UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
"ARAGON"

# " MEJORAMIENTO DEL SISTEMA DE ADQUISICION DE DATOS EN EL REGISTRO DE CAVERNAS SUBTERRANES, MEDIANTE LA SUSTITUCION DEL SISTEMA DE IMPRESION - TECLADO DE LA UNIDAD SE SERVICIOS CIBERNETICOS POR UNA COMPUTADORA PORTATIL "



**TESIS PROFESIONAL**

Que para obtener el Titulo de:

**INGENIERO EN COMPUTACION**

P r e s e n t a:

**JOSE MANUEL CHANDOMI SALUD**

San Juan de Aragón, Edo. de Méx. 1995


**UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso**

UNIVERSIDAD NACIONAL AVENIDA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGÓN
DIRECCION

JOSE MANUEL CHANDOMI SALUD
P R E S E N T E .

En contestación a su solicitud de fecha 27 de abril del año en curso, relativa a la autorización que se le debe conceder para que la profesora, Ing. SILVIA VEGA MUYTOY pueda dirigirle el trabajo de Tesis denominado " MEJORAMIENTO DEL SISTEMA DE ADQUISICION DE DATOS EN EL REGISTRO DE CAVERNAS SUBTERRANEAS, MEDIANTE LA SUSTITUCION DEL SISTEMA DE IMPRESION-TECLADO DE LA UNIDAD DE SERVICIOS CIBERNETICOS POR UNA COMPUTADORA PORTATIL ", con fundamento en el punto 6 y siguientes, del Reglamento para Exámenes Profesionales en esta Escuela, y toda vez que la documentación presentada por usted reune los requisitos que establece el precitado Reglamento; me permito comunicarle que ha sido aprobada su solicitud.

Aprovecho la ocasión para reiterarle mi distinguida consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPIRITU"
San Juan de Aragón, Mex., 1 de Mayo de 1984
EL DIRECTOR

M en I CLAUDIO C. MERRIFIELD CASTRO

c c p Lic. Alberto Ibarra Rosas, Jefe de la Unidad Académica.
c c p Ing. Silvia Vega Muytoy, Jefe de la Carrera de Ingeniería en Computación.
c c p Asesor de Tesis.

CCMC'AIR'lla.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
CAMPUS ARAGÓN

UNIDAD ACADÉMICA

Ing. SILVIA VEGA MUYTOY
Jefe de la Carrera de Ingeniería en Computación,
P r e s e n t e

En atención a la solicitud de fecha 9 de octubre del año en curso, por la que se comunica que el alumno JOSE MANUEL CHANDOMI SALUD, de la carrera de INGENIERO EN COMPUTACION, ha concluido su trabajo de investigación intitulado "MEJORAMIENTO DEL SISTEMA DE ADQUISICION DE DATOS EN EL REGISTRO DE CAVERNAS SUBTERRANEAS, MEDIANTE LA SUSTITUCION DEL SISTEMA DE IMPRESION-TECLADO DE LA UNIDAD DE SERVICIOS CIBERNETICOS POR UNA COMPUTADORA PORTATIL", y como el mismo ha sido revisado y aprobado por usted se autoriza su impresión; así como la iniciación de los trámites correspondientes para la celebración del examen profesional.

Sin otro particular, le reitero las seguridades de mi distinguida consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPÍRITU
San Juan de Aragón, México., 13 de octubre de 1995
EL JEFE DE LA UNIDAD

Lic. ALBERTO IBARRA ROSAS

c c p Asesor de Tesis.
c c p Interesado.

AIR/vr

## DEDICATORIA

Con eterno agradecimiento, respeto y amor, a los pilares de mi formación como ser humano:

mis padres :

**Ing. José H. Chandomi Palacios;** *por tus consejos siempre oportunos,*
**Sra. Hortensia Salud Toledo;** *por tu ternura y por tu ejemplo para afrontar los problemas,*

y mis hermanos:

**Griselda;** *por la confianza y por el apoyo que siempre me has brindado*
**Angel Ernesto,**
**Martha,**
**Hortensia,**
**Julio César y**
**Ricardo.**

a **Ana Lilia**, *por su compañía y apoyo moral.*

## AGRADECIMIENTOS

Quiero expresar mis sinceros agradecimientos al grupo de instrumentación de la División de Geofísica de Explotación del I.M.P., por su apoyo en la elaboración de este trabajo; en particular al **Ing. Alberto Flores Roa** por sus valiosas recomendaciones en programación.

A mi asesor de tesis, **Ing. Silvia Vega Muytoy**, por su colaboración.

Así también a todas aquellas personas que siempre me apoyaron de una u otra forma durante mis estudios universitarios, en especial a las familias:

**Morales Salud,**
**Calvo Pinacho,**
**Díaz Calvo,**
**Hernández Ramírez y**
**Falcón Salvador.**

# RESUMEN

El propósito de éste resumen es la de exponer al lector en una forma muy breve los problemas que dieron origen a la elaboración de éste trabajo, así como de la solución propuesta con el único fin de colocarlo en una posición más cómoda para una más rápida comprensión de los conceptos e ideas que se expondrán.

Siendo nuestro país uno de los más importantes productores de petróleo en el mundo, es necesario que cuente con una infraestructura sólida que le permita almacenar grandes cantidades de crudo. Para tal fin se cuentan con tanques almacenadores tanto en superficie como en el subsuelo.

El almacenamiento subterráneo se lleva a cabo utilizando cavernas en domos salinos previamente perforados. Durante y después de la creación de éstas cavidades es necesaria la supervisión de la geometría y dimensiones que la caverna va adquiriendo. En tal actividad se hace uso de una herramienta que utiliza el principio del sonar, la cual se introduce dentro de la caverna. La distancia entre el transductor que dispara el haz sónico y la pared de la cavidad se calcula tomando en cuenta el tiempo que tarda en regresar el *ECO* cuando el sonido incide en las paredes. El transductor desde donde se realiza el disparo se hace girar 360 grados para cubrir toda la superficie posible. Este proceso se realiza una y otra vez, haciendo bajar la herramienta cada cinco o diez metros, hasta que se haya hecho un barrido completo de la caverna.

La información generada de esta forma se envía a superficie en donde se encuentra una computadora ***(CSU)*** encargada de convertir toda esta información a formatos entendibles para los técnicos que la operan. La información se imprime en un papel termosensitivo colocado previamente en la ***Unidad de Impresión Teclado*** ó ***KPU***, que es la unidad con la cual los operadores se comunican con la CSU.

Al rollo de papel termosensitivo que contiene los datos de las cavernas se le denomina ***SCROLL***. Los datos contenidos en él tienen que ser capturados manualmente en una computadora para que sean procesados y se obtenga finalmente gráficas en 2D y 3D concernientes a la geometría de las cavidades.

Dentro de este proceso existen ciertas etapas que se pueden mejorar si en vez de la Unidad de Teclado-Impresora se utiliza una computadora portátil.

Para justificar ésta substitución, imaginemos primero que los datos, en vez de ser impresos en la KPU son almacenados en la memoria de la computadora portátil. Tal sustitución ofrece varias ventajas a conocer:

* **Primero**; ya no será necesaria la utilización de papel termosensitivo. La importancia de ésta ventaja radica en los costos del papel dada sus características y el origen de su elaboración.

* **Segundo**; se evita la captura de los datos del SCROLL, actividad muy tediosa pues son muchos los datos que hay que introducir.

* **Tercero**; la confiabilidad del sistema se eleva pues los archivos para el programa de generación de gráficos ya no son generados por los capturistas sino que son obtenidos directamente de la memoria de la computadora portátil.

* **Cuarto**; lo anterior redunda en menores tiempos de procesamiento.

* **Quinto**; mediante un programa adecuado dentro de la computadora portátil se puede lograr un mejor ambiente de trabajo entre el usuario y la CSU que el que nos brinda la KPU

* **Sexto**; además el programa contará con un editor de texto básico para que el usuario cambie toda aquella información que a su criterio esté errónea.

Todo lo expuesto crea un nueva filosofía de trabajo mucho más eficiente y amigable, cuyo propósito es el de hacer más fácil el trabajo de la gente que opera las máquinas en el registro de cavernas subterráneas.

# INDICE

## INDICE DE FIGURAS

# INTRODUCCION

Los objetivos de éste trabajo teórico-práctico son los siguientes:

a) Presentar un panorama general de la problemática que dio origen al desarrollo de ésta tesis, exponiendo los pros y contras que se tenían antes y después de hallar una solución.

b) Describir algunas técnicas estándares para el análisis de sistemas, a fin de enmarcar la aplicación de esas técnicas.

c) Presentar la implementación del sistema analizado y diseñado, utilizando para tal fin pseudocódigos y códigos en lenguaje C.

El documento está formado por tres capítulos. El orden en que se presentan éstos sigue más o menos el mismo orden que se sigue en un esquema en cascada para el desarrollo de software. Así pues, cada capítulo contempla:

- **Capítulo Primero :** ***Antecedentes.*** Generalidades sobre el medio ambiente en donde se desenvolverá el sistema ha desarrollar; planteamiento del problema y de su solución.

- **Capítulo Segundo :** ***Análisis de requerimientos.*** Explicación y aplicación de algunas técnicas de software para empezar a atacar la problemática. Análisis en forma individual de los subsistemas que integrarán el sistema de software a desarrollar.

- **Capítulo Tercero :** ***Diseño e implementación del sistema.*** Descripción y diseño de los principales pasos ha seguir para alcanzar el funcionamiento deseado de cada subsistema. Presentación de algunos códigos en lenguaje C a fin de reforzar la teoría que acompaña la explicación de cada subsistema.

Posterior a los tres capítulos anteriores, se presentan las conclusiones derivadas del desarrollo de la tesis, haciendo, en primer término, referencias a los objetivos que se persiguen y mencionando a continuación los logros alcanzados al finalizar el trabajo.

