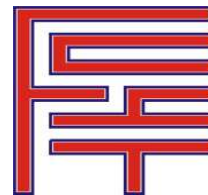




**UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
CARRERA DE INGENIERÍA CIVIL**



**MATERIAL DE APOYO DIDÁCTICO DE LA
ENSEÑANZA – APRENDIZAJE EN LA ASIGNATURA**

**“COMPUTACIÓN PARA INGENIERÍA”
CIV-217**

“TEXTO ESTUDIANTE Y TEXTO DE EJERCICIOS”

*TRABAJO DIRIGIDO, POR ADSCRIPCIÓN, PRESENTADO EN CUMPLIMIENTO
PARCIAL DE LOS REQUISITOS PARA OPTAR AL DIPLOMA ACADÉMICO DE*

LICENCIATURA EN INGENIERÍA CIVIL

Presentado Por:

Germán Camacho Choquevilca

Mauricio Andia Berazain

Tutor:

Ing. Msc. Oscar Alberto Zabalaga Montaña

**COCHABAMBA – BOLIVIA
Mayo, 2007**

DEDICATORIA:

*A mis maravillosos padres y a mis
hermanas por brindarme su apoyo
incondicional.*

Mauricio Andia Berazain

*A mi madre y a mis
hermanos por brindarme su apoyo
incondicional.*

Germán Camacho Choquevillca

AGRADECIMIENTOS

A Dios por la luz y guía espiritual en el crecimiento tanto intelectual como moral.

A nuestros padres por el amor que nos brindan sus desvelos, sus sacrificios, su amistad y compañerismo.

A nuestros hermanos por su colaboración desinteresada.

Al Ing. M.Sc. Oscar Alberto Zabalaga Montaña por sus consejos y por compartir sus conocimientos y experiencia.

A los tribunales designados para la revisión del presente trabajo de investigación la M.Sc. Ma. Leticia Blanco Coca, Ing. M.Sc. Oscar Florero Ortuño y el Ing. Juan Carlos Reinaga Vargas.

En especial a la M.Sc. Ma. Leticia Blanco Coca por sus sabios consejos y por esas horas dedicadas a esta tarea en búsqueda de una mejor conclusión del proyecto de grado.

A la universidad por abrirnos las puertas y cobijarnos hasta la culminación de nuestros estudios.

Finalmente a nuestros amigos y todos aquellos compañeros que nos acompañaron y ayudaron durante la carrera, por su buen humor, apoyo y compañía.

¡Muchas Gracias!

FICHA RESUMEN

La asignatura Computación para Ingeniería – CIV 217 corresponde al tercer semestre de la Carrera de Ingeniería Civil de la Universidad Mayor de San Simón.

En los últimos tiempos, la Universidad Mayor de San Simón ha establecido la necesidad de mejorar el proceso de aprendizaje, a través de la realización de textos que permitan mejorar y apoyar el desempeño del alumno. Es por tal razón, que la elaboración de este texto referido a la materia de “Computación para Ingeniería” surge como respuesta a la necesidad del estudiante de poder disponer de un texto adecuado, en un lenguaje simple y que cumpla cabalmente con las exigencias del contenido de la materia.

El presente documento es el producto de la investigación de abundante bibliografía sintetizada en un volumen que engloba lo más importante y útil para el aprendizaje de la materia.

El texto se divide en diez capítulos. El primer capítulo contempla una introducción a la computación desarrollando en el mismo una reseña histórica sobre la historia de la computación hasta actualidad. En el segundo capítulo se exponen los procesos iniciales, secuenciales utilizados para el desarrollo de diagramas de flujo en la resolución de diferentes tipos de problemas, que introducen al estudiante en la lógica de programación. El tercer capítulo desarrolla una introducción a la programación con Delphi en el cual se muestran todas las características del entorno del programa en si y la estructura del lenguaje que utiliza el mismo. A partir del cuarto capítulo hasta el octavo se desarrolla los conceptos básicos de programación en el lenguaje elegido. En el noveno capítulo se desarrolla el manejo de archivos y sus distintas operaciones. El décimo capítulo comprende el manejo de gráficos y las distintas propiedades. El texto de ejercicios que presenta:

- Ejercicios resueltos, propuestos y de aplicación a Ingeniería Civil.

INDICE

CAPITULO 1

INTRODUCCIÓN A LA COMPUTACIÓN

1. CONCEPTOS INTRODUCTORIOS	9
1.1 ORIGEN DEL COMPUTADOR	9
1.2 DEFINICIONES DE COMPUTADORA	12
1.3 LA PRIMERA COMPUTADORA	13
1.4 GENERACIONES DE COMPUTADORAS	15
1.4.1 PRIMERA GENERACIÓN	15
1.4.2 SEGUNDA GENERACIÓN	18
1.4.3 TERCERA GENERACIÓN	19
1.4.4 CUARTA GENERACIÓN	21
1.4.5 QUINTA GENERACIÓN	23
1.5 TIPOS DE COMPUTADORAS	24
1.5.1 ANÁLOGA	24
1.5.2 DIGITAL	24
1.6 CATEGORÍAS DE COMPUTADORAS	25
1.6.1 SUPERCOMPUTADORA	25
1.6.2 MINICOMPUTADORA	25
1.6.3 MICROCOMPUTADORA	26
2. HARDWARE (ARQUITECTURA DEL COMPUTADOR)	26
3. FUNCIONAMIENTO DE UNA COMPUTADORA	26
3.1. ENTRADA	27
3.2. UNIDAD DE PROCESAMIENTO (UP)	29
3.3. MEMORIA	30
3.4. SALIDA	35
3.5 OTROS PERIFÉRICOS	37
4. SOFTWARE	38
4.1 CLASIFICACIÓN DEL SOFTWARE	39

4.1.1 SOFTWARE DE BASE _____	39
4.1.2 LENGUAJES DE PROGRAMACIÓN _____	39
4.1.3 SOFTWARE DE USO GENERAL _____	40
4.1.4 SOFTWARE DE APLICACIÓN _____	40
4.2 UNIDAD DE INFORMACIÓN _____	40
4.2.1 BIT (DIGITO BINARIO) _____	40
4.2.2 BYTE _____	40
4.2.3 CONCEPTO DE REGISTRO _____	41
4.2.4. CONCEPTO DE ARCHIVOS _____	41
4.3. BIOS _____	41
4.4. CATEGORÍA DE LOS SISTEMAS OPERATIVOS (SO) _____	43
4.4.1. SISTEMA OPERATIVO MULTITAREAS _____	43
4.4.2. SISTEMA OPERATIVO MONOTAREAS _____	44
4.4.3. SISTEMA OPERATIVO MONOUSUARIO _____	44
4.4.4. SISTEMA OPERATIVO MULTIUSUARIO _____	44
4.5. SISTEMAS OPERATIVOS MICROSOFT WINDOWS _____	46
5. LENGUAJES DE PROGRAMACION _____	49
5.1 LENGUAJES DE PROGRAMACIÓN Y SU MIGRACIÓN AL COMPUTADOR _____	50
5.2 EVOLUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN _____	51
5.3 LENGUAJE DE MÁQUINA _____	53
5.4 LENGUAJE ENSAMBLADOR _____	53
5.5 LENGUAJES DE ALTO NIVEL _____	54
5.6 COMPILACIÓN DEL LENGUAJE DE ALTO NIVEL _____	56
5.6.1 COMPILADOR: _____	56
5.6.2 PARTES DE UN COMPILADOR _____	58
5.6.3. TIPOS DE COMPILADORES _____	58
5.7 INTÉRPRETE _____	59
6. BIBLIOGRAFIA _____	61

CAPITULO 2

PROCESOS LÓGICOS

1. INTRODUCCION _____	62
2. ALGORITMO _____	62
2.1 CONCEPTO _____	62
2.2 ESTRUCTURA BÁSICA: _____	63
2.3 IMPLEMENTACIÓN _____	64
3 PSEUDOCODIGO _____	65
4. DIAGRAMAS DE FLUJO _____	66
4.1 ELEMENTOS DE DIAGRAMACIÓN _____	67
4.2 ESTRUCTURAS DE DIAGRAMAS _____	72
4.3 EXPRESIONES _____	76
4.4 REGLAS _____	77
4.5 DISEÑO Y ELABORACIÓN DE DIAGRAMAS DE FLUJO _____	77
4.6 PRUEBA DE ESCRITORIO _____	78
5. CONCLUSION _____	79
6. BIBLIOGRAFIA _____	80

CAPITULO 3

INTRODUCCION A LA PROGRAMACION CON DELPHI

1. EVOLUCION DEL DELPHI _____	81
1.1 LENGUAJE PASCAL _____	81
1.2 EVOLUCIÓN DEL LENGUAJE OO PASCAL _____	82
2. METODOLOGIAS DE PROGRAMACION _____	83
2.1 METODOLOGÍA DE PROGRAMACIÓN ESTRUCTURADA _____	84
2.2 METODOLOGÍA DE PROGRAMACIÓN ORIENTADA AL OBJETO _	85
2.3 METODOLOGÍA DE PROGRAMACIÓN ORIENTADA AL EVENTO	86
2.4 METODOLOGÍA DE PROGRAMACIÓN FUNCIONAL _____	86
2.5 METODOLOGÍA DE PROGRAMACIÓN LÓGICA _____	86
3. ESTRUCTURA DE PROGRAMACION EN OBJECT PASCAL (DELPHI)	87

3.1 LA CLÁUSULA PROGRAM _____	89
3.2 PARTE DE DECLARACIÓN DE USO DE UNIDADES _____	89
3.3 SECCIÓN DE DECLARACIONES _____	90
4. HERRAMIENTAS DE DESARROLLO _____	90
4.1 LENGUAJE DE PROGRAMACIÓN _____	91
4.2 ENTORNO DE DESARROLLO (IDE) _____	91
4.3 BIBLIOTECA DE COMPONENTES _____	96
5. DISEÑO DE (LOS) FORMULARIOS (S) DE APLICACIONES _____	96
5.1 CARACTERÍSTICAS DE UN FORMULARIO _____	96
5.2 MÉTODOS DE UN FORMULARIO _____	97
6. ORGANIZACION DEL PROGRAMA _____	98
6.1 ARCHIVOS FUENTE DE DELPHI _____	99
6.2 OTRO ARCHIVOS USADOS PARA CONSTRUIR APLICACIONES _____	99
6.3 ARCHIVOS GENERADOS EN LA COMPILACIÓN _____	100
6.4 ARCHIVOS GENERADOS EN UN PROYECTO DELPHI _____	101
7. BIBLIOGRAFIA _____	102

CAPITULO 4

CONSTANTES VARIABLES Y TIPOS DE DATOS

1. IDENTIFICADORES _____	103
2. CONSTANTES Y VARIABLES _____	103
2.1 DECLARACIÓN DE VARIABLES _____	104
2.2 ASIGNACIÓN _____	105
2.3 DECLARACIÓN DE CONSTANTES _____	105
3. TIPOS DE DATOS _____	105
4. OPERADORES _____	107
5. FUNCIONES MATEMATICAS _____	109
6. COMENTARIOS _____	110
7. OTROS TIPOS DE DATOS _____	110
7.1 ARREGLOS _____	110
7.1.1 ARREGLOS UNIDIMENSIONALES (VECTORES) _____	111

7.1.2 TIPO DE ARREGLOS BIDIMENSIONALES (MATRICES) _____	111
7.1.3 REPRESENTACIÓN VISUAL DE ARREGLOS _____	112
7.1.4 DECLARACIÓN DE ARREGLOS _____	112
7.2 REGISTROS _____	114
7.3 CONJUNTOS (SET) _____	115
7.3.1 DECLARACIÓN DE TIPOS DE DATOS CONJUNTO _____	115
7.3.2 RELACIÓN DE PERTENENCIA EN CONJUNTOS _____	116
8. BIBLIOGRAFIA _____	117

CAPITULO 5

INSTRUCCIONES DE CONTROL

1. INSTRUCCION IF...THEN...ELSE _____	118
2. ANIDAMIENTOS _____	119
3. SELECCION MULTIPLE Y USO DE LA INSTRUCCION CASE...OF _____	120
4. BIBLIOGRAFIA _____	121

CAPITULO 6

CICLOS ITERATIVOS

1. INTRODUCCION _____	122
2. INSTRUCCIÓN REPEAT UNTIL _____	122
3. INSTRUCCIÓN WHILE DO _____	122
4. INSTRUCCIÓN FOR ... TO... DO _____	123
5. BIBLIOGRAFIA _____	125

CAPITULO 7

INSTRUCCIONES DE TRANSFERENCIA DE CONTROL

1. INTRODUCCION _____	126
2. INSTRUCCION GOTO _____	126
3. INSTRUCCION BREAK _____	127
4. INSTRUCCION CONTINUE _____	127
5. INSTRUCCION EXIT _____	127

6. INSTRUCCION HALT _____	128
7. INSTRUCCIÓN TRY/EXCEPT _____	128
8. INSTRUCCIÓN TRY/FINALLY _____	128
9. EXCEPCIONES ANIDADAS _____	129
10. CATEGORÍAS Y CLASES DE EXCEPCIONES _____	130
11. BIBLIOGRAFIA _____	132

CAPITULO 8

UNIDADES PROCEDIMIENTOS Y FUNCIONES

1. INTRODUCCION _____	133
2. PROCEDIMIENTOS – FUNCIONES _____	133
3. DECLARACION DE PROCEDIMIENTOS Y FUNCIONES _____	133
3.1 DECLARACIÓN DE PROCEDIMIENTOS _____	134
3.2 DECLARACIÓN DE FUNCIONES _____	134
4. LLAMADAS A PROCEDIMIENTOS Y FUNCIONES _____	136
5. PARAMETROS POR VALOR Y POR REFERENCIA _____	136
6. PROYECTOS Y UNIDADES _____	138
7. ENTRADA/SALIDA _____	140
7.1 ENTRADA DE INFORMACIÓN (DATOS) _____	140
7.2 SALIDA DE INFORMACIÓN (RESULTADOS) _____	141
7.2.1. SALIDA EN PANTALLA (VISUAL) _____	141
7.2.2. SALIDA DE INFORMACIÓN A IMPRESORA _____	142
8. BIBLIOGRAFIA _____	144

CAPITULO 9

MANEJO DE ARCHIVOS

1. INTRODUCCION _____	145
2. TIPOS DE ARCHIVOS EN GENERAL _____	145
3. TIPOS DE ACCESO A UN ARCHIVO _____	146
3.1 ACCESO SECUENCIAL _____	146
3.2 ACCESO DIRECTO _____	146

4. TIPOS DE ARCHIVOS EN OBJECT PASCAL _____	147
5. ARCHIVOS DE TEXTO (SECUENCIALES) _____	148
5.1 CREACIÓN DE ARCHIVOS DE TEXTO CON UN EDITOR _____	148
5.2 FUNCIONES EOLN/EOF _____	148
5.3 ARCHIVOS TIPO CHAR _____	149
5.4 TRATAMIENTO DE ARCHIVOS DE TEXTO _____	149
5.4.1 DECLARACIÓN DE UN ARCHIVO DE TEXTO _____	149
5.4.2 APERTURA DE UN ARCHIVO _____	150
5.4.3 ESCRITURA DE UN ARCHIVO _____	151
5.4.4 LECTURA DE UN ARCHIVO _____	152
5.4.5 AÑADIR DATOS UN ARCHIVO DE TEXTO _____	152
6. ESTRUCTURA DE UN ARCHIVO CON TIPO (BINARIO) _____	153
6.1 TRATAMIENTO DE ARCHIVOS DE ACCESO ALEATORIO _____	153
6.2 DECLARACIÓN DE UN TIPO DE DATOS ARCHIVO BINARIO _____	154
6.3 ASIGNACIÓN DE ARCHIVOS _____	154
6.4 APERTURA DEL ARCHIVO _____	155
6.5 OPERACIONES DE LECTURA, ESCRITURA Y FIN DE ARCHIVO _____	156
6.6 CIERRE DE UN ARCHIVO _____	156
7. MANTENIMIENTO DE ARCHIVOS ALEATORIOS _____	157
7.1 OPERACIONES DE ACCESO AL ARCHIVO _____	157
7.2 OPERACIONES PARA CONSULTAS _____	158
7.3 ACTUALIZACIÓN DE REGISTROS _____	160
7.3.1 AÑADIDO DE NUEVOS REGISTROS AL ARCHIVO _____	160
7.3.2 MODIFICACIÓN DE LOS DATOS DE UN REGISTRO YA EXISTENTE _____	162
7.3.3 BAJAS LÓGICAS DE REGISTROS _____	163
7.3.4 BAJAS FÍSICAS O ELIMINACIÓN DEFINITIVA DE REGISTROS _____	164
8. ORDENACIÓN DE ARCHIVOS _____	166
9. FUSIÓN O MEZCLA DE ARCHIVOS _____	166
10. BIBLIOGRAFIA _____	169

CAPITULO 10

MANEJO DE GRAFICOS

1. INTRODUCCION _____	170
2. GRÁFICOS EN DELPHI _____	170
2.1 GRÁFICOS CON IMAGE _____	170
2.2 GRÁFICOS CON SHAPE _____	171
2.3 EL COMPONENTE BEVEL _____	173
2.4 GRÁFICOS CON CANVAS _____	173
3. BIBLIOGRAFIA _____	177
 TEXTO DE EJERCICIOS RESUELTOS Y PROPUESTOS _____	 178
ANEXOS _____	282

CAPITULO 1

INTRODUCCIÓN A LA COMPUTACIÓN

1. CONCEPTOS INTRODUCTORIOS

1.1 Origen del Computador

La razón de repetir muchas operaciones sencillas para completar grandes proyectos no es nueva. Todo comenzó con máquinas destinadas a manejar números, es así como nos remitimos a el Ábaco figura 1, inventado por los babilonios allá por el año 1000 A.C. Utilizado sobre todo por los chinos para la realización de operaciones sencillas, esta formado por una tablilla con una serie de cuentas que sirven para efectuar sumas y restas.



Figura 1. Ábacos (<http://www.LaComputaciónenelTiempo.htm>)

Los faraones del antiguo Egipto utilizaron este concepto para construir las grandes pirámides ; cada uno de los esclavos movían bloques una pequeña distancia centenares de veces en sucesión. Las figuras pueden ser dibujadas repitiendo pequeños puntos de colores diferentes. Este concepto de reducir

laboriosas tareas a una serie de tareas repetitivas sencillas es la idea fundamental sobre la computadora.

En 1621 la primera regla deslizante fue inventada por el matemático inglés William Oughtred. La regla deslizante se llamó "Círculos de Proporción" era un juego de discos rotatorios que se calibraron con los logaritmos de Napier. Uno de los primeros aparatos de la informática analógica, la regla deslizante se usó normalmente (en un orden lineal) hasta comienzos de 1970, cuando calculadoras portátiles comenzaron a ser más populares.

A mediados del siglo XVII (1642) a sus 18 años el filósofo, matemático y teólogo francés Pascal tuvo una idea de la primera calculadora mecánica, para lo cual utilizó una serie de engranajes o ruedas dentadas que le permitían sumas y restas. En 1666 la primera máquina de multiplicar se inventó por Sir Samuel Morland, entonces sirviente a la corte del Rey Charles II de Inglaterra. El aparato constó de una serie de ruedas, cada una representaba, decenas, cientos, etc. Un alfiler del acero movía los diales para ejecutar los cálculos. A diferencia de la Pascalina figura 2, el aparato no tenía avance automático de columnas.

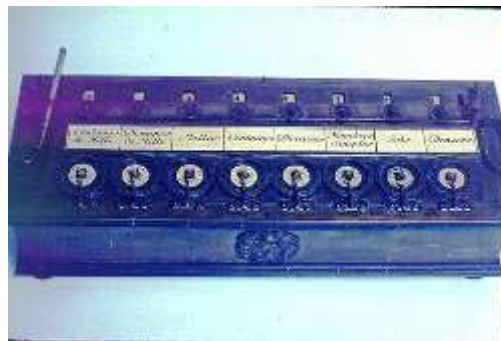


Figura 2. Pascalina (<http://www.LaComputaciónenelTiempo.htm>)

Años más tarde, en 1673, Gottfried Von Leibnitz perfeccionó los estudios de Pascal, y llegó a construir una máquina que no solo sumaba y restaba, sino que también multiplicaba, dividía e incluso calculaba raíces cuadradas.

En 1769 el Jugador de Ajedrez Automata fue inventado por Barón Empellen, un noble húngaro. El aparato y sus secretos se los dio a Johann Nepomuk Maelzel, un inventor de instrumentos musicales, quien recorrió Europa y los Estados Unidos con el aparato, a finales del siglo XVIII y inicios del siglo XIX. Pretendió ser una máquina pura, el Automaton incluía un jugador de ajedrez "robótico". El Automaton era una sensación dondequiera que iba, pero muchos comentaristas, incluso Edgar Allen Poe, escribió críticas detalladas diciendo que esa era una "máquina pura." En cambio, generalmente, siempre se creyó que el aparato fue operado por un humano oculto en el armario debajo del tablero de ajedrez. El Automata se destruyó en un incendio en 1856.

Se inventó la primera máquina lógica en 1777 por Charles Mahon, el Conde de Stanhope. El "Demostrador Lógico" era un aparato tamaño bolsillo que resolvía silogismos tradicionales y preguntas elementales de probabilidad. Mahon es el precursor de los componentes lógicos en computadoras modernas.

En 1790 Joseph-Marie Jacquard (1572-1834) utilizó tarjetas perforadas para controlar un telar figura 3. El "Jacquard Loom" se inventó en 1804 por Joseph-Marie Jacquard. La idea de Jacquard, revolucionó el hilar de seda, esta formaba la base de muchos aparatos de la informática e ideologías de la programación.



Figura 3. Joseph-Marie Jacquard (<http://es.wikipedia.org/wiki/Computadora>)

La primera calculadora de producción masiva se distribuyó, empezando en 1820, por Charles Thomas de Colmar. El "Aritmómetro" de Colmar operaba usando una variación de la rueda de Leibniz. Más de mil aritmómetros se vendieron y eventualmente recibió una medalla a la exhibición internacional en Londres en 1862.

En Inglaterra, en 1835, Charles Babbage construyó una máquina de realizar cálculos que mejoró con un ambicioso plan de la máquina analítica, que si bien no tuvo éxito completo, constituye el primer paso serio en la historia de las computadoras figura 4, creando una gran conmoción en el mundo científico; Ada Byron, a quien se lo conoce como el primer programador de la historia, trabajó con la máquina y organizó el esquema lógico de la misma.

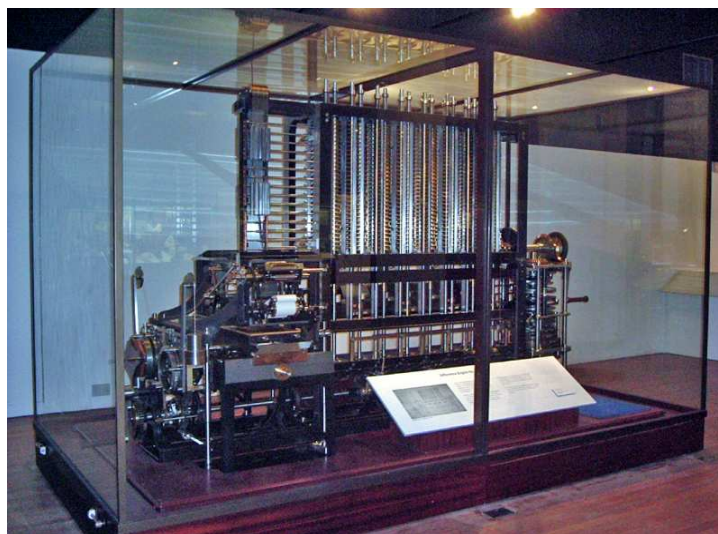


Figura 4. Máquina de Babbage (<http://es.wikipedia.org/wiki/Computadora>)

1.2 Definiciones de computadora

Máquina capaz de efectuar una secuencia de operaciones mediante un programa, de tal manera, que se realice un procesamiento sobre un conjunto de datos de entrada, obteniéndose otro conjunto de datos de salida.

Dispositivo electrónico capaz de recibir un conjunto de instrucciones y ejecutarlas realizando cálculos sobre los datos numéricos, o bien compilando y correlacionando otros tipos de información.

Es un calculador electrónico de elevada potencia equipado de memorias de gran capacidad y aparatos periféricos, que permite solucionar con gran rapidez y sin intervención humana el desarrollo de problemas lógicos y aritméticos muy complejos.

1.3 La Primera Computadora

Fue en 1830, cuando se establecieron los principios de funcionamiento de las modernas computadoras. Su paternidad se debe al matemático inglés Charles Babbage, quien tras lanzar en 1822 la denominada maquina diferencial figura 5. Con nada menos que 96 ruedas dentadas y 24 ejes, se lanzo en pos de su proyecto más relevante: la máquina analítica (1833).

La primera computadora fue la máquina analítica creada por Charles Babbage, profesor matemático de la Universidad de Cambridge en el siglo XIX. La idea que tuvo Charles Babbage sobre un computador nació debido a que la elaboración de las tablas matemáticas era un proceso tedioso y propenso a errores. En 1823 el gobierno Británico lo apoyo para crear el proyecto de una máquina de diferencias, un dispositivo mecánico para efectuar sumas repetidas.

Mientras tanto Charles Jacquard (francés), fabricante de tejidos, había creado un telar que podía reproducir automáticamente patrones de tejidos leyendo la información codificada en patrones de agujeros perforados en tarjetas de papel rígido. Al enterarse de este método Babbage abandonó la máquina de diferencias y se dedicó al proyecto de la máquina analítica que se pudiera programar con tarjetas perforadas para efectuar cualquier cálculo con una precisión de 20 dígitos figura 5.



Figura 5. Maquina diferencial y maquina de Charles Jacquard
(<http://es.wikipedia.org/wiki/Computadora>)

En 1944 se construyó en la Universidad de Harvard, la Mark I figura 6-(1-2), diseñada por un equipo encabezado por Howard H. Aiken. Esta máquina no está considerada como computadora electrónica debido a que no era de propósito general y su funcionamiento estaba basado en dispositivos electromecánicos llamados relevadores.

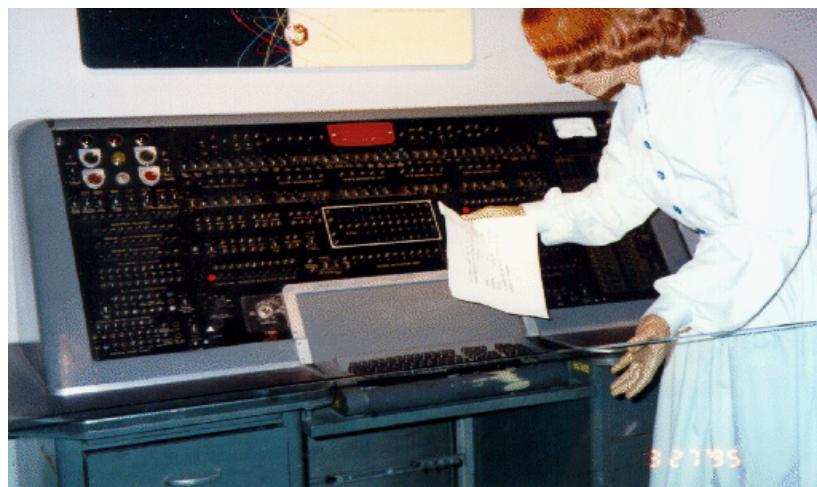


Figura 6-1. MARK I (<http://es.wikipedia.org/wiki/Computadora>)



Figura 6-2. MARK I (<http://es.wikipedia.org/wiki/Computadora>)

1.4 Generaciones de Computadoras

1.4.1 Primera Generación

En esta generación había un gran desconocimiento de las capacidades de las computadoras, puesto que se realizó un estudio en esta época que determinó que con veinte computadoras se saturaría el mercado de los Estados Unidos en el campo de procesamiento de datos.

Esta generación abarco la década de los cincuenta. Y se conoce como la primera generación. Estas máquinas tenían las siguientes características:

- Estas máquinas estaban construidas por medio de tubos de vacío.
- Eran programadas en lenguaje máquina.

Mauchly y J. Presper Eckert, Jr (1919-1995), diseñaron y construyeron, entre los años 1943 y 1946, el computador eléctrico de propósito general ENIAC. Existe una gran controversia respecto a que Mauchly copiara muchas de las ideas y conceptos del

profesor Atanasoff, para construir la computadora ENIAC figura 7. En cualquier caso en las últimas fases de su diseño y construcción aparece la importante figura de John Von Neumann (1903-1957), que actúa como consultor. En esta generación las máquinas son grandes y costosas.

En 1951 aparece la UNIVAC (Universal Computer) figura 8, fue la primera computadora comercial, que disponía de mil palabras de memoria central y podían leer cintas magnéticas, se utilizó para procesar el censo de 1950 en los Estados Unidos.

En las dos primeras generaciones, las unidades de entrada utilizaban tarjetas perforadas, retomadas por Herman Hollerith (1860 - 1929), quien además fundó una compañía que con el paso del tiempo se conocería como IBM (International Business Machines). Después se desarrolló por IBM la IBM 701 de la cual se entregaron 18 unidades entre 1953 y 1957.

Posteriormente, la compañía Remington Rand fabricó el modelo 1103, que competía con la 701 en el campo científico, por lo que la IBM desarrolló la 702, la cual presentó problemas en memoria, debido a esto no duró en el mercado.

La computadora más exitosa de la primera generación fue la IBM 650, de la cual se produjeron varios cientos. Esta computadora que usaba un esquema de memoria secundaria llamado tambor magnético, que es el antecesor de los discos actuales.

Otros modelos de computadora que se pueden situar en los inicios de la segunda generación son: la UNIVAC 80 y 90, las IBM 704 y 709, Burroughs 220 y UNIVAC 1105.

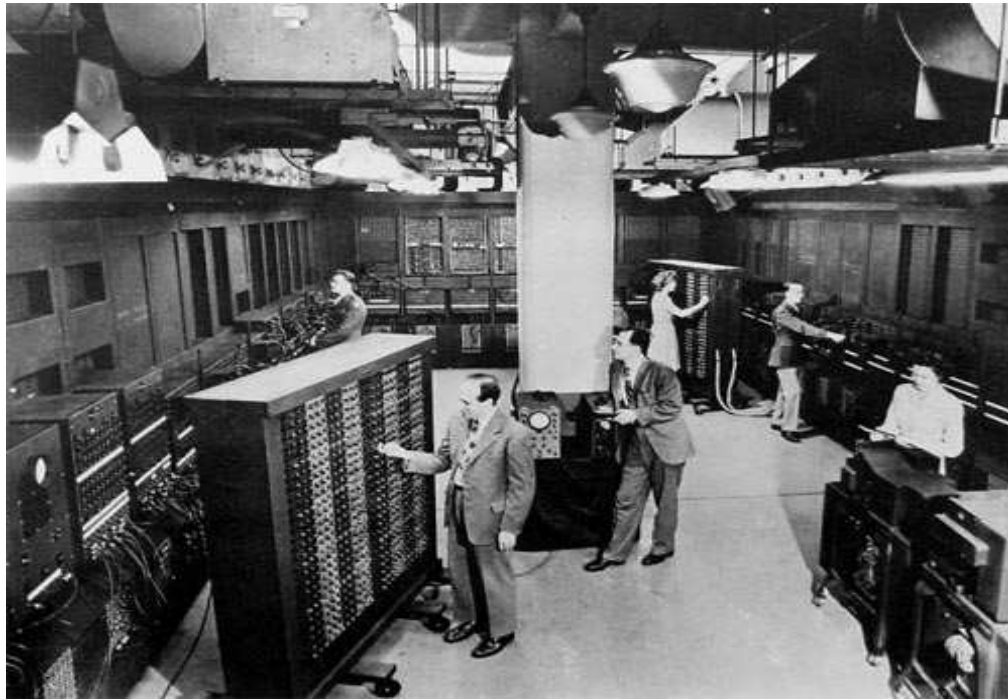


Figura 7. Computador ENIAC (<http://es.wikipedia.org/wiki/Computadora>)

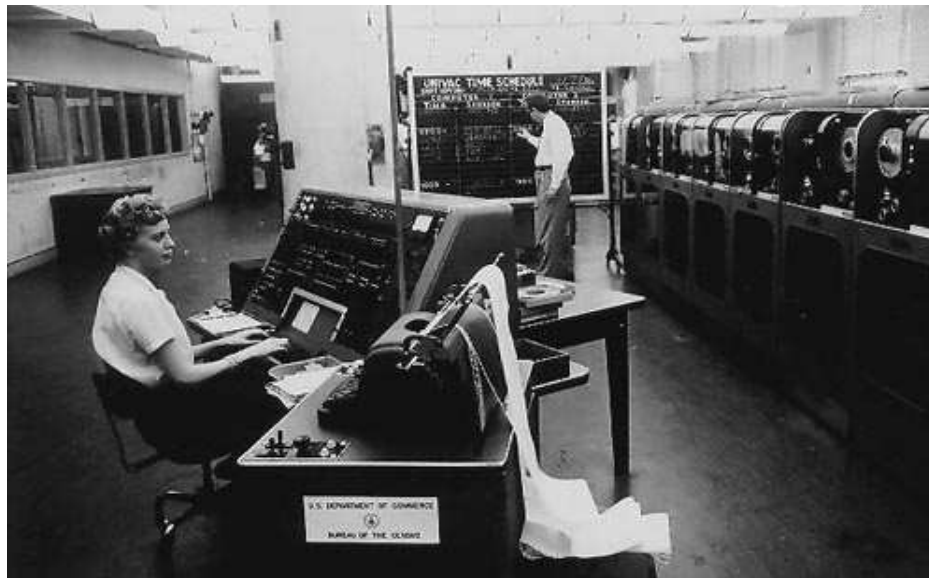


Figura 8. Primera computadora comercial UNIVAC - 1951
(<http://www.Historia de la computación4.htm>)

1.4.2 Segunda Generación

Cerca de la década de 1960, las computadoras seguían evolucionando, se reducía su tamaño y crecía su capacidad de procesamiento. También en esta época se empezó a definir la forma de comunicarse con las computadoras, que recibía el nombre de programación de sistemas.

Las características de la segunda generación son las siguientes:

- Están construidas con circuitos de transistores.
- Se programan en nuevos lenguajes llamados lenguajes de alto nivel.

En esta generación las computadoras se reducen de tamaño y son de menor costo. Aparecen muchas compañías y las computadoras eran bastante avanzadas para su época como la serie 5000 de Burroughs y la ATLAS de la Universidad de Manchester.

Algunas de estas computadoras se programaban con cintas perforadas y otras más por medio de cableado en un tablero. Los programas eran hechos a la medida por un equipo de expertos: analistas, diseñadores, programadores y operadores que se manejaban como una orquesta para resolver los problemas y cálculos solicitados por la administración. El usuario final de la información no tenía contacto directo con las computadoras. Esta situación en un principio se produjo en las primeras computadoras personales, pues se requería saberlas "programar" (introducir instrucciones) para obtener resultados; por lo tanto su uso estaba limitado a aquellos audaces pioneros que gustaran de pasar un buen número de horas escribiendo instrucciones, "corriendo" el programa resultante y verificando y corrigiendo los errores o bugs que aparecieran. Además, para no perder el "programa" resultante había que "guardarlo" (almacenarlo) en una grabadora de cassette, pues en esa época no había discos flexibles y mucho menos discos duros para las PCs; este procedimiento podía tomar de 10 a 45 minutos, según el programa. El panorama se modificó totalmente con la aparición de las computadoras personales con mejores

circuitos, más memoria, unidades de disco flexible y sobre todo con la aparición de programas de aplicación general en donde el usuario compra el programa y se pone a trabajar. Aparecen los programas procesadores de palabras como el célebre Word Star, la impresionante hoja de cálculo (spreadsheet) Visicalc y otros más que de la noche a la mañana cambian la imagen de la PC. El software empieza a tratar de alcanzar el paso del hardware. Pero aquí aparece un nuevo elemento: el usuario.

Las computadoras de esta generación fueron: la Philco 212 (esta compañía se retiró del mercado en 1964) y la UNIVAC M460, la Control Data Corporation modelo 1604, seguida por la serie 3000, la IBM mejoró la 709 y sacó al mercado la 7090, la National Cash Register empezó a producir máquinas para proceso de datos de tipo comercial, introdujo el modelo NCR 315.

1.4.3 Tercera generación

Con los progresos de la electrónica y los avances de comunicación con las computadoras en la década de los 1960, surge la tercera generación de las computadoras. Se inaugura con la IBM 360 en abril de 1964.

Las características de esta generación fueron las siguientes:

- Su fabricación electrónica esta basada en circuitos integrados.
- Su manejo es por medio de los lenguajes de control de los sistemas operativos.

La IBM produce la serie 360 con los modelos 20, 22, 30, 40, 50, 65, 67, 75, 85, 90, 195 que utilizaban técnicas especiales del procesador, unidades de cinta de nueve canales, paquetes de discos magnéticos y otras características que ahora son estándares (no todos los modelos usaban estas técnicas, sino que estaba dividido por aplicaciones).

El sistema operativo de la serie 360, se llamó OS que contaba con varias configuraciones, incluía un conjunto de técnicas de manejo de memoria y del procesador que pronto se convirtieron en estándares.

En 1964 CDC introdujo la serie 6000 con la computadora 6600 que se consideró durante algunos años como la más rápida.

En la década de 1970, la IBM produce la serie 370 (modelos 115, 125, 135, 145, 158, 168). UNIVAC compite con los modelos 1108 y 1110, máquinas en gran escala; mientras que CDC produce su serie 7000 con el modelo 7600. Estas computadoras se caracterizan por ser muy potentes y veloces.

A finales de esta década la IBM de su serie 370 produce los modelos 3031, 3033, 4341. Burroughs con su serie 6000 produce los modelos 6500 y 6700 de avanzado diseño, que se reemplazaron por su serie 7000.

A mediados de la década de 1970, aparecen en el mercado las computadoras de tamaño mediano, o mini-computadoras que no son tan costosas como las grandes (llamadas también como mainframes que significa también, gran sistema), pero disponen de gran capacidad de procesamiento. Algunas mini-computadoras fueron las siguientes: la PDP - 8 y la PDP - 11 de Digital Equipment Corporation, la VAX (Virtual Address Extended) de la misma compañía, los modelos NOVA y ECLIPSE de Data General, la serie 3000 y 9000 de Hewlett - Packard con varios modelos el 36 y el 34, la Wang y Honey - Well -Bull, Siemens de origen alemán, la ICL fabricada en Inglaterra. En la Unión Soviética se utilizó la US (Sistema Unificado, Ryad) que ha pasado por varias generaciones.

La **PDP-11** fabricada por la empresa Digital Equipment Corporation en las décadas de 1970 y 1980 figura 9. Fue la primera mini-computadora para interconectar todos los elementos del sistema-procesador, memoria y periférico a un único bus de comunicación, bidireccional, asíncrono. Este dispositivo, llamado **UNIBUS** permitía a los dispositivos enviar, recibir o intercambiar datos sin necesidad de dar un paso intermedio por la memoria.

La **PDP-11** fue instalada en la Universidad Mayor de San Simón en 1975 la cual tuvo un costo de 500.000 dólares, se utilizó como equipo central para procesos administrativos, estaba conectada a terminales tontas y trabajaba con 20 terminales pero su capacidad máxima era 256 terminales, la cual también se alquilaba a empresas como: ENDE (Empresa Nacional de Electricidad), funcionó hasta 1985 la cual fue reemplazada por la mini-computadora ALTOS en 1986 (fuente información UMSS carrera Informática).



Figura 9. Mini-computadora PDP-11 (<http://es.wikipedia.org/wiki/Computadora>)

1.4.4 Cuarta Generación

Aquí aparecen los microprocesadores que es un gran adelanto de la microelectrónica, son circuitos integrados de alta densidad y con una velocidad impresionante. Las microcomputadoras con base en estos circuitos son extremadamente pequeñas y baratas, por lo que su uso se extiende al mercado industrial. Aquí nacen las computadoras personales que han adquirido proporciones enormes y que han influido en la sociedad en general sobre la llamada "revolución informática".

En 1976 Steve Wozniak y Steve Jobs inventan la primera microcomputadora de uso masivo, y más tarde forman la compañía conocida como la Apple que fue la segunda compañía más grande del mundo, la cual saca en 1977 la famosa computadora Apple II figura 10 que se vendió extremadamente bien, la cual es antecedida tan solo por IBM; y esta por su parte es una de las cinco compañías más grandes del mundo.

En 1981 se vendieron 800000 computadoras personales, al año siguiente 1400000. Entre 1984 y 1987 se vendieron alrededor de 60 millones de computadoras personales, por lo que no queda duda que su impacto y penetración fueron enormes.

Con el surgimiento de las computadoras personales, el software y los sistemas que con ellas se manejan han tenido un considerable avance, por lo que han hecho más interactiva la comunicación con el usuario. Surgen otras aplicaciones como los procesadores de palabra, las hojas electrónicas de cálculo, paquetes gráficos, etc.

También las industrias del Software de las computadoras personales crece con gran rapidez, Gary Kildall y William Gates se dedicaron durante años a la creación de sistemas operativos y métodos para lograr una utilización sencilla de las microcomputadoras (son los creadores de CP/M y de los productos de Microsoft).



Figura 10. Computadora Apple II ([http://www Historia de la Computación.htm](http://www.Historia.de.la.Computación.htm))

No todas son micro-computadoras, por tanto, las mini-computadoras y los grandes sistemas continúan en desarrollo. De hecho las máquinas pequeñas rebasaban por mucho la capacidad de los grandes sistemas de 10 o 15 años antes, que requerían de instalaciones costosas y especiales, pero sería equivocado suponer que las grandes computadoras han desaparecido; por el contrario, su presencia era ya ineludible en prácticamente todas las esferas de control gubernamental, militar y de la gran industria.

Las enormes computadoras de las series CDC, CRAY, Hitachi o IBM por ejemplo, eran capaces de atender a varios cientos de millones de operaciones por segundo.

1.4.5 Quinta Generación

En vista de la acelerada marcha de la microelectrónica, la sociedad industrial se ha dado a la tarea de poner también a esa altura el desarrollo del software y los sistemas con que se manejan las computadoras. Surge la competencia internacional por el dominio del mercado de la computación, en la que se perfilan empresas líderes que, sin embargo, no han podido alcanzar el nivel que se desea: la capacidad de comunicarse con la computadora en un lenguaje más cotidiano y no a través de códigos o lenguajes de control especializados.

Japón lanzó en 1983 el llamado "programa de la quinta generación de computadoras", con los objetivos explícitos de producir máquinas con innovaciones reales en los criterios mencionados. Y en los Estados Unidos ya está en actividad un programa en desarrollo que persigue objetivos semejantes, que pueden resumirse de la siguiente manera:

- Procesamiento en paralelo mediante arquitecturas y diseños especiales y circuitos de gran velocidad.
- Manejo de lenguaje natural y sistemas de inteligencia artificial.

El futuro previsible de la computación es muy interesante, y se puede esperar que esta ciencia siga siendo objeto de atención prioritaria de gobiernos y de la sociedad en conjunto.

1.5 Tipos de computadoras

1.5.1 Análoga

La computadora análoga es la que acepta y procesa señales continuas, tales como: fluctuaciones de voltaje o frecuencias.

Ejemplo: El termostato es la computadora análoga más sencilla.

1.5.2 Digital

La computadora digital es la que acepta y procesa datos que han sido convertidos al sistema binario. La mayoría de las computadoras son digitales.

1.5.3 Híbrida

La computadora híbrida es una computadora digital que procesa señales análogas que han sido convertidas a forma digital. Es utilizada para control de procesos y en robótica.

1.5.4 Propósito especial

La computadora de propósito especial está dedicada a un solo propósito o tarea. Pueden ser usadas para producir informes del tiempo, monitorear desastres naturales, hacer lecturas de gasolina y como medidor eléctrico. Ejemplo: carros de control remoto, horno microondas, relojes digitales, cámaras, procesador de palabras, etc.

1.5.5 Propósito general

La computadora de propósito general se programa para una variedad de tareas o aplicaciones. Son utilizadas para realizar cálculos matemáticos, estadísticos, contabilidad comercial, control de inventario, nómina, preparación de inventario, etc. Ejemplo: "mainframes" o mini computadoras.

1.6 Categorías de computadoras

1.6.1 Supercomputadora

La supercomputadora es lo más avanzado en computadora, es la más rápida y, por lo tanto, la más cara. Se fabrican en un número determinado por varias empresas. Procesan billones de instrucciones por segundo. Son utilizadas para trabajos científicos, particularmente para crear modelos matemáticos del mundo real, llamados simulación.

Mainframe

Los "mainframe" son computadoras grandes, ligeras, capaces de utilizar cientos de dispositivos de entrada y salida. Procesan millones de instrucciones por segundo. Su velocidad operacional y capacidad de procesar hacen que los grandes negocios, el gobierno, los bancos, las universidades, los hospitales, compañías de seguros, líneas aéreas, etc. confíen en ellas. Su principal función es procesar grandes cantidades de datos rápidamente. Estos datos están accesibles a los usuarios del "mainframe" o a los usuarios de las microcomputadoras cuyos terminales están conectados al "mainframe". Su costo fluctúa entre varios cientos de miles de dólares hasta el millón. Requieren de un sistema especial para controlar la temperatura y la humedad. También requieren de un personal profesional especializado para procesar los datos y darle el mantenimiento.

1.6.2 Minicomputadora

La minicomputadora se desarrolló en la década de 1960 para llevar a cabo tareas especializadas, tales como el manejo de datos de comunicación. Son más pequeñas, más baratas y más fáciles de mantener e instalar que los "mainframes". Su costo está entre los miles de dólares. Usadas por negocios, colegios y agencias gubernamentales. Su mercado ha ido disminuyendo desde que surgieron las microcomputadoras.

1.6.3 Microcomputadora

La microcomputadora es conocida como computadora personal o PC. Es la más pequeña, gracias a los microprocesadores, más barata y más popular en el mercado. Su costo fluctúa entre varios cientos de dólares hasta varios miles de dólares. Puede funcionar como unidad independiente o estar en red con otras microcomputadoras, la cual puede ejecutar las mismas operaciones y usar los mismos programas que muchas computadoras superiores, aunque en menor capacidad.

Ejemplos: MITS Altair, Macintosh, serie Apple II, IBM PC, Dell, Compaq, Gateway, etc.

2. HARDWARE (arquitectura del computador)

Podemos denominar al hardware como todo el conjunto físico de la computadora, lo cual incluye el CPU (el cual contiene todas las tarjetas de procesamiento, ya sean de sonidos, gráficos, módem, unidades de discos, procesador, memoria RAM, etc.), el monitor, bocinas, escáner, impresora, mouse, teclado, micrófono, entre otros. El Hardware es la unión de componentes físicos capaces de realizar la comunicación entre el usuario y el software.

Todo sistema de cómputo tiene componentes de hardware dedicados a estas funciones:

1. Unidad de entrada
2. Unidad de salida
3. Unidad de procesamiento (UP).
4. Memoria y dispositivos de almacenamiento.

3. FUNCIONAMIENTO DE UNA COMPUTADORA

Una computadora consiste en una serie de dispositivos que juntos funcionan como una unidad integrada o sistema.

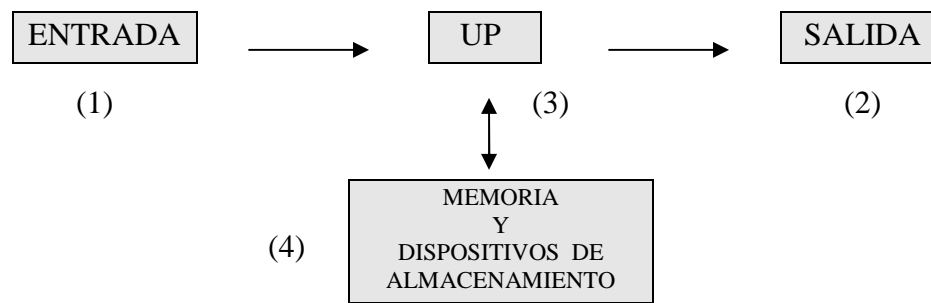


Figura 11. Diagrama de bloques de una computadora (Propia)

Las unidades de entrada / salida y de memoria y dispositivos de almacenamiento se conocen como periféricos o dispositivos.

La figura 11. Ilustra el diagrama de bloques de una computadora.

3.1. ENTRADA

Para ingresar los datos a la computadora, se utilizan diferentes dispositivos:

Teclado

Dispositivo de entrada más comúnmente utilizado que encontramos en todos los equipos computacionales. El teclado se encuentra compuesto de 3 partes: teclas de función, teclas alfanuméricas y teclas numéricas.

Mouse

Es el segundo dispositivo de entrada más utilizado. El mouse o ratón es arrastrado a lo largo de una superficie para maniobrar un apuntador en la pantalla del monitor. Fue inventado por Douglas Engelbart y su nombre se deriva por su forma la cual se asemeja a la de un ratón.

Lápiz Óptico

Este dispositivo es muy parecido a una pluma ordinaria, pero conectada a un cordón eléctrico y que requiere de un software especial. Haciendo que la pluma toque el monitor el usuario puede elegir los comandos de las programas.

Tabla Digitalizadora

Es una superficie de dibujo con un medio de señalización que funciona como un lápiz. La tabla convierte los movimientos de este apuntador en datos digitalizados que pueden ser leídos por ciertos paquetes de cómputo. Los tamaños varían desde tamaño carta hasta la cubierta de un escritorio.

Entrada de voz (reconocimiento de voz)

Convierten la emisión vocal de una persona en señales digitales. La mayoría de estos programas tienen que ser "entrenados" para reconocer los comandos que el usuario da verbalmente. El reconocimiento de voz se usa en la profesión médica para permitir a los doctores compilar rápidamente reportes. Más de 300 sistemas Kurzweil Voicemed están instalados actualmente en más de 200 Hospitales en Estados Unidos. Este novedoso sistema de reconocimiento fónico utiliza tecnología de independencia del hablante. Esto significa que una computadora no tiene que ser entrenada para reconocer el lenguaje o tono de voz de una sola persona. Puede reconocer la misma palabra dicha por varios individuos.

Pantallas sensibles al tacto (Touch Screen)

Permiten dar comandos a la computadora tocando ciertas partes de la pantalla. Muy pocos programas de software trabajan con ellas y los usuarios se quejan de que las pantallas están muy lejos del teclado. Algunas tiendas emplean este tipo de tecnología para ayudar a los clientes a encontrar los bienes o servicios dentro de la tienda.

Scanner

Convierten texto, fotografías a color ó en Blanco y Negro a una forma que puede leer una computadora. Después esta imagen puede ser modificada, impresa y almacenada. Son capaces de digitalizar una página de gráficas en unos segundos y proporcionan una forma rápida, fácil y eficiente de ingresar información impresa en una computadora; también se puede ingresar información si se cuenta con un Software especial llamado OCR (Reconocimiento Óptico de Caracteres).

*** Tarea: Existen muchos dispositivos de entrada adicionales, averiguar cuales son?**

3.2. Unidad de Procesamiento (UP)

El UCP (Unidad Central de Procesamiento) es el responsable de controlar el flujo de datos (actividades de entrada y salida E/S) y de la ejecución de las instrucciones de los programas sobre los datos. Realiza todos los cálculos (suma, resta, multiplicación, división y compara números y caracteres). Es el "cerebro" de la computadora. Se divide en 3 Componentes:

1. Unidad de Control (UC)
2. Unidad Aritmético/Lógica (UAL)
3. Área de almacenamiento primario (memoria)

Unidad de Control (UC)

Es en esencia la que gobierna todas las actividades de la computadora, así como el UP es el cerebro de la computadora, se puede decir que la UC es el cerebelo del UP. Supervisa la ejecución de los programas, coordina y controla al sistema de cómputo, es decir, coordina actividades de E/S, determina que instrucción se debe ejecutar y pone a disposición los datos pedidos por la instrucción. Determina donde se almacenan los datos y los transfiere desde las posiciones donde están almacenados. Una vez ejecutada

la instrucción la Unidad de Control debe determinar donde pondrá el resultado para salida ó para su uso posterior.

Unidad Aritmético / Lógica (UAL)

Esta unidad realiza cálculos (suma, resta, multiplicación y división) y operaciones lógicas (comparaciones). Transfiere los datos entre las posiciones de almacenamiento. Tiene un registro muy importante conocido como: Acumulador **ACC** (accumulator) al realizar operaciones aritméticas y lógicas, la **UAL** mueve datos entre ella y el almacenamiento. Los datos usados en el procesamiento se transfieren de su posición en el almacenamiento a la **UAL**. Los datos se manipulan de acuerdo con las instrucciones del programa y regresan al almacenamiento. Debido a que el procesamiento no puede efectuarse en el área de almacenamiento, los datos deben transferirse a la **UAL**. Para terminar una operación puede suceder que los datos pasen de la **UAL** al área de almacenamiento varias veces.

3.3. Memoria

La memoria consiste en la capacidad de registrar sea una cadena de caracteres o de instrucciones (programa) y volver a incorporarlo en un determinado proceso como ejecutarlo bajo ciertas circunstancias. Dentro la cual se encuentran:

Área de Almacenamiento Primario

La memoria da al procesador almacenamiento temporal para programas y datos. Todos los programas y datos deben transferirse a la memoria desde un dispositivo de entrada o desde el almacenamiento secundario (disquete), antes de que los programas puedan ejecutarse o procesarse los datos. Las computadoras usan 2 tipos de memoria primaria:

-**ROM (Read Only Memory)**, memoria de sólo lectura, en la cual se almacena ciertos programas e información que necesita la computadora las cuales están grabadas permanentemente y no pueden ser modificadas por el programador. Las instrucciones básicas para arrancar una computadora están grabadas aquí.

-**RAM** (Random Access Memory), memoria de acceso aleatorio, es utilizada por el usuario mediante sus programas, y es volátil. La memoria del equipo permite almacenar datos de entrada, instrucciones de los programas que se están ejecutando en ese momento, los datos-resultados del procesamiento y los datos que se preparan para la salida.

Los datos proporcionados a la computadora permanecen en el almacenamiento primario hasta que se utilizan en el procesamiento. Durante el procesamiento, el almacenamiento primario almacena los datos intermedios y finales de todas las operaciones aritméticas y lógicas. El almacenamiento primario debe guardar también las instrucciones de los programas usados en el procesamiento. La memoria está subdividida en celdas individuales cada una de las cuales tiene una capacidad similar para almacenar datos.

Almacenamiento Secundario

El almacenamiento secundario es un medio de almacenamiento persistente (no volátil como el de la memoria **RAM**). El proceso de transferencia de datos a un equipo de cómputo se le llama procedimiento de lectura. El proceso de transferencia de datos desde la computadora hacia el almacenamiento se denomina procedimiento de escritura. En la actualidad se pueden usar principalmente dos tecnologías para almacenar información:

- 1.- El almacenamiento Magnético.
- 2.- El almacenamiento Óptico.
- 3.- Algunos dispositivos combinan ambas tecnologías.

Dispositivos de Almacenamiento Magnético

Entre los dispositivos de almacenamiento magnéticos se puede enumerar los siguientes:

- 1.- Discos Flexibles (3 ½)

- 2.- Discos Duros
- 3.- Unidades de CD (Compact Disk o disco compacto)
- 4.- Unidades de Almacenamiento ZIP
- 5.- Unidades de DVD (Digital Versatile Disc o Disco Versátil Digital)
- 6.- Pen-Drive

*** Tarea: Existen muchos dispositivos mas, averiguar cuales son?**

Almacenamiento Óptico

La necesidad de mayores capacidades de almacenamiento han llevado a los fabricantes de hardware a una búsqueda continua de medios de almacenamiento alternativos y cuando no hay opciones, a mejorar tecnologías disponibles y desarrollar nuevas. Las técnicas de almacenamiento óptico hacen posible el uso de la localización precisa mediante rayos láser.

Leer información de un medio óptico es una tarea relativamente fácil, escribirla es otro asunto. El problema es la dificultad para modificar la superficie de un medio óptico, ya que los medios ópticos perforan físicamente la superficie para reflejar o dispersar la luz del láser.

Los principales dispositivos de almacenamiento óptico son:

- 1. - CD ROM. - CD (**R**ead **O**nly **M**emory)
- 2. - WORM. (**W**rite **O**nce, **R**ead **M**any)

*** Tarea: Existen muchos dispositivos mas, averiguar cuales son?**

Medios Magnético - Ópticos

Estos medios combinan algunas de las mejores características de las tecnologías de grabación magnética y óptica. Un disco MO (magnético-óptico) tiene la capacidad de un disco óptico, pero puede ser re-gravable con la facilidad de un disco magnético. Actualmente están disponibles en varios tamaños y capacidades.

Los dispositivos de salida de una computadora es el hardware que se encarga de mandar una respuesta hacia el exterior de la computadora, como pueden ser: los monitores, impresoras, sistemas de sonido, módem. etc.

Hard Disk (HD)

El disco duro figura¹² es el sistema de almacenamiento más importante del computador y en el se guardan los archivos de los programas – como los sistemas operativo D.O.S. o Windows XP, las hojas de cálculo (Excel, Qpro, Lotus) los procesadores de texto (Word, WordPerefct, Word Star, Word Pro), los juegos y los archivos de cartas y otros documentos que se produce.

La mayoría de los discos duros en los computadores personales son de tecnología **IDE** (Integrated Drive Electronics), que viene en las tarjetas controladoras y en todas las tarjetas madres (motherboard) de los equipos nuevos. Estas últimas reconocen automáticamente (autodetect) los discos duros que se le coloquen, hasta un tamaño de 750 giga bytes.

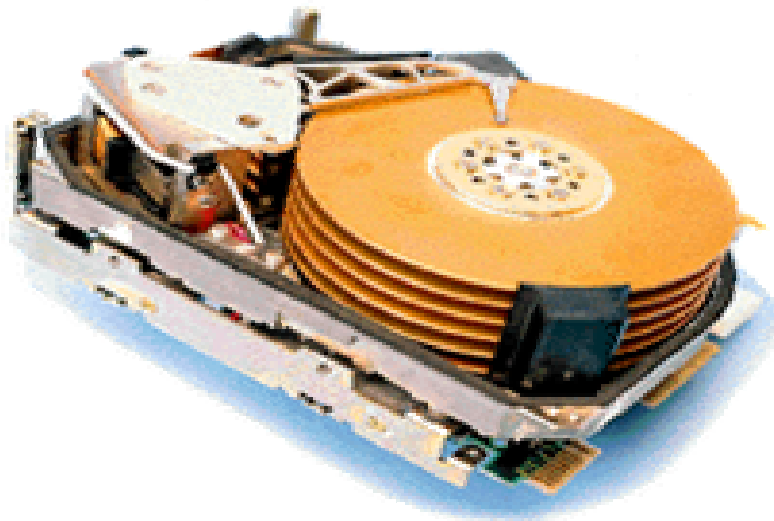


Figura 12. Disco duro ([http://www El Disco Duro \(HD\) - Monografias_com.htm](http://www.ElDiscoDuro(HD)-Monografias_com.htm))

Cuando el usuario o el software indica al sistema operativo a que deba leer o escribir a un archivo, el sistema operativo solicita que el controlador del disco duro traslade los cabezales de lectura/escritura a la tabla de asignación de archivos FAT (**File** Allocation Table o Tabla de Asignación de Archivos). El sistema operativo lee la FAT para determinar en qué punto comienza un archivo en el disco, o qué partes del disco están disponibles para guardar un nuevo archivo. Los cabezales escriben datos en los platos al alinear partículas magnéticas sobre las superficies de éstos figura 13. Los cabezales leen datos al detectar las polaridades de las partículas que ya se han alineado. Es posible guardar un solo archivo en racimos diferentes sobre varios platos, comenzando con el primer racimo disponible que se encuentra. Después de que el sistema operativo escribe un nuevo archivo en el disco, se graba una lista de todos los racimos del archivo en la FAT.

Una computadora funciona al ritmo marcado por su componente más lento, y por eso un disco duro lento puede hacer que la maquina sea vencida en prestaciones por otro equipo menos equipado en cuanto a procesador y cantidad de memoria, pues de la velocidad del disco duro depende el tiempo necesario para cargar los programas y para recuperar y almacenar los datos.



Figura 13. Cabezales del Disco duro (<http://www.Discos-duros-y-particiones.htm>)

Velocidad de Rotación (RPM)

Es la velocidad a la que gira el disco duro, más exactamente, la velocidad a la que giran el/los platos del disco, que es donde se almacenan magnéticamente los datos. La regla es: a mayor velocidad de rotación, más alta será la transferencia de datos, pero también mayor será el ruido y mayor será el calor generado por el disco duro. Se mide en número revoluciones por minuto (RPM). Una velocidad de 5400RPM permitirá una transferencia entre 10MB y 16MB por segundo con los datos que están en la parte exterior del cilindro o plato, algo menos en el interior, también existen discos con velocidades (7200 RPM = 120 Km/hora en el borde) y 10000 RPM.

3.4. SALIDA

Para la salida de datos de la computadora se utilizan los siguientes dispositivos:

Monitores

El monitor ó pantalla de vídeo, es el dispositivo de salida más común. Hay algunos que forman parte del cuerpo de la computadora y otros están separados de la misma. Existen muchas formas de clasificar los monitores, la básica es en término de sus capacidades de color, pueden ser: Monocromáticos, despliegan sólo 2 colores, uno para el fondo y otro para la superficie. Los colores pueden ser blanco y negro, verde y negro ó ámbar y negro. Escala de Grises, un monitor a escala de grises es un tipo especial de monitor monocromático capaz de desplegar diferentes tonos de grises. Color: Los monitores de color pueden desplegar de 4 hasta 1 millón de colores diferentes.

Conforme ha avanzado la tecnología han surgido los diferentes modelos: TTL, Monocromático, muy pobre resolución, los primeros no tenían capacidad de graficar. CGA, Color Graphics Adapter, desplegaba 4 colores, con muy pobre resolución a comparación de los monitores actuales, hoy en día fuera del mercado. EGA, Enhanced Graphics Adapter, manejaba una mejor resolución que el CGA, de 640x350 píxeles (los píxeles son los puntos de luz con los que se forman los caracteres y gráficas en el monitor, mientras más píxeles mejor resolución). Desplegaban 64 colores. VGA, Vídeo

Graphics **ARRAY**, los hay monocromáticos y de color. Adecuados para ambiente gráfico por su alta resolución (640x480 píxeles), pueden llegar hasta 256000 colores ó 64 tonalidades de gris dependiendo de la memoria destinada al dispositivo. PVGA, Súper Vídeo Graphics **ARRAY**, maneja una resolución más alta (1024x768), el número de colores presentados varía dependiendo de la memoria, pero puede ser mayor que 1 millón de colores.

