DFD) dentro de la arquitectura software.

4) Establecimiento de controles de jerarquía.

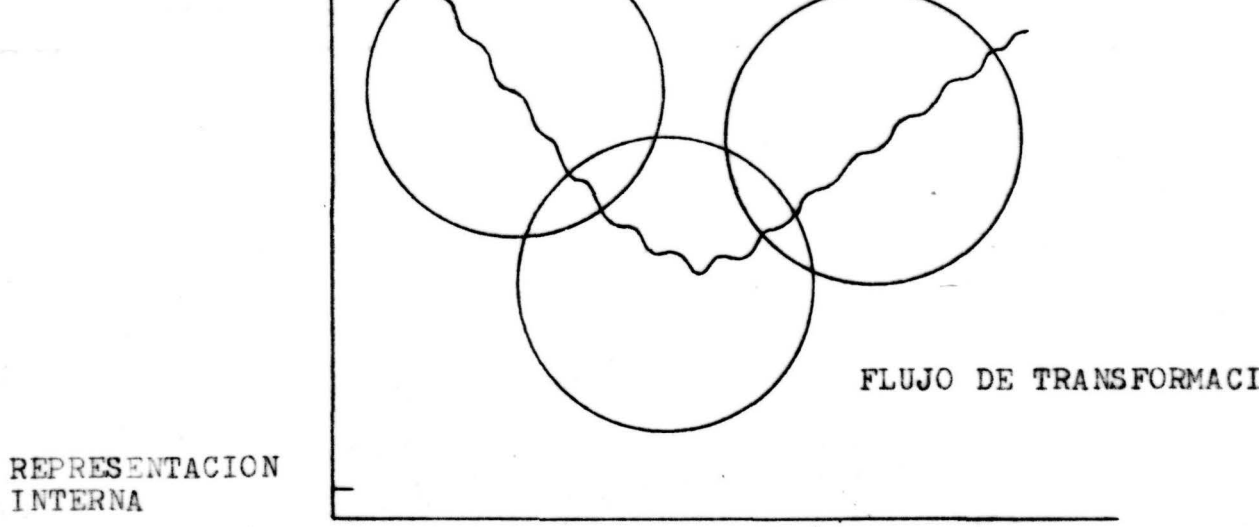
5) Refinación de la estructura resultante por la capacidad del modelo y por la jerarquía.

2.1 TRANSFORMACION DE FLUJO

En el llamado sistema de modelo fundamental, la información debe entrar y salir del software en una forma que se pueda entender por el mundo externo. Por ejemplo, los tipos de datos sobre un teclado de un terminal, los tonos en las líneas telefónicas, y las figu-

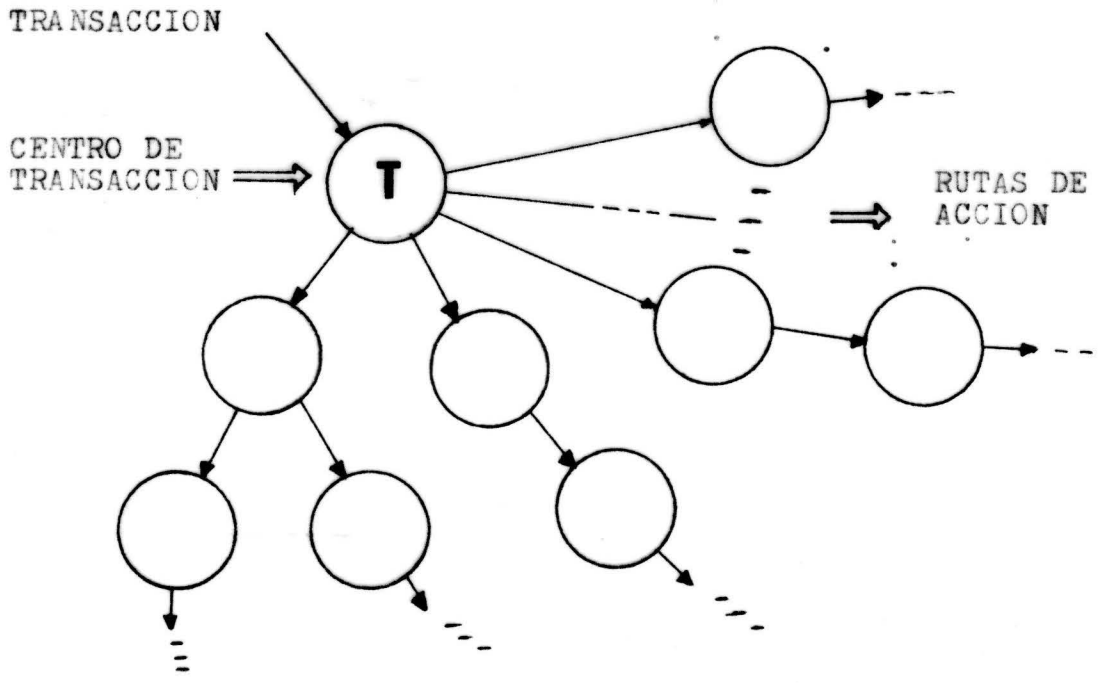
ras sobre una pantalla gráfica de una computadora son forma de información para el mundoexterno. Tales datos externos deben ser convertidos en una forma interna de procesamiento; Para esto la información entra en un sistema a través de las partes que transforman la forma de los datos externos en forma de datos internos. El flujo de ingreso a través de tales partes es llamado 'afferent' (entrada). Una vez que han entrado los datos, estos son pasados a través del 'centro de transformanión' y empieza a moverse aquí a lo largo de distintas partes que guian a los datos fuera del software. Este flujo que guía a los datos fuera del software es llamado 'efferent' (salida).

Cuando un segmento de un modelo de flujo de datos presenta estas características, decimos que una transformación de flujo está presente.



2.2 TRAMITACION DE FLUJO

El sistema de modelo fundamental implica una transformación de flujo; por lo tanto, es posible caracterizar todó el flujo de información en la anterior categoría. Sin embargo, el flujo de información frecuentemente está caracterizada por datos sencillos, llamados 'transación', los cuales lanzanun flujo de datos a lo largo de muchas partes. Cuando un diagrama de flujo de datos (DFD) toma la forma que muestra la figura, decimos que una tramitación de flujo está presente.



La transación de flujo está carecterizada por el movimiento de datos a través de las partes de recepción que convierten la información del mundo externo en una transación o tramitación. La transación es evaluada, en ba-

se al flujo de valores a través de una de las muchas rutas de acción que se inicializan. El centro del flujo de información del cual parten muchas rutas de acción es llamado 'centro de transacion'.

Por último podemos decir que, en un diagrama de flujo de datos (DFD) para un sistema largo, pueden estar presentes una transformación y una tramitación.

2.3 UN PROCESO ABSTRACTO

El desarrollo de un modelo de flujo de datos está ilustrado en la figura de a continuación. El proceso empieza con una evaluación del diagrama de flujo de datos. Luego el flujo de información (transformación o transación) es establecido, y el flujo límite que trazan los centros de transformación y transación son definidos En base a la localización del límite, las transformaciones son mapeadas en una estructura software por módulo. La presencia de mapeado y la definición de módulo es acompañada por la factorización de las estructuras aplicando las técnicas de desarrollo y la jerarquización.

MODELO DE ESTRUCTURA DE DATOS

MODELO DE ESTRUCTURA DE DATOS

1 ESTRUCTURA DE DATOS

La estructura de datos afecta tanto al proceso como al desarrollo del software. Los datos repetitivos son siempre procesados con un software que tiene la facilidad para controlar por repetición; los datos alternativos (por ejemplo, información que puede o no estar presente) hacen que el software esté condicionado con elementos de procesos; De hecho, la estructura de información es un excelente pronosticador de la estructura software.

El módelo de estructura de datos transforma una representación de estructura de datos en una representación software.

1.1 CONTRIBUIDORES

El origen del modelo de estructura de datos puede ser encontrado en la discusión de las estructuras de datos fundamentales, algoritmos, estructuras de control y datos, y en la abstracción de conceptos de datos. Más información en cuanto al tratamiento del desarrollo del software y sus relaciones con las estructura de datos pueden ser encontrada en los llamados métodos de Jackson, y Warnier.

El método de Jackson es uno de los más extendidos para el desarrollo del software. Este consiste en tomar una vista paralela de las estructuras de datos de entrada y

salida para asegurar un desarrollo con calidad. Jackson enfatiza en el desarrollo de técnicas para transformar los datos en programas estructurados.

La construcción lógica de programas (LCP), es un desarrollo metodológico dasarrollado por Warnier, y proporciona un riguroso métdo para la construcción de un software. Para la producción de conceptos fundamentales en la ciencia de la computación, Warnier desarrolló un grupo de técnicas que acompañado con un mapeado de entrada salida de las estructuras de datos producen una detallada representación del software.

Una técnica llamada construcción lógica del software (LCS) es representativa de una síntesis del módelo de flujo de datos y del módelo de estructrra de datos. El desarrollo de este método consiste en que un desarrollo lógico puede ser descrito explícitamente si el software es visto como un sistema de datos y como un sistema transformador de datos, Aunque el LCS no es una estructura de datos propiamente dicha, puede ser vista como una técnica de desarrollo.

1.2 AREAS DE APLICACION

- Aplicaciones en negocios. Las entradas y salidas tienen distintas estructuras (por ejemplo, ficheros de entrada e información de salida); el uso de una base de datos jerarquizada es común.

- Aplicaciones en sistemas. Las estructuras de datos para operar con sistema están compuestas de muchas tablas, ficheros, y listas que tienen una estructura bien definida.

- Aplicaciones en CAD/CAM. El desarrollo de ayuda para computadoras (CAD) y la ayuda para la manufacturación de computadoras (CAM) requieren sofisticadas estructuras de datos para almacenamiento de información, translación, etc.

Como se habrá podido notar al modelo de flujo de datos siempre se puede acceder para usarlo. Sin embargo, el modelo de estructura de datos también puede producir buenos resultados cuando las características propias de la información están presentes.

1.3 ESTRUCTURA DE DATOS CONTRA EL FLUJO DE DATOS

Antes de considerar las diferencias entre las estructuras de datos y el flujo de datos, es importante notar que ambas estructuras son conductoras de información, que intentan transformar información en una representación software.

El desarrollo de la estructura de datos no hace uso explícito del diagrama de flujo de datos. Por lo tanto la clasificación del flujo de transformación y de transación tiene poca relevancia en el módelo de estructura de datos. Más importante que lo anterior es crear un proce-

so de descripción del software. Los conceptos de estructura de software no están explícitamente considerados. Los módulos son considerados como un proceso, y una filosofía de módulos independientes es lo que predomina.

El módelo de estructura de datos hace uso de un diagrama jerarquizado para representar la estructrua de la información. Por lo tanto, el énfasis durante el proceso de requerimientos y análisis del software deben de estar situados sobre estos modos de representación.

2 DESARROLLO DE LOS DATOS

El modelo de estructura de datos implica que la organización de datos es una característica dominante en el desarrollo del software. El beneficio potencial del desarrollo de los datos fueron reconocidos al principio de la evolución del desarrollo software, pero la integración formal con el método de desarrollo todavía no ha ocurrido. Los conceptos de información oculta, y abstracción de datos, dan lugar a un acercamiento del desarrollo de los datos.

La mayoría de los desarrollos software están concentrados sobre estructura (arquitectura) o sobre consideraciones de proceso. Con esto corremos el riesgo de que durante el desarrollo de los datos puede romperse el desarrollo software.

A continuación veremos un grupo de principios para la

especificación de los datos.

a) El método de análisis sistemático aplicado al software debería ser aplicado a los datos. Se gasta mucho tiempo y esfuerzo, discutiendo, revisando y especificando los requirimientos del software y el desarrollo preliminar. La representación del modelo de flujo de datos y del modelo de estructura de datos debería también ser desarrollado y revisado, la organización alternativa debería ser considerada, y el impacto del desarrollo de los datos sobre el desarrollo del software debería ser evaluado.

b) Todas las estructuras de datos y las operaciones que van a ser ejecutadas en esas estructuras deben de estar identificadas. El desarrollo de una eficiente estructura de datos debe de tomar los operandos para ser ejecutados sobre una estructura de datos.

c) Un diccionario de datos debería de ser establecido y usado para definir tanto los datos como el desarrollo del software. Un diccionario de datos representa explícitamente las relaciones entre los datos y las restricciones sobre los elementos de una estructrura de datos. Los algoritmos pueden aprovecharse de las relaciones hechas para poderlos definir más fácilmente.

d) El nivel de decisión de datos debería ser situado en lo último en el proceso de desarrollo. La organización de inclusión de datos debería ser definida durante el proceso de requerimientos y análisis, refinado duran-

te el desarrollo por pasos. El método 'top-dow' accede al desarrollo de datos proporcionando beneficios que son análogos a cuando el método 'top-dow' accedía al desarrollo software (atributos de estructuras mayores son desarrolladas y evaluadas primero para que la arquitectura de los datos pueda ser establecida).

e) La representación de una estuctura de datos debería conocer sólo a aquellos módulos que deben hacer uso directo de los datos contenido en esa estructura de datos. El concepto de información escondida y las conceptos relatado de acoplamiento, proporcionan una importante introducción en la calidad de desarrollo del software. Este punto hace alusión a la importancia de estos conceptos así como a la importancia de separar la vista lógica de la vista física.

f) Una librería de estructura de datos útiles y operaciones que pueden ser aplicadas debería ser desarrollada. Las estructuras de datos y las operaciones deberían ser vistas como un recurso para el desarrollo del software. Igual que las subrutinas empaquetadas pueden reducir el tiempo de desarrollo para realizar un software útil, una librería de estructura de datos puede reducir tanto las especificaciones así como el esfuerzo de desarrollo por los datos.

g) Un desarrollo software y un lenguaje de programación debería soportar las especificaciones y la realiza

ción de tipos de datos abstractos. La implementación (y su correspondiente desarrollo) de una estructura sofisticada de datos puede ser extremademente difícil si no hay significado para las especificaciones directas de las estructuras existentes.

Todos los principios descritos anteriormente son formas básicas de desarrollo de datos que pueden ser integradas tanto en el desarrollo de módelo de flujo de datos como en el desarrollo de módelo de estructura de datos.

TESTEO Y DEPURADO DEL SOFTWARE

TESTEO Y DEPURADO DEL SOFTWARE

1 CARACTERISTICAS DEL TESTEO

El testeo presenta una interesante anomalía para la ingeniería del software. Durante los primeros pasos, en la planificación y desarrollo, la ingeniería intenta construir un software a partir de conceptos abstractos para una implementación tangible. Después de esto viene el testeo. La ingeniería crea una serie de tests los cuales están ideados para destruir el software que ha sido construido. De hecho, el testeo es uno de los pasos en el proceso de la ingeniería del software que debería ser visto (al menos, psicologicamente) más bien como destructivo antes que constructivo. Llegados a este punto nos podemos hacer la siguiente pregunta: ¿destruye realmente el testeo?. La respuesta a esta pregunta es que 'NO'. Sin embargo los objetivos del testeo son algo diferente de lo que nosotros esperamos.

1.1 OBJETIVO DEL TESTEO

Los objetivos del testeo los podemos resumir en tres puntos.

a) El testeo es un proceso de ejecución del programa con el intento de encontrar un error.

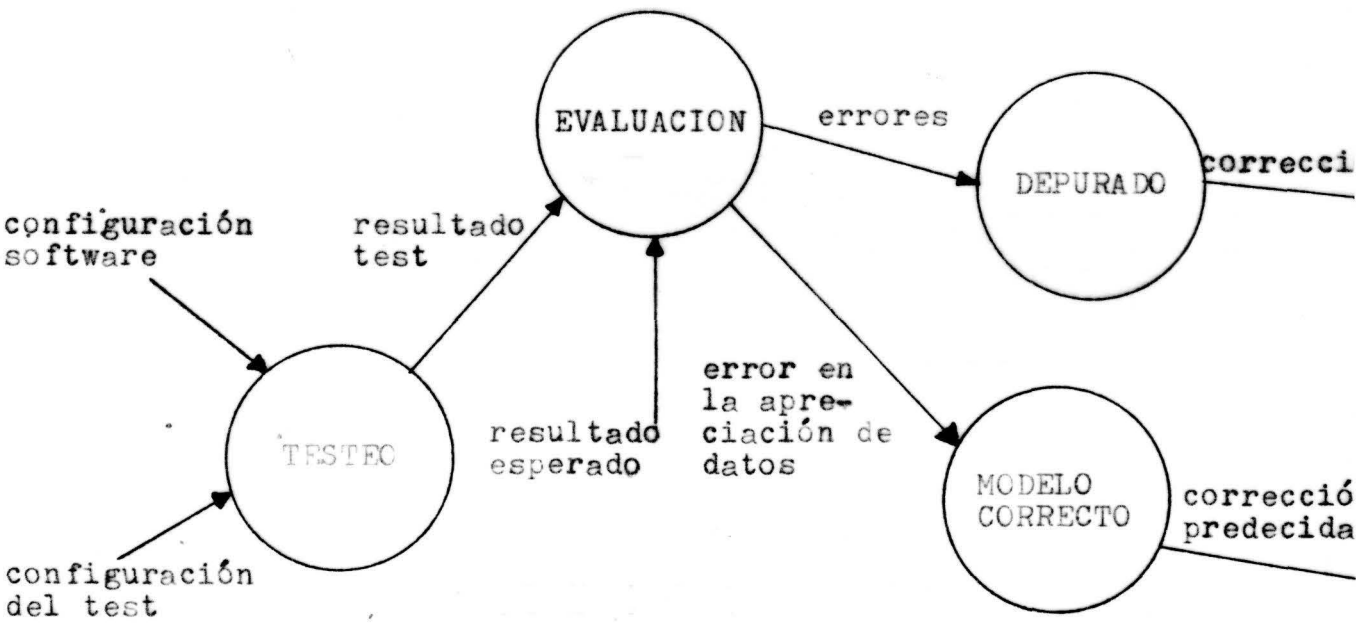
b) Un buen test es uno que tiene la probalidad de encontrar al menos un error no detectado.

c) Un test exitoso es resolver un error no descubierto.

El principal objetivo del testeo es desarrollar un test que sistemáticamente encuentre diferentes clases de errores.

1.2 TEST DE FLUJO DE INFORMACION

El flujo de información para el testeo sigue la estructura de la figura



Dos clases de entradas se requieren para el proceso de testeo; una configuración software que incluyen los requerimientos del software, la especificación de un desarrollo, y un código fuente, y por otro lado un plan y un proceso para el testeo junto con los resultados esperados. En la actualidad, la configuración del test es un subgurpo de la configuración software.

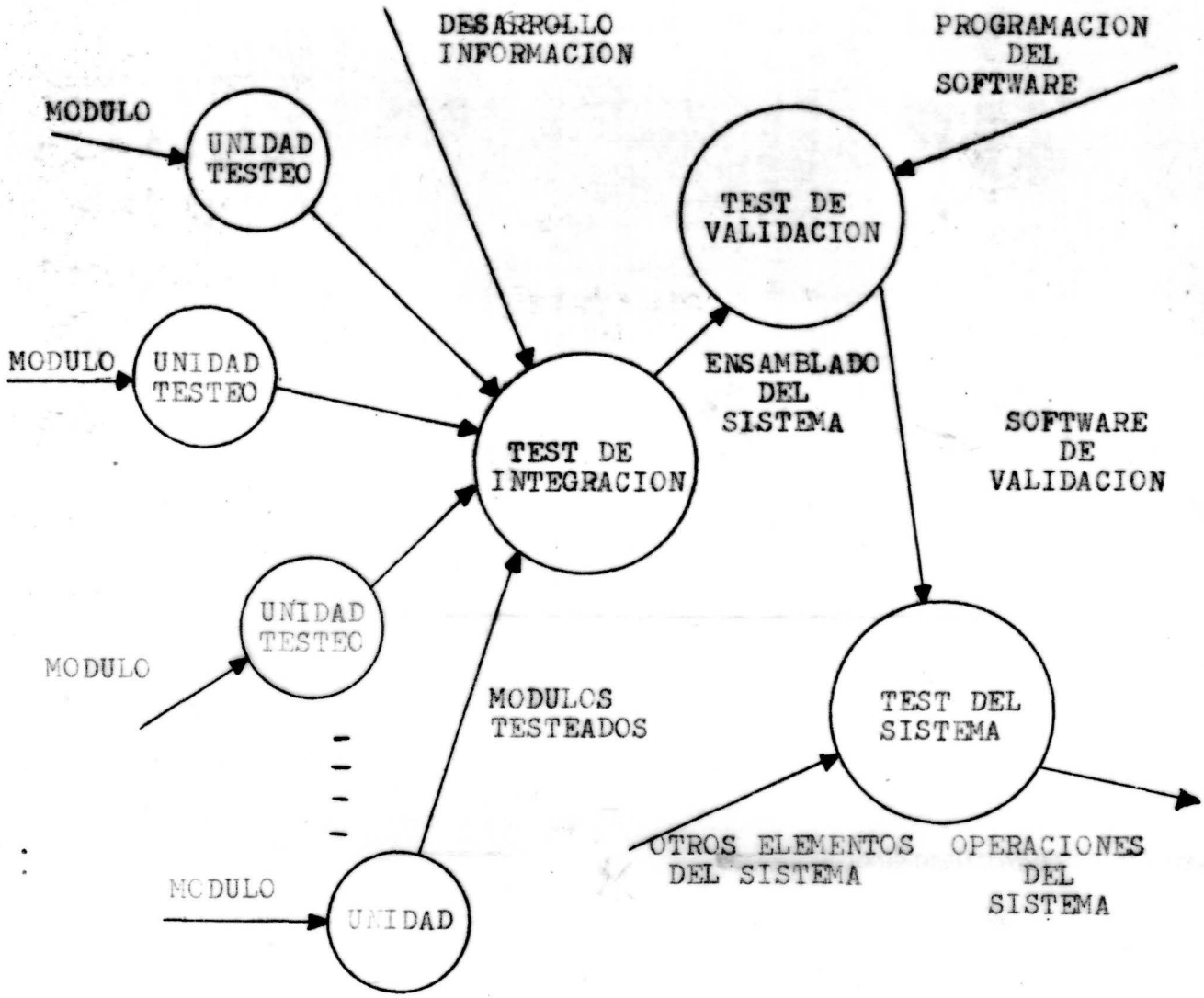
Los resultados de todos los test son evaluados. Esto quiere decir que los resultados obtenidos son comparados con los resultados esperados. Cuando se descubre un error implica que comienza un depurado. El proceso de depurado es la parte más impredecible en el testeo. Un error que indica una discrepancia del 0.01% entre los resultados esperados y los resultados obtenidos puede llevar 1 hora, 1 día, ó 1 mes para diagnosticarlo y corregirlo. Lo incierto y lo inherente del depurado hace difícil que el testeo sea digno de confiaza.

Si los grandes errores que requieren una modificación del desarrollo son encontrados con regularidad, la calidad y confianza en el software son suspendidas y otro test es iniciado. Si, por otra parte, las funciones del software parecen que trabajan correctamente y los errores encontrados son fáciles de corregir, hacen que pueda existir una de estas dos conclusiones: La calidad y confianza del software son aceptables;o el test es inadecuado para encontrar errores severos. Finalmente, si el testeo no revela errores existe la duda de que el test no dé lo suficiente y que los errores se esconden con el software. Pero, sin embargo, estos errores serán eventualmente encontrado por los usuarios y corregidos por el desarrollador durante la fase de mantenimiento.

1.3 PASOS EN EL TESTEO DEL SOFTWARE

El testeo dentro del contexto de la ingeniería del software consta de una serie de pasos que son ejecutados secuencialmente. Inicialmente, el test enfoca a cada módulo individual, asegurándose de que este funciona apropiadamente como una unidad; de aquí la denominación 'unidad de testeo'. Lo próximo, es que los módulos deben ser ensamblados, integrados para formar un paquete o un bloque. El 'testeo de integración' direcciona las salidas asociadas con los problemas de verificación y ensamblado. Finalmente, los requerimientos de validación (establecido durante la fase de planificación) deben de ser testeado. El 'testeo de validación' asegura que el software encuentra todas las funciones y las ejecuta según los requerimientos.

El último paso del testeo se sale del medio de la ingeniería del software para entrar dentro del contexto del sistema de computación. El software, una vez validado, debe ser combinado con los otros elementos del sistema (por ejemplo hardware). El 'testeo de sestema' verifica que todos los elementos forman una malla apropiada archivando todas las funciones del sistema y su ejecución.



1.3.1 UNIDAD DE TESTEO

La unidad de testeo enfoca su esfuerzo en la verificación de las pequeñas unidades de desarrollo software (los módulos). Con la descripción detallada usada como guía, se testea cada módulo para descubrir un error. La relativa complejidad del test y el descubrimiento de errores está limitado por el alcance restringido establecido por la unidad de testeo.

1.3.2 TESTEO DE INTEGRACION

A este punto llegamos una vez que han sido testeadas satisfactoriamente todas las unidades por separado. Luego de esto surge la pregunta: ¿ si todos los módulos funcionaban individualmente, porque no funcionan cuando los ponemos juntos ?. El problema consiste, desde luego, en ponerlos juntos. Los datos pueden ser perdidos a través de interfaces; un módulo puede afectar a otro idnavertidamente; las subfunciones, cuando las combinamos pueden que no produzca una función mayor; una estructura global puede presentar problemas. Desgraciadamente esta lista la podemos hacer lo larga que queramos.

El testeo de integración en una técnica sistemática para ensamblar el software al tiempo que realiza test para encontrar errores asociados con los interfaces. El objetivo es tomar módulos testeado y construir una estructura software que haya sido determinada por un equipo de desarrollo.

1.3.3 TESTEO DE VALIDACION

La culminación del testeo de integración es completado por el ensamblado como un paquete. Los errores de interfaces han sido descubiertos y corregidos, y una serie de test finales de software (test de validación) pueden empezar. La validación puede ser definida de muchas formas, pero una validación tiene lugar cuando las funcio

nes software están en una situación que puede ser esperada razonablemente por un requerimiento o un usuario.

1.3.4 TESTEO DEL SISTEMA

El paso final es el testeo de sistema frecuentemente llamado 'test de aceptación'. Un test de aceptación puede planificar una serie de test para el planteamiento y sistemática ejecución de una serie de test. El hecho de testeo de aceptación puede ser llevado a cabo por períodos de semanas o meses, por medio del reconocimiento de un cúmulo de errores que podrian degradar el sistema a través del tiempo.

2 EL ARTE DEL DEPURADO

El testeo del software es un proceso que puede ser sistemáticamente planeado y especificado. El depurado tiene lugar como consecuencia del éxito del testeo.

Aunque el depurado puede y debería ser un proceso ordenado, es todavía un arte. Un ingeniero de software, evalúa los resultados de un test, y lo confronta con un indicador de problema de software.

2.1 CONSIDERACIONES PSICOLOGICAS

Desafortunadamente, aquí parece ser una evidencia que la proeza del depurado es un intento innato en el ser humano. En algunas personas es realidad pero en

otras no. Aunque la evidencia experimental sobre el depurado está abierta a muchas interpretaciones, distintos depurados han sido realizados por programadores con la misma educación y experiencias similares.

2.2 ACERCANDONOS AL DEPURADO

El depurado tiene como objetivo por encima de todo encontrar y corregir las causas de error de software. Este objetivo es realizado por una combinación de evaluación sistemática, intuición y fortuna. Sin embargo, tres categorias de depurado pueden ser realicadas.

- Rudimentario.
- Eliminación de causa.
- Rastreo.

La categoría rudimentaria de depurado es probablemente la más común y al menos el método más eficiente para aislar la causa de un error de software. Aquí se produce una gran cantidad de información que nos puede guiar a la hora de resolver un problema, pero es mucho más frecuente la pérdida de tiempo y esfuerzo.

La segunda categoría de depurado (eliminación de causa) es manifestada por inducción o por deducción. En este método se emplean unos datos los cuales están organizados para aislar la causa potencial del error. Luego se inventa un caso hipotético, que mediante el uso de los datos anteriores sirve para aprobar o desaprobar dicha hipóte

sis. Alternativamente, se desarrolla una lista de todas las posibles causas y luego mediante un test se elimina o se verifica cada causa. Si un test inicial indica que una hipótesis de una causa particular muestra promesa, entonces los datos son refinados en un intento de aislar el error.