La estructura de la tesis pretende que el lector entienda en forma rápida lo que se pretende resolver. Con ésto lograremos que se comprendan, sin mucho esfuerzo, la información que le sigue al planteamiento del problema.

Aunque el lenguaje utilizado en éste trabajo es técnico, siento que no es requisito el tener conocimientos muy avanzados en computación, pues he tratado ser lo más claro en mis explicaciones, apoyándome siempre que lo consideré necesario, de diagramas, dibujos y algunas aclaraciones textuales.

# CAPITULO 1

## ANTECEDENTES

Dentro de la industria petrolera un factor muy importante es el almacenamiento de hidrocarburo. Nuestro país, como uno de las principales naciones petroleras en el mundo producía en 1992, 2,400 MMBD [1]. Con una producción similar la capacidad de almacenamiento en 1985 alcanzaba sólo para cinco o seis días, lo que resulta problemático cuando por fenómenos meteorológicos se tenían que cerrar los puertos de exportación, provocando que la producción de crudo en algunos pozos, se interrumpiera, sufriéndose grandes pérdidas económicas.

Por este motivo se estudió la posibilidad de disponer a mediano plazo con una infraestructura que permitiera incrementar el almacenamiento de crudo hasta 15 días, a través de cavernas subterráneas en domos salinos y así poder satisfacer la demanda interna y externa de México.

El almacenamiento de crudo en cavernas subterráneas es una técnica creciente en nuestro país; económica, por los bajos costos que implican su construcción (comparados con los gastos de construcción de tanques de almacenamiento superficiales) y útil, por el volumen de hidrocarburo que pueden alojar.

[1] Miles de millones de barriles diarios.

La aplicación de la tecnología de almacenamiento subterráneo de hidrocarburos se inició en 1985 con el proyecto Tuzandepetl, el cual consistió en la creación de varias cavernas con las siguientes características [PER 92]:

| | |
|---|---|
| Número de cavidades en la masa salina: | 12 |
| Diámetro programado para cada caverna: | 20m |
| Profundidad promedio de cada cavidad: | 950m |
| Espaciamiento entre pozos: | 225m |
| Volumen de almacenamiento por cavidad: | 146,000 $m^3$ |
| Volumen total de almacenamiento (12 cavidades) | 1,800,000 $m^3$ |

El elevado volumen de hidrocarburo que se puede almacenar en ellos, ha sido un factor muy importante para que nuestro país, a través de instituciones de investigación, busque mejorar e innovar técnicas en éste campo tecnológico. Así, en años pasados, en el Instituto Mexicano del Petróleo se desarrolló un instrumento sónico capaz de medir las dimensiones internas de las cavidades. El instrumento fué bautizado con el nombre de *herramienta de fondo* [2] del sistema SONIMP, y desde entonces hasta la fecha ha venido funcionando en conjunto con la Unidad de Servicios Cibernéticos (CSU), la computadora basada en un microprocesador PDP-11 y encargada de controlar la operación de las distintas herramientas de fondo con que trabaja.

La CSU como todo sistema de cómputo, está compuesto de unidades de salida y unidades de entrada. A continuación se dará una breve explicación de la forma en que trabaja la Unidad de Servicios Cibernéticos con sus diferentes

[2] Es el instrumento que se introduce dentro de la caverna mediante un cable electromecánico y que efectúa mediciones de los factores internos que presenta dicha caverna

periféricos, poniendo especial énfasis en uno: la Unidad de Impresión-Teclado (KPU) pues es ésta la que dió origen al desarrollo de la presente tesis.

## 1.1 Unidad de Servicios Cibernéticos

La CSU(Cyber Service Unit) es un sistema de adquisición y procesamiento de datos de registro de pozos, constituidos por una computadora digital y varios subsistemas periféricos, que se alojan dentro de un camión para transportarlos hasta el lugar en donde se va ha efectuar un registro. El sistema opera bajo control de un conjunto de programas de computadora, que difieren según la herramienta que se vaya a utilizar y las funciones que se vayan a realizar.

El cuerpo principal de instrucciones e información llamado SOFTWARE DEL SISTEMA, se encuentra grabado en disco duro (aunque se pueden utilizar cintas y cartuchos magnéticos) y desde éste se alimenta a la memoria de la computadora al inicio de cualquier servicio, manteniéndose ahí el software en espera de alguna secuencia de operación.

La circuitería o componentes físicos llamados HARDWARE DEL SISTEMA, prácticamente no varía independientemente de la herramienta de fondo que se use; el único cambio en los componentes de superficie es con el objeto de brindar la interfase adecuada para cada distinta herramienta de fondo.

En la Fig.1 se muestra un esquema del CSU. Los diferentes subsistemas se encuentran ensamblados en 3 anaqueles, el de la izquierda contiene las siguientes unidades:

- Unidad de cinta magnética ( MTU: Magnetic Tape Unit ).
- Unidad electrónica para óptica ( OEU: Optical Electronics Unit ).
- Unidad central de proceso No. 1 y 2 ( CPU: Central Processor Units ).
- Unidad de ventilación ( VBU: Ventilation Blower Unit ).

el anaquel central contiene las siguientes unidades:

- Unidad de cinta en cartucho ( CTU: Cartridge Tape Unit ).
- Unidad general de electrónica ( GEU: General Electronics Unit ).
- Unidad de teclado e impresión ( KPU: Keyboard Printer Unit ).
- Unidad auxiliar de computadora ( CAU: Computer Auxiliary Unit ).

y el anaquel de la derecha contiene las siguientes unidades:

- Unidades ópticas de película No. 1 y 2 ( OFU: Optical Film Units ).
- Unidad de interfase para herramienta ( TIU: Tool Interface Unit ).
- Unidad de monitor óptico ( OMU: Optical Monitor Unit ).
- Unidad de poder para herramienta ( TPU: Tool Power Unit ).

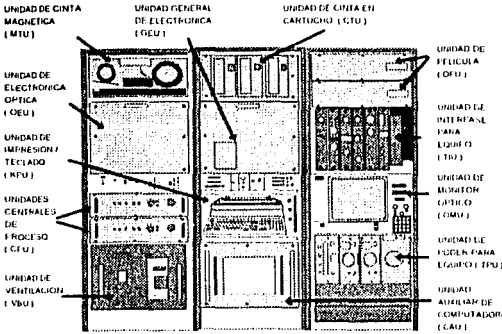



FIGURA 1 COMPONENTES DE LA UNIDAD DE SERVICIOS CIBERNETICOS (CSU)

## 1.2 Unidad de Teclado-Impresora: KPU (Keyboard Printer Unit)

Esta unidad es el principal medio de comunicación entre el operador y el procesador central. Funcionalmente está compuesto de dos subunidades, tal y como su nombre lo indica: con el teclado se mandan comandos y datos al procesador, y con la impresora se verifica que las entradas al procesador por medio del teclado fueron recibidas y correctamente interpretadas, así como para mandar mensajes y datos del procesador al operador. La impresora tiene un impresor térmico el cual está montado en una cabeza móvil. La cabeza contiene una matriz de puntos de 5X7, cada uno de los cuales puede ser energizado individualmente con el fin de producir los caracteres en el papel termo sensible usado por la impresora.

El teclado manda un código a la CPU por medio de la CAU (ver fig. 2), el interruptor de unibús y la interfase serie a la entrada del procesador central. El código es el ASCII (American Standard Code for Information Interchange) y representa caracteres enviados por el teclado. El teclado está compuesto por 55 teclas con las cuales se generan un total de 128 caracteres compuestos cada uno de 7 bits.

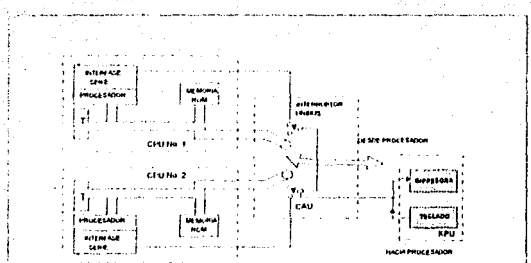



FIGURA 2 INTERCONEXION ENTRE LA UNIDAD DE SERVICIOS CIBERNETICOS Y SU UNIDAD DE TECLADO/IMPRESORA

El procesador central manda datos a la impresora para verificar que las entradas del teclado son correctas y para mandar mensajes al operador. El procesador central manda el código por medio de la interfase serie y la CAU; una vez que el código es recibido por la impresora, ésta lo interpreta e imprime o si se trata de una instrucción la ejecuta.

Aunque el teclado y la impresora constituyen una misma unidad y están dispuestos en una misma entidad física, no están conectados eléctricamente.

La fig. 3 siguiente muestra el aspecto físico que presenta la unidad en cuestión, así como la localización de algunos de los interruptores que controlan su funcionamiento [Schlumbeger 81].

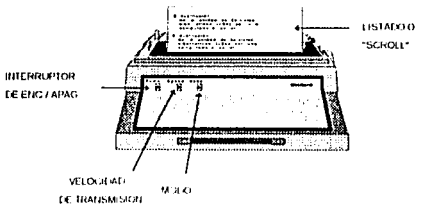



FIGURA 3. UNIDAD DE TECLADO / IMPRESORA (KPU)

### 1.2.1 Controles de la KPU

Algunos de los principales controles están identificados en la fig. 3, el resto no se aprecian por estar en la parte posterior de la unidad.

- **POWER ON/OFF:** Conecta la alimentación a la KPU, una luz roja a un lado del interruptor prende cuando la alimentación es aplicada.

**MODE:** Selector de tres posiciones:

- **LOCAL:** La KPU trabaja como una máquina de escribir normal, inhabilita las conexiones I/O, imprime directamente el caracter dado por el teclado.