La calidad de las imágenes que un monitor puede desplegar se define más por las capacidades de la tarjeta controladora de vídeo, que por las del monitor mismo. El controlador de vídeo es un dispositivo intermediario entre el UCP y el monitor. El controlador contiene la memoria y otros circuitos electrónicos necesarios para enviar la información al monitor para que la despliegue en la pantalla.

Impresoras

Dispositivo que convierte la salida de la computadora en imágenes impresas. Las impresoras se pueden dividir en 2 tipos: las de matriz de puntos y las de tinta y láser.

- **Impresoras de Matriz de puntos**

Una impresora que utiliza un mecanismo de impresión que hace impactar la imagen del carácter en una cinta y sobre el papel. Las impresoras de línea, de matriz de punto y de rueda de margarita son ejemplos de impresoras de impacto. La impresora de matriz de puntos, fue la impresora más común. Tiene una cabeza de impresión movible con varias puntillas o agujas que al golpear la cinta entintada forman caracteres por medio de puntos en el papel, Mientras mas agujas tenga la cabeza de impresión mejor será la calidad del resultado. Las hay de 10 y 15", las velocidades varían desde: 280 cps hasta 1,066 cps. Impresoras de margarita; tienen la misma calidad de una máquina de escribir mediante un disco de impresión que contiene todos los caracteres, están de salida del mercado por lentas. Impresoras de Línea: Son impresoras de alta velocidad que imprimen una línea por vez. Generalmente se conectan a grandes computadoras y a mini

computadoras. Las impresoras de línea imprimen una línea a la vez desde aproximadamente 100 a 5000 lpm.

- **Impresoras de tinta y láser**

Hacen la impresión por diferentes métodos, pero no utilizan el impacto. Son menos ruidosas y con una calidad de impresión notoriamente mejor a las impresoras de impacto. Los métodos que utilizan son los siguientes: Térmicas: Imprimen de forma similar a la máquina de matriz, pero los caracteres son formados marcando puntos por quemadura de un papel especial. Velocidad 80 cps los faxes trabajan con este método.

Impresora de inyección de tinta: Emite pequeños chorros de tinta desde cartuchos desechables hacia el papel, las hay de color. Velocidad de 4 a 12 ppm. Electrofotográficas o Láser: Crean letras y gráficas mediante un proceso de fotocopiado. Un rayo láser traza los caracteres en un tambor fotosensible, después fija el toner al papel utilizando calor. Muy alta calidad de resolución, velocidades de 4 a 20 ppm.

3.5 Otros periféricos

Se denominan periféricos tanto a las unidades o dispositivos a través de los cuales la computadora se comunica con el mundo exterior, como a los sistemas que almacenan o archivan la información, sirviendo de memoria auxiliar de la memoria principal. Algunos de estos se mencionan a continuación:

Modem

El modem es uno de los periféricos que con el tiempo se ha convertido ya en imprescindible y pocos son los modelos de computadora que no estén conectados en red que no lo incorporen. Su gran utilización viene dada básicamente por dos motivos: Internet y el fax, aunque también puede tener otros usos como son su utilización como contestador automático incluso con funciones de para conectarnos con la red local de nuestra oficina o con la central de nuestra empresa. Cuando se está conectado a una red, en este caso será la propia red la que utilizará el modem para poder conectarse a otras

redes o a Internet estando en este caso conectado a nuestro servidor o a un router figura 14. Los modem se utilizan con líneas analógicas, ya que su propio nombre indica su principal función, que es la de modular-demodular la señal digital proveniente de nuestro ordenador y convertirla a una forma de onda que sea asimilable por dicho tipo de líneas. Uno de los parámetros que lo definen es su velocidad. La velocidad máxima está en los 56 Kbps (Kilobytes por segundo). Esta norma se caracteriza por un funcionamiento asimétrico, puesto que la mayor velocidad sólo es alcanzable "en bajada", ya que en el envío de datos está limitada a 33,6 Kbps.



Figura14. Modem (<http://es.wikipedia.org/wiki/Computadora>)

4. SOFTWARE

El Software es un conjunto de programas, documentos, procedimientos, y rutinas asociadas con la operación de un sistema de cómputo. Distinguiéndose de los componentes físicos llamados hardware. Comúnmente a los programas de computación se les llama software; el software asegura que el programa o sistema cumpla por completo con sus objetivos, opera con eficiencia, esta adecuadamente documentado, y suficientemente sencillo de operar. Es simplemente el conjunto de instrucciones individuales que se le proporciona al microprocesador para que pueda procesar los datos y generar los resultados esperados. El hardware por si solo no puede hacer nada, pues es necesario que exista el software, que es el conjunto de instrucciones que hacen funcionar al hardware.

4.1 Clasificación del Software

El software se clasifica en 4 diferentes Categorías: Software de base, lenguajes de programación, software de uso general, software de aplicación. Algunos autores consideran la 3era y 4ta clasificación como una sola.

4.1.1 Software de base

Dentro esta categoría se distingue el sistema operativo (SO), que es el gestor y organizador de todas las actividades que realiza la computadora. Marca las pautas según las cuales se intercambia información entre la memoria central y la externa, y determina las operaciones elementales que puede realizar el procesador. El sistema operativo, debe ser cargado en la memoria central antes que ninguna otra información.

Un sistema Operativo (SO) es en sí mismo un programa de computadora. Sin embargo, es un programa muy especial, quizá el más complejo e importante en una computadora. El SO despierta a la computadora y hace que reconozca a la UP, la memoria, el teclado, el sistema de vídeo y las unidades de disco. Además, proporciona la facilidad para que los usuarios se comuniquen con la computadora y sirve de plataforma a partir de la cual se corran programas de aplicación.

4.1.2 Lenguajes de Programación

Mediante los programas se indica a la computadora que tarea debe realizar y cómo efectuarla, pero para ello es preciso introducir estas órdenes en un lenguaje que el sistema pueda entender. En principio, la computadora sólo entiende las instrucciones en código máquina, es decir, el específico de la computadora. Sin embargo, a partir de éstos se elaboran los llamados lenguajes de alto y bajo nivel.

Los computadores interpretan (comprenden) un lenguaje muy simple llamado lenguaje de máquina. Cada instrucción del lenguaje de máquina es elemental. Un programa escrito en lenguaje de máquina necesita muchas instrucciones para hacer cosas simples (es decir, es difícil de escribir) y sólo funciona en un computador del mismo tipo. El

lenguaje ensamblador es el primer intento de sustituir el lenguaje máquina por otro más similar a los utilizados por las personas.

Los lenguajes de programación de alto nivel constituyen un paso evolutivo y pretenden brindar cierto nivel de abstracción e independencia del computador.

4.1.3 Software de uso General

El software para uso general ofrece la estructura para un gran número de aplicaciones empresariales, científicas y personales. El software de hoja de cálculo, de diseño asistido por computadoras (CAD), de procesamiento de texto, de manejo de bases de datos, pertenece a esta categoría. La mayoría de software para uso general se vende como paquete; es decir, con software y documentación orientada a los usuarios (manuales de referencia, plantillas de teclado y demás).

4.1.4 Software de Aplicación

El software de aplicación está diseñado y escrito para realizar tareas específicas personales, empresariales o científicas como el procesamiento de nóminas, la administración de los recursos humanos o el control de inventarios, etc. Todas estas aplicaciones procesan datos (recepción de materiales) y generan información (registros de nómina) para el usuario.

4.2 Unidad de Información

4.2.1 Bit (dígito binario)

Unidad mínima de almacenamiento de la información cuyo valor puede ser 0 ó 1; o bien verdadero o falso.

4.2.2 Byte

Conjunto de 8 bits el cual suele representar un valor asignado a un carácter.

4.2.3 Concepto de Registro

Los registros del procesador se emplean para controlar instrucciones en ejecución, manejar direccionamiento de memoria y proporcionar capacidad aritmética. Los registros son espacios físicos dentro del microprocesador con capacidad de 4 bits hasta 64 bits dependiendo del microprocesador que se emplee. Los registros son direccionables por medio de una viñeta, que es una dirección de memoria. Los bits, por conveniencia, se numeran de derecha a izquierda (15, 14, 13.... 3, 2, 1, 0), los registros están divididos en seis grupos los cuales tienen un fin específico.

4.2.4. Concepto de Archivos

Los archivos, bien sean generales o históricos, tienen la enorme tarea de almacenar y organizar cantidades ingentes de información de forma tal que puedan ser recuperadas por sus usuarios potenciales. La preservación y prestación de servicios conduce a la necesidad de la restauración y reprografía. En el pasado su carácter de custodios los obligó a trabajar en tecnologías como el microfilm, que hoy resultan demasiado morosas y limitadas. Actualmente, la digitalización y organización de la información computarizada se convierte en un imperativo. Afortunadamente la teleinformática moderna ha abierto una gran cantidad de posibilidades para el manejo electrónico de archivos, para almacenar y preservar la información, para agilizar y hacer eficiente la búsqueda y recuperación de información, el manejo de índices e imágenes y similares de documentos. Las limitaciones físicas de los viejos archivos dan paso al concepto de archivos virtuales que pueden prestar servicios seguros, confiables y eficientes a través de Internet, sin que importen las consideraciones geográficas. Los investigadores y usuarios en general de los archivos ya no tienen que trasladarse físicamente. Ahora la información es la que viaja a sus usuarios.

4.3. BIOS

En computación, el sistema básico de entrada/salida **Basic Input-Output System (BIOS)** es un código de interfaz que localiza y carga el sistema operativo en la RAM; es un software muy básico instalado en la tarjeta madre que permite que ésta cumpla su

cometido. Proporciona la comunicación de bajo nivel, y el funcionamiento y configuración del hardware del sistema que, como mínimo, maneja el teclado y proporciona salida básica (emitiendo pitidos normalizados por el altavoz de la computadora si se producen fallos) durante el arranque. El BIOS usualmente está escrito en lenguaje ensamblador.

El primer término BIOS apareció en el sistema operativo CP/M (Computer **PROGRAM** Monitor), y describe la parte de CP/M que se ejecutaba durante el arranque y que iba unida directamente al hardware (las máquinas de CP/M usualmente tenían un simple cargador arrancable en la ROM, y nada más). La mayoría de las versiones de MS-DOS tienen un archivo llamado "IBMBIO.COM" o "IO.SYS" que es análogo al CP/M BIOS. En los primeros sistemas operativos para PC (como el DOS), el BIOS todavía permanecía activo tras el arranque y funcionamiento del sistema operativo. El acceso a dispositivos como la disquetera y el disco duro se hacían a través del BIOS. Sin embargo, los sistemas operativos más modernos realizan estas tareas por sí mismos, sin necesidad de llamadas a las rutinas del BIOS. Al encender la computadora, el BIOS se carga automáticamente en la memoria principal y se ejecuta desde ahí por el procesador (aunque en algunos casos el procesador ejecute el BIOS leyéndolo directamente desde la ROM que lo contiene mayor referencia ver anexos A1 – A2), cuando realiza una rutina de verificación e inicialización de los componentes presentes en la computadora, a través de un proceso denominado **POST (Power On Self Test)**. Al finalizar esta fase busca el código de inicio del sistema operativo (bootstrap) en algunos de los dispositivos de memoria_secundaria presentes, lo carga en memoria y transfiere el control de la computadora a éste.

Se puede resumir diciendo que el BIOS es el firmware presente en computadoras IBM PC y compatibles, que contiene las instrucciones más elementales para el funcionamiento de las mismas por incluir rutinas básicas de control de los dispositivos de entrada y salida. Está almacenado en un chip de memoria ROM o Flash, situado en la placa base de la computadora. Este chip suele denominarse en femenino "la BIOS", pues se refiere a una memoria (femenino) concreta; aunque para referirnos al contenido, lo

correcto es hacerlo en masculino "el BIOS", ya que nos estamos refiriendo a un sistema (masculino) de entrada/salida figura 15. Ver anexo A3 mantenimiento del ordenador.



Figura 15. Memoria ROM conteniendo la BIOS ([http://www BIOS - Wikipedia, la enciclopedia libre.htm](http://www.BIOS-Wikipedia,la-enciclopedia-libre.htm))

4.4. Categoría de los Sistemas Operativos (SO)

4.4.1. Sistema Operativo Multitareas

Es el modo de funcionamiento disponible en algunos sistemas operativos, mediante el cual una computadora procesa varias tareas al mismo tiempo. Existen varios tipos de multitareas. La conmutación de contextos (context switching) es un tipo muy simple de multitarea en el que dos o más aplicaciones se cargan al mismo tiempo, pero en el que solo se esta procesando la aplicación que se encuentra en primer plano (la que ve el usuario). Para activar otra tarea que se encuentre en segundo plano, el usuario debe traer al primer plano la ventana o pantalla que contenga esa aplicación. En la multitarea cooperativa, la que se utiliza en el sistema operativo Macintosh, las tareas en segundo plano reciben tiempo de procesado durante los tiempos muertos de la tarea que se encuentra en primer plano (por ejemplo, cuando esta aplicación esta esperando información del usuario), y siempre que esta aplicación lo permita.

En los sistemas multitarea de tiempo compartido, como OS/2, cada tarea recibe la atención del microprocesador durante una fracción de segundo. Para mantener el sistema en orden, cada tarea recibe un nivel de prioridad o se procesa en orden secuencial. Dado que el sentido temporal del usuario es mucho más lento que la velocidad de

procesamiento del ordenador, las operaciones de multitarea en tiempo compartido parecen ser simultáneas.

4.4.2. Sistema Operativo Monotareas

Los sistemas operativos monotareas son más primitivos y es todo lo contrario al visto anteriormente, es decir, solo pueden manejar un proceso en cada momento o que solo puede ejecutar las tareas de una en una. Por ejemplo cuando la computadora esta imprimiendo un documento, no puede iniciar otro proceso ni responder a nuevas instrucciones hasta que se termine la impresión.

4.4.3. Sistema Operativo Monousuario

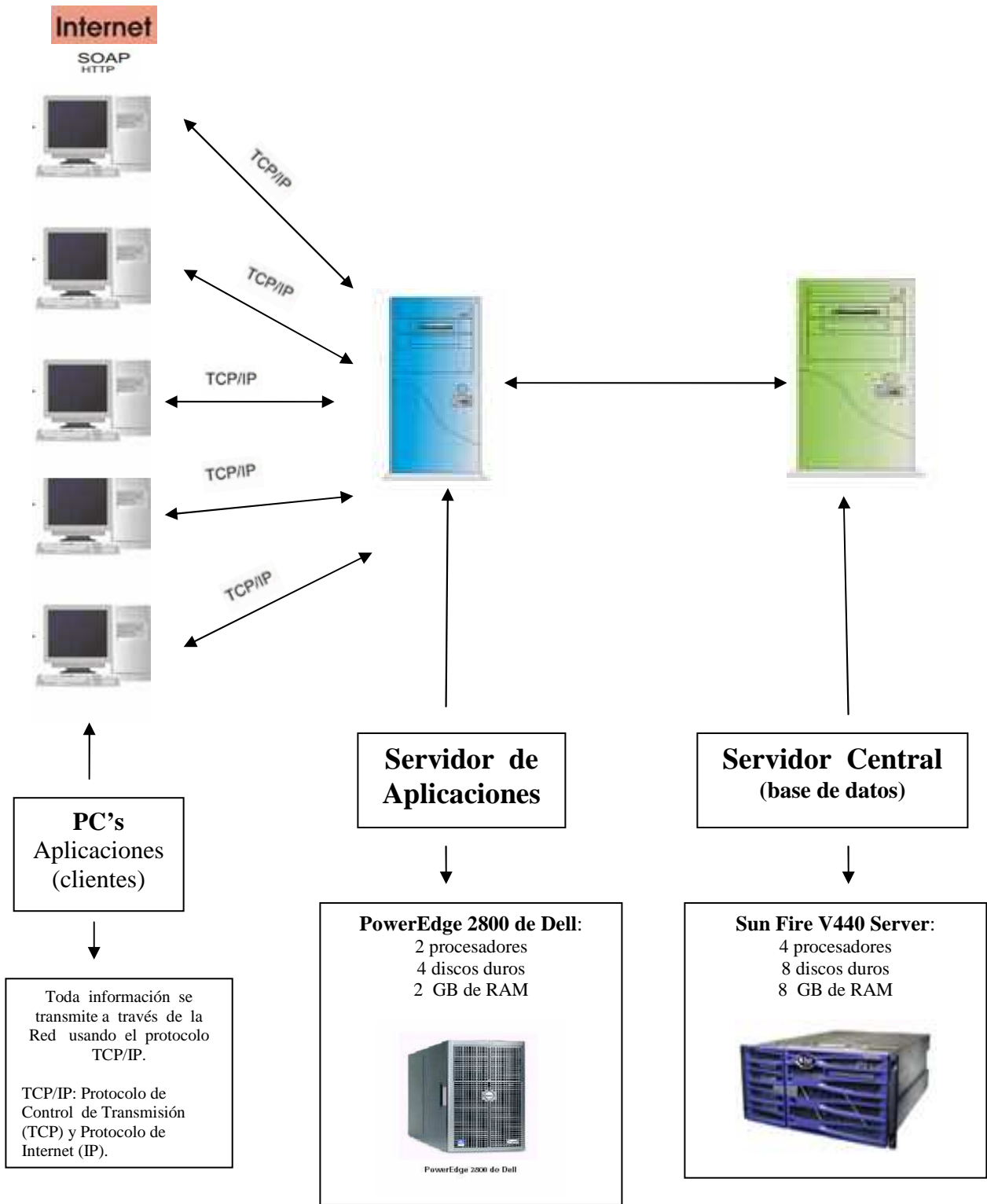
Los sistemas monousuarios son aquellos que nada más puede atender a un solo usuario, gracias a las limitaciones creadas por el hardware, los programas o el tipo de aplicación que se este ejecutando. Estos tipos de sistemas son muy simples, porque todos los dispositivos de entrada, salida y control dependen de la tarea que se esta utilizando, esto quiere decir, que las instrucciones que se dan, son procesadas de inmediato; ya que existe un solo usuario. Y están orientados principalmente por los microcomputadores.

4.4.4. Sistema Operativo Multiusuario

Multiusuario: de multi: varios; y usuarios: "apto para ser utilizado por muchos usuarios". Significa que puede estar ocupado por varios usuarios al mismo tiempo, lo cual permite reducir los tiempos ociosos en el procesador, e indirectamente la reducción de los costos de transmisión, energía y equipamiento para resolver las necesidades de cómputo de los usuarios. Ejemplo de este SO es Unix.

Un sistema operativo multiusuario, a diferencia de uno monousuario, debe resolver una serie de complejos problemas de administración de recursos, memoria, acceso al sistema de archivos, etc. Como ejemplo se muestra a continuación el sistema de funcionamiento de la Universidad Mayor de San Simón (Servidor UMSS, Inscripciones) figura 16 .

Figura 16. Sistema Multiusuario UMSS (Fuente: UMSS)



4.5. Sistemas Operativos Microsoft Windows

En la tabla 3.5. se presentan todos los sistemas operativos Microsoft Windows

Tabla 3.5. (www.microsoft.com/windowsxp/default.asp)

Sistemas Operativos Microsoft	Características
Windows 1.0	Fue lanzado en 1985 y participaron 55 programadores en su desarrollo, con las siguientes características: . Interfaz grafica con menús desplegables y soporte para Mouse (ratón). . Gráficos de pantalla e impresora independientes del dispositivo.
Windows 2	Fue lanzado en 1987 tenía mas características que Windows 1.0 como: . Iconos y ventanas traslapadas. . capacidad de ejecutar varias aplicaciones en DOS simultáneamente en memoria extendida. Se desarrollaron aplicaciones especialmente para este sistema como: . Excel, Word FOR Windows, Corel Draw, Page Maker, Ami.
Windows 386	Fue lanzado en 1987 y a pesar de ser equivalente a Windows 286, podía ejecutar varias aplicaciones en DOS en memoria extendida (multitarea) . cabe notar que cuando se lanzo Windows 386, Windows 2 fue renombrado como Windows 286
Windows 3.0	Fue lanzado en 1990 se vendió mas de 10 millones de copias, estas sus principales características: . Capacidad de memoria principal de direccionar mas de 640 K. . Modo estándar (286), con soporte para memoria grande (large memory). . Modo mejorado 386, con memoria grande y soporte de múltiples sesiones DOS. . Incorporación del Administrador de programas y Administrador de archivos. . Incorporación del soporte de Red. . Incorporación del soporte para mas de 16 colores de video.
Windows 3.1	Fue una versión de Windows con muchas mejoras de Windows 3.0 como : . Incorporación de soportes para fuentes True TYPE y OLE. . No existe soporte para el modo real (8086) o equivalentes. . Incorporación de soporte para Multimedia. . Incorporación de la capacidad para que una aplicación reinicie la maquina. . Incorporación del soporte API de multimedia y Red.
Windows FOR Workgroups 3.1	Es una versión de Windows 3.1 que puede trabajar en Red el cual se mejoro con Windows FOR Workgroups. Características principales de Windows FOR Workgroups (Windows para grupos de trabajo): . Contiene capacidades para conexión punto a punto que permite compartir archivos y impresoras. . Los archivos pueden ser accedidos desde otras maquinas corriendo DOS o Windows.
Windows 95	Características importantes de Windows 95: . Es un sistema operativo Multitarea dotada de una interfaz grafica de usuario. . Windows 95 no necesita del MS-DOS para ser ejecutado ya que es un sistema operativo completo. . Este sistema operativo esta basado en menús desplegables , ventanas en pantalla y uso del apuntador (Mouse). . Los nombres de los archivos no están restringidos a ocho caracteres y tres de extensión si no a 256 caracteres. . Posee Plug and Play tecnología con la cual un usuario puede conectar o instalar dispositivos permitiendo al sistema automáticamente instalar los controladores del Hardware sin intervención del usuario. . Se incorpora el soporte de Red (TCP/IP, IPX, SLIP, PPP Y Windows Sockets).

Windows 98	<p>Fue liberado en 1998 con las siguientes características principales:</p> <ul style="list-style-type: none"> . Incorpora al Windows 95 el Internet Explorer 4.0. . Incorpora el soporte para sistema de archivos FAT32 (File Allocation Table o Tabla de Asignación de Archivos). . Incorporación de soporte para el manejo de puertos USB (Universal Serial Bus o Bus de Serie Universal). . Incorporación de reproductor de DVD.
Windows Me	<p>Windows Me (Windows Millenium Edition) tiene las siguientes características principales:</p> <ul style="list-style-type: none"> . Incorpora un reproductor de música, que permite la transmisión de flujos (streaming) en formato de video y audio . Eliminación del modo Real (ejecución del DOS antes de correr Windows al arrancar el equipo) generando mayor velocidad al momento de cargar el sistema operativo. . Incorporación de una protección de archivos del Sistema y el soporte para la restauración del Sistema.
Windows NT	<p>Lanzado el 24 de mayo de 1993, sistema operativo para redes brinda poder, velocidad y las siguientes características:</p> <ul style="list-style-type: none"> . Contiene todo el software necesario para trabajar en redes, permitiendo ser un cliente de la Red o un servidor. . Es un sistema operativo de 32 Bits . Es un sistema Multiusuario, Multitarea y Multiprocesador
Windows 2000	<p>Es la unificación de los sistemas operativos Windows 9x y Windows NT con las siguientes características:</p> <ul style="list-style-type: none"> . Contiene un soporte de Hardware de la interfaz renovada. . Incorporación del Internet Explorer 5. . Soporte para nuevas tecnologías como USB, FAT32, NTFS (New Tecnology File Sistem) . Contiene un administrador avanzado de energía . Mejora de las aplicaciones para el trabajo con redes (redes locales y Internet) etc. . Es un sistema operativo orientado al trabajo en Red y compartición de recursos. . Windows 2000 esta integrado por cuatro versiones presentadas a continuación.
1) Windows 2000 Professional	<p>Características principales:</p> <ul style="list-style-type: none"> . Esta destinado a ser un cliente de Red seguro y una estación de trabajo Multiusuario. . Con soporte hasta de dos procesadores y es útil como un sistema operativo Autónomo. . Microsoft lo promociona como el principal sistema operativo de escritorio en un entorno de negocios.
2) Windows 2000 Server	<p>Características principales:</p> <ul style="list-style-type: none"> . Con soporte hasta de cuatro procesadores . Puede ser utilizado como controlador de dominio, servidor de impresión, servidor de archivos, servidor de aplicaciones e incluso como servidor de Internet de una empresa pequeña o mediana (servidor de Web, correo etc.)
3) Windows 2000 Advanced Server	<p>Características principales:</p> <ul style="list-style-type: none"> . Con soporte hasta de ocho procesadores . Puede trabajar como servidor departamental de aplicaciones en empresas medianas o grandes con mas de un dominio y tareas de misión critica. . Entre otras prestaciones incluye una tolerancia a fallas de Hardware.
4) Windows 2000 Data Center Server	<p>Características principales:</p> <ul style="list-style-type: none"> . Con soporte hasta de 32 procesadores solo es vendido sobre pedido . Destinado a ser utilizado por grandes empresas con requerimientos Data Warehousing, análisis econométricos, simulaciones científicas e ingenieriles a gran escala etc.

Windows XP	<p>Fue lanzado el 25 de octubre del 2001 con las nuevas características siguientes:</p> <ul style="list-style-type: none"> . Incorporación de una nueva interfaz a la cual se le dio el nombre de Luna (Moon). . Se mejoraron diversas características de redes, entre las que se incluyen soporte para redes inalámbricas, soporte para usuarios móviles y nuevas herramientas para la reparación de redes. . En Windows XP se incluye System Restore, para poder restablecer la configuración del sistema en caso de errores graves como los producidos en la instalación de un nuevo controlador. . De acuerdo a pruebas realizadas Windows XP es mas estable que Windows 98 o Windows Me y tan estable como Windows 2000, sin embargo se debe tener cuidado en la instalación de viejas aplicaciones o productos de software en micro-computadoras que ejecutan Windows XP. <p>Windows XP presenta tres versiones bien diferenciadas presentadas a continuación.</p>
1) Windows XP Home	. Esta Versión esta destinada al usuario domestico y sustituye al Windows 9x.
2) Windows XP Professional	<p>Características principales:</p> <ul style="list-style-type: none"> . Es el sucesor natural de Windows 2000 Professional. . Esta orientado a usuarios avanzados que trabajan en ambientes de redes.
3) Windows XP Professional de 64 bits	<p>Características principales:</p> <ul style="list-style-type: none"> . Windows XP Professional permite que una computadora que este ejecutando este sistema operativo, se puedan conectar mas de cinco computadoras a la misma a través de la Red, además de dar soporte a múltiples procesadores (cuando se cuenta con el software correspondiente) y hacer las veces de servidor Web. . Cada aplicación ejecutada por Windows XP es un proceso separado e independiente de los demás, por tanto cuando una aplicación sufre problemas se puede finalizar sin afectar al resto de las aplicaciones ni al propio sistema operativo . Las versiones de Windows XP cuentan con un Firewall (muro de fuego) integrado para proteger al sistema cuando se conecta a Internet. . Windows XP integro un programa de mensajería, el MSM Messenger que paso a llamarse Windows Messenger que es sucesor del NetMeeting y que forma parte del sistema operativo. . Windows XP esta equipado para la reproducción de cualquier formato Multimedia y la grabación en casi cualquier formato, es capaz de reproducir DVD, de grabar CDs de audio y CD-RWs de datos.
Windows Vista (Longhorn)	<p>El 26 de febrero de 2006 la compañía Microsoft anuncio que la próxima versión de Windows incluiría 6 ediciones. las ediciones de Windows Vista y cambios respecto a Windows XP se mencionan a continuación:</p> <ul style="list-style-type: none"> . Windows Vista tendrá una interfaz grafica completamente rediseñada cuyo nombre en código es Aero. . Tendrá la capacidad nativa para grabar DVD. . Tendrá incluida la versión del navegador Internet Explorer 7.0 . La utilidad de restauración del sistema será actualizada e implementada como herramienta de inicio de sesión facilitando el "rescate" del sistema. . Incluirá el Firewall (muro de fuego) de sistema con la capacidad de bloquear las conexiones que salen del sistema sin previa autorización (conexión Internet). . Incorpora la barra lateral o "sidebar" que es semejante a la Dashboard de Apple OS X, con la cual el usuario tiene acceso a una serie de pequeños programas denominados "gadgets". . Windows Vista no nesecitara de antivirus ya que la seguridad que incorpora "superara cualquier cosa antes vista". . Windows Vista incorporara la herramienta BitLocker Drive Encryption, para la protección de datos extraviados. . Windows Viata incluirá en algunas ediciones el reproductor de medios Windows Media Player 11 (WMP11), las ediciones que carecerán de WMP11 son diseñadas para Europa, cumpliendo la disposición de la Unión Europea que obliga a eliminar la aplicación Windows Media Player cuando el sistema operativo es comercializado en Europa. . Windows Vista cargara aplicaciones un 15% mas rápido que Windows XP. . Windows Vista iniciara el sistema un 50% mas rápido que Windows XP. . Windows Vista podrá entrar en modo de suspensión en dos segundos. . Se reducirá en un 50% la cantidad de reinicios del sistema después de las actualizaciones del sistema. . Incorporara un nuevo instalador, capaz de instalar Windows Vista en 15 minutos.

5. LENGUAJES DE PROGRAMACION

Los lenguajes de programación son herramientas que nos permiten crear programas y software. Entre ellos tenemos Object Pascal, Basic, Pascal, Java, C, etc.

Una computadora funciona bajo control de un programa el cual debe estar almacenado en la unidad de memoria; tales como el disco duro. Los lenguajes de programación de una computadora en particular se conocen como código de máquina o lenguaje de máquina. Estos lenguajes codificados en una computadora específica no podrán ser ejecutados en otra computadora diferente.

Para que estos programas funcionen para diferentes computadoras hay que realizar una versión para cada una de ellas, lo que implica el aumento del costo de desarrollo. Por otra parte, los lenguajes de programación en código de máquina son verdaderamente difíciles de entender para una persona, ya que están compuestos de códigos numéricos sin sentido nemotécnico.

Los lenguajes de programación facilitan la tarea de programación, ya que disponen de formas adecuadas que permiten ser leídas y escritas por personas, a su vez resultan independientes del modelo de computador a utilizar.

Los lenguajes de programación representan en forma simbólica y en manera de un texto los códigos que podrán ser leídos por una persona

Existen estrategias que permiten ejecutar en una computadora un programa realizado en un lenguaje de programación simbólico. Los procesadores del lenguaje son los programas que permiten el tratamiento de la información en forma de texto, representada en los lenguajes de programación simbólicos.

Hay lenguajes de programación que utilizan un compilador.

La ejecución de un programa con compilador requiere de dos etapas:

- 1) Traducir el programa simbólico a código máquina.

2) Ejecución y procesamiento de los datos.

Otros lenguajes de programación utilizan un programa intérprete o traductor, el cual analiza directamente la descripción simbólica del programa fuente y realiza las instrucciones dadas.

El intérprete en los lenguajes de programación simula una máquina virtual, donde el lenguaje de máquina es similar al lenguaje fuente.

5.1 Lenguajes de Programación y su migración al computador

Los lenguajes de programación sirven para escribir programas que permitan la comunicación usuario / máquina. Unos programas especiales llamados programas traductores (compiladores e intérpretes) convierten las instrucciones escritas en lenguajes de programación en instrucciones escritas en lenguaje de máquina (0 y 1 bits) que esta pueda entender.

Las instrucciones dentro de la computadora se representan mediante números. Por ejemplo, el código para copiar puede ser 001. El conjunto de instrucciones que puede realizar una computadora se conoce como lenguaje de máquina o código máquina. En la práctica, no se escriben las instrucciones para las computadoras directamente en lenguaje de máquina, sino que se usa un lenguaje de programación de alto nivel que se traduce después al lenguaje de la máquina automáticamente, a través de programas especiales de traducción (intérpretes y compiladores) figura 19. Algunos lenguajes de programación representan de manera muy directa el lenguaje de máquina, como los ensambladores (lenguajes de bajo nivel) y, por otra parte, los entornos de desarrollo como Delphi, que usan el lenguaje Pascal se basan en principios abstractos muy alejados de los que hace la máquina en concreto (lenguajes de alto nivel).

Un programa compilador vuelve a escribir el programa inicial en lenguaje de máquina para que la UP pueda entenderlo. Esto se hace de inmediato y el programa final se

guarda en esta nueva forma. Un programa compilado se estima que será considerablemente más largo que el original.

Un programa Intérprete traduce las declaraciones del programa original a lenguaje de máquina, línea por línea, a medida que va corriendo dicho programa original. Un programa interpretado será más pequeño que uno compilado pero insumirá más tiempo para ser ejecutado.

5.2 Evolución de los Lenguajes de Programación

Tras el desarrollo de las primeras computadoras surgió la necesidad de programarlas para que realizaran las tareas deseadas.

Los lenguajes más primitivos fueron los denominados lenguajes máquina figura 17. Como el hardware se desarrollaba antes que el software, estos lenguajes se basaban en el hardware, con lo que cada máquina tenía su propio lenguaje y por ello la programación era un trabajo costoso, válido sólo para esa máquina en concreto.

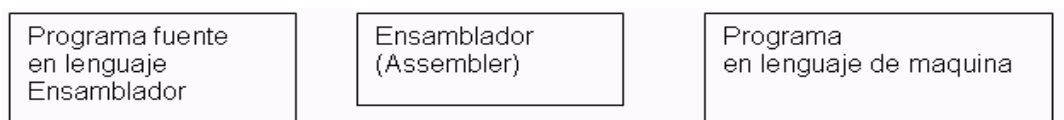


Figura 17. Lenguajes primitivos (Propia)

El primer avance fue el desarrollo de las primeras herramientas automáticas generadoras de código fuente. Pero con el permanente desarrollo de las computadoras, y el aumento de complejidad de las tareas, surgieron a partir de los años 50 los primeros lenguajes de programación de alto nivel.

Las organizaciones que se encargan de regularizar los lenguajes son ANSI (Instituto de las Normas Americanas) y ISO (Organización de Normas Internacionales). La evolución de los lenguajes de programación se muestra en la tabla 3.6. (fuente propia) :

Periodo	Influencias	Lenguajes
1950 - 1955	Computadoras primitivas	Lenguajes ensamblador. Lenguajes experimentales de alto nivel
1956 - 1960	Computadoras pequeñas, costos elevados y lentos Cintas magnéticas. Compiladores e intérpretes Optimización del código.	FORTTRAN ALGOL 58 y 60 COBOL LISP
1961 - 1965	Computadoras grandes y de costos elevados Discos magnéticos Sistemas operativos Lenguajes de propósito general.	FORTTRAN IV COBOL 61 Extendido ALGOL 60 Revisado APL
1961 - 1965	Ordenadores de diferentes tamaños, velocidades, y costos. Sistemas caros de almacenamiento masivo de datos. Sistemas operativos multitarea e interactivos. Compiladores con optimización. Lenguajes estándar, flexibles y generales.	FORTTRAN 66 (estándar) COBOL 65 (estándar) ALGOL 68 SIMULA 67 BASIC APL/360
1971 - 1975	Micro ordenadores. Sistemas pequeños y de almacenamiento masivo de datos. Programación estructurada. Lenguajes sencillos.	PASCAL COBOL 74
1976 - 1980	Ordenadores baratos y potentes. Sistemas distribuidos. Programación interactiva. Abstracción de datos. Programación con fiabilidad y fácil mantenimiento.	ADA FORTTRAN 77 PROLOG C
1980	Ordenadores más baratos y potentes. Mayor abstracción de datos. Menor costo de memorias	SmallTalk OOCOBOL C++ Object Basic
1985-1990	Ordenadores más baratos y potentes. Menor costo de memorias surgimiento del gran compilador Clipper	dBase dBase II dBase III+ dBase IV FoxBase (un clon de dBase III+)
1990-1995	Ordenadores más baratos y potentes. Menor costo de memorias desarrollaron librería externas inclusión de objetos compilador Clipper de Nantucket Corp programación con inclusión de objetos	lenguaje C y C++. Basic PASCAL lenguaje C. version 5.0 lenguajes Xbase
1995-2000	Ordenadores más baratos y potentes. Menor costo de memorias desarrollaron librería externas lenguajes orientados a objetos	OO Pascal JAVA BASIC Lenguaje C y C++

Tabla 3.6. Evolución de los lenguajes (Propia)

5.3 Lenguaje de Máquina

El lenguaje máquina es el único que entiende directamente la computadora, utiliza el alfabeto binario que consta de los dos únicos símbolos 0 y 1, denominados bits (abreviatura inglesa de dígitos binarios). Fue el primer lenguaje utilizado en la programación de computadoras, pero dejó de utilizarse por su dificultad y complicación, siendo sustituido por otros lenguajes más fáciles de aprender y utilizar, que además reducen la posibilidad de cometer errores.

Ejemplo:

0001	1010	0001	01 A1
1001	1001	1010	89 9A
1010	1001	1100	3A 9C
0100	0111	0000	74 70
1001	0010	0000	E9 20

5.4 Lenguaje Ensamblador

El lenguaje ensamblador es el primer intento de sustituir el lenguaje máquina por otro más similar a los utilizados por las personas. En este lenguaje cada instrucción equivale a una instrucción en lenguaje máquina, utilizando para su escritura palabras nemotécnicas en lugar de cadenas de bits.

Ejemplo:

INICIO:	ADD	B, 1
	MOV	A, E
	CMP	A, B
	JE	FIN
	JMP	INICIO
FIN :	END	

Este lenguaje presenta la mayoría de los inconvenientes del lenguaje máquina:

1. Cada modelo de computadora tiene un lenguaje ensamblador propio diferente del de los demás, por lo cual un programa sólo puede utilizarse en la máquina para la cual se programó.
2. El programador tiene que conocer perfectamente el hardware del equipo, ya que maneja directamente las posiciones de memoria, registros del procesador y demás elementos físicos.
3. Todas las instrucciones son elementales, es decir, en el programa se deben describir con el máximo detalle todas las operaciones que se han de llevar a cabo en la máquina para la realización de cualquier proceso.

Por otro lado, tanto el lenguaje máquina como el ensamblador gozan de la ventaja de mínima ocupación de memoria y mínimo tiempo de ejecución en comparación con el resultado de la compilación del programa equivalente escrito en otros lenguajes.

5.5 Lenguajes de Alto Nivel

La programación en un lenguaje de bajo nivel como el lenguaje de la máquina o el lenguaje simbólico tiene ciertas ventajas y desventajas:

1. Mayor adaptación al equipo.
2. Posibilidad de obtener la máxima velocidad con mínimo uso de memoria.

Pero también tiene importantes inconvenientes:

1. Imposibilidad de escribir código independiente de la máquina.
2. Mayor dificultad en la programación y en la comprensión de los programas.

Por esta razón, a finales de los años 1950 surgió un nuevo tipo de lenguaje que evitaba los inconvenientes, a costa de ceder un poco en las ventajas.

Estos lenguajes se llaman "de tercera generación" o "de alto nivel", en contraposición a los "de bajo nivel" o "de nivel próximo a la máquina".

Los Lenguajes de alto Nivel son aquellos que se encuentran más cercanos al lenguaje natural que al lenguaje máquina. Están dirigidos a solucionar problemas mediante el uso de EDD's.

Nota: EDD's son las abreviaturas de Estructuras Dinámicas de Datos, algo muy utilizado en todos los lenguajes de programación. Son estructuras que pueden cambiar de tamaño durante la ejecución del programa. Nos permiten crear estructuras de datos que se adapten a las necesidades reales de un programa.

Se tratan de lenguajes independientes de la arquitectura del ordenador. Por lo que, en principio, un programa escrito en un lenguaje de alto nivel, se puede migrar de una máquina a otra sin ningún tipo de problema esquema figura 18.

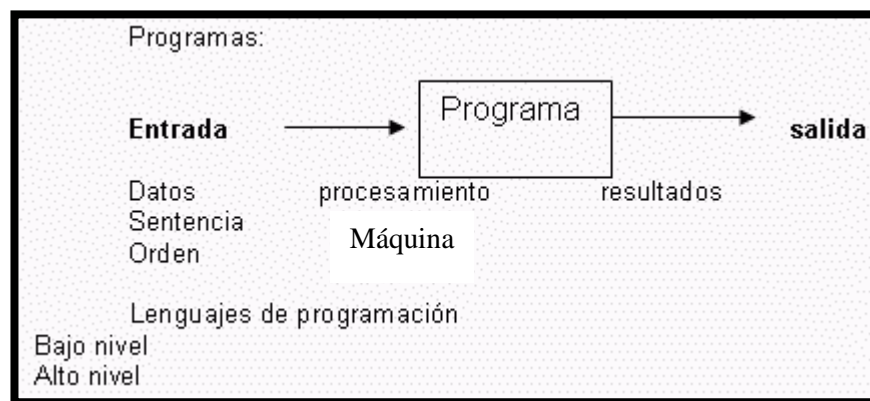


Figura 18. Migración de un lenguaje de alto nivel (Propia)

Estos lenguajes permiten al programador olvidarse por completo del funcionamiento interno. Tan solo necesitan un traductor que entienda el código fuente como las características de la máquina (compilador).

Suelen usar tipos de datos para la programación y hay lenguajes de propósito general (cualquier tipo de aplicación) y de propósito específico (como Fortran para trabajos científicos).

Principales Lenguajes de Alto Nivel

- Ada
- ALGOL
- Basic
- C (en realidad es un lenguaje de medio nivel).
- C++
- C#
- Fortran
- Java
- Lexico
- Logo
- **OO Pascal**
- Perl
- PHP
- PL/SQL
- Python
- Modula-2
- Lenguajes funcionales
 - Haskell
 - Lisp

5.6 Compilación del Lenguaje de Alto Nivel

5.6.1 Compilador:

Diagrama de bloques de la operación de un buen compilador figura 19:

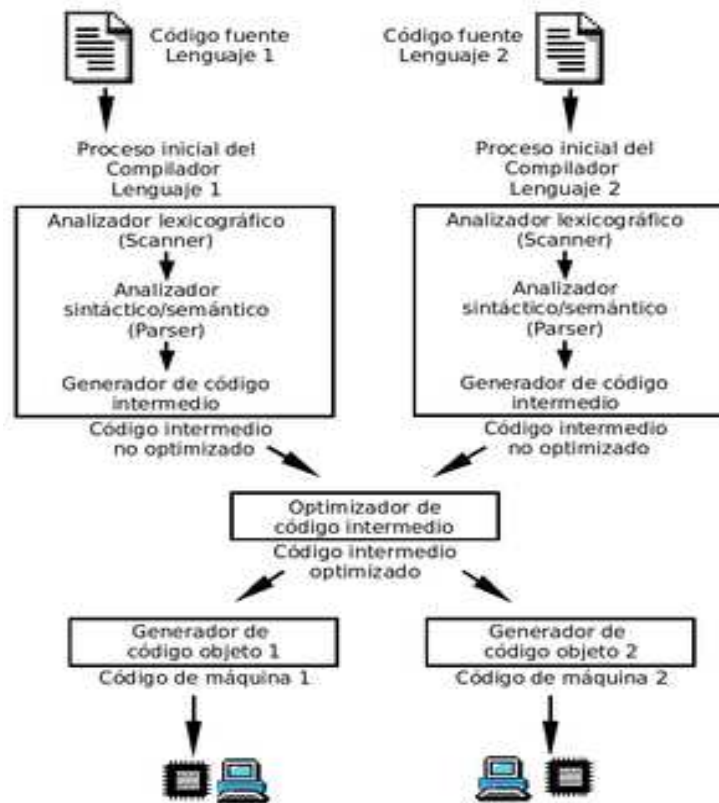


Figura 19. Diagrama de bloques de operación de un compilador
([http://www.Historia.de.la.Computación.\(Informática\).-ilustrados_com.htm](http://www.Historia.de.la.Computación.(Informática).-ilustrados_com.htm))

Un compilador acepta programas escritos en un lenguaje de alto nivel y los traduce a otro lenguaje, generando un programa equivalente independiente, que puede ejecutarse tantas veces se quiera. Este proceso de traducción se conoce como compilación. En un compilador hay que distinguir tres lenguajes diferentes tales como:

1. El de los programas de partida (LA).
2. El de los programas equivalentes traducidos (LB), normalmente el Lenguaje de máquina.
3. El lenguaje en que está escrito el propio compilador (LC), que puede ser igual o diferente a LA. Aumenta la portabilidad del compilador si está escrito en el mismo lenguaje, es decir, se puede compilar a sí mismo.

Los programas interpretados suelen ser más lentos que los compilados, pero los intérpretes son más flexibles como entornos de programación y depuración.

5.6.2 Partes de un Compilador

Normalmente los compiladores están divididos en dos partes:

1. Front END: es la parte que analiza el código fuente, comprueba su validez, genera el árbol de derivación y rellena los valores de la tabla de símbolos. Esta parte suele ser independiente de la plataforma o sistema para el cual se vaya a compilar.

2. Back END: es la parte que genera el código máquina, específico de una plataforma, a partir de los resultados de la fase de análisis, realizada por el Front **END**.

Esta división permite que el mismo Back **END** se utilice para generar el código máquina de varios lenguajes de programación distintos y que el mismo Front **END** que sirve para analizar el código fuente de un lenguaje de programación concreto sirva para la generación de código máquina en varias plataformas distintas.

El código que genera el Back **END** normalmente no se puede ejecutar directamente, sino que necesita ser enlazado por un programa enlazador (linker).

5.6.3. Tipos de Compiladores

Esta taxonomía de los tipos de compiladores no es excluyente, por lo que puede haber compiladores que se adscriban a varias categorías:

- **Compiladores cruzados:** generan código para un sistema distinto del que están funcionando.
- **Compiladores optimizadores:** realizan cambios en el código para mejorar su eficiencia, pero manteniendo la funcionalidad del programa original.
- **Compiladores de una sola pasada:** generan el código máquina a partir de una única lectura del código fuente.

- **Compiladores de varias pasadas:** necesitan leer el código fuente varias veces antes de poder producir el código máquina.
- **Compiladores JIT** (Just In Time): forman parte de un intérprete y compilan partes del código según se necesitan.

Pauta de creación de un compilador: En las primeras épocas de la informática, el software de los compiladores era considerado como uno de los más complejos existentes.

Los primeros compiladores se realizaron programándolos directamente en lenguaje máquina o en ensamblador. Una vez que se dispone de un compilador, se pueden escribir nuevas versiones del compilador (u otros compiladores distintos) en el lenguaje que compila ese compilador.

Actualmente existen herramientas que facilitan la tarea de escribir compiladores ó intérpretes informáticos. Estas herramientas permiten generar el esqueleto del analizador sintáctico a partir de una definición formal del lenguaje de partida, especificada normalmente mediante una gramática formal y barata, dejando únicamente al programador del compilador la tarea de programar las acciones semánticas asociadas.

5.7 Intérprete

En informática, un **intérprete** es un programa capaz de analizar y ejecutar otros programas, escritos en un lenguaje de alto nivel. Los intérpretes suelen contraponerse a los compiladores, ya que mientras que los segundos se encargan de traducir un programa desde su descripción en un lenguaje de programación al código máquina del sistema destino, los primeros sólo realizan la traducción a medida que sea necesario y normalmente, no guardan el resultado de dicha traducción.

Comparando su actuación con la de un ser humano, un compilador equivale a un traductor profesional que, a partir de un texto, prepara otro independiente traducido a

otra lengua, mientras que un intérprete corresponde al intérprete humano, que traduce de viva voz las palabras que oye, sin dejar constancia por escrito.

En la actualidad, uno de los entornos más comunes de uso de los intérpretes informáticos es Internet, debido a la posibilidad que estos tienen de ejecutarse independientemente de la plataforma.

6. BIBLIOGRAFIA

- Luis Joyanes Aguilar PROGRAMACION BASIC PARA MICROCOMPUTADORAS 2da Edicion, McGraw-Hill 1986
- R.H. Hamnond, W.B. Rogers, J.B. Crittenden INTRODUCCION AL FORTRAN 77 Y LA PC McGraw-Hill Interamericana de Mexico, S.A. de C.V. 1989

Paginas de Internet:

- http://www.Historia de la Computación - Monografias_com.htm
- <http://www.Historia de la Computación.htm>
- <http://www.Historia de la Computación2.htm>
- <http://www.Historia de la computación4.htm>
- <http://www.La Computación en el Tiempo.htm>
- <http://www.Linea del tiempo de la computacion.htm>
- <http://www.BIOS.htm>
- <http://www.BIOS - Wikipedia, la enciclopedia libre.htm>
- <http://www.Discos duros y particiones.htm>
- <http://www.Disco duro - Wikipedia, la enciclopedia libre.htm>
- [http://www.El Disco Duro \(HD\) - Monografias_com.htm](http://www.El Disco Duro (HD) - Monografias_com.htm)
- <http://www.usuarios1.htm>
- [http://www.Historia de la Computación \(Informatica\) - ilustrados_com.htm](http://www.Historia de la Computación (Informatica) - ilustrados_com.htm)
- <http://www.microsoft.com/windowsxp/default.asp>
- <http://www.METODOLOGIA.htm>
- <http://es.wikipedia.org/wiki/Computadora>
- http://www.delphi3000.com/articles/article_3228.aspHistoria de la Computación.htm
- <http://www.swissdelphicenter.ch/en/showcode.php?id=1155>Historia de la computación4.htm

CAPITULO 2

PROCESOS LÓGICOS

1. INTRODUCCION

La realización de Trabajos Mediante el uso de una computadora, como cualquier otra actividad (Ingeniería, Arquitectura, Informática, etc.) requiere un método que explique de un modo ordenado y secuencial hasta los últimos detalles a realizar por la máquina.

Al igual que para construir un edificio, la empresa constructora no comienza por el techo, sino que encarga a un profesional el diseño de los planos arquitectónicos (arquitecto), estudio de suelos, diseño estructural con sus respectivos planos (Ingeniero civil), cronograma de actividades, etc. O para realizar una aplicación informática, sobre todo si tiene cierta complejidad, no debe comenzar nunca por la codificación del programa, sino que exige una serie de fases previas destinadas a conocer todos los aspectos del problema planteado y estudiar las posibles soluciones.

Para ello se pueden utilizar: Algoritmos, Pseudocodigos, Diagramas de Flujo.

2. ALGORITMO

2.1 Concepto

La definición de algoritmo aún no cuenta con la formalidad científica que podría ser ideal para ciencias como las matemáticas y las ciencias de la computación (donde los algoritmos son esenciales pero a falta de formalidad no pueden incluirse fácilmente en las demostraciones formales de estas ciencias). Sin embargo, si existe un concepto intuitivo de algoritmo.

Un algoritmo es un sistema por el cual se llega a una solución, teniendo en cuenta que debe de ser definido, finito y preciso. Por preciso se entiende que cada paso a seguir tiene un orden; finito implica que tiene un determinado número de pasos, o sea, que tiene un fin; y definido, que si se sigue el mismo proceso más de un vez llegaremos al mismo resultado.

Un **algoritmo** es un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema. De un modo más formal, un algoritmo es una secuencia finita de operaciones realizables, no ambiguas, cuya ejecución da una solución de un problema en un tiempo finito.

El término algoritmo no está exclusivamente relacionado con las matemáticas, ciencias de la computación o informática. En realidad, en la vida cotidiana se emplean algoritmos en multitud de ocasiones para resolver diversos problemas. Algunos ejemplos son el uso de una lavadora (se siguen las instrucciones), o la preparación de una comida (siguiendo los pasos de una receta). También existen ejemplos de índole matemática, como el algoritmo de la división para calcular el cociente de dos números, el algoritmo de Euclides para calcular el máximo común divisor de dos enteros positivos, o incluso el método de Gauss para resolver sistemas de ecuaciones.

2.2 Estructura Básica:

1. Inicio
2. Datos de entrada (operaciones básicas)
3. Procesamiento de datos
4. Datos de salida
5. Fin

En la programación de los algoritmos encontramos tres fases fundamentales, las cuales son: identificación del problema, análisis y desarrollo del problema e implementación.

En la programación de algoritmos la identificación del problema no es más que, que es lo que se quiere hacer, para ello en esta etapa, se encuentra información de entrada o inicial, que servirá para el análisis del problema.

La segunda etapa o fase es analizar el problema, en esta etapa es conveniente dividir o segregar las tareas necesarias e identificadas que ayudaran a la solución del problema dado. De esta forma se simplificaran y serán más comprensibles para su desarrollo, es recomendable siempre ir de las tareas o actividades más simples a las más complejas. En esta fase se transforma la información de entrada recibida.

La tercera etapa en la programación de algoritmos, consiste en la implantación y puesta en desarrollo del mismo, aquí se obtiene la información y resultado final resultante de las etapas anteriores.

En la programación de algoritmos, se utiliza una nomenclatura llamada pseudocódigo, una vez realizados los algoritmos en pseudocódigo, estos pueden ser traducidos en cualquier lenguaje de programación que lo soporte.

2.3 Implementación

Algunas veces, en una red neuronal biológica (por ejemplo, el cerebro humano implementa la aritmética básica o, incluso, una rata sigue un algoritmo para conseguir comida), también en circuitos eléctricos, en instalaciones industriales o maquinaria pesada.

El análisis y estudio de los algoritmos es una disciplina de las ciencias de la computación, y en la mayoría de los casos, su estudio es completamente abstracto sin usar ningún tipo de lenguaje de programación ni cualquier otra implementación; por eso, en ese sentido, comparte las características de las disciplinas matemáticas. Así, el análisis de los algoritmos se centra en los principios básicos del algoritmo, no en los de la implementación particular. Una forma de plasmar (o algunas veces *codificar*) un algoritmo es escribirlo en pseudocódigo .

Algunos escritores restringen la definición de *algoritmo* a procedimientos que deben acabar en algún momento, mientras que otros consideran procedimientos que podrían ejecutarse eternamente sin pararse, suponiendo el caso en el que existiera algún dispositivo físico que fuera capaz de funcionar eternamente. En este último caso, la finalización con éxito del algoritmo no se podría definir como la terminación de éste con una salida satisfactoria, sino que el éxito estaría definido en función de las secuencias de salidas dadas durante un periodo de vida de la ejecución del algoritmo. Por ejemplo, un algoritmo que verifica que hay más ceros que unos en una secuencia binaria infinita debe ejecutarse siempre para que pueda devolver un valor útil. Si se implementa correctamente, el valor devuelto por el algoritmo será válido, hasta que evalúe el siguiente dígito binario. De esta forma, mientras evalúa la siguiente secuencia podrán leerse dos tipos de señales: una señal positiva (en el caso de que el número de ceros es mayor que el de unos) y una negativa en caso contrario. Finalmente, la salida de este algoritmo se define como la devolución de valores exclusivamente positivos si hay más ceros que unos en la secuencia, y en cualquier otro caso, devolverá una mezcla de señales positivas y negativas.

3 PSEUDOCODIGO

El Pseudocódigo es una herramienta de programación en la que las instrucciones se escriben en palabras similares, en castellano o ingles, a las instrucciones del lenguaje utilizado. Facilita tanto la escritura como la lectura de los programas. El pseudocódigo es una **herramienta de representación de algoritmos**, en lenguaje natural acotado.

El pseudocódigo que es un lenguaje no formal, muy simple tal como Léxico cuyos códigos pueden estar en el idioma del programador.

Aunque no existan reglas de escritura para el pseudocódigo, se debe establecer ciertas convenciones:

- Las palabras reservadas (INICIO, LEER, EJECUTAR, MOSTRAR O IMPRIMIR, FIN) representan: Inicio de programa, lectura de datos, realizar una instrucción o hacer alguna operación, imprimir en pantalla y fin del programa respectivamente.

- Las condiciones lógicas a ser evaluadas deben estar dentro de paréntesis, y tienen dos bifurcaciones o caminos, uno cuando el valor de la condición es verdadera y la otra cuando es falsa, por ejemplo:

Si ($A > 10$) Entonces

Ejecutar proceso1

Sino

Ejecutar proceso2

Fin del Si

- Los comentarios se colocarán entre (* *), también pueden utilizarse los corchetes { }, y 2 barras inclinadas //.
- Siempre que se necesite realizar una operación matemática, el dato que recibe el resultado de la operación se debe colocar a la izquierda del signo igual (=), y a su derecha, se colocará la operación correspondiente. Ejemplo: $a = b + c$
- Siempre que se quiera obtener un dato de entrada (ingresado desde el teclado de la computadora), se debe utilizar la palabra reservada Leer. Ejemplo: Leer edad.
- Siempre que se quiera devolver un dato de salida (para visualizar por pantalla y/o impresora), se debe utilizar la palabra reservada Mostrar o Imprimir. Ejemplo: Imprimir promedio

4. DIAGRAMAS DE FLUJO

Los diagramas de flujo representan la forma más tradicional para especificar los detalles algorítmicos de un proceso. Se utilizan principalmente en programación, economía, procesos industriales, etc. Estos diagramas utilizan una serie de símbolos con significados especiales. Que son la representación gráfica de los pasos de un proceso, que se realiza para entender mejor al mismo. Son modelos tecnológicos utilizados para comprender los rudimentos de la programación lineal.

Otra definición del diagrama de flujo es la siguiente:

"Es un esquema para representar gráficamente un algoritmo. Se basan en la utilización de diversos símbolos para representar operaciones específicas. Se les llama diagramas de flujo porque los símbolos utilizados se conectan por medio de flechas para indicar la secuencia de operación. Para hacer comprensibles los diagramas a todas las personas, los símbolos se someten a una normalización (ANSI/ISO American National Standard Institute/Internacional Organization **FOR** Standarization); es decir, se hicieron símbolos casi universales, ya que, en un principio cada usuario podría tener sus propios símbolos para representar sus procesos en forma de Diagrama de Flujo. Esto trajo como consecuencia que sólo aquel que conocía sus símbolos, los podía interpretar. La simbología utilizada para la elaboración de diagramas de flujo es variable y debe ajustarse a un patrón definido previamente."

4.1 Elementos de Diagramación

Principales Símbolos

Estandarizados según ISO 5807 (Norma para Diagramas de Flujo)

No es indispensable usar un tipo especial de símbolos para crear un diagrama de flujo, pero existen algunos ampliamente utilizados por lo que es adecuado conocerlos y utilizarlos, ampliando así las posibilidades de crear un diagrama más claro y comprensible para crear un proceso lógico y con opciones múltiples adecuadas.

Bloque de Inicio y Fin de programa o Terminal (Fig. 2.1)

Indican los límites del procedimiento considerado como principal. Es decir este símbolo indica el comienzo o punto final del programa.



Fig. 2.1
Bloque Inicio y Fin

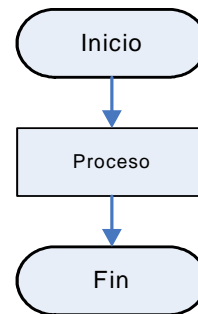


Fig. 2.2
Diagrama de flujo simple

Bloques de Proceso**Bloque de Acción Simple (Fig. 2.3)**

Representa una acción sencilla que puede ser considerada como única y que generalmente se codifica con una sola instrucción. Por ejemplo: *incrementar contador, operaciones aritméticas, transferencia de datos*, etc.



Fig. 2.3
Bloque de proceso

Bloque de Entrada desde teclado (Fig. 2.4)

Representa una acción simple de entrada de datos, generalmente desde un dispositivo periférico como el teclado. Por ejemplo: *ingresar valor, leer datos*, etc.

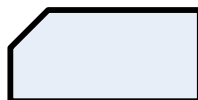


Fig. 2.4
Bloque de entrada

Bloque de Entrada/Salida – E/S (Fig. 2.5)

Representa una acción simple de entrada o salida de datos, Lectura, Escritura, generalmente desde o hacia un dispositivo periférico como el teclado, la pantalla o el disco. Por ejemplo: ingresar valor, leer registro, mostrar resultado en pantalla, etc.



Fig. 2.5
Bloque de entrada/salida

Bloques de Decisión**Bloque de Decisión Simple (Fig. 2.6)**

Representa la acción de analizar el valor de verdad de una condición, que sólo puede ser verdadera o falsa (*selección simple*). Según el resultado de esta evaluación se sigue uno u otro curso de acción. Por lo tanto, de un bloque de decisión simple siempre salen exactamente dos flujos, uno por Verdad (*Sí*) y otro por Falso (*No*).

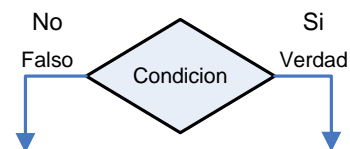
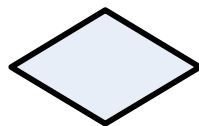


Fig. 2.6
Bloque de decisión

Bloque de Decisión Múltiple

Representa la acción de analizar el valor de una variable, que puede tomar uno entre una serie de valores conocidos (*selección múltiple*). Según el resultado de esta evaluación, se sigue uno entre varios cursos de acción. Por lo tanto, de un bloque de decisión múltiple siempre salen varios flujos, uno por cada valor esperado de la variable analizada.

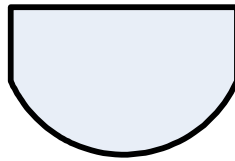


Fig. 2.6
Bloque de decisión Múltiple

Flujos y Conectores

Conector (Fig. 2.7)

Representa un punto de conexión entre procesos. Se utiliza cuando es necesario dividir un diagrama de flujo en varias partes, por ejemplo por razones de espacio o simplicidad. Una referencia debe darse dentro para distinguirlo de otros. La mayoría de las veces se utilizan números en los mismos. Son conectores en la misma página, también a otras páginas.

Flecha o Flujo (Fig. 2.8)

Indica la secuencia en que se van ejecutando las acciones al pasar de un bloque a otro.