El rastreo es una técnica común de depurado que se puede usar con éxito en pequeños programas. Lo primero que se hace es buscar la zona donde se detectan los síntomas, luego se tracea el código fuente al revés (manualmente) hasta que se encuentre el sitio del error . Desafortunadamente, como el número de líneas fuentes aumenta, el número de partes a rastrear se puede hacer interminable, siendo esto uno de los inconvenientes que presenta esta técnica.

Todo lo dicho encima puede ser suplido con herramientas de depurado. Se pueden aplicar una gran cantidad de compiladores de depurado, ayuda dinámica de depurado (sería el traceo), generación de test automático,etc.

Sin embargo, las herramientas no son un substituto debido a su cuidada evaluación basada sobre un desarrollo completo de software documentado y clarificado por el código fuente.

En muchos casos, el depurado del software de las computadoras es como la solución de un problema en el mundo de de los negocios. Existe un proceso de depurado, llamado 'el método' que es una adaptación del manejo de la técni

ca para resolver el problema. Este proceso desarrolla 'unas especificaciones de desvío' que describe el problema mediante las preguntas: ¿qué?, ¿cuándo?, ¿dónde? y ¿a qué se extiende?. Las especificaciones están representadas en la siguiente tabla.

	ES	NO ES
¿QUE OCURRE?		
¿CUANDO?		
¿DONDE?		
¿A QUE SE EXTIENDE?		

Cada una de las preguntas de arriba está dividida en 'es' y 'no es'. Estas respuestas representan una clara distinción entre lo que ha de ocurrir y lo que no ha de ocurrir. Una vez que la información acerca del error ha sido grabada, una causa hipotética es desarrollada sobre la base de distinción de las repuestas de 'es' y 'no es'.

PARTE II

INTRODUCCION

INTRODUCCION

Pascal es un lenguaje de propósito general desarrollado alrededor del año 1970 por Niklaus Wirth en el E.T.H. de Zurich.

El primer trabajo sobre Pascal aparece como un intento de obtener un sucesor del Algol 60. Sus autores son N.Wirth y C.A.R. Hoare y está publicado en 1966. Sus motivos fueron desarrollar un lenguaje que soportara de forma sistemática un software más comprensible y fiable del que disponían, y que fuera lo suficientemente compacto y pequeño como para permitir una implementación rápida y eficiente en los computadores que tenían disponible en aquel tiempo. Un primer compilador completo estaba en el E.T.H. de Zurich para un CDC 6430, que tenía dicha escuela. Dos años más tarde, el compilador completo estaba funcionando y en 1971, se desarrolla otro compilador en Queen's University en Belfast. La primera definición del lenguaje aparece en 1971 y, en 1974 se publica el User Manual and Report con lo que se conoce el Pascal Standard.

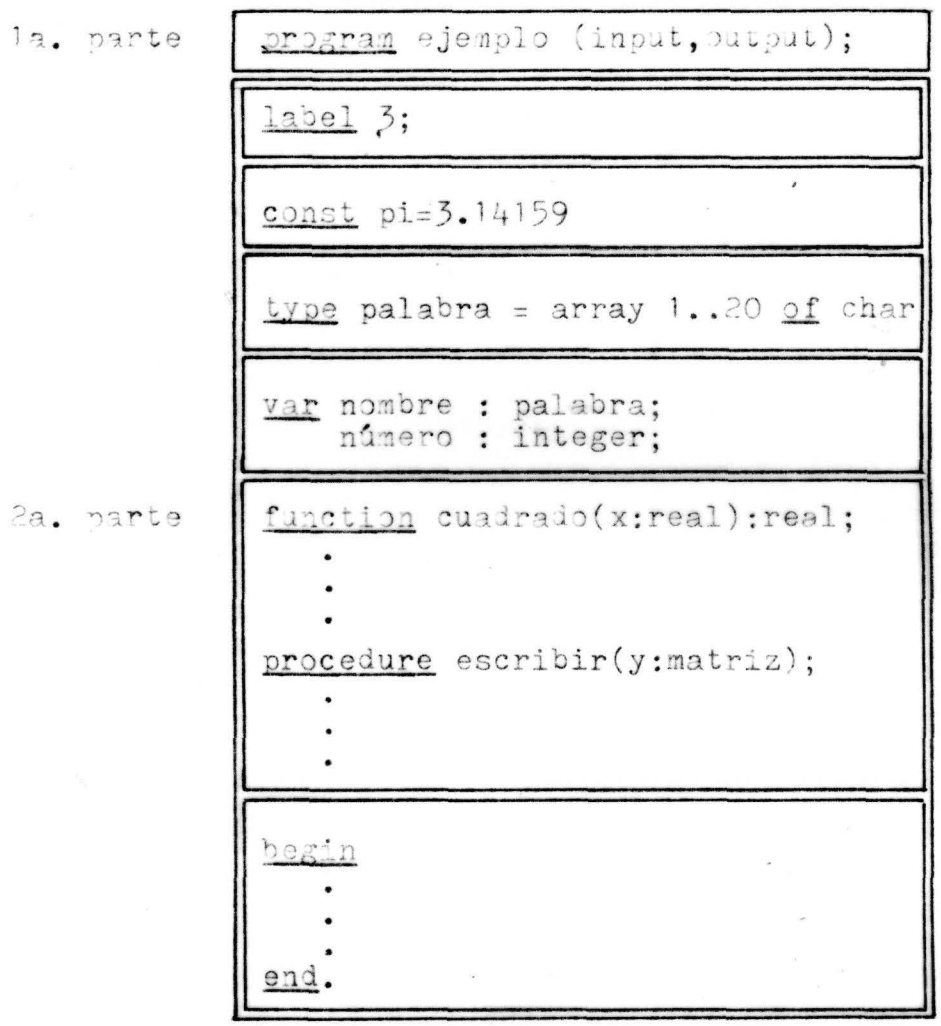
Durante el período 1972-1975, el número de usuarios aumenta pero siempre dentro del área europea y del ambiente académico.

En 1975, Kenneth Bowles, profesor de la Universidad de California en San Diego, toma el lenguaje Pascal como soporte para su curso de promagación. Ello fuer-

za a desarrollar el software nesesario para correr programas Pascal en sus computadoras, un B6700 y un PDP-11 lo que lleva a la formación de un grupo de trabajo especializado en software Pascal. Al año siguiente , 1976 este grupo suministra copias de su propio software a usuarios ajenos a la Universidad, y a partir de 1977 se dedican al desarrollo de compiladores e interprétes para distintas máquinas, sobre todo microcomputadores. Es a partir de entoces que es fácil y barato obtener un compilador Pascal y el boom estalla.

ESTRUCTURA DEL PROGRAMA PASCAL

ESTRUCTURA GRAFICA DE UN PROGRAMA EN PASCAL



ESTRUCTURA DEL PROGRAMA PASCAL

Un programa Pascal consta de dos partes: un encabezamiento y un bloque. En el encabezamiento se le da un nombre al programa y se listan sus parámetros, es decir los ficheros utilizados en dicho programa.

El bloque consta de seis secciones, que son, en el orden en que aparecen:

1) Declaración de etiquetas.
2) Definición de constantes.
3) Definición de tipos.
4) Declaración de variables.
5) Declaración de funciones y procedimientos.
6) Conjunto de instrucciones.

1.1 ENCABEZAMIENTO

Consiste en darle un nombre al programa que vamos a crear. Para ello debemos proceder como sigue:

```
PROGRAM NOMBRE;
```

Es decir, debemos poner la palabra program seguido del nombre del programa y terminado en un punto y coma. Muchas veces se verá después del nombre y entre paréntesis las palabras input y output, es decir:

```
PROGRAM NOMBRE (INPUT,OUTPUT);
```

Input y output especifica que este programa realiza operaciones de entrada y salida; input y output se llaman parámetros del programa y son los nombres de los ficheros standard del sistema de entrada y salida res-

pectivamente.

Ejemplo de encabezamiento podrían ser:

```
PROGRAM COSTE;
PROGRAM NOMBRES (INPUT,OUPPUT);
PROGRAM LISTADO (LISTING);
```

1.2 DECLARACION DE ETIQUETAS

Cualquier sentencia en un programa puede ser reseñada anteponiéndole una etiqueta. La etiqueta debe de estar formada por números enteros y hemos de tener en cuenta que el pascal-80 el mayor valor de etiqueta que se puede poner es 9999.

La forma general de etiquetado es:

```
LABEL ETIQUETA,ETIQUETA,ETIQUETA;
```

Ejemplo de esto sería:

```
LABEL 1,2,3;
LABEL 1999,5000,9999;
```

Nota: Se anconseja que siempre que sea posible no utilizar los etiquetados.

1.3 DEFINICION DE CONSTANTES

Definir una constante es establecer un identificador como sinónimo de un valor constante. El programador podrá entonces utilizar el identificador en lugar del valor.

La forma general de definición de constante es:

```
CONST IDENTIFICADOR=CONSTANTE;
```

La constante a parte de un número puede ser también una cadena de caracteres; en caso que sea una cadena

de caracteres ésta debe de estar cerrada entre comillas.

Ejemplo válido de esto es:

```
CONST FILAMAXIMA=6;
CONST NOMBRE='PEPE';
```

1.4 DEFINICION DE TIPOS

Es un mecanismo para crear un nuevo tipo de datos. Una cosa que es muy importante es que cada variable en un programa debe de estar asociada a uno y sólo un tipo de datos.

La forma general sería:

```
TYPE IDENTIFICADOR = (TIPO);
```

Ejemplo de esto sería:

```
TYPE DIAS = (LUNES,MARTES,MIERCOLES,JUEVES,VIERNES,
             SABADO,DOMINGO);
```

1.4.1 TIPO SUBRANGO

El tipo subrango es un subconjunto secuencial de un tipo de datos determinado. Consiste en un límite inferior y en otro superior separados por el símbolo especial '..'. Los límites deben de ser valores constantes del mismo tipo y el límite inferior nunca podrá ser mayor que el límite superior.

La forma general sería:

```
TYPE NOMBRE = CONSTANTE..CONSTANTE;
```

Ejemplo de esto sería:

```
TYPE DIAS_DEL_AÑO = 1..365;
```

1.5 DECLARACION DE VARIABLES

Una declaración de variables asocia un identificador con un cierto tipo .

La forma general sería:

VAR IDENTIFICADOR : TIPO;

Ejemplo de esto sería:

VAR ALPHA,BETA,GAMMA : INTEGER;

VAR NOMBRE : STRING 30 ;

Todas las variables que intervienen en un programa deberán de ser declaradas al principio del mismo.

1.6 DECLARACION DE FUNCIONES Y PROCEDIMIENTOS

Aquí es donde se define todas las subrutinas a las que el programa va a hacer referencia más tarde. Estos subprogramas se denominan funciones si devuelven un valor al término de su ejecución y procedimiento en el caso que no lo hagan.

Una cabecera de función tiene la forma:

FUNCTION IDENTIFICADOR :TIPO DE RESULTADO;

o bien,

FUNCTION IDENTIFICADOR (PARAMETROS DE LA FUNCION):
TIPO DE RESULTADO;

Una declaración de función contiene la siguiente información:

1) El nombre de la función.
2) El nombre y tipos de los parámetros de la función.
3) El tipo de resultado que se obtiene.

4) Las sentencias que constituyen el cuerpo de la función y que calculan el valor (resultado) a partir de los valores de los parámetros.

Un ejemplo de función podría ser:

```
FUNCTION TG(Z:REAL):REAL;
  BEGIN
     TG:= SENO(Z)/COS(Z);
  END
```

Indicamos que:

-tg es el nombre de la función.

-(z:real) es el parámetro de la función. En este caso la funcioń tiene un único parámetro, cuyo identificador es z y es de tipo real.

-:real define que el resultado de la función que se entregara al punto de llamada es de tipo real.

-las sentencias comprendidas entre begin y end (una sola en este caso) constituyen el cuerpo de la función.

-dentro del cuerpo de la función la sentencia de asignación: tg:=sin(z)/cos(z) ,asigna a tg(el nombre de la función) el resultado del cálculo.

Escrita la declaración de la función tg del ejemplo, dos posibles llamadas de tg son:

TG(ALFA) Y TG(GRD*PI/180)

La declaración de un procedimiento tiene la misma forma que la de un programa, comenzando aquél con una

cabecera de procedimiento la cual puede tomar una de estas dos formas:

```
PROCEDURE IDENTIFICADOR;
PROCEDURE IDENTIFICADOR(PARAMETROS DEL PRODEDIMIENTO);
```

Las declaraciones de procedimiento, igual que las declaraciones de funciones, deben insertarse entre las declaraciones de variables y el cuerpo del programa.

Un ejemplo completo de esto sería:

```
PROGRAM FECHA(INPUT,OUTPUT);
 TYPE
    DIASMES = 1..31;
    MESES =1..12;
    AÑOS = 1900..2000;
 VAR
    DIA:DIAMES;
    MES:MESES;
    YEAR:AÑOS;
 PROCEDURE DATE(DD:DIAMES;MM:MESES;AA:AÑOS);
    BEGIN
       WRITE(DD:2,'/',MM:2,'/',(AA MOD 100):2);
    END
BEGIN
     READ(DIA,MES,YEAR);
     DATE(DIA,MES,YEAR);
END.
```

El ejemplo representa un programa completo en el cual

está incluido un procedimiento(date) sin parte de declaración; inserto entre las declaraciones de las variables y el cuerpo del programa. El cuerpo del procedimiento se compone de una sola sentencia, la cual realiza la operación de escribir una fecha bajo la forma DD/MM/AA. La activación del procedimiento 'date' se realiza mediante la llamada de procedimiento;

```
DATE(DIA,MES,YEAR);
```

Los parámetros DD,MM,AA del procedimiento 'date' son todos ellos datos cuyos 'valores' iniciales al activar el procedimiento van a ser: día, mes, year. En este ejemplo no existe nigún parámetro que represente un resultado.

En los ejemplos de funciones y procedimientos vistos hasta ahora, los parámetros se han utilizado únicamente bajo un solo aspecto y es suministrar valores para ser utilizados por una funcion o un procedimiento. A los parámetros utilizados bajo esta forma se les denomina 'parámetros valor'.

Si tenemos parámetros que deben servir de salida o entrega de resultados, entonces tienen que ser de un nuevo tipo denominado 'parámetros variables' por ir precedido por la palabra reservada var.

Ahora veremos un ejemplo de un procedimiento que calcule el valor del trinomio de segundo grado ax+bx+c y que devuelva el resultado en el párametro variable y.

```
PROCEDURE TRINOMIO(A,B,C,X:INTEGER;VAR Y:INTEGER);
      BEGIN
           Y:=A SQR(X)+B X+C;
      END
```

Una llamada del procedimiento trinomio es: trinomio (i+1,4,-2,j,m) que se corresponde así:

```
PROCEDURE TRINOMIO(I+1,4,-2,J,M)
```

El cuerpo del procedimiento calcula (i+1) +4j-2, colocando el resultado en m.

```
m:=(i+1)j+4j-2
```

La sintáxis de declaración para un procedimiento en el Pascal-80 es la misma que para un pascal standard excepto en una cosa. En el Pascal-80 las funciones y procedimientos no pueden ser usada como argumentos de otras funciones o procedimientos.

CONCEPTOS BASICOS

Un programa Pascal está construído como una frase: es una sucesión de palabras (a veces números) terminada en un punto. Las palabras de un lenguaje de programación se llaman identificadores, puesto que identifican objetos y acciones. En Pascal estos identificadores pueden ser de tres tipos:

A) Los identificadores reservados, que son las palabras del lenguaje

AND	ARRAY	BEGIN
CASE	COMPONENT	CONST
DIV	DO	DOWNTO
ELSE	END	FOR
FILE	FORWARD	FUNCTION
GOTO	IF	IN
LABEL	MOD	NOT
OF	OR	PACKED
PARTITIONED	PROCEDURE	PROGRAM
PUBLIC	RECORD	REPEAT
SET	SEGMENT	SEPARATE
THEN	TO	TYPE
UNTIL	VAR	WHILE
WITH	NIL	

B) Los identificadores definidos por el usuario, que son objetos y acciones.

C) Los identificadores predefinidos.

Para que no exista riesgo de confusión entre una palabra de lenguaje (reservada) y un objeto o una acción definida, no es posible crear un idintificador ya reservado para el lenguaje. Por el contrario, es poseble crear un identificador que esté ya definido previamente pero en este caso el nuevo identificador, el predefinido, borra al antiguo. Esta posibilidad no es aconsejable pues una vez declarado el nuevo identificador, el predefinido no es ya accesible y no puede ser utilizado más.

Ahora bien, ¿cómo creamos un identificador?. En primer lugar, el identificador comienza por una letra ,seguida en ocasiones por una o más letras o cifras. Concretamente en el Pascal-80 un identificador debe comenzar por una letra y puede tener una extención de como máximo ocho caracteres (o cifras).

Ejemplos válidos en el Pascal-80

IDENTIFICADOR

I

B2

X2W473

Ordenador

Ejemplos no válidos el el Pascal-80

Identificador erróneo (No es válido por tener dos palabras).

A+1 (El signo + no es una letra ni una cifra).

2B (No vale por comenzar por una cifra).

Como hemos vistos la longitud de un identificador está limitada a 8 caracteres. Si un identificador es más largo, el compilador Pascal no tomará más que los 8 primeros, e ignora los demás. Así los 3 identificadores siguientes serán, para un compilador Pascal, idéntico , aunque no lo parezca.

DOCUMENTO

DOCUMENTAL

DOCUMENTACION

Es posible insertar el carácter subrayado '_' en un identificador con la condición de que esté disponible en el equipo. Es el único carácter no alfanumérico (no confundir con el guión) que se autoriza en el interior de un identificador. Un ejemplo de esto sería:

IMPRESORA_EN_LINEA

El carácter subrayado es ignorado por el Pascal cuando lo encuentra en un identificador. Así,los identificadores A_MAS_1 y AMAS1 son idénticos y representan el mismo objeto o la misma acción.

JERARQUIA DE LOS OPERANDORES EN PASCAL

1) Paréntesis.
2) Multiplicación y división.
3) Suma y resta.

ASIGNACION DE VARIABLES

La asignación de variables permite al promagador dar un valor a una variable durante la ejecución del progra-

ma. En Pascal está asignación se debe realizar poniendo dos puntos seguidos del signo igual (:=).

La forma general de esto sería:

```
VARIABLE:=EXPRESION
```

Veamos ahora algunos ejemplos:

```
x:=3.0
y:=2.0+3.0
nombre:='Pepe'
```

En Pascal se usa el punto y coma como un signo delimitador entre sentencia y sentencia.

Veamos un programa completo donde se aprecia esto último

```
PROGRAM SUMA;
  VAR X,Y,Z: INTEGER;
  BEGIN
    X:=3;
    Y:=4;
    Z:=5;
    SUM:=X+Y+Z
  END.
```

Notar que el punto y coma separa cada sentencia del resto. Como se observa no es necesario poner punto y coma después del BEGIN o después de la última sentencia de asignación.

ENTRADAS Y SALIDAS SIMPLES

Veamos ahora cómo se puede escribir y leer a partir del Pascal. En Pascal existen dos procedimientos de impresión que son: WRITE y WRITELN.

WRITE('A'). Permite escribir lo que está entre apóstrofes,(en este caso el caracter A) sobre la pantalla sin cambiar de línea.

WRITELN('A').Permite escribir sobre la pantalla después de cambiar de línea.

WRITELN Sólo (sin párametros) permite cambiar simplemente de línea , sin escribir nada. El próximo WRITE o WRITELN se hará entonces sobre la línea siguiente.

Veamos un ejemplo de esto:

```
PROGRAM IMPRESION;
  BEGIN
       WRITE('He aquí nues');
       WRITE('tro segun');
       WRITELN('do');
       WRITE('Programa Pasca');
       WRITELN('l')
  END.
```

El resultado una vez ejecutado el programa sería:

```
He aquí nuestro segundo
Programa  Pascal
```

Como se observa se ha pedido al ordenador cambiar de línea únicamente después de 'do' y de la letra 'l'. Los

procedimiento WRITE y WRITELN no se limitan a la impresión de caracteres. Es posible escribir la variable I (su valor) por:

WRITE(I) o WRITELN(I); según se desee o no cambiar de línea.

Se puede escribir varias constantes o variables en un solo WRITE o WRITELN separándolas por comas.

Un ejemplo de esto último sería:

```
WRITELN('I=',I,'J=',J);
```

Si deseamos imprimir un apóstrofe en medio de un texto , es nesesario indicar al compilador que este apóstrofe no es el de fin. Por esta razón conviene seguir esta importante regla: Todo apóstrofe enuna cadena de caracteres debe duplicarse.

Así para imprimir el texto:

```
Esto 'no' es así
```

se deberá poner:

```
WRITELN(EEsto ''no''es así');
```

De la misma manera que para la escritura, existen dos procedimientos para la lectura que son: READ y READLN.

READ(I). Permite leer la variable, es decir, leer caracteres hasta que sea introducido un valor posible para I.

READLN(I). Funciona de la misma manera, pero se pasa desde el final de la línea, (este final es indicado al ordenador por la tecla RETURN).

Si se le ordena READLN(I) y no se encuentra nigún valor

aceptable para I sobre la línea, el programa se parará y señalará error. READLN solo sin párametros acepta caracteres hasta que aparezca una señal de final de línea RETURN.

Igual que para los procedimientos de escritura, es posible leer variables en una sola instrucción. Así READ (I,J); o READLN(I,J); permite leer los valores de las variables I y J . (si I, J son, las dos variables numéricas, deberán separarse como es natural, por un carácter no numérico).

Cuando se desee incluir comentarios, se pueden hacer de dos formas:

A) Comenzarlos por '(+' (un paréntesis abierto seguido de un asterisco) y terminarlo por '+)' (un asterisco seguido de un paréntesis cerrado).

B) Empezarlo por '{' (una llave abierta)y acabarlo en '}' (una llave cerrada), con la condición de que el equipo acepte las llaves.

Todo lo que se encuentra entre (+ o y o +) será el comentario cualquiera que sea la longitud de éste, a condición que no contenga +) o que indicara su final. Si un programa posee un comentario abierto con llave, solo una llave podrá cerrarlo, pues de otro modo no tendrá nigún efecto colocar un asterisco y un paréntesis. Lo mismo ocurre al contrario.

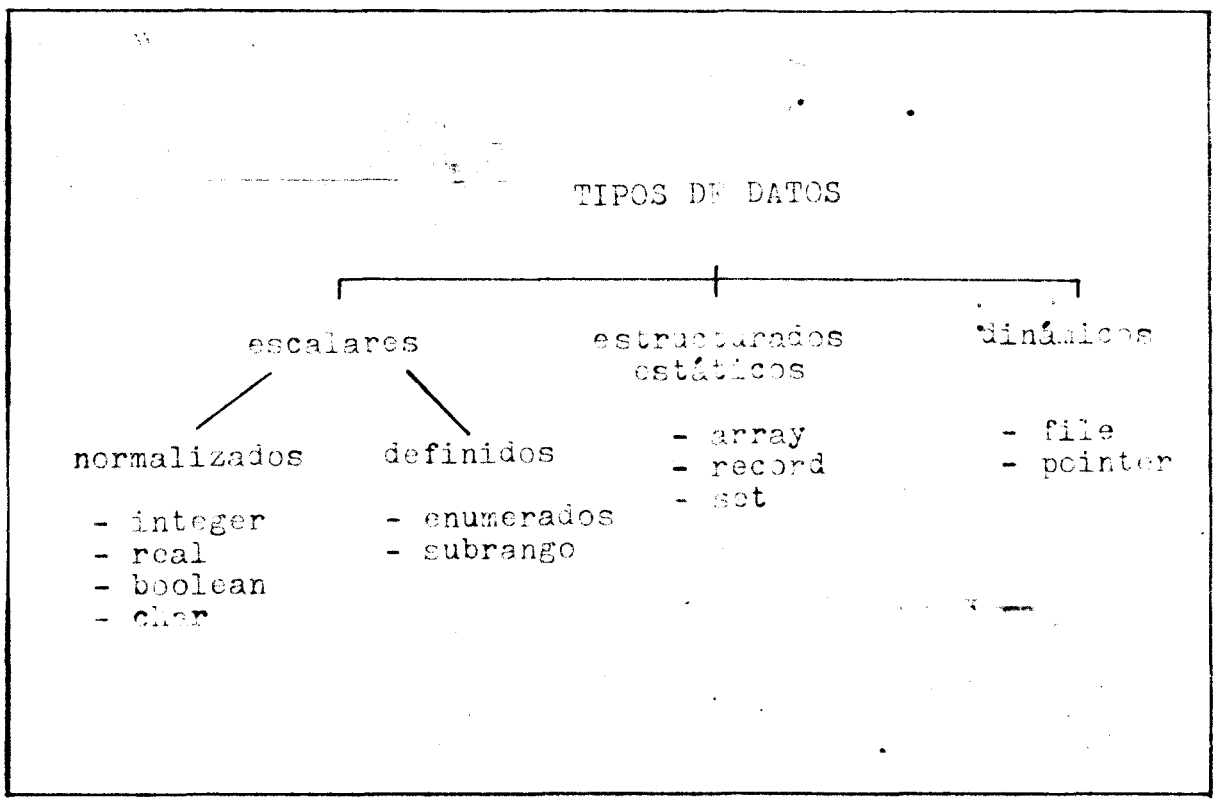
Ahora veremos un programa completo que calcula la media de tres notas.

```
PROGRAM MEDIA(INPUT,OUTPUT);
(+PROGRAMA PARA ENCONTRAR LA MEDIA+)
  VAR MARK1,MARK2,MARK3: INTEGER;
      AVG,SUM: REAL;
  BEGIN
     READ(MARK1,MARK2,MARK3);
     SUM:=MARK1+MARK2+MARK3;
     AVG:=SUM/3;
     WRITELN('MARK1=',MARK1);
     WRITELN('MARK2=',MARK2);
     WRITELN('MARK3=',MARK3);
     WRITELN('MEDIA=',AVG)
  END.
```

TIPO DE DATOS E INSTRUCCIONES

1 TIPOS DE DATOS

Los tipos de datos disponibles en Pascal se encuentran en la siguiente figura.



1.1 ESCALARES NORMALIZADOS

1.1.1 INTEGER (ENTEROS)

Es un subrango cuyos límites superiores e inferiores están comprendidos entre 'maxint' y 'minint' respectivamente. Estos valores dependen de la implimentación del Pascal en cada ordenador. En el Pascal-80 está definido como:

INTEGER= -32768..32767;

1.1.2 REAL (REALES)

Las variables declaradas REAL corresponde a los números tratados por los calculadores científicos. Se dispone de siete cifras significativas, así como de un exponente cuyos valores se extienden desde -38 hasta +38.

Las variables reales pueden, en general memorizar valores desde -3.40+10E38 hasta 1.17+10E-38 para los números negativos, y desde 1.17+10E-38 hasta 3.40+10E38 para los números positivos.

1.1.3 BOOLEAN

Las variables declaradas de tipo boolean son variables lógicas que no pueden tomar más que dos valores que son: TRUE(VERDADERO) o FALSE(FALSO), donde true es mayor que false. Un ejemplo en Pascal-80 es:

VAR NOMBRE:BOOLEAN;

1.1.4 CHAR (CARACTER)

Una variable de tipo carácter puede tomar un conjunto finito de caracteres. En Pascal-80 el grupo de caracteres que puede tomar viene definido por:

```
TYPE CHAR=CHAR())..CHAR(255);
```

Un ejemplo de asignación pudiera ser:

```
VAR PUNT,ALFA,LETRA:CHAR;
```

1.2 ESCALARES DEFINIDOS

1.2.1 ENUMERADOS

Es una situación frecuente el relacionar los números enteros con un conjunto de valores, que el computador no acepta como tales. Por ejemplo, si en una fecha queremos especificar el mes le asignamos un número que será el dato que suministraremos a la máquina. Así el mes de Enero, no es Enero sino que es 1, el de Febrero es 2 y así sucesivamente . Del mismo modo, si quisiéramos representar un conjunto de colores tendríamos que asignarle a cada uno de ellos un número o una letra, pero en nigún caso llamarle por su nombre porque el computador no lo entiende.