- **FULL:** Los datos generados en el teclado se transmiten a la computadora y esta regresa al impresor. Esta es la posición normal de operación.

-**HALF:** Los datos generados en el teclado se mandan simultáneamente a la computadora y a la impresora, al oprimir un caracter en el teclado, éste será impreso 2 veces.

- **SPEED:** Este selector determina la velocidad de transmisión y recepción de datos, normalmente se trabaja a 1200 baudios.

- **KEY BOARD MODE:** Este selector se localiza bajo la puerta de acceso al papel; en la posición izquierda (NUM) selecciona el modo numérico, en la posición central (STD) selecciona el código USASCII de 128 caracteres y en la posición de la derecha (TTY) selecciona el modo TTY 33; normalmente se usa en ésta posición.

- **PARITY ON/OFF:** Si se selecciona la posición OFF , la información transmitida es checada por paridad, pero la información que llega no es checada, esta posición también desconecta el indicador de error. En la posición ON tanto la información transmitida como la que entra es verificada por paridad par o impar

(dependiendo de la posición del selector **PARITY ODD EVEN**). Normalmente se trabaja en la posición OFF.

- **PARITY EVEN/ODD:** En la posición EVEN se trabaja con paridad PAR para la información transmitida y recibida. En la posición ODD se trabaja con paridad IMPAR. Si ocurre un error de paridad, prenderá el indicador de error y permanecerá prendido hasta que sea apagado manualmente.

- **PRINT DENSITY:** Permite controlar la intensidad de la impresión, moviéndola en sentido contrario al de las manecillas del reloj se tendrá una impresión más fuerte.

- **CARRIAGE RETURN:** Esta tecla hace que la cabeza de impresión regrese a la primera posición de impresión (parte izquierda), la CPU ejecuta la instrucción.

- **REPEAT (REPT):** Causa que cualquier tecla oprimida subsecuentemente sea repetida a la velocidad de transmisión seleccionada.

- **LINE FEED:** Causa que el papel avance una línea.

- **SHIFT:** Esta tecla hace que el teclado produzca los caracteres superiores de la tecla, también permite quitar el seguro (SHIFT LOCK) cuando está puesto.

- **CTRL:** Genera todos los códigos de control especificados en el código ASCII, se puede lograr que la cabeza de impresión borre el caracter escrito

accionando simultáneamente las teclas CTRL H, la tecla rubout (RUB) o la tecla BACKSLASH (\).

- **BREAK:** Genera un espacio. Este es usado por algunos programas para terminar la operación.

-**SHIFT LOCK:** Cuando se acciona y se suelta causa que todas la teclas transmitan los caracteres superiores. Para regresar el teclado al modo normal de operación se debe accionar la tecla SHIFT.

- **PAPER FEED:** Causa que el impresor avance 2 líneas, no se transmite ningún código, la luz roja en la tecla prenderá cuando el teclado esté operando en el modo TTY 33.

- **ESC:** Borra toda una línea, también se puede hacer con la tecla (-).

## 1.3 Interrelación de la CSU y la KPU en el registro de cavernas subterráneas

Para la creación de las cavernas subterráneas es necesario primero perforar un pozo dentro de una masa salina previamente seleccionada, hasta una profundidad de 1000m (operación que dura aproximadamente 100 días). Posteriormente se continúa con la formación de la cavidad a una profundidad programada inferior a los 1000m., mediante fases de lixiviación consistentes en inyección de agua dulce y extracción de salmuera.

Una vez que los ingenieros de perforación consideran conveniente la interrupción de los procesos de lixiviación, se procede al estudio de la forma, dimensión y volumen que presenta la cavidad. Para tal fin se utiliza la herramienta sónica SONIMP que fué diseñada por ingenieros del Departamento de Instrumentación del Instituto Mexicano del Petróleo con asesoría técnica de ingenieros de la empresa francesa Schlumberger.

La herramienta de fondo SONIMP basa su funcionamiento en el principio del sonar. El procedimiento de operación consiste en introducir la herramienta en la cavidad efectuando paradas periódicas cada 1, 5 ó 10 metros, dependiendo de las irregularidades que presente la pared de la caverna. En cada estación (parada) se realiza un barrido horizontal girando un transductor (encargado de enviar y recibir el haz sónico) hasta completar 360 grados. Este procedimiento se continúa hasta llegar al fondo. La herramienta dispone de otro transductor que se hace necesario cuando se desea conocer la profundidad de la cavidad y para determinar la variación de la forma de su pared.

El lapso de tiempo comprendido desde el momento en que el transductor envía un haz sónico y el momento en que se recibe éste haz de regreso, una vez que haya incidido y se haya reflejado en las paredes de la caverna, es directamente proporcional a la distancia que existe entre las paredes y el transductor. Tanto el tiempo registrado como el *azimuth* [3] de disparo del haz son datos que se envían a superficie para que sean procesados.

[3] Denota los grados que gira el transductor, tomando como referencia el norte magnético.

Las señales provenientes del equipo de fondo llegan primeramente a la unidad de interfase (TIU) en la que se encuentra colocado el hardware correspondiente a la herramienta de fondo que se opera.

La GEU se encarga de convertir las señales analógicas de registro a una forma digital compatible con la computadora. La CPU recibe estos datos de registro de la GEU y se encarga de procesarlos de acuerdo al programa, a los datos de calibración y a los datos proporcionados por el operador hasta obtener la información correcta del registro y enviarla posteriormente a los dispositivos periféricos de salida (OFU, MTU, KPU, OMU), en los cuales son presentados en forma legible para la persona operadora.

Durante el tiempo que dura el registro de una caverna subterránea, la información se imprime en el papel termosensible de la Unidad de Impresión Teclado (KPU), dando origen a lo que se conoce como "SCROLL" (ver fig. 4). De esta forma el SCROLL se convierte en el medio por el cual el operador conoce la profundidad, *radio* [4] y azimuth en los cuales fué disparado el haz sónico, y por el cual también lleva el seguimiento de las operaciones realizadas antes del registro.

[4] Distancia entre la pared y el transductor de la herramienta sonar.

SCAN DEPTH: 724.238 M

| ETIM S | AZIH DEG | TRAD M |
|---|---|---|
| AZIMUTH DISAGREES WITH SELECTED VALUE | | |
| 5.03900 | 15.00 | 2.412 |
| 9.99000 | 30.00 | 3.431 |
| 14.9400 | 45.00 | 3.173 |
| 19.8890 | 60.00 | 2.365 |
| 24.8390 | 75.00 | 3.244 |
| 29.7890 | 90.00 | 2.994 |
| 34.7390 | 105.0 | 2.701 |
| 39.6890 | 120.0 | 3.373 |
| 44.6391 | 135.0 | 5.792 |
| 49.5891 | 150.0 | 3.988 |
| 54.5391 | 165.0 | 2.892 |
| 59.4892 | 180.0 | 2.837 |
| 64.4392 | 195.0 | 2.431 |
| 69.3893 | 210.0 | 2.332 |
| 74.3393 | 225.0 | 2.343 |
| 79.2894 | 155.0 | 2.376 |
| 84.2394 | 255.0 | 2.271 |
| 89.1895 | 271.0 | 2.314 |
| 94.1395 | 288.0 | 2.298 |
| 99.0896 | 305.0 | 2.501 |
| 104.041 | 324.0 | 2.878 |
| 108.990 | 340.0 | 2.943 |

LOG STOPPED
01-MAR-1992 17:05

FIGURA 4. DATOS EN UN SCROLL CORRESPONDIENTES A UNA ESTACION A 724 M.

## 1.4 Problemática actual y su solución

Una vez finalizado el registro de la caverna el paso a seguir es capturar todos los datos impresos en el SCROLL. La captura se hace en forma manual, desechando toda aquella información irrelevante para los programas de interpretación. Estos programas generan cinco tipos de gráficas en alta resolución y a color:

1.- Gráficas de secciones horizontales a diferentes profundidades.

2.- Gráficas de secciones verticales a diferentes orientaciones.

3.- Gráficas tridimensionales con diferentes perspectivas.

4.- Gráficas de curva de volumen contra profundidad.

5.- Gráficas de curva de volumen incremental respecto a incremento de profundidad.

Con la generación de estas gráficas el proceso en el sistema SONIMP finaliza. Las etapas detalladas en los puntos 1.3 y 1.4 referentes al sistema SONIMP se muestran en la fig.5 en forma de diagramas de bloques.

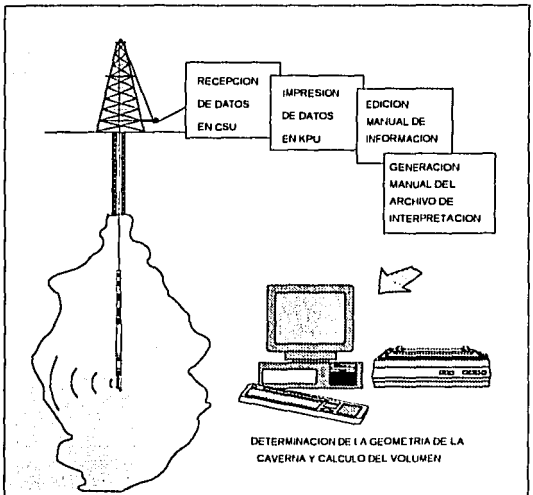



FIGURA 5. DIAGRAMA A BLOQUES QUE MUESTRA LA FORMA EN QUE OPERA EL SISTEMA SONIMP ACTUAL.

Una de las tareas mas tediosas y que por lo tanto resulta una desventaja dentro de éste proceso es la captura de la información (generación manual del archivo de interpretación ), veamos porque. En la fig. 4 se muestra una pequeña porción de un SCROLL, correspondiente a una estación a 724.238 m. de profundidad. En él, los datos a capturar manualmente son: 1) la profundidad, 2) el azimuth ( AZIH ) y 3) el radio ( HRAD ). Considerando que en promedio cada registro contiene de 60 a 70 estaciones, la cantidad de datos por capturar resulta

muy grande, con la posibilidad de que el capturista cometa algún error al momento de accesar toda ésta información en una computadora.