Fig. 2.7
Conector misma pagina

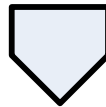


Fig. 2.7.1
Conector a otra pagina



Fig. 2.8
Flechas de Flujo

Bloque de Iteración

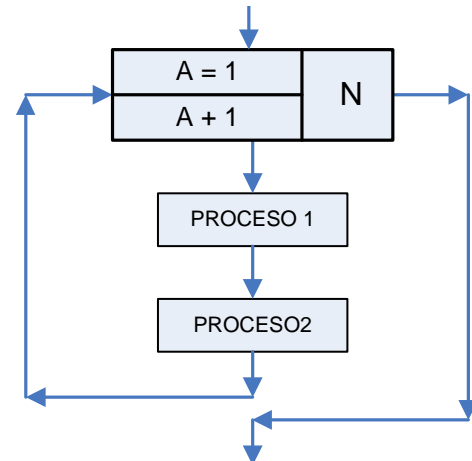
Indica la ejecución de una serie de operaciones un número determinado de veces (límite); las operaciones serán siempre las mismas, pero con datos y resultados diferentes.

Consta de 3 propiedades

1. Inicialización de la variable ($A=1$).
2. Incremento o Decremento de la variable.
 - Caso de incremento $A+1$ ($A+2$, $A+3$, etc., según el caso requerido)
 - Caso de decremento $A-1$.
3. Límite (definido por el usuario N).



Fig. 2.7
Bloque de iteración y aplicación



Bloque de Salida

Que realiza la presentación de resultados en impresora o en pantalla.

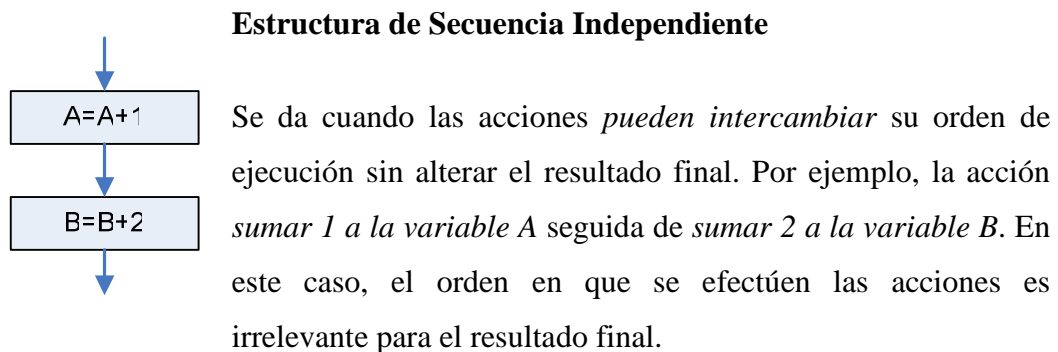


Fig. 2.8
Bloque de salida

4.2 Estructuras de Diagramas

4.2.1 Estructura de Secuencia

Se da cuando una acción sigue a la otra. Es la más simple y la más común de todas y constituye la esencia de toda tarea programada. Se reconocen dos variantes básicas: la *secuencia independiente* y la *secuencia dependiente*.



Estructura de Secuencia Dependiente

Se da cuando las acciones *no pueden intercambiar* su orden de ejecución sin alterar el resultado final. Por ejemplo, la acción *sumar 1 a la variable A* seguida de *multiplicar la variable A por 2*. En este caso, el orden en que se efectúen las operaciones es determinante del resultado final.

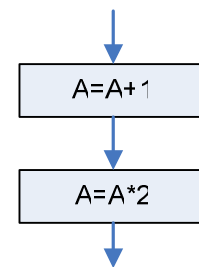


Fig. 4.2.1.2

4.2.2 Estructura de Selección

Es la que permite a los programas adaptarse a situaciones diversas verificando ciertas condiciones y tomando uno u otro curso de acción según corresponda. Se reconocen dos variantes básicas: la *selección simple* y la *selección múltiple*.

Estructura de Selección Simple

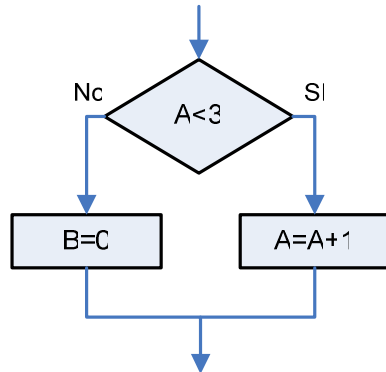


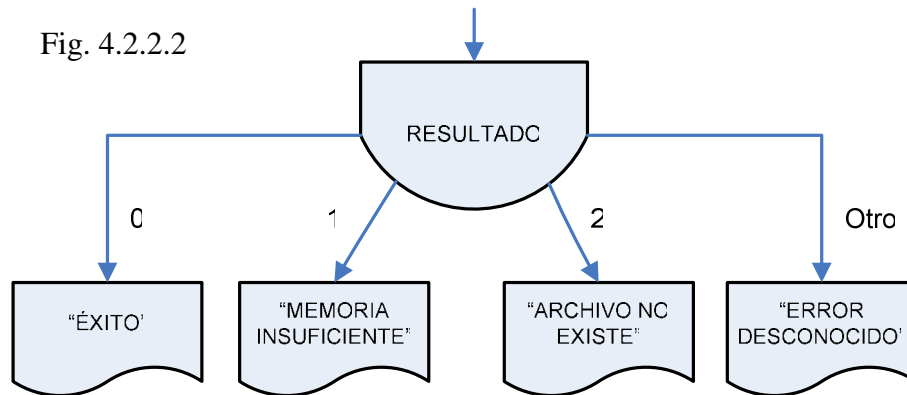
Fig. 4.2.2.1

Se da cuando existen *sólo dos alternativas*. Se evalúa una **condición** que puede tomar los valores lógicos de **verdadero** o **falso**. Si la condición es verdadera se hace una cosa y si es falsa se hace otra. Por ejemplo: *si la variable A es inferior a 3, sumarle 1; sino, poner la variable B en 0*. Es importante aclarar aquí que no siempre deben estar presentes las acciones correspondientes a ambas alternativas. En efecto, es posible y muy común que se ejecute alguna acción si se cumple alguna condición y que no se haga nada en caso contrario. Por ejemplo: *si la variable B es mayor que 20, asignarle el valor 0*. En este caso, si no se cumple la condición (esto es, si B no es superior a 20), simplemente no ocurre nada.

Estructura de Selección Múltiple

Se da cuando existen *más de dos alternativas*. En este caso la variable que determina la selección puede tomar uno entre varios valores numéricos enteros previstos y se ejecutará uno entre varios grupos de acciones, según corresponda. Puede haber también aunque no es obligatorio un curso de acción previsto para cuando la variable no toma ninguno de los valores predeterminados. Por ejemplo: *examinar la variable **resultado**; si es 0 escribir 'Éxito', si es 1 escribir 'Memoria insuficiente', si es 2 escribir 'Archivo no existe' y si es cualquier otro número escribir 'Error desconocido'*.

Fig. 4.2.2.2



4.2.3 Estructura de Iteración

Es la que permite a los programas efectuar una tarea extensa con un mínimo de código ejecutable, al reiterar una y otra vez la ejecución de un mismo conjunto de instrucciones. Esta iteración o repetición está controlada por una condición, llamada *condición de salida* que toma la forma de una selección simple y se verifica con cada ejecución del ciclo. Si la condición toma el valor adecuado (*verdadero o falso*, según corresponda) se ejecuta las instrucciones incluidas en el ciclo. En caso contrario se lo interrumpe y se abandona la estructura. Se reconocen dos variantes básicas: la *iteración con evaluación previa* y la *iteración con evaluación posterior*.

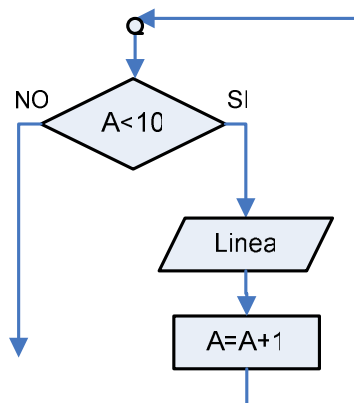


Fig. 4.2.3.1

Estructura de Iteración con Evaluación Previa

Se da cuando la condición de salida se evalúa *antes de la ejecución* de cada ciclo. Se verifica primero la condición y si resulta ser adecuada se ejecutan las acciones asociadas para volver a evaluar la condición. En este tipo de estructuras puede ocurrir que la condición sea inadecuada la primera vez que se evalúa y que, por lo tanto, las

acciones asociadas no lleguen a ejecutarse nunca. La iteración con evaluación previa debe utilizarse entonces en aquellos casos en que la ejecución de todo el ciclo esté sujeta al estado previo de una condición, y, por lo tanto, esté previsto que las acciones puedan no ejecutarse nunca. Por ejemplo: *mientras la variable A sea inferior a 10 escribir un salto de línea y sumarle 1 a la variable A*. En este caso, si ocurre que la variable A llega al ciclo con un valor superior a 10, es decir si la condición es inicialmente *falsa*- no se escribirá ningún salto de línea ni se incrementará la variable.

Estructura de Iteración con Evaluación Posterior

Se da cuando la condición de salida se evalúa *después de la ejecución* de cada ciclo. Se ejecutan primero las acciones asociadas al ciclo, se evalúa luego la condición y, si resulta ser adecuada, se repite el ciclo. En este tipo de estructuras ocurre que las acciones asociadas con el ciclo se ejecutan siempre, por lo menos una vez. Por lo tanto debe utilizarse entonces en aquellos casos en que la evaluación de la condición esté sujeta a la ejecución del ciclo y, por lo tanto, esté previsto que las acciones deban ejecutarse siempre, por lo menos una vez. Por ejemplo: *apretar una tecla mientras que la tecla apretada sea diferente de <Enter>*. En este caso es necesario primero oprimir por lo menos una tecla para poder evaluar luego la condición; si es <Enter> y repetir eventualmente la acción de oprimir una tecla.

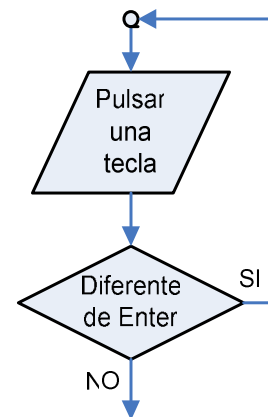


Fig. 4.2.3.2

4.3 Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operación operadores. Las expresiones matemáticas tienen igual sentido, por ejemplo:

$$X*(Y+5)-5*Z+3$$

Se pueden evaluar datos y por consiguiente obtener nuevos valores, evaluando expresiones.

- **Instrucciones de Asignación** Para ejecutar cálculos se necesitan instrucciones que indiquen a la computadora que acciones ha de ejecutar. La herramienta básica es la instrucción de asignación.

Las instrucciones de asignación son fundamentales en casi todos los lenguajes de programación. Permiten asignar valores a variables del programa.

Ejemplo: $A = 5$ El valor de 5 se asigna a la variable A.

- **Contadores** En los procesos repetitivos se necesita normalmente contar los sucesos o acciones internos del bucle, como pueden ser: elementos de un arreglo y número de iteraciones a realizar por el bucle. Para realizar esta tarea se utilizan los contadores, cuya construcción es una de las técnicas corrientes en la realización de cualquier diagrama de flujo.

Un contador es un campo de memoria que esta destinado a contener los diferentes valores que se van incrementando o decrementando en cada iteración.

Se deberá inicializar el contador, y esto consiste en poner el valor inicial de la variable que representa al contador, Ej. $A = 0$, $B = 100$, $C = -10$

Por consiguiente, el contador se representara por una instrucción de asignación del tipo:

$$A = A + 1 \qquad B = B - 1$$

Siendo “1” el incremento del contador (constante).

- **Acumulador** Un acumulador o totalizador es un campo o zona de memoria cuya misión es almacenar cantidades de variables, resultantes de sumas sucesivas. Realiza la función de un contador con la diferencia que el incremento o el decremento de cada suma es variable en lugar de constante como en el caso del contador.

Se representa por $S = S+N$; donde N es una variable y no una constante.

4.4 Reglas

De acuerdo al estándar ISO, los símbolos e incluso las flechas deben de tener ciertas características para estar dentro del estándar. En el caso de los círculos de conexión se debe usar sólo cuando se conecta con un proceso contenido dentro de la misma hoja.

También existen conectores de página, que se utilizan para unir actividades que se encuentran en otra hoja.

- Existe siempre un camino que permite arribar a una solución.
- Existe un único inicio del proceso
- Existe un único punto de fin para el proceso de flujo.

4.5 Diseño y Elaboración de Diagramas De Flujo

No existen reglas fáciles y rápidas para construir diagramas de flujo, pero existen guías muy útiles que se debe tener en mente.

Una vez realizado la solución del problema en pseudocódigo, podemos llevar a diagrama de flujo en forma mas sencilla, en caso de realizar directamente el diagrama de flujo tendremos en cuenta lo siguiente.

Cada diagrama de flujo es la solución a un problema determinado en el cual se observan:

- **Planteamiento del Problema** Esta dado por el enunciado del problema, el cual debe ser claro y completo.
- **Análisis del Problema (Procedimiento para su solución)** Entendido el problema es preciso analizar:
 - Los datos de entrada que nos suministran.

- Los datos o resultados que se esperan.
- El proceso al que se requiere someter los datos a fin de obtener los resultados esperados.
- Se debe tratar de proponer varias soluciones alternativas al problema esto con el fin de encontrar la mejor solución al problema planteado.

Una recomendación muy práctica es el que nos pongamos en el lugar del computador, y analizar que necesito que me ordenen y en secuencia, para poder producir los resultados esperados.

- **Salida de resultados** Presentar los resultados requeridos, punto final del diagrama de flujo.

A continuación se citan algunos puntos para su elaboración:

1. Especificar el objetivo del flujo grama o diagrama de flujo.
2. Describir los procesos (operaciones) para alcanzar su solución.
3. Utilizar símbolos normados o estandarizados, que representen esos procesos.
4. Graficar el diagrama de flujo en forma ordenada y secuencial y presentar sus resultados.
5. Debe realizarse de forma limpia y ordenada.

4.6 Prueba de Escritorio

Para verificar que un diagrama de flujo esta bien y para garantizar que el programa que codificamos luego también funcione correctamente, es conveniente someterlo a una prueba de escritorio. Esta prueba consiste en que damos diferentes datos de entrada al programa y seguimos la secuencia indicada en el diagrama, hasta obtener los resultados. El análisis de estos nos indicara si el diagrama esta correcto o si hay necesidad de hacer ajustes.

5. CONCLUSION

Para poder realizar el código en el lenguaje de programación elegido, se recomienda elegir el camino a seguir de acuerdo al proceso representado en el esquema siguiente.

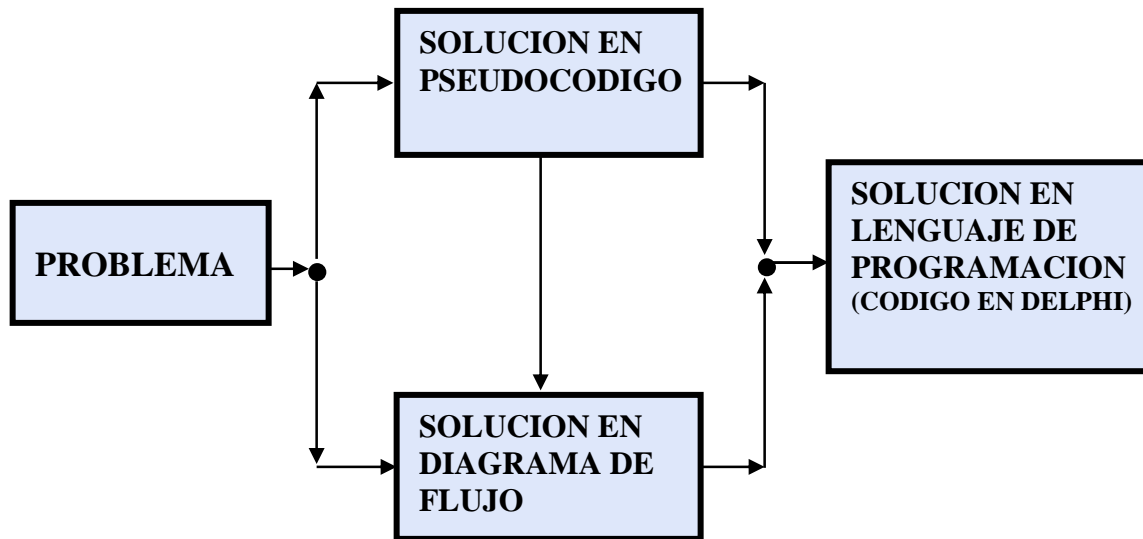


Fig. 5.1

Esquema De Programación (Fuente elaboración propia)

Es aconsejable seguir la ruta de la (Fig. 5.1):

En caso de realizar programa completo.

PROBLEMA - PSEUDOCODIGO - DIAGRAMA DE FLUJO - CODIGO

Tambien:

PROBLEMA - PSEUDOCODIGO - CODIGO

En caso de realizar diagramas de flujo:

PROBLEMA - PSEUDOCODIGO - DIAGRAMA DE FLUJO

6. BIBLIOGRAFIA

- Introduction **TO** Algorithms (2nd ed.), Cormen, T. H., Leiserson, C. E., Rivest, R. L. y Stein, C.
- Eufren Llanque DIAGRAMAS DE FLUJO, Ediciones UMSA La Paz -Bolivia

Paginas de internet

- <http://es.wikipedia.org/wiki/Algoritmo>
- <http://www.algoritmica.com.ar>
- <http://www.algoritmica.com.ar/apu/est/main.html>
- <http://lenguajes-de-programacion.com/programacion-de-algoritmos.shtml>
- http://es.wikipedia.org/wiki/Diagramas_de_flujo
- <http://www.monografias.com/trabajos14/flujograma/flujograma.shtml>
- <http://www.hci.com.au/hcsite2/toolkit/flowchar.htm>
- <http://www.iso.org/iso/en/aboutiso/introduction/index.html>

CAPITULO 3

INTRODUCCION A LA PROGRAMACION CON DELPHI

1. EVOLUCION DEL DELPHI

1.1 Lenguaje Pascal

Pascal es un lenguaje de programación desarrollado por el profesor suizo Niklaus Wirth a finales de los años 60. Su objetivo era crear un lenguaje que facilitara el aprendizaje de la programación a sus alumnos. Sin embargo con el tiempo su utilización excedió el ámbito académico para convertirse en una herramienta para la creación de aplicaciones de todo tipo.

Pascal se caracteriza por ser un lenguaje de programación estructurado fuertemente tipificado. Esto implica que:

- 1.- El código está dividido en porciones fácilmente legibles llamadas funciones o procedimientos. De esta forma Pascal facilita la utilización de la programación estructurada en oposición al antiguo estilo de programación monolítica (programación antigua).
- 2.- El tipo de dato de todas las variables debe ser declarado previamente para que su uso quede habilitado.

El nombre de Pascal fue escogido en honor al matemático Blaise Pascal.

Características únicas del lenguaje Pascal

Aunque no es una característica a diferencia de lenguajes de programación descendientes de C, Pascal utiliza el símbolo ($:=$) para la asignación, en vez de ($=$). Si

bien el segundo es más conciso, la práctica ha demostrado que muchos utilizan el símbolo de igualdad para comparar valores en lugar del comparador de C que es el símbolo (`==`). Pascal no permite dentro sus expresiones y utiliza una sintaxis distinta para asignaciones y comparaciones.

Otra diferencia importante es que en Pascal, el tipo de una variable se fija en su definición; la asignación a variables de valores de tipo incompatible no están autorizadas (En C, en cambio, el compilador hace el mejor esfuerzo para dar una interpretación a casi todo tipo de asignaciones). Esto previene errores comunes donde las variables son usadas incorrectamente porque el tipo es desconocido. Esto también evita la necesidad de notación húngara, estos son prefijos que se añaden a los nombres de las variables y que indican su tipo.

*Notación Húngara: sistema usado normalmente para crear los nombres de variables. Ejemplo: prefijo y significado → **b** booleano, **c** carácter (un byte), **e** enumeración.

1.2 Evolución del lenguaje OO Pascal

OO Pascal tiene todas las características de Turbo Pascal, el cual durante la época de los 80 fue uno de los lenguajes más populares en el desarrollo de aplicaciones para las computadoras personales debido a su facilidad de aprendizaje.

A principios de 1995, la empresa Borland International lanzaba Delphi como una opción mas en el desarrollo de aplicaciones visuales basadas en el entorno Windows, que hasta el momento dominaba Visual Basic de Microsoft y al cual no tardó en quitarle la hegemonía.

La característica principal que los desarrolladores advirtieron en Delphi, y que Visual Basic no ofrecía, era que generaba un verdadero código ejecutable ya que se trataba de un compilador y, por lo tanto las aplicaciones funcionaban mas rápido. Además que con su biblioteca de controles cubría todo lo necesario para la programación en Windows 3.1 y crear controles propios para agregarlos a la librería.

El Object Oriented Pascal es el lenguaje que Delphi utiliza para crear las aplicaciones orientadas a objetos que hoy en día es el paradigma mas utilizado. Debido a que Delphi pertenece a la empresa Borland, la potencia de su compilador puede compararse con el compilador de C++.

Actualmente existen varias versiones fundamentales de Delphi: Delphi 1 para Windows 3.1 y Windows 3.11, Delphi 2 junto a Delphi 3, para Windows 95-Windows NT. En la aparición de la versión 4, solo se agregó el soporte de la arquitectura Cliente/Servidor. Hasta la posterior aparición de las nuevas versiones de Delphi 5,6,7 a Delphi 8.

2. METODOLOGIAS DE PROGRAMACION

El desarrollo de un programa que resuelva un problema dado es una tarea compleja, ya que es necesario tener en cuenta de manera simultánea muchos elementos. Por lo tanto, es indispensable usar una metodología de programación.

Una metodología de programación es un conjunto o sistema de métodos, principios y reglas que permiten enfrentar de manera sistemática el desarrollo de un programa.

Estas metodologías generalmente se estructuran como una secuencia de pasos que parten de la definición del problema y culminan con un programa que lo resuelve figura 1.

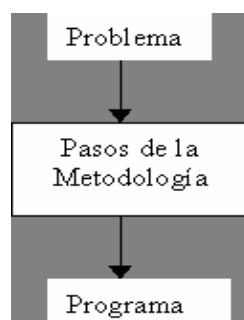


Figura 1. Secuencia de pasos (Propia)

Los lenguajes de programación pueden ser muchos y también pueden ser tomados de diferentes perspectivas. Un programador debe saber que conceptos utilizar u omitir al momento de programar. Por lo tanto es importante tener claro las características o estilos de programación para lograr determinar cual es la herramienta o lenguaje ideal según las características del sistema a implementar.

Algunas de estas metodologías son la:

- 1.- Programación estructurada y las estructuras básicas
- 2.- Programación orientada a objetos
- 3.- Programación orientada a eventos
- 4.- Programación funcional
- 5.- Programación lógica, etc.

2.1 Metodología de programación Estructurada

La Programación Imperativa está basada en el modelo de Von Neumann, donde un conjunto de operaciones primitivas realizan una ejecución secuencial. Realiza una abstracción en el manejo de variables, expresiones e instrucciones y para programar es necesario declarar las variables necesarias y diseñar una secuencia adecuada de instrucciones (asignaciones).

Ejemplo: Intercambio de valores en Pascal

PROGRAM intercambio;

VAR x,y, tmp;

BEGIN

 tmp := x;

 x := y;

 y := tmp;

END.

Algunos de los lenguajes de este tipo son Pascal, Ada y C.

2.2 Metodología de programación orientada al objeto

La programación orientada al objeto esta basada en los objetos, clase, método, envío y recepción de mensajes, herencia y polimorfismo. En donde:

Objetos: Es cualquier cosa que se ofrece a la vista y afecta a los sentidos. También se define como una entidad tangible que exhibe algún comportamiento bien definido. En términos de programación, un objeto no necesariamente es algo tangible (por ejemplo: un proceso). Lo que sí puede decirse de todo objeto es que tiene estado, comportamiento e identidad.

Clases: Una clase es una colección de datos y métodos que operan sobre los datos y sirven para definir el contenido y capacidades de algunos objetos.

Encapsulación: Modularidad y ocultamiento de información de un objeto.

Polimorfismo: es la cualidad que poseen los objetos para responder de distinto modo ante el mismo mensaje.

Herencia: la herencia permite definir nuevas clases y comportamientos basados en clases. Algunos de los lenguajes de este tipo son C++, Java y Smalltalk.

Desarrollo de la programación orientada a objetos figura 2:

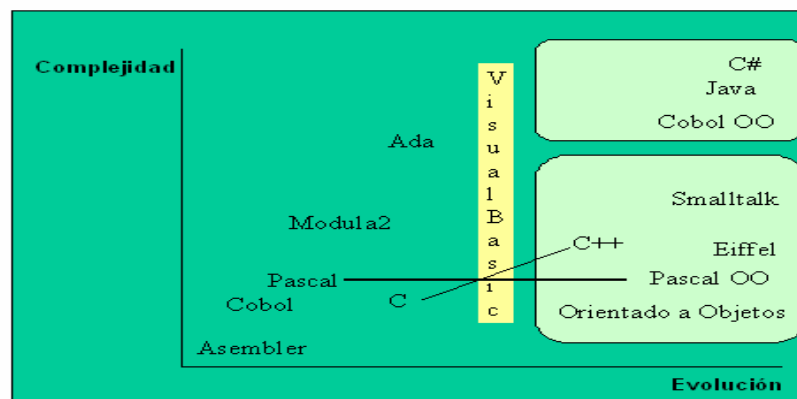


Figura 2.Desarrollo de la Programación
(http://www.delphi3000.com/articles/article_3228.aspHistoria de la Computación.htm)

2.3 Metodología de programación orientada al evento

Esta metodología de programación es el resultado de la programación orientada al objeto. Este tipo de programación permite trabajar con objetos y clases estándar previamente definidas por la aplicación, las cuales manejan los conceptos de encapsulación. Las herramientas que trabajan de esta forma por lo general trabajan con código original de lenguajes imperativos. Algunas herramientas de este tipo son Visual Basic (Basic), Delphi (Pascal) y Power Builder (C).

2.4 Metodología de programación funcional

El desarrollo de un programa es una función (o un grupo de funciones) usualmente compuesto de funciones más simples.

La relación entre las funciones son muy simples:

Una función puede llamar a otra función, o el resultado de una función puede ser usado como el argumento de otra función. Las variables, comandos y efectos laterales son exclusivos. Los programas son escritos enteramente dentro del lenguaje de expresiones, funciones y declaraciones.

Tres de estos lenguajes son Écheme, ML y Haskell.

2.5 Metodología de programación lógica

La metodología de programación lógica está basada en la noción de relación, debido a que la relación es un concepto más general de una aplicación. La programación lógica es potencialmente de alto nivel, en donde:

Relación: Considera 2 conjuntos de valor S y T, R es la relación entre S y T: Para todo X que pertenece a S y Y que pertenece a T y $R(X,Y)$ es verdadero o falso.

Los lenguajes de programación lógica pueden explotar la inteligencia artificial. Un lenguaje de este tipo es Prolog.

En Object Pascal no es posible definir o declarar un objeto en cualquier parte del programa; estas definiciones se agrupan en la parte reservada a las declaraciones.

Esta propiedad se traduce en una regla que obliga a que los objetos nuevos han de ser contruidos únicamente con la ayuda de objetos ya conocidos o predefinidos. Así por ejemplo, el programa:

```
PROGRAM Novalido;
```

```
VAR
```

```
    lista_edades : [ ARRAY 1..edadmaxima OF ] integer;
```

```
CONST
```

```
    edadmaxima = 99;
```

```
BEGIN
```

```
    ...
```

```
END.
```

* Es incorrecto, ya que se referencia a la constante edadmaxima en la declaración de variables antes de ser definida en la declaración de constantes.

3.1 La cláusula PROGRAM

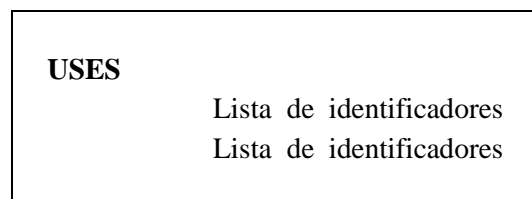
La cláusula **PROGRAM** especifica el nombre del programa (identificador) y sus parámetros. Este nombre no necesita ser el mismo que el de su código fuente ni del código maquina que produce el compilador.

Ejemplo:

PROGRAM DemoPrimero;

3.2 Parte de declaración de uso de unidades

Object Pascal difiere de ISO Pascal estándar en que permite la compilación separada. Si desea utilizar módulos compilados independientes de su programa, debe tener la parte de declaración de uso de unidades: la cláusula **USES** cuadro 2.



Cuadro 2. (Propia)

Unidades estándar de Objet Pascal y unidades creadas por el usuario; por ejemplo: Dialogs, Sysutils, Demo, etc.

Ejemplo:

USES Dialogs, Sysutils, Demo; *{Dialogs, Sysutils: unidades estándar; Demo: unidad creada por el usuario}*

* La cláusula **USES** identifica todas las unidades utilizadas por el programa, y significa que a partir de esa línea se podrán utilizar todas las rutinas (subprogramas) incluidas en dichas unidades.

La mayoría de sus programas tendrán una cláusula **USES**. Es opcional, pero si existe, debe ser la primera instrucción no comentario a continuación de la cláusula **PROGRAM**.

3.3 Sección de declaraciones

Las declaraciones **LABEL**, **CONST**, **TYPE** y **VAR** pueden aparecer mas de una vez, y en cualquier orden, en la parte reservada a las declaraciones, con la sola condición de que ellas respeten la regla enunciada anteriormente. Esta flexibilidad permite a veces una presentación aun mas clara, reagrupando declaraciones que tratan objetos lógicamente relacionados.

Otra particularidad es que las declaraciones de constantes, tipos y variables se pueden situar entre procedimientos.

4. HERRAMIENTAS DE DESARROLLO

La manera en que se desarrollan las aplicaciones cambia constantemente y la dinámica de este cambio se ha acelerado considerablemente durante los últimos años.

Hoy en la actualidad se tienen herramientas de desarrollo, las cuales están compuestas principalmente por:

- Lenguaje de programación.
- El Entorno de desarrollo.
- La Biblioteca de componentes.

4.1 Lenguaje de Programación

El lenguaje de programación es la piedra fundamental de todas las herramientas de desarrollo, o de casi todas. La orientación a objetos es una característica necesaria para calificar de actual a un lenguaje de programación.

Hay diferencias importantes entre los lenguajes de programación actuales, para el desarrollador de aplicaciones (por ejemplo aplicaciones cliente/servidor) estas son diferencias menores. Sin importar la herramienta de desarrollo que se utilice, la mayor parte del código estará dedicado a declarar variables y clases, crear objetos, ejecutar métodos de esos objetos y utilizar instrucciones de control tales como **IF**, **WHILE**, **FOR**, **CASE**, **SWITCH** y otras. Todas estas operaciones son soportadas por la mayoría de los lenguajes de programación actuales por lo que la principal dificultad al cambiar de lenguaje de programación reside en aprender la sintaxis del nuevo lenguaje.

4.2 Entorno de Desarrollo (IDE)

Un entorno de desarrollo integrado o en inglés Integrated Development Environment (**IDE**) Figura 3. Es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste de un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI (Graphic User **INTERFACE**). Los IDEs pueden ser aplicaciones por si solas o pueden ser parte de aplicaciones existentes.

El entorno de desarrollo es el segundo componente de una herramienta de desarrollo. Hace unos años todos los entornos de desarrollo se parecían mucho entre si. En todos ellos se podrá encontrar:

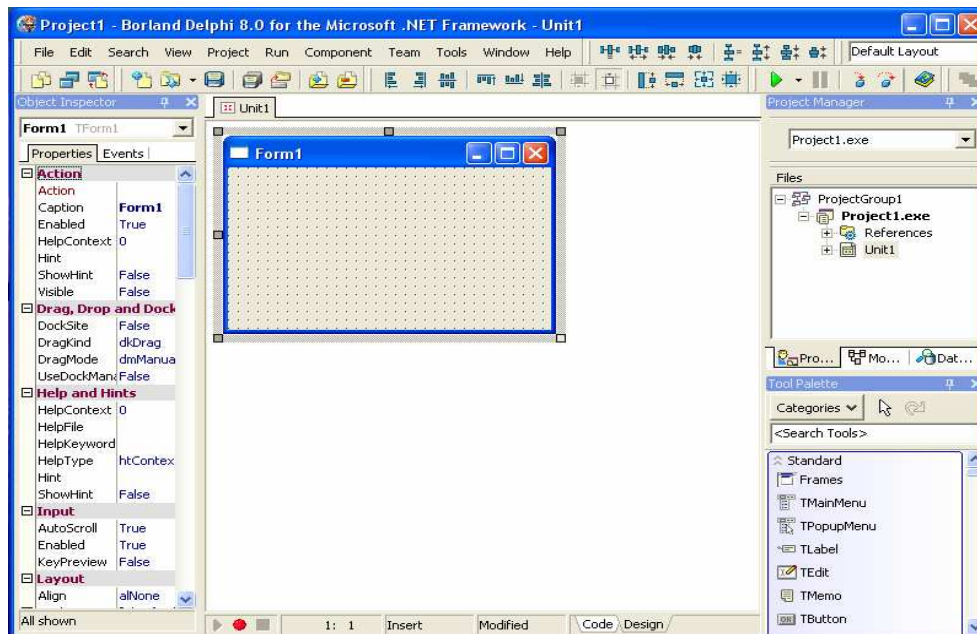


Figura 3. Entorno de desarrollo (IDE)

- **Un editor de código:** Contiene todo un arsenal de herramientas para facilitar y acelerar la escritura de código fuente (figura 4).

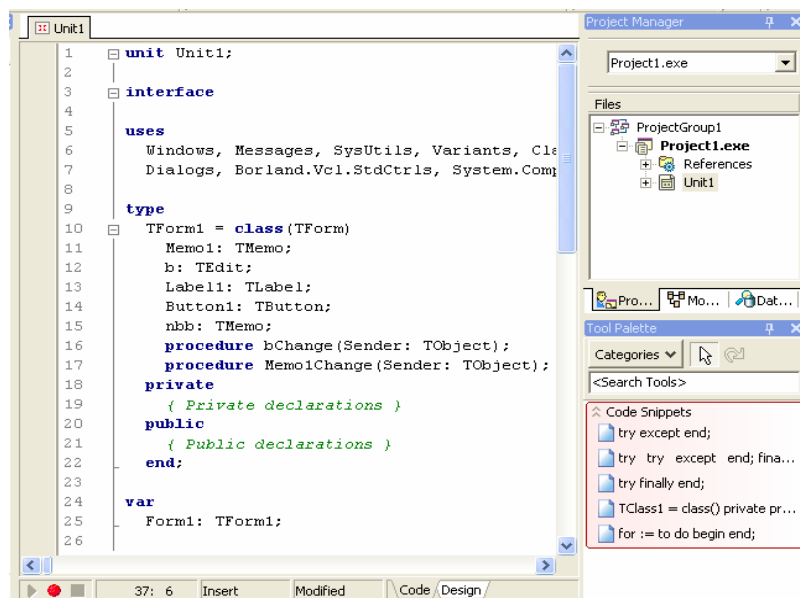


Figura 4. Editor de código (Delphi)

- **Un diseñador visual:** Permite ver en tiempo de diseño como se verán las cosas en tiempo de ejecución figura 5.

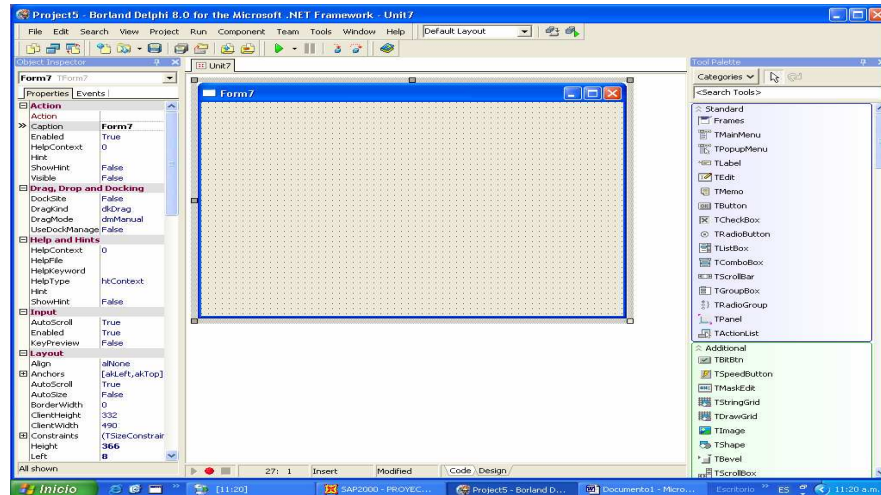


Figura 5. Diseñador visual (Delphi)

- **Una paleta de componentes:** Es desde la cual se selecciona los componentes que se colocan en el diseñador visual figura 6.

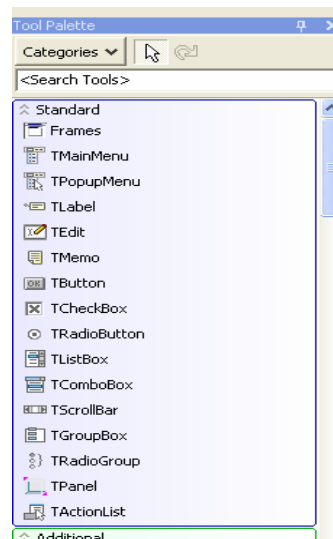


Figura 6. Paleta de componentes (Delphi)

- **Un editor de propiedades:** se utiliza para acceder a las propiedades y eventos de los componentes con los que se trabaja en el diseñador visual figura 7.

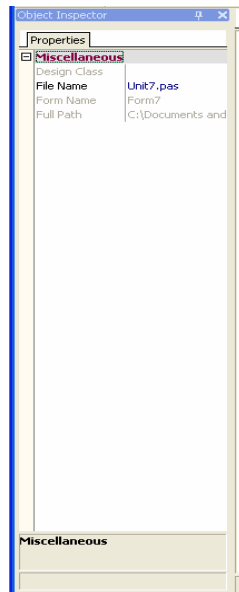


Figura 7. Editor de propiedades (Delphi)

- **Un administrador de proyectos:** Es mediante el cual se puede acceder a todos los archivos de los proyectos y administrarlos como una unidad figura 8.

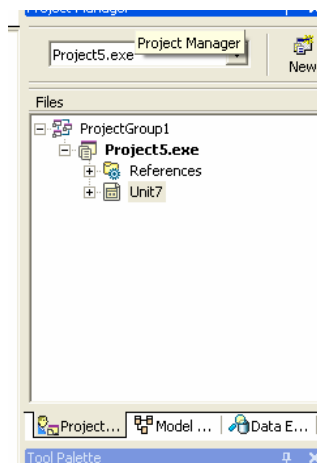


Figura 8. Administrador de proyectos (Delphi)

- **Un depurador integrado:** Se utiliza para seguir la ejecución del código fuente paso a paso (tecla F8).
- **Asistentes:** Son de todo tipo, color, que permiten dar los primeros pasos en el desarrollo de distintos tipos de aplicaciones y/o módulos.
- **Ayuda integrada:** Es mediante la cual, con solo presionar una tecla, se puede acceder a la documentación en línea del lenguaje de programación, la herramienta de desarrollo y la biblioteca de componentes.
- **Compilador:** Si bien forma parte del entorno de desarrollo, casi nunca se compila una aplicación desde la línea de comandos si no mas bien desde el entorno de desarrollo.
- **Integración:** Con herramientas de control de versiones y desarrollo en equipo.

Los entornos de desarrollo actuales permiten el desarrollo rápido de aplicaciones en un entorno visual basado en componentes. Las similitudes entre los entornos de desarrollo de las distintas herramientas de desarrollo hacen que ir de un entorno de desarrollo a otro no sea muy dificultoso.

Algunos ejemplos de entornos integrados de desarrollo (IDE) son los siguientes:

- Eclipse
- NetBeans
- IntelliJ IDEA
- JBuilder de Borland
- JDeveloper de Oracle
- KDevelop
- Anjuta
- Clarion
- MS Visual Studio .NET de Microsoft
- **Delphi de Borland**
- Visual Basic
- RadRails para Ruby on Rails
- CodeBlocks

4.3 Biblioteca de Componentes

La biblioteca de componentes lo es prácticamente todo y ciertamente es la parte más difícil de aprender a utilizar. Todas las herramientas de desarrollo proveen una biblioteca de componentes considerando que estas bibliotecas suelen ser mucho mas que una simple biblioteca de componentes.

Además de todos los componentes visuales y no visuales, la biblioteca de componentes esta compuesta por una jerarquía de clases que constituyen la base sobre la cual se construyen las aplicaciones.

Para mayor referencia a los componentes utilizados en el texto de ejercicios ver anexo A6 Lista de Componentes

5. DISEÑO DE (LOS) FORMULARIOS (S) DE APLICACIONES

Delphi es un entorno de programación visual orientado a objetos para desarrollo rápido de aplicaciones (RAD) de propósito general, incluyendo aplicaciones cliente/servidor, desarrollo de bases de datos, auténtica capacidad de reutilización de código orientada a objetos y compilador de código original de alto rendimiento.

Un formulario es la interfaz que presentan las aplicaciones de Windows. El formulario, es una caja de color gris que tiene una barra de color azul en la parte superior. Todos los lenguajes de programación que sean para Windows, o estén orientados a objetos, crean de modo automático la ventana y es ese el punto de partida de toda aplicación.

5.1 Características de un Formulario

Un formulario, no es más que un objeto más dentro de la programación, por lo tanto, cuenta también con métodos, propiedades y eventos. Los formularios poseen muchas propiedades, algunas de las más importantes son:

- **ActiveControl:** En el cual se pone el nombre del objeto que tendrá el enfoque cuando se ejecute la aplicación, de no poner nada, tendrá el enfoque el objeto que se insertó primero.

- **AutoScroll:** Si tiene el valor True, el formulario presentará barras de desplazamiento que permitan acceder a todas las partes del formulario.
- **AutoSize:** Si tiene el valor True, el ancho y alto del formulario se ajustará automáticamente para que ocupe el menor espacio posible conteniendo a todos los componentes.
- **BorderIcons:** Propiedad compuesta que determina si los botones de maximizar, minimizar, salir estarán visibles en el formulario.
- **BorderStyle:** Determina como será el borde del formulario.
- **BorderWidth:** Contiene el tamaño en píxeles del borde de la ventana.
- **Caption:** Es un texto que aparece en la parte superior del formulario.
- **Color:** Color de fondo del formulario.
- **Name:** Nombre del formulario en código.
- **Position:** Es la posición que ocupará en la pantalla el formulario.

5.2 Métodos de un Formulario

Los métodos, son las acciones que se pueden realizar sobre un formulario figura 9, entre las más comunes se encuentran las siguientes:

- **Close:** Invocando a este procedimiento, el formulario se cierra.
- **Show:** Este procedimiento hace que el formulario aparezca en pantalla.
- **ShowModal:** Esta función hace aparecer el formulario, y le da el control de la aplicación, para que el usuario pueda acceder a sus botones, cuadros de texto. Mientras el usuario no cierre este formulario, no puede seguir con la aplicación.
- **Hide:** Oculta el formulario, es el procedimiento contrario a Show.

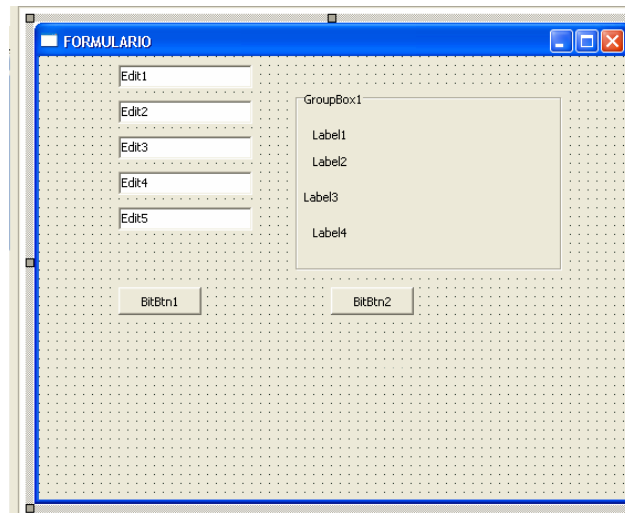


Figura 9. Formulario (Delphi)

6. ORGANIZACION DEL PROGRAMA

Los programas de Delphi están usualmente divididos en módulos de código fuente llamadas unidades. La mayoría de programas comienzan con un encabezado utilizando la palabra reservada **PROGRAM**, lo cual especifica un nombre para el programa. El encabezado de **PROGRAM** es seguido por una cláusula optativa **USES**, luego un bloque de declaraciones e instrucciones. La cláusula **USES** lista unidades que se enlazaron en el programa; estas unidades, las cuales pueden ser compartidas por diferentes programas. La cláusula **USES** provee al compilador de información acerca de las dependencias entre los módulos. Porque esta información es almacenada en los módulos mismos, la mayoría de programas en lenguaje OO Pascal (IDE Delphi) no requieren makefiles, archivos del encabezado, o preprocesador incluido directivas.

6.1 Archivos Fuente de Delphi

El compilador espera encontrar código fuente Delphi en tres tipos de archivos:

- Archivos Fuente de Unidades (los cuales tienen la extensión **.pas**)
- Archivos de Proyecto (los cuales tienen la extensión **.dpr**)
- Archivos Fuente de Paquetes (los cuales tienen la extensión **.dpk**)

Los archivos fuente de unidades típicamente contienen más del código en una aplicación. Cada aplicación tiene un solo archivo de proyecto y varios archivos de unidades. El archivo de proyecto, el cual corresponde al archivo de programa en pascal tradicional, organiza los archivos de unidades en una aplicación. Las herramientas de desarrollo Borland automáticamente mantienen un archivo de proyecto para cada aplicación. Si se compila un programa por la línea de comando, entonces se puede poner todo el código fuente en archivos de unidad (.pas). Si se usa el IDE para construir su aplicación, entonces producirá un archivo de proyecto (.dpr). Los archivos fuente de paquetes son similares a archivos de proyectos, pero se usan para construir bibliotecas especiales dinámicamente vinculables llamados paquetes.

6.2 Otros Archivos Usados para Construir Aplicaciones

En adición para los módulos de código de fuente, los productos Borland usan varios archivos non-Pascal para construir aplicaciones. Estos archivos son mantenidos automáticamente por el IDE, e incluyen:

- Archivos VCL form (los cuales tienen extensión **.dfm** en Win32, y **.nfm** en .NET)
- Archivos de Recursos (tienen extensión **.res**)
- Archivos de Opciones de Proyecto (tienen extensión **.dof**)

Un archivo VCL form contiene la descripción de las propiedades del formulario y los componentes que posee. Cada archivo form representa un solo formulario, el cual usualmente corresponde a una ventana o ventana de diálogo en una aplicación. El IDE permite ver y editar archivos form como texto, y para guardar archivos form como texto (un formato muy adecuado para el control de versión) o binario. Aunque el

comportamiento predeterminado es para guardar archivos form como texto, no son usualmente editados manualmente, es más común usar las herramientas visuales del diseño de Borland para este propósito. Cada proyecto tiene al menos un formulario, y cada formulario tiene una unidad asociada (.pas), por defecto, tiene el mismo nombre del archivo form.

Además de archivos VCL form, cada proyecto usa un archivo de recurso (.res) para mantener el icono de la aplicación y otros recursos como cadenas. Por defecto, este archivo tiene el mismo nombre del archivo de proyecto (.dpr). Un archivo de opciones de proyecto (.dof) contiene opciones del compilador y vinculador, información del camino de búsqueda, información de versión, y así sucesivamente. Cada proyecto tiene un archivo asociado de opciones de proyecto con el mismo nombre del archivo de proyecto (.dpr). Usualmente, las opciones en este archivo son fijadas en las opciones de proyecto. Las herramientas diversas en el IDE almacenan datos en archivos de otros tipos. Los archivos de configuracion desktop (.dsk) contienen información acerca de la distribución de ventanas y otras opciones de configuración; que pueden ser específicos en proyecto o ambiente. Estos archivos no tienen efecto directo en la compilación

6.3 Archivos Generados en la Compilación

La primera vez que se construye una aplicación o un paquete, el compilador produce un archivo de unidad compilado (.dcu en Win32, .dcul en .NET) para cada unidad nueva usada en el proyecto; Todos los archivos .dcu/ .dcul en el proyecto están vinculados para crear un solo ejecutable o paquete compartido. La primera vez que se construye un paquete, el compilador produce un archivo para cada unidad nueva contenida en el paquete, y luego crea un .dcp y un archivo del paquete. Cuando se construye un proyecto, las unidades individuales no son recompiladas a menos que los archivos fuente (.pas) han cambiado desde la última compilación, sus archivos .dcu/ .dpu no pueden ser encontrados, explícitamente se le dice al compilador que los reprocese, si la interfaz de la unidad depende de otra unidad que ha cambiado. De hecho, no es necesario que para el archivo fuente de una unidad estén presentes en todo, mientras el compilador pueda

encontrar el archivo compilado de la unidad y esa unidad no tiene dependencias en otras unidades que han cambiado.

Puede ser específico en proyecto o extenso en ambiente. Estos archivos no tienen efecto directo en la compilación.

6.4 Archivos Generados en un proyecto DELPHI

PROJECT1.DPR	Fichero del Proyecto
UNIT1.PAS	Ficheros Fuente
UNIT1.DFM	Definición de <i>Forms</i> y su contenido
UNIT1.DCU	Unidad compilada
PROJECT1.RES	Icono del proyecto
PROJECT1.DOF	Opciones de compilación y vinculación
PROJECT1.EXE (o DLL)	Ejecutable del proyecto

Opcionales

PROJECT1.DSK	Estado del Interfaz de Desarrollo
PROJECT1.DSM	Tabla de símbolos, tal y como se quedó
PROJECT1.MAP	Fichero de texto con detalles para depurar

Para tener información de instalación de Delphi 8 ver anexo A4.

Relación de Delphi con otros programas anexo A5.

7. BIBLIOGRAFIA

- Luis Joyanes Aguilar, Antonio Muñoz Clemente, BORLAND DELPHI Iniciacion y Referencia, McGraw-Hill/Interamericana de España S.A.U.
- Francisco Charte, PROGRAMACION CON DELPHI, Ediciones Anaya Multimedia S.A.
- Victor Moral, DELPHI 4, Prentice Hall Iberia S.R.L.
- Victor A. Melchor Espinoza DELPHI 8 Editorial MACROPERU.

Paginas de Internet:

- http://www.delphi3000.com/articles/article_3228.aspHistoria de la Computación.htm
- <http://www.swissdelphicenter.ch/en/showcode.php?id=1155>Historia de la computación4.htm
- <http://www.freepascal.org/sdown.html>
- <http://es.wikipedia.org/wiki/Delphi>
- <http://www.borland.com/delphi>

CAPITULO 4

CONSTANTES VARIABLES Y TIPOS DE DATOS

1. IDENTIFICADORES

Los identificadores son conjuntos de letras y/o números que se utilizan para simbolizar todos los elementos que se utilizan en un programa, son definidas por el usuario (programador o ingeniero de software).

En BORLAND DELPHI un identificador es una palabra compuesta de letras y/o números de hasta 32 caracteres significativos, empezando siempre con una letra.

2. CONSTANTES Y VARIABLES

Son contenedores de datos, se caracterizan por tener un identificador, contenido y un tipo. Se utilizan para almacenar datos generados durante la ejecución de un programa.

Cuando se realiza un programa, constantemente se tiene que almacenar valores dentro del código, valores que pueden ser resultado de una operación. Las variables y las constantes sirven para almacenar valores. Una variable puede cambiar su valor en cualquier momento del programa, una constante siempre mantendrá el mismo valor durante el programa. Las constantes y variables deben tener un nombre, como por ejemplo:

nombre, edad, sdodiario, ingmensual, perímetro, calif1, etc.

Delphi obliga a declarar explícitamente cada una de las variables que se vaya a usar en la aplicación, además hay que declararlas según el tipo de dato que vaya a contener.

La forma de usar una variable o constante es a través de su identificador (nombre).

Para crear una variable o una constante se debe tener en cuenta que:

- El identificador no tiene que haber sido utilizado para identificar otra variable.
- El identificador no debe ser palabras reservadas del lenguaje.
- El identificador no puede empezar con un número, ni tener espacios en blanco.
- El identificador no puede contener caracteres “raros”, como comillas, acentos, símbolos de puntuación ni la letra ñ. Sin embargo puede tener letras, números (siempre que no sea al principio) y el símbolo “_”.
- Debe tener un tipo de valores a contener.

2.1 Declaración De Variables

Las variables se declaran de modo distinto a como lo hacen las constantes. Antes de utilizarla hay que declarar qué es lo que contendrá (tipo), para que reserve el espacio en memoria para ella. Antes de empezar a definir variables se debe de poner la palabra **VAR** y deberá ser antes de la palabra de inicio (**BEGIN**), abajo el nombre de la variable, seguido de 2 puntos y el tipo de valor que contendrá.

Ejemplo.-

```
VAR  
  NomVar:Tipo;
```

```
VAR  
  edad : byte;  
  nombre : string;  
  sis : cardinal;  
  curso : byte;  
  puntuación : real;  
  varon : boolean;
```

Cada variable va acorde con lo que almacenará, pues edad tiene un tipo byte, y no almacenará un valor grande, varon es de tipo boolean para que TRUE indique “SI” y FALSE indique “NO”.

Para declarar más de una variable de un mismo tipo, se hará:

```
VAR  
  NomVar1, NomVar2:Tipo;
```

```
VAR  
  a,b,c : Integer;
```

Es decir, separando las variables con una coma.

2.2 Asignación

Una vez definida la variable, hay que darle un valor. Esto se hace siguiendo la sintaxis siguiente:

NomVar := Expresión;

Esto es asignación, únicamente se debe anteponer el nombre de la variable, seguido de 2 puntos y un signo igual y finalmente el valor que almacenará.

Ejemplo:

Edad := 26;

Nombre := 'Gerson Rojas';

También es posible realizar operaciones aritméticas en una asignación como por

Ejemplo:

A := 1+2;

Res := A+2;

2.3 Declaración de Constantes

Las constantes se declaran del siguiente modo: en primer lugar se pone la palabra reservada **CONST** seguida de la constante, y el valor de la constante (sin olvidar el punto y coma reglamentario).

CONST NomCte = Expresión;

Donde la expresión debe ser conocida en tiempo de compilación.

Por ejemplo:

CONST pi = 3.141592;

CONST B = 18;

CONST anio = 2006;

3. TIPOS DE DATOS

A toda variable que se use en un programa, se le debe asociar (generalmente al principio del programa) un tipo de dato específico. Un tipo de dato define todo el posible rango de valores que una variable puede tomar al momento de ejecución del programa y determina las operaciones posibles que se pueden realizar con ellos.

Los tipos de datos más comunes en BORLAND DELPHI son:

Tipo	Rango de valores	Detalle
Byte	0 a 255	Enteros de 8 bits sin signo.
ShortInt	-128 a 127	Enteros de 8 bits, con signo
SmallInt	-32768 a 32767	Enteros de 16 bits, con signo
Word	0 a 65535	Enteros de 16 bits, sin signo
Integer	Depende del S.O. de -32767 a 32767 de -2147483648 a 2147483647	Enteros con signo, En sistemas de 16 bits En sistemas de 32 bits
LongInt	de -2147483648 a 2147483647	Enteros de 32 bits, con signo
Cardinal	Depende del S.O. 0 a 32767 0 a 2147483647	Enteros sin signo, En sistemas de 16 bits En sistemas de 32 bits
Boolean	True o False	Verdadero o Falso
ByteBool	True o False	Ocupa un byte (igual que Boolean)
WordBool	True o False	Ocupa 2 bytes
LongBool	True o False	Ocupa 4 bytes
Real	2.9×10^{-39} a 1.7×10^{38}	Coma flotante, 11-12 dígitos significativos Este tipo existe por compatibilidad con versiones anteriores, no recomendado ya que es más lento.
Single	1.5×10^{-45} a 3.4×10^{38}	Coma flotante, 7-8 dígitos significativos
Double	5.0×10^{-324} a 1.7×10^{308}	Coma flotante, 15-16 dígitos significativos
Extended	-3.6×10^{4951} .. 1.1×10^{4932}	Coma flotante, 19-20 dígitos significativos
Comp	$-2^{63}+1$.. $2^{63}-1$	Enteros de 64 bits con signo, 19-20 dígitos significativos
Currency	-922337203685477.5808 a 922337203685477.5807	Enteros con 4 decimales, 19-20 dígitos significativos
Char	Un caracter de 1 byte	Un caracter de 1 byte (Ansi)
AnsiChar	Un caracter de 1 byte	Un caracter de 1 byte (Ansi)
WideChar	Un caracter de 2 bytes	Un caracter de 2 bytes (Unicode)
String	Depende del S.O. 255 caracteres ilimitado	Cadena de caracteres En sistemas de 16 bits En sistemas de 32 bits
ShortString	1 a 255 caracteres	Cadena de caracteres, de un máximo 255 caracteres
AnsiString	2^{31} caracteres	Cadena de caracteres
PChar		Puntero a una cadena de caracteres ASCIIZ (terminada en Null)
PAnsiChar		Puntero a una cadena de caracteres ASCIIZ
PWideChar		Puntero a cadena Unicode
Pointer		Puntero
Variant		Admite cualquier tipo de datos

4. OPERADORES

4.1 Operadores Aritméticos

Un operador es un símbolo especial que indica al compilador que debe efectuar una operación matemática o lógica. DELPHI reconoce los siguientes operadores aritméticos:

Operador	Operación	Ejemplo
+	SUMA	$A + B$
-	RESTA	$C - D$
*	MULTIPLICACION	$A * D$
/	DIVISION REAL	$T / 4$
Div	DIVISION ENTERA	$D \text{ div } 4$
Mod	MODULO O RESIDUO	$B \text{ mod } 5$

4.2 Operadores Lógicos

En Pascal se llaman expresiones booleanas a las expresiones que indican una condición y que producen un resultado de verdadero o falso.

Una expresión booleana se puede colocar de dos modos distintos:

1. Una variable o constante seguida de un operador de comparación y seguida de otra variable.
2. Una expresión booleana seguida de un operador booleano y de otra operación booleana.

Los operadores que se utilizan en Delphi son:


Operador	Descripción	Ejemplo
>	Mayor que	$A > b$
<	Menor que	$B < 4$
>=	Mayor o igual que	$A >= 3$
<=	Menor o igual que	$C <= 7$
<>	Distinto	$A <> 0$
=	Igual	$A = 4$

Las operaciones booleanas que sirven para unir expresiones y evaluarlas, mencionadas anteriormente son las que a continuación se presentan.

Operación	Descripción	Ejemplo
AND	Y Inclusión	(a > b) AND (c < 3)
OR	O Exclusión	(a > b) OR (c < 3)
NOT	No Negación	NOT (a = b)

Un ejemplo de una estructura de decisión con una expresión booleana es la siguiente:

IF ((a > b) AND (c < 3)) OR NOT (a > 7) THEN c:= 0;



Condición a ser Evaluada

4.3 Operadores De Cadenas

Las cadenas de tipo string, utilizan operadores especiales para trabajar con ellas:

Unión.- Podemos unir dos cadenas en una sola utilizando el signo +, por ejemplo:

```

nombre := 'Gerson';
apellido := 'Osornio';
nombrecompleto := nombre + ' ' + apellido;
  
```

4.4 Operadores De Conjuntos (SET)

Los siguientes operadores toman conjuntos como operandos.

Operador	Operación	Tipo de Operandos	Tipo de Resultado	Ejemplo
+	Unión	SET	SET	Set1 + Set2
-	Diferencia	SET	SET	S - T
*	Intersección	SET	SET	S * T
<=	Sub-conjunto	SET	Boolean	Q <= MiSet
>=	Super-conjunto	SET	Boolean	S1 >= S2
=	Igualdad	SET	Boolean	S2 = MiSet
<>	Distinto	SET	Boolean	MiSet <> S1
in	Pertenencia	SET	Boolean	A in Set1

5. FUNCIONES MATEMATICAS

Delphi tiene funciones que permiten cambiar tipos de datos, para poder utilizarlas debemos escribir el nombre de la unidad **Sysutils**, debajo de la instrucción **Uses**. A continuación se mencionan las funciones mas utilizadas.

TRUNC.- Esta función permite asignar un número real a una variable de tipo byte, word o integer. Recibe como parámetro un número real y retorna dicho número sin parte decimal. No hace redondeo, solo elimina la parte que está después del punto decimal.

Ejemplo:

```
a : byte;
f : real;
f := 1.8934;
a := TRUNC (f);    // Resultado a = 1
```

STRTOINT.- Permite asignar un tipo string (cadena de caracteres) que contiene en su interior un número a una variable de tipo entero. Si la cadena no contiene un número, sino letras, se produce un error. Ejemplo:

```
a : string;
b : integer;
a := '123';
b := STRTOINT (a);    // Resultado b=123
```

INTTOSTR.- Es la contraria de la función anterior. Permite convertir un número entero a cadena. Ejemplo:

```
a : integer;
b : string;
a := 12;
b := 'el valor de a es ' + INTTOSTR (a);    // b= 'el valor de a es  12'
```

STRTOFLOAT.- Permite asignar un string que contiene en su interior un número a una variable de tipo real. Si la cadena no contiene un número, sino letras, se produce un error. Ejemplo:

```
a := '123.45';
b := STRTOINT (a);    // Resultado b=123.45
```

FLOATTOSTR.- Es la contraria de la función anterior. Permite convertir un número real a string. Ejemplo:

```
a := FLOATTOSTR (b);    // Resultado b='123.45'
```

6. COMENTARIOS

Es posible incluir en el código, texto aclarativo que ayude a entender lo que la instrucción esta realizando. A ese texto se le denomina comentario.

En Delphi hay 3 formas distintas de poner comentarios:

- Encerrando el texto aclarativo entre llaves ({ }).
- Encerrando el texto aclarativo entre paréntesis y asterisco (* *).
- Anteponiendo al texto dos símbolos de diagonal (//). Únicamente una línea de texto.