Este problema (no llamarle a cada cosa por su monbre,que se presta a confusiones y errores),el Pascal lo solventa permitiendo al usuario definir el tipo que necesita en la sección de definición de tipos. Cuando un usuario define un tipo y más tarde se lo asigna a una variable,

está realmente dándole al compilador la información del conjunto de valores que puede tomar esa variable.
La forma de definir un tipo es yuxtaponiendo a la palabra reservada TYPE el nombre de tipo seguido de un igual y entre paréntesis, todos los valores que puede tomar una variable de ese tipo. En los dos casos anteriores los tipos adecuados serían:

```
TYPE MESES = (ENERO,FEBRERO,MARZO,ABRIL,MAYO,JUNIO,
              JULIO,AGOSTO,SEPTIEMBRE,OCTUBRE,
              NOVIEMBRE,DICIEMBRE);
TYPE COLORES = (ROJO,NEGRO,BLANCO,AZUL,VERDE);
```

Cualquiera de los tipos anteriores define un conjunto ordenado. El orden lo fija la secuencia de enumeración. Es decir, en el caso del ejemplo de los meses ABRIL es mayor que ENERO.

1.2.2 SUBRANGO

Otra situación habitual es la de una variable que sólo puede tomar valores dentro de un rango de un tipo base, fuera del cual su valor es ilegal. Como este rango lo podemos acotar por solo dos valores, el menor y el mayor, porque partimos de un conjunto ordenado, podemos definir un tipo especial para esa variable especificando sus valores máximo y mínimo. En realidad el nuevo conjunto que estamos especificando es un subconjunto del conjunto base.

Como ejemplo veamos el caso de una variable que representa los días laborables de lunes a viernes.

```
TYPE DIAS_LABORABLES = LUNES..VIERNES;
```

1.3 ESTRUCTURADOS ESTATICOS

1.3.1 ARRAY

El array es el tipo estructurado más sencillo y es aquel en el que los elementos que los constituyen son todos de igual tipo.

La estructura de un array puede ser asociada a la de un casillero en el que cada casilla tiene un nombre(índice) para poder acceder a ella, y cada una de ellas está ocupada por elementos del mismo tipo. Por tanto la definición de un array implica la deficición de dos tipos: uno para los elementos del array (contenido de la casilla), y otro para el índice (nombre de la casilla).

La sintaxis de definición requiere que el tipo del índice se indique entre corchetes y después de la palabra array' mientras que el de los elementos se especifique a continuación y después de la palabra 'of'. Por ejemplo, si para formar una palabra reservamos 20 casillas para 20 letras el tipo palabra se definiría:

```
TYPE PALABRA = ARRAY 1..20  OF CHAR;
```

En este ejemplo el tipo del índice es un subrango de los enteros mientras que el tipo base es el adecuado para caracteres, es decir, char.

Del mismo modo un vector matemático de 15 elementos se definiría:

```
TYPE VECTOR = ARRAY  1..15  OF INTEGER;
```

Si los elementos de un array son a su vez arrays obtenemos arrays multidimensionales. Esta puede ser la solución adecuada para definir un tablero de damas, que no es más que un casillero de dos dimensiones cuyo contenido sólo puede ser una pieza negra, una blanca o bien estar vacio. La definición completa comprende la de dos tipos y es la siguiente:

```
TYPE PIEZAS = (BLANCA,NEGRA,VACIA);
     TABLERO = ARRAY  1..8,1..8  OF PIEZAS;
```

También podemos añadir la palabra PACKED (empaquetar) a un array o a un record que lo que hace es ahorrar memoria. Veamos un ejemplo para que quede más claro:

Sea la fecha 10/12/1961. Esta fecha almacenará en 3 palabras:

-Una palabra para almacenar el número del día.

-Una segunda palabra para almacenar el número del mes.

-Una tercera para almacenar el año.

Ahora bien si en cada palabra cabe hasta 4 cifras, en el ejemplo anterior los valores del dia y el mes podrían almacenarse en una sola palabra.

```
1012  1961
```

Y por consiguiente la fecha ocupa ahora 2 palabras en lugar de 3.

Esta forma de tratar los datos, es decir conseguir un mayor aprovechamiento de la memoria, se denomina forma empaquetada (packed).

1.3.2 RECORD

Consiste en la formación de un tipo estructurado agrupando elementos de cualquier tipo. Sería el caso de un casillero en el que el contenido de las casillas no sea siempre del mismo tipo.
Esta estructura es idónea para definir una fecha, en la que los tres elementos que la componen son de distinto tipo. El día es un número entre el 1 y 31, ambos inclusive, el mes responde al tipo meses definido anteriormente y el año es otro número pero perteneciente a un rango distinto al de días. La forma de definirlo es listar entre las palabras 'record' y 'end' los elementos del record y su tipo.

El ejemplo de la fecha sería:

```
TYPE FECHA = RECORD  DIA:1..31;
                     MES:MESES;
                     AÑO:1900..2000
               END;
```

El mismo caso ocurre cuando queremos describir a una persona a partir de unas pocas características relevantes como su nombre y apellidos, fecha de nacimiento, sexo y

estado civil. La definición adecuada sería un record de la siguiente rorma:

```
TYPE PERSONA = RECORD NOMBRE,APELLIDO:STRING 10 ;
                      NACIMIENTO: FECHA;
                      ESTADO CIVIL: (SOLTERO,CASADO,
                                     VIUDO,SEPARADO)
                END ;
```

Pascal, además, posee la posibilidad de declarar variantes de un mismo tipo record, permitiendo la posibilidad de elegir unas determinadas características de la persona en función de su, por ejemplo, estado civil.

1.3.3 STRING

El Pascal-80 tiene un tipo de string predeclarado el cual es similar a un packed array of char, pero también tiene una longitud dínámica de byte. Sin embargo aunque la asignación de longitud de una cadena permanezca fija, esta longitud dinámica puede variar de acuerdo con la asignación hecha. La asignación de longitud de una cadena puede ser de 1 a 255 caracteres. Si en una cadena no especificamos nada, el compilador nos tomará una longitud máxima de 80 caracteres.

Si por ejemplo tenemos:

```
NAME:STRING 30 ;
```

La variable nombre es del tipo string, con una longitud máxima de 30 caracteres.

Si por ejemplo tenemos:

```
ADDRESS:STRING;
```

La variable address es del tipo string, con una longitud máxima de 80 caracteres.

1.3.4 SET

Un set es el último tipo estructurado estático del que disponemos y agrupa un conjunto de elementos de un tipo base del que, en un momento dado, nos puede interesar uno solo de ellos, o uno o varios subconjuntos del conjunto base. Es decir, un set define un conjunto de subconjuntos de tipo base. Conserva la misma idea que el conjunto matemático del que toma las propiedades y operaciones básicas.

El tratamiento de textos utiliza la idea de conjunto a la hora de discriminar grupos de caracteres. Por ejemplo, si de un texto cualquiera solo procesamos las letras, podemos agrupar los restantes caracteres (signo de puntuación y blancos) en un conjunto , de modo que si el caracter que estamos examinando pertenece al él no sea procesado.

La variable que representa este conjunto la podemos denominar noletras y definir su tipo como un 'set' de caracteres. Esta variable es preciso inicializarla asignándole los caracteres adecuados, es decir, los signos de puntuación y los blancos. La declaración de la variable

junto con la asignación del tipo y la inicialización son las siguientes:

```
VAR NOLETRAS: SET OF CHAR;
    .
    .
    .
    NOLETRAS:=  ' ',',','.',';',':'
```

1.4 ESTRUCTURAS DINAMICAS

Los tipos de datos anteriores (simples y estructurados estaticos) todavía dejan pendiente un problema: La formación dinámica de variables. Las variables dinámicas son creadas en tiempo de ejecución, cuando se presenta su necesidad y eliminadas cuando esta necesidad deja de existir.

En Pascal disponemos de una estructura dinámica (el fichero y el tipo puntero) que en unión con el record nos permite estructuras dinámicas de más complejidad.

1.4.1 FICHERO

La estructura dinámica más común (sobre todo en preceso de datos) es el fichero o secuencia. Un fichero es la concatenación de cero o más componentes de un tipo base. Por ejemplo, un grupo de personas podría representarse como:

```
TYPE CLUB = FILE OF PERSONA;
```

En un fichero aparte de inicializarlo y obtener el elemento siguiente al que estamos procesando, podemos añadir elementos en tiempo de ejecución, dándole a esta operación el caracter dinámico que pretendíamos.

Un tipo de fichero muy habitual es el constituído por caracteres y cuya definición es:

```
TYPE TEXTO = FILE OF CHAR;
```

Este tipo facilita la manipulación de textos y en algunos lenguajes es un tipo ya definido al que se le suele denominar 'string'.

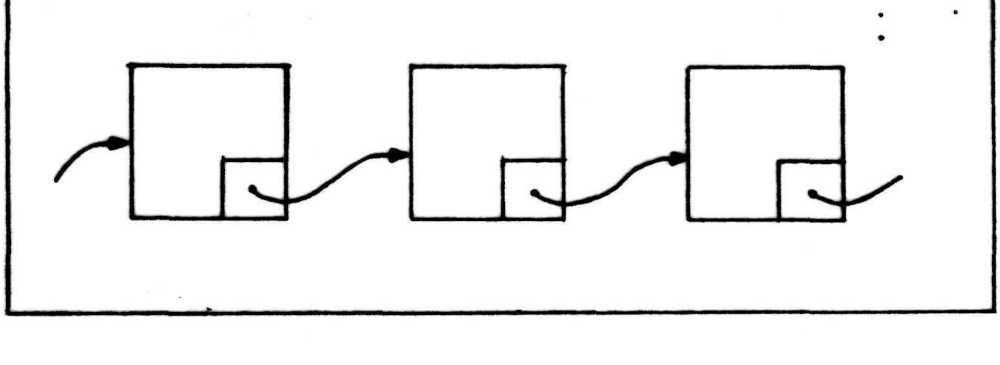
El tipo fichero presenta un inconveniente y es que sólo se puede añadir nuevos elementos al final del mismo, no permitiendo mantener un orden preestablecido.

Si queremos un fichero donde estén reflejados todos los datos personales de un grupo de personas debemos poner la siguiente declaración:

```
TYPE DATOS = RECORD
                NOMBRE,APELLIDO:STRING 20 ;
                DOMICILIO:STRING 30 ;
                EDAD:INTEGER;
                FECHA_DE_NAC:INTEGER;
                TELEFONO:INTEGER;
             END;
VAR DATOS_PERSONALES:FILE OF DATOS;
```

1.4.2 PUNTERO

Hay aplicaciones en las que nos puede interesar que una variable no contenga un valor determinado, sino una referencia a otra variable.
Haciendo un paralelismo con los direccionamientos de los uP, sería el equivalente al direccionamiento indirecto.
Esta referencia a una variable puede ser un campo más dentro de un elemento, permitiendonos formar estructuras como la de la figura. Esta estructura sería la adecuada para manterner ordenado el fichero anterior, ya que la eliminación o inserción de un nuevo elemento implica únicamente el cambio de un par de enlaces.



En esta figura cada cuadro o nodo podría representar una persona, de forma que cada una de ellas lleva el control de quién es la siguiente en la lista. Tal como hemos apuntado, el siguiente en la lista puede cambiar en el caso de inserción o anulación de algún miembro cambiando el contenido de los campos enlace, es decir, cam-

biando los finales de las flechas.

Esta estructura se puede construir en Pascal con el tipo record y el tipo puntero. El record es cada cuadro o nodo, uno de cuyos campos es el puntero, flecha o enlace de un elemento con el siguiente.

Para definir un campo puntero sólo hace falta el tipo del elemento a quien apunta, que en este caso es un nodo igual a aquel del cual forma parte.

La definición completa de cada nodo es:

```
TYPE PERSONA = RECORD .
                      .
                      SIGUIENTE:  PERSONA;
                      .
               END;
```

Además de las listas lineales, el puntero nos permite el tratamiento de estructuras más complejas como son los árboles o las matrices dispersas.

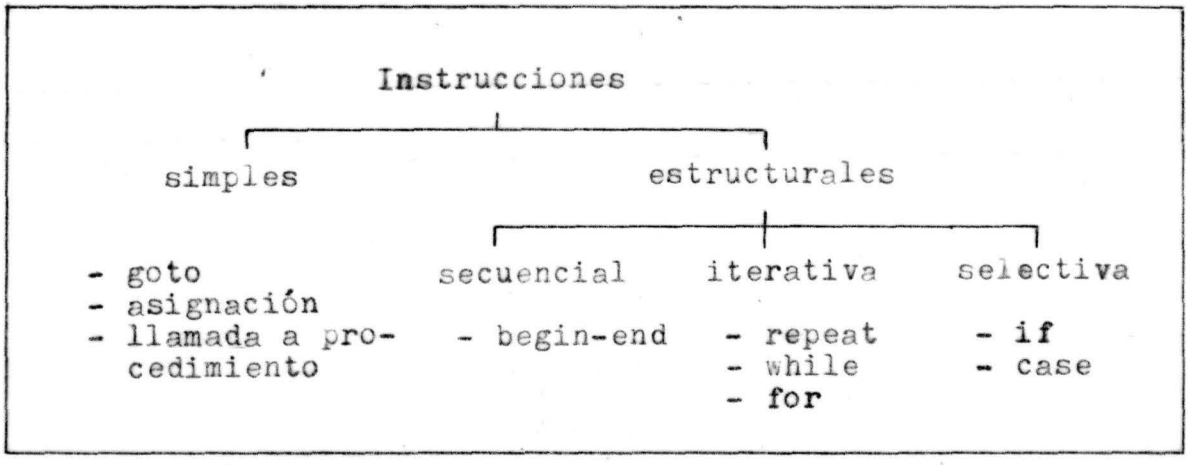
2 INSTRUCCIONES

Tal como habíamos visto al hablar de la estructura de un programa, la última sección (de las 6 que lo componían) contiene el grupo de instrucciones o acciones que ha de realizar nuestro programa.

Dos instrucciones muy comunes son las de asignación de un valor a una variable y la llamada a una subrutina. Son instrucciones muy simples que no suelen ocupar más allá de una línea. Otra instrucción de este grupo es el salto incondicional.

Hay otro tipo de instrucciones más elaboradas que representan esquemas de razonamiento más complejos. Son las instrucciones estructuradas de cuyos tres tipos (secuencial, iterativo y selectivo) vamos a hablar en los apartados siguientes.

En la siguiente figura se muestran las instrucciones disponibles en Pascal.



2.1 INSTRUCCIONES SIMPLES

Como ya henos dicho hay tres instrucciones simples: la asignación, la llamada a subrutina y el salto incindicional.

La asignación reemplaza el valor de una variable por una expresión determinada, cuyo tipo sea coherente con el de la variable, Para dos variables, una llamada número tipo entero y otra llamada encontrado de tipo boolean, son válidas las asignaciones siguientes:

```
NÚMERO := NUMERO+8;
ENCONTRADO := FALSE;
```

La instrucción de llamada a subrutina consiste simplemente en el nombre de la función o procedimiento junto con la lista de sus parámetros si es que existen.

Para un procedimiento llamado leer que transfiere elementos de un fichero llamado entrada a otro denominado salida una llamada válida sería:

```
LEER(ENTRADA,SALIDA);
```

donde hemos supuesto que los únicos parámetros son los dos ficheros.

Si henos definido una función que realiza el cuadrado de un número, cuyo nombre es cuadrado y cuyo único parámetro es el número que hemos de elevar al cuadrado una utilización correcta sería:

```
NUMERO := CUADRADO(NUMERO);
```

por último, la instrucción salto incondicional indica que

el flujo del programa se desvía a otro punto del mismo sin haber realizado ningún test ni haber sastifecho ninguna condición. Evidentemente, requiere la declaración de una etiqueta como punto de llegada. Esta declaración y la utilización de esta instrucción se muestran a continuación:

```
LABEL 1;
  .
  .
GOTO 1;
  .
  .
1:ENCONTRADO := FALSE;
  .
  .
```

2.2 ISTRUCCIONES ESTRUCTURADAS

2.2.1 ESTRUCTURA SECUENCIAL

Es la estructura más sencilla y consiste en la ejecución de varias instrucciones una detras de otra, es decir, en secuencia, pero que a efectos del programa actúen como una sola. La forma en que Pascal expresa esta estructura es encerrando este grupo de instrucciones en un parêntesis formado por los símbolos 'begin' y 'end'.

2.2.2 ESTRUCTURA ITERATIVA

Es frecuente encontrar en un programa, una acción que se ejecuta en un determinado número de veces. Este número puede ser conocido de antemano, o bien estar controlado por una condición. En este último caso, la acción puede realizarse mientras se cumple dicha condición, o bien repetirse hasta que se cumpla. Para todo esto diponemos de tres estructura que son:

WHILE-DO

RDPEAT - UNTIL

FOR - TO - DO

La estructura for se puede dividir en dos según sea ascendente o descendente.

-For ascendente. Su forma general sería:

FOR Variable de control := Valor inicial TO Valor final DO Sentencia ;

-For descendente. Su forma general sería:

FOR Variable de control := Valor inicial DOWNTO Valor final DO Sentencia ;

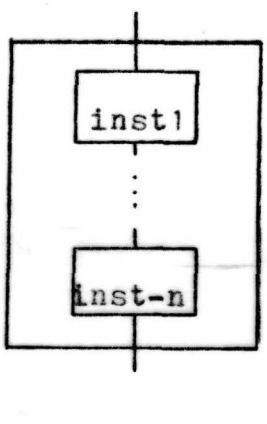
2.2.3 ESTRUCTURA SELECTIVA

La elección de una o varias acciones bajo una condición es tambien un proceso habitual en un programa. Si sólo existe una posible acción a tomar cuando se cumple la condición obtenemos la estructura 'if then'. Si también podemos tomar otra acción cuando la concición

no se cumple, la estructura a utilizar es 'if then else'. Para el casQ general de 'n' acciones a escoger bajo el valor que toma un selector en un rango de 'n' valores, utilizaremos la estructura 'case of'.

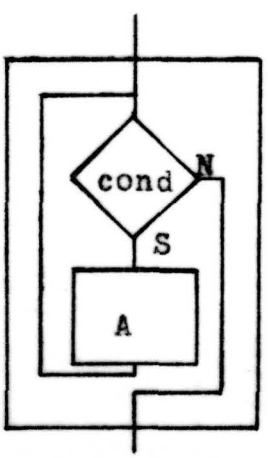
A continuación pondremos las figuras de las estructuras explicadas anteriormente.

A) ESTRUCTURA SECUENCIAL.

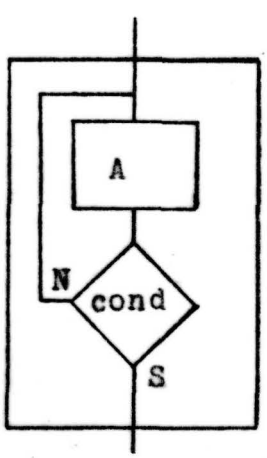



```
begin
      inst-1;
        .
        .
      inst-n
end;
```

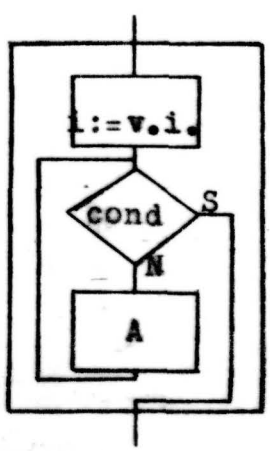
B) ESTRUCTURA ITERATIVA.



While cond **do** A

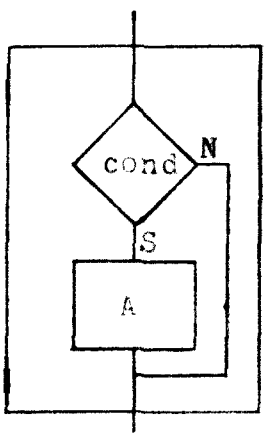


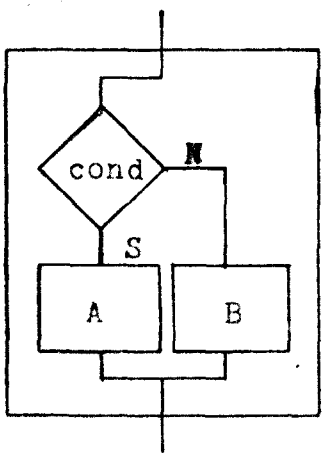
repeat A **until** cond

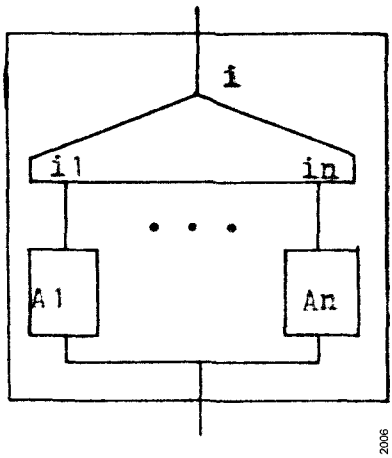


for i:=v.i. **to** v.f. **do** A

C) ESTRUCTURA SELECTIVA.







if cond **then** A

if cond **then** A
else B

case i **of**
i1 : A1
•
•
in : An
end

INDICE DE LOS PROGRAMAS

TITULO	PAGINA
MEDIA	200
PONTECIA	200
LEER_Y_SUMAR	201
VOLUMEN	201
NOTAS	202
CLASIFICACION	202
RAIZCUADRADA	203
SERIE	203
MAYOR_MENOR	204
COMPARACION	205
FIBONACCI	206
FIBONACCI2	207
NOTA2	208
ORDENACION	209
MAYOR	210
FICH1	211
FICH2	212
FICH3	214
FICH4	217
FICHE	220

PROCEDIMIENTOS PREDECLARADOS

El Pascal-80 contiene un número de funciones y procedimientos predeclarados. Estos procedimientos y funciones se encuentran declarados en el sistema de run-time que es el que se encarga del compilado de un programa en Pascal. Las funciones y procedimientos predeclarado incluyen generalmente rutinas útiles como pueden ser rutinas que permiten a los programas en Pascal-80 acceder al poderoso sistema del Isis II

1 PROCEDIMIENTO DE MANIPULACION DE FICHEROS

1.1 RESET (F)

Resetea el fichero F, y posiciona el puntero del fichero en la primera grabación (record) de ese fichero F. La función eof(F) se hace true si el fichero está vacio, de otra manera eof(F) se hace false.

1.2 RESET (F, STRING)

Abre un fichero existente en el Isis II con el nombre STRING , asociando F con ese fichero, y luego ejecuta un RESET (F).

1.3 REWRITE (F, STRING)

Crea un fichero en el Isis II con el nombre STRING , asociando F con ese fichero, y poniendo el puntero al principio de ese fichero.

1.4 CLOSE (F)

Cierra el fichero asociado con F y quita esa asignación.

1.5 PUT (F)

Escribe el valor de la variable buffer F en la posición en que esté apuntando el puntero y avanza hacia la próxima componente.

1.6 GET (F)

Asigna el valor de la componente en que esté apuntando el puntero a la variable buffer F , y avanza el puntero del fichero a la próxima componente. Si la componente a la que apunta el puntero antes de la asignación no existe, la función eof(F) se pone a true, y el valor del buffer queda indefinido. La función eof(f) debe ser falsa de entrada.

1.7 SEEK (F, INTEGER)

Coloca el puntero del fichero F en la componente INTEGER . Hemos de tener en cuenta que lo coloca al final de dicha componente.

1.8 READ (F,V1,...,VJ)

Este procedimiento debe de ser usado sólo en ficheros de tipo o en fichero interactivo. Si F se

omite, la entrada es supuesta. Notar que esta construcción es equivalente a:

READ(F,V1);...;READ(F,VJ)

Si 'VI' es del tipo string (cadena) read(f,vi), leerá hasta el final de la línea poniéndose eoln(F) a true.

1.9 READLN (F,V1,...,VJ)

Esta construcción es equivalente a la secuencia:

READ(F,V1,...,VJ);READLN(F)

Aquí readln(F) se usa para leer saltando una línea posteriormente. Si el fichero no está agotado, eof(F) y eofln(F) se ponen a false.

1.10 WRITE (F,V1,...,VJ)

Al igual que su correspondiente para lectura solo puede ser usado con ficheros tipos y ficheros interactivos. Si F se omite, la salida es supuesta. Notar que esta construcción es equivalente a:

WRITE(F,V1);....;WRITE(F,VJ)

1.11 WRITELN (F,V1,...,VJ)

Esta construcción es equivalente a;

WRITE(F,V1,....,VJ);WRITELN(F)

Aquí writeln(f) escribe un fin de línea sobre el fichero F.

1.12 PAGE(F)

Hace un 'form-feed' para que un carácter sea escrito en un fichero texto o en un fichero interactivo.

2 FUNCIONES DE MANIPULACION DE FICHEROS

2.1 BUFFERREAD(F, ARRAY , LENGTH BREAK-CHAR))
BUFFERERITE(F, ARRAY , LENGHT BREAK-CHAR)

Estas funciones son usadas para leer y escribir en un buffer longitudes arbitrarias desde/hacia un fichero F. El máximo número de bytes que puede ser transferido viene dado por LENGTH ,así que ARRAY, un packed array of char, debería ser al menos de esta longitud.
Si BREAK-CHAR es una expresión entera , empieza a transferir bytes hasta que sea transferido el BREAk-CHAR.
El valor de estas funciones son como sigue, donde LENGTH significa el número de bytes transferido.

BUFFER R/W (F,A,N)=N si no eof,
= LENGTH si eof.

BUFFER R/W (F,A,N,B)= B si se para en el BREAK-CHAR
=LENGTH si eof o no BREAK-CHAR

2.2 BLOCKREAD(F, ARRAY , BLOCKS START-BLOCK)
BLOCKWRITE(F, ARRAY , BLOCKS START-BLOCK)

Estas funciones se usan para leer y escribir bloques de 512bytes desde/hacia un fichero F.

Ambas funciones devuelven el número de bloques actualmente transferido. La longitud del ARRAY (un packed array of char) debe ser multiPlo de 512. Si START-BLOCK se especifica, un SEEK(F,512+START-BLOCK) será ejecutado para que el puntero del fichero F se posicione donde nosotros deseamos; De otra manera la transferencia empezaría por el primer bloque.

2.3 EOF (F) : BOOLEAN

Devuelve verdad si el fichero F se pone al final del fichero,si no devuelve falso. Si F es un fichero disco, eof(F) se hará verdad cuando F esté agotado, por ejemplo cuando el marcador de fichero es igual a la longitud del fichero. Si F está asociado con :CI:,presionando control Z el lugar de un dato de entrada hace que eof se haga verdad.

2.4 EOFLN (F) : BOOLEAN

Devuelve verdad si el fichero texto(F) o el fichero interactivo(F) se pone al final de una línea (de caracteres)si no devuelve falso.

2.5 IORESULT : INTEGER

Devuelve el código de error de la última operación de I/O (entrada/salida). Si no se detecta error,el valor que devuelve es cero.

3 PROCEDIMIENTOS DINAMICOS DE COLOCACION

3.1 NEW (P)

Coloca una variable nueva T y asigna la dirección de T al puntero variable P, donde P viene definida como:

VAR P:T;

Si el tipo de T es un record con variantes, new(P) coloca un area bastante larga de almacenamiento para poder acomodar las variantes más largas.

3.2 NEW (P,T1,...,TJ)

Coloca una variable de tamaño apropiado para la variante, con un valor de campo igual a la constante T1,...TJ; y asigna la dirección de esa variable al puntero variable P.

3.3 MARK (P)

Asigna la dirección de la parte alta de HEAP al puntero variable P.

3.4 RELEASE (P)

Pone en la parte alta de HEAP el valor del puntero variable P.

3.5 MEMAVAIL : INTEGER

Devuelve la deferencia aritmética entre

la parte baja de la ejecución del STACK y la parte alta del HEAP en una palabra de 16 bits.