Otra desventaja que presenta el sistema SONIMP se debe a la utilización de papel térmico para almacenar la información. Este papel resulta muy caro dado sus características, además de que se tiene que importar.

Una tercera desventaja es la desorganización que se presenta en el SCROLL cuando se genera un dato erróneo. Para ser más claros, hagamos referencia de nueva cuenta a la fig. 4 y supongamos que a una azimuth de "x" grados se obtuvo un valor incongruente con el resto de los datos, por lo que debemos repetir la operación a esa profundidad y a esa azimuth. Para repetir la operación debemos esperar a que la herramienta haya girado los 360 grados que cubren la superficie de la caverna a esa profundidad. Una vez logrado lo anterior entonces sí podemos continuar.

Como resultado de éste problema se obtiene al final datos aislados o mejor dicho, datos que no se encuentran en el espacio al que originalmente pertenecían. Para relacionar esos datos aislados con el conjunto de información a la que pertenecen, se debe dibujar una línea que los relacione para que la persona que los vaya a capturar sepa a donde pertenecen y no cometa errores.

Por estas razones se decidió sustituir a la KPU por una computadora portátil (LAPTOP) que sea de fácil transporte y que cargada con un sistema de software conveniente de solución a los problemas anteriormente planteados.

Tal computadora se comunicará con la CSU vía puerto serie utilizando una interfaz RS-232.

Toda la información transmitida tanto por el usuario a través de la LAPTOP como por la CSU será desplegada en pantalla y guardada en disco rígido.

Esta última característica eliminará la actividad de captura manual de los datos del registro que se realiza actualmente.

Por otro lado, toda la información recibida de la CSU será almacenada en memoria RAM y controlada por un programa editor en LAPTOP, para que el usuario pueda consultarla en el momento en que lo desee, simplemente presionando las teclas de desplazamiento vertical (flecha arriba y flecha abajo). Además de todas éstas ventajas, el programa tendrá opciones para imprimir la información, cambiar los parámetros de comunicación y generar archivos en un formato específico para los programas de interpretación (de generación de gráficas).

El nuevo sistema SONIMP tendría entonces un nuevo flujo de información, que en forma esquemática quedaría representado de la siguiente forma:

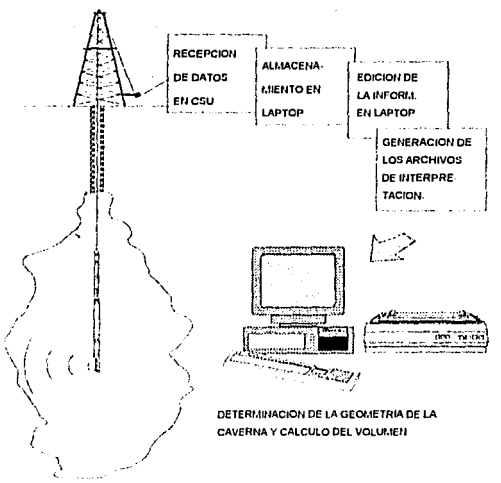



FIGURA 6 NUEVO PROCESO EN EL SISTEMA SONIMP MEDIANTE EL USO DE UNA LAPTOP

Las ventajas que ofrecerá esta nueva forma de funcionamiento del sistema SONIMP serán:

*- MENOR TIEMPO DE OPERACION:*

El tiempo que se tarda en desplegar en pantalla la información una computadora AT es mucho menor al que se puede tardar una impresora de matriz de puntos (como es el caso de la KPU) imprimiendo resultados.

*- MAYOR CONFIABILIDAD:*

Debido a que no será necesario capturar manualmente los datos del registro para generar los archivo de interpretación, la posibilidad de error disminuirá porque la confiabilidad estará en función de la calidad de transmisión entre la CSU y la KPU y no en función a errores humanos generados por el capturista.

*- AHORRO DE PAPEL:*

Dado que no será necesario el uso de la KPU, no se utilizará más papel termo sensitivo, tampoco se tendrán que comprar cabezas térmicas. Esto redundará en menores costos de operación, pues ambos insumos tienen que ser comprados en el extranjero, aparte de que tienden a ser tecnología obsoleta.

*-RAPIDEZ DE ENTREGA:*

La automatización en la adquisición de datos disminuirá notablemente el tiempo de entrega de información útil para los programas de interpretación y por ende los cinco tipos de gráficas que ofrece el sistema SONIMP se generarán en menor tiempo.

Cabe señalar que aunque las características de operación del programa de adquisición de datos estarán orientados a trabajar con el sistema SONIMP, no se limitará su utilización con otras herramientas, de tal forma que el programa pueda brindar apoyo en cursos de capacitación del uso y manejo de la CSU que se imparten en el IMP.

## 1.5 Objetivos

**Objetivo General:**

Sustituir a la Unidad de Impresión-Teclado ( KPU ) por una computadora portátil LAPTOP para mejorar los procesos de adquisición e interpretación de datos del sistema SONIMP en el estudio de cavernas subterráneas.

**Objetivos específicos:**

1.- Analizar el mecanismo de enlace existente entre el equipo CSU y su unidad KPU partiendo del conocimiento previo de que se trata de una comunicación asíncrona.

2.- Diseñar un programa en base al análisis anterior para comunicar al CSU y a la computadora portátil.

3.- Analizar la interfaz USUARIO-KPU-CSU para la elaboración de un programa capaz de desplegar, guardar, editar y convertir la información proveniente de una herramienta sonar.

# CAPITULO 2

## ANALISIS DE REQUERIMIENTOS

Una vez visualizado y planteado el problema, es tiempo de empezar a atacarlo de una manera formal; es decir, utilizando reglas, técnicas y metodologías que nos permitan realizar el mejor análisis posible para optimizar los resultados.

El análisis como elemento fundamental dentro del ciclo de vida de los sistemas de software, no es exclusivo de la ciencia computacional; de hecho el hombre común lo ha aplicado desde mucho tiempo atrás para resolver problemas cotidianos en una forma meramente intuitiva.

En su libro "Introducción a la Metodología de la Investigación", Zorrilla Arena menciona que "hacer el análisis de una realidad o de un objeto cualquiera, presupone llegar a los elementos básicos que componen a esa realidad" y agrega "pensar en términos analíticos nos ha dado muchas ventajas a lo largo de la historia. A la filosofía analítica le debemos el impulso a la ciencia atómica, la lógica y la matemática contemporánea...". Pero llegar a esos elementos básicos no es siempre una tarea fácil, requiere de cierto cuidado. Así pues, el propósito de éste análisis, será el de identificar los requisitos y las características que debe tener nuestro sistema de software (que en lo sucesivo se le denominará ***sistema***

*adquisitor* [1] ), para empezar a proponer una solución aunque sea global, del problema presentado.

Dentro del ambiente computacional existen varios métodos de análisis que difieren en su enfoque para realizarlo y por tanto en el tipo de sistema de software en el que será aplicado. Aquí, se utilizarán las técnicas del *"análisis estructurado"* por dos razones: su simplicidad y su extensa aplicación en programación.

## 2.1 Análisis Estructurado

El análisis estructurado es una actividad de construcción de modelos. Mediante una notación, que es exclusiva de éste método de análisis, se crean diagramas que describen el flujo y contenido de la información, partiendo el sistema funcionalmente.

La esencia de éste análisis reside en la idea de que los datos sufren cambios o transformaciones cuando fluyen a través de los diferentes procesos que componen a un sistema de software basado en computadora. Los esquemas que muestran estos cambios se denominan *Diagramas de Flujo de Datos (DFD)* y utilizan una simbología fácil de comprender (fig. 7).

[1] Hablando en términos estrictos, el sistema adquisitor del SONIMP está constituido, además del enlace entre la CSU y su unidad KPU, por la herramienta sónica. Entonces, resulta presuntuoso llamarle *sistema adquisitor* al programa a desarrollar, que, aunque sí formará parte del sistema de adquisición, no lo será en su totalidad. El motivo por el cual le he llamado así, obedece mas bien, a lo práctico que resulta leer y escribir ese nombre, en vez de otro que englobe las tareas de adquisición, edición y conversión de información.

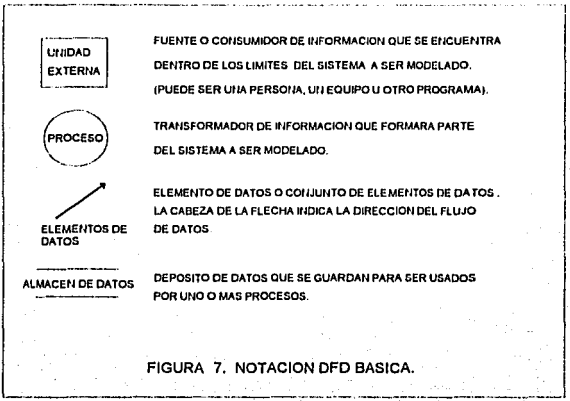



FIGURA 7. NOTACION DFD BASICA.

El rectángulo se usa para representar una **entidad externa** (p. ej.: hardware, una persona, otro programa), u otro sistema que produzca información a ser transformada por el software o que reciba información producida por el software. Un círculo representa un **proceso o transformación** que se aplica a los datos y los cambia a alguna otra forma. Todas la flechas de un DFD deben ir acompañadas con algún texto que indique el nombre del dato que fluye. Las líneas paralelas representan un **almacén de información;** información que será utilizada por el software.