Ejemplo:

```
(*  
Todo esta parte es un comentario  
*)  
BEGIN {Esto también es un comentario}  
a := b + c; // Aquí se realiza una suma  
showmessage(a);  
END;
```

Los comentarios no son considerados por Delphi como parte del código. Su única función es la de aclarar el código.

7. OTROS TIPOS DE DATOS

7.1 Arreglos

Los arreglos son variables agrupadas bajo un mismo nombre, que permite tener distintos valores de un mismo tipo de dato.

Un arreglo con un solo índice se denomina unidimensional (vector), un arreglo con dos o más índices se llama multidimensional. El número de índices necesario para especificar un elemento de un arreglo se denomina dimensión.

Las matrices son arreglos bidimensionales, que se asemejan a una tabla o conjunto de filas (primer índice) y columnas (segundo índice).

Los elementos de una matriz se representan por dos expresiones separadas por un coma y encerradas entre corchetes.

Ejemplo.

$A[i,j]$ en Delphi

Se refieren al elemento de la fila “i” y la columna “j”

En matemáticas se suelen representar con la siguiente notación:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad 3 \times 3$$

Matriz de 3 filas por 3 columnas

7.1.1 Arreglos Unidimensionales (Vectores)

Un vector es un conjunto ordenado de elementos que contiene un número fijo de ellos. Los vectores son arreglos unidimensionales, sus elementos deben ser del mismo tipo de datos: enteros, reales, caracteres, cadenas, etc.

Ejemplo.

Vector: (2 , 3 , 5 , 1 , 20) : De 1 fila por 5 columnas

Y representa una lista de elementos.

La longitud de un vector es el número total de elementos.

Cada elemento de un vector se representa por el nombre del vector y un subíndice.

Ejemplo.- Si X es el nombre del vector, los elementos son:

$$X_1=2 \quad , \quad X_2=3 \quad , \quad X_3=5 \quad , \quad X_4=1 \quad , \quad X_5=20$$

7.1.2 Tipo de Arreglos Bidimensionales (Matrices)

Las matrices pueden ser tanto de caracteres (string) como numéricas o de otro tipo, para ello se tendrá que revisar el punto ‘3 Tipos de Datos’, para saber cual es el modelo que más se ajuste a los intereses deseados.

7.1.3 Representación Visual de arreglos

Para tener una idea mas clara de lo que son los arreglos, a continuación se presenta en forma visual los tipos de arreglos:

- **Arreglos unidimensionales (Vectores)**

Vector de Enteros (Integer)

Num: **ARRAY** [1..6] **OF** Integer

10	12	13	15	27	10
1	2	3	4	5	6

Vector de Caracteres (Char)

Let: **ARRAY**[1..8] **OF** Char

A	D	Z	X	F	T	R	S
1	2	3	4	5	6	7	8

Posición

Nombres: **ARRAY**[1..3] **OF** String

Luis	Carlos	Pedro
1	2	3

- **Arreglos bidimensionales (Matrices)**

Matriz de 3 filas x 4 columnas

10	12	1	3
20	14	7	9
15	65	5	11

3x4

- Matriz de números Enteros
- Numero total de elementos =12

7.1.4 Declaración de Arreglos

Suponer que se quiere grabar los nombres de diez amigos, si se lo hace por variables se tendría que tener 10 variables, una por cada amigo o sea:

nombre1, nombre2, nombre3..... y así hasta nombre10

Esto significa que se tendría que manejar diez variables, para buscar un nombre en concreto.

Con arreglos, esa misma declaración de nombres sería de la siguiente manera:

Ejemplo.

VAR

nombres: **ARRAY**[1..10] **OF** string

También se puede declarar de la siguiente forma.

Caso Unidimensional

TYPE

vec:=**ARRAY**[1..10]**OF** string;

VAR

nombres:vec;

Caso Bidimensional

TYPE

mat:=**ARRAY**[1..10,1..10]**OF** integer;

VAR

nombres:mat;

Con ésta simple declaración se tiene declarados diez nombres, pues supongan que en vez de 10 fueran 10000, si se lo hiciera con variables, se tendría que declarar 10000 variables, mientras que con los arreglos, se declararía con una sola línea, poniendo de 1 a 10000 elementos.

Ésta es la principal ventaja del uso de los arreglos, y su forma de declararla es como se ha indicado anteriormente.

Así mismo si se quiere ir a un elemento en concreto del arreglo, solo se tendrá que poner el nombre del arreglo y su índice y/o índices.

Ejemplo:

VAR

amigos: **ARRAY**[1..10] **OF** string

BEGIN

```
    amigos[1]:='Antonio';
    amigos[2]:='Luis';
    amigos[3]:='Pedro';
    amigos[4]:='Ronal';
    amigos[5]:='María';
```

Si se quiere guardar en la variable “nombre” el nombre de Ronal, solo se tendría que realizar lo siguiente::

```
    nombre := amigos[4];
```

Con esto ya se tiene puesto en la variable “Nombre” el nombre del amigo que se encuentra en la posición 4 del arreglo ‘amigos’.

Como se puede observar, es fácil tanto el manejo como la utilización de arreglos para almacenar distintos nombres, en una misma variable, para no tener que manejar muchas variables a la hora de trabajar con ellas.

7.2 Registros

Un registro representa un conjunto heterogéneo de elementos. Cada elemento es llamado campo, la declaración de un registro especifica un nombre y un tipo de datos para cada campo.

En algunas ocasiones, se necesitan construir variables que se compongan a su vez de otras variables. Por ejemplo, si se desea construir una variable de los datos de una persona y que esta a su vez este constituida de otras variables como nombre, teléfono, edad, etc.

Declaración de Registros

Los registros se declaran del siguiente modo:

VAR	TYPE
nombva: RECORD	nombvar: RECORD
subvariable: tipo;	subvariable: tipo;
subvariable: tipo;	subvariable: tipo;
END;	END;
	VAR
	nombreg:variable;

Donde nombvar es el nombre de la variable “contenedora”(llamada registro) y las subvariables, son los nombres de las variables contenidas, las que se denominan “campos del registro”. Ejemplo:

```
VAR
fichapersona : RECORD
    nombre: string;
    edad: byte;
    telefono: longint;
END;
```

De este modo se declara una variable de registro llamada Fichapersona que se compone de 3 variables: Nombre, Edad y Telefono. Para ingresar los datos a este registro, se deberán colocar los valores del siguiente modo.

```
Fichapersona.Nombre := 'Gerson Camacho';
Fichapersona.Edad := 26;
Fichapersona.Telefono := 77550164;
```

7.3 Conjuntos (SET)

Conjunto es una colección de valores del mismo tipo. Los valores no tienen un orden sistemático, ni tampoco un valor podrá ser incluido dos veces en un conjunto.

Como sucede en matemáticas, los conjuntos de Object Pascal solamente referencian una única vez a un elemento.

Ejemplo:

Notación matemática para descripción de los elementos de un conjunto:

Con llaves

$A=\{1,2,3,4,5,6\}$ $B=\{3,4,5\}$ $C=\{1,2\}$ $D=\{ \}$ (Conjunto Vacío)

Con diagramas de Venn Euler



7.3.1 Declaración de tipos de datos Conjunto

Se pueden declarar los tipos conjuntos de tres maneras diferentes:

- Como constantes literales de conjuntos. Para ello se enumeran entre corchetes los elementos que forman el conjunto.
- Como tipo de dato conjunto. En el apartado de declaración de tipos se declara un identificador y, después del signo igual (=) las palabras reservadas de la sintaxis **SET OF** seguidas del tipo ordinal que será la base de los elementos del conjunto.
- Como variables de tipo conjunto. Para lo cual después del identificador y de dos puntos (:) se situaran las palabras reservadas de la sintaxis **SET OF** seguidas del tipo ordinal que será la base de los elementos del conjunto.

Ejemplo:

TYPE

```
semana= [ L , M , R , J , V , S , D ]  
conjuntoEnteros = SET OF integer;  
conjuntoLetras = SET OF char;  
conjuntoDiasSemana = SET OF semana;
```

VAR

```
enteros: conjuntoenteros;  
letras: conjuntoletras;  
dias: conjuntodiassemana;
```

7.3.2 Relación de pertenencia en Conjuntos

La relación de pertenencia en un conjunto se consigue utilizando el operador **IN** (ver punto 3.4 operadores de conjuntos), que tienen como primer operando un elemento y como segundo operando un conjunto. El resultado de una expresión de pertenencia es true si el elemento es uno de los del conjunto y será false en caso contrario. La relación de no pertenencia se consigue por medio de la negación de la relación.

Ejemplo:

REPEAT

```
opc:= STRTOINT(inputbox('Clave','Introduzca un Numero',' '));  
UNTIL opc IN [ 1 0 0 , 2 0 , 5 0 ] ;  
showmessage('Clave Aceptada');
```

Para mayor información ver: Ayuda de Borland Delphi (Guía Delphi-Tipos de Datos Constantes y Variables-Datos Estructurados)

También consultar el libro: Borland Delphi de Luis Joyanes Aguilar/Antonio Muñoz Clemente, Capitulo 5 (Tipos de datos definidos por el usuario)

8. BIBLIOGRAFIA

- Luis Joyanes Aguilar, Antonio Muñoz Clemente, BORLAND DELPHI Iniciacion y Referencia, McGraw-Hill/Interamericana de España S.A.U.
- Francisco Charte, PROGRAMACION CON DELPHI, Ediciones Anaya
- Victor A. Melchor Espinoza DELPHI 8 Editorial MACROPERU.
- Ayuda del programa (BORLAND DELPHI HELP)

Paginas de Internet:

- <http://www.elguille.info/delphi/apuntesDelphi/apuntesDelphi.htm>
- <http://www.programacionfacil.com>
- <http://www.mailxmail.com/curso/informatica/delphi>

CAPITULO 5

INSTRUCCIONES DE CONTROL

1. INSTRUCCION IF...THEN...ELSE

Esta instrucción permite evaluar una condición y realizar distintas operaciones, si la condición resulta verdadera o falsa; es decir en caso que la condición resulte falsa se hará un proceso o procesos, si es verdadera se ejecutará otro proceso o varios.

Se sigue la siguiente sintaxis:

IF *condición* **THEN** *instrucción1*
ELSE *instrucción2*;

Después de la instrucción **IF** (“si”) debe ir la *condición* que ha de evaluarse. A continuación de ésta, se coloca la palabra **THEN** (“entonces”) seguida de la *instrucción1*, que se ejecutará si la *condición* es cierta o se cumple. En caso contrario, se ejecutará la *instrucción2*, que esta precedida por la palabra **ELSE** (“si no”). Ejemplo:

IF a = b **THEN** c := 1
ELSE c := 2;

En caso de que se deseen poner mas instrucciones en un solo bloque, se deberá realizar del siguiente modo:

IF a = b THEN BEGIN c := 1; d := 2; e := 3; END ELSE BEGIN c := 5; d := 3; e := 0;	}	Cuerpo: Conjunto de instrucciones
--	---	--

END;

La parte de la instrucción **ELSE** no es obligatoria, y puede ser omitida, con lo cual la instrucción quedará reducida, sin embargo, si la parte **ELSE** se omite, se deberá poner punto y coma al final de la instrucción. Ejemplo:

```
IF a = b THEN  
BEGIN  
    c := 1;  
    d := 2;  
    e := 3;  
END;
```

* En el ejemplo, si las variables **a** y **b** son iguales se darán valores a las variables **c**, **d** y **e**, en caso contrario no se hará nada.

2. ANIDAMIENTOS

A continuación de la palabra reservada **THEN** se pueden ejecutar secuencias de instrucciones las cuales pueden ser de cualquier tipo. Es posible colocar un **IF** dentro del bloque **THEN** de otro, a estas instrucciones de “**IF’s**” encadenados se le denomina anidamiento.

Es muy importante a tomar en cuenta que cuando se colocan anidamientos es recomendable colocar las instrucciones **BEGIN** y **END** entre **THEN** y **ELSE** de este modo se sabe exactamente a que **IF** pertenece cada **ELSE**.

Ejemplo:

```
IF a = 7 THEN  
BEGIN  
    IF b = 2 THEN  
        c := 3  
    ELSE  
        c := 2;  
    END  
ELSE  
    c := 1;
```

3. SELECCION MULTIPLE Y USO DE LA INSTRUCCION CASE...OF

Cuando se realiza un programa, es frecuente encontrarse con alguna variable que según su valor realizará alguna acción. Esto se podría realizar con muchos **IF's** anidados, pero resultaría algo dificultoso y ampuloso.

Ejemplo: Si se evalúa la variable **a**, cuando sea igual a 1, **c** tome el valor 10, cuando **a** tenga el valor 2, **c** tome el valor 15, cuando **a** tenga el valor 3, **c** tome el valor 20 y cuando no sea alguno de los 3 valores, entonces que **c** tome el valor 0:

```
IF a = 1
THEN c := 10 ELSE
IF a = 2
THEN c := 15 ELSE
IF a = 3
THEN c := 20 ELSE
c := 0;
```

Esta forma de tomar decisiones resulta muy poco ortodoxa. Por lo cual para dicho propósito existe otra forma más fácil de hacerlo, mediante la palabra reservada **CASE OF**. La sintaxis de dicha instrucción es la siguiente.

```
CASE variable OF
valor1: acción1;
valor2: acción2;
....
ELSE acción N;
END;
```

Donde **variable** es el identificador de la variable que será comprobada y **valor1**, **valor2...** son los diferentes valores que puede tomar dicha variable. Si se toma el ejemplo anteriormente planteado, tendremos que la solución sería de este modo:

```
CASE a OF
1: c := 10;
2: c := 15;
3: c := 20;
ELSE c := 0;
END;
```

4. BIBLIOGRAFIA

- Luis Joyanes Aguilar, Antonio Muñoz Clemente, BORLAND DELPHI Iniciacion y Referencia, McGraw-Hill/Interamericana de Espana S.A.U.
- Francisco Charte, PROGRAMACION CON DELPHI, Ediciones Anaya Multimedia S.A.
- Victor Moral, DELPHI 4, Prentice Hall Iberia S.R.L.
- Victor A. Melchor Espinoza DELPHI 8 Editorial MACROPERU.
- Ayuda del programa (BORLAND DELPHI HELP)

Paginas de Internet:

- <http://www.freepascal.org/sdown.html>
- <http://es.wikipedia.org/wiki/Delphi>
- <http://www.borland.com/delphi>
- <http://www.delphi 5.htm>
- <http://www.programacionfacil.com>
- <http://www.elguille.info/delphi/indice.htm>
- <http://www.marcocantu.com/>
- <http://www.marcocantu.com/epascal>
- <http://www.marcocantu.com/epascal/Spanish/default.htm>

CAPITULO 6

CICLOS ITERATIVOS

1. INTRODUCCION

En algunos programas, es necesario repetir la misma acción un número determinado de veces, hasta que se cumpla una acción determinada.

En Delphi existen 3 tipos distintos de bucles o ciclos.

2. INSTRUCCIÓN REPEAT UNTIL

Se utiliza para repetir las mismas acciones hasta que se cumpla una condición determinada.

REPEAT

 <Cuerpo>

UNTIL condición;

En el momento en que se cumpla la condición, terminará el bucle. Ejemplo:

 a := 1;

 j := 1;

REPEAT

 a := a * j;

 j := j+1;

UNTIL j = 10;

Esta instrucción tiene la particularidad de ejecutar el cuerpo por lo menos una vez.

3. INSTRUCCIÓN WHILE DO

Se utiliza para repetir las acciones mientras se cumpla una condición. Su sintaxis es:

WHILE condición **DO**

BEGIN

 <Cuerpo>

END;

En el momento que la condición deje de cumplirse, el bucle terminará y seguirá con la siguiente instrucción del código. Ejemplo:

```
a := 1;  
j := 1;  
WHILE (j <=10) DO  
BEGIN  
    a := a +j;  
    j := j +1;  
END;
```

La diferencia con el bloque **REPEAT** es que el **WHILE**, evalúa la condición antes de ejecutar las acciones, en consecuencia el cuerpo es posible que no llegue a ejecutarse nunca.

4. INSTRUCCIÓN FOR ... TO... DO

Se utiliza para efectuar un número concreto de ocasiones la misma secuencia de acciones. Su sintaxis es:

```
FOR variable := inicio TO fin DO  
BEGIN  
    <Cuerpo>  
END;
```

Donde:

inicio = Valor inicial.

fin = Valor final.

En este caso *variable* es la que cuenta la cantidad de veces de repetición.

Para repetir una suma 10 veces, se utilizará la siguiente instrucción:

```
a := 1  
FOR i := TO 10 DO  
BEGIN  
    a := a + 1;  
END;
```

INSTRUCCIÓN FOR ... DOWNTO... DO

Se utiliza para efectuar un número concreto de ocasiones la misma secuencia de acciones, pero a diferencia del anterior va decrementando el contador en una unidad. Su sintaxis es:

```
FOR variable := inicio downto fin DO  
BEGIN  
    <Cuerpo>  
END;
```

Ejemplo.

```
a := 500  
FOR i :=20 DOWNTO 5 DO  
BEGIN  
    a := a - 1;  
END;
```

IMPORTANTE: En operaciones con ciclos, se debe tener mucho cuidado ya que una mala utilización de estos, podría incurrir en iteraciones infinitas del cual no podría salir el programa que se esta ejecutando. Por lo tanto es recomendable revisar el código del programa antes de ejecutarlo.

5. BIBLIOGRAFIA

- Luis Joyanes Aguilar, Antonio Muñoz Clemente, BORLAND DELPHI Iniciación y Referencia, McGraw-Hill/Interamericana de España S.A.U.
- Francisco Charte, PROGRAMACION CON DELPHI, Ediciones Anaya
- Victor A. Melchor Espinoza DELPHI 8 Editorial MACROPERU.
- Ayuda del programa (BORLAND DELPHI HELP)

Paginas de Internet:

- <http://www.elguille.info/delphi/apuntesDelphi/apuntesDelphi.htm>
- <http://www.programacionfacil.com>
- <http://www.mailxmail.com/curso/informatica/delphi>

CAPITULO 7

INSTRUCCIONES DE TRANSFERENCIA DE CONTROL

1. INTRODUCCION

Una instrucción es la descripción de una acción o proceso que se puede ejecutar. Estas pueden ser simples o compuestas. Las instrucciones simples son aquellas que solamente definen una acción y no incluyen ninguna otra instrucción. Por el contrario las instrucciones compuestas son aquellas que incluyen otras instrucciones que posibilitan la ejecución de otras acciones. Es así que se pueden utilizar diferentes instrucciones o instrucciones, para poder realizar transferencia de control dentro del programa; a continuación se describen algunas de estas instrucciones.

2. INSTRUCCION GOTO

Transfiere la ejecución del programa a la instrucción que sigue, a la etiqueta que en la instrucción se especifique. Si bien es una instrucción cuyo uso debe evitarse, no obstante en el caso de usarla se debe tener cuidado.

Este recurso (**GOTO/LABEL**) nos permite abandonar de forma “radical” un loop (bucle), antes que sea ejecutado o al final por vías normales. **GOTO** es un comando “fuerte” que puede abandonar no solo un loop controlado por **WHILE** como también los loops generados por los otros dos comandos de repetición (**FOR** y **REPEAT**).

Ejemplo.

PROGRAM Prog10;

VAR

a,b:integer;

LABEL

fin;

BEGIN

fin:

a:=**inputbox**('numeros', 'ingrese 1º numero', '10');

b:= **inputbox** ('numeros', 'ingrese 2º numero', '20');

IF a>b **THEN**

BEGIN

GOTO fin;

END;

END.

3. INSTRUCCION BREAK

La instrucción **BREAK** para la ejecución de un ciclo si alguna condición se cumple.

4. INSTRUCCION CONTINUE

La instrucción **CONTINUE** puede ser utilizada para saltar inmediatamente a la siguiente iteración, saltando el resto del bloque de código, actualizando la variable contador si el ciclo es un a **FOR**.

5. INSTRUCCION EXIT

La instrucción **EXIT** puede ser utilizada para salir inmediatamente de un ciclo si alguna condición se cumple, y pasar al siguiente bloque de código y continuar con la ejecución del programa.

Se recomienda no utilizar las instrucciones **GOTO**, **BREAK**, **CONTINUE**, **HALT**.

6. INSTRUCCION HALT

Esta instrucción es utilizada para parar detener la ejecución de procedimientos, líneas de código, que es similar a utilizar un punto de quiebre en el código.

La instrucción **HALT** suspende la ejecución, pero que a diferencia de terminar el programa no cierra ningún archivo o despeja el valor de cualquier variable, a menos que se encuentre dentro de un ejecutable archivo (**.exe**).

7. INSTRUCCIÓN TRY/EXCEPT

Para proteger una porción de código se debe encerrar en un bloque **TRY...EXCEPT**. Entre estas dos palabras reservadas se ubica el código que está expuesto a errores (excepciones), después de **EXCEPT** se procesan estos últimos, cerrando todo el bloque con **END**. La sintaxis es:

```
TRY
    :
    {Bloque de código propenso a errores}
    :
EXCEPT
    ON <clase de excepción> DO
    : {Una sola instrucción o un bloque BEGIN..END}
    ON <otra excepción diferente> DO
    :
END;
```

8. INSTRUCCIÓN TRY/FINALLY

Hay ocasiones que es necesario ejecutar una porción de código suceda un error o no. Para esto, existe en Delphi la estructura **TRY...FINALLY**. Cuando se produce un error dentro de este bloque, se suspende el tratamiento de la excepción para ejecutar el código que sigue a **FINALLY**. Luego de terminado, se sigue con el proceso normal del proceso de error. Su sintaxis es la siguiente:

```
TRY
    {Código expuesto a errores}
FINALLY
    {Código de ejecución obligatoria}
END;
```

9. EXCEPCIONES ANIDADAS

Los bloques **TRY...EXCEPT** y **TRY...FINALLY** pueden anidarse. Se muestra un ejemplo de esta técnica a continuación.

Ejemplo. Lectura de archivos, si se produce un error mientras accedemos al archivo o procesamos sus datos, no se ejecutará la orden **CLOSEFILE** y el archivo permanecerá abierto. Esto puede ocasionar pérdida de datos ya que el sistema de directorios se vuelve inestable. Agreguemos el código para cerrar el archivo aunque se produzca un error:

```
VAR
f: File;
b: ARRAY[0..49] OF byte;
BEGIN
    TRY
        TRY
            ASSIGNFILE(f,'nombre');
            RESETFILE(f,1);
            BLOCKREAD(f,b,50);
        EXCEPT
            ON eInOutError DO
                showmessage('Error al trabajar con el archivo');
        END;
    FINALLY
        CLOSEFILE(f);
    END;
END;
```

10. CATEGORÍAS Y CLASES DE EXCEPCIONES

Las excepciones en Delphi se pueden dividir en las siguientes cinco categorías:

10.1. Conversión de tipo

Se producen cuando se trata de convertir un tipo de dato en otro, por ejemplo utilizando las funciones **INTTOSTR**, **STRTOINT**, **STRTOFLOAT**. Delphi dispara una excepción **EConvertError**.

10.2. Tipo forzado (typecast)

Se producen cuando se trata de forzar el reconocimiento de una expresión de un tipo como si fuera de otro usando el operador **AS**. Si no son compatibles, se dispara la excepción **EInvalidCast**.

10.3. Aritmética de punto flotante

Se producen al hacer operaciones con expresiones de tipo real. Existe una clase general para este tipo de excepciones –**EmathError**, pero Delphi utiliza sólo los descendientes de ésta:

\$ **EinvalidOp**: el procesador encontró una instrucción inválida.

\$ **EzeroDivide**: división por cero.

\$ **Eoverflow**: se excede en más la capacidad aritmética (números demasiado grandes).

\$ **Eunderflow**: se excede en menos la capacidad aritmética (números demasiados pequeños).

10.4. Aritmética entera

Se producen al hacer operaciones con expresiones de tipo entero. Existe una clase general definida para este tipo de excepciones llamada **EintError**, pero Delphi sólo utiliza los descendientes:

\$ **EDivByZero**: división por cero.

\$ **ERangeError**: número fuera del rango disponible según el tipo de dato. La comprobación de rango debe estar activada (indicador \$R).

\$ **EIntOverflow**: se excede en más la capacidad aritmética (números demasiado grandes).

La comprobación de sobrepasamiento debe estar activada (indicador \$O).

10.5. Entrada/Salida

Se producen cuando hay un error al acceder a dispositivos de entrada/salida o archivos.

Se define una clase genérica **-EInOutError-** con una propiedad que contiene el código de error **-ErrorCode**.

11. BIBLIOGRAFIA

- Luis Joyanes Aguilar, Antonio Muñoz Clemente, BORLAND DELPHI Iniciacion y Referencia, McGraw-Hill/Interamericana de Espana S.A.U.
- Francisco Charte, PROGRAMACION CON DELPHI, Ediciones Anaya Multimedia S.A.
- Victor Moral, DELPHI 4, Prentice Hall Iberia S.R.L.
- Victor A. Melchor Espinoza DELPHI 8 Editorial MACROPERU.
- Ayuda (BORLAND°HELP) del programa DELPHI 8

Paginas de Internet:

- <http://www.freepascal.org/sdown.html>
- <http://es.wikipedia.org/wiki/Delphi>
- <http://www.borland.com/delphi>
- <http://www.delphi 5.htm>
- <http://www.programacionfacil.com>
- <http://www.elguille.info/delphi/indice.htm>
- <http://www.marcocantu.com/>
- <http://www.marcocantu.com/epascal>
- <http://www.marcocantu.com/epascal/Spanish/default.htm>

CAPITULO 8

UNIDADES PROCEDIMIENTOS Y FUNCIONES

1. INTRODUCCION

Delphi, utiliza un lenguaje estructurado como Object Pascal, lo cual indica que los programas escritos en este lenguaje, pueden descomponerse en pequeños módulos que pueden ser llamados cuando se los necesite. Estos módulos en Delphi se llaman funciones y procedimientos, se identifican mediante un nombre.

2. PROCEDIMIENTOS – FUNCIONES

Los procedimientos y las funciones, referidas colectivamente como rutinas, contienen bloques de código.

Un procedimiento es una rutina que no devuelve un valor.

Una función es una rutina que devuelve un valor cuando se ejecuta.

Toda función que no devuelve valores, o que no realiza operaciones matemáticas o genera resultados numéricos, recibe el nombre de procedimiento. Se puede realizar alguna función o tarea específica que arroje información de modo no numérico, o sin realizar operaciones, es entonces cuando se utilizan los procedimientos. Mayor información ver ayuda de Delphi 8 sección Procedimientos y Funciones.

3. DECLARACION DE PROCEDIMIENTOS Y FUNCIONES

Cuando se declara un procedimiento o función, se debe especificar su nombre, el número y tipo de parámetros que toma, y, en el caso de una función, el tipo del valor que retorna, esta parte de la declaración es algunas veces llamado prototipo, encabezamiento, o cabecera. A continuación se escribe el bloque de código que se ejecuta cuando la función es llamada, esta parte es llamada el cuerpo de la rutina.

3.1 Declaración de Procedimientos

Un procedimiento se declara utilizando la palabra reservada “**PROCEDURE**”, y al final de la lista de argumentos no se pone ningún tipo de valor de respuesta, pues no arroja ningún resultado.

La declaración de un procedimiento tiene la siguiente forma:

PROCEDURE Procedurename (parameterList); directives;

localdeclarations;

BEGIN

statements;

END;

Donde *Procedurename* es cualquier identificador valido (nombre), *statements* es una secuencia de líneas de código que se ejecuta cuando es llamado, y (*parameterList*), *directives* y *localdeclarations*, son opcionales.

3.2 Declaración de Funciones

La declaración de una función es como la declaración de procedimientos, excepto que se utiliza la palabra reservada “**FUNCTION**”, se especifica un tipo de regreso y un valor de regreso. La declaración de una función tiene la siguiente forma:

FUNCTION functionName(parameterList): returnType; directives;

localDeclarations;

BEGIN

statements;

END;

Donde *functionName* es cualquier identificador valido (nombre), *returnType* es un identificador de tipo, *statements* es una secuencia de líneas de código que se ejecuta cuando la funcion es llamada, y (*parameterList*), *directives* y *localDeclarations*, son opcionales.

Ejemplo:

```
PROGRAM Ejemplo2;  
  PROCEDURE di (mensaje: string);  
  BEGIN  
    showmessage(mensaje);  
  END;  
BEGIN  
  di ('Materia: ');  
  di ('Computación para Ingeniería');  
END.
```

También es posible construir funciones o procedimientos que no requieran información adicional. Por ejemplo:

```
PROGRAM Ejemplo3;  
  
  PROCEDURE saluda;  
  BEGIN  
    showmessage('Hola a todos');  
  END;  
  
  PROCEDURE materia;  
  BEGIN  
    showmessage('Computación para Ingeniería');  
  END;  
  
  PROCEDURE despidete;  
  BEGIN  
    showmessage('Adiós a todos');  
  END;  
  
  BEGIN  
    saluda;  
    materia;  
    despidete;  
  
  END.
```

4. LLAMADAS A PROCEDIMIENTOS Y FUNCIONES

Cuando se llama a un procedimiento o función, el control del programa pasa del punto donde la llamada es hecha al cuerpo de la rutina. Se puede hacer la llamada usando el “nombre” de la rutina (con o sin calificadores) o usando una variable procesal que apunta a la rutina. En uno u otro caso, si la rutina es declarada con parámetros, la llamada debe pasar parámetros que corresponden en orden y tipo a la lista de parámetros de la rutina. Los parámetros que se pasa a la rutina son llamados parámetros actuales, mientras los parámetros en la declaración de la rutina son llamados parámetros formales. Las expresiones usadas para pasar tipos **CONST** (constantes) y parámetros deben ser compatibles con los parámetros formales correspondientes.

Las expresiones usadas para pasar **VAR** (variables) y parámetros de salida deben ser idénticamente del mismo tipo con los parámetros formales correspondientes, a menos que los parámetros formales sean sin tipo.

Sólo las expresiones asignables pueden usarse para pasar **VAR** (variables) y parámetros de salida.

Si los parámetros formales de una rutina son sin tipo, números y constantes verdaderas con valores numéricos no pueden ser utilizados como parámetros actuales.

5. PARAMETROS POR VALOR Y POR REFERENCIA

Los parámetros por valor son parámetros unidireccionales que se utilizan para proporcionar información al procedimiento, pero no pueden devolver valores.

Los parámetros por referencia o variable están precedidos por la palabra **VAR**, se utilizan tanto para recibir como para devolver valores.

Ejemplo: Parámetros por valor.

PROGRAM Ejemplo;

PROCEDURE cuadrado (a: byte);

BEGIN

 a := a * a;

END;

```
VAR c, d :byte;  
  BEGIN  
    c := 3;  
    cuadrado (c); // Se realiza la llamada al procedimiento  
    d := c;  
END.
```

En el ejemplo anterior se envió el valor de 'c'(c=3) a la variable 'a' del procedimiento 'cuadrado', y el procedimiento no retorna ningún valor. Por lo tanto 'a' es un parámetro por valor.

Si se desea que el valor de 'c' cambie de acuerdo al procedimiento 'cuadrado', se debe enviar 'a' como parámetro por referencia.

Para especificar un parámetro por referencia se debe indicar: (**VAR** a:byte), en la sección correspondiente a los parámetros.

Ejemplo: Parámetros por referencia.

```
PROGRAM Ejemplo;  
  PROCEDURE cuadrado (VAR a: byte);  
  BEGIN  
    a := a * a;  
  END;  
  
VAR c, d :byte;  
BEGIN  
  c := 3;  
  cuadrado (c); // Se realiza la llamada al procedimiento  
  d := c;  
  
END.
```

A diferencia del ejemplo anterior 'd' tendrá el valor de 9.

Es recomendable que cuando se utilice funciones y procedimientos, estos sean lo más cortos posibles y si algún procedimiento o función es demasiado extenso, puede ser dividido en procedimientos más pequeños, de no más de 30 líneas de código.

6. PROYECTOS Y UNIDADES

¿QUÉ ES UN PROYECTO?

Los programas se componen de instrucciones, las cuales se agrupan en bloques y en funciones. Además existe un nivel superior de agrupación de código: Las unidades y los proyectos.

Delphi, es un nivel estructurado de Pascal, conocido como Object Pascal, que permite la utilización del mismo código tantas veces como sea necesario, sin la necesidad de repetir todas las instrucciones.

En este caso, se puede realizar una de 2 alternativas: repetir en la nueva aplicación las mismas sentencias e instrucciones, lo que supone mas trabajo; o bien, poner los módulos compartidos en un fichero aparte, de modo que todas las aplicaciones puedan utilizar estas rutinas sin necesidad de repetir todo el código.

A estos ficheros se les llama unidades, de este modo, una unidad es una colección de funciones y procedimientos que tienen una característica común en cuanto a las tareas que realizan. Por ejemplo, si se desea generar un programa que realice algunas funciones matemáticas, se podría tener en una unidad donde se almacenen las instrucciones para realizar los cálculos aritméticos, otra que sea la representación visual de los resultados, otra para que interactúe con el usuario, etc.

En Delphi, los procedimientos y funciones que forman una unidad, se almacenan en un fichero independiente, de manera que cada una de ellas se encuentra en su propio archivo.

Al construir una aplicación, se debe especificar mediante indicaciones, que unidades se utilizaran. Estas indicaciones es a lo que se llama proyecto.

En realidad, un proyecto es una lista de los archivos que se utilizarán para construir una aplicación.

Creación De Unidades

Cuando se desee crear una unidad, será necesario seleccionar la opción **UNIT** en la ventana que aparece al elegir la opción New del menú **File** (Delphi 7).

En una unidad, se deben tener 2 partes claramente definidas:

- **La Parte de INTERFACE:** en la que únicamente se definen las cabeceras de funciones, procedimientos y las declaraciones de los tipos y variables que posee la unidad. Esta parte va precedida por la palabra reservada “**INTERFACE**”. Su utilidad es permitir que las aplicaciones sepan que contiene la unidad. Es la parte pública y a la que se puede tener acceso.
- **La Parte de Implementación:** Es en la que se escribe el código de las funciones y procedimientos. Esta parte va precedida de la palabra reservada “**IMPLEMENTATION**”, y termina con la palabra seguida de un punto. Es la parte privada, a la cual no se puede acceder.

Ejemplo: Declaración de una unidad, que muestre en pantalla un mensaje y pregunte el nombre mediante una caja de mensaje:

UNIT Mensajes;

INTERFACE

// Solo cabeceras de las funciones y procedimientos.

PROCEDURE di (mensaje:string);

FUNCTION preguntanombre:string;

IMPLEMENTATION

// Implementación de las rutinas

PROCEDURE di(mensaje:string);

BEGIN *// Muestra en pantalla el mensaje que se pasa por parámetro*

showmessage(mensaje);

END;

FUNCTION preguntanombre:string;

BEGIN

 preguntanombre := **inputbox**('Identificación',' Ingrese su nombre', '(Desconocido)');

END;

END.

Cuando se desee utilizar este código desde el programa principal, deberemos poner al principio de la palabra reservada **USES**, seguido de los nombres de unidades que deseamos utilizar, separados por comas.

Ejemplo:

PROGRAM Ejemplo;

USES

Mensajes, Calculos;

VAR a : word;

BEGIN

di ('Hola');

a := logaritmo(10);

END;

En este ejemplo, se emplea un procedimiento 'di' que se encuentra en la unidad Mensajes. Por ello, antes de utilizar dicho procedimiento, se debe declarar el uso de dicha unidad, mediante la palabra **USES** seguida de los nombres de unidades que se emplearan separadas por comas.

7. ENTRADA/SALIDA

7.1 Entrada de información (Datos)

La entrada de datos mediante el teclado, es el proceso común para obtener datos ingresados por el usuario, estos datos que son útiles para realizar diferentes procesos o cálculos. Para ello se usa:

La función **inputbox** (Caja de Entrada), Cuadros de texto (Tedit), y muchos componentes que tiene Delphi en su entorno de desarrollo.

Función inputbox

FUNCTION inputbox (CONST ACaption, APrompt, ADefault: string): string;

Hace la llamada a una caja de dialogo, lista para que el usuario ingrese una cadena de texto mediante el teclado. Para poder utilizar la función se utiliza la unidad Dialogs, en la cláusula **USES**.

ACaption es el título de la ventana.

APrompt es el texto que pregunta al usuario para que ingrese un valor.

ADefault es el valor que aparece en la caja de texto cuando aparece por primera vez.

Ejemplo.

```
N: =inputbox(' Título de la ventana ', ' Ingrese Contraseña ', ' Clave ' );
```

Se almacena en la variable N el valor introducido por el usuario, o el valor por defecto, es decir N = 'Clave'

7.2 Salida de Información (Resultados)

Muchas veces es necesario mostrar mensajes o presentar resultados al usuario de manera de atraer su atención. Para cumplir este objetivo, es generalmente deseable que el mensaje sea presentado en una ventana secundaria que se mostrará sobre las demás y tendrá que ser cerrada para poder continuar trabajando en el mismo programa (ventana modal).

7.2.1. Salida en pantalla (Visual)

Veamos ahora una función de Pascal que hace posible mostrar una ventana de notificación en forma simple y rápida.

Procedimiento showmessage

PROCEDURE showmessage(CONST msg: string);

Presenta en pantalla una caja de diálogo con el mensaje pasado como parámetro centrado sobre un botón OK.

El título de la ventana es el nombre del ejecutable y no se puede cambiar. Es sólo para presentar mensajes en una forma fácil y rápida, generalmente usada por los programadores durante el desarrollo.

Ejemplo.

```
showmessage ( ' Registro de empresa almacenado ' );
```

```
a: = '25';      // La variable A es de tipo String
```

```
showmessage ( A );
```

7.2.2. Salida de información a impresora

- **Impresión mediante el uso de la unidad Printers**

Se puede imprimir los datos y resultados deseados de dos formas distintas, la primera al estilo de Turbo Pascal y la segunda mediante graficación, que consiste en colocar el dato o resultado en la posición deseada utilizando una clase como **canvas**.

Ejemplo 1. Estilo de Turbo Pascal

```

VAR
impres:textfile;
BEGIN
    Assignprn(impres);           // Asigna "impres" a la impresora
    Rewrite(impres);             // Abre la Impresora
    Writeln(impres,' numero cuadrado cubo');
    Write(impres,' Borland Delphi'
    Closefile(impres);           // Cierra la Impresora
END;

```

Ejemplo 2. Utilizando **canvas** (graficación)

```

BEGIN
WITH Printer DO
    BEGIN
        BeginDoc;                // Abre el documento para impresión
        canvas.TextHeight('10'); // Altura del texto
        canvas.TextOut(400,400,'COMPUTACION PARA INGENIERIA');
        canvas.MoveTo(400,500);   // Dibuja una linea de x=400, y=500
        canvas.LineTo(4000,500); // A x=4000, y=500
        canvas.TextOut(400,800,'ENTRADA DE DATOS');
        EndDoc;                   // Cierra el documento de impresión
    END;
END;

```

Para mayor referencia ver: capítulo 10 Manejo de Gráficos con **canvas**

- **Impresión mediante el uso del componente RichEdit**

Para ello se debe colocar el componente RichEdit en el formulario, y adicionar todo lo que se quiera imprimir. Ejemplo.

```

BEGIN
    // Adicionamos el texto al componente
    Richedit1.Lines.Add( 'COMPUTACION PARA INGENIERIA' );
    Richedit1.Lines.Add( 'CARRERA DE INGENIERIA CIVIL' );
    Richedit1.Lines.Add( Edit1.Text );

```

```
Richedit1.Print( 'Impresion' );    // Impresión Directa de todo el contenido
END;
```

- **Impresión mediante el uso de QuickReport**

Primero se debe cargar al IDE el componente dclqrt70.bpl ubicado en: c:\Program Files\Borland\Delphi7\bin (para el Delphi 7).

Luego se debe colocar un QuickRep en un formulario, configurar tamaño de hoja, margen, etc.

QReport, usa el modelo de bandas para la construcción de reportes. Este componente QRBand, en su propiedad BandType, permite construir los siguientes tipos importantes de banda.

Permite poner dentro componentes QRlabel, colocar dentro de esta banda los componentes de datos impresos, QRDBText y QRDBImage.

Ejemplo. El procedimiento es el siguiente, crear otro formulario o ventana, colocar un componente QuickRep, colocar dentro un QRBand y dentro de este colocar los componentes QRlabel donde uno desee. El código a continuación muestra el desarrollo de impresión.

```
BEGIN
```

```
WITH form2 DO
```

```
BEGIN
```

```
Qrlabel1.Caption:=form1.Label1.Caption;    // Captura el valor de la etiqueta
Qrlabel2.Caption:=form1.Edit1.Text;        // Captura el valor de la caja de texto
Qrlabel3.Caption:='Computacion Para Ingenieria';    // Captura el texto
Qrimage1.Picture:=form1.Image1.Picture;    // Captura la imagen
QuickRep1.Preview;                        // Vista previa de la pagina a imprimir
Quickrep1.Print;                          // Impresión de la pagina
```

```
END;
```

```
END;
```

Nota. Los valores encerrados entre comillas ‘ ‘ son de tipo string, al igual que el valor de un componente Edit en su propiedad **Text**, y otros componentes en su propiedad **Caption**. Para mayor referencia ver los programas en Delphi aplicados a la carrera de Ingeniería Civil, en los procedimientos para impresión.

8. BIBLIOGRAFIA

- Luis Joyanes Aguilar, Antonio Muñoz Clemente, BORLAND DELPHI Iniciacion y Referencia, McGraw-Hill/Interamericana de España S.A.U.
- Francisco Charte, PROGRAMACION CON DELPHI, Ediciones AnayaVictor A. Melchor Espinoza DELPHI 8 Editorial MACROPERU.
- Ayuda del programa (BORLAND DELPHI HELP)

Paginas de Internet:

- <http://www.elguille.info/delphi/apuntesDelphi/apuntesDelphi.htm>
- <http://www.programacionfacil.com>
- <http://www.mailxmail.com/curso/informatica/delphi>

CAPITULO 9

MANEJO DE ARCHIVOS

1. INTRODUCCION

Un archivo o fichero es una estructura de datos que, normalmente, se almacena en la memoria auxiliar (disco duro, cinta o disquete) de una computadora. No tiene un tamaño fijo, necesariamente. Se referencia por un nombre o identificador. La información se traspasa de la memoria principal hacia él, ó de él a la memoria principal a través de una memoria intermedia (buffer), y por medio de procedimientos de lectura/escritura (Entradas/Salidas).

2. TIPOS DE ARCHIVOS EN GENERAL

Los principales tipos de archivo son:

- Archivos de entrada o colecciones de datos almacenados en un dispositivo de entrada.
- Archivos de salida o colecciones de datos visualizados por el monitor, impresos a través de la impresora o enviados a terminales remotos a través de un modulador/demodulador (modem).
- Archivos de programa o conjunto de ordenes de un determinado lenguaje almacenado en un dispositivo de almacenamiento.
- Archivos de texto o grupo de caracteres almacenados en un dispositivo de almacenamiento.

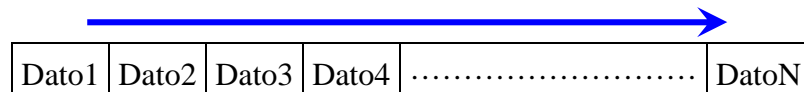
3. TIPOS DE ACCESO A UN ARCHIVO

Se puede acceder a un archivo de modo secuencial o directo (acceso aleatorio).

Un acceso secuencial debe tratar los datos del archivo elemento a elemento, comenzando desde el primero. Un acceso directo puede tratar solo uno de los elementos del archivo sin necesidad de tener que recorrer los demás. La siguiente figura muestra las diferencias entre un acceso secuencial y un acceso directo.

3.1 Acceso secuencial

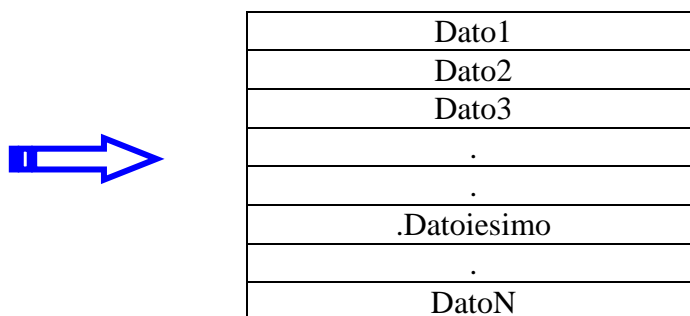
Para llegar al elemento DatoN se habrán de procesar primero los N-1 datos anteriores cuadro 3.1.



Cuadro 3.1

3.2 Acceso directo

Se puede acceder directamente a un elemento Datoiesimo sin recorrer previamente los anteriores cuadro 3.2.



Cuadro 3.2

4. TIPOS DE ARCHIVOS EN OBJECT PASCAL

Los archivos de datos pueden ser de tres clases:

- Archivos de texto (**Text**) o secuenciales, a los que se accede secuencialmente.
- Archivos con tipo (**File Of**) con un tipo base simple o estructurado, a los que se accede directamente.
- Archivos sin tipo (**File**) usados para operaciones de entrada/salida a bajo nivel.

Procedimientos y Funciones Estandar para Gestion de archivos

Para los archivos de texto cuadro A:

PROCEDIMIENTOS: Append, Assign, ChDir, Close, Erase, Flush, GetDir, MKDir, Read, Readln, Rename, Reset, Rewrite, RmDir, SetTextBuf, Write, Writeln FUNCIONES: Eof, Eoln, SeeKEof, SeeKEof, SeeKEln

Cuadro A

Para los archivos con tipo cuadro B:

PROCEDIMIENTOS: Assingn, ChDir, Close, Erase, GetDir, MkDir Read, Rename, Reset, Rewrite, RmDir, Seek, Truncate, Write FUNCIONES: Eof, Filepos, Filesize, Ioresult

Cuadro B

Para los archivos sin tipo cuadro C:

PROCEDIMIENTOS: Assign, BlockRead, BlockWrite, ChDir, Close
Erase, GetDir, MKDir, Rename, Reset, **Rewrite**, Rmdir, **Seek**,
Truncate.
FUNCIONES: Eof, **Filepos**, **Filesize**, Ioresult

Cuadro C

5. ARCHIVOS DE TEXTO (SECUENCIALES)

Un archivo de texto es un grupo de líneas cada una de las cuales contiene palabras, cantidades, caracteres especiales, etc.

La marca de final de línea la determinan los caracteres CR (retorno de carro) y LF (avance de línea) que se introducen como últimos caracteres de la línea cada vez que, al escribir, se pulsa la tecla INTRO.

5.1 Creación de archivos de texto con un editor

Existen editores de texto desde los mas sencillos como el editor EDIT del DOS, hasta los mas sofisticados como pueden ser Word, WordPerfect, Page Maker, etc.

Con estos editores se pueden confeccionar los archivos de texto, pero habrá de tenerse en cuenta que, dado que los editores avanzados tienen un formato especial, será preciso guardar esos documentos en formato ASCII-formato DOS para que luego puedan usarse en una aplicación sin necesidad de tener que utilizar el entorno en el que se escribió el archivo de texto.

5.2 Funciones EOLN/EOF

Tanto la función **EOLN** como la función **EOF** devuelven un valor de tipo lógico que es verdadero (True) si se alcanzan las marcas de final de línea o de archivo, respectivamente, devuelven falso (False) cuando no se han alcanzado esas marcas.

Ejemplo:

```
WHILE NOT EOF (Fichero) DO  
    Read (Fichero, Registro);  
WHILE NOT EOLN (Fichero) DO  
    Read (Fichero, Línea) ;
```

5.3 Archivos tipo char

Los archivos de tipo carácter (tipo **char**) son, en realidad, archivos de texto con las siguientes salvedades:

- Un archivo de texto se divide en líneas, uno de carácter (tipo **char**) no.
- Un archivo de carácter se lee y escribe carácter a carácter; uno de texto no necesariamente.

5.4 Tratamiento de archivos de texto

Los pasos para el tratamiento de un archivo de texto son:

- Declaración del manipulador interno del archivo.
- Asignación del manipulador interno a un archivo externo.
- Apertura del archivo para el modo de proceso que se iniciara.
- Lectura o escritura de los datos desde o en el archivo (según la apertura).
- Cierre del archivo.

5.4.1 Declaración de un archivo de texto

Las operaciones para declarar un archivo son las siguientes:

- Declarar una variable de tipo **Text**.
- Asociar a esta variable (manipulador interno) un nombre de archivo externo.

Ejemplo:

```
PROGRAM archivodetext;  
USES Dialogs;  
VAR  
documento: Text;
```

Esta operación asocia una variable de tipo archivo externo almacenado en el disco.

Ejemplo:

```
PROGRAM archivodetexto;  
USES Dialogs;  
VAR  
    documento: Text;  
BEGIN  
    Assignfile (documento, 'carta.doc' );  
END.
```

5.4.2 Apertura de un archivo

La apertura de un archivo puede realizarse de tres maneras, según la operación que con el se vaya a realizar.

Para crear un archivo no existente se utiliza el procedimiento de apertura **Rewrite**. Si el archivo ya existe, anula del mismo el contenido del anterior.

Para leer un archivo ya existente se utiliza el procedimiento de apertura **Reset**.

Este procedimiento sitúa el puntero al comienzo del archivo y solo posibilita sobre el mismo operaciones de lectura. Si el archivo externo no existe se produce un error de entrada/salida.

Para añadir mas líneas a un archivo ya existente se utiliza el procedimiento de adición **Append**.

Ejemplo:

```

PROGRAM Archivodetexto;
USES Dialogs;
VAR
    documento:Text;
BEGIN
    Assign(documento, `Carta.Doc`);
    {$I-}
    Reset (documento);
    {$I+}
    IF IOResult (documento) <> 0 THEN
        BEGIN
            Rewrite (documento);
            llenar (documento)
            Close (documento);
            Reset (Documento);
            ver (Documento);
            Close (Documento);
        END
    ELSE
        BEGIN
            ver (documento);
            Close (documento);
        END;
    Readkey;
END.

```

5.4.3 Escritura de un archivo

Los procedimientos para volcar información desde la memoria a un archivo de texto son **Write** y **Writeln**.

Ejemplo:

```

PROCEDURE llenar (VAR d:Text);
VAR
    linea:string;
BEGIN
    showmessage (`Frase: Para Fin – 999`);
    linea:=inputbox('Archivos', 'Ingrese una linea', '2');

```

```
    WHILE Linea <> -999 DO
    BEGIN
        Writeln (d,Linea);    // Escritura en el archivo
    END
END;
```

5.4.4 Lectura de un archivo

Los procedimientos para leer información desde un archivo de texto a la memoria principal son **Read** y **Readln**.

Ejemplo:

```
PROCEDURE ver (VAR d:Text);
VAR
    linea: string;
BEGIN
    Readln (d, linea);
    WHILE NOT (Eof(d)) DO
    BEGIN
        showmessage (linea);
        Readln (d,linea)
    END;
END;
```

5.4.5 Añadir datos un archivo de texto

Para añadir mas líneas a un archivo se utiliza el procedimiento de apertura **Append**. Si el archivo no existe se produce un error y si existe el puntero se sitúa al final del archivo. Si el archivo estuviera ya abierto, se cierra e inmediatamente se vuelve a abrir.

Ejemplo:

```
PROCEDURE nuevaslineas (VAR d:Text);
VAR
    linea: string;
BEGIN
    Append(d);
    linea:= inputbox('Archivos', 'Ingrese una linea', '2') ;
    WHILE linea <> -999 DO
```

```
      BEGIN
      Writeln (d,linea);
    END;
  END;
```

6. ESTRUCTURA DE UN ARCHIVO CON TIPO (BINARIO)

Los archivos con tipo tienen como tipo base uno de los tipos estándar de Object Pascal o estructuras de tipo registro. Estos elementos de los archivos con tipo no acupan, necesariamente, posiciones contiguas de memoria. Para acceder a cada elemento no es preciso recorrer los registros anteriores para situarse en el indicado. Solamente será necesario indicar su posición para tener acceso a un determinado elemento. El almacenamiento de los elementos de estos archivos es en un formato binario comprimido, por lo que no se puede generar con un editor de textos y será, por tanto, preciso utilizar un programa que lo cree y lo gestione.

6.1 Tratamiento de archivos de acceso aleatorio

Para la creación, lectura y escritura de un archivo de acceso aleatorio son necesarias las siguientes condiciones:

- Definir el tipo básico (registro) del archivo si no es un tipo estándar.
- Declarar el tipo de archivo.
- Definir las variables de tipo archivo y registro previamente declarados en la sección de declaración de tipos.
- Enlazar la variable de tipo archivo (manipulador interno) con el nombre que se le de al archivo de datos (archivo externo).
- Realizar la apertura del archivo en el modo de creación o de lectura/escritura si ya existe.
- Situar el puntero del archivo en una determinada dirección del mismo para realizar una operación de lectura o de escritura.
- Cerrar el archivo.

6.2 Declaración de un tipo de datos archivo binario

La declaración del tipo de datos se realiza en el apartado de declaraciones del programa principal. Su sintaxis es la que se muestra en la siguiente figura. Notar que en el diagrama de sintaxis general para todos los archivos, se aplicara a la dirección de la sintaxis **File OF TipoBase**.

Si el **TipoBase** es de tipo registro deberá definirse antes de la declaración del tipo archivo.

Ejemplo:

```
PROGRAMA ArchivoAleatorio;  
USES Dialogs,Sysutils;  
TYPE  
tiporegistro = RECORD  
    apellidos: string [25] ;  
    nombre: string [20] ;  
    telefono: string [15];  
END;  
TipoArchivo = File OF tiporegistro;  
VAR  
    reg: tiporegistro;  
    archivo: tipoarchivo;
```

6.3 Asignación de archivos

Esta operación consiste en asociar el manipulador interno del archivo (variable que se ha declarado de tipo archivo) con el nombre el archivo de datos que el programa va a procesar.

Ejemplo:

```
PROGRAMA ArchivoAleatorio;  
USES Dialogs;  
TYPE  
tiporegistro = RECORD  
    apellidos: string [25];  
    nombre: string [20];  
    telefono: string [15];
```

```

END;
tipoarchivo = File OF TipoRegistro;
VAR
    reg: tiporegistro;
    archivo: tipoarchivo;
BEGIN
    Assign (Archivo, `Agenda.Dat `);
    --
    --
END.

```

6.4 Apertura del archivo

En Object Pascal existen 2 formas para abrir un archivo con tipo, para ello se utilizan los procedimientos: **Rewrite** y **Reset**.

El primer tipo de apertura (**Rewrite**) abre el archivo para crearlo. Si no existe, lo crea. Puede darse el caso de que el archivo ya exista y, si es así, el archivo que se crea con la instrucción **Rewrite** reemplazará al archivo existente. Por tanto es importante cerciorarse de la existencia o no del archivo. El segundo tipo de apertura (**Reset**) abre el archivo para lectura y escritura. Por lo tanto el archivo debe existir, si el archivo no existe se producirá un error.

Ejemplo:

```

PROGRAMA ArchivoAleatorio;
USES Dialogs, Sysutils;
TYPE
    tiporegistro = RECORD
        apellidos: string [25];
        nombre: string [20];
        telefono: string [15];
END;
tipoarchivo = File OF tiporegistro;
VAR
    reg: tiporegistro;
    archivo: tipoarchivo;
BEGIN
    Assign (Archivo, `Agenda.Dat `);
    Rewrite (Archivo);  {Se abre para la creación}
END.

```

6.5 Operaciones de lectura, escritura y fin de archivo

Para este ejemplo hay que considerar que las declaraciones siguen siendo las de los ejemplos anteriores y, también, las que se precisan para controlar cinco (5) entradas de fichas en la agenda.

Ejemplo:

BEGIN

Assign (archivo, `agenda.dat`);

Rewrite (Archivo);

FOR I: = 0 **TO** 4 **DO**

BEGIN

reg.apellidos:= **inputbox**('archivos', ingrese apellido', 'rodriguez');

reg.nombre:= **inputbox**('archivos', ingrese nombre', 'raul');

reg.telefono:= **inputbox**('archivos', ingrese telefono', '4125631');

Seek (archivo, i);

Write (archivo, reg);

END;

END.

* Las operaciones de manipulación de archivos suponen los siguientes procesos:

- Declaración del tipo.
- Declaración de la variable (también llamada manipulador interno).
- Asignación del manipulador a un archivo externo.
- Apertura, proceso de los datos y cierre del archivo.

6.6 Cierre de un archivo

Para evitar pérdidas de información de lectura y escritura de archivos, es necesario cerrar el archivo que ha sido abierto. Y se utilizarán los procedimientos **Close** y **Closfile**.

Ejemplo:

BEGIN

Assign (Archivo, `Agenda.Dat `);

Rewrite (Archivo);

(Instrucciones de operaciones con archivos)

Close (Archivo) *// La instrucción cierra el archivo*

END.

7. MANTENIMIENTO DE ARCHIVOS ALEATORIOS

Las operaciones para el mantenimiento de archivos en general y, en particular, de los archivos de acceso aleatorio se pueden dividir en grupos según la finalidad del mantenimiento de los mismos. Se agrupara estas operaciones de mantenimiento en tres apartados o grupos:

- Operaciones de acceso al archivo.
- Operaciones para consultas.
- Operaciones para actualización de los registros y del archivo.

7.1 Operaciones de acceso al archivo

Tras la creación de un archivo, es preciso realizar con sus registros distintas operaciones entre las que se pueden incluir las búsquedas de un determinado registro, la modificación de los datos del registro encontrado o la baja temporal o definitiva de los datos que el registro que se esta procesando contiene. Para conseguir el acceso a un determinado registro Object Pascal dispone de procedimientos y funciones internas, cuya sintaxis ya se ha descrito, y que son las siguientes:

- Posicionamiento en un determinado registro (procedimiento **Seek**).
- Gestión del registro en el que esta posicionado el puntero en el archivo (función **Filepos**).
- Obtención del tamaño del archivo (función **Filesize**).

Ejemplo:

```
PROCEDURE leer (VAR f:archaula);  
VAR  
    seguir: char;  
    alumno: aula;  
BEGIN  
    Reset (f);  
    Seek (f,0); {Se accede al archivo f y al registro 0}  
    WHILE NOT (Eof (f)) DO  
        BEGIN  
            Read (f,alumno);  
            Writeln ( `EXPEDIENTE: `,alumno.expe:4);  
            Writeln ( `CURSO: `,alumno.curso:2);  
            Writeln ( `GRUPO: `,alumno.grupo:2);  
            Write ( `APELLIDOS: `,alumno.apelli1:16);  
            Writeln (alumno.apelli2:16);  
            Writeln (NOMBRE: `,alumno.nombre:15);  
            Writeln ( `NOTA: `,alumno.nota:4:2);  
            Writeln ( `CALIFICACION: `,alumno.calificacion);  
            seguir:= ReadKey  
        END;  
    Close (f)  
END;
```

Lectura/Escritura de los registros

Luego del posicionamiento se accede al registro por medio de las operaciones de lectura (**Read**) y escritura (**Write**) para recoger del archivo o grabar en el mismo información.

7.2 Operaciones para consultas

Una vez posicionados en un registro determinado las operaciones para consultas de los registros suponen la lectura de los datos del mismo y su presentación a través del periférico elegido para la operación.

Para consultar un único registro se precisa:

- Posicionarse en el mismo.
- Pasar del archivo a la memoria principal los datos del registro (leerlo).
- Gestionar la salida de los datos del registro a través del periférico elegido.

Ejemplo:

```

PROCEDURE veruno (VAR f:archaula);
VAR
    alumno:aula;
    expebus:string;
BEGIN
    Reset (f);
    Write ( `Expediente del alumno: `);
    Readln (expeBus); {se lee el expediente para buscarlo}
    Seek (f , 0);
    Read (f , alumno);
    WHILE (Not (Eof(f))) AND NOT (Encontrado (alumno,expebus)) DO
        Read (f , alumno);
        IF encontrado (alumno, expebus) THEN
            presentar (alumno)
        ELSE
            Write ( `No existe `,expebus);
    Close (f)
END;

```

* Observar que el procedimiento solamente gestiona la búsqueda de un registro, y si lo encuentra, lo presenta. Sino lo encuentra, emite el oportuno mensaje de error.

Para consultar todos los registros se precisa:

- Posicionarse en el primer registro del archivo.
- Pasar del archivo a la memoria principal los datos del registro.
- Gestionar la salida de los datos del registro a través del periférico elegido.
- Continuar repitiendo los dos pasos anteriores hasta alcanzar el final del archivo.

Ejemplo:

```
PROCEDURE veruarios (VAR f:archaula);  
VAR  
    alumno:aula;  
BEGIN  
    Reset (f);  
    Seek (f,0);  
    Read (f,alumno); (.../...)  
    (.../...)  
    WHILE NOT (Eof(f)) DO  
    BEGIN  
        presentar (alumno);  
        Read (f,alumno)  
    END;  
    Close (f)  
END;
```

7.3 Actualización de registros

Las operaciones de actualización de los registros de un archivo son:

- Añadir nuevos registros.
- Modificar datos de registros ya existentes.
- Bajas lógicas de registros.
- Eliminación definitiva de registro.

7.3.1 Adición de nuevos registros al archivo

Esta operación de la actualización de los registros de un archivo depende de las condiciones de generación de los registros por **clave única** o por **clave repetida**.

Si se gestiona una única clave, las operaciones suponen:

- Confirmación de la **no existencia** de la clave del nuevo registro en el archivo.
- Petición de los demás datos del nuevo registro si el paso anterior resulta verdadero.

- Posicionamiento en la dirección lógica en que se debe grabar el registro.
 1. Si el archivo no tiene limitado el tamaño, el posicionamiento será al final del archivo.
 2. Si esta organizado para poder acceder al mismo por medio de una función **Hash**, el posicionamiento se realizará en la zona principal si la posición no esta ocupada, o en la zona de sinónimos si lo estuviera.
 3. Escritura en esa posición del archivo del nuevo registro generado.