4 FUNCIONSS ARITMETICAS

4.1 ABS (X) : TYPE OF X

Realiza el valor absoluto de X. El tipo de X debe ser o entero o real, y el tipo de la función es el tipo de X.

4.2 SQR (X) : TYPE OF X

Realiza X*X. El tipo de X debe ser o entero o real, y el tipo de la función es el tipo de X.

4.3 SIN(X), COS(X), LOG(X), EXP(X), LN(X), SQRT(X), ARCTAN(X) : REAL

Todas estas funciones requieren que X sea entera o real, y todas devuelven un resultado de tipo real.

4.4 TRUNC (X) : INTEGER

X debe de ser de tipo real. El resultado es el mayor entero igual o menor a X para X mayor o igual a cero y el menor entero mayor o igual a X para X menor que cero.

TRUNC(5.78)=5

TRUNC(-2.50)=-2

4.5 ROUND (X) : INTEGER

X debe de ser de tipo real. El resultado de tipo entero es el redondeo de X.

5 ATRIBUTOS

5.1 ODD(X) : BOOLEAN

Devuelve verdad si X es un entero impar, si no devuelve falso. X debe de ser de tipo entero.

6 FUNCIONES DE TRANSFERENCIA

6.1 ORD(X) : INTEGER

Obtiene el número ordinal del argumento X en el conjunto de valores definidos como del tipo de X. Notar que X debe de ser de tipo escalar.

6.2 CHR(X) : CHAR

Obtiene el carácter cuyo número ordinal es el valor de X (si existe). X debe de ser del tipo integer.

7 RUTINAS DIVERSAS

7.1 SUCC(X) : TYPE OF X

Obtiene el valor sucesor de X (si existe). X debe ser del tipo escalar o del tipo subrango.

7.2 PRED(X) :TYPE OF X

Obtiene el valor predecesor de X (si existe).

X debe ser del tipo escalar o del tipo subrango.

7.3 SIZEOF (VARIABLE TYPE IDENTIFICADOR):INTEGER

Obtiene el número de bytes que debería ser o son colocados por el argumento.

7.4 EXIT (IDENTIFICADOR DE PRODEDIMIENTO/FUNCION)

Causa una salida del procedimiento o función nombrada, la cual debería pasar por parte de la ejecución dinámica. Si el identificador es el nombre del propio programa, cuando se ejecute, el control será devuelto al comando teclado (consola) de el Pascal-80.

7.5 GOTOXY (COLUMNA,FILA)

Posiciona el cursor en la columna y fila especificada. Ambos argumentos deben de ser enteros. La posición (0,0) es la esquina superior izquierda de la pantalla.

7.6 SETPOINTER (P,V)

Sitúa la dirección de la variable V dentro del puntero variable P.

7.7 PORTINPUT (P,V)

Mete por un port (puerto) P de I/O del 8080/8085 un valor, poniendo la variable V a ese valor.

7.8 PORTOUT (P,E)

Saca el valor de una expresión entera E a un port (puerto) P de I/O del 8080/8085.

8 RUTINAS DE MANIPULACION DE CADENAS

8.1 LENGTH(S) : INTEGER

Devuelve la longitud de la cadena S.

8.2 POS(EXPRESION ,S) : INTEGER

Devuelve la posición en la cadena S de donde tiene lugar 'expresion'. Si 'expresión' no es una subcadena de S, entonces devuelve cero. La posición que devuelve (valor decimal) es la posición donde empieza 'expresión'.

8.3 CONCAT(S1,...,SJ) : STRING

Devuelve una cadena la cual es la concatenación de las cadenas S1 a SJ. No hay restrición en el número J excepto que la cadena resultante debe de ser menor que 256 caracteres.

8.4 COPY(S,INDEX,LENGTH) : STRING

Devuelve una subcadena de S, empezando en la posición indicada por 'index' de 'length' caracteres.

8.5 INSERT(S1,S2,INDEX)

Inserta la cadena S1 dentro de la cadena S2.

Comienza a insertar en la posición indicada por 'index',

8.6 DELETE(S,INDEX,LENGTH)

Borra 'length' caracteres de la cadena S, comenzando en la posición indicada por 'Index'.

9 RUTINAS DE MANIPULACION DE ARRAY DE CARACTERES

Estas rutinas operan con estructuras de packed array of char. Aquí no se realiza 'chequeo' por lo que deben ser usadas con sumo cuidado.

9.1 SCAN(LENGTH,EXPRESION PARCIAL,A) : INTEGER

Examina el array 'a' para un carácter para ver cuál satisface la 'expresión parcial', el cual debe ser de la forma '<>' o '=' seguido por 'carácter' donde 'carácter' es una expresión la cual se evalúa para un valor de carácter. El array 'a' puede ser suscrito para indicar un punto de comienzo; de otra manera el scan comienza al principio de 'a'. El número de caracteres examinados es devuelto. Si el primer carácter satisface la 'expresion parcial', devuelve cero; si el carácter no es sastifecho devuelve 'length'. Si 'length' es negativa, el array es examinado al revés y el valor devuelto será menor o igual que cero.

Si tenemos:

```
CA:='0123456789';
```

Los siguientes ejemplos

```
WRITELN(SCAN(10,='4',CA),SCAN(10,='4',CA 2 ));
WRITELN(SCAN(-10,±'8',CA 9 ),SCAN(-10,='x'CA 9 ));
```

nos daría respectivamente:

```
4  2
-1  -10
```

9.2 MOVELEFT(FUENTE,DESTINO,LONGITUD)
MOVERIGHT(FUENTE,DESTINO,LONGITUD)

Ambas rutinas mueven'longitud' bytes desde el array 'fuente' hacia el array 'destino'; Una o ambas ' fuentes' y 'destino' pueden ser suscritas para indicar una posición de comienzo distinta de la del primer caracter. La rutina moveleft comienza en el final izquierdo de ambos array y copia moviendo hacia la derecha. La rutina moveright comienza en el final derecho de ambos array y copia bytes moviéndose hacia la izqueerda.

9.3 FILLCHAR(A,LONGITUD,CARACTER)

Llena el array 'a' en la longitud indicada por 'longitud' con los caracteres indicados en 'carácter'.

A continuación se muestra una colección de programas en donde se muestra lo más importante de este cápítulo. seguidamente pondremos lo que intenta explicar cada programa.

PROGRAM THREE1.Muestra los procedimientos predeclarados seset,rewrite, get,put, y close.

PROGRAM THREE2. Muestra los procedimientos predeclarados reset, seek, y get.

PROGRAM THREE3. Muestra los procedimientos predeclarados read y readln.

PROGRAM THREE4. Muestra los procedimientos predeclarados new,mark y release.

PROGRAM THREE5. Muestra los procedimientos predeclarados ord y chr.

PROGRAM THREE6. Muestra el procedimiento predeclarado exit.

PROGRAM THREE7. Muestra las rutinas de manipulación de cadenas length,pos,concat,copy,insert, y delete.

PROGRAM THREE8. Muestra las rutinas de nanipulación de array de caracteres scan,moveleft,moveright, y fillchar.

```
program three1;

  var
    i: integer;
    datafile: file of integer;

  begin  ( 3-1 )
    rewrite(datafile,':F1:DDATA.DAT');   ( create DDATA.DAT on drive 1 )
    for i := 0 to 10 do                  ( write 0..10 onto the file )
      begin
        datafile↑ := i;
        put(datafile);
      end;
    reset(datafile);                     ( re-position file to beginning )
    repeat                               ( read in and display the integers )
      writeln(datafile↑:5);
      get(datafile);
    until eof(datafile);
    close(datafile);                     ( and then close the file )
  end.  ( 3-1 )
```

Executing this program will result in the following output:

```
    0
    1
    2
    3
    4
    5
    6
    7
    8
    9
   10
```

```
program three2;

  var
    i: integer;
    datafile: file of integer;

  begin  { 3-2 }
    reset(datafile,':F1:DDATA.DAT');  { open the file DDATA.DAT on drive 1 }
    for i := 10 downto 6 do
      begin
        seek(datafile,i);                          { seek to a record }
        get(datafile);                             { get its value }
        write(output,'Record number:',i:3);        { and display the data }
        write(output,'   Record value:',datafile↑:3);
        writeln(output);
      end;
    close(datafile);
  end.  { 3-2 }
```

(
Executing this program will result in the following output:

```
Record number:  10    Record value:  10
Record number:   9    Record value:   9
Record number:   8    Record value:   8
Record number:   7    Record value:   7
Record number:   6    Record value:   6
```

)

```
program three3;

  var
    a,b,c,y,z: char;
    datafile: text;

  begin  ( 3-3 )
    rewrite(datafile,':F1:TDATA.DAT');   ( create a temporary file on drive 1 )
    writeln(datafile,'AB');              ( write a two character line )
    writeln(datafile,'YZ');              ( and a two character line )
    reset(datafile);                     ( re-position file to the beginning )
    a := '1';                            ( initialize variables )
    b := '2';
    c := '3';
    y := '8';
    z := '9';
    read(datafile,a,b,c,y,z);            ( c := ' ' since at end-of-line )
    writeln(a,b,c,y,z);                  ( display the values )
    reset(datafile);
    a := '1';
    b := '2';
    c := '3';
    y := '8';
    z := '9';
    readln(datafile,a);                  ( read the 'A'; skip to the next line )
    readln(datafile,y);                  ( read the 'Y' and skip to eof )
    writeln(a,b,c,y,z);
    close(datafile);
  end.   ( 3-3 )
```

(
Executing this program will result in the following output:

```
AB YZ
A23Y9
```
)

```
)

program three4;

  type
    intarray = array[0..10] of integer;

  var
    i: integer;
    htop, htop1: ↑integer;
    buffer: ↑intarray;

  begin
    mark(htop);                          { record the top of the HEAP }
    new(buffer);                         { allocate an array of integers from
                                           the top of the HEAP. This moves the
                                           top of the HEAP upwards by 22 bytes. }
    mark(htop1);                         { record the new HEAP top }
    writeln(ord(htop1) - ord(htop));     { display the amount of allocated space }

    for i:=0 to 10 do                    { set the values of the array }
      buffer↑[i] := i*i;

      { process the array }

    release(htop);                       { set the top of the HEAP back to its
                                           original value, which was saved in
                                           htop. }

    { The pointer variable 'buffer' now points to locations above the
      valid area of the HEAP, and must not be used until, using the
      'new' procedure again, it points to a valid array. }

  end.   { 3-4 }


{
Executing this program will result in the following output:

        22
}
```

```
program three5;

  type
    colors = (red,green,yellow,blue);

  begin
    writeln(ord(red):3,ord(green):3,ord(yellow):3,ord(blue):3);
    writeln(chr(ord('A')));
  end.   ( 3-5 )
```

(
Executing this program will result in the following output:

```
  0  1  2  3
A
)
```

```
program three6;

  var
    i: integer;

  procedure alpha;
    forward;
  procedure beta;
    forward;
  procedure gamma;
    forward;

  procedure alpha;
    begin
      beta;
      writeln('Exit alpha');
    end;  { alpha }

  procedure beta;
    begin
      gamma;
      writeln('Exit beta');
    end;  { beta }

  procedure gamma;
    begin
      write('Please enter a digit: ');
      readln(i);
      writeln;
      writeln('Thank you');
      case i of
        1: exit(three6);
        2: exit(alpha);
        3: exit(beta);
        4: exit(gamma);
      end;  { case }
      writeln('Exit gamma');
    end;  { gamma }

  begin
    alpha;
    writeln('Exit program');
  end.  { 3-6 }
```

(Executing this program will result in the following output
  sequences, depending upon the value of <n>:

Please enter a digit: <n>   (<n> is entered by the user)

Thank you	Thank you Exit program	Thank you Exit alpha Exit program	Thank you Exit beta Exit alpha Exit program	Thank you Exit gamma Exit beta Exit alpha Exit program
(<n> = 1)	(<n> = 2)	(<n> = 3)	(<n> = 4)	(<n> >= 5 or <n> <= 0)

)

```
program three7;

  var
    sa,sb,sc,sd: string[70];

  begin
    sa := 'First STRING to be defined';
    sb := 'Second STRING we are defining';
    sc := ' in this example';
    sd := '';                          ( null string )
    writeln(length(sa):3,length(sd):3);
    writeln(pos('nd ',sb):3, pos('XXX',sc):3);
    writeln(concat(sb,sc));
    writeln(copy(sa,7,3));
    insert('Pascal ',sb,8);
    writeln(sb);
    delete(sa,14,6);
    writeln(sa);
  end.   ( 3-7 )
```

Executing this program results in the following output:

```
 26   0
  5   0
Second STRING we are defining in this example
STR
Second Pascal STRING we are defining
First STRING defined
```

```
program three8;

  var
    ca,cb: packed array[0..9] of char;
    sa: string[30];

  begin
    ca := '0123456789';
    writeln(scan(10,='4',ca):3, scan(10,='4',ca[2]):4);
    writeln(scan(10,='X',ca):3, scan(5, ='X',ca):4);
    writeln(scan(-10,='8',ca[9]):3, scan(-10,='X',ca[9]):4);

    moveleft(ca,cb,sizeof(ca));      ( copy ca to cb both ways )
    writeln(cb);
    moveright(ca,cb,sizeof(ca));
    writeln(cb);
    moveleft(ca,ca[1],9);            ( the incorrct way to move
                                       characters up by one position )
    writeln(ca);
    moveright(cb,cb[1],9);         ( the correct way )
    writeln(cb);

    sa := 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    writeln(sa);
    fillchar(sa[1],25,'0');          ( a string is a packed array of char )
    writeln(sa);
  end.   ( 3-8 )
```

(
Executing this program results in the following output:

```
  4   2
 10   5
 -1 -10
0123456789
0123456789
0000000000
0012345678
ABCDEFGHIJKLMNOPQRSTUVWXYZ
00000000000000000000000000Z
```

)

INSTRUCCIONES DE OPERACION

INSTRUCCIONES DE OPERACION

El Pascal-80 del sistema de desarrollo está compuesto de un sistema de Run-Time (Run-Time System (RTS)) y un compilador, que es el que se encarga de compilar el programa en Pascal una vez que ha sido cargado el RTS. Para cargar el RTS hay que poner la palabra Pascal. El compilador del Pascal-80 lo que hace es convertir el programa fuente en pascal en un código intermedio, el cual es el que interpreta el RTS.

El editor de texto bajo el control del Isis II lo empleamos para crearun fichero en el cual va a estar nuestro programa fuente en pascal. Una vez que hayamos realizado la compilación, el compilador nos crea un listado de ese programa para, en el caso de que haya error durante la compilación, saber dónde se ha producido; El listado del programa lo podemos sacar en pantalla poniendo el nombre del fichero seguido de .LST (un punto seguido de la palabra LST).

1 MODO DE EMPEZAR A TRABAJAR CON EL PASCAL-80

Cada usuario está provisto con todo el software nesesario para que el sistema de desarrollo pueda funcionar bajo el control del Pascal-80. Este software está contenido enuno de estos dos discos (simple o doble densidad); Hemos de tener en cuenta el utilizar el disco apropiado para nuestro hardware.

Cada disco de Pascal-80 viene con el siguiente software:

COMPILER	RTS	UTILES	LIBRERIA	fIGURAS	EJEMPLOS
COMP.COD	PASCAL	JOIN.COD	P8ORUN.LIB	FIG31.PAS	EX.PAS
	PASCAL.RFS	GENOBJ.COD	P8OISS.LIB	FIG32.PAS	BUFFER.PAS
			P8ORAR.LIB	FIG33.PAS	SEEKEX.PAS
			P8OEXT.LIB	FIG34.PAS	ERROR.PAS
			P8OISS.PLB	FIG35.PAS	PEOPLE.DAT
				FIG36.pAS	
				FIG37.PAS	
				FIG38.PAS	

1.1 PASOS INICIALES

Antes de empezar a trabajar con el Pascal-80 es bastante conveniente seguir los siguientes pasos que se muestran a continuación.

a) Coger un disco en blanco y formatearlo utilizando el comando IDISK (ha de ser un disco de sistema).

b) Copiar todos los ficheros de la figura de encima sobre el disco de sistema Isis II para así poder crear un disco de sistema de Pascal-80.

c) Si vamos a correr un programa fuera del disco de sistema del Isis II, notar que para poderlo correr en Pascal 80 se necesita los siguientes ficheros.

ISIS.DIR ISIS.LAB

ISIS.MAP ISIS.BIN

ISIS.TO ISIS.CLI

d) Para mayores aplicaciones, los siguientes ficheros sobre el disco de sistema Isis II son también usados pero no requeridos para hacer correr a un programa bajo el Pascal-80

DIR	COPY
DELETE	RENAME
ATTRIB	SUBMIT

e) Es bastante recomendable que los ficheros del Pascal-80 sean protegidos contra escritura; Para ello empleamos el comando ATTRIB del Isis II. La manera de usar este comando es poner 'attrib seguido del nombre del fichero a proteger y seguido de W1 que significa protegido contra escritura. Los ficheros que debemos proteger principalmente son:

PASCAL PASCAL.RES COMP.COD

1.2 COMPILADOR

En caso de tener varios 'driver' (no es nuestro caso) el compilador del Pascal-80 se carga del 'driver' que contiene el COMP.COD

1.3 RUN-TIME-SYSTEM (RTS)

Ejecuta el código intermedio fuera del compilador. Este código está contenido en los ficheros Pascal y Pascal.Res

1.4 LIBRERIA

P8ORUN.LIB,P8OISS.LIB y P8ORAR.LIB son módulos de librería del Isis II usados en la generación de carga e ida (load and go) de los programas en Pascal.

1.5 PROGRAMAS EJEMPLOS

EX.PAS,BUFFER.PAS,ERROR.PAS Y SEEKEX.PAS son simples demostraciones de programas, suministrados en forma fuente, la cual muestra muchas de las facilidades presentes en el Pascal-80 y las cuales pueden ser usadas con aquella persona que se inicia en el Pascal-80 para que se familiarice con el sistema. SEEKEX.PAS es el programa más completo.

1.6 PROGRAMAS FIGURAS

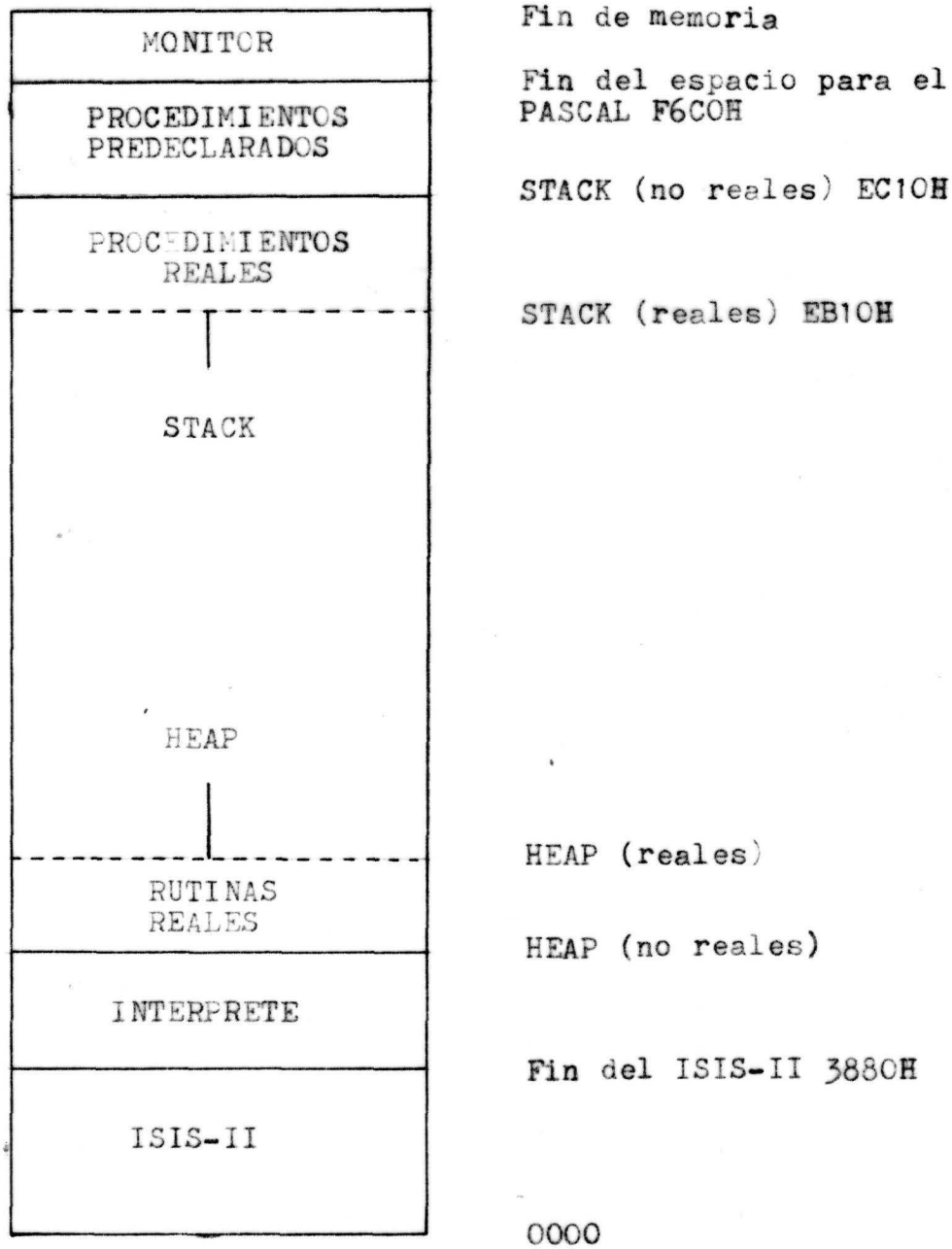
Cada uno de los programas del capítulo anterior están suministrados en forma fuente, así que ellos pueden ser compilados por el usuario. Estos ficheros están nombrados como FIG3N.PAS, donde N es un dígito que va desde 1 a 8.

1.7 DISTRUBUCION DE LA MEMORIA PARA EL RTS

Es RTS del Pascal-80 está compuesto de un intérprete y un número de funciones y procedimientos predeclarados escritos en Pascal. Durante la ejecución de la interpretación, hay dos estructuras dinámicas de

datos mayores: el STACK y el HEAP. El STACK contiene todo el código intermedio y las estructuras de datos; También tiene todas las variables dinámicas de colocación del HEAP.

La distribucion de la memoria principal para un sistema de 64 Kbytes es:



2 OPERACIONES CON EL SISTEMA PASCAL-80

2.1 LLAMADA DEL PASCAL-80

Para llamar al sistema del Pascal-80 desde el Isis II, hay que poner lo siguiente.

PASCAL [DIRECTIVO] [NOMBRE DEL FICHERO CODIGO [OPCION]]

El directivo es un campo opcional, el cual, cuando está presente, hace que el sistema del Pascal-80 sea cargado sin el código para los números reales. La forma de esta opción es:

-R

Esta opción es usada para aumentar la memoria disponible para cuando el programa que vamos ha hacer no utilice número reales. Con la opción -R hay unos 2800 bytes adicionales disponibles . Por ejemplo:

```
PASCAL -R
PASCAL -80Vv.r[No reales]
>
```

El nombre del fichero código es también un campo opcional el cual, cuando está presente, tiene en cuenta una ejecución automática del programa Pascal. Este hecho será discutido en la sección de ejecución de programas automáticamente.

En el caso de que no haya nigún campo opcional, el Pascal-80 responde con:

```
PASCAL-80Vv.r
>
```

2.2 COMANDO DE SINTAXIS DE LINEA PARA EL PASCAL

Cuando el RTS del Pascal-80 es llamado, responde con el caracter '>' como hemos visto en el apartado anterior,lo cual indica que está esperando por un comando el cual se difine como sigue:

COMANDO ::= COMANDO DE FICHERO / COMANDO DIRECTO

COMANDO DE FICHERO ::= NOMBRE DEL FICHERO CODIGO [OPCION]

COMANDO DIRECTO ::= TRACEON/TRACEOFF/QUIT/SIZEON/ SIZEOFF/STATS

El 'nombre del fichero codigo' especifica un compilado del programa pascal el cual será cargado por el RTS y ejecutado.

El significado del campo 'opcion' es determinado por el programa que está siendo ejecutado.

Los próximos dos comandos, 'traceon' y 'traceoff' son usados para poder seguir la traza de un programa en pascal.

El comando 'quit' devuelve el control al comando de línea del interprete del Isis II

Los comandos 'sizeon' y 'sizeoff' son usados para controlar el tamaño del STACK y del HEAP el el monitor del RTS.

El comando 'stats' detiene un cierto 'run-time' para poder ser visualizado.

Ejemplos de comandos válidos son los siguientes:

```
> COMP TEST.PAS
> TEST
>TRACENN
>TRACEOFF
>QUIT
```

El primer comando compilará el fichero TEST.PAS que es donde se encuentra el programa fuente en Pascal. El segundo comando ejecutará el TEST.COD que es el programa en Pascal objeto, pero esto implica que previamente la compilación ha sido realizada. El tercer comando conecta el flag TRACE que es el que nos permite llevar una traza de nuestro programa. El cuarto comando desconecta el flag TRACE, mientras el quinto devuelve el control al Isis II.

2.2.1 ESPECIFICACION DEL NOMBRE DEL FICHERO CODIGO

El nombre de un fichero código puede ser expresado de las siguientes formas:

TIPO DE NOMBRE	NOMBRE USADO
NOMBRE_FICHERO.EXT	NOMBRE_FICHERO.EXT
NOMBRE_FICHERO	NOMBRE_FICHERO.COD
NOMVRE_FICHERO.	NOMBRE_FICHERO

2.3 VENTAJAS PARA PROGRAMAS QUE EMPLEAN TRAZO

El sistema Pascal-80 incorpora ventajas para programas que emplean trazo, teniendo en cuenta que dicho trazo muestra la ejecución (cómo está siendo ejecutado) del programa. Cuando el flag de TRACE está puesto, el número de línea de cada instucción que está siendo ejecutada es sacado por :co: (pantalla de CRT), encerrado entre corchetes. Usando esta información junto con el listado del compilador, el cual asocia número de línea con las declaraciones del programa, un programador puede saber mucho más fácil lo que el programa está haciendo.

2.3.1 EL FLAG TRACE

El flag TRACE puede ser manipulado de dos maneras; El comando TRACEON, el cual conecta el flag y el TRACEOFF, el cual desconecta el flag. Adicionalmente cada vez que el interruptor de la interrupción 4 sobre el sistema de desarrollo sea presionado, hará que el flag TRACE se conecte si estaba desconectado, o se desconecte si estaba conectado. Esto permite selecionar aquellos trozos de programas donde estén las preguntas, y ejecutar el resto normalmente.

2.3.2 INSTRUCCIONES DE TRAZO

Para que el sistema Pascal-80 pueda llevar el trazo de un programa, o parte de un programa, las

instrucciones de trazo deben ser presentadas en el código del compilador. El compilador del Pascal-80 inserta estas instrucciones normalmente en el código objeto, pero estas instrucciones pueden ser omitidas usando el directivo del compilador {ST-}. Cuando el compilador encuentre este directivo, no ejecutará ninguna instrucción de trazo en el código posterior que está siendo generado. Esta situación durará hasta que se encuentre el directivo {ST+} que es cuando las instrucciones de trazo se empiezan a generar otra vez. Sin embargo, usando estos directivos, un programa entero o siplemente parte de un programa, puede contener instrucciones de trazo.

2.4 VISUALIZACION DEL RUN-TIME

El sistema de 'run-time' del Pascal-80 tiene la capacidad de visualizar continuamente el tamaño dinámico del STACK y HEAP. Desde que ocurre una degradación en la ejecución cuando se está visualizando, el usuario podría conectar y desconestar esta visualización con los comandos SIZEON y SIZEOFF. Cuando se está llevando a cabo un ajuste de visualización, el sistema mantiene la pista de tres valores, los cuales varian cuando se hace correr un programa. Estos tres valores son:

- El tamaño máximo de STACK a cualquier hora.
- El tamaño máximo del HEAP a cualquier hora.
- El tamaño máximo de combinación del STACK y HEAP a

cualquier hora.