Podemos utilizar estos DFD a cualquier nivel de "visualización" que tengamos del problema a modelar. En un DFD de nivel 0, también denominado *modelo fundamental del sistema o modelo de contexto*, nuestro diagrama estará constituido por una burbuja que será el sistema de software a elaborar, con una flecha a la entrada y otra a la salida, denotando los datos de entrada y el resultado

de la transformación realizada sobre ellos, respectivamente. Este proceso de diagramación debe continuarse, realizando niveles de refinación que muestren mayor flujo de información y un mayor detalle funcional.

Es importante señalar que el DFD no proporciona ninguna indicación explícita de la secuencia de procesamiento. El procedimiento o la secuencia queda implícita en el diagrama, dejando la representación procedimental explícita pospuesta hasta el diseño del software [PRE 93].

## 2.2 DFD general para el sistema adquisitor

En esencia el sistema adquisitor tendrá que cubrir 2 requisitos importantes para cumplir con los objetivos propuestos:

a) La sustitución de la KPU, mediante el almacenamiento de datos en memoria principal y secundaria de la computadora portátil, de los datos generados en la caverna y

b) La generación de archivos (que de ahora en adelante se les denominará ***archivos de interpretación***) derivados de los registros de cavernas para su futura interpretación y transformación en gráficas.

Para construir nuestro primer DFD o de nivel 0, podemos utilizar cualquiera de las dos tareas anteriores como salida. La más conveniente a usar es la (b), puesto que engloba a la tarea (a), o en otras palabras, para que la computadora genere los archivos de interpretación, es necesario tener guardado previamente, en memoria principal la información proveniente de la CSU. Dicho lo anterior, el modelo de contexto del sistema adquisitor queda como sigue:

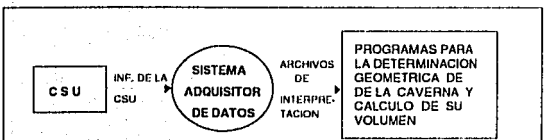



FIGURA 8. DFD DE NIVEL 0 PARA EL PROGRAMA ADQUISITOR

Para continuar con niveles más detallados, primero se debe entender al sistema en términos de los programas ó módulos que lo conformarán. Para determinarlos, debemos contestar una pregunta ¿Que se necesita para crear un sistema útil que cumpla con los requisitos especificados?; primero que nada, un programa de comunicación que se ajuste a las características de comunicación de la CSU y que "entienda" su formato para su correcta interpretación; segundo, un programa que nos permita grabar la información en una forma ordenada dentro de la LAPTOP, para ofrecer al usuario la posibilidad de consultarla en el momento en que lo desee; tercero, una interfaz USUARIO-LAPTOP amigable, con la cual, el operador realice las operaciones que estime convenientes; y finalmente un algoritmo que convierta la información del registro de la caverna (almacenada en disco duro de la computadora portátil), en información útil para la creación de gráficos.

Cada actividad anterior debe ser llevada a cabo por un módulo[6] que controlado por otro llamado principal, darán "forma" al sistema adquisitor.

En el siguiente nivel del DFD (fig. 9), podemos observar como pueden interactuar los módulos anteriores y cual es el flujo de datos entre ellos.

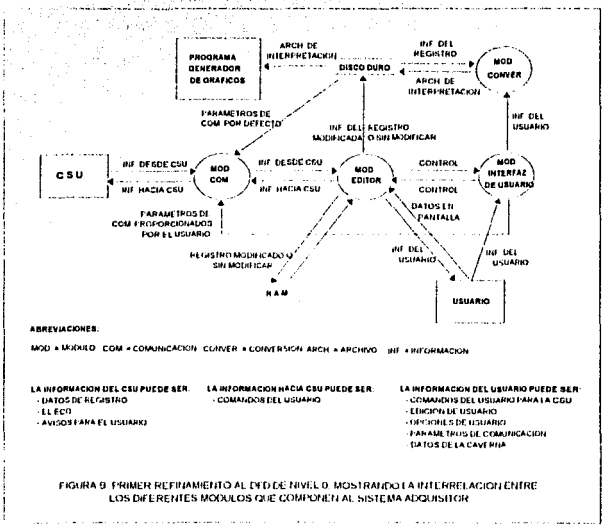



FIGURA 9 PRIMER REFINAMIENTO AL DFD DE NIVEL 0 MOSTRANDO LA INTERRELACION ENTRE LOS DIFERENTES MODULOS QUE COMPONEN AL SISTEMA ADQUISITOR

[6] Programa que realiza un proceso muy específico. En programación modular, un sistema de software se divide en varios módulos, a fin de que su análisis y diseño se facilite.

En la elaboración de los refinamientos de estos diagramas, debemos cuidar que tanto la entrada como la salida sean las mismas que las del diagrama predecesor. Así, en la fig. 9, hemos cuidado que la entrada y salida permanezcan igual a las del DFD de la fig. 8. En la fig. 9, la CSU está simbolizada con un rectángulo, denotándola como fuente de información. Las flechas que conectan la fuente con el módulo de comunicación ván en ambas direcciones; esto significa que la CSU aparte de enviar datos, necesita conocer las operaciones a realizar por parte del usuario, comportándose también como consumidor de datos. El usuario como fuente y consumidor de información únicamente puede comunicarse con la CSU a través del módulo editor, que a su vez hace uso del módulo de comunicación.

Cualquier modificación, sobre la información recibida de la CSU, podrá ser realizada utilizando el módulo editor. Todo cambio efectuado se grabará tanto en memoria principal como en disco duro.

El módulo de conversión obtiene sus insumos del disco duro y hacia el dirige su salida; es decir, una vez que el módulo de conversión haya seleccionado la información para el programa de interpretación, los datos así obtenidos serán almacenados en disco duro, bajo un nombre de archivo específico. Cada vez que se seleccione éste módulo, será necesario comunicarle algunos datos referentes al nombre de la caverna de donde se obtuvo el registro, el número de operación y la velocidad del fluido [*].

[*] Se refiere a la velocidad del haz sónico en el medio de propagación líquido existente en la caverna (agua salada o petróleo).

En la fig. 9, obsérvese que el módulo de interfaz mantiene relación con cada uno de los módulos que componen al sistema adquisitor; esto se debe a que sólo a través de este módulo el usuario conseguirá realizar las operaciones de comunicación, edición y conversión.

Para lograr un mejor análisis y un orden más adecuado, los refinamientos sucesivos al último DFD se realizarán en forma independiente, obedeciendo a la división funcional establecida; comenzando con el módulo de comunicación.

## 2.3 Módulo de Comunicación

Básicamente existen dos métodos de comunicación, determinados por la independencia o dependencia entre los relojes de las terminales receptoras y transmisoras. Si la información a ser transmitida está constituida por bloques completos de datos, es decir, múltiples bytes o caracteres, y además los relojes de las terminales receptoras y transmisoras se encuentran en sincronía, entonces estamos tratando con una comunicación tipo síncrona.

Si por el contrario, el dato a transmitir está conformado por una cadena de caracteres con intervalos de tiempo aleatorio entre cada caracter, la comunicación es asíncrona [HAL 89]. Tal es el caso de la comunicación existente entre la CSU y su unidad KPU, en donde claramente podemos observar que el usuario teclea caracteres a intervalos de tiempo indeterminados, con espacios de tiempo diferentes entre un "teclazo" y el siguiente, lo que significa que la línea de comunicación quedará en estado vacio durante intervalos de tiempo más o menos

grandes. Cada caracter transmitido en forma asíncrona está compuesto de 10 bits: 1 bit de comienzo, 7 bits de datos, 1 bit de paridad y 1 bit de paro.


FIGURA 10. FORMATO DE UN CARACTER EN LA COMUNICACION ENTRE LA CSU Y LA KPU. LA VELOCIDADOCIDAD DE TRANSMISION ES DE 1200 BAUDIOS.

Con 7 bits de datos se pueden generar $2^7$ caracteres correspondientes a los primeros 128 caracteres del código ASCII ( ver apéndice A ). Los tres bits restantes (fig. 10) se usan para determinar el comienzo y final de un caracter y para verificar si existió o no algún error en la transmisión (bit de paridad).

La KPU cuenta en su parte frontal y posterior con algunos selectores para establecer la velocidad de transmisión, el tipo de paridad y otros (para más información refiérase al apartado 1.2.1 "controles de la KPU") que resultan irrelevantes a la hora de trabajar con la computadora portátil (como ejemplo mencionaré el que se refiere a la intensidad de impresión dentro de la KPU). Aunque por defecto la velocidad de transmisión-recepción es de 1200 baudios la KPU puede trabajar sin problemas desde los 300 baudios hasta los 9600 (dependiendo del modelo de ésta). La capacidad de la KPU de trabajar con diferentes velocidades deberá ser cubierta por el módulo de comunicación. El

usuario será quien se encargue de establecer la velocidad y el puerto serie de comunicación, mediante el uso de las ventanas que facilitará el módulo de interfaz USUARIO-LAPTOP.

Realmente el problema a enfrentar en éste módulo no es el establecimiento de los parámetros de comunicación, ni la flexibilidad del programa para comunicarse a diferentes velocidades. Después de todo, los lenguajes de programación que he usado (Turbo Pascal, Turbo C, GW basic) tienen la capacidad de hacerlo mediante el uso de sus funciones. El verdadero problema se genera cuando la información que llega al puerto de comunicación tiene que ser almacenada en memoria secundaria ó auxiliar. Como es sabido, los accesos a discos ya sea flexibles o duros son mucho mas tardados comparados con los accesos a memoria RAM, lo cual origina que la información se pierda, en el momento en que la información es llevada desde el puerto serie al disco duro. La pérdida se puede remediar, programando adecuadamente el puerto, basándose en el uso de interrupciones. Por el momento, dejaremos los detalles para el capítulo 3 y veremos al proceso de interrupción como una caja negra, cuya función es la de llamar la atención del microprocesador cuando un caracter está listo para "leerse" ó "escribirse" desde y en el puerto serie.