Ejemplo:

```

PROCEDURE unomas (VAR f:archaula);
VAR
  alumno:aula;
  expebus: string;
  encontrado: boolean;
BEGIN
  Reset (f);
  encontrado:= False;
  Write ( `Expediente del campo Nuevo alumno: `);
  Readln (ExpeBus); {Se lee el expediente para buscarlo}
  Seek (f,0); Read (f, Alumno);
  encontrado:= alumno. expe = expebus
  WHILE (Not(Eof(f))) AND (not(encontrado)) DO
    IF NOT (encontrado) THEN
      BEGIN
        Read (f, alumno);
        encontrado:= alumno. expe = expebus
      END;
    IF NOT (encontrado) THEN
      BEGIN
        otroscampos (alumno);
        Seek (f, Filesize (f));
        Write (f, alumno);
      END;
    Close (f)
  END;

```

* Observar que el procedimiento sitúa el registro al final del archivo por lo que se considera que el archivo no tiene limitación en el número de registros.

7.3.2 Modificación de los datos de un registro ya existente

La modificación de todos o algunos de los campos de un determinado registro ya existente suponen:

- Buscar el registro.
 - Presentarlo en la pantalla.
 - Posibilitar la modificación del campo deseado.
1. Si no es el campo clave que se modifica, se confirma y se graba en la misma posición que ocupa el registro.
 2. Si es el campo clave, el archivo es de clave única y al archivo se accede por medio de una función **Hash**, el proceso de modificación supone una baja física y una nueva alta de registro, así como el traslado a la zona principal del archivo del registro que, ocupando la zona de colisiones, produzca la misma dirección lógica que la del registro que se ha eliminado.
 3. Si es el campo clave y el archivo no es de clave única, se elimina el registro que se quiere modificar, se graba uno nuevo con los datos actualizados y se compacta el archivo.

Ejemplo:

```
PROCEDURE modificar (VAR f:archaula);  
VAR  
    alumno:aula;  
    expebus: string;  
    encontrado: boolean;  
BEGIN  
    Reset (f);  
    encontrado:= False;  
    Write ( `Expediente del campo Nuevo alumno: `);
```

```
Readln (expebus); {Se lee el expediente para buscarlo}  
Seek (f,0); Read (f, alumno);  
Read (f, alumno):  
encontrado: = alumno. expe = expebus  
{...}  
WHILE (not (Eof(f))) AND (not (encontrado)) DO  
IF NOT (encontrado) THEN  
  BEGIN  
    Read (f , alumno);  
    encontrado : = alumno.expe =expebus  
  END;  
IF encontrado THEN  
  BEGIN  
    Seek (f, Filepos (f) - 1);  
    Write (f, alumno);  
  END;  
Close (f)  
END;
```

La instrucción **Seek** (f, **Filepos** (f) – 1); sitúa el puntero en el registro que acaba de leer, que es el anterior al registro al que apunta en ese momento.

7.3.3 Bajas lógicas de registros

La baja lógica de un registro consiste en marcar el registro de manera que esa marca sirva para excluirlo o no de algunos procesos del tratamiento del archivo.

Los pasos para una baja lógica son los siguientes:

- Buscar el registro.
- Presentarlo en pantalla.
- Pedir confirmación de la baja lógica.
- Si se confirma la baja, marcar el campo que se haya diseñado para la baja y gravar el registro en la misma posición que ocupaba.
- Si no se confirma la baja se continuará con el proceso de tratamiento general del archivo.

Ejemplo:

```

PROCEDURE BajaLogica (VAR f:archaula);
VAR
    alumno:aula;
    expebus: string;
    encontrado: boolean;
BEGIN
    Reset (f);
    encontrado:= False;
    Write ( `Expediente del campo Nuevo alumno: `);
    Readln (expebus); {Se lee el expediente para buscarlo}
    Seek (f,0);
    Read (f, alumno);
    encontrado := alumno.expe = expebus
    WHILE (NOT (Eof(f))) AND (NOT (encontrado)) DO
    IF NOT (encontrado) THEN
        BEGIN
            Read (f , alumno);
            encontrado := alumno.expe =expebus;
        END;
    IF encontrado THEN
        BEGIN
            presentar (alumno);
            confirmado := confirmabaja (alumno);
            IF confirmado THEN
                BEGIN
                    marcarregistro (alumno);
                    Seek (f, Filepos (f) – 1 );
                    Write (f, alumno);
                END;
            END;
        Close (f)
    END;

```

7.3.4 Bajas físicas o eliminación definitiva de registros

La baja física o eliminación definitiva de un registro supone la pérdida definitiva de los datos que ese registro contiene. A veces, estos registros que se eliminan del archivo maestro se guardan en otro archivo, llamado archivo histórico, del que se hace uso en el caso de que, posteriormente, se quieran incorporar de

nuevo al archivo maestro registros que previamente se hubieran eliminado de el. Los pasos para la baja física son los siguientes:

- Buscar el registro.
- Ver si está marcado como baja lógica.
- Si está marcado como baja lógica, continuar con el siguiente paso. Si no esta marcado como baja lógica, se suele interrumpir la operación tras emitir el oportuno mensaje de error. Esta cautela tiene la finalidad de establecer diferentes filtros para que, si se quiere eliminar el registro, el usuario deba confirmarlo varias veces de modo que no se pierda la información que el mismo contiene por equivocación.
- Presentarlo en pantalla.
- Pedir confirmación de la baja física.
- Si se confirma la baja, y el archivo se ha diseñado para un acceso por **Hash**, se graba en la dirección lógica que ocupa el registro un registro en blanco y, en caso de que en la zona de colisiones alguna clave produzca la dirección del registro que se elimina, el primer registro que produzca esa dirección se pasa a la zona principal.
- Si no se confirma la baja se continua con el proceso de tratamiento general del archivo.

Ejemplo:

```
PROCEDURE EliminarRegistro (VAR f:archaula);  
VAR  
    alumno:aula;  
    expebus: string;  
    encontrado, confirmado: boolean;  
BEGIN  
    Reset (f);  
    encontrado:= false;  
    Write (`expediente del nuevo alumno:`);  
    Readln (expebus); {Se lee el expediente para buscarlo}  
    Seek (f , 0);  
    Read (f, alumno);
```

```

encontrado := alumno.expe = expebus
WHILE (NOT (Eof(f))) AND (NOT (encontrado)) DO
IF NOT (encontrado) THEN
  BEGIN
    Read (f , alumno);
    encontrado := alumno.expe = expebus
  END;
IF encontrado THEN
  BEGIN
    presentar (alumno);
    confirmado := confirmabaja (alumno);
    IF confirmado THEN
      BEGIN
        registroblanco (alumno);
        Seek (f, Filepos (f) – 1 );
        Write (f, alumno);
        compactar (f);
      END
    END;
  Close (f)
END;

```

* El dar un registro como baja definitiva de un archivo es una operación en la que se han de tomar todas las precauciones posibles. Además de pedir confirmación al borrado de la ficha, sería conveniente articular algún sistema que guarde las fichas eliminadas (por ejemplo, en un archivo histórico).

8. ORDENACIÓN DE ARCHIVOS

Para ordenar archivos se pueden seguir los mismos algoritmos que ya se vieron en el tratamiento de vectores y listas, con la diferencia de que el resultado de la ordenación deberá volcarse en un dispositivo externo para que los datos queden ordenados de modo permanente.

9. FUSIÓN O MEZCLA DE ARCHIVOS

Se entiende como mezcla o fusión de archivos el proceso que permite combinar en un solo archivo los registros de dos archivos, con idéntica estructura de registros, ya ordenados por su campo clave.

El ejemplo mezcla en el archivo ACTUAL.DAT los archivos de operaciones de la Caja de cierta entidad comercial CAJA 1.DAT y CAJA 2.DAT.

Tanto ACTUAL.DAT como CAJA.DAT y CAJA.DAT tienen registros con la estructura del cuadro 9.1:

Nombre campo	Tipo de dato	Tamaño
Referencia	CADENA	5
Unidades	ENTERO	
TotalArticulo	REAL	

Cuadro 9.1

Ejemplo:

```

PROCEDURE mezclar (VAR a:archivo;c1,c2: archivo);
VAR
    reg1, reg2: registro;
BEGIN
    {Se suponen abiertos los tres archivos}
    i:= 0; j:= 0; k:= 0 ;
    Seek (c1,i); Seek (c2,j); Seek (a,k);
    Read (c1, reg1);
    Read (c2, reg2);
    WHILE (NOT (Eof (C1))) AND (NOT (Eof (C2))) DO
        BEGIN
            IF Reg1. Referencia <= Reg2. Referencia THEN
                BEGIN
                    Write (a,reg1);
                    i:= i + 1;
                    Seek (c1,i);
                    Read (c1,reg1)
                END
            ELSE
                BEGIN
                    Write (a,reg2);
                    j:= j + 1 ;
                    Seek (c2 , j) ;
                    Read (c2,reg2)
                END;
            k:= k + 1 ;
            Seek (a,k)
        END
    END

```

```
END;  
  
IF Eof(c1) THEN  
    WHILE NOT (Eof (c2)) DO  
        BEGIN  
            Write (a , reg2);  
            Read (c2 , reg2);  
            k: = k+1;  
            Seek (a , k);  
        END  
    ELSE  
        WHILE NOT (Eof (c1)) DO  
            BEGIN  
                Write (a , reg1);  
                Read (c1 , reg1);  
                k: = k + 1;  
                Seek (a , k)  
            END;  
    Close (a); Close (c1); Close (c2);  
END;
```

10. BIBLIOGRAFIA

- Luis Joyanes Aguilar, Antonio Muñoz Clemente, BORLAND DELPHI Iniciacion y Referencia, McGraw-Hill/Interamericana de Espana S.A.U.
- Francisco Charte, PROGRAMACION CON DELPHI, Ediciones Anaya Multimedia S.A.
- Victor Moral, DELPHI 4, Prentice Hall Iberia S.R.L.
- Victor A. Melchor Espinoza DELPHI 8 Editorial MACROPERU.
- Ayuda del programa (BORLAND HELP)

Paginas de Internet:

- http://www.delphi3000.com/articles/article_3228.aspHistoria de la Computación.htm
- <http://www.swissdelphicenter.ch/en/showcode.php?id=1155>Historia de la computación4.htm
- <http://www.freepascal.org/sdown.html>
- <http://es.wikipedia.org/wiki/Delphi>
- <http://www.borland.com/delphi>
- <http://www.delphi5.htm>
- <http://www.programacionfacil.com>
- <http://www.elguille.info/delphi/indice.htm>
- <http://www.marcocantu.com/>
- <http://www.marcocantu.com/epascal>
- <http://www.marcocantu.com/epascal/Spanish/default.htm>

CAPITULO 10

MANEJO DE GRAFICOS

1. INTRODUCCION

Los gráficos permiten agregar un toque profesional a las aplicaciones. Y de este modo conseguir unas aplicaciones mucho más atractivas para el usuario.

2. GRAFICOS EN DELPHI

En Delphi es posible incluir gráficos mediante 5 distintos objetos:

- **Image** en la página *Additional*: Este objeto permite insertar en los formularios imágenes contenidas en ficheros con extensión JPG, JPEG, BMP, ICO, EWF o WMF.
- **Shape** en la página *Additional*: Permite generar figuras geométricas estáticas en el formulario.
- **Bevel** en la página *Additional*: Permite dar un efecto tridimensional a las aplicaciones.
- **Canvas** de *Image*: Permite crear figuras geométricas de forma dinámica en el formulario.
- **PaintBox** de la página *System*: es una mejora del componente *Image*, pues utiliza menos memoria.

2.1 GRAFICOS CON IMAGE

Este objeto tiene 2 propiedades importantes, que son:

1. **Picture**: Que hace mención al archivo que será visualizado.

2. **Autosize:** Si se encuentra en True, si la imagen es mayor o menor, el objeto se acoplará al tamaño de la imagen, en caso contrario, se deberá acoplar al tamaño de dicho objeto.

2.2 GRAFICOS CON SHAPE

Este objeto es rara vez usado, pues no recibe ningún tipo de datos del usuario, ni muestra datos o resultados.

Los valores de la propiedad *Shape* de dicho objeto, demarcarán el tipo de figura geométrica que se presentará:

Constante	Figura Geométrica
StCircle	Círculo
StEllipse	Elipse
StRectangle	Rectángulo
StRoundSquare	Cuadrado con las esquinas redondeadas
StRoundRectangle	Rectángulo con las esquinas redondeadas
StSquare	Cuadrado

Así pues *Shape* permite dibujar figuras geométricas, las cuales pueden tener propiedades como color de contorno, color de relleno, estilo de trazo, etc. Todas estas cualidades se encuentran en la propiedad *Pen*. Las subpropiedades que se encuentran en esta propiedad son:

- *Color:* Color de contorno de la figura. Contiene una lista con los colores disponibles.
- *Mode:* Es el modo como se dibuja el contorno, esta propiedad es algo complicada.
- *Width:* Ancho del contorno.
- *Style:* Tipo de trazo en el contorno, punteado, continuo, etc. En esta se encuentran los siguientes valores:

Valor	Significado
PsSolid	Trazo continuo
PsDash	Trazo discontinuo
PsDot	Trazo punteado
PsDashDot	Trazo formado por una línea y un punto
PsDashDotDot	Trazo formado por una línea y dos puntos
PsClear	Sin borde

La propiedad *Pen* se refiere al contorno de dicho objeto, y la propiedad *Brush* se refiere al relleno de dicho objeto. Las subpropiedades de la propiedad *Brush* son:

- *Color*: Color de relleno de la figura.
- *Style*: estilo de relleno de la figura. Esta subpropiedad tiene distintos valores, los cuales son:

Valor	Significado
bsDiagonal	Relleno de líneas diagonales a la izquierda.
bsClear	Sin relleno.
bsCross	Relleno a cuadrículas.
bsDiagCross	Relleno a cuadrículas diagonales.
bsFDiagonal	Relleno de líneas diagonales a la derecha.
bsSolid	Con relleno sólido.
bsHorizontal	Relleno de líneas horizontales.
bsVertical	Relleno de líneas verticales.

2.3 EL COMPONENTE BEVEL

Este componente únicamente da un efecto de 3D a nuestra aplicación. Tiene 2 propiedades principales, las cuales son:

- *Shape*: la cual marca el tipo de forma que tomará dicho objeto. Esta puede ser:

Valor	Significado
bsBox	Forma de rectángulo 3D (sobresale del formulario).
bsFrame	Forma un rectángulo incrustado en el formulario.
bsTopLine	Forma una línea horizontal superior incrustada.
bsBottomLine	Forma una línea horizontal inferior incrustada.
bsLeftLine	Forma una línea vertical izquierda incrustada.
bsRightLine	Forma una línea vertical derecha incrustada.

- *Style*: Que puede tomar 2 distintos valores:

Valor	Significado
bsLowered	Las formas aparecen incrustadas en el formulario.
bsRaised	Las formas aparecen sobresalidas del formulario.

2.4 GRAFICOS CON CANVAS

Canvas provee capacidades gráficas para controles gráficos y formularios. Brinda posibilidades para dibujar líneas, elipses, polígonos, textos y otros. Su manejo es simple y práctico.

Canvas tiene 5 subpropiedades importantes que son:

Propiedad	Descripción
Brush	Brocha, se utiliza para especificar el relleno de las figuras.
Font	Permite indicar las propiedades del texto gráfico.
Pen	Permite determinar el tipo, color, estilo, etc. De la línea.
PenPos	Punto actual del lápiz gráfico.
Pixels	Matriz con los puntos gráficos del componente.

Además, Canvas permite utilizar muchos métodos de dibujo sobre si mismo. Los más importantes son:

Método	Descripción	Ejemplo
Ellipse	Dibuja una elipse. Recibe como parámetros, las coordenadas del rectángulo que bordea la elipse.	Ellipse(10,20,100,150);
Arc	Dibuja una porción de elipse. Recibe como parámetros las coordenadas del rectángulo que bordea la elipse, y las coordenadas de los puntos iniciales y finales del segmento de la elipse.	Arc(10,20,100,150,10,20,200,200);
MoveTo	Define la nueva posición actual.	MoveTo(5,5);
LineTo	Dibuja una línea que va desde la posición actual hasta las coordenadas indicadas.	LineTo(100,100);
Rectangle	Dibuja un rectángulo, toma como parámetros las coordenadas de las esquinas superior izquierda e inferior derecha.	Rectangle(10,10,80,50);
TextOut	Imprime texto en las coordenadas indicadas.	TextOut(10,10,'Hola');

Ejemplo.- Utilizando Canvas

En un proyecto nuevo en blanco, se debe realizar la siguiente secuencia de pasos.

1. Se insertará un componente **Image** en el formulario.
2. Se modificarán las propiedades del objeto.
 - **Left:** 350
 - **Name:** *Imagen*
 - **Top:** 50
 - **Height:** 350
3. En el inspector de objetos, en la ficha *events*, se deberá seleccionar el evento **OnPaint** del objeto formulario.
4. En este, se deberá colocar el siguiente código:

```
PROCEDURE TFormulario.FormPaint(Sender:TObject);
VAR x:word;
BEGIN
  Imagen.canvas.pen.color := clBlue;    {Color de borde azul}
  Imagen.canvas.ellipse(120,12,220,220); {ellipse}
  Imagen.canvas.pen.color := clRed;      {Color de borde rojo}
  Imagen.canvas.rectangle(200,150,280,300); {Rectángulo}
  Imagen.canvas.pen.color := clPurple; {Color de borde púrpura}
  Imagen.canvas.brush.color := clPurple; {Color de relleno púrpura}
  Imagen.canvas.brush.style := bsFDiagonal; {estilo de relleno líneas diagonales}
  Imagen.canvas.ellipse(300,100,600,150); {Elipse}
  Imagen.canvas.pen.width := 3;    {Figura de borde ancho}
  Imagen.canvas.brush.color := clRed; {Color de relleno rojo}
  Imagen.canvas.brush.style := bsDiagCross; {diagonales cruzadas}
  Imagen.canvas.pen.color := clYellow; {Color de borde amarillo}
  Imagen.canvas.ellipse(10,200,150,300); {Elipse}
  Imagen.canvas.brush.style := bsSolid; {Relleno sólido}
  Imagen.canvas.pen.color := clGreen; {color de borde verde}
  Imagen.canvas.pen.width := 1;    {Borde fino}
```

FOR x:=1 **TO** 100 **DO**

BEGIN

*{línea que va de (x*10,10) a (300-x*10,80)}*

imagen.canvas.MoveTo (x*10,10);

imagen.canvas.LineTo (300-x*10,80);

END;

Imagen.canvas.brush.color := clNone; *{Sin color de relleno}*

imagen.canvas.Font.color := clBlack; *{Color de texto negro}*

imagen.canvas.Font.size := 16; *{Tamaño de texto 16}*

imagen.canvas.TextOut(100,300,'Hola esta es una demostración de Canvas');

END;

Comentario.- En este ejemplo, una porción de figura de la elipse púrpura quedará fuera del objeto *Image*, esta porción de figura no será mostrada.

Para utilizar la propiedad *Canvas* sobre el formulario, se utilizan las mismas instrucciones, a diferencia de que no se hace referencia a ningún objeto al principio de cada línea de código. Por ejemplo, se sustituiría la línea:

Imagen.canvas.ellipse(20,50,10,20)

Por la línea:

Canvas.ellipse(20,50,10,20)

3. BIBLIOGRAFIA

- Luis Joyanes Aguilar, Antonio Muñoz Clemente, BORLAND DELPHI Iniciacion y Referencia, McGraw-Hill/Interamericana de España S.A.U.
- Francisco Charte, PROGRAMACION CON DELPHI, Ediciones Anaya
- Victor A. Melchor Espinoza DELPHI 8 Editorial MACROPERU.
- Ayuda del programa (BORLAND DELPHI HELP)

Paginas de Internet:

- <http://www.elguille.info/delphi/apuntesDelphi/apuntesDelphi.htm>
- <http://www.programacionfacil.com>
- <http://www.mailxmail.com/curso/informatica/delphi>

TEXTO DE EJERCICIOS

CONTENIDO

TEXTO DE EJERCICIOS

1. PSEUDOCODIGO Y DIAGRAMAS DE FLUJO	182
PROBLEMA 1.	183
PROBLEMA 2.	184
PROBLEMA 3.	185
PROBLEMA 4.	186
2. DIAGRAMA DE FLUJO	188
PROBLEMA 5.	189
PROBLEMA 6.	190
PROBLEMA 7.	191
PROBLEMA 8.	192
PROBLEMA 9.	193
3. PROGRAMACIÓN CON DELPHI	197
PROBLEMA 11.	198
PROBLEMA 12.	201
PROBLEMA 13.	206
PROBLEMA 14.	208
PROBLEMA 15.	211
PROBLEMA 16.	215
PROBLEMA 17.	220
PROBLEMA 18.	223
PROBLEMA 19.	229
PROBLEMA 20.	232
ALTERNATIVA 2	235
EJERCICIOS APLICADOS A INGENIERIA CIVIL	244
APLICACION 1.	245
APLICACION CARRETERAS	245
APLICACION 2.	252
APLICACION HIDRAULICA	252

APLICACION 3. _____	262
APLICACION HORMIGON ARMADO _____	262
APLICACION 4. _____	270
APLICACION ROCAS _____	270
APLICACION 5. _____	279
APLICACION SUELOS (TALUDES) _____	279
EJERCICIOS PROPUESTOS _____	290
PROBLEMA 1. _____	291
PROBLEMA 2. _____	291
PROBLEMA 3. _____	291
PROBLEMA 4. _____	291
PROBLEMA 5. _____	291
PROBLEMA 6. _____	292
PROBLEMA 7. _____	292
PROBLEMA 8. _____	292
PROBLEMA 9. _____	292
ANEXOS _____	293
A1. INICIO DEL SISTEMA OPERATIVO _____	294
A1.1 ARRANQUE DEL SISTEMA _____	294
A1.2 BIOS _____	294
A1.3 EL SECTOR MAESTRO DE ARRANQUE Y EL SECTOR DE INICIO	294
A1.4 EL ARCHIVO NTLDR _____	295
A1.5 CARGA DE DRIVERS Y SERVICIOS _____	296
A1.6 INICIANDO WINDOWS _____	298
A1.7 DESPUÉS DE LA AUTENTIFICACIÓN _____	298
A1.8 OTRAS CLAVES UTILIZADAS DURANTE LA INICIALIZACIÓN DE WINDOWS _____	299
A1.9 DETECCIÓN DE DISPOSITIVOS PLUG AND PLAY _____	300
A2. DIAGRAMA DEL PROCESO DE INICIO _____	301
A3. MANTENIMIENTO DEL ORDENADOR _____	302
A3.1 MANTENIMIENTO DE HARDWARE _____	302

A3.2 MANTENIMIENTO DE SOFTWARE _____	302
A3.2.1 SCANDISK (COMPROBACIÓN DE ERRORES) _____	302
A3.2.2 DESFRAGMENTADOR DE DISCO _____	302
A3.2.3 LIBERADOR DE ESPACIO EN DISCO _____	303
A4. INSTALACIÓN DE PROGRAMAS _____	304
A5. RELACIÓN DE DELPHI CON OTROS PROGRAMAS _____	305
A5.1 DELPHI – AUTOCAD _____	305
A5.2 DELPHI – EXCEL _____	305
A5.3 EJEMPLOS _____	306
A6. LISTA DE COMPONENTES _____	310
BIBLIOGRAFÍA GENERAL _____	314

1. PSEUDOCODIGO Y DIAGRAMA DE FLUJO

En esta primera parte se plantean ejercicios con la finalidad de entrenar en el uso de pseudocódigo y diagrama de flujo.

Como se recomendó en el capítulo 2, la forma más fácil de elaborar un diagrama de flujo es realizar el pseudocódigo de la solución del problema planteado, y luego se tendrá una idea mas clara de los bloques a utilizarse, para elaborar el respectivo diagrama de flujo.

PROBLEMA 1.

Realizar un diagrama de flujo que permita ingresar tres números, e imprimir lo siguiente:
Suma y promedio de dichos números.

Solución:**1. Pseudocodigo**

Inicio del **Programa** Suma, Promedio

Leer (A,B,C) *//A,B,C Variables que almacenan 3 números*

Ejecutar $SUM=A+B+C$ *// Se almacena en la Variable SUM la suma*

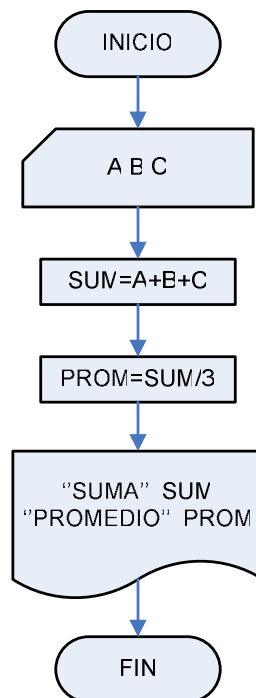
Ejecutar $PROM=SUM/3$ *// Se almacena en la Variable PROM el promedio*

Imprimir 'SUMA=' SUM *// Impresión de resultados*

Imprimir 'PROMEDIO=' PROM

Fin del **Programa**

Una vez realizado el pseudocodigo se tiene una idea mas clara de los símbolos a utilizarse para realizar el diagrama de flujo (ver capitulo 2 Procesos Lógicos).

2. Diagrama de Flujo

PROBLEMA 2.

Realizar un diagrama de flujo que permita leer las notas del primer y segundo parcial y determinar si la nota promedio es de aprobado o reprobado.

Solución:**Pseudocodigo**

Inicio del **Programa** Notas Parciales

Leer (Nota1, Nota2) // *Nota1, Nota2 Variables que almacenan 2 notas*

Ejecutar $PROM = (Nota1 + Nota2) / 2$ // *Se calcula el promedio de notas*

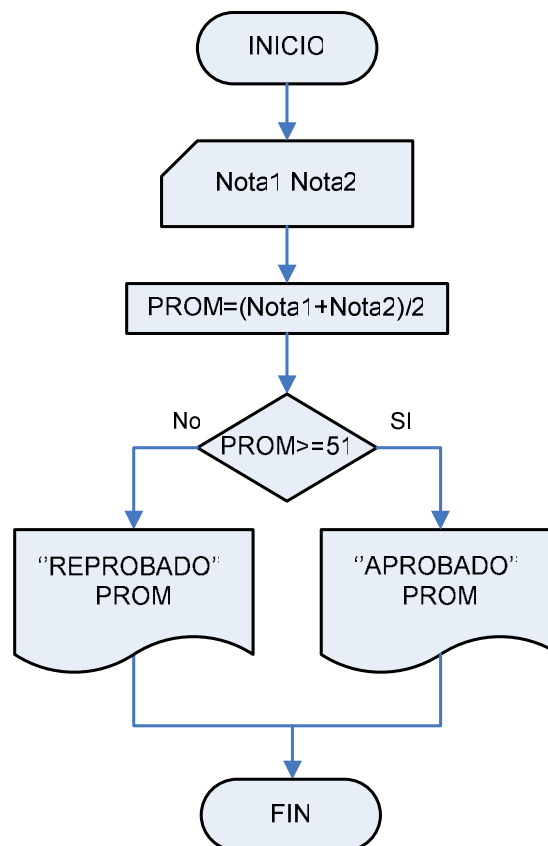
Si $(PROM \geq 51)$ Entonces // *Se analiza la condición*

Imprimir 'Aprobado' PROM // *Impresión de resultados*

Sino Imprimir 'Reprobado' PROM

Fin del Si

Fin del Programa

Diagrama de Flujo

PROBLEMA 3.

Realizar un diagrama de flujo que permita leer los lados de un triángulo y determinar si el triángulo es: Equilátero, Isósceles, o Escaleno.

Solución:**Pseudocódigo**

Inicio del **Programa** Triangulo

Leer (L1, L2, L3) // *L1,L2,L3 Variables que almacenan 3 lados del triangulo*

Si (L1=L2 AND L2=L3) Entonces // *Se analiza la condición*

 Imprimir 'Triangulo Equilatero'

 Sino Si (L1=L2 OR L1=L3 OR L2=L3) Entonces

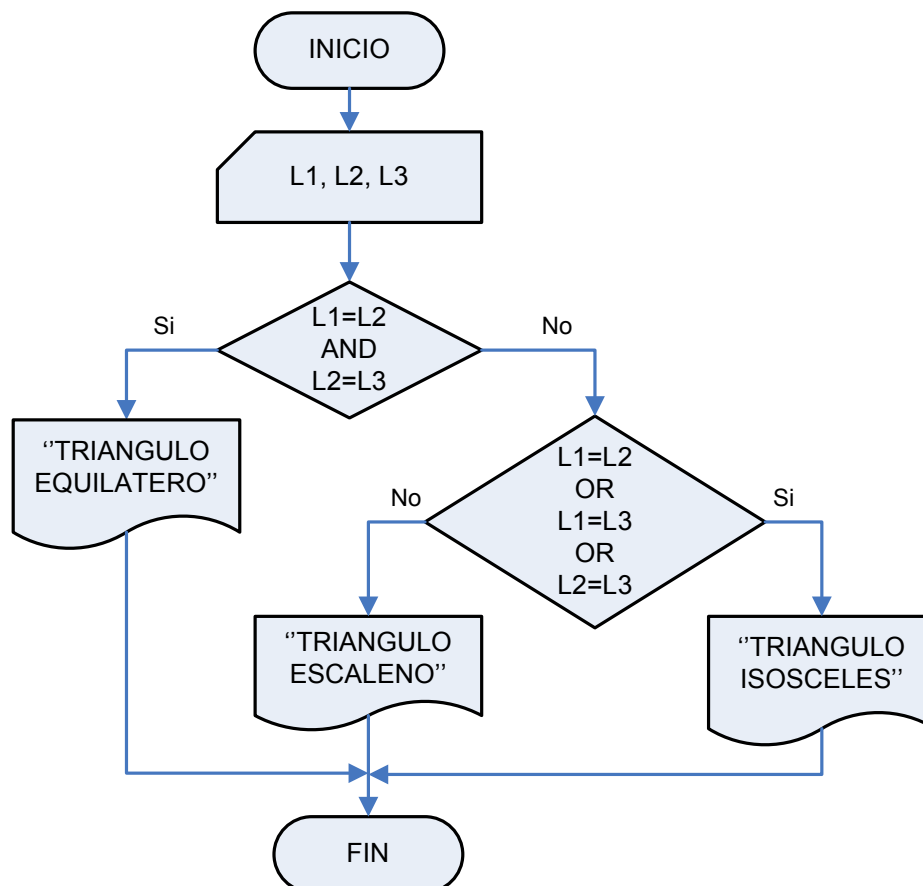
 Imprimir 'Triangulo Isosceles'

 Sino Imprimir 'Triangulo Escaleno'

 Fin del Si

Fin del Si

Fin del Programa



PROBLEMA 4.

Realizar un diagrama de flujo que permita leer un conjunto de notas, cada nota corresponde a un alumno. Se pide obtener lo siguiente:

- Imprimir la cantidad de alumnos aprobados y reprobados.
- Imprimir la mayor y menor nota.
- Imprimir el promedio general de notas.

El Programa debe finalizar cuando el valor de la nota ingresada es negativa.

Solución:

Inicio del **Programa** Notas Alumnos

Ejecutar CAP=0, CRP=0, SUM=0, C=0, MAY=0, MEN=100

Leer (NOTA)

Si $NOTA \geq 0$ Entonces

Si $NOTA \geq 51$ Entonces

Ejecutar CAP=CAP+1

Sino Ejecutar CRP=CRP+1

Fin del Si

Si ($NOTA > MAY$) Entonces

Ejecutar MAY=NOTA

Fin del Si

Si ($NOTA < MEN$)

Ejecutar MEN=NOTA

Fin del Si

Ejecutar SN=SN+SUM

Ejecutar C=C+1

Sino Ejecutar PROM=SUM/C

Imprimir 'Cantidad de Aprobados', CAP

Imprimir 'Cantidad de Reprobados'. CRP

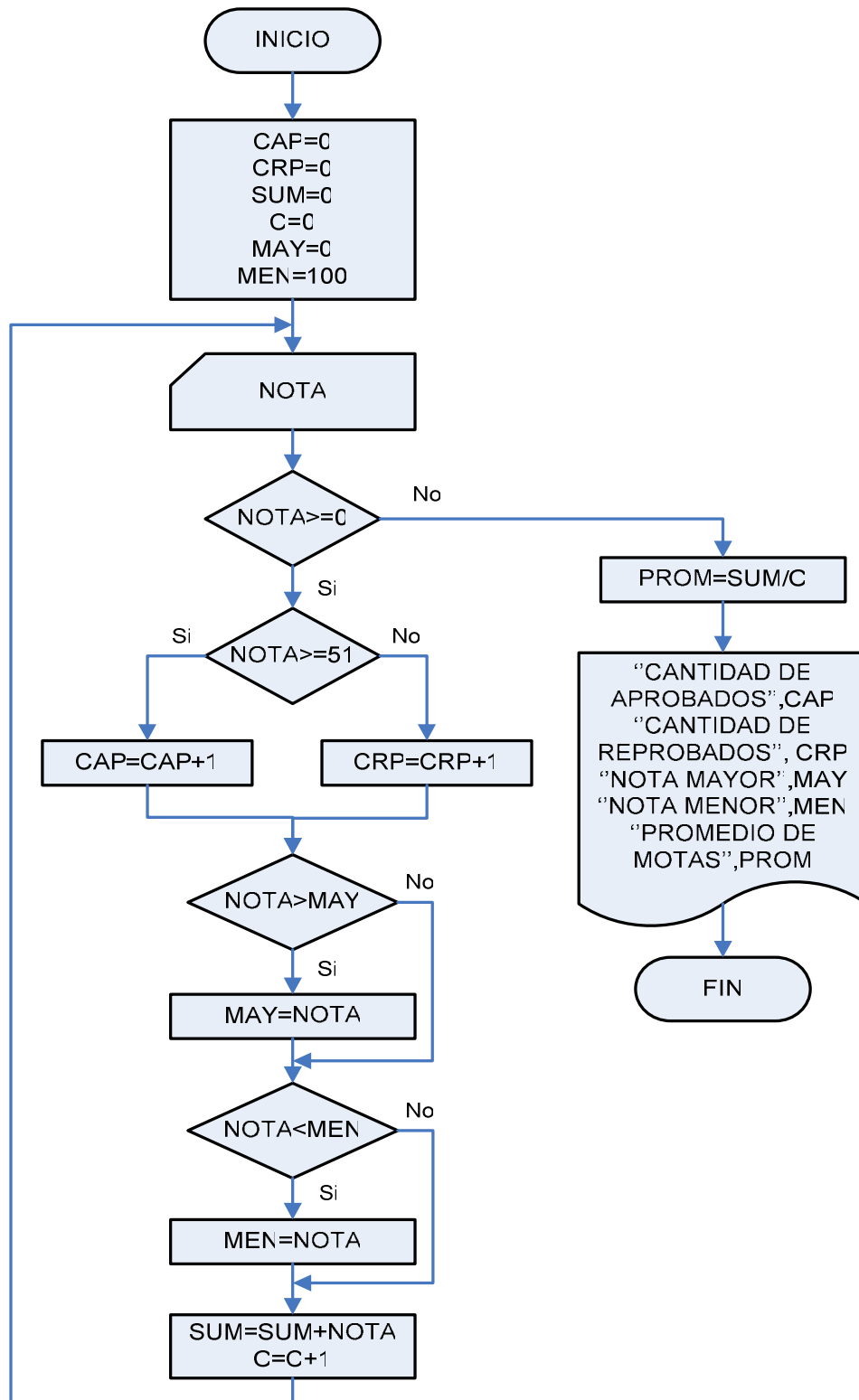
Imprimir 'Nota Mayor', MAY

Imprimir 'Nota Menor' MEN

Imprimir 'Promedio de Notas', PROM

Fin del Si

Fin del Programa

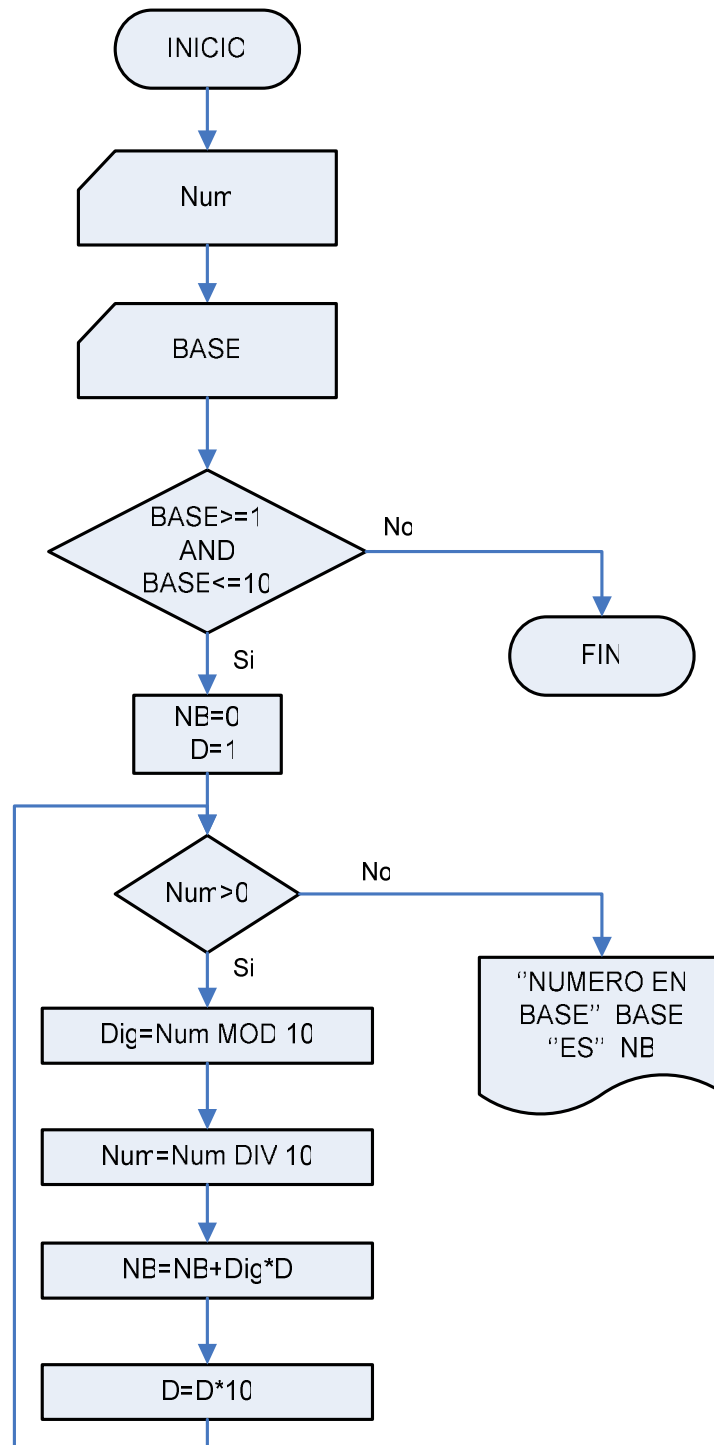


2. DIAGRAMA DE FLUJO

Continuando con los ejercicios en esta segunda parte, se plantean ejercicios con la finalidad de entrenar en el uso y elaboración de diagramas de flujos. Ingresando a esta sección se tendrá ya experiencia en la elaboración de diagramas de flujo sencillos y se presentan ejercicios planteados un poco más complicados para continuar con el entrenamiento para empezar un programa en el lenguaje elegido..

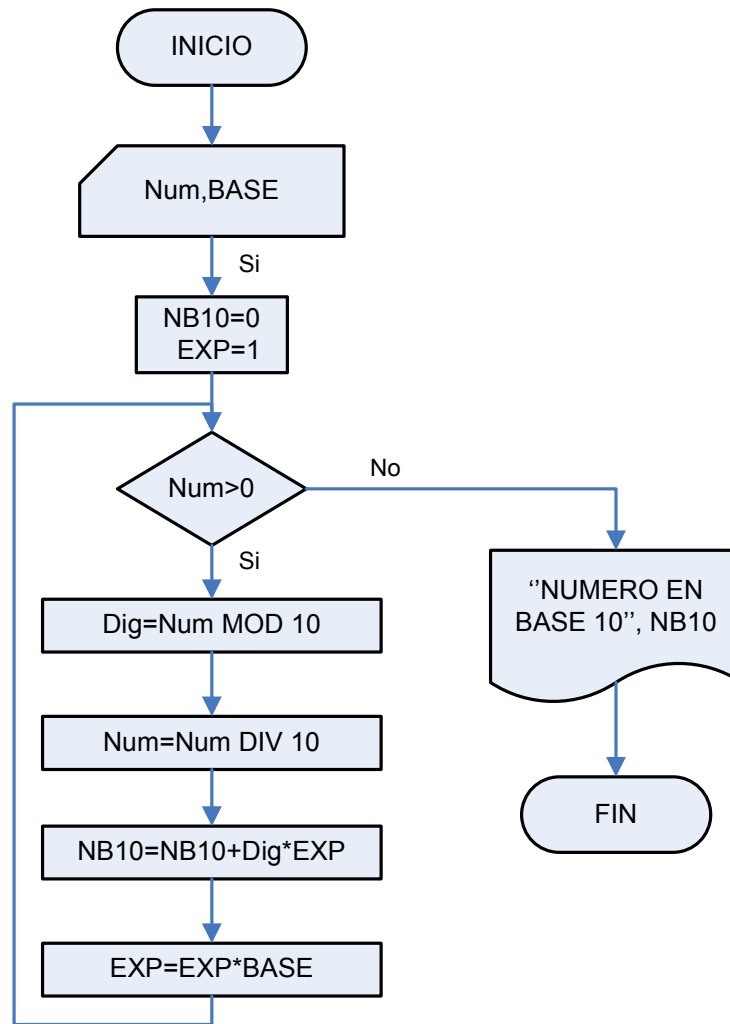
PROBLEMA 5.

Realizar un diagrama de flujo que permita leer un número en base 10 y convertir a cualquier base menor de 10.

Solución:

PROBLEMA 6.

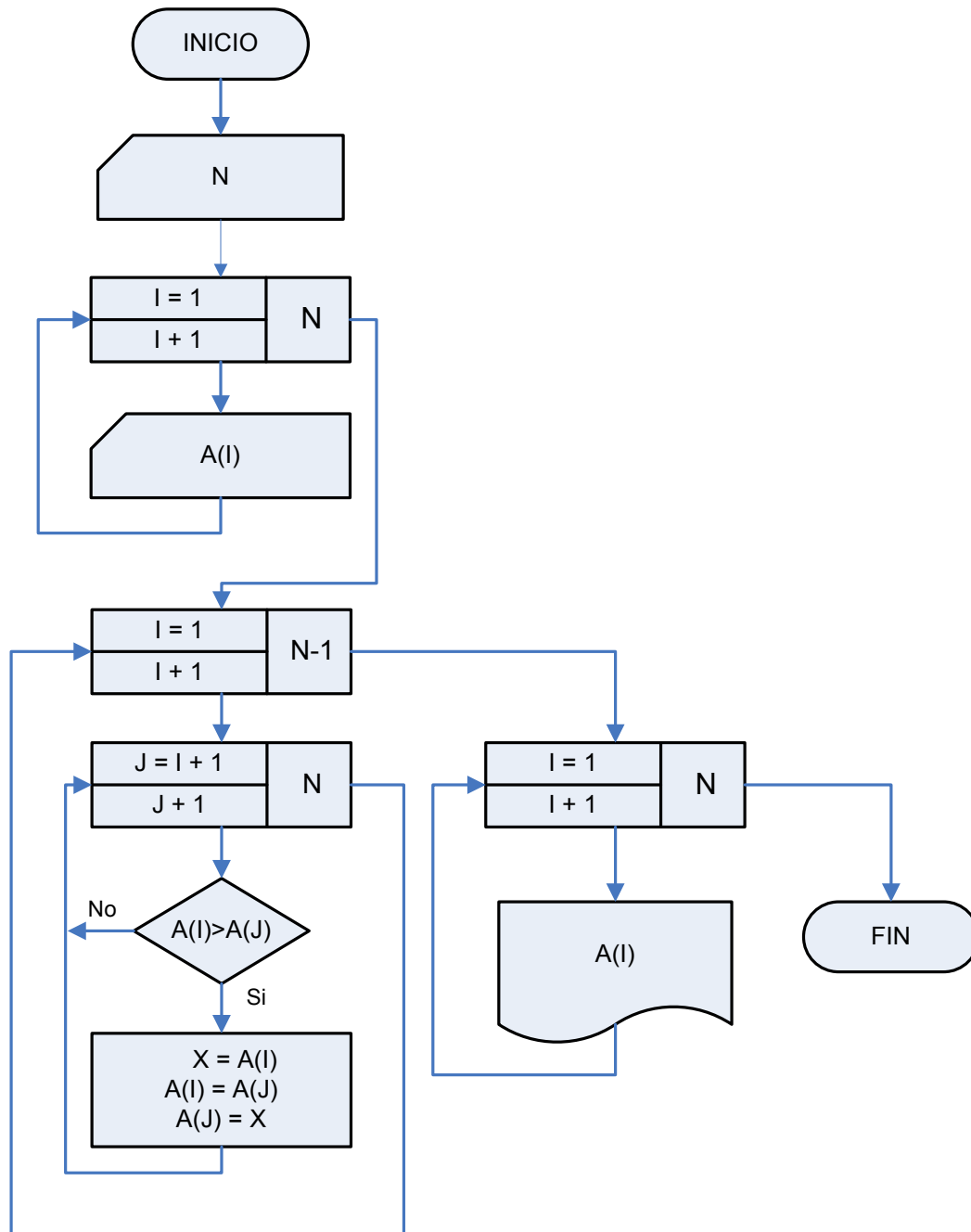
Realizar un diagrama de flujo que permita leer un número en cualquier base menor a 10 y convertir el número a base 10.

Solución:

*** Tarea: Realizar la traducción de los diagramas de flujo a pseudocódigo correspondientes a esta sección.**

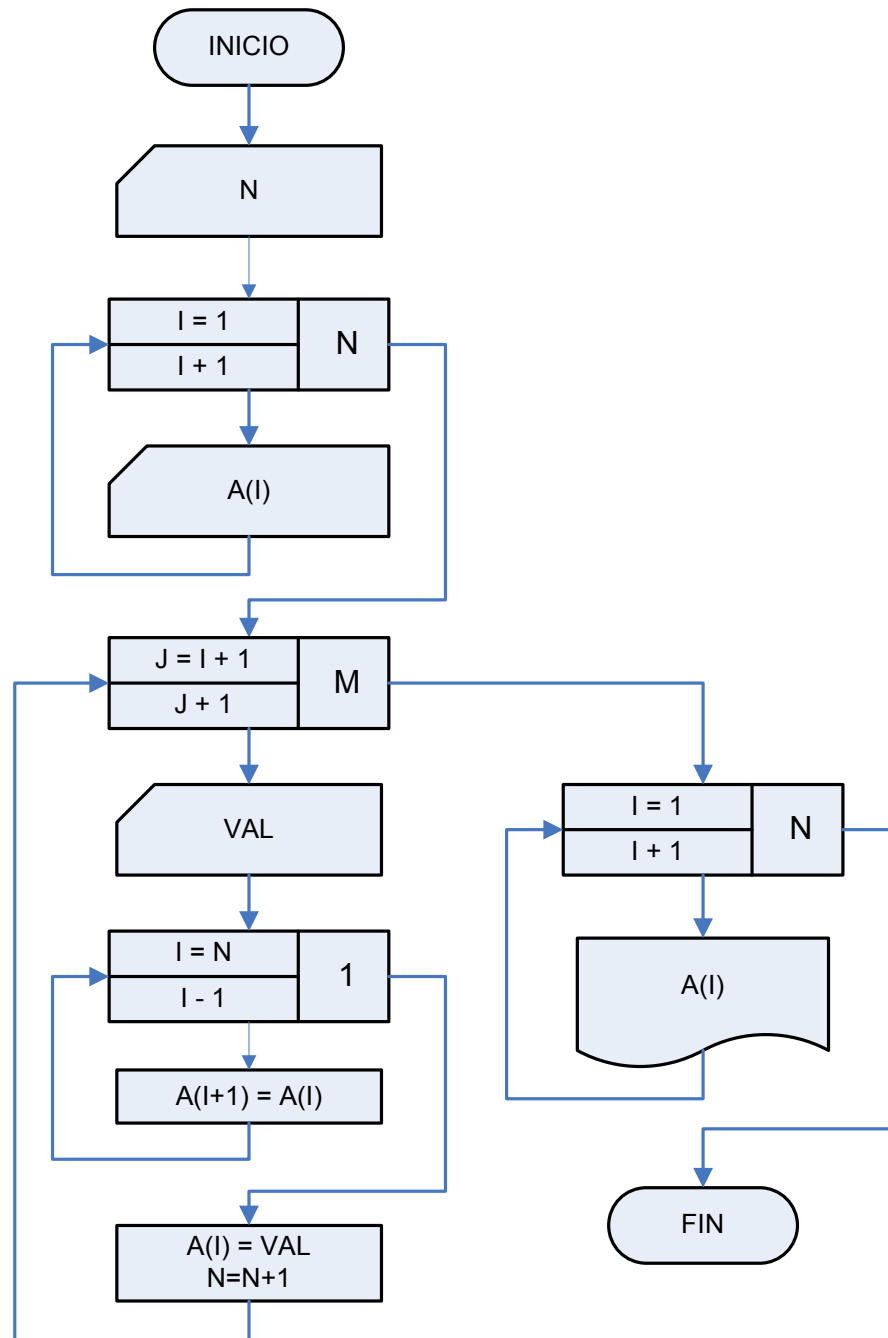
PROBLEMA 7.

Realizar un diagrama de flujo que permita leer un vector de N elementos, ordenar en forma ascendente e imprimir el vector resultante.

Solución:

PROBLEMA 8.

Realizar un diagrama de flujo que permita leer un vector de N elementos, e insertar M valores al comienzo del vector.

Solución:

PROBLEMA 9.

Dados dos arreglos unidimensionales A(I) con $I=1..N$ y B(J) con $J=1..M$. sin repetición interna de sus elementos , se pide obtener dos arreglos C(K) y D(L).

Donde: C(K) contendrá los elementos que se repiten en A y B

D(L) contendrá los elementos que no se repiten en A y B

Ejemplo.

Arreglos Unidimensionales A y B

$$A(I) = \begin{array}{|c|c|c|c|c|} \hline 10 & 15 & 3 & 20 & 8 \\ \hline \end{array}$$

1 N

$$B(J) = \begin{array}{|c|c|c|c|c|} \hline 3 & 1 & 20 & 7 & 10 \\ \hline \end{array}$$

1 M

Resultado y obtención de los arreglos C y K

$$C(K) = \begin{array}{|c|c|c|} \hline 10 & 3 & 20 \\ \hline \end{array}$$

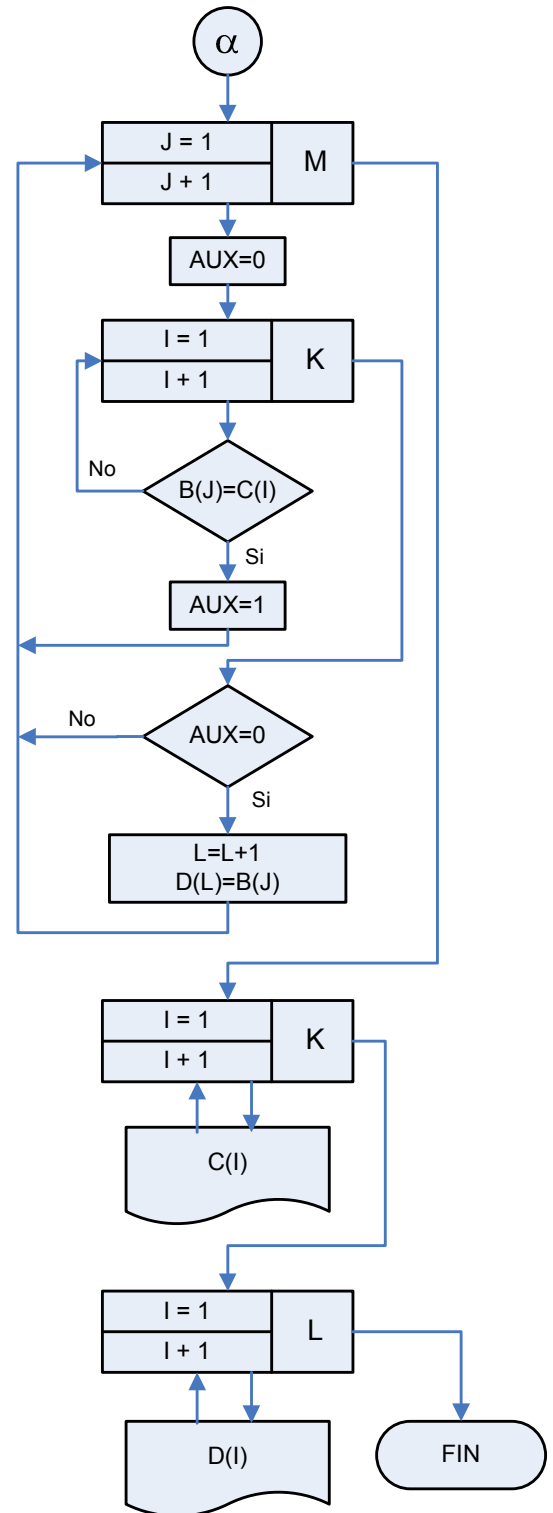
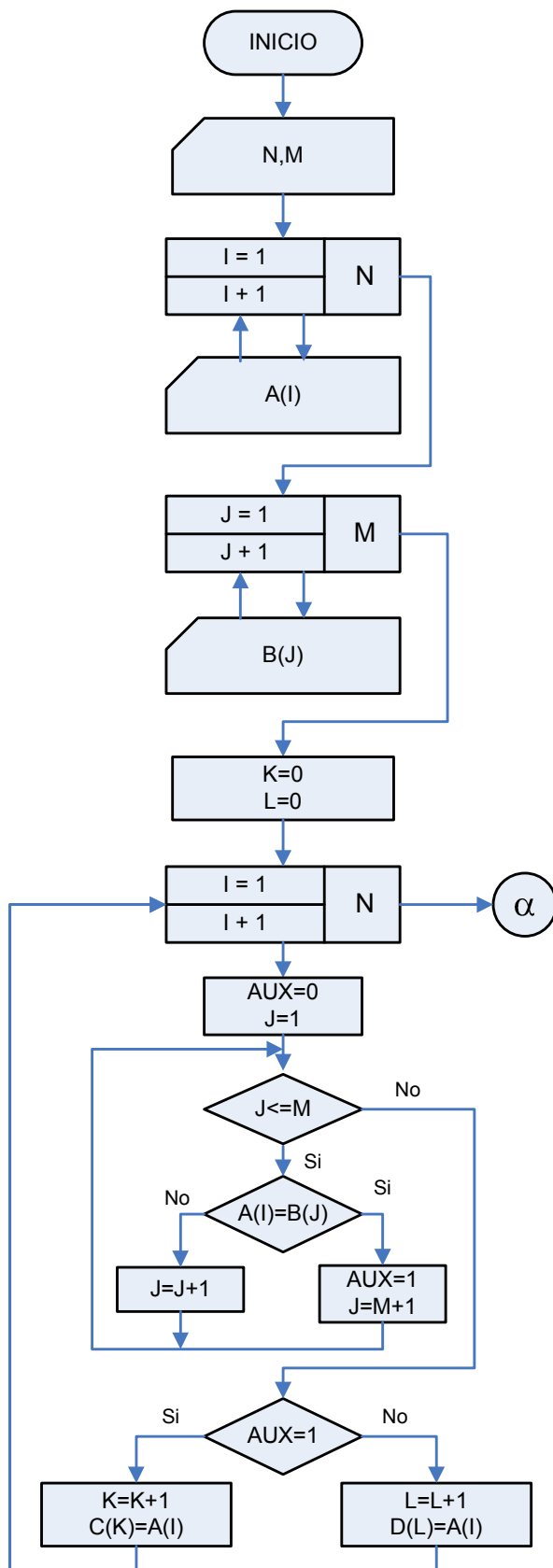
1 K Elementos repetidos en A y B

$$D(L) = \begin{array}{|c|c|c|c|} \hline 15 & 8 & 1 & 7 \\ \hline \end{array}$$

1 L Elementos no repetidos en A y B

Solución:

Diagrama de Flujo



PROBLEMA 10.

Dado una matriz de N por M, realizar un diagrama de flujo de tal forma que mueva las columnas de dicha matriz de manera tal que los elementos de la última fila queden ordenados en forma decreciente.

Ejemplo.

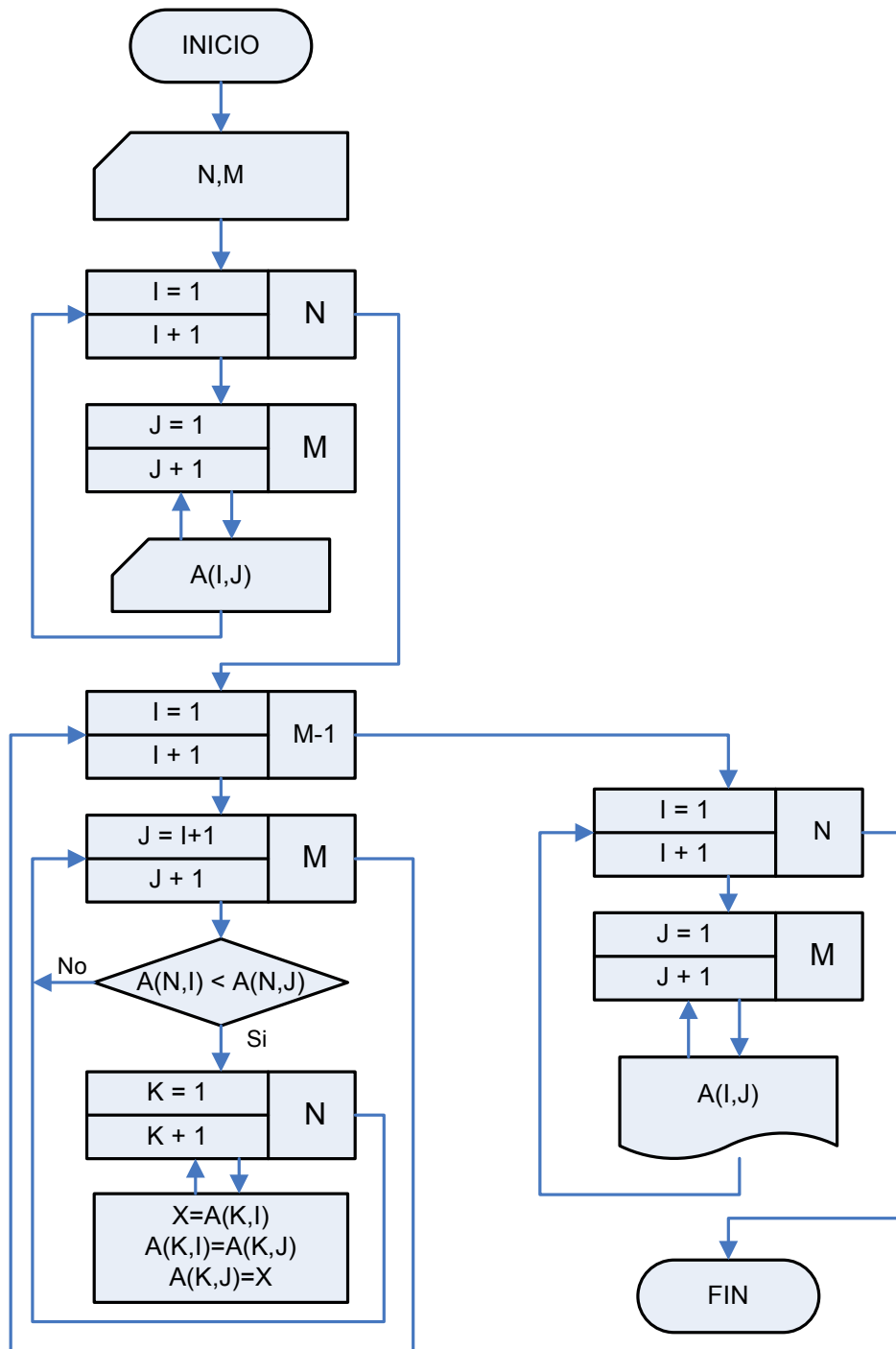
Matriz Inicial

Matriz Resultante

$$\begin{pmatrix} 2 & 70 & 8 & 4 \\ 10 & 7 & 1 & 7 \\ 3 & 1 & 15 & 5 \end{pmatrix} \Rightarrow \begin{pmatrix} 8 & 4 & 2 & 70 \\ 1 & 7 & 10 & 7 \\ 15 & 5 & 3 & 1 \end{pmatrix}$$

Solución:

El problema se resuelve como el ordenamiento de un arreglo, partiendo con el primer elemento de la última fila y realizando una comparación con el elemento siguiente hasta llegar al último elemento. La única diferencia sucede en la comparación del elemento con el elemento siguiente, si es menor entonces se debe intercambiar todos los elementos de ambas columnas.



3. PROGRAMACIÓN CON DELPHI

Teniendo ya experiencia en la elaboración de pseudocódigo y diagramas de flujo, lo cual ayuda en la elaboración de aplicaciones en Delphi.

En esta sección se presentan ejercicios planteados para su codificación en Delphi. Teniendo en cuenta que en Delphi la secuencia de pasos para elaborar aplicaciones (programas) son los siguientes:

- Se tiene la solución del problema planteado en pseudocódigo ó en diagrama de flujo.
- Para la elaboración de la aplicación es muy importante el aspecto visual (diseño del formulario), que datos se desea presentar: datos de entrada, resultados.
Utilizando los componentes que posee Delphi en su entorno de desarrollo diseñar el interfaz gráfico.
- Luego escribir el algoritmo de solución en el lenguaje que utiliza Delphi (Object Pascal)

PROBLEMA 11.

El enunciado es el mismo del problema 5 y 6, el programa resuelve la transformación de un número en base 10 a cualquier base menor que 10 y viceversa.