Estos valores son inicializados a cero cuando el programa comienza su ejecución, y son guardados cuando regresamos al comando de intérprete de línea de el Pascal. El tamaño del STACK no incluye el espacio consumido por el código residente del programa Pascal, pero sí incluye el espacio ocupado por cada construcción de datos, incluyendo la construcción de datos globales de un programa y también la longitud de algún procedimiento SEGMENT.

El comando STATS visualiza esos valores guardados. Estos valores guardados son útiles cuando se genera una versión de 'load and go' de un programa Pascal, o cuando se construyen un RMX/80 PASCAL, ya que la combinación de STACK yHEAP nos da el máximo espacio libre nesesitado por el programa. Notar, sin embargo, que ese espacio libre sólo es válido cuando el programa tiene características de ejecución similares a aquéllas durante el momento actual de visualización, y no debería ser aceptado como el máximo espacio libre que el programa necesitará siempre. Si el programa tiene funciones y procedimientos declarados, los cuales son recursives, el número de niveles de recursión es un factor que contribuye en el tamaño del STACK, y sin embargo cada invocación o llamada a el programa requerirá una cantidad diferente dependiendo del nivel de recursión alcanzado. También

la secuencia particular de llamada o regreso a procedimiento, así como el modelo de localización variable del HEAP, determina el espacio libre usado durante alguna especificación de llamada de un programa.

2.5 VISUALIZACION DEL STATS

La ejecución del comando STATS causará la visualización de ciertas variables del 'run-time'. El tamaño de las variables de 'monitoring' (monitor) , descrita en la partado anterior, así como el espacio libre (entre la parte alta del HEAP y la parte baja del STACK) es tambien visualizada. El formato del comando STATS es como sigue:

```
>STATS
  Interpreter versión: Vx.y
  Available Free space: size-1
  Maximum HEAP SIZE: size-2
  Maximum STACK SIZE: size-3
  Maximum Combined SIZE: size-4
  Run-time Monitoring: flag-value-1
  Run-time Tracing: flag-value-2
```

Donde:

x.y Es la versión y revisión del intérprete del Pascal

size-1. Nos da el número de bytes disponibles para los programas que nosotros queramos hacer.

size-2,size-3,size-4. Nos da el tamaño de las variables del monitor.

flag-value-1,flag-value-2. Es el estado de cómo se encuentran los parámetros asociados al 'run-time'. Los posibles valores de estos parámetros son ON y OFF.

2.6 EJECUCION AUTOMATICA DE UN PROGRAMA

Si un fichero código está especificado en un comando de línea del Isis II, cuando llamemos al Pascal, el sistema Pascal-80 es cargado y el fichero código es compilado sin más entrada por parte del usuario. Por ejemplo si ponemos:

```
PASCAL COMP EX.PAS
```

aparecerá lo siguiente en la pantalla

```
PASCAL-80 Compiler Vv.r
COMPILING EX.PAS
Symbol table space remaining: 12234 bytes
37 lined compiled
PASCAL-80 Vv.r
>
```

2.7 INTERRUPCION DE UN PROGRAMA EN EJECUCION

Cuando apretamos la interrupción 3 sobre el sistema de desarrollo la ejecución del programa Pascal cesará y el control será devuelto al sistema Pascal-80. Un mensaje de error será visualizado en el segmento, procedimiento o instrucción donde fue suspendida su ejecución.

3 OPERANDO CON EL COMPILADOR DEL PASCAL-80

3.1 SINTAXIS DE LOS COMANDOS DE LINEA PARA EL COMPILADOR

La sintaxis del comando de línea para compilar un programa Pascal es como sigue:

COMP NOMBRE_DEL_FICHERO DIRECTIVO

donde 'nombre_del_fichero' es el nombre del fichero en código fuente. Si no se pone el nombre del fichero se producirá un mensaje de error,devolviendo el control al sistema del Pascal-80.

El fichero fuente puede ser especificado de las siguientes formas:

NOMBRE ESPECIFICADO	NOMBRE USADO
NOMBRE_DEL_FICHERO.EXT	NOMBRE_DEL_FICHERO.EXT
NOMBRE_DEL_FICHERO	NOMBRE_DEL_FICHERO.PAS
NOMBRE_DEL_FICHERO.	NOMBRE_DEL_FICHERO

El 'directivo' son una serie de directivos de compilación opcionales. Cada directivo debe de ser separado de los siguientes por uno o dos espacios.

Si un comando COMP es más largo que una línea de nuestra pantalla (el cual no debe de ser mayor de 122 caracteres), podemos continuar en la línea siguiente pero antes de presionar RETURN debemos poner un 'ampersand' (&). El '&' no puede aparecer con el nombre de un directivo ni con el nombre de un fichero.

COMP crea un fichero de trabajo llamado P8OWRK.TMP. Si nosotros tenemos un fichero con este nombre, éste será destruido, por lo que debemos tener sumo cuidado.

3.2 DIRECTIVOS DE COMPILACION

El sistema Pascal-80 reconece varios directivos, los cuales son usados para controlar varias fases y detalles en un proceso de compilación. Estos directivos se dividen en dos clases. La primera clase son directivos de 'comando de linea', los cuales se ponen después del nombre del fichero fuente antes de que sea compilado. La segunda clase de directivos son los llamados directivos encajados o introducidos, los cuales se encuentran en el texto del programa que está siendo compilado. Los directivos de 'comando de línea' serán descritos a continuación.

3.3 DIRECTIVOS DE COMANDOS DE LINEA PARA EL COMPILADOR

Una descripción de los directivos de comando de línea disponible es:

- NOLIST. No lista el fichero producido.
- NOCODE. El fichero producido no presenta código intermedio.
- ERRLIST. El listado está limitado a aquellas líneas que contienen errores de sintaxis.

- LIST (external-file-name). Fspecifica el ficnero hacia el cuál el listado está siendo dirigido. Si hay fallo,el defecto se corrige con: source-filename.LST (nombre del fichero fuente seguido de .LST).

- CODE (external-file-name). Especifica el fichero hacia el cual el código está siendo dirigido. El defecto se corrige con: source-filename.COD

- NOECHO. Los errores de línea son repetidos sobre la pantalla,al menos que se especifique este directivo.

- GLOBAL (external-file-name). Especifica un fichero el cual contiene una tabla global de símbolo cuando un programa Pascal se compila separadamente por parte. El fallo se corrige con: source-filename.SYM

- WORKFILE (device). Especifida que diskete va a ser usado para la compilación del fichero de trabajo P8OWRK. TMP (Esto se usa para cuando tenemos más de un driver). El fallo es el dispositivo de salida del fichero código.

- DATE (date). Especifica los datos a ser incluidos en el encabezamiento de una página cuando estamos realizando un listado de compilación. El 'date' es una secuencia de nueve caracteres no conteniendo parêntesis. El defecto se puede corregir poniendo todo espacios.

- ETAB (external-file-name). Especifica un fichero el cual contiene una tabla externa cuando compilamos un programa Pascal con referencias esternas. El defecto se puede corregir con: source-filename.ERT

- NOSTATISTICS. Especifica que el compilador no debería acumular o listar la disposición de los datos de procedimientos ni el tamaño de los parámetros.

3.3.1 COMBINACIONES EXCLUIDAS

Ciertos directivos vistos anteriormente no pueden ser usados en combinación. La siguiente tabla muestra qué directivos son excluido si el directivo de la izquierda está especificado.

DIRECTIVO	DIRECTIVO EXCLUIDO
NOLIST	LIST
	ERRLIST
NOCODE	CODE

3.4 DIRECTIVOS ENCAJADOS

Los directivos encajados están insertos en el código fuente y tienen la siguiente forma:

{$directive [directive]} o (*$directive [directive]*)

Muchos de los directivos tienen la forma:

directive+ o directive-

Se el + o el - no están presente, el + es supuesto.

A continuación veremos la especificación de cada directivo.

- C. Este directivo hace que el texto seguido después de C sea puesto en el fichero código. El directivo debe aparecer en lo alto del programa ; De otra manera

el compilador dará error.

- I. Cuando este directivo está seguido por + o - , el chequeo de las I/O (entrada/salida) es afectado como sigue:

a) I+.Hace que el compilador genere un código después de cada sentencia de I/O para poder chequear si las I/O se realizaron bien. En caso que las I/O no se hayan realizado perfectamente,un error del 'run-time' ocurrirá.

b) I-. Hace que el compilador genere un código para que no se produzca chequeo de I/O.

- R. Este directivo afecta al chequeo del rango de el compilador. Si R está seguido de +, el código del chequeo del rango es generado por el compilador, de tal manera que si se produce un error de rango mientra el programa se está ejecutando, un error del 'run-time' tendrá lugar. Cuando R está seguido por - no se produce código para el chequeo del rango.

- O. Este directivo indica con qué memoria está trabajando el compilador. Si tenemos O+ indica que el compilador está trabajando con toda la memoria. Si tenemos O- indica que el compilador no trabaja con toda la menoria . Con está ultima forma (O-) disponemos de 6300 bytes menos de memoria para la tabla de símblo,pero el tiempo de compilación se reduce ya que el compilador no tiene que estar leyendo tan

frecuentemente como si estuviera en el modo O+. El modo O- es útil para la compilación de programas pequeños o medianos.

- T. Este directivo determina si el compilador genera instrucciones de traza, y puede aparecer en el código fuente. Si tenemos T+ las instrucciones de traza son insertadas en el código objeto. Si tenemos T- no se producen instrucciones de traza.

- H. Este directivo especifica un encabezamiento para ser impreso en la segunda línea de cada página del listado de un programa, justo debajo de la línea del título del programa.

3.5 SUMARIO DE INFORMACION SOBRE LA PANTALLA

La forma general de un comando de línea es:

COMP nombre_del_fichero [directivo] .

El compilador responderá con:

PASCAL-80 Compiler Vv,r

Cada directivo es luego reconocido por el compilador en líneas separadas contestando ACCEPTED o REJESTED (aceptado o rechazado). Después que todos los directivos fueron reconocidos y ninguno rechazado, abre su fichero y comienza la compilación. Si algún directivo fuera rechazado, se visualizaría:

** Compilation terminated **

y el control es devuelto al sistema Pascal-80.

Si todos los directivos fueran aceptados se visualizaría el mensaje:

COMPILING nombre_del_fichero

Si se produce algún fallo en la abertura de algún fichero, se visualizaría:

nombre_del_fichero failed to open

Cuando la compilación es terminada el control es devuelto al sistema Pascal-80 visualizando previamente:

Symbol table space remaining:nnnnnbytes

mmmmmlined compiled

donde nnnnn es el número de bytes de memoria que no fueron usados durante la compilación del programa. Estos nos dicen cuantos símbolos más podemos poner en la tabla de espacio para el programa en uso. Las mmmmm son el número de líneas en el programa.

3.6 FORMATO DEL LISTADO DEL COMPILADOR

El formato general de salida es como sigue:

```
PASCAL-80Compiler Vv.r      filename  date  page:nnn
   Line  Seg  Proc  Lev  Disp
   nnnn  nn   nnn    n   nnnn   Sentencia 1
    .
    .
   nnnn  nn   nnn    n   nnnn   Sentencia n
```

Las dos primeras líneas serían el encabezamiento, el cual aparecerá en todas las páginas. 'Line' es el nú

mero de línea. 'Seg' es el número de segmento (es cuando tenemos un programa dividido en segmentos para compilarlo por separado cada segmento. Si no hacemos lo anterior siempre aparecerá un 1). 'Proc' es el número de . procedimiento. 'lev' es el nivel de alojamiento. 'Disp' es el cambio de situación o estamento en el procedimiento.
Estos números son utilizados en el 'debugging' (depurado) ya que,cuando ocurre algún error, el número de segmento, el número de procedimiento y la instrucción son visualizados.

3.7 COMPILACION INICIAL

Cuando desarrollamos un programa Pascal, frecuentemente se desea un chequeo rápido de la sintaxis y la semántica. Compilando un programa con NOSTATISTICS NOLIST y NOCODE como comando de línea se ejecurarán estos chequeos en un tiempo mínimo.

COMPILACION SEPARADA, UNION DE PROGRAMAS RELOCALIZACION Y EJECUCION

COMPILACION SEPARADA, UNION DE PROGRAMAS RELOCALIZACION Y EJECUCION

Este capítulo describe las técnicas disponibles para partir un programa largo en módulos, los cuales pueden ser compilados independientemente; también describe cómo unir programas en Pascal-80 con módulos externos escritos en PL/M, FORTRAN-80, o ASM-80, así como la construcción de un 'load-and-go' (carga y ejecución) de nuestro programa Pascal, el cual es directamente ejecutable a través de los comandos de línea del Isis II.

1 PARTICION DE UN PROGRAMA PASCAL

Para poder desarrollar un programa Pascal largo, hemos de tener en cuenta que un programa puede ser separados en COMPONENTES o unidades de compilación. Esos COMPONENTES luego son compilados separadamente y combinados con el JOIN, el cual tiene la ventaja para producir el fichero código final.

Ya que los COMPONENTES son compilados separadamente, un cambio en uno de ellos requiere que ese sólo COMPONENTE sea recompilado, no haciendo falta la recompilación de todo el programa. Esta recompilación es seguida luego por el procedimiento JOIN. El programa entero debe ser recompilado sólo si los datos globales en el programa son alterados.

1.1 ESTRUCTURA DE UN PROGRAMA PARTIDO

La estructura de un programa partido consiste en un programa principal, el cual tiene que estar especificado con la palabra reservada PARTITIONED, y de uno o más COMPONENTES separados. El programa principal PARTITIONED contiene los datos globales y las declaraciones del bloque de salida, así como todos los procedimientos globales residentes. Cada COMPONENTE contiene uno o más segmentos de procedimientos (segment procedure) globales. Cada uno de los COMPONENTES, así como el programa principal PARTITIONED, es una unidad de compilación y, por lo tanto, es compilado separadamente.

Cada segmento de procedimiento global (segment procedure) contenido en un COMPONENTE debe ser declarado en en programa principal como un segmento de procedimiento separado (segment procedure SEPARATE) , como sigue:

```
SEGMENT PROCEDURE ANT;
SEPARATE;
```

La sintaxis de un programa PARTITIONED es como sigue:

Un programa PARTITIONED en un encabezamiento y un bloque.

El encabezamiento lo tendremos que poner como:

```
PARTITIONED PROGRAM IDENTIFICADOR;
```

El bloque estaría formado por:

1) Declaración de etiquetas.

2) Definición de constantes.

3) Definición de tipos.

4) Declaración de variables.

5) Declaración de las funciones y procedimientos separate.

6) Conjunto de instrucciones.

La sintaxis de un procedimiento o función SEPARATE es como sigue:

Para un procedimiento sería:

SEGMENT PROCEDURE nombre_del_procedimiento;

SEPARATE;

Para una función sería:

SEGMENT FUNCTION nombre_de_la_función;

SEPARATE;

La sintaxis de un COMPONENTE es como sigue:

COMPONENT identificador;

SEGMENT PROCEDURE/FUNCTION nombre_proced./func.;

Sentencias que componen el procedimiento/función del segment. Después de todas las sentencias hay que poner:

BEGIN

END.

A continuación veremos un programa ejemplo donde se verá todo lo explicado .

```
{ Main Program Compilation Unit - Source on SIX1.PAM }

partitioned program six1;
  const header = 'This is procedure ';
  type procname = string[10];
  var level: integer;
  segment procedure aa;
    separate;
  segment procedure bb;
    separate;
  segment procedure cc;
    separate;
  function increment(i: integer): integer;
    begin
      writeln('    level: ',i:3);
      increment := i + 1;
    end; { increment }

  begin { six1 }
    level := 1;
    write('This is the main program');
    level := increment(level);
    aa;
  end. { six1 }

{ Component 1 Compilation Unit - Source on SIX1.PA1 }

component six1c1;
  segment procedure aa;
    var name: procname;
    begin
      name := 'aa';
      write(header,name);
      level := increment(level);
      bb;
    end; { aa }

  begin
  end. { six1c1 }

{ Component 2 Compilation Unit - Source on SIX1.PA2 }

component six1c2;
  segment procedure bb;
    begin
      write(header,'bb');
      level := increment(level);
      cc;
    end; { bb }
  segment procedure cc;
    begin
      write(header,'cc');
      level := increment(level);
    end; { cc }

  begin
  end. { six1c2 }
```

Como se ve en este ejemplo, tres 'segment procedure' han sido alejados del programa principal y se han usado en los 'component'. El programa completo está compuesto de un 'partitioned program',six1, y de dos 'component', six1c1 y six1c2. La compilación de cada una de estas unidades se encuentran en ficheros independientes; así el 'partitioned program six1' está en el fichero SIX1.PAM, el 'component six1c1' está en el fichero SIX1,PA1, y el 'component six1c2' está en SIX1.PA2.

Las estructuras de un PARTITIONED PROGRAM así como la de un COMPONENT son similares a la de un programa standard. Las principales diferencias están descritas a continuación.

Un PARTITIONED PROGRAM tiene las siguientes características.

- La palabra reservada PARTITIONED en el encabezamiento.

- La presencia de una o más funciones y procedimientos SEPARATE.

Un COMPONENT tiene las siguientes características:

- La palabra reservada COMPONENT, la cual reemplaza a PROGRAM en el encabezamiento.

- Las etiquetas, constantes y tipos de variables, no pueden ser declaradas dentro del COMPONENT. Deben ser declarados en el programa principal PARTITIONED.

- No hay bloque de sentencias de salida; después del BEGIN sigue el END.

- Cada función y procedimiento SEPARATE deben ser declarada como un SEGMENT en el nivel global.

- Cada función y procedimiento SEPARATE puede hacer referencia a constantes, tipos, variables, procedimientos y funciones.

1.2 CONSTRUCCION DE UN FICHERO CODIGO

La ejecución final del fichero código está construida con los ficheros códigos de todas las unidades de compilación. El programa principal PARTITIONED y cada uno de los COMPONENT son compilados separadamente. El programa PARTITIONED debe ser compilado antes de que nigún COMPONENT pueda ser compilado.

Durante la compilación del programa PARTITIONED, el compilador escribe una tabla de símbolos globales sobre un fichero. Este fichero de símbolos globales es luego leído durante la compilación de cada uno de los COMPONENTS. Por esta razón, cada una de las funciones y procedimientos SEPARATE es declarado en el programa principal PARTITIONED.

El directivo compilador GLOBAL(file_name) permite que el nombre de la tabla de símbolos globales sea especificada. En ausencia de este directivo, el compilador usa el file_name seguido de .SYM (punto seguido

SYM).

file_name-fuente.SYM

Por ejemplo si SIX1.PAM es un programa PARTITIONED, el comando:

COMP SIX1.PAMGLOBAL(MYSYM.XYZ)

hace que la tabla de símblos globales sea escrita sobre MYSYM.XYZ, mientras que el comando:

COMP SIX1.PAM,

hace que la tabla de símbolos globales sea puesta sobre el fichero SIX1.SYM

Después de que todas las unidades de compilación de un programa PARTITIONED han sido compiladas correctamente, se emplea el comando JOIN para combinar todos los ficheros código final para que pueda ser ejecutato como todo un bloque.

El formato del comando JOIN ES:

JOIN code-file {,code-file} TO code-file

Por ejemplo, el siguiente comando combinará SIX1.COM, SIX1.CO1 y SIX1.CO2 en SIX1.COD

JOIN SIX1.COM,SIX1.CO1,SIX1.CO2 TO SIX1.COD

La secuencia de comandos mostrada en la figura de a continuación compilará cada una de las unidades de compilación del programa visto dos páginas atrás para luego combinar el resultado de los ficheros código en un fichero código ejecutable. El fichero SIX1.SYM se usa como el fichero que contiene la tabla de

de símbolos globales, ya que ninguno de los comando de compilación especifica un directivo GLOBAL. Para entender esto facilmente, el disco que empleamos contiene los tres ficheros fuentes SIX1.PAM,SIX1.PA1 y SIX1.PA2, así como el fichero SUBMIT que contiene todos los comandos de la figura que vamos a poner

```
PASCAL
COMP SIX1.PAM CODE(SIX1.COM)
COMP SIX1.PA1 CODE(SIX1.CO1)
COMP SIX1.PA2 CODE(SIX1.CO2)
JOIB SIX1.COM,SIX1.CO1,SIX1.CO2 TO SIX1.COD
QUIT
```

2 UNION DE MODULOS NO PASCAL

Para más aplicación,puede ser necesario o deseable el poder codificar una o más funciones y procedimientos en el lenguaje distinto del Pascal. Estas funciones y procedimientos externos pueden ser escritos en PL/M-80, FORTRAN-80, y 8080/8085 ensamblador y traducidos por el procesador de lenguaje apropiado.
Los resultados establecido en módulos objetos pueden ser unidos, localizado y luego ejecutado como una parte integral de un programa Pascal-80.

PROCEDIMIENTOS PREDECLARADOS

2.1 EXTENSION DEL PASCAL-80

Para podernos comunicar con un módulo externo, un programa Pascal debería poder especificar qué variables, procedimientos y/o funciones son de hecho externas a la compilación. Estas especificaciones tienen dos propósitos. Primero, declarar objetos de tal manera que un programa Pascal los pueda manipular como si de ellos fueran. Segundo: la especificación causa referencias externas para ser generada, para que el comando LINK pueda combinar estos módulos externos con el programa Pascal.

Un programa Pascal-80 declara variables externas, funciones y procedimientos en una grabación de declaración pública (public declaration record). La sintaxis de un PDR es como sigue:

```
PUBLIC  nombre ;
    Declaración de constantes;
    Declaración de tipos;
    Declaración de variables;
    Funciones y procedimientos;
END;
```

Un ejemplo de PDR sería:

```
PUBLIC asum;
  TYPE rango=0..1000;
  VAR aresult: integer;
  FUNCTION asum(a,b,c:rango):integer;
END;
```

Todos los términos de un PDR están definidos utilizando la sintaxis de un programa Pascal. La única diferencia entre una declaración normal y una declaración con PDR es que la declaración normal causa un almacenamiento para ser asignado dentro del programa Pascal, mientras que la declaración con PDR necesita una asignación externa al programa para poder acceder al lugar de almacenamiento del programa Pascal.

En todas las otras formas, un programa Pascal-80 trata variables particulares y variables externas identicamente.

Hay que tener en cuenta que es responsabilidad del programador el asegurar que la estructura y tamaño de una variable externa esté conforme con la declaración de la variable dentro del PDR. Si esto no es así, ocurre que obtenemos resultados impredecibles.

El 'nombre' es un campo opcional que, si está presente, es aceptado por el compilador y si no, es ignorado.

Ya que la declaración de constantes y la declaración de tipos no causa nigún almacenamiento para ser asignados no están explícitamente asociados con ninguna localización externa de almacenamiento. Sin embargo, es necesario que, dentro de un PDR las variables, procedimientos y funciones, estén definidas adecuadamente.

La declaración de variables declara variables cuyo almacenamiento está asignado externamente a un progra

ma Pascal. El compilador asigna una marca indirecta de referencia para cada una de tales variables externas . Una marca indirecta de referencia es una palabra (en longitud) la cual se usa para direccionar la variable externa.

Los procedimientos y funciones declaran procedimientos y funciones, las cuales están definidas y asignadas externamente a el programa Pascal. El compilador asigna para cada uno de tales procedimientos y funciones una marca indirecta de referencia.

La estructura general de un programa, el cual accede a objetos externos, es como sigue:

```
PROGRAM nombre;
    Declaración de PUBLIC;
    Declaración de etiquetas;
    Declaración de constantes;
    Declarqción de tipos;
    Declaración de variables;
    Declaración de funciones y procedimientos;
    Sentencias del programa;
```

La declaración de un PUBLIC la vimos en dos páginas atrás.

Un ejemplo de uno de tales programas fué el que mostramos al principio de este capítulo. Este programa Pascal hace referencia a dos módulos externos ASUM y PSUM, en dónde cada uno declara una variable entera y una función

entera con tres parámetros de tipo rango. El módulo ASUM está escrito en lenguaje ensamblador 8080/8085 y el módulo PSUM está escrito en PL/M-80. Cada una de las funciones externas simplemente ejecuta la suma de sus tres parámetros, almacena la suma en una variable PUBLIC, y luego devuelve el resultado como valor de la función. Como se muestra en este ejemplo, el módulo codificado en PL/M no debe ser un módulo principal.

Notar que ahí puede haber multiples PDR, pero todas ellas deben aparecer antes que nigún objeto particular esté declarado. Esta restricción hace que todo los objetos externos estén en el nivel de un programa Pascal (global environment). Como para todos los identificadores globales, el nombre de cada variable externa, procedimiento, y función deben ser diferentes de los otros nombres externos y del nombre de cada variable en particular. Si la condición no es satisfecha (por ejemplo dos nombres externos iguales, un nombre externo igual a una variable global), el compilador dará un error 101 que significa identificador declarado dos veces, y no aceptará el nombre duplicado.

Sin embargo, es posible que dos PDR separados puedan contener declaración para variables del mismo tipo. Si este tipo no es un tipo de dato predeclarado, estos PDR duplicarán el tipo de declaración. Para tener en cuenta esta situación, y no causar un tipo duplicado o un

error de declaración de constante, el compilador permitirá tipos y constantes para ser redeclaradas en PDR si la nueva declaración es idéntica a la original.

El ejemplo que hemos visto contiene un tipo, rango, el cual ha sido declarado en un PDR y luego redeclarado en otro. Esta situación es necesaria, ya que cada módulo debe declarar el tipo rango independientemente de nigún otro módulo.

Un PDR puede ser incluido en un fichero fuente, usando la construcción {$Inombre_del_fichero} , de la misma forma que un texto en Pascal.

2.2 LLAMANDO A UNA FUNCOIN O PROCEDIMIENTO EXTERNO

Como fue discutido antes excepto para la localización física del objeto, un objeto externo y un objeto particular son tratados idénticamente por el compilador en el nivel de lenguaje fuente. Esto implica, en particular, que un programa Pascal llama a una función o procedimiento externo de la misma manera que a una función o procedimiento particular. La unión (linkaje) necesaria para las rutinas externas son manejadas por el compilador y por el interprete.

Desde el punto de vista de un módulo externo, una función o procedimiento externo es llamada desde un programa Pascal por el intérprete Pascal usando las subrutinas standard de conversión del PL/M. Los detalles de esta conversión y su relación con el Pas

cal-80 es como sigue:

- El valor de un parámetro actual es pasado a un procedimiento externo si el formato del correspondiente parámetro está especificado como un parámetro de valor.

- Si el procedimiento externo para funciones está codificado en PL/M o FORTRAN, cada parámetro actual del Pascal es asignado a su correspondiente formato de parámetro PL/M o FORTRAN.

- Si las funciones o procedimientos externos están codificados en lenguaje ensamblador, encontrará que sus argumentos se encuentra en las posiciones standard siguientes:

a) Un parámetro simple es pasado al par de registro BC

b) Para uno o dos parámetros de funciones o procedimientos, el primer parámetro es pasado comoencima y el segundo es pasado de una forma similar al par de registro DE.

c) Para una función o procedimiento con más de dos parámetros, los dos últimos son pasados como se describe encima(penúltimo en BC y el último en DE) y el resto se pone sobre el STACK. Estos son empujados sobre el Stack en orden de izquierda a derecha en la lista de parámetro, seguida por la localización de regreso en Pascal, el cual es el estado del STACK cuando la llamada a función o procedimiento empieza la ejecución.

- La única restricción en las subrutinas standard de conversión del PL/M es que una función externa PL/M debe ser un procedimiento direccionadoaun cuando el valor que devuelve ocupe un byte o menos. Similarmente, si una función externa es codificada en lenguaje ensamblador, siempre debe devolver su valor en el par de registro HL. Si se devolviera un byte en el registro A, debería copiarse el contenido de A en L y limpiar el registro H antes de que un programa sea devuelto.