Con la información anterior el DFD que corresponde al módulo de comunicación tiene el siguiente aspecto:

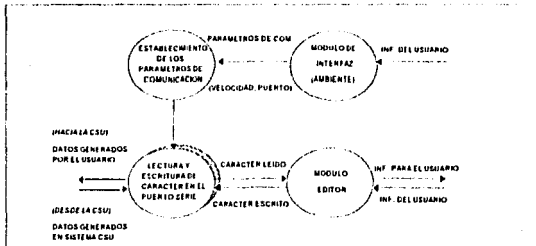



FIGURA 11. DFD CORRESPODIENTE AL PROGRAMA O MODULO DE INTERACTUANDO CON LOS MODULOS DE INTERFAZ Y

**Nota:** El sombreado en la burbuja de lectura y escritura de caracter en el puerto serie no es propia de la simbología de los DFD, su único propósito es la de señalar la actividad que se realiza cuando se genera una interrupción para atención del puerto.

## 2.4 Módulo Editor

Las tareas a cubrir mediante éste módulo están orientadas a dar solución al "problema de organización", al cual se hizo referencia en 1.4. Este problema es causado por la obtención de algún valor incongruente con el resto de los datos en el momento de hacer algún registro en una caverna subterránea. Por dato incongruente debemos entender a aquella cantidad con valor muy por encima o muy por debajo de la demás cantidades. Para verificar que el dato realmente está erróneo, las personas que operan el equipo de superficie, deben repetir las

operación de disparo del haz sónico a esa profundidad y con el mismo azimuth, variando la ganancia de amplificación de señal sónica a disparar. Ejemplificando lo anterior, veamos el siguiente listado:

SCAN DEPTH: 910 026

| ETIM S | AZIV DEG | VRAD M | PVRD M | PVR M | ELEV DEG |
|---|---|---|---|---|---|
| 5 0610 | 14 00 | 12 03 | 920 452 | 6 085 | 30 00 |
| AZIMUTH DISAGREES WITH SELECTED VALUE | | | | | |
| 10 0110 | 29 00 | 10 86 | 919 386 | 5 472 | 30 00 |
| 99 1113 | 300 00 | 33 09 | 938 687 | 16 60 | 30 00 |
| 104 0610 | 315 00 | 38 09 | 943 028 | 19 10 | 30 00 |
| 109 0110 | 330 00 | 35 65 | 940 922 | 17 90 | 30 00 |
| 113 9620 | 345 00 | 75 75 | 975 671 | 37 96 | 30 00 |

LOG STOPPED
> CH AZIM 345
0 DEG -> 345 DEG ? Y
AZIMUTH SELECTED = 345 DEG
DEPTH 910 10 M 12-JUN-1991 14 50

| | | | | | |
|---|---|---|---|---|---|
| 39 4220 | 345 0 | 28 93 | 926 436 | 9 539 | 30 00 |
| 42 9220 | 345 0 | 28 87 | 926 385 | 9 507 | 30 00 |
| 46 4230 | 345 0 | 28 87 | 926 385 | 9 507 | 30 00 |

FIGURA 12. PARTE DE UN SCROLL, MOSTRANDO LA EDICION MANUAL DE SU INFORMACION

Dentro del listado, pongamos atención al valor de VRAD = 75.75, correspondiente a un azimuth AZIV = 345.0 y hagamos una comparación con el valor inmediato anterior de VRAD. La diferencia entre ambos valores es muy

grande comparada a la que existen entre el resto de los datos, en donde la variación es de 3 a 5 * unidades solamente.

El operador que efectúo este registro, se dio cuenta de éste hecho y decidió repetir la operación con el mismo azimuth, en donde el valor de VRAD se disparó en su valor. Los datos nuevos así obtenidos se muestran en la parte inferior del listado.

Una vez obtenido el nuevo valor de VRAD, el operador tiene que sustituir el dato erróneo (en caso de que constituya verdaderamente un error) por el dato correcto, relacionando con una flecha ambos valores. Esta edición manual de información queda en desuso en el momento en que la LAPTOP sustituye a la KPU. Esta mejora en la edición de información, se debe a que toda la información proveniente de la CSU se guardará en memoria RAM de la LAPTOP, lo que trae como consecuencia poder desplegarla ó modificarla, si así es requerido.

Por tanto, nuestro módulo editor tendrá como objetivos, cubrir las siguientes tareas:

- Organizar en RAM, la información proveniente de la CSU, así como la que introduzca el usuario.

- Desplegar la información en la pantalla y proveer al usuario, facilidades para poder consultarla. Estas facilidades implicarán el uso de las teclas de desplazamiento, las teclas de cambio de página y las teclas de posicionamiento al inicio y final de línea.

- Grabar la información en disco duro.

* De ninguna manera, esta variación debe ser tomada como un patrón. La diferencia entre un valor y otro, está en función de como actuaron las fases de lixiviación al momento de crear la caverna.

- Proveer al usuario la capacidad de modificar la información, así como de salvar los cambios hechas sobre ésta.

- Insertar y borrar líneas en el SCROLL.

La última actividad tiene relación con una omisión que se hizo en el programa de control de la herramienta sonar, lo cual origina que no se imprima el dato correspondiente a 0 grados ó 360 grados. ( Obsérvese ambos listados: el que se presenta en la fig. 4 del capítulo 1 y el que se presenta en este apartado). Como el dato es necesario para los programas de interpretación, el operador debe hacer una nueva medición en ese azimuth e insertar el valor obtenido en el espacio correspondiente, dentro del listado, en 0 grados.

Por lo anterior, podemos darnos cuenta de la importancia de éste módulo, que expresado en forma de DFD, tendrá el siguiente aspecto:

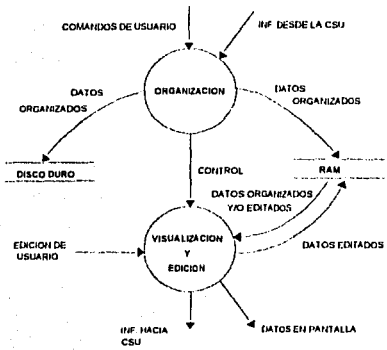



FIGURA 13. DFD CORRESPONDIENTE AL MODULO EDITOR.

La palabra "control" que aparece en un elemento de dato (flecha) de la fig. 13, indica que entre los procesos conectados por ese elemento de dato, no existirá flujo de datos, es decir, no habrá paso de parámetros de un proceso a otro; en cambio, si habrá un paso de control. Este paso de control, sucede cuando un proceso termina, dando lugar a que otro proceso se efectúe por la computadora.

Para finalizar el análisis de éste módulo, mencionaré que el editor a programar, corresponde al tipo de editor denominado de *pantalla*, los cuales permiten la visualización de varias líneas a la vez y funcionan en relación al movimiento del cursor.

## 2.5 Módulo de Conversión

Todas las actividades contempladas dentro de éste módulo constituirán una mejora al proceso descrito en 1.3. La idea surgió, de las ventajas que ofrece el hecho de tener toda la información almacenada, tanto en memoria RAM como en disco duro de la computadora portátil LAPTOP. Mediante un programa, que procesará la información referente al SCROLL y previamente grabada en disco duro, se generará un archivo que deberá tener un formato especial, compatible con el programa de interpretación, eliminándose así, la tediosa tarea de capturar en forma manual, los datos de interés dentro del SCROLL.

El archivo de interpretación al que se ha hecho referencia en varios párrafos de éste trabajo, debe tener un formato específico, conformado por un conjunto de datos, necesarios para crear los gráficos en 2 y 3 dimensiones. Los datos y la forma en que deberán estar organizados, es la siguiente:

FECHA
NOMBRE DE LA CAVERNA
No. DE OPERACION
PROFUNDIDAD DE LA HERRAMIENTA
VELOCIDAD DEL FLUIDO
AZIMUTH
RADIO
PROFUNDIDAD
AZIMUTH
RADIO
PROFUNDIDAD

-999
AZIMUTH
RADIO
PROFUNDIDAD
.
.
.
-999

Los tres primeros datos de la lista anterior (fecha, nombre de la caverna, número de operación) así como la velocidad del fluido, no se obtendrán de los datos almacenados en disco duro, sino que deberán ser accesados por los usuarios, a través del módulo de interfaz. La repetición de la información correspondiente al azimuth, radio y profundidad, obedece a que a cada giro (azimuth) de la herramienta sónica, se obtienen diferentes radios. El número -999, es un delimitador, que indica la finalización del barrido en cierta profundidad o estación.

En general, las tareas a realizar dentro de éste módulo serán:

1) abrir dos archivos: uno, en donde se encuentra toda la información concerniente al registro de la caverna en cuestión (archivo de registro), y el otro, en donde se almacenará el archivo de interpretación,

2) grabar en el archivo de interpretación aquellos datos introducidos por el usuario y que corresponden a los datos de la caverna (fecha, nombre de la caverna, número de operación y velocidad del fluido),

3) hacer una selección en el archivo de registro de los datos útiles para la creación de los gráficos (azimuth, radio y profundidad) y grabarlos en el archivo de interpretación y

4) cerrar los archivos abiertos y regresar al control del sistema (ver fig. 14). Al llegar a éste punto, el alcance de la presente tesis habrá sido abarcado en su

totalidad. Ya dentro del sistema SONIMP, lo que restará, será correr el programa de interpretación con los archivos de interpretación generados para finalmente obtener las gráficas.

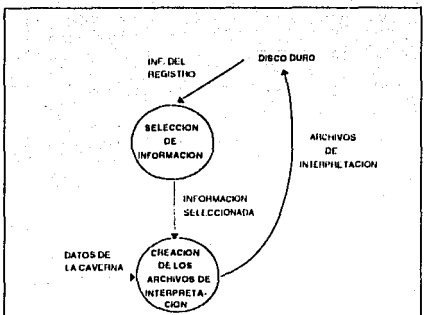



FIGURA 14. DFD CORRESPONDIENTE AL MODULO DE CONVERSION.