Solución:

Para ello se necesitaran:

- 1 Formulario (Form)
- 2 Botones de opción (TRadioButton)
- 2 Cajas de texto (TEdit)
- 1 Caja combinada (TComboBox)
- 1 Boton (TBitBtn)
- 3 Etiquetas (TLabel)
- 3 Cajas agrupadoras (TGroupBox)
- Tal como se muestra en la figura 1

Modificar:

En la propiedad **Caption** del Form escribir: PROGRAMA CAMBIO DE BASE

En la propiedad **Caption** de TGroupBox BASE DEL NUMERO, ENTRADA DE DATOS, RESULTADO, para cada caja agrupadora respectivamente. También llenar en la propiedad **Caption** de cada objeto (Tlabel) tal como se muestra en la figura1.

Elegir **bkOK** en la propiedad **Kind** del TBitBtn , escribir &CONVERTIR en **Caption**

Escribir 1,2,3,4,5,6,7,8,9 en la propiedad **Ítems** del TComboBox .

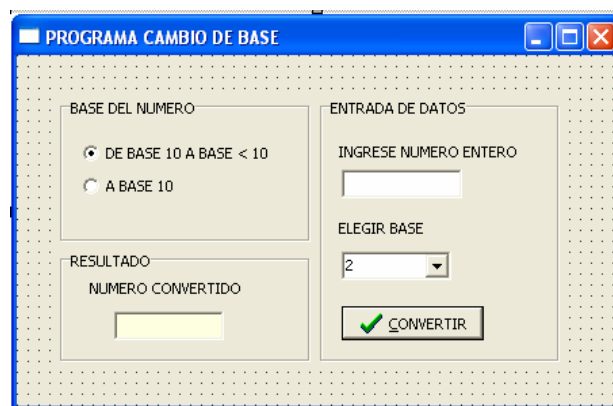


Fig. 1

Una vez diseñado el formulario, escribir el código del programa dentro del evento del boton (Bitbtn).

UNIT ucambase;

INTERFACE

USES

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, Borland.Vcl.StdCtrls, Borland.Vcl.Buttons, Borland.Vcl.ExtCtrls,
System.ComponentModel;

TYPE

```
Tform1 = Class(Tform)
  Groupbox1: Tgroupbox;
  Cb1: Tcombobox;
  Label2: TLabel;
  Label3: TLabel;
  Radiogroup1: Tradiogroup;
  Op1: Tradiobutton;
  Op2: Tradiobutton;
  Ed1: Tedit;
  Groupbox2: Tgroupbox;
  Label4: TLabel;
  Ed2: Tedit;
  Bt: Tbitbtn;
```

PROCEDURE btclick(sender: tobject);

private

{ **private** declarations }

public

{ **public** declarations }

END;

VAR

Form1: Tform1;

IMPLEMENTATION

{ \$r *.nfm }

*** Tarea: Realizar la verificación de los programas de esta seccion, copiando el codigo en un nuevo proyecto de Delphi. Comente en clase de los errores en la compilación del programa.**

PROCEDURE Tform1.btclick(sender: tobject);

VAR

num,base,dig,nb,d,exp:integer;

BEGIN

num:=**STRTOINT**(ed1.text);

base:=**STRTOINT**(cb1.text);

IF op1.checked=true **THEN**

BEGIN

nb:=0; d:=1;

WHILE num >0 **DO**

BEGIN

dig:=num mod base;

num:=num div base;

nb:=nb+dig*d;

d:=d*10;

END;

END

ELSE

BEGIN

exp:=1;

nb:=0;

WHILE num >0 **DO**

BEGIN

dig:=num mod 10;

num:=num div 10;

nb:=nb+dig*exp;

exp:=exp*base;

END;

END;

Ed2.text:=inttostr(nb);

END;

END. *// FIN DEL PROGRAMA*

PROBLEMA 12.

Ver enunciado del problema 7

Solución:

Program ordenar_vector;

USES

Dialogs, Sysutils;

// DECLARACION DE CONSTANTES, ARREGLOS Y VARIABLES

CONST max=10;

TYPE

Vec = **ARRAY**[1..max] **OF** real;

VAR

A:vec;

I,j,n:integer;

X:real;

S:string;

BEGIN

// LECTURA DEL TAMAÑO DEL VECTOR

n:=**STRTOINT**(**inputbox**('ordenar vector','ingrese tamaño del vector',' '));

// LECTURA DE VALORES DEL VECTOR

FOR i:=1 **TO** n **DO**

a[i]:=**STRTOFLOAT**(**inputbox**('a'+[''+inttostr(i)+''],'ingrese valor',' '));

// ORDENAMIENTO DEL VECTOR

FOR i:=1 **TO** n-1 **DO**

BEGIN

FOR j:=i+1 **TO** n **DO**

BEGIN

IF a[i]>a[j] **THEN**

BEGIN

x:=a[i];

a[i]:=a[j];

a[j]:=x;

END;

END;

END;

// VECTOR ORDENADO EN PANTALLA

FOR i:=1 **TO** n **DO**

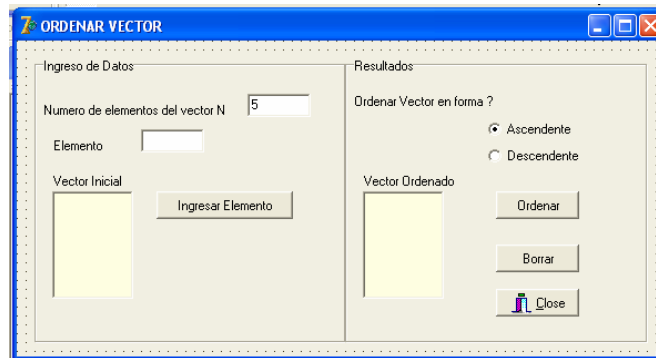
s:=s+**FLOATTOSTR**(a[i])+#9;

showmessage('el vector ordenado es:'+#13+[''+#9+ s+']);

END. *// FIN DEL PROGRAMA*

Alternativa2

El código siguiente ordena el vector en forma ascendente y descendente utilizando el interfaz grafico siguiente:



UNIT uOrden;

INTERFACE

USES

Windows, Messages, SysUtils, **Variants**, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, Buttons, XPMAN, OleCtrls;

TYPE

vec=**ARRAY**[1..100]**OF** integer;

TForm1 = class(TForm)

 GroupBox1: TGroupBox;

 Label1: TLabel;

 ListBox1: TListBox;

 BitBtn1: TBitBtn;

 Label2: TLabel;

 Edit1: TEdit;

 Edit2: TEdit;

 GroupBox2: TGroupBox;

 Label3: TLabel;

 Label4: TLabel;

 ListBox2: TListBox;

 RadioButton1: TRadioButton;

 RadioButton2: TRadioButton;

 Label5: TLabel;

 BitBtn2: TBitBtn;

 XPManifest1: TXPManifest;

 BitBtn3: TBitBtn;

 BitBtn4: TBitBtn;

PROCEDURE BitBtn1Click(Sender: TObject);

PROCEDURE BitBtn2Click(Sender: TObject);

PROCEDURE Edit2KeyPress(Sender: TObject; **VAR** Key: Char);

PROCEDURE Edit1Change(Sender: TObject);

PROCEDURE BitBtn4Click(Sender: TObject);

private

 { Private declarations }

public

 { Public declarations }

END;

VAR

Form1: TForm1;

a:vec;

i,n:integer;

IMPLEMENTATION

```
{ $R *.dfm }  
PROCEDURE mostrar;  
VAR  
i:integer;  
BEGIN  
  form1.ListBox2.Clear;  
  FOR i:=1 TO n DO  
    form1.ListBox2.Items.Add(inttostr(a[i]));  
END;  
  
PROCEDURE ascendente;  
VAR i,j,x:integer;  
BEGIN  
  // ORDENANDO EL VECTOR EN FORMA ASCENDENTE  
  FOR i:=1 TO n-1 DO  
    BEGIN  
      FOR j:=i+1 TO N DO  
        BEGIN  
          IF a[i]>a[j] THEN  
            BEGIN  
              x:=a[i];  
              a[i]:=a[j];  
              a[j]:=x;  
            END;  
          END;  
        END;  
      END;  
    END;  
  END;  
  
PROCEDURE descendente;  
VAR i,j,x:integer;  
BEGIN  
  // ORDENADO EL VECTOR EN FORMA DESCENDENTE  
  FOR i:=n downto 2 DO  
    BEGIN  
      FOR j:=i-1 downto 1 DO  
        BEGIN  
          IF a[i]>a[j] THEN  
            BEGIN  
              x:=a[i];  
              a[i]:=a[j];  
              a[j]:=x;  
            END;  
          END;  
        END;  
      END;  
    END;  
  END;  
  
END;
```



```
PROCEDURE TForm1.BitBtn1Click(Sender: TObject);  
BEGIN
```

```
  i:=i+1;
```

```
  n:=STRTOINT(edit1.text);
```

```
  IF listbox1.Count<n THEN
```

```
    BEGIN
```

```
      label1.Caption:='Elemento N° '+inttostr(i+1);
```

```
      listbox1.Items.Add(edit2.Text);
```

```
      a[i]:=STRTOINT(edit2.Text);
```

```
    END;
```

```
    edit2.SetFocus;
```

```
    edit2.Clear;
```

```
  END;
```

```
PROCEDURE TForm1.BitBtn2Click(Sender: TObject);
```

```
BEGIN
```

```
IF radiobutton1.Checked=true THEN
```

```
  ascendente      // LLAMADA AL PROCEDIMIENTO
```

```
ELSE
```

```
  descendente;    // LLAMADA AL PROCEDIMIENTO
```

```
END;
```

```
PROCEDURE TForm1.Edit2KeyPress(Sender: TObject; VAR Key: Char);
```

```
BEGIN
```

```
IF NOT (key IN ['0'..'9',#8 {backspace}]) THEN
```

```
  BEGIN
```

```
    key:=#0;
```

```
    beep;
```

```
  END;
```

```
END;
```

```
PROCEDURE TForm1.Edit1Change(Sender: TObject);
```

```
BEGIN
```

```
  listbox1.Clear;
```

```
  listbox2.Clear;
```

```
  i:=0;
```

```
  label1.Caption:='Elemento N° 1';
```

```
END;
```

```
PROCEDURE TForm1.BitBtn4Click(Sender: TObject);
```

```
BEGIN
```

```
  listbox1.Clear;
```

```
  listbox2.Clear;
```

```
  edit1.Clear;
```

```
END;
```

```
END.
```

PROBLEMA 13.

Ver enunciado del **problema 9**

Solución:

Program vec_igua_dif;

USES

Dialogs, Sysutils;

// DECLARACION DE CONSTANTES, ARREGLOS Y VARIABLES

CONST max=10;

TYPE

Vec =**ARRAY**[1..max] **OF** real;

VAR

A,b,c,d:vec;

K,l,n,m:integer;

// PROCEDIMIENTO PARA LA LECTURA DE LOS VECTORES

PROCEDURE leer (m:string;**VAR** arr:vec;fila:integer);

VAR

i :integer;

BEGIN

FOR i :=1 **TO** fila **DO**

arr[i]:=**STRTOFLOAT**(**inputbox**(m+'['+inttostr(i)+']','ingrese valor'+#13+'(debe ser un numero real)',' '));

END; (* fin de leer *)

// PROCEDIMIENTO PARA MOSTRAR LOS VECTORES C Y D

PROCEDURE mostrar(m:string;a:vec;t:integer);

VAR

l:integer;

S:string;

BEGIN

FOR i:=1 **TO** t **DO**

s:=s+**FLOATTOSTR**(a[i])+#9;

showmessage(m+#13+'['+#9+ s+']');

END;

{ PROCEDIMIENTO PARA DETERMINAR:

VECTORES CON ELEMENTOS IGUALES Y DIFERENTES }

PROCEDURE calcular (**VAR** a,b,c,d:vec;n,m:integer);

VAR

i,j,aux :integer;

LABEL

//DECLARACION DE ETIQUETAS

E1;

```

BEGIN
  k:=0;l:=0;
  FOR i:=1 TO n DO
    BEGIN
      aux:=0; j:=1;
      WHILE j<=m DO
        BEGIN
          IF a[i]=b[j] THEN
            BEGIN
              aux:=1;j:=m+1;
            END
          ELSE
            j:=j+1;
          END;
          IF aux=1 THEN
            BEGIN
              k:=k+1;c[k]:=a[i];
            END
          ELSE
            BEGIN
              l:=l+1; d[l]:=a[i];
            END;
          END;
        END;
      FOR j:=1 TO m DO
        BEGIN
          aux:=0;
          FOR i:=1 TO k DO
            BEGIN
              IF b[j]=c[i] THEN BEGIN aux:=1;Goto E1;END
            END;
          IF aux=0 THEN
            BEGIN
              l:=l+1; d[l]:=b[j];
            END; E1:
          END;
        END;
      mostrar('vector con elementos iguales c( ): ',c,k);
      mostrar('vector con elementos diferentes d( ): ',d,l);
    END;
  BEGIN    (* BLOQUE PRINCIPAL *)
    n:=STRTOINT(inputbox('vector a','ingrese tamaño del vector a','4'));
    m:=STRTOINT(inputbox('vector b','ingrese tamaño del vector b','5'));

    // LLAMADA A PROCEDIMIENTOS
    leer('a',a,n);
    leer('b',b,m);
    calcular(a,b,c,d,n,m);
  END.    // FIN DEL PROGRAMA

```

PROBLEMA 14.

Dado un arreglo unidimensional A[I], donde $I=1..N$. Realizar el programa en lenguaje Delphi que construya otro arreglo B[I] donde $I=1..2N$. Además debe presentar las siguientes características:

- En los lugares pares del arreglo B [I] se colocar los elementos de arreglo A[I] ordenado en forma creciente.
- En los lugares impares del arreglo B [I] se colocar los elementos de arreglo A[I] ordenado en forma decreciente.

Ejemplo:

VECTOR INICIAL

A(I) =	20	15	26	3	1	6
	1	2	3	4	5	6
						N

VECTOR ORDENADO

A(I) =	1	3	6	15	20	26
	1	2	3	4	5	6
						N

VECTOR RESULTANTE

B(J) =	26	1	20	3	15	6	6	15	3	20	1	26
	1	2	3	4	5	6	7	8	9	10	11	12
												2N

Solución:

Program doble_vec;

USES

Dialogs, Sysutils;

// DECLARACION DE CONSTANTES, ARREGLOS Y VARIABLES

CONST max=100;

TYPE

Vec =**ARRAY**[1..max] **OF** real;

VAR *//VARIABLES GLOBALES*

A,b:vec;

N:integer;

// LECTURA DE VALORES DE LOS VECTORES

PROCEDURE leer (m:string;**VAR** arr:vec;fila:integer);

VAR *//VARIABLES LOCALES*

i :integer;

BEGIN

FOR i :=1 **TO** fila **DO**

arr[i]:=**STRTOFLOAT**(**inputbox**(m+'['+inttostr(i)+']','ingrese valor'+#13+'(debe ser un numero real)',' '));

END; (** FIN DE LEER **)

// PROCEDIMIENTO PARA MOSTRAR LOS VECTORES C Y D

PROCEDURE mostrar(m:string;a:vec;t:integer);

VAR

l:integer;

S:string;

BEGIN

FOR i:=1 **TO** t **DO**

s:=s+**FLOATTOSTR**(a[i])+#9;

showmessage(m+#13+'['+#9+ s+']');

END;

// PROCEDIMIENTO PARA DETERMINAR EL VECTOR DOBLE

PROCEDURE calcular (**VAR** a:vec;n:integer);

VAR

i,j,k :integer;

BEGIN

k:=2*n+1;j:=0;

FOR i:=1 **TO** n **DO**

BEGIN

j:=j+2;

b[j]:=a[i];

k:=k-2;

```

        b[k]:=a[i];
    END;
    mostrar('vector inicial a( ):',a,n);
    mostrar('vector resultante 2a( ):',b,2*n);
END;

```

//PROCEDIMIENTO PARA ORDENAR EL VECTOR EN FORMA ASCENDENTE

PROCEDURE ordenar(a:vec;n:integer);

VAR

I,j:integer;

X:real;

BEGIN

FOR i:=1 TO n-1 DO

BEGIN

FOR j:=i+1 TO n DO

BEGIN

IF a[i]>a[j] THEN

BEGIN

x:=a[i];

a[i]:=a[j];

a[j]:=x;

END;

END;

END;

calcular(a,n);

END;

BEGIN *// BLOQUE PRINCIPAL*

n:=STRTOINT(inputbox('vector a','ingrese tamaño del vector a','4'));

//LLAMADA A PROCEDIMIENTOS

leer('a',a,n);

ordenar(a,n);

END. *// FIN DEL PROGRAMA*

PROBLEMA 15.

Dado una matriz de orden impar realizar un programa que la suma de los elementos de las filas, columnas y diagonales de la matriz sea común.

Solución:

Para ello preparar el siguiente entorno gráfico ver fig.

The screenshot shows a window titled "CUADRADO MAGICO" with a blue border and standard Windows window controls. The main area has a light gray dotted background. On the left, a yellow text box contains the following text:

Cuadrado Magico de orden N contiene todos los enteros desde 1 hasta $N \times N$ asignados a una matriz por lo tanto la suma de los enteros en cada fila, columna y diagonales son iguales.

El programa construye un cuadrado Magico de orden impar {1,3,5,7, etc.}. Se utiliza un algoritmo conocido hace 500 años y dice:

"Empezar con 1 en el medio de la fila superior, luego ir arriba y a la izquierda asignado numeros orden creciente a las celdas vacias; si se completo la fila superior, realizar el mismo procedimiento, si la celda esta ocupada mover abajo y continuar"

To the right of the text box is a 5x5 grid with a blue square in the top-left cell (row 1, column 1).

Below the text box, there is a label "INGRESE NUMERO (<=51)" followed by a text input field containing the number "5".

Below the input field is a button with a green checkmark icon and the text "CALCULAR".

At the bottom, there is a label "SumLb1" followed by a large, empty rectangular area for displaying the result.

UNIT uCuadrado_Magico;

INTERFACE

USES

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, Buttons, Grids;

TYPE

```
TForm1 = class(TForm)
  Label1: TLabel;
  StringGrid1: TStringGrid;
  CalcBtn: TBitBtn;
  IntEdit1: TEdit;
  Memo1: TMemo;
  SumLbl: TLabel;
PROCEDURE CalcBtnClick(Sender: TObject);
PROCEDURE IntEdit1KeyPress(Sender: TObject; VAR Key: Char);
PROCEDURE FormActivate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
END;
```

VAR

Form1: TForm1;

IMPLEMENTATION

```
{ $R *.DFM }
FUNCTION isodd(n:integer):boolean;
BEGIN
  IF n mod 2 =0 THEN result:=false
  ELSE result:=true;
END;
```

// REALIZA EL LLENADO DE LA MATRIZ

```
PROCEDURE TForm1.CalcBtnClick(Sender: TObject);
VAR
  i,j,cont,orden,tam:integer;
  anti,antj:integer;
BEGIN
  orden:=STRTOINT(intedit1.text);
  IF (isodd(orden)) AND (orden<=51) THEN
    BEGIN
      WITH stringgrid1 DO
```



```

BEGIN
  {INICIALIZANDO}
  rowcount:=orden+1;
  colcount:=orden+1;
  FOR i:=0 TO colcount-1 DO
    FOR j:=0 TO rowcount-1 DO cells[i,j]:= "";
  tam:=orden*orden;
  {ASIGNAR UNA FUENTE GRANDE PARA CUADRADOS PEQUEÑOS, FUENTE PEQUEÑA PARA CUADRADOS GRANDES}
  IF tam<100 THEN font.size:=10 ELSE font.size:=8;
  {AJUSTE DEL TAMAÑO DE CELDAS APROPIADAS A LA FUENTE}
  canvas.font:=font;
  defaultcolwidth:=canvas.textwidth(inttostr(tam))+10;
  defaulttrowheight:=defaultcolwidth;
  {AJUSTE DEL TAMAÑO DE LA GRIDA}
  width:=(defaultcolwidth+1)*orden+5;
  height:=((defaulttrowheight+1)*orden+5);
  i:=orden div 2;
  j:=0;
  cont:=0;
  REPEAT
    inc(cont);
    cells[i,j]:=inttostr(cont);
    IF cont<tam THEN
      BEGIN
        anti:=i;
        dec(i);
        IF i<0 THEN i:=orden-1;
        antj:=j;
        dec(j);
        IF j<0 THEN j:=orden-1;
        IF cells[i,j]<>" THEN
          BEGIN
            i:=anti;
            j:=antj+1;
            IF j>orden THEN j:=0;
          END;
        IF cells[i,j]<>" THEN showMessage('ERROR DE ALGORITMO');
      END;
    UNTIL cont=tam;
  END;
  {SUMA DE FILAS}
  SumLbl.caption:='SUMA DE CADA FILA, COLUMNA, DIAGONAL ES '
    +inttostr((tam*orden+orden) div 2);
  Sumlbl.visible:=true;
END
ELSE beep;

```

END;

// INGRESO DE NUMEROS ENTEROS

PROCEDURE TForm1.IntEdit1KeyPress(Sender: TObject; **VAR** Key: Char);
{INGRESO DE NUMEROS ENTEROS}

BEGIN

IF NOT (key **IN** ['0'..'9',#8 {backsapace}]) **THEN**

BEGIN

key:=#0;

beep;

END;

END;

// MAXIMIZANDO VENTANA

PROCEDURE TForm1.FormActivate(Sender: TObject);

BEGIN

windowstate := wsMaximized;

END;

END. *//FIN DEL PROGRAMA*

PROBLEMA 16.

Dado una matriz de N filas x M columnas, escribir un programa de manera que mueva las columnas de dicha matriz, de manera tal que los elementos de la ultima fila queden ordenados en forma decreciente. Ver ejemplo del problema 10.

Solución:

Para ello preparar el siguiente entorno gráfico ver fig.

OPERACIONES CON MATRICES

COMPUTACION PARA INGENIERIA

Dada una matriz de orden NxM
El programa mueve las columnas de tal forma
que los elementos de la ultima columna
quedan ordenados en forma decreciente.

Ejemplo:

Matriz Inicial		Matriz Resultante
2 70 8 4	→	8 4 2 70
10 7 1 7		1 7 10 7
3 1 15 5		15 5 3 1

DATOS DE LA MATRIZ

Numero de Filas (<=10)

Numero de Columnas (<=15)

MATRIZ INICIAL

MATRIZ RESULTANTE

UNIT uOrden_Cols;

INTERFACE

USES

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, Buttons, Grids, Printers;

CONST Maxf=10;

CONST Maxc=15;

TYPE

TForm1 = class(TForm)
 StringGrid1: TStringGrid;
 CalcBtn: TBitBtn;
 Memo1: TMemo;
 GroupBox1: TGroupBox;
 Label2: TLabel;
 Label3: TLabel;
 Edit1: TEdit;
 BitBtn1: TBitBtn;
 Edit2: TEdit;
 Label1: TLabel;
 GroupBox2: TGroupBox;
 StringGrid2: TStringGrid;
 BitBtn2: TBitBtn;
 Button1: TButton;

PROCEDURE FormActivate(Sender: TObject);

PROCEDURE Edit2KeyPress(Sender: TObject; **VAR** Key: Char);

PROCEDURE BitBtn1Click(Sender: TObject);

PROCEDURE Edit1KeyPress(Sender: TObject; **VAR** Key: Char);

PROCEDURE CalcBtnClick(Sender: TObject);

PROCEDURE Button1Click(Sender: TObject);

private

 { Private declarations }

public

 { Public declarations }

END;

// DECLARACION DEL TIPO BASE DE MATRIZ

Mat = **ARRAY**[1..Maxf, 1..Maxc] **OF** Real;

VAR

Form1: TForm1;

A: Mat;

N, M, tam: integer;

IMPLEMENTATION

{ \$R *.DFM }

```

PROCEDURE TForm1.FormActivate(Sender: TObject);
BEGIN
    windowstate := wsMaximized; {VENTANA PANTALLA COMPLETA}
END;

```

// ENTRADA DE NUMEROS ENTEROS

```

PROCEDURE TForm1.Edit2KeyPress(Sender: TObject; VAR Key: Char);
BEGIN
IF NOT (key IN ['0'..'9',#8 {backspace}]) THEN
    BEGIN
        key:=#0;
        beep;
    END;
END;

```

```

PROCEDURE TForm1.BitBtn1Click(Sender: TObject);
VAR
    N,M,tam:integer;
BEGIN
    N:=STRTOINT(edit1.Text);
    M:=STRTOINT(edit2.Text);
    WITH stringgrid1 DO
        BEGIN
            {INICIALIZANDO}
            rowcount:=N;
            colcount:=M;
            tam:=N*M;

```

{ASIGNAR UNA FUENTE GRANDE PARA CUADRADOS PEQUEÑOS,FUENTE PEQUEÑA PARA CUADRADOS GRANDES}

```

    IF tam<100 THEN font.Size:=12 ELSE font.Size:=8;
    {AJUSTE DEL TAMAÑO DE CELDAS APROPIADAS A LA FUENTE}
    canvas.font:=font;
    defaultcolwidth:=canvas.textwidth(inttostr(M*M))+12;
    defaultrowheight:=canvas.textwidth(inttostr(N*N))+12;
    {AJUSTE DEL TAMAÑO DE LA GRIDA}
    width:=(defaultcolwidth+1)*M+5;
    height:=((defaultrowheight+1)*N+5);
    enabled:=true
    END;
END;

```

```

PROCEDURE TForm1.Edit1KeyPress(Sender: TObject; VAR Key: Char);
BEGIN
IF NOT (key IN ['0'..'9',#8 {backspace}]) THEN
    BEGIN
        key:=#0;

```

```

    beep;
  END;
END;
PROCEDURE TForm1.CalcBtnClick(Sender: TObject);
VAR
  i,j,k:integer;
  x:real;
BEGIN
  N:=STRTOINT(edit1.Text);
  M:=STRTOINT(edit2.Text);
  WITH stringgrid2 DO
    BEGIN
      rowcount:=N;
      colcount:=M;
      tam:=N*M;
      //ASIGNANDO TAMAÑO DE TEXTO
      IF tam<100 THEN font.Size:=12 ELSE font.Size:=8;
      canvas.font:=font;
      //MODIFICANDO TAMAÑO DE GRIDAS Y CELDAS
      defaultcolwidth:=canvas.textwidth(inttostr(M*M))+12;
      defaultrowheight:=canvas.textwidth(inttostr(N*N))+12;
      width:=(defaultcolwidth+1)*M+5;
      height:=((defaultrowheight+1)*N+5);
      enabled:=true
    END;
    //GUARDANDO LOS DATOS DEL SRINGGRID1 EN LA MATRIZ A
    FOR j:=0 TO M-1 DO
      BEGIN
        FOR i:=0 TO N-1 DO
          a[i+1,j+1]:=STRTOFLOAT(stringgrid1.cells[j,i]);
        END;
      //ORDENANDO LA ULTIMA FILA DE LA COLUMNA
      FOR i:=1 TO M-1 DO
        BEGIN
          FOR j:=i+1 TO M DO
            BEGIN
              IF a[N,i]<a[N,j] THEN
                BEGIN
                  FOR k:=1 TO N DO
                    BEGIN
                      x:=a[k,i];
                      a[k,i]:=a[k,j];
                      a[k,j]:=x;
                    END;
                  END;
                END;
            END;
          END;
        END;
      END;
    END;
  END;

```

// MOSTRANDO LA MATRIZ ORDENADA EN EL STRINGGRID 2

FOR i:=1 **TO** N **DO**

BEGIN

FOR j:=1 **TO** M **DO**

stringgrid2.Cells[j-1,i-1]:=FLOATTOSTR(a[i,j]);

END;

groupbox2.Visible:=true;

stringgrid2.Visible:=true;

END;

//IMPRESION DE MATRICES INICIAL Y RESULTANTE EN IMPRESORA

PROCEDURE TForm1.Button1Click(Sender: TObject);

VAR

i,j:integer;

BEGIN

WITH Printer **DO**

BEGIN

BeginDoc;

Canvas.Brush.Style := bsClear;

Canvas.TextHeight('10');

Canvas.TextOut(400,400,'REPORTE DEL PROGRAMA ORDENAR ULTIMA
FILA DE MATRIZ EN FORMA ASCENDENTE');

Canvas.TextOut(400,600 , 'MATRIZ INICIAL');

Canvas.TextOut(2400,600 , 'MATRIZ RESULTANTE');

FOR j:=0 **TO** stringgrid2.Colcount **DO**

BEGIN

FOR i:=0 **TO** stringgrid2.Rowcount **DO**

BEGIN

Canvas.TextOut(400+i*200,800 + (j *200),stringgrid1.Cells[i,j]);

Canvas.TextOut(2400+i*200,800 + (j *200),stringgrid2.Cells[i,j]);

END;

END;

Canvas.Brush.Color := clBlack;

EndDoc;

END;

END;

END. *// FIN DEL PROGRAMA*

PROBLEMA 17.

Se tiene datos de empresas constructoras: nombre de la empresa, código, dirección, teléfono. Se pide guardar esos datos en un archivo de texto y grabar en el disco duro y pedir al usuario la ruta en donde será grabada.

Solución:

Preparar un formulario de acuerdo a la figura siguiente fig2.

Modificar propiedades como **Caption** (texto a mostrar), **Kind** (tipo de boton figura)

Text (cajas de texto).

Fig2. Programa Agenda de empresas

Luego crear :

Procedimiento **sololectura**

Procedimientos de los cuatro botones en el evento **onclick** (pestaña **Events** del Object Inspector) o hacer un doble clic en el boton en el cual se desea que haga alguna acción.

UNIT uagenda;

INTERFACE

USES

windows, messages, sysutils, variants, classes, graphics, controls, forms, dialogs, borland.vcl.stdctrls, system.componentmodel, borland.vcl.buttons;

TYPE

```
tform1 = class(tform)
  bitbtn1: tbitbtn;
  edit1: tedit;
  edit2: tedit;
  edit3: tedit;
  edit4: tedit;
  bitbtn2: tbitbtn;
  button1: tbutton;
  button3: tbutton;
  label2: tlabel;
  label3: tlabel;
  label4: tlabel;
  label5: tlabel;
  PROCEDURE bitbtn1click(sender: tobject);
  PROCEDURE button1click(sender: tobject);
  PROCEDURE button3click(sender: tobject);
  PROCEDURE bitbtn2click(sender: tobject);
private
  { private declarations }
public
  { public declarations }
END;
```

VAR *//VARIABLES GLOBALES*

```
form1: tform1;
f:textfile;
ruta:string;
```

IMPLEMENTATION

```
{ $r *.nfm }
```

PROCEDURE sololectura(valor:boolean);

BEGIN

WITH form1 **DO**

BEGIN

```
      edit1.readonly:=valor;
      edit2.readonly:=valor;
```

```

        edit3.readonly:=valor;
        edit4.readonly:=valor;
    END;
END;

```

// PROCEDIMIENTO PARA ESCRIBIR ENCABEZADOS EN EL ARCHIVO

PROCEDURE tform1.bitbtn1click(sender: tobject);

BEGIN

Ruta:=**inputbox**('archivo','ruta y nombre','c:\ficheros\agenda.txt');

Assignfile(f,ruta);

Rewrite(f);

Writeln(f,#9,#9,'Relacion de Empresas Constructoras');

Writeln(f,'codigo',#9,'nombre de empresa',#9,'direccion',#9,'telefono');

Closefile(f);

Edit1.setfocus;

Bitbtn1.enabled:=false;

END;

// PROCEDIMIENTO PARA ALMACENAR DATOS EN EL ARCHIVO

PROCEDURE tform1.button1click(sender: tobject);

BEGIN

Assignfile(f,ruta);

Append(f);

Write(f,edit1.text,#9);

Write(f,edit2.text,#9);

Write(f,edit3.text,#9);

Writeln(f,edit4.text);

Closefile(f);

showmessage('registro de empresa almacenado');

Button1.enabled:=false;

Sololectura(true);

Button3.setfocus;

END;

{ PROCEDIMIENTO PARA BORRAR LOS DATOS QUE CONTIENEN LAS CAJAS DE TEXTO }

PROCEDURE tform1.button3click(sender: tobject);

BEGIN

Sololectura(false);

Edit1.clear;

Edit2.clear;

Edit3.clear;

Edit4.clear;

Edit1.setfocus;

Button1.enabled:=true;

Bitbtn1.enabled:=true;

END;

// PROCEDIMIENTO PARA CERRAR LA VENTANA

PROCEDURE tform1.bitbtn2click(sender: tobject);

BEGIN

Close;

END;

END. *// FIN DEL PROGRAMA*

PROBLEMA 18.

Realizar un programa que permita crear un archivo con los siguientes datos, carnet de identidad, nombre del alumno, nombre de la material y nota; una vez creado el archivo obtener la siguiente información:

- a) Obtener el nombre del mejor alumno y su nota.
- b) Obtener la cantidad de alumnos reprobados y aprobados.
- c) Obtener el promedio general de notas.

Solución:

Alternativa1: Sin utilizar ventanas y objetos.

Program pnotaar;

USES

sysutils,

dialogs;

TYPE

regis = record

ci:integer;

nota:integer;

nom:string[20];

mate:string[20]

END;

Archi= file **OF** regis;

VAR

Anotas:archi;

Rn:regis;

Op:string;

May, cap, crp, sn, ca:integer;

Prom:real;

Nom2, materia:string;

BEGIN

Assign(anotas, 'notas.dat');

Rewrite(anotas);

materia:='carreteras i';

REPEAT

```
rn.ci:=STRTOINT(inputbox('archivo de notas','ingrese ci',' '));
rn.nom:=inputbox('archivo de notas','ingrese nombre',' ');
rn.mate:=inputbox('archivo de notas','ingrese materia',materia);
rn.nota:=STRTOINT(inputbox('archivo de notas','ingrese nota',' '));
Write(anotas,rn);
op:=inputbox('archivo de notas','desea continuar s/n','s ');
materia:=rn.mate;
```

UNTIL op='n';

Close(anotas);

// EN ESTE BLOQUE SE OBTIENE EL MAJOR ALUMNO, NOTA, PROMEDIO

Reset(anotas);

may:=0;cap:=0;crp:=0;sn:=0;ca:=0;

WHILE NOT Eof (anotas) **DO**

BEGIN

read(anotas,rn);

IF rn.nota>may **THEN**

BEGIN

may:=rn.nota;

nom2:=rn.nom;

END;

IF rn.nota>=51 **THEN** cap:=cap+1 **ELSE** crp:=crp+1;

sn:=sn+rn.nota;

ca:=ca+1;

END;

prom:=sn/ca;

showmessage('mejor alumno es:'+nom2+#9+'nota='+inttostr(may));

showmessage('cantidad de aprobados'+inttostr(cap)+#13+'reprobados ='+'+inttostr(crp));

showmessage('promedio general ='+'+FLOATTOSTR(prom));

Close(anotas);

END.

Alternativa2:

Para resolver el problema utilizaremos el ambiente de diseño grafico de Delphi.

Preparar un formulario con los siguientes objetos como se muestra en la figura 3:

- 6 Botones, 4cajas de texto, 2 cajas agrupadoras, 4 etiquetas y un StrinGrid (celdas de texto)
- Modificar propiedades de StrinGrid:
 - 1) **FixedRows**, **FixedCols** igual a cero
 - 2) **Options: GoEditing** = true (habilita edición de celdas), **GoTabs** = trae
 - 3) **RowCount** =15 , **ColCount** = 4

The screenshot shows a Delphi form titled "Form1". It contains a table with the following headers: "NOMBRE Y APELLIDO", "CARNET DE IDENTIDAD", "MATERIA", and "NOTA O PUNTAJE". Below the table, there is a section with several buttons: "GENERAR" (with a green checkmark), "INGRESAR DATOS" (with a green checkmark), "REGISTRAR" (with a green checkmark), "CALCULAR", "IMPRIMIR", and "CERRAR" (with a red X). To the right of these buttons, there is a "RESULTADOS" section with labels and text boxes for "MAYOR NOTA", "CANTIDAD DE ALUMNOS APROBADOS", "CANTIDAD DE ALUMNOS REPROBADOS", and "PROMEDIO DE NOTAS".

Fig. 3 archivo con tipo (archivo de registros)

UNIT uarnot;

INTERFACE

USES

windows, messages, sysutils, variants, classes, graphics, controls, forms, dialogs, buttons, stdctrls, grids;

TYPE

```
tform1 = class(tform)
  groupbox1: tgroupbox;
  button1: tbutton;
  bitbtn1: tbitbtn;
  groupbox2: tgroupbox;
  label5: tlabel;
  label6: tlabel;
  label7: tlabel;
  label8: tlabel;
  edit5: tedit;
  edit6: tedit;
  edit7: tedit;
  edit8: tedit;
  edit4: tedit;
  bitbtn2: tbitbtn;
  bitbtn3: tbitbtn;
  button2: tbutton;
  stringgrid1: tstringgrid;
  label1: tlabel;
  label2: tlabel;
  label3: tlabel;
  label4: tlabel;
  bitbtn4: tbitbtn;
  PROCEDURE bitbtn3click(sender: tobject);
  PROCEDURE bitbtn1click(sender: tobject);
  PROCEDURE button1click(sender: tobject);
  PROCEDURE bitbtn2click(sender: tobject);
  PROCEDURE bitbtn4click(sender: tobject);
private
  { private declarations }
public
  { public declarations }
END;
regis = record
  ci:integer;
  nota:integer;
  nom:string[20];
  mate:string[20]
```

```

END;
Archi= file OF regis;
VAR
  form1: tform1;
  anotas:archi;
  rn:regis;
  may,cap,crp,sn,ca,num:integer;
  prom:real;
  nom2:string;
  nomar:string;

```

IMPLEMENTATION

```
{ $r *.dfm }
```

// PROCEDIMIENTO PARA ASIGNAR EL NOMBRE AL ARCHIVO

```
PROCEDURE tform1.bitbtn3click(sender: tobject);
```

```
BEGIN
```

```
  Assignfile(anotas,'notas.dat');
```

```
  Rewrite(anotas);
```

```
  Closefile(anotas);
```

```
  Bitbtn3.enabled:=false
```

```
END;
```

// PROCEDIMIENTO PARA ESCRIBIR LOS DATOS DEL STRINGGRID AL ARCHIVO

```
PROCEDURE tform1.bitbtn1click(sender: tobject);
```

```
VAR n,i:integer;
```

```
BEGIN
```

```
  Assignfile(anotas,'notas.dat');
```

```
  Reset(anotas);
```

```
  n:=filesize(anotas);
```

```
  seek(anotas,n);
```

```
  FOR i:=0 TO num-1 DO
```

```
    BEGIN
```

```
      rn.nom:=stringgrid1.cells[0,i];
```

```
      rn.ci:=STRTOINT(stringgrid1.cells[1,i]);
```

```
      rn.mate:=stringgrid1.cells[2,i];
```

```
      rn.nota:=STRTOINT(stringgrid1.cells[3,i]);
```

```
      Write(anotas,rn);
```

```
    END;
```

```
  Closefile(anotas);
```

```
  showmessage('registro almacenado');
```

```
  stringgrid1.enabled:=false;
```

```
END;
```

// PROCEDIMIENTO PARA CALCULAR LO REQUERIDO

```

PROCEDURE tform1.button1click(sender: tobject);

BEGIN
Reset(anotas);
may:=0;cap:=0;crp:=0;sn:=0;ca:=0;
WHILE NOT Eof (anotas) DO
BEGIN
    read(anotas,rn);
    IF rn.nota>may THEN
        BEGIN
            may:=rn.nota;
            nom2:=rn.nom;
        END;
    IF rn.nota>=51 THEN cap:=cap+1 ELSE crp:=crp+1;
    sn:=sn+rn.nota;
    ca:=ca+1;
END;
prom:=sn/ca;
edit8.text:=FLOATTOSTR(prom); edit8.visible:=true;
edit4.text:=inttostr(may); edit4.visible:=true;
edit5.text:=nom2; edit5.visible:=true;
edit6.text:=inttostr(cap); edit6.visible:=true;
edit7.text:=inttostr(crp); edit7.visible:=true;
Closefile(anotas);
END;
PROCEDURE tform1.bitbtn2click(sender: tobject);

BEGIN
Close; // PROCEDIMIENTO PARA CERRAR EL PROGRAMA

END;
// PROCEDIMIENTO PARA INSERTAR DATOS EN EL STRINGGRID
PROCEDURE tform1.bitbtn4click(sender: tobject);

VAR
i,j:integer;
BEGIN
    REPEAT
        num:=STRTOINT(inputbox('archivo de notas','ingrese numero de
datos(filas)','3'));
    UNTIL num IN [1..15];
    FOR j:=0 TO 4 DO
        BEGIN
            FOR i:=0 TO 20 DO
                stringgrid1.cells[j,i]:=' ';
            END;
        stringgrid1.enabled:=true;
    END;
END. // FIN DEL PROGRAMA

```


PROBLEMA 19.

Realizar un programa de dibujo que grafique las siguientes formas: rectángulo, cuadrado, elipse, con diferentes tipos de relleno, grosores de línea, colores. Utilizando el componente Shape de Delphi.

Solución:

Para ello preparar el siguiente entorno gráfico ver fig. 4.

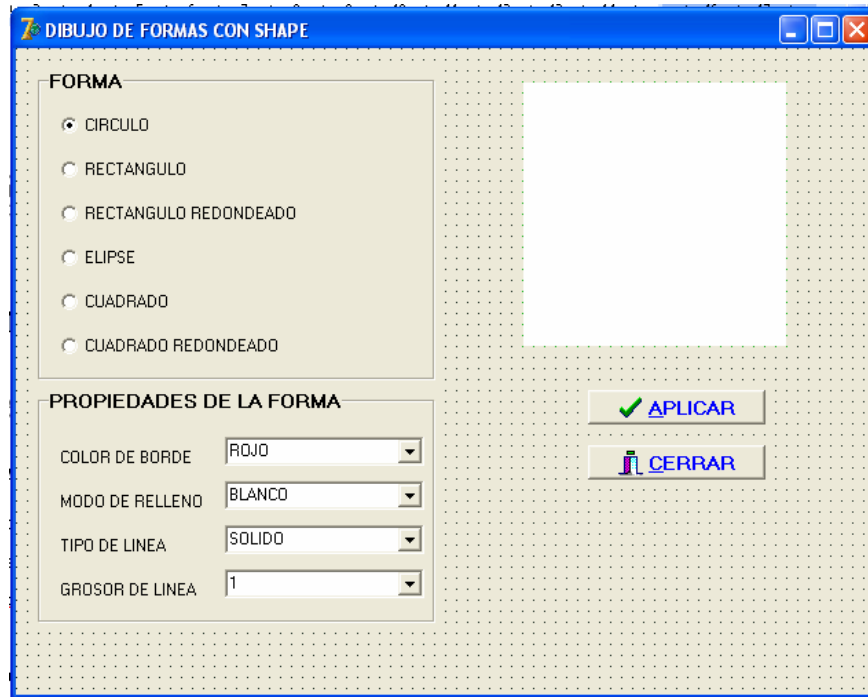


Fig. 4 Programa de dibujo con el componente **Shape**

Adicionar el componente **Shape** de la barra de objetos Additional y posicionar en el extremo derecho superior.

Codigo en Delphi

UNIT ugrafshape;

INTERFACE

USES

windows, messages, sysutils, variants, classes, graphics, controls, forms, dialogs, stdctrls, extctrls, buttons;

TYPE

```
tform1 = class(tform)
  shape1: tshape;
  radiogroup1: tradiogroup;
  radiobutton1: tradiobutton;
  radiobutton2: tradiobutton;
  radiobutton3: tradiobutton;
  radiobutton4: tradiobutton;
  radiobutton5: tradiobutton;
  radiobutton6: tradiobutton;
  bitbtn1: tbitbtn;
  groupbox1: tgroupbox;
  combobox1: tcombobox;
  combobox2: tcombobox;
  combobox3: tcombobox;
  combobox4: tcombobox;
  label1: tlabel;
  label2: tlabel;
  label3: tlabel;
  label4: tlabel;
  bitbtn2: tbitbtn;
```

PROCEDURE bitbtn1click(sender: tobject);

private

{ **private** declarations }

public

{ **public** declarations }

END;

VAR

form1: tform1;

IMPLEMENTATION

{ \$r *.dfm }

PROCEDURE tform1.bitbtn1click(sender: tobject);

VAR

Op:tshape**Type**;

Color:tcolor;

Relleno:tpenmode;

Tipol:tpenstyle;

Grosor:integer;

BEGIN

// DETERMINA QUE OPCION ESTA SELECCIONADA

IF radiobutton1.checked=true **THEN** op:=stcircle;

IF radiobutton2.checked=true **THEN** op:=strectangle;

IF radiobutton3.checked=true **THEN** op:=stellipse;

IF radiobutton4.checked=true **THEN** op:=stroundrect;

IF radiobutton5.checked=true **THEN** op:=stsquare;

IF radiobutton6.checked=true **THEN** op:=stroundsquare;

// DETERMINA EL COLOR DE LA FORMA

IF (combobox1.text)='rojo' **THEN** color:=clred;

IF (combobox1.text)='azul' **THEN** color:=clblue;

IF (combobox1.text)='verde' **THEN** color:=clgreen;

IF (combobox1.text)='amarillo' **THEN** color:=clyellow;

IF (combobox1.text)='plomo' **THEN** color:=clgray;

// DETERMINA EL TIPO DE RELLENO

IF (combobox2.text)='negro' **THEN** relleno:=pmblack;

IF (combobox2.text)='blanco' **THEN** relleno:=pmcopy;

IF (combobox2.text)='transparente' **THEN** relleno:=pmmask;

IF (combobox2.text)='union' **THEN** relleno:=pmmerge;

// DETERMINA EL TIPO DE LINEA

IF (combobox3.text)='solido' **THEN** tipol:=pssolid;

IF (combobox3.text)='segmentado' **THEN** tipol:=psdash;

IF (combobox3.text)='punteado' **THEN** tipol:=psdot;

grosor:=**STRTOINT**(combobox4.text);

// DIBUJA LA FORMA Y SUS OPCIONES

shape1.shape:=op;

shape1.pen.color:=color;

shape1.pen.mode:=relleno;

shape1.pen.style:=tipol;

shape1.pen.width:=grosor;

END;

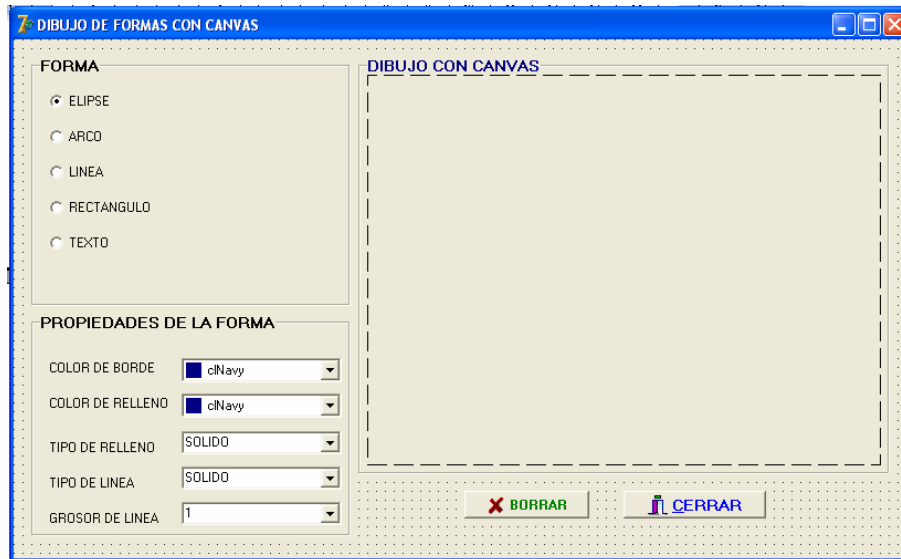
END.

PROBLEMA 20.

Realizar un programa de dibujo que grafique las siguientes formas: rectángulo, cuadrado, elipse, con diferentes tipos de relleno, grosores de línea, colores. Utilizando el componente Canvas de Delphi.

Solución:

Para ello preparar el siguiente entorno gráfico ver fig.



UNIT uCanvas;

INTERFACE

USES

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, ExtCtrls, Buttons;

TYPE

```
TForm1 = class(TForm)
  RadioGroup1: TRadioGroup;
  RadioButton1: TRadioButton;
  RadioButton2: TRadioButton;
  RadioButton3: TRadioButton;
  RadioButton4: TRadioButton;
  RadioButton5: TRadioButton;
  GroupBox1: TGroupBox;
  ComboBox3: TComboBox;
  ComboBox4: TComboBox;
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  Label4: TLabel;
  BitBtn2: TBitBtn;
  ColorBox1: TColorBox;
  ColorBox2: TColorBox;
  ComboBox1: TComboBox;
  Label5: TLabel;
  GroupBox2: TGroupBox;
  Image1: TImage;
  BitBtn1: TBitBtn;
PROCEDURE Image1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
PROCEDURE BitBtn1Click(Sender: TObject);
PROCEDURE Image1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
```

```
private
  { Private declarations }
public
  { Public declarations }
END;
```

VAR

```
Form1: TForm1;
xi,yi:integer;
```

IMPLEMENTATION

```
{ $R *.dfm }
```

```
// CAPTURA LAS COORDENADAS DEL RATON CUANDO SE PRESIONA EL BOTON
```

```
PROCEDURE TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;  
  Shift: TShiftState; X, Y: Integer);  
BEGIN  
  xi:=x;yi:=y;  
END;
```

```
PROCEDURE TForm1.BitBtn1Click(Sender: TObject);  
VAR  
  ARect: TRect;  
BEGIN  
  WITH Image1.Canvas DO  
    BEGIN  
      CopyMode := cmWhiteness;  
      ARect := Rect(0, 0, Image1.Width, Image1.Height);  
      CopyRect(ARect, Image1.Canvas, ARect);  
      //CopyMode := cmSrcCopy;  
    END;  
END;
```

```
PROCEDURE TForm1.Image1MouseUp(Sender: TObject; Button: TMouseButton;  
  Shift: TShiftState; X, Y: Integer);  
VAR  
  OP:TSHAPETYPE;  
  COLOR:TCOLOR;  
  RELLENO:TPENMODE;  
  TIPre:TBrushStyle;  
  TIPOL:TPENSTYLE;  
  GROSOR:INTEGER;  
  texto:string;  
BEGIN
```

```
// DETERMINAEL TIPO DE RELLENO
```

```
IF (Combobox1.Text)='SOLIDO' THEN TIPre:=bsSolid;  
IF (Combobox1.Text)='HORIZONTAL' THEN TIPre:=bsHorizontal;  
IF (Combobox1.Text)='VERTICAL' THEN TIPre:=bsVertical;  
IF (Combobox1.Text)='DIAGONAL' THEN TIPre:=bsFDiagonal;  
IF (Combobox1.Text)='TRANSVERSAL' THEN TIPre:=bsCross;
```

```
// DETERMINAEL TIPO DE LINEA
```

```
IF (Combobox3.Text)='SOLIDO' THEN TIPOL:=psSolid;  
IF (Combobox3.Text)='SEGMENTADO' THEN TIPOL:=psDash;  
IF (Combobox3.Text)='PUNTEADO' THEN TIPOL:=psDot;
```

```
GROSOR:=STRTOINT(COMBOBOX4.Text);
```

```
// DIBUJA LA FORMA Y SUS OPCIONES
```

```
image1.Canvas.Pen.Color:= colorbox1.Selected;
```

```
image1.Canvas.Brush.Color:=colorbox2.Selected;
```

```
image1.Canvas.Pen.Style:=TIPOL;
```

```
image1.Canvas.Pen.Width:=grosor;
```

```
image1.Canvas.Brush.Style:=TIPre;
```

```
IF RadioButton1.Checked=true THEN
```

```
    image1.Canvas.Ellipse(xi,yi,X,Y);
```

```
IF RadioButton2.Checked=true THEN
```

```
    image1.Canvas.Arc(xi,yi,X,Y,xi,(yi+y)div 2,X,(yi+y)div 2);
```

```
IF RadioButton3.Checked=true THEN
```

```
    BEGIN
```

```
        image1.Canvas.MoveTo(xi,yi);
```

```
        image1.Canvas.LineTo(x,y);
```

```
    END;
```

```
IF RadioButton4.Checked=true THEN
```

```
    image1.Canvas.Rectangle(xi,yi,x,y);
```

```
IF RadioButton5.Checked=true THEN
```

```
    BEGIN
```

```
        texto:=inputbox('Componente Canvas','Ingrese Texto a Mostrar','Canvas');
```

```
        image1.Canvas.TextOut(x,y,texto);
```

```
        image1.Canvas.Chord(10,10,100,10,20,50,20,100);
```

```
    END;
```

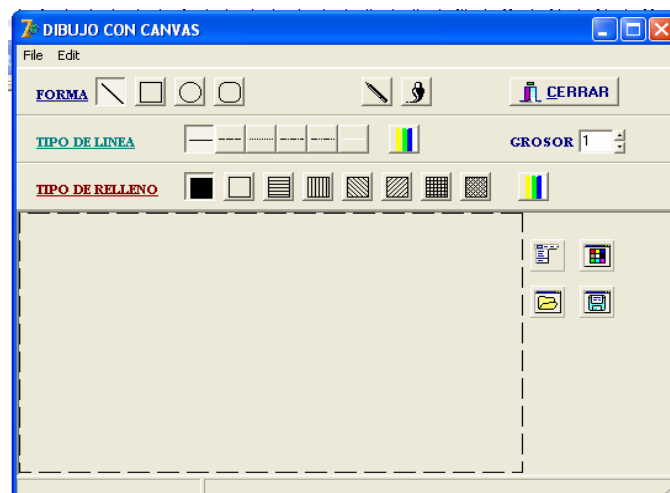
```
END;
```

```
END. //FIN DEL PROGRAMA
```

ALTERNATIVA 2

Solución:

Para ello preparar el siguiente entorno gráfico ver fig.



UNIT GRAFICOS;

INTERFACE

USES

SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs,
Buttons, ExtCtrls, StdCtrls, ComCtrls, Menus;

TYPE

```
TDrawingTool = (dtLine, dtRectangle, dtEllipse, dtRoundRect);  
TForm1 = class(TForm)  
    Panel1: TPanel;  
    LineButton: TSpeedButton;  
    RectangleButton: TSpeedButton;  
    EllipseButton: TSpeedButton;  
    RoundRectButton: TSpeedButton;  
    PenButton: TSpeedButton;  
    BrushButton: TSpeedButton;  
    PenBar: TPanel;  
    BrushBar: TPanel;  
    SolidPen: TSpeedButton;  
    DashPen: TSpeedButton;  
    DotPen: TSpeedButton;  
    DashDotPen: TSpeedButton;  
    DashDotDotPen: TSpeedButton;  
    ClearPen: TSpeedButton;  
    PenWidth: TUpDown;  
    PenSize: TEdit;  
    StatusBar1: TStatusBar;  
    ScrollBox1: TScrollBox;  
    Image: TImage;  
    SolidBrush: TSpeedButton;  
    ClearBrush: TSpeedButton;  
    HorizontalBrush: TSpeedButton;  
    VerticalBrush: TSpeedButton;  
    FDiagonalBrush: TSpeedButton;  
    BDiagonalBrush: TSpeedButton;  
    CrossBrush: TSpeedButton;  
    DiagCrossBrush: TSpeedButton;  
    PenColor: TSpeedButton;  
    BrushColor: TSpeedButton;  
    ColorDialog1: TColorDialog;  
    MainMenu1: TMainMenu;  
    File1: TMenuItem;  
    New1: TMenuItem;  
    Open1: TMenuItem;  
    Save1: TMenuItem;
```



```

Saveas1: TMenuItem;
N1: TMenuItem;
Exit1: TMenuItem;
Edit1: TMenuItem;
Cut1: TMenuItem;
Copy1: TMenuItem;
Paste1: TMenuItem;
OpenDialog1: TOpenDialog;
SaveDialog1: TSaveDialog;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
BitBtn1: TBitBtn;
PROCEDURE FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
PROCEDURE FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
PROCEDURE FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
PROCEDURE LineButtonClick(Sender: TObject);
PROCEDURE RectangleButtonClick(Sender: TObject);
PROCEDURE EllipseButtonClick(Sender: TObject);
PROCEDURE RoundRectButtonClick(Sender: TObject);
PROCEDURE PenButtonClick(Sender: TObject);
PROCEDURE BrushButtonClick(Sender: TObject);
PROCEDURE SetPenStyle(Sender: TObject);
PROCEDURE PenSizeChange(Sender: TObject);
PROCEDURE FormCreate(Sender: TObject);
PROCEDURE SetBrushStyle(Sender: TObject);
PROCEDURE PenColorClick(Sender: TObject);
PROCEDURE BrushColorClick(Sender: TObject);
PROCEDURE Exit1Click(Sender: TObject);
PROCEDURE Open1Click(Sender: TObject);
PROCEDURE Save1Click(Sender: TObject);
PROCEDURE Saveas1Click(Sender: TObject);
PROCEDURE New1Click(Sender: TObject);
PROCEDURE Copy1Click(Sender: TObject);
PROCEDURE Cut1Click(Sender: TObject);
PROCEDURE Paste1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
  BrushStyle: TBrushStyle;
  PenStyle: TPenStyle;
  PenWide: Integer;

```

```

    Drawing: Boolean;
    Origin, MovePt: TPoint;
    DrawingTool: TDrawingTool;
    CurrentFile: string;
PROCEDURE SaveStyles;
PROCEDURE RestoreStyles;
PROCEDURE DrawShape(TopLeft, BottomRight: TPoint; AMode: TPenMode);
END;

VAR
    Form1: TForm1;
IMPLEMENTATION

USES BMPDlg, Clipbrd;

{$R *.dfm}
// CAPTURA COORDENADAS DEL RATON
PROCEDURE TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
BEGIN
    Drawing := True;
    Image.Canvas.MoveTo(X, Y);
    Origin := Point(X, Y);
    MovePt := Origin;
    StatusBar1.Panels[0].Text := Format('Origin: (%d, %d)', [X, Y]);
END;

PROCEDURE TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
BEGIN
    IF Drawing THEN
        BEGIN
            DrawShape(Origin, Point(X, Y), pmCopy);
            Drawing := False;
        END;
    END;

PROCEDURE TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
BEGIN
    IF Drawing THEN
        BEGIN
            DrawShape(Origin, MovePt, pmNotXor);
            MovePt := Point(X, Y);
            DrawShape(Origin, MovePt, pmNotXor);
        END;
        StatusBar1.Panels[1].Text := Format('Current: (%d, %d)', [X, Y]);
    END;

```

// DEFINE QUE OPCION DE FORMA ES SELECCIONADA

PROCEDURE TForm1.LineButtonClick(Sender: TObject);

BEGIN

DrawingTool := dtLine;

END;

PROCEDURE TForm1.RectangleButtonClick(Sender: TObject);

BEGIN

DrawingTool := dtRectangle;

END;

PROCEDURE TForm1.EllipseButtonClick(Sender: TObject);

BEGIN

DrawingTool := dtEllipse;

END;

PROCEDURE TForm1.RoundRectButtonClick(Sender: TObject);

BEGIN

DrawingTool := dtRoundRect;

END;

// DIBUJA LA FORMA Y SUS OPCIONES

PROCEDURE TForm1.DrawShape(TopLeft, BottomRight: TPoint; AMode: TPenMode);

BEGIN

WITH Image.Canvas **DO**

BEGIN

Pen.Mode := AMode;

case DrawingTool **OF**

dtLine:

BEGIN

Image.Canvas.MoveTo(TopLeft.X, TopLeft.Y);

Image.Canvas.LineTo(BottomRight.X, BottomRight.Y);

END;

dtRectangle: Image.Canvas.Rectangle(TopLeft.X, TopLeft.Y, BottomRight.X,
BottomRight.Y);

dtEllipse: Image.Canvas.Ellipse(TopLeft.X, TopLeft.Y, BottomRight.X,
BottomRight.Y);

dtRoundRect: Image.Canvas.RoundRect(TopLeft.X, TopLeft.Y, BottomRight.X,
BottomRight.Y, (TopLeft.X - BottomRight.X) div 2,
(TopLeft.Y - BottomRight.Y) div 2);

END;

END;

END;

PROCEDURE TForm1.PenButtonClick(Sender: TObject);

BEGIN

```
PenBar.Visible := PenButton.Down;  
END;
```

```
PROCEDURE TForm1.BrushButtonClick(Sender: TObject);  
BEGIN  
BrushBar.Visible := BrushButton.Down;  
END;
```

```
PROCEDURE TForm1.SetPenStyle(Sender: TObject);  
BEGIN  
WITH Image.Canvas.Pen DO  
  BEGIN  
    IF Sender = SolidPen THEN Style := psSolid  
    ELSE IF Sender = DashPen THEN Style := psDash  
    ELSE IF Sender = DotPen THEN Style := psDot  
    ELSE IF Sender = DashDotPen THEN Style := psDashDot  
    ELSE IF Sender = DashDotDotPen THEN Style := psDashDotDot  
    ELSE IF Sender = ClearPen THEN Style := psClear;  
  END;  
END;
```

```
PROCEDURE TForm1.PenSizeChange(Sender: TObject);  
BEGIN  
Image.Canvas.Pen.Width := PenWidth.Position;  
END;
```

```
PROCEDURE TForm1.FormCreate(Sender: TObject);  
VAR  
Bitmap: TBitmap;  
BEGIN  
Bitmap := nil;  
  TRY  
    Bitmap := TBitmap.Create;  
    Bitmap.Width := 200;  
    Bitmap.Height := 200;  
    Image.Picture.Graphic := Bitmap;  
  finally  
    Bitmap.Free;  
  END;  
END;
```

```
PROCEDURE TForm1.SetBrushStyle(Sender: TObject);  
BEGIN  
WITH Image.Canvas.Brush DO  
  BEGIN  
    IF Sender = SolidBrush THEN Style := bsSolid  
    ELSE IF Sender = ClearBrush THEN Style := bsClear
```

```
ELSE IF Sender = HorizontalBrush THEN Style := bsHorizontal
ELSE IF Sender = VerticalBrush THEN Style := bsVertical
ELSE IF Sender = FDiagonalBrush THEN Style := bsFDiagonal
ELSE IF Sender = BDiagonalBrush THEN Style := bsBDiagonal
ELSE IF Sender = CrossBrush THEN Style := bsCross
ELSE IF Sender = DiagCrossBrush THEN Style := bsDiagCross;
END;
END;
```

```
PROCEDURE TForm1.PenColorClick(Sender: TObject);
BEGIN
  ColorDialog1.Color := Image.Canvas.Pen.Color;
  IF ColorDialog1.Execute THEN
    Image.Canvas.Pen.Color := ColorDialog1.Color;
END;
PROCEDURE TForm1.BrushColorClick(Sender: TObject);
BEGIN
  ColorDialog1.Color := Image.Canvas.Brush.Color;
  IF ColorDialog1.Execute THEN
    Image.Canvas.Brush.Color := ColorDialog1.Color;
END;
```

```
PROCEDURE TForm1.Exit1Click(Sender: TObject);
BEGIN
  Close;
END;
```

//APERTURA DE UN ARCHIVO DE DIBUJO

```
PROCEDURE TForm1.Open1Click(Sender: TObject);
BEGIN
  IF OpenFileDialog1.Execute THEN
    BEGIN
      CurrentFile := OpenFileDialog1.FileName;
      SaveStyles;
      Image.Picture.LoadFromFile(CurrentFile);
      RestoreStyles;
    END;
END;
```

// GUARDA EL DIBUJO EN UN ARCHIVO

```
PROCEDURE TForm1.Save1Click(Sender: TObject);
BEGIN
  IF CurrentFile <> EmptyStr THEN
    Image.Picture.SaveToFile(CurrentFile)
  ELSE SaveAs1Click(Sender);
END;
```

```
PROCEDURE TForm1.Saveas1Click(Sender: TObject);  
BEGIN  
  IF SaveDialog1.Execute THEN  
    BEGIN  
      CurrentFile := SaveDialog1.FileName;  
      Save1Click(Sender);  
    END;  
END;
```

//CREA UN NUEVO DIBUJO

```
PROCEDURE TForm1.New1Click(Sender: TObject);  
VAR  
  Bitmap: TBitmap;  
BEGIN  
  WITH NewBMPForm DO  
    BEGIN  
      ActiveControl := WidthEdit;  
      WidthEdit.Text := IntToStr(Image.Picture.Graphic.Width);  
      HeightEdit.Text := IntToStr(Image.Picture.Graphic.Height);  
      IF ShowModal <> idCancel THEN  
        BEGIN  
          Bitmap := nil;  
          TRY  
            Bitmap := TBitmap.Create;  
            Bitmap.Width := STRTOINT(WidthEdit.Text);  
            Bitmap.Height := STRTOINT(HeightEdit.Text);  
            SaveStyles;  
            Image.Picture.Graphic := Bitmap;  
            RestoreStyles;  
            CurrentFile := EmptyStr;  
          finally  
            Bitmap.Free;  
          END;  
        END;  
      END;  
    END;  
END;
```

// COPIA , CORTA Y PEGA UN DIBUJO

```
PROCEDURE TForm1.Copy1Click(Sender: TObject);  
BEGIN  
  Clipboard.Assign(Image.Picture);  
END;
```

```
PROCEDURE TForm1.Cut1Click(Sender: TObject);  
VAR  
  ARect: TRect;  
BEGIN
```

```
Copy1Click(Sender);
WITH Image.Canvas DO
BEGIN
    CopyMode := cmWhiteness;
    ARect := Rect(0, 0, Image.Width, Image.Height);
    CopyRect(ARect, Image.Canvas, ARect);
    CopyMode := cmSrcCopy;
END;
END;

PROCEDURE TForm1.Paste1Click(Sender: TObject);
VAR
    Bitmap: TBitmap;
BEGIN
    IF Clipboard.HasFormat(CF_BITMAP) THEN
        BEGIN
            Bitmap := TBitmap.Create;
            TRY
                Bitmap.Assign(Clipboard);
                Image.Canvas.Draw(0, 0, Bitmap);
            finally
                Bitmap.Free;
            END;
        END;
    END;
END;

PROCEDURE TForm1.SaveStyles;
BEGIN
    WITH Image.Canvas DO
        BEGIN
            BrushStyle := Brush.Style;
            PenStyle := Pen.Style;
            PenWide := Pen.Width;
        END;
    END;

PROCEDURE TForm1.RestoreStyles;
BEGIN
    WITH Image.Canvas DO
        BEGIN
            Brush.Style := BrushStyle;
            Pen.Style := PenStyle;
            Pen.Width := PenWide;
        END;
    END;

END.           //FIN DEL PROGRAMA
```

EJERCICIOS APLICADOS A INGENIERIA CIVIL

APLICACION 1.