La forma del FORTRAN-80 para llamar a una función con 'n' argumentos es pasar 'n+1' direcciones a la función rutina. La primera dirección es la localización para el almacenamiento del resultado y las próximas direcciones son las de localización de los 'n' argumentos (ya que, en FORTRAN-80, todos los argumentos son pasados por referencia).Por lo tanto si queremos llamar a una función externa de 'n' argumentos, escrita en FORTRAN-80, desde un programa Pascal, deberíamos declarar un procedimiento externo de 'n+1' parámetros, cada uno de los cuales es un parámetro VAR, y el primero de ellos es la variable la cual recibe el resultado de la función FORTRAN-80.

2.3 TABLA DE REFERENCIA EXTERNA

Si un programa Pascal-80 hace referencia a uno o más módulos externos, el compilador generará automáticamente una tabla de referencia externa sobre el

Isis II con los ficheros objetos relocalizables.El directivo del compilador $TAB(nombre-del-fichero) concede el nombre del fichero a la tabla de referencia externa para que sea especificada. En ausencia de este directivo, el compilador usa:

nombre-del-fichero-fuente.ERT

La tabla de referencia externa contiene una referencia externa para cada objeto externo declarado en el programa Pascal, y define la relación entre la colocación de cada objeto externo y la colocación se su marca indirecta de referencia en el nivel de un programa Pascal. La dirección de una tabla externa de referencia está definida por el símbolo público PQ$TAB.

3 EJECUCION DE UN PROGRAMA CON OBJETOS EXTERNOS

Para ejecutar un programa Pascal-80,el cual hace referencia a objetos externos,en necesario:

1) Compilar el programa Pascal.

2) Compilar y/O ensamblar los módulos externos.

3) Lincar los módulos externos con la tabla de referencia externa.

4) Realizar un 'locate' (situar) de los módulos producidos en el paso 3.

5) Cargar el módulo en memoria y llamar al RTS del Pascal-80.

Cada uno de estos pasos será explicado usando el

ejemplo que pusimos al principio.

3.1 COMPILANDO UN PROGRAMA PASCAL

Si el programa Pascal es una unidad de compilación simple, lo compilará para producir el fichero código y el fichero de la tabla externa de referencia. Si el programa Pascal es un programa PARTITIONED, compilará el módulo principal y cada uno de los COMPONENTS. El compilador generará el fichero de tabla externa de referencia cuando se compile el módulo principal; luego, llamando al comando JOIN, podremos combinar todos los ficheros código en otro fichero código resultante.

Por ejemplo, el comando:

PASCAL COMP SIX3

compilará SIX3.PAS y generará los siguientes tres ficheros:

SIX3.COD : Fichero código ejecutable.

SIX3.LST : Listado del fichero por el compilador.

SIX3.ERT : Fichero de la tabla de referencia externa.

3.2 COMPILAR Y/O ENSAMBLAR LOS MODULOS EXTERNOS

Traduce cada uno de los módulos externos codificados en lenguaje ensamblador usando unos procedimientos apropiados para el lenguaje. Asegura que cada referencia externa, en un programa pascal , sea igua-

lada a una declaración PUBLIC en la colección de módu los externos.

Por ejemplo los comandos:

PLM80 PSUM.PLM

ASM80 ASUM.ASM

producirán los siguientes ficheros:

PSUM.OBJ. Fichero objeto relocalizable.

PSUM.LST Listado del fichero que produce el com pilador.

ASUM.OBJ Fichero objeto relocalizable.

ASUM.LST Listado del fichero en ensamblador.

3.3 LINCAJE DE TODOS LOS MODULOS

Usando el comando LINK del Isis II, podemos lincar todos los módulos objetos externos con la tabla de referencia externa y el fichero librería P80EXT.LIB en un módulo objeto relocalizable.

Notar que el fichero conteniendo la tabla de referencia externa debe ser el primer fichero que aparezca en la lista de entrada de LINK, así que la tabla de referencia se sitúa al principio del módulo externo.

El formato del comando LINK es como sigue:

LINK ert-file,P80EXT.LIB,E1....EK TO code-file

donde E1....EK, son los ficheros objetos relocalizables de los módulos externos y algún fichero de librería el cual serviría de soporte.

Usando nuestro programa ejemplo, el comando apropiado es:

```
LINK SIX3.ERT,P80EXT.LIB,PSUM.OBJ,ASUM.OBJ,PLM80.LIB
                                    TO EXTMOD.OBJ
```

El cual creará un módulo objeto externo completo sobre EXTMOD.OBJ. Ya que SIX3.ERT es el primer comando mencionado en el comando de arriba, la tabla de referencia externa será correctamente situada al principio del segmento CODE.

3.4 COLOCACION DE LOS MODULOS EXTERNOS

Como hemos visto, durante una operación normal del Pascal-80, el RTS ocupa toda la memoria del sistema entre la parte alta del Isis II y la base de el monitor. Sin embargo, si un programa Pascal hace referencia a un módulo externo, el código para estos módulos debe residir en la memoria del sistema durante la ejecución de un programa Pascal. Para permitir esta situación, el RTS del Pascal-80 tiene un mecanismo por la cual puede trabajar en la 'linde' superior de su espacio de trabajo, para hacer puesto para un módulo externo.

El uso de este mecanismo requiere que uno de tales módulos externos sea colocado tan alto como sea posible en la memoria, para no restringir indebidamente el espacio de trabajo del Pascal-80.

El RTS sólo puede usar la memoria que está en la parte de abajo del módulo externo; nigún espacio de memoria entre la parte alta del módulo externo y el monitor será usada por el Pascal-80. Lo que veremos a continuación presenta las consideraciones cuando localizamos un módulo externo en un sistema INTELLEC de 64k.

Un sistema INTELLEC de 64k tiene su más alta dirección disponible(que podemos usar) en F6COH. Por lo tanto,la base del módulo externo debería ser localizado en:

F6COH- longitud total del módulo

Por ejemplo, si la longitud del módulo externo es de 4096 bytes (1000H), la base del módulo debe ser puesta a E6COH o más baja en el paso 'locate'.

Como fue mencionado encima, la tabla de referencia externa debe ser situada al principio del módulo externo. Si la operación de lincaje fuera ejecutada correctamente,como describimos en la sección previa, la tabla de referencia externa sería situada al principio del segmento CODE en el fichero objeto relocalizable del módulo externo. Por lo tanto , si el fichero segmento CODE es el primer segmento en el fichero objeto ejecutable generado por 'locate', la tabla de referencia externa será posicionada correctamente. El uso del comando 'locate' del Isis II posicionará el segmento CODE como el primer segmento, si el control de 'order' no es usado en el comando 'locate' o, si es usado, CO

DF se menciona de antemano en el nombre de algún otro segmento.

Por ejemplo:

LOCATE EXTMOD.OBJCODE(0E6C0H)

coloca el segmento CODE del módulo(y, por tanto, el módulo entero) en E6C0H, y escribe el módulo objeto absoluto sobre EXTMOD. Al menos que se requiera un particular STACK, el control STACKSIZE debería ser especificado, ya que el RTS coloca el STACK desde su espacio de trabajo.

Como un chequeo sobre la localización relativa de la tabla de referencia externa, la opción PUBLIC debería llamar al comando 'locate'.Esta opción causará que el valor del símbolo PQFTAB(entre otros) sea listado. Este valor debería ser exactamente el mismo que la dirección de base especificada en el comando 'locate'.

3.5 CARGA DEL MODULO EXTERNO EN MEMORIA Y LLAMADA AL PASCAL-80 RTS

El módulo externo localizado, EXTMOD, puede ahora ser cargado en la memoria del INTELLEC con su dirección propia con el comando DEBUG.

DEBUG EXTMOD

#nnnn

El comando DEBUG carga el fichero en la memoria y llama al monitor. El monitor responde con la dirección de comienzo del módulo. El monitor, luego, visualiza un punto (.) y espera hasta que se introduzca un co-

mando. Si respodemos con G8 seguido de 'CR'(carraige retutn), se retorna al Isis II.

En este punto, para que no se produzcan errores con los módulos externos cargados en la parte alta de la memoria, debemos llamar inmediatamente al Pascal-80 RTS con el comando:

PASCAL [R]*

El control * causa que el Pascal-80 se ponga en la linde superior de su espacio de trabajo para el valor del símbolo público PQETAB, el cual está definido para ser la base del módulo externo.

El Pascal-80 se usa ahora de una forma standard para la ejecución de un programa Pascal. Todas las referencias externas en el programa serán automáticamente lincadas a la asociación de objetos externos.

Por ejemplo, s el comando SIX3 se da al comando de intérprete de línea del Pascal-80, el programa ejemplo será ejecutado y accederá a las funciones y variables externas especificadas.

4 GENERACION DE UN PROGRAMA EN LA FORMA CARGA Y EJECUCION (LOAD AND GO)

En todo lo discutido hasta ahora, la única forma de ejecutar un programa en Pascal-80 era primero llamar al Pascal RTS, y luego el programa deseado. Mientras estos dos pasos son desarrollados, puede ha

ber un bit engorroso(molesto) para el depurado completo de un programa en producción. Para tener en cuenta esta última situación, es posible la construcción de un programa en la versión de carga y ejecución. Este 'load and go' es llamado directamente desde un comando de intérprete de línea del Isis II.

4.1 GENERACION DE UN MODULO OBJETO RELOCALIZABLE

El compilador del Pascal-80 produce un fichero código el cual está en un formato apropiado para la ejecución por el Pascal RTS. Sin embargo, este formato no es compatible con el formato de los ficheros objetos relocalizables, así que el fichero código debe ser transformado en un fichero objeto relocalizable con el comando GFNOBJ. El formato de este comando es:

GFNOBJ code-file TO obj-file [FREE size]

donde el control de FREE especifica el tamaño inicial del espacio libre, desde que el HEAP y el STACK son colocados. El HEAP crece hacia arriba desde la parte baja del espacio libre, mientras que el STACK crece hacia abajo desde lo alto.

La cantidad de espacio libre necesitado por un programa depende de las características estáticas y dinámicas del programa. Por ejemplo, de un fragmento de datos local de un procedimiento, afectará el espacio usado por el STACK, así como a la llamada profundidad del grupo

de procedimientos recursives. Para saber la cantidad de espacio libre necesitado por un programa Pascal, el Pascal-80 RTS tiene el comando SIZEON. Esta característica hace que el RTS visualice continuamente la cantidad de espacio libre ocupado por un programa, para así saber a qué máximo valor podemos llegar. Este valor puede ser luego visualizado mediante el comando STATS. Recordar, sin embargo, que el valor obtenido durante una ejecución podría no ser válido durante otra ejecución, si el programa tiene diferentes características de ejecución.

En suma, para visualizar el tamaño del STACK y el HEAP durante una o más ejecuciones, se deben tener en cuenta los siguientes factores:

- Los datos globales están colocados sobre el STACK cuando el programa empieza la ejecución.
- Los datos de una función o procedimiento estan colocado sobre el STACK sólo cuando esa función o procedimiento sea activada.
- Si una función o procedimiento es llamada recursivamente, cada una de sus llamadas tiene su propio fragmento de datos colocados sobre el STACK.
- Todos los parámetros para procedimientos, funciones y operaciones son pasados sobre el STACK.
- El código de un segmento de procedimiento o función activo es colocado sobre el STACK.
- Todas las variables dinámicas son colocadas sobre

el HEAP por el procedimiento predeclarado NEW(v).

Considerando esta guía de factores, obtenemos los siguientes conclusiones:

- Un procedimiento largo llamado cadena es la que más espacio del STACK usa.

- El más profundo nivel de recursión de un procedimiento o una función es la que más espacio del STACK usa.

- Dos procedimientos los cuales nunca son activados a la vez no ocuparán parte del STACK a la vez.

- La expresión más compleja en pascal es la que más espacio del STACK usa durante su evaluación.

4.2 LINCAJE CON MODULOS OBJETO

El módulo objeto relocalizable, creado desde el fichero código del programa Pascal, debe ahora ser lincado con el RTL (run-time-libraries) y asociado con los módulos externos. Para ello empleamos el comando LINK con el siguiente formato:

LINK input-list TO link-file controles de lincaje

La 'input-list' es una lista de ficheros del Isis II de la forma:

program-module,&

external-ref-table,....,external-module,&

P80RUN.LIB,P80RAR.LIB,P80ISS.LIB

donde:

- 'program-module' es el fichero objeto relocalizable creado desde el fichero código del programa Pascal por GFNOBJ, y debería ser el primer fichero sobre la lista.

- 'external-ref-table' es el fichero que contiene la tabla de referencia externa. Este término está presente sólo si el programa Pascal hace referencia a objetos externos.

- 'external-module' representa uno o más ficheros objetos los cuales contienen el código para los módulos externos. Este término está presente sólo si el programa Pascal hace referencia a objetos externos.

- Los tres últinos ficheros son la librería Pascal, requerida para que un programa Pascal corra bajo el Isis II.

4.3 LOCALIZANDO A UN MODULO OBJETO

El módulo objeto relocalizable, producido por el lincaje prévio, debe ahora ser localizado en una posición absoluta. Este procedimiento es ejecutado con el comando del Isis II LOCATE, cuyo formato es como sigue:

LOCATE link-file [TO output-file][controles]

La única restricción de este comando es que el segmento CODE debe ser localizado justo en el byte que forma límite. De esta manera, el usuario está libre

en la elección de localización para los segmentos varios.

4.4 EJECUCION DE UN PROGRAMA

El lincado y ejecución de un programa Pascal puede ser ahora cargado y ejecutado simplemente presentando su nombre en el comando de interprete de línea del Isis II.

Por ejemplo:

MYPROG [opciones]

cargara y ejecutará el programa Pascal, y no generará nigún tipo de mensaje automático.

PARTE III

ANALISIS DEL PROGRAMA

PROGRAMA DE GESTION PARA UNA BIBLIOTECA

En este apartado analizaré y explicaré el programa que he realizado, diseñado específicamente para llevar la gestión de una biblioteca.

Antes de entrar propiamente en materia espicificaré que el programa consta de 20 procedimientos, siendo éstos pequeños módulos que realizan siempre una misma tarea.

Comenzaré matizando que el programa lo primero que hace es llamar al procedimiento 'MENU', el cual nos muestra todas las opciones disponibles que son: (A)Operación de información, (B)Operación de borrar algún libro, y (C) Operación de registro de algún libro. Al final de este procedimiento se visualizan dos tipos de comentarios, que son: para cualquier opción teclear la letra asociada, y para acabar, presionar 'F'. Luego, el programa se encuentra en actitud de espera hasta que se le introduzca una opción. A continuación, el programa realiza un 'IF', cuya misión es comparar si la opción que hemos introducido corresponde con 'F'; en tal caso, el programa finaliza y nos devuelve el control al Isis II. En caso de que la opción introducida sea cualquiera menos la 'F', se produce una llamada al procedimiento 'MAT' que muestra la relación de materias existentes, en este caso 11, para conocer con qué materias podemos hacer operaciones. Seguidamente se llama al procedimiento 'OPC' que nos muestra, dependiendo de la opción que hallamos presionado en el proce-

dimiento 'MENU', los siguientes comentarios: para la opción A, presionar el número asociado para información; para la opción B, seleccionar la materia en la cual se desea borrar; y para la opción C, seleccionar la materia en la cual se desea introducir el libro. Tras esto, el programa se introduce en un 'CASE' que permite las tres opciones del MENU. A continuación explicaremos cada opción.

1) OPCION C (operación de registrar algún libro). Aquí se llama a 4 procedimientos. Al primero que se llama es al procedimiento 'FICHE', cuya misión es abrirnos el fichero de la materia que hayamos selecionado en el procedimiento 'OPC' (hemos de notar que todos los ficheros de materias han sido creados previamente a la ejecución del programa mediante el comando 'CREDIT' del Isis II). Después de abrir el fichero correspondiente, se observará si existe alguna grabación en ese fichero para que, en caso de que sea cierto, poner el puntero al final de la última grabación. Al segundo procedimiento que se llama dentro de esta opción es al 'INT', que se encarga de preguntarnos todo acerca del libro que vamos a introducir. También comprueba, mediante la comparación del título y los autores que lo escribieron, si el citado libro existe ya en el fichero. En caso de que la comparación anterior resulte positiva, nos aparece el siguiente comentario por la pantalla: 'el citado libro ya existe', provocando posteriormente una salida de este procedimiento. El tercer procedimiento que se utiliza dentro de esta opción

es el 'INTA', que toma toda la información del procedimiento 'INT' y la introduce en un buffer, para luego introducirlo en el fichero seleccionado mediante la sentencia PUT(BI). Al último procedimiento que se llama dentro de esta opción es al 'MT' el cual peegunta: '¿quiéres meter un nuevo libro?'. En caso afirmativo nos manda al procedimiento 'INT' y luego al 'INTA', cuya misiones hemos visto arriba. Si contestamos que no, nos manda al comentario: 'seleccionar una nueva operación a realizar(A,B,C, y F)'.

2) OPCION A (operacion de información). Aquí se realizan una serie de llamadas que explicaremos a continuación. Primero se llama al procedimiento 'FICHE' cuya misión fue explicada anteriromente. Luego se obtiene el comentario siguiente: elegir forma de listado (general o autor). Seguidamente explicaremos cada forma.

- Listado general (G). Si elegimos esta forma, se produce la llamada a 4 procedimientos, siendo el primero de estos el 'CA', que nos pone una cabecera en la pantalla en donde se indica título, autor(es), y la referencia. Seguidamente se llama al procedimiento 'LIST', cuyo objetivo es sacarnos un listado de todos los libros existentes en el fichero con el cual estamos trabajando. Al siguiente procedimiento que se llama es al 'INFO', que sólo nos muestra los siguientes comentarios: 'para cualquier información presionar la referencia', y, 'si no desea información presionar un cero (0)'. Tras esto, se llama al últi

mo procedimiento dentro de la opción listado general, que es el 'ELEC' donde, dependiendo de lo que hallamos contestado en el procedimiento 'INFO' nos realizará una cosa u otra. En caso que hayamos contestado con un cero, se producirá una salida del 'CASE', llevándonos el control al principio del programa, concretamente al siguiente comentario: 'seleccionar una nueva operación a realizar (A,B,C, y F)'. En caso de haber contestado con la referencia, se realizará lo siguiente:

Primero se llama al procedimiento 'SE', que pone el puntero del fichero al principio de la grabación que queremos consultar, asignando toda la información de esa grabación a la variable buffer, y luego pasa todo el contenido del buffer a la variable 'DP' (el objeto de esta doble asignación se debe a que el buffer mantiene sólo la última información que usa. Si no hicieramos esto, la información inicial que contiene el fichero se nos perdería de nada que hicieramos alguna operación con el buffer).

Segundo, se llama al procedimiento 'FICH' que es el encargado de visualizar por la pantalla la ficha de cada libro. Llegados a este punto, puede ocurrir que la variable 'disponibilidad' de esa ficha sea 'S' o 'N'.

a) En caso que la variable 'dispinibilidad' sea 'S' (quiere decir que se encuentra disponible en la biblioteca) nos pregunta: '¿ se desea realizar operación de entrada/salida ?.' A continuación se llama al procedimiento

'CAM', que es donde se contesta la pregunta anteriormente formulada. En caso de contestar que 'N' (no), se produce una salida del procedimiento que nos lleva al comentario de: 'seleccionar una nueva operación a realizar (A,B,C, y F)'. En caso de contestar que 'S' (si), lo primero que hace es preguntarnos lo siguiente: '¿qué operación desea realizar, entrada o salida (E/S)?'.

En caso de contestar con 'S' (salida) se pone el puntero al principio de la grabación para cambiar el contenido de la variable 'disponibilidad', que tenía 'S' (de disponible) por 'N' (de no disponible en la biblioteca). A continuación se llama al procedimiento 'ELI' que es el que pregunta: '¿quién va a realizar la operación de entrada/salida, alumno o profesor (a/P)?'. Cualquiera que sea la respuesta, hace que se nos pidan los datos personales del alumno o profesor, según el caso. Tras esto se llama al procedimiento 'FEC', que es el encargado de preguntarnos por la ficha de entrada y por la fecha de salida. También se encarga de introducir todos los datos de este procedimiento y del anterior (el ELI) en la variable buffer para luego meterlo en el fichero.

En caso de contestar con 'N' (entrada) , lo primero que se hace es poner el puntero al principio de la grabación donde se encuentra el libro que se desea entrar. Luego se le asigna a la variable 'disponibilidad', que tenía 'N' (no disponible), un 'S' (si disponible); seguidamente esta asignación la introducimos en el buffer para luego in

troducirla en el fichero (lo de introducirla en el buffer, se debe a que toda información que se desee archivar en un fichero debe entrar en dicho fichero a través de un buffer). Después de meterlo en el fichero, se produce una salida de este procedimiento CAM, así como una salida de ELEC para llevarnos al comentario : 'seleccionar una nueva operación a realizar (A,B,C, y F)'.

b) En caso de que la variable 'disponibilidad' sea 'N' (quiere decir que no se encuentra disponible en la biblioteca) entonces se ejecutan 3 nuevos procedimientos. El primero de ellos es el 'WHO', que es el que nos da los datos de la persona que tiene el libro (tanto alumno como profesor). El segundo procedimiento es el 'PERS', que es el que se encarga de mostrarnos la fecha de salida y la fecha de entrada. En este mismo procedimiento, al final, se pregunta: '¿se desea realizar alguna operación de entrada/salida?'. El tercer procedimiento es el 'CAM', que es donde se contesta la pregunta anterior y cuya misión ya hemos explicado anteriormente. El siguiente paso es la salida del procedimiento ELEC para llevarnos al comentario: 'seleccionar una nueva operación a realizar (A,B,C, y F)'.

- Listado por autores (A). En esta forma de listado funciona todo igual que en el general, menos en el procedimiento LIS. Como hemos visto, para obtener un listado general debíamos llamar a los siguientes procedimientos: CA,LIS,INFO, y ELEC. Ahora bien, para un listado

por autores, los procedimientos que se llaman son: CA, BU, INFO, y ELEC. Como se aprecia, sólo difiere en el procedimiento 'BU'. En esta opción (listado por autores) primeramente se pregunta: '¿nombre del autor (en caso de varios poner uno solo)?'. El procedimiento 'BU' lo que hace es comparar cada autor del fichero en cuestión con el autor que nosotros hemos introducido, de tal manera que, cuando sean iguales, proporcione una salida por pantalla del título, autor(es), y referencia. Todo lo demás funciona igual que para el listado general. Cuando ejecute todos los procedimientoe mencionados arriba, nos lleva al comentario: 'seleccionar una nueva operación a realizar (A, B, C, y F)'.

3) OPCION B (operacion de borrar algún libro). Aquí lo primero que se hace es llamar al procedimiento 'FICHE' para abrir el fichero donde queremos borrar. Luego, mediante los dos procedimientos siguientes, el CA y el LIST, se saca un listado general, como se ha visto en las explicación del listado general. Finalmente se llama al procedimiento 'BORR', que es el encargado de preguntar: '¿estamos seguro de querer borrar este libro?'. Si contestamos que 'N' (no), se produce una salida de 'BORR', llevándonos el programa al comentario: 'seleccionar una nueva operación (A, B, C, y F)'. Si contestamos que 'S' (si), entonces se llama al procedimiento 'INT', que sirve para introducir los datos del nuevo libro. Luego se coloca el puntero en

el lugar que deseamos borrar para terminar llamando al procedimiento 'INTA', que es el encargado de introducirnos los datos del nuevo libro en el lugar del antiguo. Después de todo esto, el desarrollo del programa nos lleva al comentario: seleccionar nueva operación a realizar (A,B,C, y F).

Cada vez que el programa pasa por el comentario: 'seleccionar nueva operación a realizar (A,B,C, y F)', se produce un cierre del fichero que estaba abierto.

También hemos de hacer notar que el programa está preparado para que cada pregunta pueda ser contestada con una letra solamente.

Otra cosa que es importante es la manera de introducir los autores. Si un libro ha sido escrito únicamente por un autor, entonces se pone el nombre del autor seguido de una coma (el poner la coma es muy importante). En caso que haya sido escrito por varios autores se pone cada autor y entre cada autor una coma, incluso al último se le debe poner una coma.

El modo para poder trabajar con este programa es introducir el disco que contiene el programa. Una vez introducido, se resetea para luego llamar al Pascal. Después de que nos haya contestado el sistema de desarrollo, pasaremos a correr el programa poniendo BI.