## 2.6 Módulo de Interfaz de Usuario

La interfaz de usuario es el mecanismo a través del cual se establece un diálogo entre el programa y la persona que lo utiliza. Para su diseño intervienen muchos factores que tienen que ver con el ambiente en donde trabajará el sistema, la gente que lo manejará y con aspectos de la tecnología. Lo que se busca es que el sistema en general sea fácil de aprender, simple de utilizar, directo y no muy estricto.

Cuando se considera una **interfaz hombre máquina (IHM)** se tiene que tomar en cuenta los sistemas visual, táctil y auditivo. Esto permite al usuario de un sistema basado en computadora percibir información, almacenarla en su memoria y procesarla. Los aspectos antes mencionados rebasan el ámbito de éste análisis, así que lo más viable será diseñar una interfaz que trabaje en forma similar a interfaces de programas conocidos, con una ventaja que surge inmediatamente para las personas que ya hayan manejado estos programas: estarán previamente familiarizados con algunas formas de interactuar con el programa que se diseñará.

### 2.6.1 Estilos de interacción entre hombre y máquina

A través del tiempo han surgido una gran variedad de estilos de IHM, los cuales han estado íntimamente ligadas a la evolución de las computadoras. Las opciones para el estilo de interacción han aumentado a medida que el hardware se ha ido haciendo más sofisticado.

En los inicios de las computadoras (antes de las pantallas gráficas, el ratón, etc.) la única forma real de IHM era textual, compuesta de una serie de preguntas y órdenes, las cuales, aunque eran concisas, también eran muy estrictas y bastante difíciles de aprender [PRE 93]. De esta forma, el usuario podía comunicarse con el sistema, especificando una orden como la siguiente:

```
$ cpas arch

output file for compliled listing? ( <cr> for none ) _
```

Posteriormente surgió una variación a esa interfaz, que consistía en la presentación en pantalla de un menú simple. El usuario tenía que escoger de entre varios procesos el de su conveniencia para teclear su código u opción.

Elija la opción deseada:

1) Altas a proveedores.

2) Bajas a proveedores.

3) Cambios.

4) Generar lista de proveedores.

5) Otras.

6) Salir.

Seleccionar opción ? _

Frecuentemente en este tipo de menús, para llegar a un proceso, es necesario pasar por menús previos, cambiando cada vez de pantalla, lo que resulta tedioso y molesto para el usuario.

Así como el hardware se ha hecho más eficiente, los estudios sobre factores humanos y su impacto en el diseño de la interfaz han también mejorado, apareciendo modernas interfaces que incluyen ventanas con opción de señalar y elegir. Esta interfaz de "tercera generación" hace un uso extenso de los menús desplegables, los cuales permiten al usuario realizar tareas de control y diálogo en forma sencilla.

### 2.6.2 Menús Desplegables y de Acceso de Información

El módulo de interfaz a diseñar, se basará en el manejo de menús despegables y de acceso de información, los cuales facilitarán las tareas a realizar dentro del sistema adquisitor, permitiéndole al usuario tener un control rápido sobre sus actividades.

A continuación se dará una breve explicación de las características y de como funcionan estos dos tipos de menús, para que al final se dibuje el DFD correspondiente.

Cuando un menú de selección es activado, éste se superpone sobre lo que había en la pantalla en ese momento, presentando al usuario una lista de opciones. Después de hacer la selección, la pantalla se restaura a su estado previo. El usuario puede elegir una opción desde un menú de este tipo de una de estas dos formas: 1) presionado una tecla clave (*hot key* en literatura en inglés ), la cual es una letra asociada a las distintas opciones del menú ó 2) usando las teclas de desplazamiento para mover la sobre-iluminación a la opción que elija, presionando a continuación la tecla ENTER.

La apariencia que muestran estos menús en pantalla es:

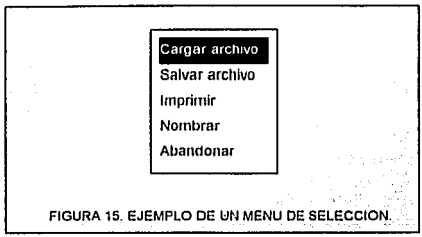



FIGURA 15. EJEMPLO DE UN MENU DE SELECCION.

En los menús de acceso de información, se le presenta al usuario una serie de preguntas y líneas en blanco. El usuario entonces debe llenar una a una éstas líneas, hasta que las preguntas sean totalmente agotadas. Un ejemplo de este tipo de menús es el siguiente:

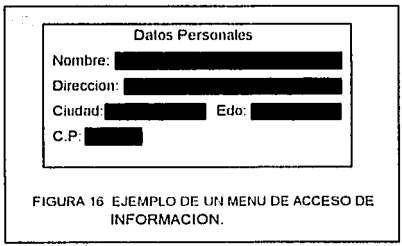



FIGURA 16 EJEMPLO DE UN MENU DE ACCESO DE INFORMACION.

Todas las entradas del usuario deben ser debidamente validadas para que el sistema procese datos correctos.

Para que el sistema adquisitor opere bajo las condiciones que hasta ahora se han planteado, deberá contener al menos cuatro menús que nos proporcionen un completo control sobre la información que llega desde la CSU. Por tanto, en primer término necesitamos contar con algunas opciones que permitan nombrar, guardar e imprimir el archivo en donde se almacenará la información de un registro. Para lograrlo un menú que puede llamarse ***ARCHIVO*** y que contenga esas opciones deberá ser desplegado. A éste menú podemos agregarle dos opciones más: una que nos permita cambiar al modo de edición y otra que nos permita salir del sistema. Por otro lado, si se requiere variar los parámetros de comunicación tales como número de puerto serie y velocidad de transmisión-recepción, el módulo de interfaz de usuario deberá proporcionar los medios suficientes para llevar a cabo tales actividades. Así, un segundo módulo al cual se le denominará ***CONFIGURACION*** deberá incluirse. El tercer menú nombrado ***CONVERSION*** deberá encargarse de solicitar los datos referentes a la caverna objeto de estudio, así como también "llamar" al módulo de conversión de formato. Para finalizar debemos incluir un menú principal que se encargará de indicarle al módulo de interfaz el menú a desplegar (uno de los tres anteriores) cada vez que el usuario seleccione su opción. La distribución en pantalla de los tres menús se muestra en la fig. 17.

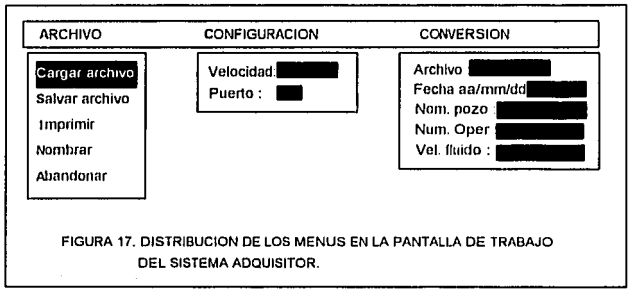



FIGURA 17. DISTRIBUCION DE LOS MENUS EN LA PANTALLA DE TRABAJO DEL SISTEMA ADQUISITOR.

En la fig. 17, observemos que algunos caracteres aparecen con mayor intensidad que otros. Tales caracteres son las letras claves asociadas a cada opción y le permiten al usuario llevar a cabo alguna actividad de una forma más directa y rápida; es decir, sin necesidad de llevar la sobre-ilumunación hasta la opción deseada para después teclear ENTER. También podemos ver la sobre-iluminación que nos indica nuestra posición en el menú. La sobre-iluminación está en video inverso (fondo negro, letras blancas).

Siguiendo con la misma metodología que se ha aplicado hasta el momento, el paso siguiente será diseñar el DFD que nos ilustre de una forma general las etapas que tienen que ejecutarse dentro del módulo de interfaz.

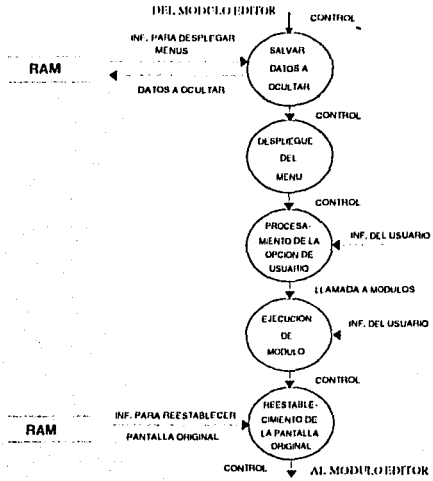



FIGURA 18. DFD PARA LA INTERFAZ DE USUARIO

Una característica común de los menús de acceso de información y de los de selección, es que siempre se superponen sobre la pantalla actual de trabajo, ocultando la información que queda justo debajo de ellos. Por tanto, una de las primeras tareas a realizar para la creación de éstos menús, será la de salvar la información que quedará oculta, para que una vez que dejemos hacer uso de los menús, esa información sea reestablecida en pantalla.

Salvada la información, lo que sigue es escribir el texto dentro del menú, para posteriormente procesar la opción del usuario y ejecutar uno de los tres módulos restantes del sistema adquisitor. Una vez que el usuario dé por terminado

sus operaciones con el módulo elegido, seguramente deseará continuar dentro del sistema adquisitor para realizar otra operación, por lo que la información que fué previamente ocultada por un menú deberá ser reestablecida a su condición original.

### 2.6.3 Características adicionales del HIM para facilitar el trabajo del usuario

Las características que a continuación se detallarán, aunque no son novedosas, harán del programa de interfaz más llamativo, y proveerá al usuario una forma más cómoda de interacción con la LAPTOP.