APLICACION CARRETERAS

El programa calcula las propiedades de una curva horizontal, aplicable al diseño geométrico de carreteras.

Entonces preparar el siguiente interfaz gráfico en Delphi fig. 1. Que consta de 2 formularios: 1 para la aplicación y el otro para impresión utilizando el componente Richedit.

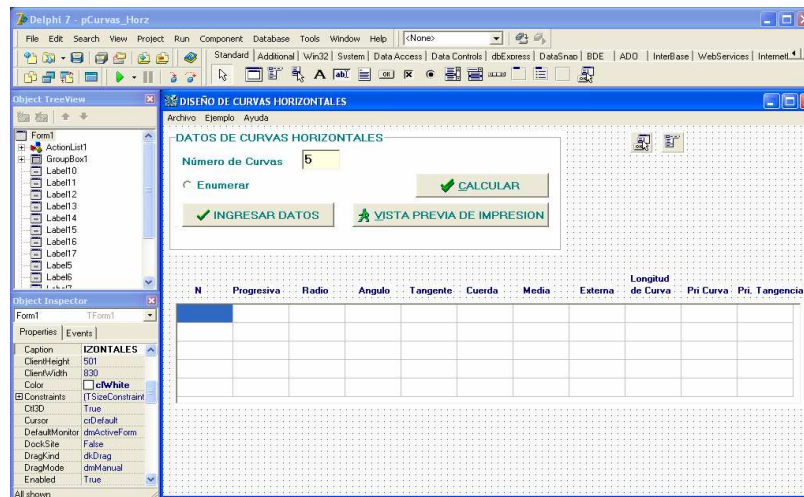
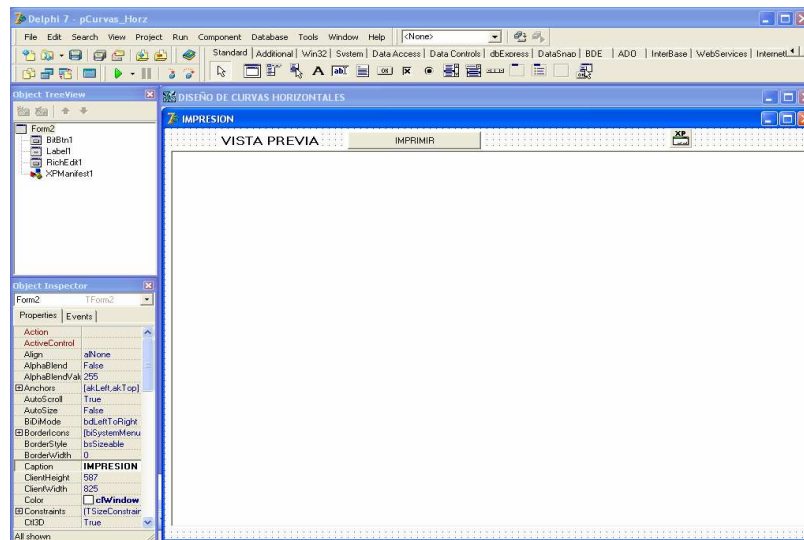


Fig. 1 Interfaz grafico.



UNIT uCurvas_Horz;

INTERFACE

USES

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, Grids, Buttons, ExtCtrls, Math, QuickRpt, QRCtrls,
XPMAN, Menus, Printers, OleServer, WordXP, ActnList, ExtActns;

TYPE

```
TForm1 = class(TForm)
  GroupBox1: TGroupBox;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  Edit1: TEdit;
  Label9: TLabel;
  Label10: TLabel;
  Label11: TLabel;
  Label12: TLabel;
  Label13: TLabel;
  Label14: TLabel;
  Label15: TLabel;
  StringGrid2: TStringGrid;
  BitBtn3: TBitBtn;
  Bevel1: TBevel;
  XPManifest1: TXPManifest;
  MainMenu1: TMainMenu;
  Archivo1: TMenuItem;
  Ejemplo1: TMenuItem;
  Ayuda1: TMenuItem;
  Informacion1: TMenuItem;
  Creditos1: TMenuItem;
  Recuperar1: TMenuItem;
  Borrar1: TMenuItem;
  Calcular1: TMenuItem;
  Nuevo1: TMenuItem;
  Guardar1: TMenuItem;
  N1: TMenuItem;
  Cerrar1: TMenuItem;
  Imprimir1: TMenuItem;
  Label5: TLabel;
  Label6: TLabel;
  Label7: TLabel;
  Label8: TLabel;
  Label16: TLabel;
  Label17: TLabel;
  RadioButton1: TRadioButton;
```

```

    ActionList1: TActionList;
    FileRun1: TFileRun;
PROCEDURE Edit1KeyPress(Sender: TObject; VAR Key: Char);
PROCEDURE BitBtn1Click(Sender: TObject);
PROCEDURE BitBtn2Click(Sender: TObject);
PROCEDURE Cerrar1Click(Sender: TObject);
PROCEDURE Recuperar1Click(Sender: TObject);
PROCEDURE Borrar1Click(Sender: TObject);
PROCEDURE Nuevo1Click(Sender: TObject);
PROCEDURE BitBtn3Click(Sender: TObject);
PROCEDURE Creditos1Click(Sender: TObject);
PROCEDURE Guardar1Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
END;
Mattipo=ARRAY[1..20,1..20] OF real;
VAR
    Form1: TForm1;
    F: TextFile;
    Dirc:string;
    N,tam:integer;
    mat1,mat2:Mattipo;
IMPLEMENTATION

USES uCurvas_Horz2;
{$R *.dfm}
{PROCEDURE MinCuad2(sender:tstringgrid;Num:integer;a:Mattipo);
BEGIN
    sender.cells[0,1]:='78';
END;}

PROCEDURE TForm1.Edit1KeyPress(Sender: TObject; VAR Key: Char);
BEGIN
    IF NOT (key IN ['0'..'9',#8 {backsapace}]) THEN
        BEGIN
            key:=#0;
            beep;
        END;
    END;
PROCEDURE Borrar;
VAR i,j:integer;
BEGIN
    WITH form1.StringGrid2 DO
        BEGIN
            FOR j:=0 TO 10 DO

```

```

BEGIN
  FOR i:=0 TO N DO
    cells[j,i]:=' ';
  END;
END;
END;

```

```

PROCEDURE TForm1.BitBtn1Click(Sender: TObject);
VAR i:integer;

```

```

BEGIN
  showmessage('Ingresar Datos de Cada Curva'+#13+'
  ';; PI, Progresiva, Radio, Angulo Delta');
  N:=STRTOINT(edit1.Text);
  WITH stringgrid2 DO
    BEGIN
      {INICIALIZANDO}
      rowcount:=N;
      {AJUSTE DEL TAMAÑO DE LA GRIDA}
      height:=((defaultrowheight+1)*(N)+5);
      enabled:=true;
      IF radiobutton1.Checked=true THEN
        BEGIN
          FOR i:=0 TO N-1 DO
            Cells[0,i]:='PI-'+inttostr(i+1);
          END;
          SetFocus;
        END;
      END;
    END;

```

```

PROCEDURE TForm1.BitBtn2Click(Sender: TObject);

```

```

VAR
  i:integer;
  prog,r,d,t,c,m,e,lc,pric,prit:real;
BEGIN
  WITH StringGrid2 DO
    BEGIN
      FOR i:=0 TO N-1 DO
        BEGIN
          prog:=STRTOFLOAT(stringgrid2.Cells[1,i]);
          r:=STRTOFLOAT(stringgrid2.Cells[2,i]);
          d:=STRTOFLOAT(stringgrid2.Cells[3,i]);
          t:=r*tan(pi*d/360);cells[4,i]:=format('%0:3f3',[t]);
          c:=2*r*sin(pi*d/360);cells[5,i]:=format('%0:3f3',[c]);
          m:=r*(1-cos(pi*d/360));cells[6,i]:=format('%0:3f3',[m]);
          e:=r*(1/(cos(pi*d/360))-1);cells[7,i]:=format('%0:3f3',[e]);
          lc:=r*pi*d/180;cells[8,i]:=format('%0:3f3',[lc]);
          pric:=prog-t;cells[9,i]:=format('%0:3f3',[pric]);
          prit:=prog+t;cells[10,i]:=format('%0:3f3',[prit]);
        END;
      END;
    END;

```

```

END;
END;
END;

```

```

PROCEDURE TForm1.Cerrar1Click(Sender: TObject);
BEGIN
  close;
END;

```

```

PROCEDURE TForm1.Recuperar1Click(Sender: TObject);
BEGIN
  { WITH stringgrid1 DO
    BEGIN
      Cells[0,0]:=FLOATTOSTR(269);Cells[1,0]:=FLOATTOSTR(0);
      Cells[0,1]:=FLOATTOSTR(206.7);Cells[1,1]:=FLOATTOSTR(6.9);
      Cells[0,2]:=FLOATTOSTR(503.5);Cells[1,2]:=FLOATTOSTR(27.6);
      Cells[0,3]:=FLOATTOSTR(586.5);Cells[1,3]:=FLOATTOSTR(31);
      Cells[0,4]:=FLOATTOSTR(683.3);Cells[1,4]:=FLOATTOSTR(69);
    END;
    N:=5;}
END;

```

```

PROCEDURE TForm1.Borrar1Click(Sender: TObject);
BEGIN
  borrar;
END;

```

```

PROCEDURE TForm1.Nuevo1Click(Sender: TObject);
BEGIN
  borrar;
END;

```

```

PROCEDURE TForm1.BitBtn3Click(Sender: TObject);
VAR
  i,j:integer;
  cad:string;
BEGIN
  WITH form2 DO
    BEGIN
      richedit1.Lines.Add(#9+'COMPUTACION PARA INGENIERIA');
      richedit1.Lines.Add(#9+'-----'+
      '-----'+
      '-----'+#13);
      richedit1.Lines.Add(#9+'REPORTE DEL PROGRAMA DISEÑO DE CURVAS
      HORIZONTALES');
      richedit1.Lines.Add('
      '+#9+'N'+#9+'Progre.'+#9+'Radio'+#9+'Angulo'+#9+'Tangente'+

```

```

'Cuerda'+#9+'Media'+#9+'Externa'+#9+'L. Curva'+#9+'PCurva'+#9+'PTangencia');
richedit1.Lines.Add(#9+'-----'+
'-----'+
'-----'+#13);
FOR i:=0 TO N-1 DO
BEGIN
  cad:=' ';
  FOR j:=0 TO 10 DO
    cad:=cad+stringgrid2.cells[j,i]+#9;
    richedit1.Lines.Add(#9+cad);
  END;
richedit1.Lines.Add(#13+#13+#13+#13+#13+#13+#13+#13+#13+#13+#13+#13);
richedit1.Lines.Add(#9+'UNIVERSIDAD MAYOR DE SAN SIMON');
richedit1.Lines.Add(#9+'FACULTAD DE CIENCIAS Y TECNOLOGIA');
richedit1.Lines.Add(#9+'CARRERA DE INGENIERIA CIVIL'+#9+#9+
'Elaborado por: GERMAN CAMACHO - MAURICIO ANDIA');
form2.Show;
END;
END;

PROCEDURE TForm1.Creditos1Click(Sender: TObject);
BEGIN
  showmessage('UNIVERSIDAD MAYOR DE SAN SIMON'+ #13+
'FACULTAD DE CIENCIAS Y TECNOLOGIA'+ #13+
'CARRERA DE INGENIERIA CIVIL'+ #13+
'Programa realizado para uso Educacional.'+ #13+
'De Aplicación a la carrera de Ingeniería Civil'+#13+
'Carreteras I'+#13+
'Realizado por: German Camacho Ch. '+#13+
'      Mauricio Andia B. ');
END;

PROCEDURE TForm1.Guardar1Click(Sender: TObject);
VAR
i,j:integer;
cad:string;
BEGIN
Dir:=inputbox('ARCHIVO','RUTA Y NOMBRE','C:\FICHEROS\SALCURVAS.TXT');
AssignFile(F,Dir);
Rewrite(F);
  Writeln(F,#9,'COMPUTACION PARA INGENIERIA');
  Writeln(F,'-----'+
'-----');
  Writeln(F,#9,'DISEÑO DE CURVAS HORIZONTALES');
  Writeln(F,#9,' TABLA DE PROPIEDADES DE CURVAS HORIZONTALES');
  Writeln(F,'-----'+
'-----');

```

```

Writeln(F, ' ', 'N', #9, 'Progr.', #9, 'Radio', #9, 'Angulo', #9, 'Tangente'+
'Cuerda'+#9+'Media'+#9+'Externa'+#9+'LCurva'+#9+'PCurva'+#9+'PTangencia');
FOR i:=0 TO N-1 DO
  BEGIN
    cad:=' ';
    FOR j:=0 TO 10 DO
      cad:=cad+stringgrid2.cells[j,i]+#9;
    Writeln(F, cad);
  END;
Closefile(F);
showmessage('Archivo Guardado');
END;
END.

```

```

UNIT uCurvas_Horz2;

```

```

INTERFACE

```

```

USES

```

```

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, Buttons, XPMAN;

```

```

TYPE

```

```

  TForm2 = class(TForm)
    RichEdit1: TRichEdit;
    Label1: TLabel;
    XPManifest1: TXPManifest;
    BitBtn1: TBitBtn;
    PROCEDURE BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  END;

```

```

VAR

```

```

  Form2: TForm2;

```

```

IMPLEMENTATION

```

```

{$R *.dfm}

```

```

PROCEDURE TForm2.BitBtn1Click(Sender: TObject);

```

```

BEGIN

```

```

  Richedit1.Print('Impresion');

```

```

END;

```

```

END.

```

APLICACION 2.

APLICACION HIDRAULICA

El programa calcula las propiedades hidráulicas de una sección de canal, aplicable al diseño hidráulico de canales.

Para ello preparar el siguiente interfaz gráfico en Delphi fig. 2. Que consta de 2 formularios: 1 para la aplicación y el otro para impresión utilizando el componente QuickReport

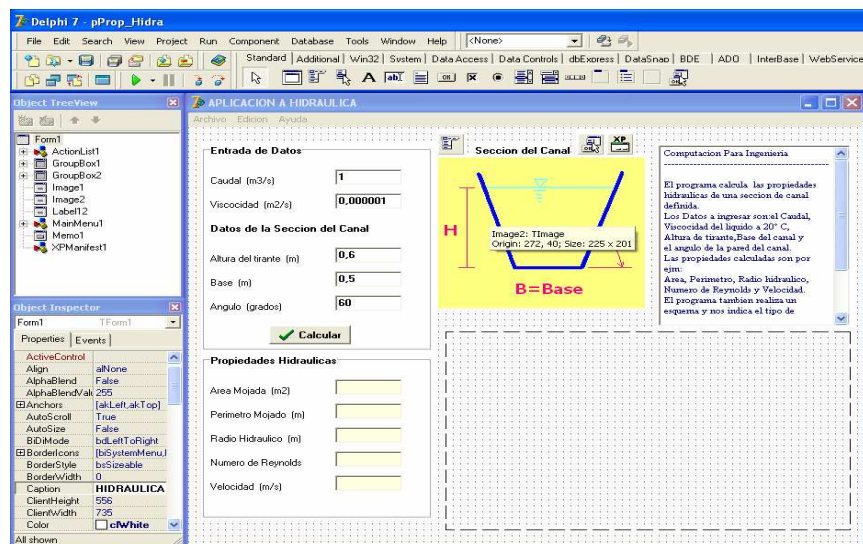
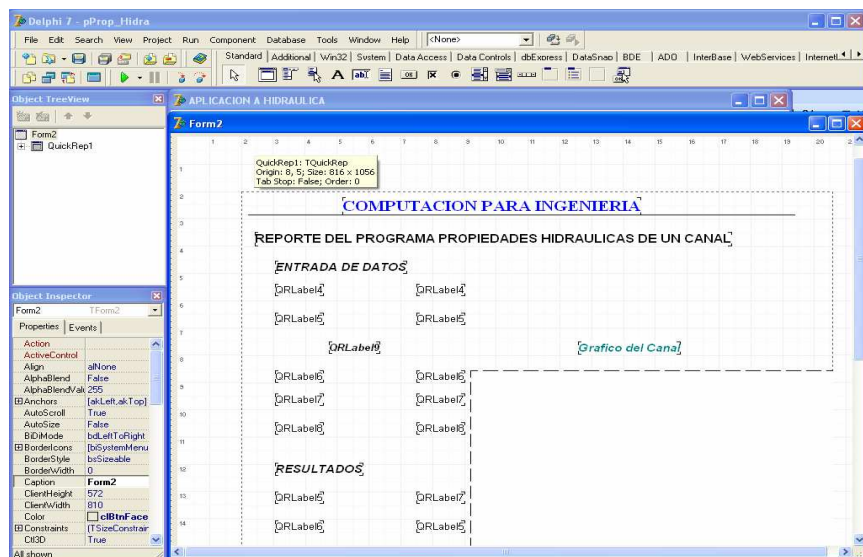


Fig. 2 Interfaz grafico.



UNIT uProp_Hidra;

INTERFACE

USES

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, Math, Buttons, ExtCtrls, jpeg, Menus, Printers, XPMAN,
ActnList, ExtActns;

TYPE

```
TForm1 = class(TForm)
  GroupBox1: TGroupBox;
  Edit1: TEdit;
  Edit2: TEdit;
  Edit3: TEdit;
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  Edit4: TEdit;
  Label4: TLabel;
  GroupBox2: TGroupBox;
  Label6: TLabel;
  Label7: TLabel;
  Label5: TLabel;
  Label8: TLabel;
  Label9: TLabel;
  Edit5: TEdit;
  Edit6: TEdit;
  Edit7: TEdit;
  Edit8: TEdit;
  Edit9: TEdit;
  Label10: TLabel;
  Label11: TLabel;
  Edit10: TEdit;
  BitBtn1: TBitBtn;
  Image1: TImage;
  Label12: TLabel;
  Image2: TImage;
  Memo1: TMemo;
  MainMenu1: TMainMenu;
  Archivo1: TMenuItem;
  Edicion1: TMenuItem;
  Propiedades1: TMenuItem;
  Ayuda1: TMenuItem;
  Nuevo1: TMenuItem;
  Borrar1: TMenuItem;
  Cargar1: TMenuItem;
```

```

N1: TMenuItem;
Cerrar1: TMenuItem;
Informacion1: TMenuItem;
Creditos1: TMenuItem;
Calcular1: TMenuItem;
Borrar2: TMenuItem;
Imprimir1: TMenuItem;
XPManifest1: TXPManifest;
ActionList1: TActionList;
FileRun1: TFileRun;
VistaPrevia1: TMenuItem;
ImprimirconQReport1: TMenuItem;
ConfigurarPagina1: TMenuItem;
PROCEDURE BitBtn1Click(Sender: TObject);
PROCEDURE Borrar2Click(Sender: TObject);
PROCEDURE Cerrar1Click(Sender: TObject);
PROCEDURE Nuevo1Click(Sender: TObject);
PROCEDURE Cargar1Click(Sender: TObject);
PROCEDURE Imprimir1Click(Sender: TObject);
PROCEDURE N1Click(Sender: TObject);
PROCEDURE Creditos1Click(Sender: TObject);
PROCEDURE ImprimirconQReport1Click(Sender: TObject);
PROCEDURE VistaPrevia1Click(Sender: TObject);
PROCEDURE ConfigurarPagina1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
END;

VAR
  Form1: TForm1;
  F:TextFile;
  Dirc:string;
  q,h,b,ang1,ang,vis:real;
  flu:string;
IMPLEMENTATION

USES uInformeHidra;

{$R *.dfm}

```

PROCEDURE Calcular;

VAR

a,p,rh,v,re:real;

BEGIN

WITH form1 **DO**

BEGIN

q:=STRTOFLOAT(Edit1.text);

vis:=STRTOFLOAT(Edit10.text);

h:=STRTOFLOAT(Edit2.text);

b:=STRTOFLOAT(Edit3.text);

ang1:=STRTOFLOAT(Edit4.text);

ang:=degtorad(ang1);

IF (ang1<=90) **AND** (ang1>0) **THEN**

BEGIN

IF ang1<>90 **THEN**

BEGIN

a:=sqr(h)/(2*tan(ang))+b*h;

p:=2*h/sin(ang)+b;

END

ELSE

a:=h*b;edit5.Text:=format('%0:3f3',[a]);

p:=2*h+b;edit6.Text:=format('%0:3f3',[p]);

rh:=a/p;edit7.Text:=format('%0:3f3',[rh]);

v:=q/a;edit9.Text:=format('%0:3f3',[v]);

re:=4*v*rh/vis;edit8.Text:=format('%0:3f3',[re]);

END;

END;

IF re <=2000 **THEN** flu:='Flujo Laminar';

IF (re>2000) **AND** (re<=4000) **THEN** flu:='Flujo en Transición';

IF re>4000 **THEN** flu:='Flujo Turbulento';

END;

PROCEDURE borrarimagen;

VAR

ARect: TRect;

BEGIN

WITH form1.Image1.Canvas **DO**

BEGIN

CopyMode := cmWhiteness;

ARect := Rect(0, 0, form1.Image1.Width, form1.Image1.Height);

CopyRect(ARect, form1.Image1.Canvas, ARect);

END;

END;

```

PROCEDURE dibujar;
VAR
x,wx:real;
x1,h1,b1,xw,xo,yo:integer;
BEGIN
  borrarimagen;
  IF angl=90 THEN
    BEGIN
      x:=0;wx:=0;
    END
  ELSE
    BEGIN
      x:=h/tan(ang);wx:=h/(6*tan(ang));
    END;
  h1:=trunc(h*200);b1:=trunc(b*200);x1:=trunc(x*200);xw:=round(wx*200);
  WITH form1 DO
    BEGIN
      IF (image1.Width)<(2*x1+b1+10) THEN
        BEGIN
          h1:=trunc(h*100);b1:=trunc(b*100);x1:=trunc(x*100);xw:=round(wx*100);
        END;
        xo:=trunc((image1.Width-(2*x1+b1+10))/2);
        yo:=trunc((image1.Height-h1-40)/2);
        image1.Canvas.Pen.Color:=clblack;
        image1.Canvas.Pen.Width:=5;
        image1.Canvas.MoveTo(xo,yo);
        image1.Canvas.LineTo(xo+x1,yo+h1);
        image1.Canvas.LineTo(xo+x1+b1,yo+h1);
        image1.Canvas.LineTo(xo+2*x1+b1,yo);
        image1.Canvas.Pen.Color:=clblue;
        image1.Canvas.Pen.Width:=2;
        image1.Canvas.MoveTo(xo+xw-1,yo+10);
        image1.Canvas.LineTo(xo+2*x1+b1-xw+1,yo+10);
        image1.Canvas.TextOut(trunc((xo+2*x1+b1)/2),yo+h1+7,'H='+FLOATTOSTR(h));
        image1.Canvas.TextOut(trunc((xo+2*x1+b1)/2),yo+h1+22,'B='+FLOATTOSTR(b));

        image1.Canvas.TextOut(trunc((xo+2*x1+b1)/2),yo+h1+37,'Ang='+FLOATTOSTR(ang1)
        );
        image1.Canvas.TextOut(trunc((xo+2*x1+b1)/2),yo+h1+52,flu);
      END;
    END;
  END;

```

PROCEDURE asignar;

BEGIN

WITH form2 **DO**

BEGIN

qrlabel4.Caption:=form1.label1.Caption;qrlabel10.Caption:=form1.edit1.text;
qrlabel5.Caption:=form1.label11.Caption;qrlabel11.Caption:=form1.edit10.text;
qrlabel6.Caption:=form1.label2.Caption;qrlabel12.Caption:=form1.edit2.text;
qrlabel7.Caption:=form1.label3.Caption;qrlabel13.Caption:=form1.edit3.text;
qrlabel8.Caption:=form1.label4.Caption;qrlabel14.Caption:=form1.edit4.text;
qrlabel9.Caption:=form1.label10.Caption;
qrlabel23.Caption:=form1.label7.Caption;qrlabel16.Caption:=form1.edit5.text;
qrlabel24.Caption:=form1.label6.Caption;qrlabel17.Caption:=form1.edit6.text;
qrlabel25.Caption:=form1.label8.Caption;qrlabel18.Caption:=form1.edit7.text;
qrlabel26.Caption:=form1.label5.Caption;qrlabel19.Caption:=form1.edit8.text;
qrlabel27.Caption:=form1.label9.Caption;qrlabel20.Caption:=form1.edit9.text;
qrimage1.Picture:=form1.Image1.Picture;

END;

END;

PROCEDURE TForm1.BitBtn1Click(Sender: TObject);

BEGIN

Calcular;

dibujar;

END;

PROCEDURE TForm1.Borrar2Click(Sender: TObject);

BEGIN

borrarimagen;

edit5.Clear;edit6.Clear;edit7.Clear;

edit8.Clear;edit9.Clear;

END;

PROCEDURE TForm1.Cerrar1Click(Sender: TObject);

BEGIN

close;

END;

PROCEDURE TForm1.Nuevo1Click(Sender: TObject);

BEGIN

edit1.Clear;edit2.Clear;edit3.Clear;

edit4.Clear;edit5.Clear;edit6.Clear;

edit7.Clear;edit8.Clear;edit9.Clear;

edit10.Clear;

borrarimagen;

END;

PROCEDURE TForm1.Cargar1Click(Sender: TObject);

BEGIN

dit1.Text:='1';

edit2.Text:='0,6';

edit3.Text:='0,5';

edit4.Text:='60';

edit10.Text:='1E-6';

END;

PROCEDURE TForm1.Imprimir1Click(Sender: TObject);

BEGIN

WITH Printer **DO**

BEGIN

BeginDoc;

Canvas.Brush.Style := bsClear;

Canvas.TextHeight('10');

Canvas.TextOut(400,400,'COMPUTACION PARA INGENIERIA');

Canvas.MoveTo(400,500);

Canvas.LineTo(4000,500);

Canvas.TextOut(400,600,'REPORTE DEL PROGRAMA PROPIEDADES
HIDRAULICAS');

Canvas.TextOut(400,800,'ENTRADA DE DATOS');

Canvas.TextOut(400,1000, label1.Caption+' '+edit1.Text);

Canvas.TextOut(400,1200, label11.Caption+' '+edit10.Text);

Canvas.TextOut(400,1400, label2.Caption+' '+edit2.Text);

Canvas.TextOut(400,1600, label3.Caption+' '+edit3.Text);

Canvas.TextOut(400,1800, label4.Caption+' '+edit4.Text);

Canvas.TextOut(400,2200,'RESULTADOS');

Canvas.TextOut(400,2400, label7.Caption+' '+edit5.Text);

Canvas.TextOut(400,2600, label6.Caption+' '+edit6.Text);

Canvas.TextOut(400,2800, label8.Caption+' '+edit7.Text);

Canvas.TextOut(400,3000, label5.Caption+' '+edit8.Text);

Canvas.TextOut(400,3200, label9.Caption+' '+edit9.Text);

Canvas.TextHeight('8');

Canvas.TextOut(400,6000,' UNIVERSIDAD MAYOR DE SAN SIMON');

Canvas.TextOut(400,6100,' FACULTAD DE CIENCIAS Y TECNOLOGIA');

Canvas.TextOut(400,6200,' CARRERA DE INGENIERIA CIVIL');

Canvas.TextOut(2500,6200,' Elaborado por: GERMAN CAMACHO -
MAURICIO ANDIA');

EndDoc;

END;

END;

```

PROCEDURE TForm1.N1Click(Sender: TObject);
BEGIN
Dir:=inputbox('ARCHIVO','RUTA Y
NOMBRE','C:\FICHEROS\SALIDAHIDRA.TXT');
AssignFile(F,Dir);
Rewrite(F);
Writeln(F,#9,'COMPUTACION PARA INGENIERIA');
Writeln(F,'-----');
Writeln(F,' ');
Writeln(F,'CALCULO DE PROPIEDADES HIDRAULICAS DE UN CANAL');
Writeln(F,' ');
Writeln(F,#9,' DATOS DE ENTRADA');
Writeln(F,#9,' -----');
Writeln(F,label1.caption,' = ',edit1.text);
Writeln(F,label11.caption,' = ',edit10.text);
Writeln(F,label2.caption,' = ',edit2.text);
Writeln(F,label3.caption,' = ',edit3.text);
Writeln(F,label4.caption,' = ',edit4.text);
Writeln(F,' ');
Writeln(F,#9,' SALIDA DE RESULTADOS');
Writeln(F,#9,' -----');
Writeln(F,label7.caption,' = ',edit5.text);
Writeln(F,label6.caption,' = ',edit6.text);
Writeln(F,label8.caption,' = ',edit7.text);
Writeln(F,label5.caption,' = ',edit8.text);
Writeln(F,label9.caption,' = ',edit9.text);
Writeln(F,flu);
Closefile(F);
showmessage('Archivo Guardado');

END;

```

```

PROCEDURE TForm1.Creditos1Click(Sender: TObject);
BEGIN
showmessage('UNIVERSIDAD MAYOR DE SAN SIMON'+ #13+
'FACULTAD DE CIENCIAS Y TECNOLOGIA'+ #13+
'CARRERA DE INGENIERIA CIVIL'+ #13+
'Programa realizado para uso Educacional.'+ #13+
'De Aplicación a la carrera de Ingeniería Civil'+#13+
'En el area de Hidraulica'+#13+
'Realizado por: German Camacho Ch. '+#13+
'          Mauricio Andia B. ');
END;

```

```
PROCEDURE TForm1.ImprimirconQReport1Click(Sender: TObject);  
BEGIN  
    asignar;  
    form2.QuickRep1.Print;  
END;
```

```
PROCEDURE TForm1.VistaPrevia1Click(Sender: TObject);  
BEGIN  
    asignar;  
    form2.QuickRep1.Preview;  
END;
```

```
PROCEDURE TForm1.ConfigurarPagina1Click(Sender: TObject);  
BEGIN  
    asignar;  
    form2.QuickRep1.PrinterSetup;  
END;  
END.
```

```
UNIT uInformeHidra;
```

INTERFACE

USES

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, QRCtrls, QuickRpt, ExtCtrls;

TYPE

```
TForm2 = class(TForm)  
    QuickRep1: TQuickRep;  
    QRBand1: TQRBand;  
    QRLabel1: TQRLabel;  
    QRShape1: TQRShape;  
    QRLabel2: TQRLabel;  
    QRLabel3: TQRLabel;  
    QRLabel4: TQRLabel;  
    QRLabel10: TQRLabel;  
    QRLabel11: TQRLabel;  
    QRLabel5: TQRLabel;  
    QRLabel6: TQRLabel;  
    QRLabel12: TQRLabel;  
    QRLabel13: TQRLabel;  
    QRLabel7: TQRLabel;  
    QRLabel8: TQRLabel;  
    QRLabel14: TQRLabel;  
    QRLabel9: TQRLabel;
```



```
QRLabel22: TQRLabel;  
QRLabel23: TQRLabel;  
QRLabel17: TQRLabel;  
QRLabel18: TQRLabel;  
QRLabel24: TQRLabel;  
QRLabel19: TQRLabel;  
QRLabel25: TQRLabel;  
QRLabel26: TQRLabel;  
QRLabel16: TQRLabel;  
QRLabel20: TQRLabel;  
QRLabel27: TQRLabel;  
QRLabel15: TQRLabel;  
QRLabel21: TQRLabel;  
QRLabel28: TQRLabel;  
QRLabel29: TQRLabel;  
QRImage1: TQRImage;  
QRLabel30: TQRLabel;
```

```
private  
  { Private declarations }  
public  
  { Public declarations }  
END;
```

VAR

```
Form2: TForm2;
```

IMPLEMENTATION

```
{ $R *.dfm }
```

END.

APLICACION 3.

APLICACION HORMIGON ARMADO

El programa realiza el cálculo del acero requerido de una sección sometida a flexión, para resistir el momento actuante en la viga. De aplicación al diseño vigas de hormigón armado. Entonces se debe preparar el siguiente interfaz gráfico en Delphi fig. 3.

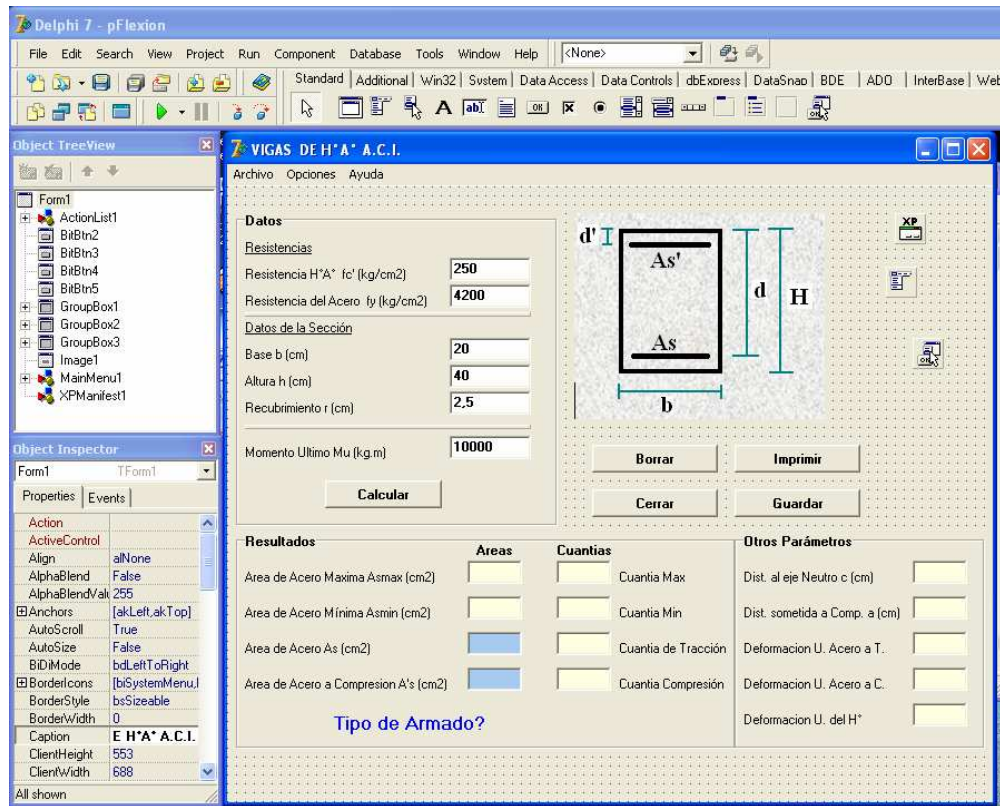


Fig. 3 Interfaz grafico.

UNIT uFlexion;

INTERFACE

USES

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, Buttons, jpeg, ExtCtrls, Printers, Menus, XPMAN,
ActnList, ExtActns;

TYPE

```
TForm1 = class(TForm)
  GroupBox1: TGroupBox;
  Emu: TEdit;
  Eb: TEdit;
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  Label4: TLabel;
  Eh: TEdit;
  Er: TEdit;
  Label5: TLabel;
  BitBtn1: TBitBtn;
  GroupBox2: TGroupBox;
  Label6: TLabel;
  Eamax: TEdit;
  Label7: TLabel;
  Label8: TLabel;
  Label9: TLabel;
  Easp: TEdit;
  Eas: TEdit;
  Eamin: TEdit;
  Label10: TLabel;
  Eromax: TEdit;
  Eromin: TEdit;
  Ero: TEdit;
  Erop: TEdit;
  Label11: TLabel;
  Label12: TLabel;
  Label13: TLabel;
  Label14: TLabel;
  GroupBox3: TGroupBox;
  Label15: TLabel;
  Ec: TEdit;
  Label16: TLabel;
  Ea: TEdit;
  Eesp: TEdit;
  Ees: TEdit;
```

```

Label17: TLabel;
Label18: TLabel;
Ltip: TLabel;
Label19: TLabel;
Efc: TEdit;
Efy: TEdit;
Label20: TLabel;
Label21: TLabel;
BitBtn2: TBitBtn;
BitBtn3: TBitBtn;
BitBtn4: TBitBtn;
BitBtn5: TBitBtn;
Image1: TImage;
Panel1: TPanel;
Panel2: TPanel;
Eco: TEdit;
Label22: TLabel;
Label23: TLabel;
MainMenu1: TMainMenu;
Archivo1: TMenuItem;
Opciones1: TMenuItem;
Ayuda1: TMenuItem;
Informacion1: TMenuItem;
Creditos1: TMenuItem;
Nuevo1: TMenuItem;
Guardar1: TMenuItem;
Imprimir1: TMenuItem;
N1: TMenuItem;
Cerrar1: TMenuItem;
Calcular1: TMenuItem;
Borrar1: TMenuItem;
Recuperar1: TMenuItem;
XPManifest1: TXPManifest;
ActionList1: TActionList;
FileRun1: TFileRun;
PROCEDURE BitBtn1Click(Sender: TObject);
PROCEDURE BitBtn4Click(Sender: TObject);
PROCEDURE BitBtn2Click(Sender: TObject);
PROCEDURE BitBtn3Click(Sender: TObject);
PROCEDURE BitBtn5Click(Sender: TObject);
PROCEDURE Nuevo1Click(Sender: TObject);
PROCEDURE Recuperar1Click(Sender: TObject);
PROCEDURE Creditos1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }

```

```

END;
CONST Mdes=2040000;
CONST econ=0.003;
VAR
  Form1: TForm1;
  F:TextFile;
  Dirc:String;
  fc,fy,h,b,d,dp,Mu,r:real;
  c,a,es,esp,ey:real;
  asmax,asmin,ars,asp,fi,fs,romax,romin,ro,rop,b1:real;

```

IMPLEMENTATION

```

{$R *.dfm}
PROCEDURE SimpleArmado;
BEGIN
  WITH form1 DO
    BEGIN
      a:=ars*fy/(0.85*fc*b);ea.Text:=format('%0:3f3',[a]);
      c:=a/b1;ec.Text:=format('%0:3f3',[c]);
      es:=0.003*d/c-0.003;ees.Text:=FLOATTOSTR(es);
      ro:=ars/(b*d);ero.Text:=FLOATTOSTR(ro);
      eamax.Text:=format('%0:3f3',[asmax]);
      eamin.Text:=format('%0:3f3',[asmin]);
      IF ars<asmin THEN ars:=asmin;
      eas.Text:=format('%0:3f3',[ars]);
      eromax.Text:=FLOATTOSTR(romax);
      eromin.Text:=FLOATTOSTR(romin);
      ltip.Caption:='VIGA SIMPLEMENTE ARMADA';
      eco.Text:=FLOATTOSTR(econ);
    END;
  END;
PROCEDURE DobleArmado;
VAR
  difars:real;
BEGIN
  WITH form1 DO
    BEGIN
      es:=0.005;ees.Text:=FLOATTOSTR(es);
      c:=3*d/8;ec.Text:=format('%0:3f3',[c]);
      a:=b1*c;ea.Text:=format('%0:3f3',[a]);
      esp:=econ*(c-dp)/c;eesp.Text:=FLOATTOSTR(esp);
      fs:=esp*mde;eco.Text:=FLOATTOSTR(econ);
      IF fs>fy THEN fs:=fy;
      difars:=a*0.85*fc*b/fy;
      asp:=(mu/fi-0.85*fc*b*a*(d-a/2))/(fs*(d-dp));
      rop:=asp/(b*d);erop.Text:=FLOATTOSTR(rop);
      ars:=asp+difars;eas.Text:=format('%0:3f3',[ars]);
    END;
  END;

```

```

romax:=3/7*0.85*b1*fc/fy+rop*fs/fy;eromax.Text:=FLOATTOSTR(romax);
asmax:=romax*b*d;eamax.Text:=format('%0:3f3',[asmax]);
IF asp<asmin THEN asp:=asmin;
easp.Text:=format('%0:3f3',[asp]);
eamin.Text:=format('%0:3f3',[asmin]);
eromin.Text:=FLOATTOSTR(romin);
eamin.Text:=format('%0:3f3',[asmin]);
ero.Text:=FLOATTOSTR(ars/(b*d));
ltip.Caption:='VIGA DOBLEMENTE ARMADA';
eas.Font.Size:=10;easp.Font.Size:=10;
END;
END;
PROCEDURE TForm1.BitBtn1Click(Sender: TObject);
VAR
asum:real;
cont:integer;
LABEL
err;
BEGIN
fy:=STRTOFLOAT(efy.Text);
fc:=STRTOFLOAT(efc.Text);
h:=STRTOFLOAT(eh.Text);
b:=STRTOFLOAT(eb.Text);
r:=STRTOFLOAT(er.Text);
mu:=100*STRTOFLOAT(emu.Text);
d:=h-r-0.8-1;dp:=h-d;
ey:=fy/mdes;
IF fc<=280 THEN b1:=0.85 ELSE b1:=0.85-0.05*(fc-280)/70;
romax:=3*0.85*b1*fc/(7*fy);
asmax:=romax*b*d;
romin:=14/fy;
asmin:=romin*b*d;
fi:=0.9;ars:=1; cont:=0;
REPEAT
cont:=cont+1;
asum:=ars;
ars:=mu/(fi*fy*(d-ars*fy/(2*0.85*fc*b)));
IF cont=20 THEN break;
UNTIL abs(asum-ars)<=0.001;
IF cont=20 THEN goto err;
IF ars<asmax THEN
SimpleArmado
ELSE
BEGIN
err:
DobleArmado;
END;

```

END;

PROCEDURE TForm1.BitBtn4Click(Sender: TObject);

BEGIN

close;

END;

PROCEDURE TForm1.BitBtn2Click(Sender: TObject);

BEGIN

eamax.Clear;

eamin.Clear;

eas.Clear;

easp.Clear;

ero.Clear;

eromax.Clear;

eromin.Clear;

erop.Clear;

eesp.Clear;

ea.Clear;

ec.Clear;

eco.Clear;

ees.Clear;

ltip.Caption:='PRESIONE EL BOTON CALCULAR';

END;

PROCEDURE TForm1.BitBtn3Click(Sender: TObject);

BEGIN

WITH Printer **DO**

BEGIN

BeginDoc;

Canvas.Brush.Style := bsClear;

Canvas.TextHeight('10');

Canvas.TextOut(400,400,'COMPUTACION PARA INGENIERIA');

Canvas.MoveTo(400,500);

Canvas.LineTo(4000,500);

Canvas.TextOut(400,600,'REPORTE DEL PROGRAMA DISEÑO DE VIGAS(A.C.I.-2002)');

Canvas.TextOut(400,800 , 'ENTRADA DE DATOS');

Canvas.TextOut(400,1000 ,label20.Caption+' = '+efc.Text);

Canvas.TextOut(400,1200 ,label21.Caption+' = '+efy.Text);

Canvas.TextOut(400,1400 ,label2.Caption+' = '+eb.Text);

Canvas.TextOut(400,1600 ,label3.Caption+' = '+eh.Text);

Canvas.TextOut(400,1800 ,label4.Caption+' = '+er.Text);

Canvas.TextOut(400,2000 ,label1.Caption+' = '+emu.Text);

Canvas.TextOut(400,2200 , 'RESULTADOS');

Canvas.TextOut(400,2400 ,label7.Caption+' = '+eamax.Text);

Canvas.TextOut(400,2600 ,label8.Caption+' = '+eamin.Text);

```

Canvas.TextOut(400,2800 ,label6.Caption+' = '+eas.Text );
Canvas.TextOut(400,3000 ,label9.Caption+' = '+easp.Text );
Canvas.TextOut(2500,2400 ,label15.Caption+' = '+ec.Text );
Canvas.TextOut(2500,2600 ,label16.Caption+' = '+ea.Text );
Canvas.TextOut(2500,2800 ,label17.Caption+' = '+ees.Text );
Canvas.TextOut(2500,3000 ,label18.Caption+' = '+eesp.Text );
Canvas.TextOut(2500,3200 ,label22.Caption+' = '+eco.Text );
Canvas.TextHeight('8');
Canvas.TextOut(400,6000,' UNIVERSIDAD MAYOR DE SAN SIMON');
Canvas.TextOut(400,6100,' FACULTAD DE CIENCIAS Y TECNOLOGIA');
Canvas.TextOut(400,6200,' CARRERA DE INGENIERIA CIVIL');
Canvas.TextOut(2500,6200,' Elaborado por: GERMAN CAMACHO - MAURICIO
ANDIA');

```

```

EndDoc;
END;
END;

```

PROCEDURE TForm1.BitBtn5Click(Sender: TObject);

BEGIN

Dir:=**inputbox**('ARCHIVO','RUTA Y NOMBRE','C:\FICHEROS\SALIDA.TXT');

AssignFile(F,Dir);

Rewrite(F);

Writeln(F,#9,'COMPUTACION PARA INGENIERIA');

Writeln(F,'-----');

Writeln(F,#9,'DISEÑO DE VIGAS (A.C.I.-2002)');

Writeln(F,#9,' DATOS DE ENTRADA');

Writeln(F,#9,' -----');

Writeln(F,label20.caption,' = ',efc.text);

Writeln(F,label21.caption,' = ',efy.text);

Writeln(F,label2.caption,' = ',eb.text);

Writeln(F,label3.caption,' = ',eh.text);

Writeln(F,label4.caption,' = ',er.text);

Writeln(F,label11.caption,' = ',emu.text);

Writeln(F,#9,' SALIDA DE RESULTADOS');

Writeln(F,#9,' -----');

Writeln(F,label7.caption,' = ',eamax.text);

Writeln(F,label8.caption,' = ',eamin.text);

Writeln(F,label6.caption,' = ',eas.text);

Writeln(F,label9.caption,' = ',easp.text);

Writeln(F,label15.caption,' = ',ec.text);

Writeln(F,label16.caption,' = ',ea.text);

Writeln(F,label17.caption,' = ',ees.text);

Writeln(F,label18.caption,' = ',eesp.text);

Writeln(F,label22.caption,' = ',eco.text);

Closefile(F);

showmessage('Archivo Guardado');

END;

PROCEDURE TForm1.Nuevo1Click(Sender: TObject);

BEGIN

efc.Clear;efy.Clear;eb.Clear;

eh.Clear;er.Clear;emu.Clear;

eamax.Clear;eamin.Clear;eas.Clear;

easp.Clear;ero.Clear;eromax.Clear;

eromin.Clear;erop.Clear;eesp.Clear;

ea.Clear;ec.Clear;eco.Clear;ees.Clear;

ltip.Caption:='INGRESE DATOS Y LUEGO PRESIONE CALCULAR';

END;

PROCEDURE TForm1.Recuperar1Click(Sender: TObject);

BEGIN

efc.Text:='250';efy.Text:='4200';

eb.Text:='20';eh.Text:='40';

er.Text:='2,5';emu.Text:='10000';

END;

PROCEDURE TForm1.Creditos1Click(Sender: TObject);

BEGIN

showmessage('UNIVERSIDAD MAYOR DE SAN SIMON'+ #13+
 'FACULTAD DE CIENCIAS Y TECNOLOGIA'+ #13+
 'CARRERA DE INGENIERIA CIVIL'+ #13+
 'Programa realizado para uso Educacional.'+ #13+
 'De Aplicación a la carrera de Ingeniería Civil'+#13+
 'A plicado a Hormigon Armado'+#13+
 'Realizado por: German Camacho Ch. '+#13+
 ' Mauricio Andia B.');

END;

END.

APLICACION 4.

APLICACION ROCAS

El programa determina las características geomecánicas del macizo rocoso, aplicando la teoría de los mínimos cuadrados, utilizando los criterios de Mohr – Coulomb y Hoek – Brown.

Entonces se debe preparar el siguiente interfaz gráfico en Delphi fig.4.

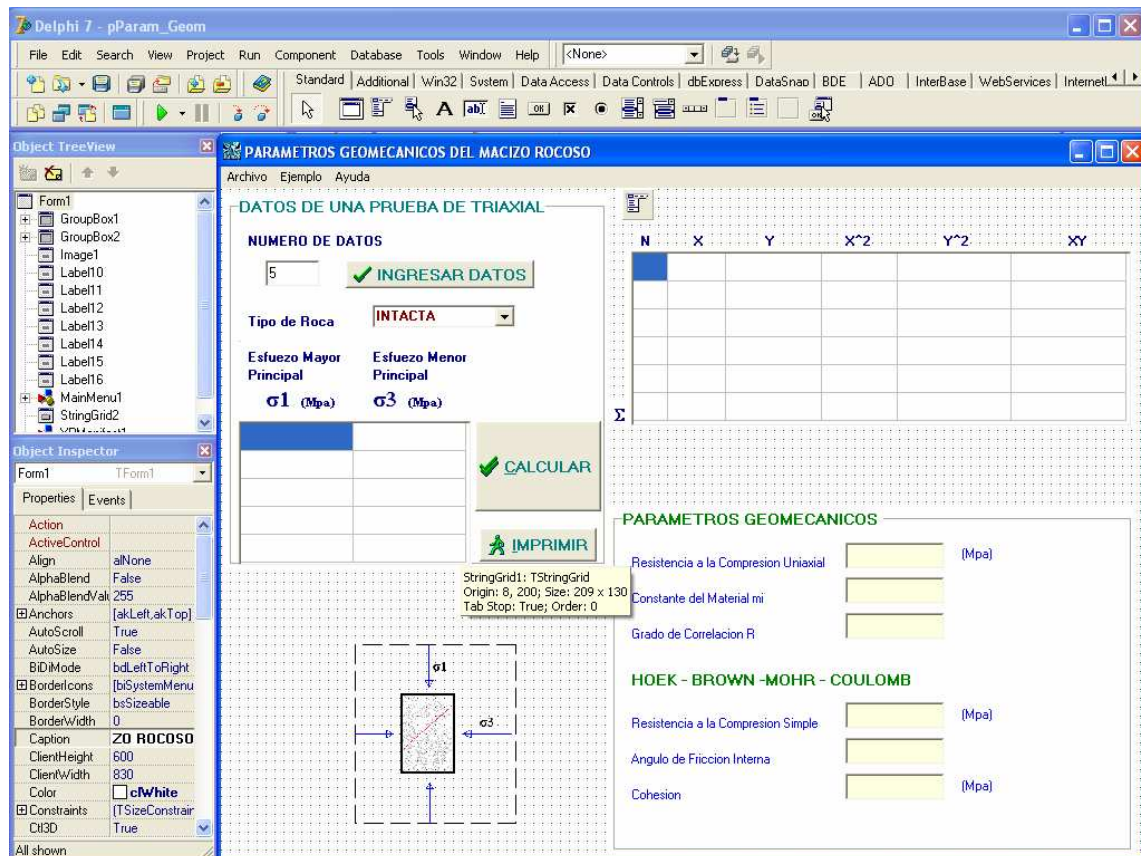


Fig. 4 Interfaz grafico.