LISTADO DEL PROGRAMA

```
PROGRAM BIBLIO;
    CONST LON=100;
    TYPE FECHA=RECORD
                 DIA:1..31;
                 MES:1..12;
                 ANO:INTEGER;
               END;
         DATOS=RECORD
                 AUT:STRING[20];
                 TIT:STRING[40];
                 MAT:STRING[14];
                 F←D←S,F←D←E,F←D←R:FECHA;
                 S←E←B:STRING[8];
                 S←A,XX:CHAR;
                 N←A,N←P,CU:STRING[20];
                 N←D←C,D←D:STRING[10];
               END;
         0=STRING[20];

    VAR BI:FILE OF DATOS;
        DP:DATOS;
        CH,Z,F:CHAR;
        W:STRING[2];
        N:ARRAY[1..LON] OF 0;
        J,ZZ:INTEGER;
        AU,H:0;
        AA:1..11;

PROCEDURE MENU(MEN:BOOLEAN);
    BEGIN
      WRITELN;WRITELN;
      WRITELN('                                        MENU');
      WRITELN('                                        ----');
      WRITELN;WRITELN;WRITELN;
      WRITELN('                          (A) OPERACION DE INFORMACION ');
      WRITELN;
      WRITELN('                          (B) OPERACION DE BORRAR ALGUN LIBRO');
      WRITELN;
      WRITELN('                          (C) OPERACION DE REGISTRAR ALGUN LIBRO');
      WRITELN;WRITELN;WRITELN;
      WRITELN('(PARA CUALQUIER OPCION TECLEAR LA LETRA ASOCIADA)');WRITELN;
      WRITELN('(PARA ACABAR PRESIONAR LA LETRA F)');WRITELN;
```

```
PROCEDURE MAT(MATE:BOOLEAN);
    BEGIN
      WRITELN;WRITELN;
      WRITELN('::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::');
      WRITELN;
      WRITELN('                          RELACION DE MATERIAS EXISTENTES ');
      WRITELN;
      WRITELN('::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::');
      WRITELN;WRITELN;
      WRITELN('(1) ALGEBRA                    (2) CALCULO                    (3) DIBUJO');
      WRITELN;
      WRITELN('(4) ELECTRONICA                (5) FISICA                     (6) MECANICA');
      WRITELN;
      WRITELN('(7) NAVALES                    (8) OBRAS PUBLICAS             (9) ORDENADORES');
      WRITELN;
      WRITELN('(10) QUIMICA                   (11) TOPOGRAFIA');
      WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;
    END;

PROCEDURE OPC(OPCI:BOOLEAN);
    BEGIN
        CASE CH OF
          'A':BEGIN
                WRITELN('(PRESIONAR EL NUMERO ASOCIADO PARA INFORMACION)');
                WRITELN;
              END;
          'B':BEGIN
                WRITELN('(SELECIONAR LA MATERIA EN LA CUAL SE DESEA BORRAR)');
                WRITELN;
              END;
          'C':BEGIN
                WRITELN('(SELECCIONAR LA MATERIA EN LA CUAL SE DESEA INTRODUCIR EL LIBRO)');
                WRITELN;
              END;
        END (CASE);
        REPEAT
          READLN(AA);
        UNTIL AA IN [1..11];
    END;
```

```
PROCEDURE FICH(FIC:BOOLEAN);
    BEGIN
      WRITELN('*************************************************************************************');
      WRITELN('*                                                                                   *');
      WRITELN('*     TITULO : ',BI↑.TIT:40,'  REG: ',ZZ:3,'                *');
      WRITELN('*                                                                                   *');
      WRITELN('*     AUTOR(ES) : ',BI↑.AUT:20,'                                                *');
      WRITELN('*                                                                                   *');
      WRITELN('*     MATERIA DE : ',BI↑.MAT:14,'   FECHA DE REGISTRO : ',
                 BI↑.F←D←R.DIA:3,BI↑.F←D←R.MES:3,BI↑.F←D←R.ANO:5,
'            *');
      WRITELN('*                                                                                   *');
      WRITELN('*************************************************************************************');
      WRITELN('*     SITUACION EN LA BIBLIOTECA : ',BI↑.S←E←B:8,
              '  DISPONIBILIDAD : ',BI↑.S←A:2,'                *');
      WRITELN('*************************************************************************************');
    END;

PROCEDURE ELI(ELIC:BOOLEAN);
    BEGIN
      SEEK(BI,ZZ-1);
      REPEAT
       WRITE('QUIEN VA A REALIZAR LA OPERACION DE ENTRADA/SALIDA, ALUMNO O PROFESOR(A/P)');READLN(Z);
      UNTIL Z IN ['A','P'];
      DP.XX:=Z;
        CASE Z OF
          'A':BEGIN
                WRITE('NOMBRE DEL ALUMNO : ');READLN(INPUT,DP.N←A);WRITELN;
                WRITE('CURSO : ');READLN(INPUT,DP.CU);WRITELN;
                WRITE('NUMERO DEL CARNET DE LA ESCUELA : ');READLN(INPUT,DP.N←D←C);
                WRITELN;
              END;
          'P':BEGIN
                WRITE('NOMBRE DEL PROFESOR : ');READLN(INPUT,DP.N←P);WRITELN;
                WRITE('PERTENECIENTE AL DEPARTAMENTO DE : ');READLN(INPUT,DP.D←D);
                WRITELN;
              END;
        END (CASE);
    END;
```

```
PROCEDURE WHO(QUI:BOOLEAN);
    BEGIN
          CASE BI↑.XX OF
            'A':BEGIN
                  WRITELN('*                                                                                *');
                  WRITELN('*     EL PRESENTE LIBRO SE ENCUENTRA EN POSESION DEL ALUMNO                      *');
                  WRITELN('*     NOMBRE : ',BI↑.N←A:20,'  CURSO : ',BI↑.CU:20,'                             *');
                  WRITELN('*     NUMERO DEL CARNET DE LA ESCUELA : ',BI↑.N←D←C:10,'                         *');
                END;
            'P':BEGIN
                  WRITELN('*                                                                                *');
                  WRITELN('*     EL PRESENTE LIBRO SE ENCUENTRA EN POSESION DEL PROFESOR                    *');
                  WRITELN('*     NOMBRE : ',BI↑.N←P:20,'  DEPARTAMENTO DE : ',BI↑.D←D:10,'                  *');
                END;
          END (CASE);
    END;
```

```
PROCEDURE PERS(PER:BOOLEAN);
    BEGIN
      WRITELN('*                                                                                *');
      WRITELN('*     FECHA DE SALIDA : ',BI↑.F←D←S.DIA:3,BI↑.F←D←S.MES:3,
                  BI↑.F←D←S.ANO:5,'     FECHA DE ENTRADA : ',BI↑.F←D←E.DIA:3,
                  BI↑.F←D←E.MES:3,BI↑.F←D←E.ANO:5,'          *');
      WRITELN('*                                                                                *');
      WRITELN('**********************************************************************************');WRITELN;
      WRITE('DESEA REALIZAR ALGUNA OPERACION DE ENTRADA/SALIDA ?: ');
    END;

PROCEDURE FICHE(FI:BOOLEAN);
    BEGIN
        CASE AA OF
          1:RESET(BI,'ALG');
          2:RESET(BI,'CAL');
          3:RESET(BI,'DIB');
          4:RESET(BI,'ELE');
          5:RESET(BI,'FIS');
          6:RESET(BI,'MEC');
          7:RESET(BI,'NAV');
          8:RESET(BI,'OBP');
          9:RESET(BI,'ORD');
         10:RESET(BI,'QUI');
         11:RESET(BI,'TOP');
        END (CASE);
          WHILE NOT EOF(BI) DO
             BEGIN
               GET(BI);
             END;
    END;
```

```
PROCEDURE INT(ITRO:BOOLEAN);
    BEGIN
            WRITELN;
            WRITE('ESCRIBA EL TITULO: ');READLN(INPUT,DP.TIT);WRITELN;
            WRITELN('ESCRIBA EL NOMBRE DEL AUTOR(ES),(DESPUES DE CADA AUTOR PONER UNA COMA');WRITELN;
            READLN(INPUT,DP.AUT);WRITELN;
            RESET(BI);
            REPEAT
              IF (DP.TIT=BI↑.TIT) AND (DP.AUT=BI↑.AUT) THEN
                BEGIN
                  WRITELN;WRITELN('EL CITADO LIBRO YA EXISTE');WRITELN;
                  EXIT(INT);
                END;
              GET(BI);
            UNTIL EOF(BI);
            WRITE('FECHA DE REGISTRO : ');READLN(INPUT,DP.F←D←R.DIA,
               DP.F←D←R.MES,DP.F←D←R.ANO);WRITELN;
            WRITE('SITUACION EN LA BIBLIOTECA : ');READLN(INPUT,DP.S←E←B);
            WRITELN;
            WRITE('DISPONIBILIDAD : ');READLN(INPUT,DP.S←A);WRITELN;
            WRITE('EL CITADO LIBRO SE DESEA INTRODUCIR EN LA MATERIA DE : ');
            READLN(INPUT,DP.MAT);
            WRITELN;
    END;

PROCEDURE INTA(INTE:BOOLEAN);
   BEGIN
     BI↑:=DP;
     PUT(BI);
   END;

PROCEDURE MT(MET:BOOLEAN);
  VAR Y:CHAR;
  BEGIN
    REPEAT
      WRITE('QUIERE METER UN NUEVO LIBRO?: ');READLN(Y);
      CASE Y OF
        'S':BEGIN
              INT(TRUE);
              INTA(TRUE);
            END;
        'N':EXIT(MT);
      END(CASE);
      WRITELN;
    UNTIL Y IN ['N'];
  END;
```

```
PROCEDURE FEC(FCH:BOOLEAN);
    BEGIN
      WRITE('FECHA DE SALIDA : ');READLN(INPUT,DP.F↑D↑S.DIA,DP.F↑D↑S.MES,
                                         DP.F↑D↑S.ANO);WRITELN;
      WRITE('FECHA DE ENTRADA : ');READLN(INPUT,DP.F↑D↑E.DIA,DP.F↑D↑E.MES,
                                         DP.F↑D↑E.ANO);WRITELN;
      BI↑:=DP;
      PUT(BI);
    END;

PROCEDURE LIS(LIT:BOOLEAN);
    BEGIN
      J:=0;
      RESET(BI);
      REPEAT
        J:=J+1;
        WRITELN(BI↑.TIT:40,'   ',BI↑.AUT:20,'   ',J);
        GET(BI);
      UNTIL EOF(BI);
      WRITELN;
    END;

PROCEDURE INFO(INF:BOOLEAN);
    BEGIN
      WRITELN('PARA CUALQUIER INFORMACION PRESIONAR LA REFERENCIA');WRITELN;
      WRITELN('SI NO SE DESEA INFORMACION PRESIONAR UN CERO (0)');WRITELN;
    END;

PROCEDURE BORR(BOR:BOOLEAN);
 VAR PP:CHAR;
 BEGIN
  WRITELN('PARA CUALQUIER BORRADO PRESIONAR LA REFERENCIA ASOCIADA');WRITELN;
  WRITELN('(TENER EN CUENTA QUE HAY QUE INTRODUCIR UN LIBRO EN EL LUGAR DEL');
  WRITELN('QUE SE BORRA)');WRITELN;
  READLN(ZZ);WRITELN;
  WRITELN('ESTA USTED SEGURO DE QUERER BORRAR ESE LIBRO?: ');READLN(PP);
  WRITELN;
  CASE PP OF
   'S':BEGIN
         INT(TRUE);
         SEEK(BI,ZZ-1);
         INTA(TRUE);
       END;
   'N':EXIT(BORR);
  END(CASE);
 END;
```

```
PROCEDURE BU(BUSQUE:BOOLEAN);
    VAR R,I,L,LO:INTEGER;
        RO:ARRAY[1..100] OF STRING[20];
    BEGIN
      J:=0;
      RESET(BI);
      WRITELN;
      I:=0;
      REPEAT
        I:=I+1;J:=J+1;
        RO[I]:=BI↑.AUT;
        R:=1;
        L:=POS(',',RO[I]);
        REPEAT
           LO:=L-R;
           H:=COPY(RO[I],R,LO);
           FILLCHAR(RO[I],L+1,'0');
           IF H=AU THEN
             BEGIN
               WRITELN(BI↑.TIT:40,'   ',BI↑.AUT:20,'   ',J);
             END;
           R:=R+(LO+1);
           L:=POS(',',RO[I]);
        UNTIL L=0;
        GET(BI);
      UNTIL EOF(BI);
      WRITELN;
    END;

PROCEDURE CA(CAB:BOOLEAN);
    BEGIN
      WRITELN;
      WRITELN(':::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::');
      WRITELN('                                          :                              :');
      WRITELN('               TITULO                     :       AUTOR(ES)              :  REFERENCIA ');
      WRITELN('                                          :                              :');
      WRITELN(':::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::');
      WRITELN;
    END;

PROCEDURE SE(SEE:BOOLEAN);
    BEGIN
      SEEK(BI,ZZ-1);
      GET(BI);
      DP:=BI↑;
    END;
```

```
PROCEDURE CAM(CC:BOOLEAN);
    VAR Q,WW:CHAR;
    BEGIN
     REPEAT
      READLN(Q);
     UNTIL Q IN ['S','N'];
        IF Q='S' THEN
          BEGIN
            WRITE('QUE OPERACION DESEA REALIZAR ENTRADA O SALIDA (E/S)?: ');
            READLN(WW);
            CASE WW OF
              'E':BEGIN
                    SEEK(BI,ZZ-1);
                    DP.S<-A:='S';
                    BI↑:=DP;
                    PUT(BI);
                    EXIT(CAM);
                  END;
              'S':BEGIN
                    SEEK(BI,ZZ-1);
                    DP.S<-A:='N';
                  END;
            END(CASE);
            ELI(TRUE);
            FEC(TRUE);
          END;
    END;
```

```
PROCEDURE ELEC(EE:BOOLEAN);
    BEGIN
      READLN(ZZ);
      WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;
      IF ZZ<>0 THEN
        BEGIN
          SE(TRUE);FICH(TRUE);
          CASE BI↑.S←A OF
            'S':BEGIN
                  WRITELN;
                  WRITE('DESEA REALIZAR OPERACION DE ENTRADA/SALIDA ?:');
                  CAM(TRUE);
                END;
            'N':BEGIN
                  WHO(TRUE);
                  PERS(TRUE);
                  CAM(TRUE);
                END;
          END (CASE);
        END;
    END;
```

```
    WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;
    MENU(TRUE);
REPEAT
    REPEAT
      READLN(CH);
    UNTIL CH IN ['A','B','C','F'];
    IF CH='F' THEN
        BEGIN
          EXIT(BIBLIO);
        END;
    WRITELN;
    MAT(TRUE);
    OPC(TRUE);
    CASE CH OF
      'C':BEGIN
            FICHE(TRUE);
            INT(TRUE);
            INTA(TRUE);
            MT(TRUE);
          END;
      'B':BEGIN
            FICHE(TRUE);
            CA(TRUE);
            LIS(TRUE);
            BORR(TRUE);
          END;
      'A':BEGIN
            FICHE(TRUE);
            REPEAT
              WRITELN;
              WRITE('ELEGIR FORMA DE LISTADO (GENERAL O AUTOR)');READLN(F);
              WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;
            UNTIL F IN ['G','A'];
              CASE F OF
                'G':BEGIN
                      CA(TRUE);
                      LIS(TRUE);
                      INFO(TRUE);
                      ELEC(TRUE);
                    END;
                'A':BEGIN
                      WRITE('NOMBRE DEL AUTOR (EN CASO DE VARIOS PONER UNO SOLO)');READLN(AU);
```

```
                        CA(TRUE);
                        BU(TRUE);
                        INFO(TRUE);
                        ELEC(TRUE);
                      END;
                END (CASE);
            END;
        END (CASE);
    CLOSE(BI);
    WRITELN('SELECCIONAR NUEVA OPERACION A REALIZAR. (A,B,C,F)');
    WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;
    UNTIL CH IN ['F'];
END(BIBLIO).
```

APFNDICE A

LISTADO DE PROGRAMAS

A continuación se muestra una colección de programas en donde está todo lo explicado hasta ahora. Primeramente veremos lo que hace cada programa.

PROGRAM MEDIA. Este programa nos calcula la media de dos números.

PROGRAM PONTENCIA. Este nos calcula cuanto es un número elevado a otro.

PROGRAM LEER_Y_SUMAR. Mediante este programa leemos diez números y ejecuatamo su suma.

PROGRAM VOLUMEN. Calcula el volumen de una esfera.

PROGRAM NOTAS. Lee un nombre y una nota, y si la nota es mayor que seis nos devuelve el nombre del alumno y como nota pone'apto'.En caso que sea menor que seis pone 'no apto'.

PROGRAM CLASIFICACION.Lee un nombre y nos dice si está incluido entre la 'D' y la 'G'.

PROGRAM RAIZCUADRADA. Nos lee un número formandonos una tabla con el número y su raiz cuadrada.

PROGRAM SERIE. Nos calcula la serie S:=S+1/I a partir de un número que le damos nosotros.

PROGRAM MAYOR_MENOR. Metemos tres enteros y nos dice cual es mayor y cual menor.

PROGRAM COMPARACION.Metemos cuatro enteros y nos hace una comparación de todos ellos.

PROGRAM FIBONACCI. Introducimos un valor de 'N' y nos calcula la serie de fibonacci hasta ese número.

PROGRAM FIBONACCI2. Metemos un valor y nos calcula el primer termino mayor que ese dentro de la serie de fibonacci.

PROGRAM NOTA2. Obtiene la media de cinco notas indicandonos si una determinada persona ha superado esa media o no.

PROGRAM ORDENACIÖN. Con este programa podemos meter enteros al azar que al final el programa nos lo devuelve ordenado.

PROGRAM MAYOR. Nos calcula cual es el mayor de dos números mediante una función.

PROGRAM FICH1. Nos crea un fichero en donde podemos ir introduciendo libros. Al final de introducir los libros nos proporciona un listado de todos ellos según lo hemos introducidos.

PROGRAM FICH2. Nos crea dos ficheros en donde uno es para el nombre y el otro para el teléfono. Una vez que tengamos una lista de nombres con sus teléfonos correspondientes, basta con introducir un nombre para obtener el teléfono que tiene asociado ese nombre.

PROGRAM FICH3. Nos crea dos ficheros. Uno para el nombre y otro para los datos presonales. El objetivo de este programa es conocer los datos personales de una persona determinada con sólo introducir su nombre. También tiene la posibilidad de cambiar los datos de una persona.

PROGRAM FICH4 . Es igual que el FICH1,pero el listado de los libros en vez de ser según se introducieron es por orden alfabético.

PROGRAM FICHE. Nos crea dos ficheros. Uno para el título y otro para los autor/es. El objetivo de este programa es obtener un listado de todos aquellos dibros que han sido escrito por un mismo autor. También, cuando un libro ha sido escrito por más de un autor, el programa está preparado para poder ponerlos todos sin que ello afecte el listado de los dibros de un determinado autor.

```
program media;
var rmed,a,b:real;
begin{media}
  writeln('meter datos');
  read(a,b);
  rmed:=(a+b)/2.0;
  writeln('la media es:',rmed);
end{media}.
```



```
PROGRAM POTENCIA (CALCULA A↑B);
    VAR A,B,POT,K:INTEGER;
    BEGIN
      K:=0;
      POT:=1;
      WRITELN('METER DATOS:');
      READ(A,B);
      WHILE K<B DO
      BEGIN
          POT:=POT*A;
          K:=K+1;
      END;
      WRITELN('EL RESULTADO ES:',POT:0);
    END.
```

```
PROGRAM LEER←Y←SUMAR;
   VAR SUM,DATO,CONTA:INTEGER;
   BEGIN
     SUM:=0;
     CONTA:=0;
     WHILE CONTA<10 DO
        BEGIN
          WRITE('METER LOS DATOS:');
          READLN(DATO);
          WRITELN(DATO);
          SUM:=SUM+DATO;
          CONTA:=CONTA+1;
        END;
     WRITELN('LA SUMA DE LOS 10 ENTEROS ES=',SUM:0);
   END.
```



```
PROGRAM VOLUMEN(CALCULA VOLUMEN DE UNA ESFERA);
  CONST PI=3.1415;
  VAR RADIO,VOLUM:REAL;
  BEGIN
    WRITE('METER EL RADIO:');
    READ(RADIO);
    VOLUM:=4/3*PI*RADIO*RADIO*RADIO;
    WRITELN('EL VOLUMEN ES:',VOLUM:0);
  END.
```

```
PROGRAM NOTAS;
  VAR NOTA:INTEGER;
      NOMBRE:STRING[15];
      CALF:STRING[7];
  BEGIN
    WRITELN('INTRODUCIR NOMBRE(CR) Y  NOTA:');
    READLN(NOMBRE,NOTA);
    IF (NOTA < 6) THEN
       CALF:='NO APTO'
       ELSE CALF:='APTO';
    WRITELN('ALUMNO:',NOMBRE,'       CALIFICACION:',CALF);
  END.
```

```
PROGRAM CLASIFICACION;
  VAR NOMBRE:STRING[4];
  BEGIN
    WRITE('INTRODUCIR EL NOMBRE:');
    READ(NOMBRE);
    IF ((NOMBRE<'D') OR (NOMBRE>'G')) THEN
       WRITELN('NO ESTA INCLUIDO ENTRE D Y G')
       ELSE WRITELN('SI ESTA INCLUIDO ENTRE D Y G');
  END.
```

```
PROGRAM RAIZCUADRADA;
  CONST N1=6.0;
        N11=7.5;
        D=0.1;
  VAR Z:REAL;
  BEGIN
    WRITELN('NUMERO  RAIZ CUADRADA');
    WRITELN;
    Z:=N1;
    WHILE Z<=N11 DO
      BEGIN
        WRITELN(Z:12:6,SQRT(Z):12:6);
        Z:=Z+D;
      END;
  END.
```



```
PROGRAM SERIE;
  VAR I,N:INTEGER;
      S:REAL;
  BEGIN
    WRITE('METER EL VALOR DE N:');
    READ(N);
    S:=0.0;
    FOR I:=N DOWNTO 1 DO
    S:=S+1/I;
    WRITELN('S(',N,')=',S);
  END.
```

```
PROGRAM MAYOR←MENOR;
   LABEL 5,7,8,12,16,17;
   VAR V1,V2,V3,VMAX,VMIN:INTEGER;
 BEGIN
     WRITE('METER TRES ENTEROS:');
     READLN(V1,V2,V3);
     IF V1>V2
       THEN GOTO 7
       ELSE IF V1<V3
              THEN GOTO 8
              ELSE VMIN:=V3;
   5:VMAX:=V2;
     GOTO 17;
   7:IF V1>V3
       THEN GOTO 12
       ELSE VMAX:=V3;
            VMIN:=V2;
   8:VMIN:=V1;
     IF V2>V3
       THEN GOTO 5
       ELSE VMAX:=V3;
     GOTO 17;
  12:VMAX:=V1;
     IF V3>V2
       THEN GOTO 16
       ELSE VMIN:=V3;
     GOTO 17;
  16:VMIN:=V2;
  17:WRITELN('EL MAYOR ES:',VMAX,'  EL MENOR ES:',VMIN);
 END.
```

```
ISIS-II PASCAL-80 Compiler V2.0 invoked by:
>:F0:COMP.COD P6.PAS


  Line Seg Proc Lev Disp

     1   1    1          1  PROGRAM COMPARACION;
     2   1    1          3    VAR A,B,C,D:INTEGER;
     3   1    1    0     0    BEGIN
     4   1    1    1     0      WRITE('METER 4 DATOS:');
     5   1    1    1    27      READ(A,B,C,D);
     6   1    1    1    69      WRITE(A,B,C,D);
     7   1    1    1   111        IF A<B
     8   1    1    1   114          THEN BEGIN
     9   1    1    3   120            WRITELN(A,' ES MENOR QUE',B);
    10   1    1    3   174            IF A<C
    11   1    1    3   177              THEN BEGIN
    12   1    1    5   183                WRITELN(A,' ES MENOR QUE',C);
    13   1    1    5   237                IF A<D
    14   1    1    5   240                  THEN BEGIN
    15   1    1    7   246                    WRITELN(A,' ES MENOR QUE',D);
    16   1    1    6   300                  END(COMPARACION A,D);
    17   1    1    4   302                END(COMPARACION A,C);
    18   1    1    2   304            END(COMPARACION A,B)
    19   1    1    1   306          ELSE WRITELN(A,' ES MAYOR QUE LOS DEMAS');
    20   1    1    0   362    END.
```

```
ISIS-II PASCAL-80 Compiler V2.0 invoked by:
>:F0:COMP.COD P11.PAS


  Line Seg Proc Lev Disp

     1   1    1          1  PROGRAM FIBONACCI;
     2   1    1          3   VAR PRIMERO,SEGUNDO,SIGUIENTE,N,I:INTEGER;
     3   1    1    0     0    BEGIN
     4   1    1    1     0    WRITELN('METER EL VALOR DE N:');
     5   1    1    1    41    READ(N);
     6   1    1    1    53    PRIMERO:=0;
     7   1    1    1    58    SEGUNDO:=1;
     8   1    1    1    63    WRITELN(PRIMERO);
     9   1    1    1    83    WRITELN(SEGUNDO);
    10   1    1    1   103    FOR I:=3 TO N DO
    11   1    1    2   116       BEGIN
    12   1    1    3   118         SIGUIENTE:=PRIMERO+SEGUNDO;
    13   1    1    3   125         WRITELN(SIGUIENTE);
    14   1    1    3   145         PRIMERO:=SEGUNDO;
    15   1    1    3   150         SEGUNDO:=SIGUIENTE;
    16   1    1    2   155       END;
    17   1    1    0   164    END.
```

```
ISIS-II PASCAL-80 Compiler V2.0 invoked by:
>:F0:COMP.COD P17.PAS
```

Line	Seg	Proc	Lev	Disp	
1	1	1		1	PROGRAM FIBONACCI2;
2	1	1		3	VAR PEN,ULT,J,SUM,N:INTEGER;
3	1	1	0	0	BEGIN
4	1	1	1	0	PEN:=0;
5	1	1	1	5	ULT:=1;
6	1	1	1	10	J:=2;
7	1	1	1	15	WRITE('METER EL VALOR DE N:');
8	1	1	1	48	READ(N);
9	1	1	1	60	REPEAT
10	1	1	2	62	SUM:=ULT+PEN;
11	1	1	2	69	ULT:=SUM;
12	1	1	2	74	PEN:=ULT;
13	1	1	2	79	J:=J+1;
14	1	1	1	86	UNTIL SUM>N;
15	1	1	1	93	WRITELN('EL PRIMER TERMINO MAYOR QUE:',N,' ES:',J);
16	1	1	0	178	END.

```
ISIS-II PASCAL-80 Compiler V2.0 invoked by:
>:F0:COMP.COD P18.PAS
```

```
  Line Seg Proc Lev Disp

     1   1    1          1  PROGRAM NOTA2;
     2   1    1          3     VAR NOTA:ARRAY[1..5] OF REAL;
     3   1    1         13         NOMBRE:ARRAY[1..5] OF STRING[15];
     4   1    1         53         MEDIA,SUMA:REAL;
     5   1    1         57         I:INTEGER;
     6   1    1    0     0     BEGIN
     7   1    1    1     0       SUMA:=0.0;
     8   1    1    1    12       I:=1;
     9   1    1    1    17       REPEAT
    10   1    1    2    19           WRITE('INTRODUCIR EL',I,' NOMBRE:');
    11   1    1    2    75           READ(NOMBRE[I]);
    12   1    1    2    97           WRITE('INTRODUCIR LA',I,' NOTA:');
    13   1    1    2   151           READ(NOTA[I]);
    14   1    1    2   172           SUMA:=SUMA+NOTA[I];
    15   1    1    2   196           I:=I+1;
    16   1    1    1   204       UNTIL I>5;
    17   1    1    1   212       MEDIA:=SUMA/5.0;
    18   1    1    1   229       I:=1;
    19   1    1    1   234       REPEAT
    20   1    1    2   236           WRITE('NOMBRE DE QUIEN SE DESEA LA NOTA:');
    21   1    1    2   282           READ(NOMBRE[I]);
    22   1    1    2   304           WRITE('NOTA A COMPARAR:');
    23   1    1    2   333           READ(NOTA[I]);
    24   1    1    2   354           IF NOTA[I]>=MEDIA
    25   1    1    2   369             THEN WRITELN('ALUMNO:',NOMBRE[I],'    APTO')
    26   1    1    2   443             ELSE WRITELN('ALUMNO:',NOMBRE[I],'    NO APTO');
    27   1    1    2   514             I:=I+1;
    28   1    1    1   522       UNTIL I>5;
    29   1    1    0   530     END.
```

```
Line Seg Proc Lev Disp

   1    1    1          1  PROGRAM ORDENACION;
   2    1    1          3     VAR PASADA,N,MIN,I1,J,I,TRAS:INTEGER;
   3    1    1         10       X:ARRAY[1..100] OF INTEGER;
   4    1    1    0     0     BEGIN
   5    1    1    1     0       WRITE('METER EL VALOR DE N:');
   6    1    1    1    33       READ(N);
   7    1    1    1    45       BEGIN
   8    1    1    2    47         IF N>100
   9    1    1    2    50           THEN WRITELN('FORMATO DESBORDADO ,SOLO PUEDEN SER 100')
  10    1    1    2   114           ELSE FOR J:=1 TO N DO
  11    1    1    4   130                BEGIN
  12    1    1    5   132                  WRITE('X(',J,')=');
  13    1    1    5   170                  READLN(X[J]);
  14    1    1    4   198                END
  15    1    1    1   200       END(FORMATO);
  16    1    1    1   207       BEGIN(SE INICIA EL LAZO DE PASADAS)
  17    1    1    2   209         FOR PASADA:=1 TO N DO
  18    1    1    3   223          BEGIN
  19    1    1    4   225            I1:=PASADA;
  20    1    1    4   230            MIN:=PASADA;
  21    1    1    4   235            FOR I:=I1 TO N DO
  22    1    1    5   249             BEGIN(SE BUSCA EL MENOR VALOR)
  23    1    1    6   251               IF X[I]<X[MIN]
  24    1    1    6   274                 THEN MIN:=I;
  25    1    1    6   283             IF MIN<>PASADA
  26    1    1    6   286                          THEN BEGIN
  27    1    1    8   292                               TRAS:=X[MIN];
  28    1    1    8   307                               X[MIN]:=X[PASADA];
  29    1    1    8   331                               X[PASADA]:=TRAS;
  30    1    1    7   345                               END;
  31    1    1    5   347             END(FIN DE BUSQUEDA DEL MENOR VALOR)
  32    1    1    3   349          END;
  33    1    1    1   363       END;
  34    1    1    1   365       BEGIN(ESCRIBE VECTOR ORDENADO)
  35    1    1    2   367                            WRITELN('EL VECTOR ORDENADO ES');
  36    1    1    2   409                            FOR I:=1 TO N DO
  37    1    1    3   423                             BEGIN
  38    1    1    4   425                               WRITELN(X[I])
  39    1    1    3   455                             END
  40    1    1    1   455         END(VECTOR ORDENADO);
  41    1    1    0   462     END(PROGRAMA).
```

```
ISIS-II PASCAL-80 Compiler V2.0 invoked by:
>:F0:COMP.COD P20.PAS
```

Line	Seg	Proc	Lev	Disp	
1	1	1		1	PROGRAM MAYOR;
2	1	1		3	VAR A,B,X,Y:INTEGER;
3	1	1		7	
4	1	2		3	FUNCTION MAY(X,Y:INTEGER):INTEGER;
5	1	2	0	0	BEGIN
6	1	2	1	0	IF X>Y
7	1	2	1	3	THEN MAY:=X
8	1	2	1	9	ELSE MAY:=Y;
9	1	2	0	19	END(FUNCTION);
10	1	2	0	34	
11	1	1	0	0	BEGIN
12	1	1	0	0	
13	1	1	1	0	WRITE('INTRODUCIR LOS VALORES DE A Y B:');
14	1	1	1	45	READ(A,B);
15	1	1	1	67	WRITELN('EL MAYOR DE A=',A,' Y B=',B,' ES',MAY(A,B));
16	1	1	1	169	
17	1	1	0	169	END(PROGRAMA PRINCIPAL).

```
PROGRAM FICH1;
  LABEL 5;
  VAR
    TITULO:FILE OF STRING;
    I,B,R:CHAR;
    A:STRING;
  BEGIN
    WRITELN('QUIERES BORRAR LOS LIBROS QUE EXISTEN EN EL FICHERO');
    READLN(R);
    IF R IN ['s','S'] THEN
       BEGIN

         REWRITE(TITULO,':F0:TDATA.DAT');
         GOTO 5;

       END;

    RESET(TITULO,':F0:TDATA.DAT');

5:  WRITELN('QUIERES METER UN NUEVO LIBRO');
    READLN(I);
    IF I IN['s','S']THEN
       BEGIN
         WRITELN('ESCRIBA EL TITULO DEL LIBRO');
         READLN(A);
         TITULO↑:=A;
         PUT(TITULO);
         GOTO 5;
       END;
    RESET(TITULO);
    WRITELN('***************LA LISTA DE LIBROS QUE EXISTE ES***************');
    WRITELN;
    REPEAT
       WRITELN(TITULO↑);
       GET(TITULO);
    UNTIL EOF(TITULO);
    WRITELN;WRITELN;
    WRITELN('DESEA ACABAR YA');
    READLN(I);
    IF I IN['n','N']THEN
       BEGIN
         GET(TITULO);
         GOTO 5;
       END;
  END.
```

```
PROGRAM FICH2;

  LABEL 2,4;
  VAR
     NOMB,TLFO:FILE OF STRING;
     NOMBRE:STRING[30];
     A,B,C:CHAR;
     TLF:STRING[10];

BEGIN

   WRITE('QUIERES CREAR UN FICHERO:');
   READLN(C);
   WRITELN;
   IF C IN ['s','S'] THEN

      BEGIN
        REWRITE(NOMB,':F0:DATOS.DAT');
        REWRITE(TLFO,':F0:TELEF');
        GOTO 2;
      END;

4: RESET(NOMB,':F0:DATOS.DAT');
   RESET(TLFO,':F0:TELEF');
   WHILE NOT EOF(NOMB) DO
      BEGIN
        GET(NOMB);GET(TLFO);
      END;
2: WRITE('QUIERE METER ALGUN NOMBRE:');
   READLN(A);
   WRITELN;
   IF A IN ['s','S'] THEN

        BEGIN

            WRITE('ESCRIBA EL NOMBRE:');READLN(NOMBRE);
            NOMB↑:=NOMBRE;
            WRITE('  TELEFONO:');READLN(TLF);
            TLFO↑:=TLF;
            PUT(NOMB);
            PUT(TLFO);
            GOTO 2;

        END;

   WRITE('QUIERE SABER ALGUN TELEFONO:');
```

```
    READLN(B);WRITELN;
    IF B IN ['s','S'] THEN

          BEGIN

               RESET(NOMB);RESET(TLFO);
               WRITELN('DE QUIEN DESEA SABER EL TELEFONO');
               WRITELN;
               READ(NOMBRE);
               WRITELN;
               WHILE NOMB↑<>NOMBRE DO
                    BEGIN
                      GET(NOMB);
                      GET(TLFO);
                    END;
               WRITELN;
               WRITELN(NOMB↑,'    TLF:',TLFO↑);
               WRITELN;
          END;

    CLOSE(NOMB);CLOSE(TLFO);
    WRITE('DESEA ACABAR YA:');READLN(B);
    IF B IN ['n','N'] THEN
       BEGIN
         GOTO 4;
       END;
END.
```

```
PROGRAM FICH3;
  LABEL 2,4,6;
  TYPE
    NOMBRE=RECORD
             NOM:STRING[30];
           END;
    FECHA=RECORD
            DIA:1..31;
            MES:1..12;
            ANO:1000..2000;
          END;
    PERSONA=RECORD
              EDAD:STRING[3];
              NAC:FECHA;
              L←D←N:STRING[20];
              DOMICILIO:STRING[20];
            END;

  VAR
    PERSONAS:FILE OF NOMBRE;
    DP:FILE OF PERSONA;
    A,B,C,D:CHAR;
    PE:NOMBRE;
    DATS:PERSONA;

  PROCEDURE IMP(IMPRIMIR:BOOLEAN);
    BEGIN
      WRITELN('                LOS DATOS PERSONALES SON');WRITELN;
      WRITELN('*****************************************************');
      WRITELN('*                                                   *');
      WRITELN('* NOMBRE: ',PERSONAS↑.NOM:30,'          *');
      WRITELN('*                                                   *');
      WRITELN('* EDAD: ',DP↑.EDAD:3,'                                    *');
      WRITELN('*                                                   *');
      WRITELN('* FECHA DE NACIMIENTO: ',DP↑.NAC.DIA:3,
              DP↑.NAC.MES:3,DP↑.NAC.ANO:5,'                *');
      WRITELN('*                                                   *');
      WRITELN('* LUGAR DE NACIMIENTO: ',DP↑.L←D←N:20,'        *');
      WRITELN('*                                                   *');
      WRITELN('* DOMICILIO: ',DP↑.DOMICILIO:20,'                   *');
      WRITELN('*                                                   *');
      WRITELN('*****************************************************');
```

```
PROCEDURE NO(NOMBRE:BOOLEAN);
  BEGIN
   WRITE('NOMBRE: ');READLN(INPUT,PE.NOM);WRITELN;
   PERSONAS↑:=PE;
   PUT(PERSONAS);
  END;

PROCEDURE DA(DATOS:BOOLEAN);
  BEGIN
      WRITE('EDAD: ');READLN(INPUT,DATS.EDAD);WRITELN;
      WRITE('FECHA←DE←NACIMIENTO: ');READ(INPUT,DATS.NAC.DIA,DATS.NAC.MES,
                                          DATS.NAC.ANO);WRITELN;
      WRITE('LUGAR←DE←NACIMIENTO: ');READLN(INPUT,DATS.L←D←N);
      WRITELN;
      WRITE('DOMICILIO: ');READLN(INPUT,DATS.DOMICILIO);WRITELN;
      DP↑:=DATS;
  END(FIN DE INTRODUCIR DATOS);

PROCEDURE FICH(FICHERO:BOOLEAN);
  BEGIN
      RESET(PERSONAS,'PERS');RESET(DP,'DPERS');
      IF EOF(PERSONAS)=TRUE THEN
         BEGIN
           REWRITE(PERSONAS,'PERS');
           REWRITE(DP,'DPERS');
         END;
      WHILE NOT EOF(PERSONAS) DO
         BEGIN
           GET(PERSONAS);GET(DP);
         END;
  END;
BEGIN
 WRITE('QUIERES CREAR UN FICHERO ?:');READLN(C);WRITELN;
 IF C IN ['s','S'] THEN
    BEGIN
      REWRITE(PERSONAS,'PERS');
      REWRITE(DP,'DPERS');
      GOTO 2;
    END;
4:RESET(PERSONAS,'PERS');
  RESET(DP,'DPERS');
  WHILE NOT EOF(PERSONAS) DO
    BEGIN
```

```
        GET(PERSONAS);GET(DP);
      END;
2:WRITE('DESEA INTRODUCIR ALGUNA PERSONA ?:');READLN(A);
  IF A IN ['s','S'] THEN
      BEGIN
       NO(TRUE);
       DA(TRUE);PUT(DP);
      END;
6:RESET(PERSONAS);RESET(DP);
  WRITE('DESEA SABER LOS DATOS PERSONALES DE ALGUNA PERSONA?:');READLN(B);
  WRITELN;
  IF B IN ['s','S'] THEN
      BEGIN

            WRITE('INTRODICIR EL NOMBRE: ');READLN(PE.NOM);
            WRITELN;
            WHILE PE<>PERSONAS↑ DO
               BEGIN
                 GET(PERSONAS);GET(DP);
               END;
            IMP(TRUE);
            WRITE('DESEA CAMBIAR LOS DATOS ?:');READLN(D);WRITELN;
            IF D IN ['s','S'] THEN
               BEGIN
                 RESET(PERSONAS);RESET(DP);
                 GET(PERSONAS);
                 WHILE PE<>PERSONAS↑ DO
                    BEGIN
                      GET(PERSONAS);GET(DP);
                    END;
                 DA(TRUE);
                 PUT(DP);
                 GOTO 6;
               END;

      END;
  CLOSE(PERSONAS);CLOSE(DP);
  WRITE('DESEA ACABAR YA?:');READLN(D);WRITELN;
  IF D IN ['n','N'] THEN
      BEGIN
        GOTO 4;
      END;
END.
```

```
PROGRAM FICH4;
 LABEL 2,4;
 TYPE Q=STRING[50];
 VAR
    TITULO:FILE OF Q;
    C,E:CHAR;
    A:STRING[50];
    J:INTEGER;
    N:ARRAY[1..20] OF Q;

 PROCEDURE TI(TITU:BOOLEAN);
  LABEL 5;
  VAR D:CHAR;
   BEGIN
   5:WRITELN;
     WRITE('QUIERE METER ALGUN LIBRO ?:');READLN(D);
     WRITELN;
     IF D IN ['s','S'] THEN
        BEGIN
          WRITELN('ESCRIBA EL TITULO DEL LIBRO');WRITELN;
          READLN(A);
          TITULO↑:=A;
          PUT(TITULO);
          GOTO 5;
        END;
   END;

 PROCEDURE ALFB(ALFAB:BOOLEAN);
  VAR
    I1,MIN,I,PASADA,K:INTEGER;
    TRAS:STRING[50];
   BEGIN
     RESET(TITULO);
     J:=0;
     REPEAT
       J:=J+1;
       N[J]:=TITULO↑;
       GET(TITULO);
     UNTIL EOF(TITULO);
     FOR K:=1 TO J DO
      BEGIN
```

```
      I1:=K;
      MIN:=K;
      FOR I:=I1 TO J DO
          BEGIN
            IF NC I]<NC MIN] THEN
               BEGIN
                  MIN:=I;
               END;
          END;
      IF MIN<>K THEN
          BEGIN
            TRAS:=NC MIN];
            NC MIN]:=NC K];
            NC K]:=TRAS;
          END;
     END;
    J:=0;
    RESET(TITULO);
    REPEAT
      BEGIN
        J:=J+1;
        WRITELN(NC J]);
        GET(TITULO);
      END;
    UNTIL EOF(TITULO);
  END;

PROCEDURE LIS(LISTADO:BOOLEAN);
  BEGIN
    WRITELN;WRITELN;
    WRITELN('*************************************************************************');
    WRITELN;
    WRITELN('                              LA LISTA DE LIBROS ES    ');
    WRITELN;
    WRITELN('*************************************************************************');
    WRITELN;
```

```
    ALFB(TRUE);
  END;

BEGIN
  WRITE('QUIERES CREAR UN FICHERO ?:');READLN(C);
  WRITELN;
  IF C IN ['s','S'] THEN
     BEGIN
       REWRITE(TITULO,'LIBR');
       GOTO 2;
     END;
4:RESET(TITULO,'LIBR');
  WHILE NOT EOF(TITULO) DO
     BEGIN
       GET(TITULO);
     END;
2:TI(TRUE);
  LIS(TRUE);
  CLOSE(TITULO);
  WRITELN;WRITELN;
  WRITE('DESEA ACABAR YA ?:');READLN(E);
  IF E IN ['n','N'] THEN
     BEGIN
       GOTO 4;
     END;
END.
```

```
PROGRAM FICHE;
  LABEL 2,3,4;
  TYPE TITULO=STRING[50];
       AUTOR=STRING[50];
  VAR R,J:INTEGER;
      B,U,Y:CHAR;
      H:STRING[10];
      AUT:FILE OF AUTOR;
      TIT:FILE OF TITULO;
      N:ARRAY[1..100] OF AUTOR;
      AU:AUTOR;
      T:TITULO;
 PROCEDURE FICH(FICHE:BOOLEAN);
   LABEL 1;
  BEGIN
   WRITE('QUIERE CREAR UN FICHERO ?:');READLN(B);
   IF B IN ['s','S'] THEN
     BEGIN
       REWRITE(AUT,'AUTO');
       REWRITE(TIT,'TITE');
       GOTO 1;
     END;
   RESET(AUT,'AUTO');
   RESET(TIT,'TITE');
   WHILE NOT EOF(AUT) DO
     BEGIN
       GET(AUT);
       GET(TIT);
     END;
1:END;

 PROCEDURE TITU(TITULO:BOOLEAN);
  LABEL 6;
  VAR D:CHAR;
 BEGIN
   REPEAT
     WRITE('DESEA INTRODUCIR ALGUN LIBRO ?:');READLN(D);
     WRITELN;
   UNTIL D IN ['S','N'];
   CASE D OF
    'S':BEGIN
          WRITE('ESCRIBA EL TITULO: ');READLN(INPUT,T);
```

```
          RESET(TIT);RESET(AUT);
          REPEAT
            IF TIT↑=T THEN
              BEGIN
                 WRITELN;
                 WRITELN('EL CITADO LIBRO YA EXISTE');WRITELN;
                 EXIT(TITU);
              END;
            GET(TIT);GET(AUT);
          UNTIL EOF(TIT);
          TIT↑:=T;
          PUT(TIT);
          WRITELN('ESCRIBA EL NOMBRE DEL AUTOR(ES),(DESPUES DE CADA AUTOR PONER UNA COMA)');READLN(INPUT,A
          AUT↑:=AU;
          PUT(AUT);
        END;
    'N':BEGIN
          EXIT(TITU);
        END;
  END(CASE);
END;

PROCEDURE BU(BUSQUE:BOOLEAN);
 LABEL 4,5;
 VAR L,LO:INTEGER;
 BEGIN
   RESET(AUT);
   RESET(TIT);
   WRITE('NOMBRE DEL AUTOR (EN CASO DE VARIOS CON UNO BASTA):');READLN(AU);
   J:=0;
   REPEAT
      J:=J+1;
      N[J]:=AUT↑;
      R:=1;
    4:L:=POS(',',N[J]);
      IF L=0 THEN
        BEGIN
          GOTO 5;
        END;
      LO:=L-R;
```

```
      H:=COPY(NC J ],R,LO);
      FILLCHAR(NC J ],L+1,'0');
      IF H<>AU THEN
            BEGIN
              R:=R+(LO+1);
              GOTO 4;
            END;
      WRITE('AUTOR: ',AUT↑);WRITELN('     TITULO: ',TIT↑);
    5:GET(AUT);GET(TIT);
  UNTIL EOF(AUT);
END;

BEGIN
3:FICH(TRUE);
  TITU(TRUE);
  WRITE('QUIERE SABER ALGUN TITULO ?:');READLN(U);
  IF U IN ['s','S'] THEN
    BEGIN
      BU(TRUE);
    END;
  CLOSE(AUT);CLOSE(TIT);
  WRITE('DESEAS ACABAR YA ?:');READLN(Y);
  IF Y IN ['n','N'] THEN
    BEGIN
      GOTO 3;
    END;
END.
```

APENDICE B

MENSAJES DE ERROR DEL COMPILADOR

1: Error in simple type
2: Identifier expected
3: 'PROGRAM' expected
4: ')' expected
5: ':' expected
6: Illegal symbol (maybe missing ';' on the line above)
7: Error in parameter list
8: 'OF' expected
9: '(' expected
10: Error in type
11: '[' expected
12: ']' expected
13: 'END' expected
14: ';' expected
15: Integer expected
16: '=' expected
17: 'BEGIN' expected
18: Error in declaration part
19: Error in <field-list>
20: ',' expected
21: '.' expected

50: Error in constant
51: ':=' expected
52: 'THEN' expected
53: 'UNTIL' expected
54: 'DO' expected
55: 'TO' or 'DOWNTO' expected in for statement
56: 'IF' expected
57: 'FILE' expected
58: Error in <factor> (bad expression)
59: Error in variable

101: Identifier declared twice
102: Low bound exceeds high bound
103: Identifier is not of the appropriate class
104: Undeclared identifier
105: Sign not allowed
106: Number expected
107: Incompatible subrange types
108: File not allowed here
109: Type must not be real
110: <tagfield> type must be scalar or subrange
111: Incompatible with <tagfield> part
112: Index type must not be real
113: Index type must be a scalar or a subrange
114: Base type must not be real
115: Base type must be a scalar or a subrange
116: Error in type of standard procedure parameter
117: Unsatisfied forward reference
118: Forward reference type identifier in variable declaration
119: Respecified parameters for a forward declared procedure
120: Function result type must be scalar, subrange, or pointer
121: File value parameter not allowed

122: A forward declared function's result type can not be respecified
123: Missing result type in function declaration
124: F-format for reals only
125: Error in type of standard procedure parameter
126: Number of parameters does not agree with declaration
127: Illegal parameter substitution
128: Result type does not agree with declaration
129: Type conflict of operands
130: Expression is not of set type
131: Tests on equality allowed only
132: Strict inclusion not allowed
133: File comparison not allowed
134: Illegal type of operand(s)
135: Type of operand must be boolean
136: Set element type must be scalar or subrange
137: Set element types must be compatible
138: Type of variable is not array
139: Index type is not compatible with the declaration
140: Type of variable is not record
141: Type of variable must be file or pointer
142: Illegal parameter solution
143: Illegal type of loop control variable
144: Illegal type of expression
145: Type conflict
146: Assignment of files not allowed
147: Label type incompatible with selecting expression
148: Subrange bounds must be scalar
149: Index type must be integer
150: Assignment to standard function is not allowed
151: Assignment to formal function is not allowed
152: No such field in this record
153: Type error in read
154: Actual parameter must be a variable
155: Control variable cannot be formal or non-local
156: Multidefined case label
157: Too many cases in case statement
158: No such variant in this record
159: Real or string tagfields not allowed
160: Previous declaration was not forward
161: Again forward declared
162: Parameter size must be constant
163: Missing variant in declaration
164: Substitution of standard proc/func not allowed
165: Multidefined label
166: Multideclared label
167: Undeclared label
168: Undefined label
169: Error in base set
170: Value parameter expected
171: Standard file was redeclared
172: Undeclared external file
174: Pascal function or procedure expected
190: Previous declaration was segment declaration
191: Separate declaration valid only on global level
192: Separate procedure must be a segment
193: Comment or heading valid only on global level.

201: Error in real number - digit expected
202: String constant must not exceed source line
203: Integer constant exceeds range

250: Too many scopes of nested identifiers
251: Too many nested procedures or functions
252: Too many forward references of procedure entries
253: Procedure too long
259: Expression too complicated
261: Too many segment declarations

300: Division by zero
302: Index expression out of bounds
303: Value to be assigned is out of bounds
304: Element expression out of range
398: Implementation restriction
399: Implementation restriction

400: Illegal character in text
401: Unexpected end of input
402: Error in writing code file, not enough room
403: Error in reading include file
404: Error in writing list file, not enough room
405: Error in reading globals file
406: Error in writing globals file
407: Wrong version of globals file

APENDICE C

MENSAJES DE ERROR DEL RUN-TIME

APENDICE C.

MENSAJE DE ERROR DEL RUN-TIME

Cuando estamos ejecutando un programa y se produce un error en un procedimiento, el sistema de 'run-time' indica que se ha producido un fallo en el procedimiento que había sido llamado mediante un mensaje en la pantalla. Después que el mensaje ha sido visualizado se fuerza al programa a terminar. Cada error del 'run-time' causa un mensaje especifico para que sea visualizado, junto con el múmero de segmento, número de procedimiento, y el número de la instrucción. Estos números anteriores hacen referencia al listado del compilador de tal manera que si poseemos dicho listado será sumamente fácil averiguar el error.

Cuando en error de 'run-time' tiene lugar se visualizará en la pantalla lo siguiente:

```
error message

segment:    s
procedure:  p
instruction:i
```

donde s,p,y i son enteros, y error message es uno de los que sigue:

```
Stack overflow (4)
Value range error (1)
```

Exit from uncalled procedure (3)

Integer overflow (5)

String overflow (13)

Divide by zero (60

NIL pointer reference (7)

System I/O error (10)

Floating point error (12)

Unimplemented instruction (11)

Program interrupted by user(Interrupt) (8)

ISIS-II Error e donde'e' representa un error el el Isis II.

Los números que siguen a cada mensaje es el número del error.

APENDICE D

MENSAJES DE ERROR DEL ISIS-II

0 No error detected.
1 Limit of 19 buffers exceeded.
2 AFTN does not specify an open file.
3 Attempt to open more than six files simultaneously.
4 Illegal filename specification.
5 Illegal or unrecognized device specification in pathname.
6 Attempt to write to a file open for input.
7 Operation aborted; insufficient disk space.
8 Attempt to read from a file open for output.
9 No more room in disk directory.
10 Pathnames do not specify the same disk.
11 Cannot rename file; name already in use.
12 Attempt to open a file already open.
13 No such file.
14 Attempt to open for writing or to delete or rename a write-protected file.
15 Attempt to load into ISIS-II area or buffer area.
16 Illegal format record.
17 Attempt to rename/delete a non-disk file.
18 Unrecognized system call.
19 Attempt to seek on a non-disk file.
20 Attempt to seek backward past beginning of file.
21 Attempt to rescan a non-lined file.
22 Illegal ACCESS parameter to OPEN or access mode impossible for file specified.
23 No filename specified for a disk file.
24 Disk error (see below).
25 Incorrect specification of echo file to OPEN.
26 Incorrect SWID parameter in ATTRIB system call.
27 Incorrect MODE parameter in SEEK system call.
28 Null file extension.
29 End of file on console input.
30 Drive not ready.
31 Attempted seek on write-only (output) file.
32 Can't delete an open file.
33 Illegal system call parameter.
34 Bad RETSW argument to LOAD.
35 Attempt to extend a file opened for input by seeking past end-of-file.

201 Unrecognized switch.
202 Unrecognized delimiter character.
203 Invalid command syntax.
204 Premature end-of-file.
206 Illegal disk label.
207 No END statement found in input.
208 Checksum error.
209 Illegal records sequence in object module file.
210 Insufficient memory to complete job.
211 Object module record too long.
212 Bad object module record type.
213 Illegal fixup record specified in object module file.
214 Bad parameter in a SUBMIT file.
215 Argument too long in a SUBMIT invocation.
216 Too many parameters in a SUBMIT invocation.
217 Object module record too short.
218 Illegal object module record format.
219 Phase error in LINK.
220 No end-of-file record in object module file.
221 Segment overflow during Link operation.
222 Unrecognized record in object module file.
223 Fixup record pointer is incorrect.
224 Illegal record sequence in object module file in LINK.
225 Illegal module name specified.
226 Module name exceeds 31 characters.
227 Command syntax requires left parenthesis.
228 Command syntax requires right parenthesis.
229 Unrecognized control specified in command.
230 Duplicate symbol found.
231 File already exists.
232 Unrecognized command.
233 Command syntax requires a "TO" clause.
234 File name illegally duplicated in command.
235 File specified in command is not a library file.
236 More than 249 common segments in input files.
237 Specified common segment not found in object file.
238 Illegal stack content record in object file.
239 No module header in input object file.
240 Program exceeds 64K bytes.

APENDICE E

CARACTERISTICAS DEL PASCAL-80

CARACTERISTICAS PRINCIPALES DEL PASCAL-80

- El máximo número de caracteres en una variable de tipo String es de 255.

- El máximo número de elementos en un Set es de 255+16=4080.

- El máximo número de funciones y procedimiento en un segmento es de 127.

- El máximo número de 'segment procedure' y 'segment function' que puede ser usado en un programa con particiones es de 7.

- El máximo número de bytes de un código objeto en un procedimiento o una función es de 1200.

- El máximo tamaño de un fragmento de datos es de 16383 palabras.

PRINCIPALES DIFERENCIAS ENTRE EL PASCAL-80 Y EL PASCAL STANDART

- En Pascal-80, la sentencia 'goto' solamente puede ser usada con etiquetas locales;por ejemplo etiquetas declarada en el mismo bloque. Un 'goto' no puede hacer referencia a una etiqueta declarada en un bloque cerrado como si lo puede ser en un Pascal Standard. Sin embargo el Pascal-80 soluciona este incoveniente con el procedimiento predeclarado 'exit' que incluso proporciona más capacidad.

- El Pascal-80 no soporta los procedimientos stan-

dard 'pack' y 'unpack'.

- El Pascal-80 no soporta la construcción en la cual procedimiento y funciones pueden ser declarados como parametro de un procedimiento o funcióm.

- El Pascal-80 no soporta el procedimiento predeclarado 'dispose'. Sin embargo, el procedimiento 'mark' y 'release' pueden ser usasos para aproximarnos a la acción de 'dispose'.

BIBLIOGRAFIA UTILIZADA

1. Pressman, Roger S. : "Software Engineering: A Practitioner's Approach", McGraw-Hill, New York, 1982.
2. Tremblay, y Bunt.: "Introducción a la ciencia de las computadoras", McGraw-Hill, New York, 1981.
3. Sanchis Llorca, y Morales Lozano.:" Programación con el lenguaje Pascal", Paraninfo, Madrid, 1982.
4. Wirth, N.: "Algoritmos+estructuras de datos=Programas", Prentice-Hall, 1976.
5. "Pascal-80 User's Guide.
6. Davidson, G.:"Programas prácticos en Pascal", Osborne/ McGraw-Hill, Madrid, 1983.
7. Revista informática Ordenador Personal. Nº14, Abril 83, pag 77; Nº15, Mayo 83, pag47; Nº16, Junio 83, pag77; Nº18, Agosto 83, pag 17.
8. Tremblay, Bunt, y Opseth.: "Pascal Structured", McGraw-Hill, New York, 1980.

BIBLIOGRAFIA RECOMENDADA

1. Wirth, N. and C.A.R. Hoare: "A Contribution of the Development of ALGOL". vol.9, no.6, June 1966.

2. Wirth, N.: "The Design of a Pascal Compiler", Software-Practise and Experience, vol.1 no.4, April 1971.

3. Welsh, J. and Quinn, C.: "A Pascal Compiler for the ICL 1900 Series Computer", Software-Practique and Experience, vol.2 no.1, Feb. 1972.

4. Wirth, N.: "The Programming Language Pascal", Acta Informática, vol.1, no.1 Jan. 1971.

5. Jensen, K. and Wirth, N.: "Pascal User Manual and Report", 2nd. ed., Springer-Verlag, New York, 1975.

6. Lecarme, O.: "Structured Programming, Programming Teaching and the Language Pascal", SIGPLAN Notices, vol.9, no.7. July, 1974.

7. Schillington, K.: "Structure: the key to Pascal's problemsolving power", Datamation, July 1979.

8. Bowles, K.L.: "Microcomputer problem solving using Pascal", pringer-Verlag, New York 1977.

9. Scheneider, G.M., Weingart, S.W. and Perlman, D.N.: "An Introduction to programming and problem solving with Pascal", John Wiley and Sons, New York 1978.

10. Grogono, P.: "Programming in Pascal", Addison-Wesley 1978.

11. Fletcher, D.: "Users love it, vendors are getting the message, and standards are on the way, Datamation, July 1979.

12. Posa, J.G.: "Pascal people unhappy over standard", Electronics. Feb. 1979.

13. Ravenel, B.W.: "Toward a Pascal Standard", Computer, April 1979.

14. Conrad, M.: "A high-level language for mocros and minis", Datamation, July 1979.

15. Schindler, M.: "Pascal marches into the 1980's thru Portal, a modular upgrade", Electronic Desing, June 1979.

16. Lienhard, M.: "Trends in industrial software", Journees d'electronique 1979.

17. Glass, R.L.: "From Pascal to Pebbleman... and Beyond", Datamation, July 1979.

18. Ichbiah, J.D., Heliard, J.C., Roubine, O., Barnes, J.G.Krieg-Brueckner, B. and Wichmann, B.A.: "Rationale for Desinn of the ADA Programming Language", SIGPLAN Notices, vol.14, no.6 June 1979.

19. Van der Boos, J.: "Comments on ADA process communication", SIGPLAN Notices, vol.15 no.6, June 1980.

20. Wirth, N.: "The use of Modula", Software-Practice and Experience, vol.7 1979.