UNIT uParam_Geom;

INTERFACE

USES

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, Grids, Buttons, ExtCtrls, Math, QuickRpt, QRCtrlrs,
XPMAN, Menus, Printers;

TYPE

```
TForm1 = class(TForm)
  GroupBox1: TGroupBox;
  StringGrid1: TStringGrid;
  Label1: TLabel;
  Label2: TLabel;
  Label3: TLabel;
  Label4: TLabel;
  Label5: TLabel;
  Label6: TLabel;
  Label7: TLabel;
  Label8: TLabel;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  Edit1: TEdit;
  Label9: TLabel;
  Label10: TLabel;
  Label11: TLabel;
  Label12: TLabel;
  Label13: TLabel;
  Label14: TLabel;
  Label15: TLabel;
  StringGrid2: TStringGrid;
  Label16: TLabel;
  BitBtn3: TBitBtn;
  GroupBox2: TGroupBox;
  Edit2: TEdit;
  Label17: TLabel;
  Edit3: TEdit;
  Label18: TLabel;
  Label19: TLabel;
  Label20: TLabel;
  Edit4: TEdit;
  Label21: TLabel;
  ComboBox1: TComboBox;
  Label22: TLabel;
  Bevel1: TBevel;
  Edit5: TEdit;
```

```

Label23: TLabel;
Label24: TLabel;
Edit6: TEdit;
Label25: TLabel;
Edit7: TEdit;
Label26: TLabel;
Label27: TLabel;
Image1: TImage;
Label28: TLabel;
XPManifest1: TXPManifest;
MainMenu1: TMainMenu;
Archivo1: TMenuItem;
Ejemplo1: TMenuItem;
Ayuda1: TMenuItem;
Informacion1: TMenuItem;
Creditos1: TMenuItem;
Recuperar1: TMenuItem;
Borrar1: TMenuItem;
Calcular1: TMenuItem;
Nuevo1: TMenuItem;
Guardar1: TMenuItem;
N1: TMenuItem;
Cerrar1: TMenuItem;
Imprimir1: TMenuItem;
PROCEDURE Edit1KeyPress(Sender: TObject; VAR Key: Char);
PROCEDURE BitBtn1Click(Sender: TObject);
PROCEDURE BitBtn2Click(Sender: TObject);
PROCEDURE Cerrar1Click(Sender: TObject);
PROCEDURE Recuperar1Click(Sender: TObject);
PROCEDURE Borrar1Click(Sender: TObject);
PROCEDURE Nuevo1Click(Sender: TObject);
PROCEDURE BitBtn3Click(Sender: TObject);
PROCEDURE Creditos1Click(Sender: TObject);
PROCEDURE Guardar1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
END;
Mattipo=ARRAY[1..20,1..20] OF real;
VAR
  Form1: TForm1;
  F:TextFile;
  Dirc:string;
  N,tam:integer;
  mat1,mat2:Mattipo;
IMPLEMENTATION

```

```
{ $R *.dfm }
{ PROCEDURE MinCuad2(sender:tstringgrid;Num:integer;a:Mattipo);
BEGIN
  sender.cells[0,1]:='78';
END;}
```

```
PROCEDURE TForm1.Edit1KeyPress(Sender: TObject; VAR Key: Char);
BEGIN
  IF NOT (key IN ['0'..'9',#8 {backsapace}]) THEN
    BEGIN
      key:=#0;
      beep;
    END;
  END;
PROCEDURE Borrar;
VAR i,j:integer;
BEGIN
  WITH form1.StringGrid1 DO
    BEGIN
      FOR i:=0 TO N-1 DO
        BEGIN
          cells[0,i]:=' ';cells[1,i]:=' ';
        END;
      END;
    WITH form1.StringGrid2 DO
      BEGIN
        FOR j:=0 TO 5 DO
          BEGIN
            FOR i:=0 TO N DO
              cells[j,i]:=' ';
            END;
          END;
        END;
      END;
    END;
```

```
PROCEDURE TForm1.BitBtn1Click(Sender: TObject);
VAR hi:integer;
BEGIN
  showmessage('Ingresar Datos de Esfuerzos Principales'+#13+
    '¡¡¡ Como Minimo 5 pares de datos');
  N:=STRTOINT(edit1.Text);
  WITH StringGrid1 DO
    BEGIN
      {INICIALIZANDO}
      rowcount:=N;
      {AJUSTE DEL TAMAÑO DE LA GRIDA}
      hi:=height;
      height:=((defaultrowheight+1)*N+5);
```

```

    GroupBox1.Height:=GroupBox1.Height+height-hi;
    enabled:=true;SetFocus;
END;
    borrar;

```

END;

PROCEDURE TForm1.BitBtn2Click(Sender: TObject);

VAR

```

i,hi:integer;
r,s1,s3,sx,sy,sx2,sy2,sxy,sci,mi:real;
val,s1c,s3c,s:real;
sux,suy,sux2,suy2,suxy:real;
a,b,scmc,k,angrad,angdeg,c:real;

```

BEGIN

WITH StringGrid2 **DO**

BEGIN

{INICIALIZANDO}

rowcount:=N+1;

{AJUSTE DEL TAMAÑO DE LA GRIDA}

hi:=height;

height:=((defaultrowheight+1)*(N+1)+5);

label16.Top:=label16.Top+height-hi;

sx:=0;sy:=0;sx2:=0;sy2:=0;sxy:=0;

FOR i:=0 **TO** N-1 **DO**

BEGIN

s1:=**STRTOFLOAT**(stringgrid1.Cells[0,i]);

s3:=**STRTOFLOAT**(stringgrid1.Cells[1,i]);

cells[0,i]:=inttostr(i+1);

cells[1,i]:=FLOATTOSTR(s3);sx:=sx+s3;

cells[2,i]:= FLOATTOSTR(sqr(s1-s3));sy:=sy+(sqr(s1-s3));

cells[3,i]:= FLOATTOSTR(sqr(s3));sx2:=sx2+sqr(s3);

cells[4,i]:=FLOATTOSTR(sqr(sqr(s1-s3)));sy2:=sy2+sqr(sqr(s1-s3));

cells[5,i]:=FLOATTOSTR(s3*sqr(s1-s3));sxy:=sxy+s3*sqr(s1-s3);

END;

cells[0,N]:=inttostr(N+1);

cells[1,N]:=FLOATTOSTR(sx);

cells[2,N]:=FLOATTOSTR(sy);

cells[3,N]:=FLOATTOSTR(sx2);

cells[4,N]:=FLOATTOSTR(sy2);

cells[5,N]:=FLOATTOSTR(sxy);

END;

sci:=sqrt(sy/N-((sxy-sx*sy/N)/(sx2-(sqr(sx))/N))*sx/N);

edit2.Text:=format('%0:3f',[sci]);

mi:=((sxy-sx*sy/N)/(sx2-(sqr(sx))/N))/sci;

edit3.Text:=format('%0:3f',[mi]);

r:=(sqr(sxy-sx*sy/N))/((sx2-(sqr(sx))/N)*(sy2-(sqr(sy))/N));

```

edit4.Text:=format('%0:3f',[r]);
IF (r>=0.9) AND (r<=1) THEN label28.Caption:='Existe una Buena'+#13+ 'Correlación';
//CRITERIO HOEK-BROWN Y MOHR-COULOMB
s:=1;
val:=0;
sux:=0;suy:=0;suxy:=0;sux2:=0;suy2:=0;
FOR i:=1 TO 11 DO
BEGIN
    mat1[i,1]:=0+val; // X
    sux:=sux+mat1[i,1];
    mat1[i,2]:=mat1[i,1]+sci*sqrt(mi*mat1[i,1]/sci+s); // Y
    suy:=suy+mat1[i,2];
    mat1[i,3]:=mat1[i,1]*mat1[i,2]; // XY
    suxy:=suxy+mat1[i,3];
    mat1[i,4]:=sqr(mat1[i,1]); // X^2
    sux2:=sux2+mat1[i,4];
    mat1[i,5]:=sqr(mat1[i,2]); // Y^2
    suy2:=suy2+mat1[i,5];
    val:=val+0.1;
END;
a:=(suy*sux2-sux*suxy)/(11*sux2-sqr(sux));
edit5.Text:=format('%0:3f',[a]);
b:=(11*suxy-sux*suy)/(11*sux2-sqr(sux));
angrad:=arcsin((b-1)/(b+1));
angdeg:=radtodeg(angrad);
edit6.Text:=format('%0:3f',[angdeg]);
c:=a*(1-sin(angrad))/(2*cos(angrad));
edit7.Text:=format('%0:3f',[c]);
END;

PROCEDURE TForm1.Cerrar1Click(Sender: TObject);
BEGIN
    close;
END;

PROCEDURE TForm1.Recuperar1Click(Sender: TObject);
BEGIN
    WITH stringgrid1 DO
        BEGIN
            Cells[0,0]:=FLOATTOSTR(269);Cells[1,0]:=FLOATTOSTR(0);
            Cells[0,1]:=FLOATTOSTR(206.7);Cells[1,1]:=FLOATTOSTR(6.9);
            Cells[0,2]:=FLOATTOSTR(503.5);Cells[1,2]:=FLOATTOSTR(27.6);
            Cells[0,3]:=FLOATTOSTR(586.5);Cells[1,3]:=FLOATTOSTR(31);
            Cells[0,4]:=FLOATTOSTR(683.3);Cells[1,4]:=FLOATTOSTR(69);
        END;
        N:=5;

```

END;

PROCEDURE TForm1.Borrar1Click(Sender: TObject);

BEGIN

borrar;

END;

PROCEDURE TForm1.Nuevo1Click(Sender: TObject);

BEGIN

borrar;

END;

PROCEDURE TForm1.BitBtn3Click(Sender: TObject);

VAR

i,c,j:integer;

BEGIN

WITH Printer **DO**

BEGIN

BeginDoc;

Canvas.Brush.Style := bsClear;

Canvas.TextHeight('10');

Canvas.TextOut(400,400,'COMPUTACION PARA INGENIERIA');

Canvas.MoveTo(400,500);

Canvas.LineTo(4000,500);

Canvas.TextOut(400,600,'REPORTE DEL PROGRAMA PARAMETROS
GEOMECHANICOS DEL MACIZO ROCOSO');

Canvas.TextOut(400,1000,label5.Caption);

Canvas.TextOut(400,800,'DATOS DE ENTRADA'+ ' '+ENSAYO TRIAXIAL
DEL M.R.');

Canvas.TextOut(800,1000,label7.Caption);

Canvas.MoveTo(400,1100);Canvas.LineTo(920,1100);

FOR i:=0 **TO** stringgrid1.RowCount **DO**

BEGIN

Canvas.TextOut(400,1200+i*200,stringgrid1.Cells[0,i]);

Canvas.TextOut(800,1200+i*200,stringgrid1.Cells[1,i]);

END;

Canvas.TextOut(1300,1000,label15.Caption);

Canvas.TextOut(1700,1000,label10.Caption);

Canvas.TextOut(2100,1000,label11.Caption);

Canvas.TextOut(2500,1000,label12.Caption);

Canvas.TextOut(2900,1000,label13.Caption);

Canvas.TextOut(3600,1000,label14.Caption);

Canvas.MoveTo(1300,1100);Canvas.LineTo(4000,1100);

FOR j:=0 **TO** stringgrid2.Colcount **DO**

BEGIN

FOR i:=0 **TO** stringgrid2.RowCount **DO**


```

BEGIN
  IF i>4 THEN c:=450 ELSE c:=400 ;
  Canvas.TextOut(1300+i*c,1200+j*200,stringgrid2.Cells[i,j]);
END;
END;
Canvas.MoveTo(1300,750+j*200);Canvas.LineTo(4000,750+j*200);
Canvas.MoveTo(1300,900+j*200);Canvas.LineTo(4000,900+j*200);
Canvas.Font.Name:='Symbol';Canvas.TextHeight('14');
Canvas.TextOut(1150,800+J*200,label16.Caption+' = ');
Canvas.Font.Name:='MS San Serif';Canvas.TextHeight('10');
Canvas.TextOut(400,1100+J*200,Groupbox2.Caption);
Canvas.TextOut(400,1300+J*200,label17.Caption+' = '+edit2.Text+' Mpa');
Canvas.TextOut(400,1500+J*200,label18.Caption+' = '+edit3.Text);
Canvas.TextOut(400,1700+J*200,label19.Caption+' = '+edit4.Text);
Canvas.TextOut(400,1900+J*200,label21.Caption);
Canvas.TextOut(400,2100+J*200,label23.Caption+' = '+edit5.Text+' Mpa');
Canvas.TextOut(400,2300+J*200,label24.Caption+' = '+edit6.Text);
Canvas.TextOut(400,2500+J*200,label25.Caption+' = '+edit7.Text+' Mpa');
Canvas.TextHeight('8');
Canvas.TextOut(400,6000,' UNIVERSIDAD MAYOR DE SAN SIMON');
Canvas.TextOut(400,6100,' FACULTAD DE CIENCIAS Y TECNOLOGIA');
Canvas.TextOut(400,6200,' CARRERA DE INGENIERIA CIVIL');
Canvas.TextOut(2500,6200,' Elaborado por: GERMAN CAMACHO - MAURICIO
ANDIA');
EndDoc;
END;
END;

PROCEDURE TForm1.Creditos1Click(Sender: TObject);
BEGIN
  showmessage('UNIVERSIDAD MAYOR DE SAN SIMON'+ #13+
'FACULTAD DE CIENCIAS Y TECNOLOGIA'+ #13+
'CARRERA DE INGENIERIA CIVIL'+ #13+
'Programa realizado para uso Educacional.'+ #13+
'De Aplicación a la carrera de Ingeniería Civil'+#13+
'Area Geotecnia'+#13+
'Realizado por: German Camacho Ch. '+#13+
'          Mauricio Andia B. ');
END;

```

```

PROCEDURE TForm1.Guardar1Click(Sender: TObject);
VAR
i,j:integer;
cad:string;
BEGIN
Dir:=inputbox('ARCHIVO','RUTA Y NOMBRE','C:\FICHEROS\SALIDA.TXT');
AssignFile(F,Dir);
Rewrite(F);
  Writeln(F,#9,'COMPUTACION PARA INGENIERIA');
  Writeln(F,'-----');
  Writeln(F,#9,'PARAMETROS GEOMECHANICOS DEL MACIZO ROCOSO');
  Writeln(F,#9,' DATOS DE ENTRADA');
  Writeln(F,#9,' -----');
  Writeln(F,'Esfuerzo Mayor',#9,'Esfuerzo Menor');
  FOR i:=0 TO N-1 DO
    Writeln(F,stringgrid1.cells[0,i],#9,#9, stringgrid1.cells[1,i]);
  Writeln(F,#9,' SALIDA DE RESULTADOS');
  Writeln(F,#9,' -----');
  Writeln(F,'N',#9,'X',#9,'Y',#9,#9,'X^2',#9,'Y^2',#9,#9,'XY');
  FOR j:=0 TO 5 DO
    BEGIN
      cad:="";
      FOR i:=0 TO N DO
        cad:=cad+stringgrid2.cells[i,j]+#9;
      Writeln(F,cad,#9);
    END;
  Writeln(F,Groupbox2.caption);
  Writeln(F,label17.caption,' = ',edit2.text,'Mpa');
  Writeln(F,label18.caption,' = ',edit3.text);
  Writeln(F,label19.caption,' = ',edit4.text);
  Writeln(F,label21.caption);
  Writeln(F,label23.caption,' = ',edit5.text,'Mpa');
  Writeln(F,label24.caption,' = ',edit6.text);
  Writeln(F,label25.caption,' = ',edit7.text,'Mpa');
Closefile(F);
showmessage('Archivo Guardado');
END;

END.

```

APLICACION 5.

APLICACION SUELOS (TALUDES)

El programa calcula el ángulo del talud y ángulo del plano de falla para un factor de seguridad dado, aplicable al diseño de taludes.

Entonces se debe preparar el siguiente interfaz gráfico en Delphi fig. 1. Que consta de 2 formularios: 1 para la aplicación y el otro para impresión utilizando el componente QuickReport.

Fig. 1 Interfaz gráfico.

UNIT uEstabilidadTaludes;

INTERFACE

USES

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, ExtCtrls, XPMAN, StdCtrls, Grids, Buttons, Math, Menus, ActnList, ExtActns;

TYPE

```
TForm1 = class(TForm)
  Image1: TImage;
  XPManifest1: TXPManifest;
  Label1: TLabel;
  GroupBox1: TGroupBox;
  Label2: TLabel;
  Label3: TLabel;
  Label4: TLabel;
  Label5: TLabel;
  Label6: TLabel;
  Edit1: TEdit;
  Edit2: TEdit;
  Edit3: TEdit;
  Edit4: TEdit;
  Label7: TLabel;
  Label8: TLabel;
  Label9: TLabel;
  Label10: TLabel;
  Label11: TLabel;
  Label12: TLabel;
  GroupBox2: TGroupBox;
  StringGrid1: TStringGrid;
  Label13: TLabel;
  ComboBox1: TComboBox;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  BitBtn3: TBitBtn;
  Edit5: TEdit;
  Edit6: TEdit;
  GroupBox3: TGroupBox;
  Edit7: TEdit;
  Edit8: TEdit;
  Label14: TLabel;
  Label15: TLabel;
  Label16: TLabel;
  Label17: TLabel;
  Label18: TLabel;
```

```

Label19: TLabel;
MainMenu1: TMainMenu;
Archivo1: TMenuItem;
Opciones1: TMenuItem;
Ayuda1: TMenuItem;
Nuevo1: TMenuItem;
Guardar1: TMenuItem;
Configurarimpresora1: TMenuItem;
Imprimir1: TMenuItem;
Cerrar1: TMenuItem;
Cerrar2: TMenuItem;
Calcular1: TMenuItem;
Borrar1: TMenuItem;
RecuperarEjemplo1: TMenuItem;
Informacion1: TMenuItem;
Creditos1: TMenuItem;
BitBtn4: TBitBtn;
ActionList1: TActionList;
FileRun1: TFileRun;
PROCEDURE BitBtn1Click(Sender: TObject);
PROCEDURE BitBtn3Click(Sender: TObject);
PROCEDURE BitBtn4Click(Sender: TObject);
PROCEDURE Imprimir1Click(Sender: TObject);
PROCEDURE Creditos1Click(Sender: TObject);
PROCEDURE Configurarimpresora1Click(Sender: TObject);
PROCEDURE Guardar1Click(Sender: TObject);
PROCEDURE RecuperarEjemplo1Click(Sender: TObject);
PROCEDURE Nuevo1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
END;
Mattipo=array[1..100,1..100] of real;
VAR
  Form1: TForm1;
  Res:Mattipo;
  fil:integer;
  f:TextFile;
  Dirc:string;
IMPLEMENTATION

USES uInformeTaludes;

{$R *.dfm}

```

```

PROCEDURE Enumerar;
BEGIN
WITH form1 DO
BEGIN
  IF ComboBox1.Text='Completamente Saturado' THEN
    BEGIN
      WITH StringGrid1 DO
        BEGIN
          ColCount:=9;Width:=635;
          cells[0,0]:='Numero';cells[1,0]:='Ang Talud';
          cells[2,0]:='Ang Falla';cells[3,0]:='( Zc / H )';
          cells[4,0]:='P';cells[5,0]:='Q';
          cells[6,0]:='R';cells[7,0]:='S';
          cells[8,0]:='F.S.';
        END;
      END
    ELSE
      BEGIN
        WITH StringGrid1 DO
          BEGIN
            ColCount:=7;Width:=513;
            cells[0,0]:='Numero';cells[1,0]:='Ang Talud';
            cells[2,0]:='Ang Falla';cells[3,0]:='( Zc / H )';
            cells[4,0]:='P';cells[5,0]:='Q';
            cells[6,0]:='F.S.';
          END;
        END;
      END;
    END;
  END;
PROCEDURE asignar;
VAR
  i,j:integer;
  cad:string;
BEGIN
  WITH form2 DO
    BEGIN
      qrmemo1.Lines.Clear;
      qrlabel4.Caption:=form1.label13.Caption;qrlabel10.Caption:=form1.ComboBox1.Text;
      qrlabel5.Caption:=form1.label2.Caption;qrlabel11.Caption:=form1.edit1.text;
      qrlabel6.Caption:=form1.label3.Caption;qrlabel12.Caption:=form1.edit2.text;
      qrlabel20.Caption:=form1.label9.Caption;
      qrlabel23.Caption:=form1.Label10.Caption;
      qrlabel21.Caption:=form1.Label11.Caption;qrlabel22.Caption:=form1.Label12.Caption;
      qrlabel7.Caption:=form1.label5.Caption;qrlabel13.Caption:=form1.edit3.text;
      qrlabel8.Caption:=form1.label6.Caption;qrlabel14.Caption:=form1.edit4.text;
      qrlabel15.Caption:=form1.label7.Caption;qrlabel16.Caption:=form1.edit5.text;
    END
  END

```

```

qlabel18.Caption:=form1.label8.Caption;qlabel17.Caption:=form1.edit6.text;
qlabel25.Caption:=form1.label14.Caption;qlabel26.Caption:=form1.edit7.Text;
qlabel24.Caption:=form1.label15.Caption;qlabel27.Caption:=form1.edit8.Text;
qlabel26.Caption:=form1.Edit7.Text+'°';
qlabel27.Caption:=form1.Edit8.Text+'°';
qrimage1.Picture:=form1.Image1.Picture;
FOR i:=fil-5 to fil+5 DO
  BEGIN
    cad:=' ';
    FOR j:=0 to form1.stringgrid1.colcount-1 DO
      cad:=cad+form1.stringgrid1.cells[j,i]+'  ';
      form2.QRMemo1.Lines.Add(cad);
    END;
  IF form1.stringgrid1.colcount=7 THEN
    BEGIN
      qlabel46.Caption:='FS';
      qlabel47.Caption:='';
      qlabel48.Caption:='';
    END
  ELSE
    BEGIN
      qlabel46.Caption:='R';
      qlabel47.Caption:='S';
      qlabel48.Caption:='FS';
    END;
  END;
END;
END;

```

PROCEDURE TForm1.BitBtn1Click(Sender: TObject);

VAR

h,fs,fi,fir,c,per,pew:real;
 ap,apr,afr,zch,p,q,r,s,af,fsc,men:real;
 i:integer;

BEGIN

Enumerar;
 h:=strtofloat(edit1.Text);
 fs:=strtofloat(edit2.Text);
 fi:=strtofloat(edit3.Text);
 c:=strtofloat(edit4.Text);
 per:=strtofloat(edit5.Text);
 pew:=strtofloat(edit6.Text);
 fir:=degtorad(fi);
 men:=2;
 stringgrid1.RowCount:=92;

IF ComboBox1.Text='Completamente Drenado' **THEN**

BEGIN

FOR i:=1 to 89 **DO**

BEGIN**TRY**

```

stringgrid1.Cells[0,i]:=inttostr(i);
af:=i;ap:=(af+fi)/2;res[i,1]:=af;res[i,2]:=ap;
stringgrid1.Cells[1,i]:=FLOATTOSTR(af);
stringgrid1.Cells[2,i]:=FLOATTOSTR(ap);
afr:=degtorad(af);apr:=degtorad(ap);
zch:=1-sqrt(cotan(afr)*tan(apr));res[i,3]:=zch;
stringgrid1.Cells[3,i]:=FLOATTOSTR(zch);
p:=(1-zch)*csc(apr);res[i,4]:=p;
stringgrid1.Cells[4,i]:=FLOATTOSTR(p);
q:=((1-sqr(zch))*cot(apr)-cot(afr))*sin(apr);res[i,5]:=q;
stringgrid1.Cells[5,i]:=FLOATTOSTR(q);
fsc:=(2*c/(per*h)*p+q*cot(apr)*tan(fir))/q;res[i,6]:=q;
stringgrid1.Cells[6,i]:=FLOATTOSTR(fsc);

```

IF (abs(fsc-fs))< men **THEN****BEGIN**

men:=abs(fsc-fs);

fil:=i;

END;**EXCEPT****ON** EzeroDivide **DO**

Continue;

END;**END;****END****ELSE****BEGIN****FOR** i:=1 to 89 **DO****BEGIN****TRY**

```

stringgrid1.Cells[0,i]:=inttostr(i);
af:=i;ap:=(af+fi)/2;res[i,1]:=af;res[i,2]:=ap;
stringgrid1.Cells[1,i]:=FLOATTOSTR(af);
stringgrid1.Cells[2,i]:=FLOATTOSTR(ap);
afr:=degtorad(af);apr:=degtorad(ap);
zch:=1-sqrt(cotan(afr)*tan(apr));res[i,3]:=zch;
stringgrid1.Cells[3,i]:=FLOATTOSTR(zch);
p:=(1-zch)*csc(apr);res[i,4]:=p;
stringgrid1.Cells[4,i]:=FLOATTOSTR(p);
q:=((1-sqr(zch))*cot(apr)-cot(afr))*sin(apr);res[i,5]:=q;
stringgrid1.Cells[5,i]:=FLOATTOSTR(q);
r:=(pew/per)*zch;res[i,6]:=r;
stringgrid1.Cells[6,i]:=FLOATTOSTR(r);
s:=zch*sin(apr);res[i,7]:=s;
stringgrid1.Cells[7,i]:=FLOATTOSTR(s);
fsc:=(2*c/(per*h)*p+(q*cot(apr)-r*(p+s))*tan(fir))/(q+r*s*cot(apr));res[i,8]:=q;

```



```
stringgrid1.Cells[8,i]:=FLOATTOSTR(fsc);  
IF (abs(fsc-fs))< men THEN  
BEGIN  
    men:=abs(fsc-fs);  
    fil:=i;  
END;  
EXCEPT  
ON EzeroDivide DO  
    Continue;  
END;  
END;  
END;  
edit7.Text:=FLOATTOSTR(res[fil,1]);  
edit8.Text:=FLOATTOSTR(res[fil,2]);  
END;
```

```
PROCEDURE borrar;  
VAR  
i,j:integer;  
BEGIN  
FOR i:=0 to form1.stringgrid1.ColCount DO  
BEGIN  
    FOR j:=0 to form1.stringgrid1.RowCount DO  
        form1.stringgrid1.Cells[i,j]:=' '  
END;  
END;  
PROCEDURE TForm1.BitBtn3Click(Sender: TObject);  
BEGIN  
    borrar;  
END;
```

```
PROCEDURE TForm1.BitBtn4Click(Sender: TObject);  
BEGIN  
    Close;  
END;
```

```
PROCEDURE TForm1.Imprimir1Click(Sender: TObject);  
BEGIN  
    asignar;  
    form2.QuickRep1.Print;  
END;
```

```

PROCEDURE TForm1.Creditos1Click(Sender: TObject);
BEGIN
  showmessage('UNIVERSIDAD MAYOR DE SAN SIMON'+ #13+
  'FACULTAD DE CIENCIAS Y TECNOLOGIA'+ #13+
  'CARRERA DE INGENIERIA CIVIL'+ #13+
  'Programa realizado para uso Educacional.'+ #13+
  'De Aplicación a la carrera de Ingeniería Civil'+#13+
  'Area Geotecnia'+#13+
  'Realizado por: German Camacho Ch. '+#13+
  '          Mauricio Andia B. ');
END;

```

```

PROCEDURE TForm1.Configurarimpresora1Click(Sender: TObject);
BEGIN
  asignar;
  form2.QuickRep1.Preview;
END;

```

```

PROCEDURE TForm1.Guardar1Click(Sender: TObject);
VAR
  i,j:integer;
  cad:string;
BEGIN
  Dir:=inputbox('ARCHIVO','RUTA Y
  NOMBRE','C:\FICHEROS\SALIDATALUD.TXT');
  AssignFile(F,Dir);
  Rewrite(F);
  Writeln(F,#9,'COMPUTACION PARA INGENIERIA');
  Writeln(F,'-----');
  Writeln(F,' ');
  Writeln(F,'CALCULO DE ESTABILIDAD DE TALUDES');
  Writeln(F,' ');
  Writeln(F,#9,' DATOS DE ENTRADA');
  Writeln(F,#9,' -----');
  Writeln(F,label13.caption,' = ',combobox1.text);
  Writeln(F,label2.caption,' = ',edit1.text,',',label9.caption);
  Writeln(F,label3.caption,' = ',edit2.text);
  Writeln(F,label4.caption);
  Writeln(F,label5.caption,' ',edit3.text);
  Writeln(F,label6.caption,' ',edit4.text,',',label10.caption);
  Writeln(F,label7.caption,' = ',edit5.text,',',label11.caption);
  Writeln(F,label8.caption,' = ',edit6.text,',',label12.caption);
  Writeln(F,' ');
  Writeln(F,#9,' SALIDA DE RESULTADOS');
  Writeln(F,#9,' -----');
  Writeln(F,label14.caption,' = ',edit7.text);

```

```

Writeln(F,label15.caption,' = ',edit8.text);
FOR i:=0 to stringgrid1.colcount DO
cad:=cad+stringgrid1.cells[i,0]+#9;
Writeln(F,cad);
FOR i:=fil-7 to fil+7 DO
  BEGIN
    cad:=' ';
    FOR j:=0 to stringgrid1.colcount DO
      cad:=cad+stringgrid1.cells[j,i]+#9;
      Writeln(F,cad);
    END;
Closefile(F);
showmessage('Archivo Guardado');
END;

PROCEDURE TForm1.RecuperarEjemplo1Click(Sender: TObject);
BEGIN
edit1.Text:=FLOATTOSTR(30);
edit2.Text:=FLOATTOSTR(1.5);
edit3.Text:=FLOATTOSTR(30);
edit4.Text:=FLOATTOSTR(50);
edit5.Text:=FLOATTOSTR(26);
edit6.Text:=FLOATTOSTR(10);

END;

PROCEDURE TForm1.Nuevo1Click(Sender: TObject);
BEGIN
edit1.Clear;edit2.Clear;edit3.Clear;
edit4.Clear;edit5.Clear;edit6.Clear;
edit7.Clear;edit8.Clear;
borrar;
END;

END.

```

```

UNIT uInformeTaludes;

```

```

INTERFACE

```

```

USES

```

```

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, QRCtrls, QuickRpt, ExtCtrls;

```

```

TYPE

```

```

  TForm2 = class(TForm)

```

QuickRep1: TQuickRep;
QRBand1: TQRBand;
QRLabel1: TQRLabel;
QRLabel2: TQRLabel;
QRLabel3: TQRLabel;
QRLabel4: TQRLabel;
QRLabel10: TQRLabel;
QRLabel5: TQRLabel;
QRLabel11: TQRLabel;
QRLabel6: TQRLabel;
QRLabel12: TQRLabel;
QRLabel7: TQRLabel;
QRLabel13: TQRLabel;
QRLabel8: TQRLabel;
QRLabel14: TQRLabel;
QRLabel30: TQRLabel;
QRImage1: TQRImage;
QRLabel9: TQRLabel;
QRLabel15: TQRLabel;
QRLabel16: TQRLabel;
QRLabel17: TQRLabel;
QRLabel18: TQRLabel;
QRLabel19: TQRLabel;
QRLabel20: TQRLabel;
QRLabel24: TQRLabel;
QRLabel25: TQRLabel;
QRLabel26: TQRLabel;
QRLabel27: TQRLabel;
QRLabel28: TQRLabel;
QRLabel29: TQRLabel;
QRLabel31: TQRLabel;
QRLabel32: TQRLabel;
QRLabel33: TQRLabel;
QRLabel23: TQRLabel;
QRLabel21: TQRLabel;
QRLabel22: TQRLabel;
QRLabel34: TQRLabel;
QRLabel35: TQRLabel;
QRLabel36: TQRLabel;
QRLabel37: TQRLabel;
QRMemo1: TQRMemo;
QRLabel38: TQRLabel;
QRLabel39: TQRLabel;
QRLabel40: TQRLabel;
QRLabel41: TQRLabel;
QRLabel42: TQRLabel;
QRLabel43: TQRLabel;

```
QRLabel44: TQRLabel;  
QRLabel45: TQRLabel;  
QRLabel46: TQRLabel;  
QRLabel47: TQRLabel;  
QRLabel48: TQRLabel;  
QRShape1: TQRShape;  
QRLabel49: TQRLabel;  
QRLabel50: TQRLabel;  
QRLabel51: TQRLabel;  
QRShape2: TQRShape;  
private  
  { Private declarations }  
public  
  { Public declarations }  
END;  
  
VAR  
  Form2: TForm2;  
  
implementation  
  
{$R *.dfm}  
  
END.
```

EJERCICIOS PROPUESTOS

PROBLEMA 1.

Realizar un diagrama de flujo y programa que determine si un número entero ingresado por el usuario es primo o no, mostrar en pantalla dicho número e imprimir.

Un numero es primo, si es divisible entre si y la unidad.

Ejm. 1, 3, 5, 7, 11, etc.

PROBLEMA 2.

Realizar un diagrama de flujo y programa que permita leer 30 números y averiguar, cuales son positivos, negativos e iguales a cero.

PROBLEMA 3.

Realizar un diagrama de flujo y programa que permita leer 20 números y obtener lo siguiente:

- a) Imprimir la cantidad de #s pares y el promedio de números impares.
- b) Imprimir el mayor y menor numero.

PROBLEMA 4.

Realizar un diagrama de flujo y programa que permita leer un número y obtener la siguiente información:

- a) Imprimir el factorial del primer número.
- b) Imprimir todos los dígitos de cada número.
- c) Imprimir el número previamente invertido los dígitos.

PROBLEMA 5.

Realizar un diagrama de flujo y programa que permita leer un vector con N elementos y obtener la siguiente información:

- a) Imprimir el promedio de los elementos.
- b) Imprimir la sumatoria de todos los elementos que sean múltiplos de 3.
- c) Imprimir el mayor y menor elemento.

PROBLEMA 6.

Realizar un diagrama de flujo y programa que permita leer un vector con N elementos, e insertar M valores en una posición dada.

PROBLEMA 7.

Realizar un diagrama de flujo y programa que permita leer un vector con N elementos, y eliminar M elementos del vector.

PROBLEMA 8.

Realizar un diagrama de flujo y programa que permita leer una matriz de N por M elementos, y obtener lo siguiente:

- a) Imprimir el mayor y menor elemento y la posición en la cual se encuentra.
- b) Imprimir el promedio de elementos.

PROBLEMA 9.

Realizar un diagrama de flujo y programa que permita leer dos matrices: A de N x M elementos y B de M x L, multiplicar las dos matrices e imprimir la matriz resultante.

ANEXOS

A1. INICIO DEL SISTEMA OPERATIVO

A1.1 Arranque del sistema

Desde que arranca una PC hasta que es mostrado el sistema operativo, la máquina hace una serie de procesos. A continuación serán desglosados esos procesos, desde que se presiona el botón de encendido hasta que el sistema operativo es inicializado.

A1.2 BIOS

Nada más pulsar el botón de encendido del PC, se cargan las instrucciones contenidas en el POST (power-ON self test o ‘autotest de encendido’) del BIOS.

El POST hace unas comprobaciones básicas y toma la configuración del CMOS (complementary metal-oxide semiconductor o semiconductor complementario de óxido metálico). Son las instrucciones del CMOS las que determinan, entre otras cosas, el orden de los dispositivos de arranque.

Durante este proceso se verifica cual es el primer dispositivo de arranque, y si este es un disco duro Básico, le pasará el control al MBR (master boot record o registro maestro de arranque).

El MBR es el encargado de llevar a cabo las siguientes operaciones.

Busca en la tabla de particiones cual es la primera partición activa para transferirle el control.

Revisa cual es el sector de inicio de la partición que esté configurada como activa.

Carga una copia del sector de inicio desde la partición activa en memoria y finalmente transfiere el control al código ejecutable del sector de inicio.

A1.3 El sector Maestro de Arranque y el Sector de Inicio

Una vez terminado el trabajo del MBR este, transfiere el control al Sector de Arranque o Boot Sector, de nuestra partición activa en sistemas Windows, la primera partición activa es básica puesto que suele contener los archivos básicos de arranque de Windows y suele coincidir con la letra “C”.

En cualquier caso, nuestro sector de arranque asume las siguientes operaciones:

- Las instrucciones de inicialización para CPU basada en x86 (esta es la familia de procesadores Intel en la que funciona habitualmente Windows XP)

- La identificación original del fabricante de nuestro PC, en el caso de que nuestro sistema sea OEM.
- BIOS Parameter Block, BPB
- BIOS Parameter Block, Extendida.
- El código ejecutable que inicia nuestro sistema operativo.

Así pues, el siguiente paso, una vez leídas las instrucciones de inicialización para CPU basadas en x86, sería cargar la BPB.

La BPB contiene la estructura básica del volumen y las controladoras de disco utilizan este sector para leer y configurar los parámetros básicos de los volúmenes contenidos en nuestro disco.

También es un proceso básico puesto que es el encargado de transferirle el control al código ejecutable, que es en sí, el que va a iniciar nuestro sistema operativo.

A1.4 El archivo NTLDR

Ahora empiezan a intervenir los archivos propios del sistema operativo.

En todos los sistemas con núcleo NT (por ejemplo Windows XP o Windows 2000) el cargador se llama ntldr (de “nt loader”), y se encuentra en la partición activa o volumen de sistema (habitualmente C).

El archivo ntldr es el encargado de leer el sistema de archivos tanto de una partición NTFS como de una FAT.

Lo primero que hace el ntldr es cargar un serie de controladores básicos de dispositivos que van embebidos en este archivo, justo a continuación lee la información contenida en el archivo boot.ini que se encuentra oculto en el directorio raíz del volumen de sistema es decir en C, y que referencia qué tipo de Sistema Operativo tiene que cargar.

Nuestro archivo ntddetect.com, se encarga de cargar la información contenida en el perfil de hardware y las tablas de la ACPI, y las envía para su inclusión al archivo de arranque ntldr, para ser agregadas en la clave del registro.

HKEY_LOCAL_MACHINE\HARDWARE

El Kernel (núcleo) utiliza datos internos que provee el propio ntldr para crear la clave mencionada, cuya información será renovada en cada arranque de la máquina, en base al hardware que se detecte durante cada inicio de la máquina.

De vuelta al ntldr, este pasará el control al archivo ntoskrnl.exe, es decir, el núcleo del sistema operativo (el nombre del archivo proviene de “nt operating system kernel”), y al archivo HAL (hardware abstraction layer o capa de abstracción del hardware), y leerá la información contenida en el fichero “system” que se ubica en la carpeta \windows\system32\config.

Son el HAL y el kernel los encargados de poner en funcionamiento el software, mediante una serie de componentes llamada Windows Executive. Estos componentes están almacenados en los “conjuntos de control” del registro (control sets). Concretamente los podemos encontrar referenciados en la clave del registro.

HKEY_LOCAL_MACHINE\SYSTEM

Esta clave es utilizada para múltiples propósitos entre ellos determinar qué drivers deben ser cargados durante cada arranque del sistema, en base al hardware cargado.

A1.5 Carga de Drivers y servicios

Ahora debe prestarse atención a los drivers o controladores y a los servicios.

Están contenidos, respectivamente, en las siguientes carpetas a las que puede accederse directamente desde Inicio > Ejecutar y escribir los siguientes:

- Drivers -> %systemroot%\System32\Drivers
- Servicios -> %systemroot%\System
- Puesto que los drivers también son servicios, en este momento el núcleo lee la información de la clave
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

Se arrancarán primero los servicios que tengan el valor Start puesto a ‘0’ (los drivers de arranque) y luego los que tengan dicho valor puesto a ‘1’.

Ahora llega el proceso en el que interviene el Administrador de sesión (Session Manager, smss.exe).

Éste crea variables de entorno, cambia a modo gráfico, y además de otras cosas, se encarga de arrancar el Administrador del logon (Logon Manager, winlogon.exe). El administrador de sesión lee tres claves:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SessionManager

En la que, por ejemplo, podría haber una referencia al “autochk.exe” (una versión del chkdsk), por si se necesita cargarse:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SessionManager\Subsystem

Con los distintos subsistemas:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

Y acudiría a los servicios cuyo valor Start esté en “Auto-load”.

Si antes del logon quisiéramos que se nos cargase alguna aplicación (no configurada como servicio, sino de forma “normal”) se tendra que colocarlas en alguna de estas dos claves del registro:

[1] HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce

[2] HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices

Estas dos claves se cargan de forma asíncrona. Por tanto, su contenido puede cargarse al mismo tiempo y no necesariamente en el orden especificado en las claves. Es más, aunque se cargan en el orden propuesto (primer [1]y luego [2]), se cargan también de forma asíncrona con el propio proceso de Inicio en sí, por lo que podría ser que continuasen cargándose después de introducir el nombre de usuario y contraseña.

Precisamente, como ya se ha indicado, la utilidad de estas dos claves es introducir aplicaciones en ellas para que se carguen antes del logon.

A1.6 INICIANDO WINDOWS

En este momento es winlogon.exe quien toma el control.

Inicia el Administrador de control de servicios (Service control manager) el Local security authority (lsass.exe) y la Autenticación e Identificación Gráfica (Graphical Identification **AND** Authentication, GINA) y pide al usuario (si así está configurado) un nombre y una contraseña.

De la autenticación de ese usuario y contraseña se encarga el protocolo Kerberos V5 o bien NTLM. (Como comentario totalmente aparte, comentar que Kerberos es el nombre inglés del perro de dos cabezas Cancerbero que era encargado de custodiar las puertas del infierno en la mitología).

NTLM proviene de NT Lan Manager, y se usa sobre todo en grupos de trabajo, Windows 95 y Windows 98.

Es en esta fase cuando ciertas claves del registro son adaptadas si el arranque ha tenido éxito, entre ellas las claves Control sets que anteriormente se menciono, haciendo una nueva copia de la misma.

A1.7 Después de la autenticación

Tras introducir el usuario y la contraseña, se cargan las claves:

[3] HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce\Setup

[4] HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce

Esta clave [4] se carga de forma síncrona aunque su contenido no tiene por qué cargarse en el orden especificado. Esto también significa que si esta clave no se carga completamente no se cargan las siguientes.

Igualmente, esta clave [4] no se carga hasta que no se han cargado las anteriores claves [1], [2] y [3].

A continuación se carga la clave:

[5] HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnceEx.

A partir de aquí, y hasta el ítem [12] el orden de carga vuelve a ser asíncrono y, así, todo se carga al mismo tiempo (más o menos) pudiéndose solapar unas cosas con otras. A continuación se carga lo siguiente:

[6] Sección Load del Win.ini

Y luego:

[7] Sección Run del Win.ini

A continuación la clave:

[8] HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run

Que contiene los programas instalados en la máquina y la clave:

[9] HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

Que contiene los programas instalados específicamente en el usuario autenticado.

También se carga la clave:

[10] HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce

Hasta ahora lo que se ha cargado ha sido el núcleo de Windows y el contenido del registro.

Ahora se acude a las carpetas del menú inicio. Primero, las de todos los usuarios.

[11] Documents and Settings\All Users\Menú Inicio\Programas\Inicio

Y luego los del usuario autenticado:

[12] Documents and Settings\[usuario]\Menú Inicio\Programas\Inicio

Donde [usuario] es el nombre de la cuenta en la que se ha iniciado sesión. A continuación se carga la clave:

[13] HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce

Y finalmente las tareas programas que se tengan.

Volvemos a repetir que hasta [12] el proceso de carga se hace de forma asíncrona, por lo que podrían solaparse unas claves con otras y unos archivos con otros. Los elementos del menú de inicio se cargan de forma alfanumérica. Por ejemplo, “a01.exe”, “a10.exe” y “b2p.exe” se cargan en ese orden. Si nos interesa alterar el orden de carga, se tendría que renombrar los archivos.

A1.8 Otras claves utilizadas durante la inicialización de Windows

Todo lo del apartado anterior suelen ser las utilizadas comúnmente para inicializar programas, antes o después de la fase de autenticación, pero quizás sea conveniente anotar alguna ubicación más en el registro que tiene el mismo propósito: iniciar los programas del usuario.

Una de las ubicaciones más esotéricas y menos conocidas para iniciar programas es la clave Load, ubicada en la rama:

HKEY_CURRENT_USER\Software\Microsoft\WindowsNT\CurrentVersion\Windows\load.

En esta clave y en esta rama, podríamos especificar programas que se iniciarían sólo en la sesión de usuario que esté actualmente logado (autenticado).

La clave análoga que afectaría a todos los usuarios de la máquina es denominada userinit, que podremos encontrar en la rama:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\Userinit

Esta última clave, a diferencia de la clave Load, que anteriormente se comento, se encuentra presente en todos los sistemas y suele contener un único valor que apunta al ejecutable o proceso userinit.exe, que está encargado de inicializar los parámetros de inicialización de la Shell de Windows. Esta clave acepta valores separados por coma (CSV) por lo que es posible agregar más valores al ya comentado en esta clave.

Finalmente también podemos ejecutar diferentes procesos en la clave Explorer\Run

Que se podran encontrar en las ramas:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run.

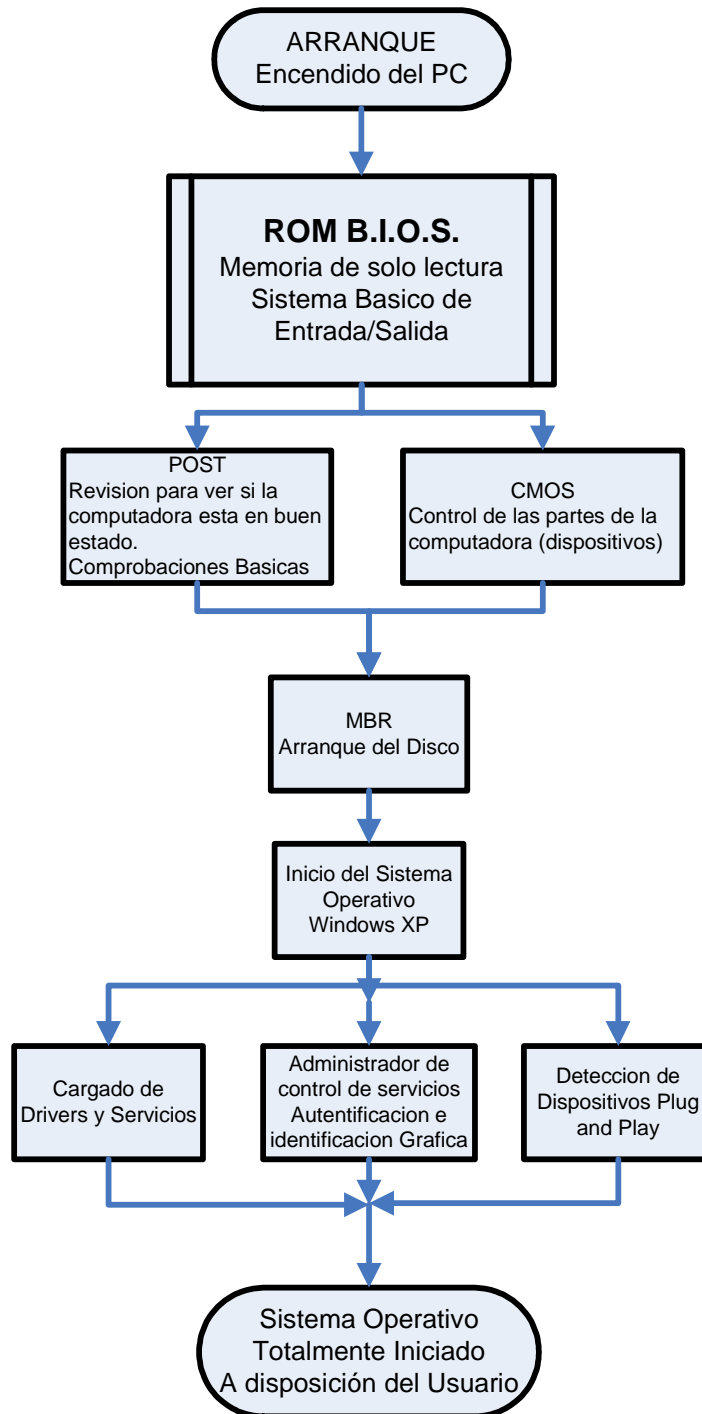
A1.9 Detección de dispositivos Plug AND Play

La detección de este tipo de dispositivos que son detectados e instalados casi sin que el usuario interactúe para nada con el sistema, son cargados asincrónicamente durante la fase de autenticación. Esta fase de detección se basa fundamentalmente, en el firmware de cada dispositivo hardware conectado y en las características internas que el S.O. posee para la detección de nuevos tipos de dispositivos. Windows XP está optimizado para la detección de dispositivos que cumplan con la normativa ACPI.

Esta es una explicación bastante detallada del Inicio del Sistema Windows XP desde que se pulsa el botón de encendido hasta que está totalmente iniciado.

A2. DIAGRAMA DEL PROCESO DE INICIO

- SISTEMA OPERATIVO WINDOWS XP



A3. MANTENIMIENTO DEL ORDENADOR

Para que el ordenador personal este en un estado adecuado, para su respectivo uso, se debe realizar un mantenimiento periódico y sistemático. Esto involucra realizar un mantenimiento de hardware y software.

A3.1 Mantenimiento de Hardware

De acuerdo al lugar donde se ubique el ordenador, exposición a la intemperie, se debe realizar una limpieza general del polvo que se deposita en los componentes del ordenador como ser placa madre, procesador, fuente, disco duro, etc.

Cabe hacer notar, que este mantenimiento lo debe realizar una persona con conocimientos básicos de hardware para su correcta manipulación.

A3.2 Mantenimiento de Software

Existen herramientas que ayudan a realizar este tipo de mantenimiento, que se encuentran en el mismo sistema operativo, estas herramientas son:

A3.2.1 Scandisk (comprobación de errores)

Esta herramienta que es parte del sistema operativo de Windows, realiza una comprobación de errores del sistema de archivos y sectores defectuosos en el disco duro, como también realiza la reparación de estos sectores y presenta un informe general.

A3.2.2 Desfragmentador de disco

El Desfragmentador de disco concentra los archivos y carpetas fragmentados en el disco duro del equipo, para que cada uno ocupe un solo espacio contiguo en el volumen. Como consecuencia, el sistema tendrá acceso a los archivos y carpetas, y guardará los nuevos de una forma más eficaz. Mediante la concentración de los archivos y carpetas, el Desfragmentador de disco también concentra el espacio libre, lo que hace menos probable la fragmentación de los archivos nuevos.

A3.2.3 Liberador de espacio en disco

El Liberador de espacio en disco ayuda a liberar espacio en el disco duro. El Liberador de espacio en disco busca en la unidad y muestra los archivos temporales, archivos de caché de Internet y archivos de programa innecesarios que puede eliminar de forma segura. Puede hacer que Liberador de espacio en disco elimine algunos o todos estos archivos.

A4. INSTALACIÓN DE PROGRAMAS

Todos los programas computacionales vienen con un CD instalador, el cual realiza el proceso de instalación en un computador. Para poder instalar solo se debe ejecutar el archivo Setup.exe o Install.exe y seguir las instrucciones de instalación de cada programa.

A4.1 Definición

Copia de archivos, carpetas de un programa (Delphi, Sap, Autocad, etc.) desde un CD (generalmente), al disco duro de un computador, configurando el sistema operativo para que el programa instalado funcione correctamente.

Antes de proceder a instalar cualquier programa en un computador, es preciso ver los requerimientos mínimos de instalación, como por ejemplo:

- Intel 486, Pentium (recomendado)
- Microsoft Windows 95, 98, Me, Windows 2000, Windows NT4.0, Windows XP (recomendado).
- 64 MB de RAM (128 recomendado).
- 200 MB de espacio en disco.
- 640 x 480 VGA video (1024 x 768 recomendado)
- Ratón o digitalizador compatible con Windows.
- Lector de CD-ROM (para iniciar la instalación).
- Tener instalado programa de gráficos CAD (Auto Cad 2004,2005)

Los requerimientos anteriores son del programa Eagle Point 2005, estos requerimientos varían de acuerdo a la versión del producto.

Es recomendable reiniciar el sistema operativo después de terminar la instalación de cualquier programa.

Esta es una pequeña guía de los pasos por los que pasará durante el proceso de instalación del programa Delphi 8:

1. Localice el archivo Prereqs.exe que instalara programas que son requisitos para poder instalar Delphi 8.
2. Una vez ha comenzado la instalación es cuestión de seguir cada paso lo que indica el computador.
3. Terminado el proceso de instalación debe reiniciar el computador.

A5. RELACIÓN DE DELPHI CON OTROS PROGRAMAS

Desde aplicaciones desarrolladas en Delphi se puede tener acceso y controlar muchos programas, entre ellos están AutoCAD, Excel, Windows, etc.

A5.1 Delphi – AutoCAD

Esta relación permite a desarrolladores usar la ingeniería de gráficos de AutoCAD y controlar desde aplicaciones de Delphi. En AutoCAD, todos los objetos son expuestos y son accesibles usando automatización OLE [1]. Esta capacidad permite a desarrolladores crear (acorde con Autodesk) controladores de objetos, macros e incluso aplicaciones completas.

A5.2 Delphi – Excel

Se pueden realizar intercambios de datos entre Delphi y Excel, proteger hojas en Excel. Usando aplicaciones en Delphi se puede expandir la potencialidad de Excel, utilizar todas las funciones que Excel posee.

[1] OLE Object Linking and Embedding: Estándar desarrollado por Microsoft, que permite crear objetos en una aplicación y luego insertarlos en otra aplicación compartiendo información entre aplicaciones.

A5.3 EJEMPLOS

Relación de Delphi con otros programas.

Ejemplo 1

Dibujo de objetos en AutoCad

PROCEDURE TForm1.Button1Click(Sender: TObject);

VAR

Acad, Circle, vPoint, Mspace : OleVariant;

BEGIN

// Creación de las coordenadas del punto y asignación de valores

vPoint := VarArrayCreate([0,2], VT_R8);

vPoint[0] := 2.0; vPoint[1] := 4.0; vPoint[2] := 0.0;

// Obtener la aplicacion de AutoCAD

Acad := GetActiveOleObject('AutoCAD.Application.2002');

// Obtener el documento activo Model space

Mspace := Acad.ActiveDocument.Modelspace;

//Llamar al metodo AddCircle() para crear un circulo con un radio de 10 unidades

Circle := Mspace.AddCircle(VarArrayRef(vPoint), 10.0);

Circle := Mspace.AddCircle(Vpoint, 10.0);

// Actualizar el circulo

Circle.Update;

END;

Ejemplo 2

Dibujo de objetos en AutoCad

PROCEDURE TForm1.Button1Click(Sender: TObject);

VAR

p1, p2, p3: OleVariant; *// Puntos inicial y final de la linea*

Mspace, Acad : OleVariant;

BEGIN

// Crear vector para las coordenadas

// VT_R8 = 5; { 8 byte real definido en /Source/RTL/Win/ActiveX.Pas }

p1 := VarArrayCreate([0, 2], VT_R8);

p2 := VarArrayCreate([0, 2], VT_R8);

p3 := VarArrayCreate([0, 2], VT_R8);

// Asignación de valores a los elementos del vector

p1[0] := 2.0; p1[1] := 4.0; p1[2] := 0.0; *// De 2,4,0*

p2[0] := 12.0; p2[1] := 14.0; p2[2] := 0.0; *// A 12,14,0*

p3[0] := 7.0; p3[1] := 8.0; p3[2] := 0.0;

// Obtener los objetos Aplicacion y el ModelSpace

TRY

// Ver si AutoCAD se esta ejecutando

Acad := GetActiveOleObject('AutoCAD.Application.2002');

EXCEPT

// Si no se esta ejecutando – Inicialarlo

Acad:= CreateOleObject('AutoCad.Application.14');

END;

// Poner a AutoCAD en primer plano (visible)

Acad.visible:= True;

Mspace := Acad.ActiveDocument.ModelSpace;

// Usar metodos de AutoCAD para dibujar una linea y 3 circulos

Mspace.AddLine(VarArrayRef(p1), VarArrayRef(p2)).Update;

Mspace.AddCircle(VarArrayRef(p1), 1.5).Update;

Mspace.AddCircle(VarArrayRef(p2), 1).Update;

```

MSpace.AddCircle(VarArrayRef(p3), 2.0).Update;
// Usar metodos de AutoCAD para dibujar otras formas y texto
MSpace.AddArc(VarArrayRef(p3), 1.2, 1, 2).Update;
MSpace.AddBox(VarArrayRef(p2), 5, 3, 2).Update;
MSpace.AddCone(VarArrayRef(p1), 1.3, 2).Update;
MSpace.AddCylinder(VarArrayRef(p3), 1.7, 1.5).Update;
MSpace.AddMtext(VarArrayRef(p3), 10, 'Borland Delphi !!!').update;
END;

```

Ejemplo 3

Envio de Datos a Excel

PROCEDURE TForm1.BitBtnToExcelOnClick(Sender: TObject);

VAR

WorkBk : _WorkBook; *// Definir un Libro (Excel)*

WorkSheet : _WorkSheet; *// Definir una Hoja (Excel)*

I, J, K, R, C : Integer;

IIndex : OleVariant;

TabGrid : Variant;

BEGIN

IF GenericStringGrid.Cells[0,1] <> " **THEN**

BEGIN

IIndex := 1;

R := GenericStringGrid.RowCount;

C := GenericStringGrid.ColCount;

// Creacion de vector

TabGrid := VarArrayCreate([0,(R - 1),0,(C - 1)],VarOleStr);

I := 0;

// Definir el ciclo para el llenado

REPEAT

FOR J := 0 **TO** (C - 1) **DO**

TabGrid[I,J] := GenericStringGrid.Cells[J,I];

Inc(I,1);


```

UNTIL
  I > (R - 1);
  // Conectarse con el servidor TExcelApplication
  XLApp.Connect;
  // Adicionar un libro a la Aplicacion Excel
  XLApp.WorkBooks.Add(xlWBatWorkSheet,0);
  // Seleccionar el primer libro
  WorkBk := XLApp.WorkBooks.Item[IIndex];
  // Definir la primera Hoja
  WorkSheet := WorkBk.WorkSheets.Get_Item(1) as _WorkSheet;
  // Asignar la matriz de Delphi a la Hoja asociada
  Worksheet.Range['A1',Worksheet.Cells.Item[R,C]].Value := TabGrid;
  // Adaptacion de la Hoja de Excel
  WorkSheet.Name := 'Customers';
  Worksheet.Columns.Font.Bold := True;
  Worksheet.Columns.HorizontalAlignment := xlRight;
  WorkSheet.Columns.ColumnWidth := 14;
  // Adaptar la primera Columna
  WorkSheet.Range['A' + IntToStr(1),'A' + IntToStr(R)].Font.Color := clBlue;
  WorkSheet.Range['A' + IntToStr(1),'A' + IntToStr(R)].HorizontalAlignment :=
xlHAlignLeft;
  WorkSheet.Range['A' + IntToStr(1),'A' + IntToStr(R)].ColumnWidth := 31;
  // Mostrar Excel
  XLApp.Visible[0] := True;
  // Desconectarse del Servidor
  XLApp.Disconnect;
  // En Disponibilidad la matriz de Delphi
  TabGrid := Unassigned;
END;
END;


```

A.6 LISTA DE COMPONENTES

Los componentes más utilizados en los ejercicios y aplicaciones realizadas en el presente texto son:

- De la paleta de componentes **Standard** se tiene los siguientes :


Etiquetas de texto: Label

Representado por el botón  cuyo nombre es **Label**. Mediante este control se puede mostrar un texto estático en el form, fijando su posición, color de letra, tamaño y, obviamente, el texto a mostrar.

Su propiedad más importante es **Caption**.

Ejemplo. `Label1.Caption:='Ingrese un número';`


El componente Edit

Uno de los controles que permiten la entrada de datos por teclado es el que se representa con el botón  que recibe el nombre de **Edit**. Cuya propiedad más importante es **Text**

Ejemplo. `Edit1.Text:='Borland Delphi';`


El control Memo

Las posibilidades del control **Edit** a veces no son suficientes para cubrir todas las necesidades de entrada de datos, especialmente cuando lo que se solicita al usuario del programa es un texto que puede ser más o menos extenso, ya que dicho control sólo facilita la edición de una línea.

Para dar cabida a estas necesidades, existe el control llamado **Memo**, cuyo botón en la paleta de componentes es  Podemos tratar este control de forma casi idéntica a un control **Edit**, con la diferencia de que es posible trabajar con una mayor extensión de texto, que puede estar distribuido en múltiples líneas. Cuya propiedad más importante es **Lines**. Que puede adicionarse texto desde el inspector de objetos o desde código.

Ejemplo. `Memo1.Lines.Add('Texto a mostrarse');`


El control Button

Un botón aparece como un área rectangular que contiene un texto en su interior y que, al pulsarlo, lleva a cabo una determinada acción. En Delphi este control aparece en la paleta de componentes con el icono  y recibe el nombre **Button**.

La propiedad más importante es **Caption**.

El evento más importante es **Button1Click**

El control CheckBox


Mediante los controles CheckBox, o Cajas de Selección, se puede obtener una entrada de datos. Este componente aparece en la paleta de componentes como el siguiente botón:  que se llama **CheckBox**. Este control permite al usuario activar o desactivar una cierta opción sin necesidad de escribir nada, bastará con que realice una pulsación sobre el control.

El título que aparecerá junto a la caja de selección será el que asignemos a la propiedad *Caption*, pudiendo existir una tecla de acceso rápido como en los botones y etiquetas de texto.


Habitualmente, este control puede aparecer en dos estados distintos, marcado o sin marcar. El estado actual lo podremos conocer mediante la propiedad **Checked**, que tomará el valor *True* si está marcado o el valor *False* en caso contrario.

El componente ListBox

En ocasiones, el número de datos que debe introducir el usuario de un programa o la cantidad de opciones distintas entre las que seleccionar es tan amplia, que el uso de múltiples controles de edición, botones de radio, etc., es poco práctico. En estos casos es mucho más cómodo usar una lista, que es un control capaz de contener cadenas de caracteres, cada una de las cuales aparece como elemento de la lista.

En caso de que el número de elementos exceda las dimensiones de la lista, en ésta aparecerán las barras de desplazamiento correspondientes. En Delphi la lista está representada por el control **ListBox**, seleccionando el botón  de la Paleta de componentes. Cuya propiedad principal es **Items**.

El componente ComboBox

Representado por 

Existen muchas ocasiones en donde el usuario del programa tiene que proporcionar datos que provienen de un conjunto finito y muy pequeño de posibles respuestas, esto significa que cada vez que se ejecute el programa, el usuario estará proporcionando las mismas respuestas.

El componente GroupBox




Este componente es otra forma standard de agrupamiento de componentes de programas en Windows, se usa para agrupar componentes relacionados dentro de una forma.

También se utiliza para separar áreas lógicas dentro de una ventana de Windows.

Cuya propiedad principal es **Ítems**.

El control RadioButton

A veces es necesario dar al usuario a elegir sólo una de las opciones disponibles, creando un grupo de opciones exclusivas entre sí.


Los requerimientos serán cubiertos con el control **RadioButton**, que aparece en la Paleta de componentes como . El aspecto de este control es circular, en lugar de un cuadrado, y sólo uno de los controles que insertemos en el form podrá estar activo.

El título que aparecerá a la derecha del control será facilitado como siempre en la propiedad *Caption*. El estado actual del botón, seleccionado o no, se lo conoce mediante la propiedad *Checked*, que será *True* en caso afirmativo o *False* en caso negativo.

- De la paleta de componentes **Win 32** se tiene :

El control RichEdit

Una de las grandes limitaciones del control **TMemo**, que usamos para permitir la entrada de texto por parte del usuario, es que no permite usar atributos diferentes para el texto, como tamaños o estilos, ni dispone de capacidad alguna de alineación, sangrado de párrafos, etc. Entre los controles de Windows 95 (paleta Win32) nos encontramos con uno,

llamado **TRichEdit**, situado en la paleta de componentes con este botón: , que en cierta manera funciona como un **TMemo**, con la diferencia de que permite editar texto en

formato Rich Text Format, o RTF, en el que es posible utilizar atributos en el texto, ajustar párrafos, etc.

- De la paleta de componentes **Adittional** se tiene :

El componente BitButton



Este componente visual permite realizar en forma fácil toda una serie de tareas comunes en Windows.

En principio es parecido al componente Button, pero que incluye un gráfico o bitmap, que lo hace mas agradable y visible al usuario.

Es en su propiedad **Kind**, en el inspector de objetos, donde se pueden definir cualquiera de sus diez opciones, como lo muestra la siguiente pantalla.

El componente StringGrid



Este componente visual permite realizar manejo de celdas al estilo de Excel, en el cual se admiten tipos de dato ‘String’; es decir texto.

Su propiedad más importante es:

Colcount: Numero de columnas.

Rowcount: Numero de filas.

Options: Que tiene varias subpropiedades para su respectiva edición.

Cells[j,i]: Que hace referencia a una celda del **StringGrid**, donde ‘j’ es la columna y ‘i’ es la fila.

BIBLIOGRAFIA GENERAL:

- Luis Joyanes Aguilar PROGRAMACION BASIC PARA MICROCOMPUTADORAS
2da Edicion, McGraw-Hill 1986
- R.H. Hammond, W.B. Rogers, J.B. Crittenden INTRODUCCION AL FORTRAN 77 Y
LA PC McGraw-Hill Interamericana de Mexico, S.A. de C.V. 1989
- Introduction to Algorithms (2nd ed.), Cormen, T. H., Leiserson, C. E., Rivest, R. L. y
Stein, C.
- Eufren Llanque DIAGRAMAS DE FLUJO, Ediciones UMSA La Paz –Bolivia
- Luis Joyanes Aguilar, Antonio Muñoz Clemente, BORLAND DELPHI Iniciación y
Referencia, McGraw-Hill/Interamericana de España S.A.U.
- Francisco Charte, PROGRAMACION CON DELPHI, Ediciones Anaya Multimedia
S.A.
- Victor Moral, DELPHI 4, Prentice Hall Iberia S.R.L.
- Victor A. Melchor Espinoza DELPHI 8 Editorial MACROPERU.
- Ayuda del programa (BORLAND DELPHI HELP)

REFERENCIAS BIBLIOGRAFICAS DE LA WEB

Paginas de Internet:

- http://www.Historia de la Computación - Monografias_com.htm (abril/ 2006)
- <http://www.Historia de la Computación.htm> (abril/ 2006)
- <http://www.Historia de la Computación2.htm> (abril/ 2006)
- <http://www.Historia de la computación4.htm> (mayo/ 2006)
- <http://www.La Computación en el Tiempo.htm> (mayo/ 2006)
- <http://www.swissdelphicenter.ch/en/showcode.php?id=1155Historia de la computación4.htm> (mayo/ 2006)
- <http://www.Linea del tiempo de la computacion.htm> (mayo/ 2006)
- <http://www.BIOS.htm> (mayo/ 2006)
- <http://www.BIOS - Wikipedia, la enciclopedia libre.htm> (mayo/ 2006)
- <http://www.Discos duros y particiones.htm> (mayo/ 2006)
- <http://www.Disco duro - Wikipedia, la enciclopedia libre.htm> (mayo/ 2006)
- [http://www.El Disco Duro \(HD\) - Monografias_com.htm](http://www.El Disco Duro (HD) - Monografias_com.htm) (mayo/ 2006)
- <http://www.usuarios1.htm> (mayo/ 2006)
- [http://www.Historia de la Computación \(Informática\) - ilustrados_com.htm](http://www.Historia de la Computación (Informática) - ilustrados_com.htm) (mayo/ 2006)
- <http://www.microsoft.com/windowsxp/default.asp> (mayo/ 2006)
- <http://es.wikipedia.org/wiki/Computadora> (mayo/ 2006)
- http://www.delphi3000.com/articles/article_3228.aspHistoria de la Computación.htm (mayo/ 2006)
- <http://es.wikipedia.org/wiki/Algoritmo> (mayo/ 2006)
- <http://www.algoritmica.com.ar> (mayo/ 2006)
- <http://www.algoritmica.com.ar/apu/est/main.html> (mayo/ 2006)
- <http://lenguajes-de-programacion.com/programacion-de-algoritmos.shtml> (mayo/ 2006)
- http://es.wikipedia.org/wiki/Diagramas_de_flujo (mayo/ 2006)
- <http://www.monografias.com/trabajos14/flujograma/flujograma.shtml> (mayo/ 2006)

- <http://www.hci.com.au/hciste2/toolkit/flowchar.htm> (mayo/ 2006)
- <http://www.iso.org/iso/en/aboutiso/introduction/index.html> (mayo/ 2006)
- http://www.delphi3000.com/articles/article_3228.asp Historia de la Computación.htm (mayo/ 2006)
- <http://www.metodologia.htm> (mayo/ 2006)
- <http://www.freepascal.org/sdown.html> (mayo/ 2006)
- <http://es.wikipedia.org/wiki/Delphi> (mayo/ 2006)
- <http://www.borland.com/delphi> (junio/ 2006)
- <http://www.programacionfacil.com> (junio/ 2006)
- <http://www.mailxmail.com/curso/informatica/delphi> (junio/ 2006)
- <http://www.freepascal.org/sdown.html> (junio/ 2006)
- <http://es.wikipedia.org/wiki/Delphi> (junio/ 2006)
- <http://www.delphi 5.htm> (junio/ 2006)
- <http://www.programacionfacil.com> (junio/ 2006)
- <http://www.elguille.info/delphi/indice.htm> (junio/ 2006)
- <http://www.marcocantu.com/> (junio/ 2006)
- <http://www.marcocantu.com/epascal> (junio/ 2006)
- <http://www.marcocantu.com/epascal/Spanish/default.htm> (junio/ 2006)
- <http://www.elguille.info/delphi/apuntesDelphi/apuntesDelphi.htm> (junio/ 2006)