#### 2.6.3.1 Facilidades para nombrar los archivos de trabajo

En el estudio de la geometría y cálculo del volumen de una caverna subterránea se llevan a cabo varios registros, pues es necesario ir conociendo como van actuando los procesos de lixiviación dentro de ella. El tiempo que separa la realización de un registro y otro para una misma caverna, puede ser de varios días. Para que el usuario sepa a que información pertenece tal registro, la computadora debe proporcionarle alguna vía por la cual nombre la información, que, en el momento de realizar el registro, se obtenga. La única restricción para nombrar la información del registro, será la de llevar siempre la extensión .SCN[4] . Así, se conseguirá también hacer una distinción entre los archivos de registro con extensión .SCN y los archivos de interpretación, que se generarán de los primeros. Ambos tendrán el mismo "nombre base" (por tener información en común), pero

[4] de la palabra en inglés scan que significa explorar.

diferente extensión. La extensión del archivo de interpretación será .GFX, que es la extensión que acepta el programa de interpretación.

A veces resulta incómodo estar recordando las extensiones de los archivos; esto sucede con más frecuencia cuando no se está familiarizado con un programa. Por tal motivo se diseñará una interfaz que añada la extensión del archivo automáticamente, ahorrándole al usuario la tarea de escribir la extensión correctamente.

### 2.6.3.2 Manejo de colores

El manejo de colores dentro de la interfaz del programa adquisitor es importante pues es necesario resaltar algunos mensajes (como los de error o aviso) o simplemente limitar ciertas áreas de trabajo. Así por ejemplo, para un mejor orden dentro de la pantalla se manejarán 6 áreas diferentes (alguna de ellas no pueden ser consideradas de trabajo), caracterizadas cada una por un color de fondo diferente. Aquí el contraste que tenga el color del caracter y el color de fondo será un factor a tomar muy en cuenta.

### 2.6.3.3 Manejo de errores

El despliegue en pantalla de mensajes de error constituye un factor muy importante en las IHM. Mediante estos el usuario sabe que la forma en que se procedió fué errónea ó que necesita cumplir cierto requisito para llevar a cabo algún proceso.

En general, todos los mensajes de error o de aviso producidos por un sistema deberían tener las siguientes características:

- El mensaje debe describir el problema en un lenguaje que comprenda el usuario.
- El mensaje debe proporcionar una información constructiva para poder solventar el problema.
- El mensaje debe ir acompañado de una clave visual o audible. Esto es, puede generarse un sonido acompañando la visualización de un mensaje, o bien éste puede parpadear momentáneamente y ser visualizado en un color fácilmente reconocible como "color de error".
- El mensaje no debe aportar un juicio sobre lo ocurrido. Esto es, no debe culparse al usuario.

El manejador de errores a diseñar utilizará una ventana en la parte inferior de la pantalla la cual cambiará su valor de fondo de rojo a gris dependiendo si el texto a desplegar es un error o simplemente un aviso al usuario.

El mecanismo de funcionamiento es sencillo. Basta guardar en la memoria principal de la computadora una lista de errores y avisos para que después mediante un código asociado a ellos se obtengan de esa lista y se despliegen en la pantalla mencionada.

# CAPITULO 3

## DISEÑO E IMPLEMENTACION DEL SISTEMA

Èste es el último de los tres capítulos de que consiste ésta tesis. En él se diseñará e implementará el sistema adquisitor en base a la solución formulada en el capítulo 2. Para abordarlo, comenzaremos exponiendo a manera de introducción algunos conceptos sobre programación estructurada y modular, a fin de enmarcar el desarrollo de los códigos fuentes de los módulos de comunicación, edición, de interfaz de usuario y conversión.

### 3.1 Tópicos sobre diseño e implantación de sistemas

El diseño puede definirse como: "*...el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso ó sistema con los suficientes detalles para permitir su realización física*" [ PRE 93]. Analizando las líneas anteriores caeremos en la cuenta que el diseño por sí solo no da solución final al problema, o en otras palabras, sólo permite la realización física, mas no la crea. La solución se dá en una etapa posterior, denominada implementación que tiene como objetivos convertir el diseño en códigos fuentes dispuestos a correrse en una computadora. Aunque el diseño y la implementación son etapas diferentes en el desarrollo de software, en éste capítulo se diseñará e implementará al mismo tiempo, ésto con el deseo de no hacer tan tediosa la exposición de las principales ideas que dieron solución final a los problemas intrínsecos de la sustitución de la KPU por una LAPTOP. La manera de conducir nuestro diseño estará basada en la

presentación de los principales pseudocódigos y códigos en Turbo C que fueron utilizados en cada uno de los módulos que componen el sistema adquisitor. En la construcción tanto de los pseudocódigos como de los códigos se utilizarán algunos conceptos sobre programación modular y estructurada; de éstas dos últimas hablaré a continuación, como preámbulo a lo que será propiamente el diseño e implementación.

### 3.1.1 Metodologías de programación a aplicar.

Resulta difícil resolver un problema complejo si no se aplican técnicas tendientes a facilitar, de una manera ordenada, su solución. La realización de un programa sin seguir un método de programación, aunque funcione, dará como resultado un código que no será mas que un conjunto de instrucciones. La consecuencia inmediata de lo expuesto puede resumirse en la siguiente lista de problemas:

- Los programas suelen carecer de flexibilidad necesaria para adaptarlos a diferentes configuraciones de máquinas.
- Los programadores gastan más tiempo corrigiendo sus errores.
- Si el programa es lo considerablemente grande que implique la programación de varias personas, la comunicación entre programadores se dificulta.
- Las modificaciones en las aplicaciones son muy difíciles de realizar, implican mucho tiempo y elevado costo de mantenimiento. Ello conduce a realizar correcciones que complican cada vez más el diseño inicial o bien a que el programa caiga en desuso.

- Deficiencia en la documentación:
  - Descripciones incompletas o escasas.
  - Falta de diagramas que expliquen el funcionamiento del programa.

Así pues, a la hora de diseñar un programa, se deberá hacer de tal forma que éste reúna ciertas características fundamentales:

- FIABILIDAD: Producir los resultados requeridos.
- LEGIBILIDAD: Debe ser entendido por cualquier programador a fin de que sea fácilmente modificable.
- DEPURABILIDAD: Debe ser fácil la localización y corrección de errores.

Para conseguir óptimos resultados que cumplan con las características mencionadas se aplicarán dos de las metodologías más usadas:

- Programación modular.
- Programación estructurada.

Ambas técnicas se complementan entre sí, pues el análisis de un problema puede utilizar criterios de programación modular para dividirlo en partes independientes y utilizar métodos estructurados en la programación de cada módulo [JOY 87].

### 3.1.1.1 Programación modular

"Divide y vencerás" dice un viejo proverbio popular, el cual no encontró mejor acogida que en la programación modular.

El soporte más elemental para la programación modular se dió con la invención de la "subrutina" a principios de los 50's. Una subrutina o procedimiento es un subprograma independiente que ejecuta una tarea particular, y que es llamado por otro programa que generalmente es el denominado principal o base. Cada vez que la rutina termina su trabajo, regresa el control del programa al módulo que la llamó. Obsérvese en la definición anterior que hice referencia primero a la subrutina y no al módulo que es de donde realmente proviene el nombre de *programación modular*. Este hecho se debe a que un módulo puede consistir en:

- un programa,
- una función,
- una subrutina o procedimiento.

### 3.1.1.2 Programación estructurada

La programación estructurada surgió como una necesidad de desarrollar programas con un estilo consistente y disciplinado a finales de los años 60. En [JOY 87] se define a la programación estructurada como: " *técnica de construcción de programas que utilizan al máximo los recursos del lenguaje, limita el conjunto de estructuras aplicables a leer y presenta una serie de reglas*

*que coordinan adecuadamente el desarrollo de las diferentes fases de la programación* " y menciona que es difícil encontrar una definición exacta, pues ninguna de ellas es aceptable en todos los niveles de programación. Lo que se da como un hecho, es que en donde quiera que se esté aplicando la programación estructurada se encontrarán conceptos como el de *diseño descendente (top-down)*, el de *recursos abstractos* o el de *estructuras básicas de control*.

### 3.1.1.3 Metodología descendente "top-down"

La metodología descendente establece varios niveles jerárquicos de menor a mayor complejidad para darle solución a un problema. En esencia consiste en establecer relaciones entre un nivel jerárquico y el nivel inmediato inferior, mediante el intercambio de información.

En la aplicación de éste método debemos basarnos en dos características muy importantes: representación en forma de árbol y descomposición funcional.

En la descomposición funcional debemos partir de un primer nivel que resuelve totalmente el problema (en forma muy general); los siguientes niveles son refinaciones sucesivas del primero *(stepwise)*. El proceso se termina cuando ya no es posible refinar mas las etapas anteriores.

### 3.1.1.4 Recursos Abstractos

En el diccionario Larousse se encuentra la siguiente definición para abstracción: " *conocimiento de una cosa prescindiendo de las demás que están con ella* ". Trasladando ésta definición a lo que es la programación, podemos decir que la abstracción es aquello que nos permite darle solución a un problema mediante la elaboración y aplicación de un programa, independientemente de los detalles irrelevantes de bajo nivel. Entonces diseñar un problema en términos abstractos consiste en no tener en cuenta el tipo de máquina en donde se implementará el programa, así como el lenguaje de programación que se va a utilizar.

En el proceso de refinación de la metodología descendente *(top-down)*, a cada nivel de refinación corresponde un nivel de abstracción diferente. " *Durante el análisis de los requisitos del software, se establece la solución del software en términos de lo que es familiar al entorno del problema. Conforme nos movemos desde lo preliminar al diseño detallado, se reduce el nivel de abstracción* " [PRE 93]. Esta idea encaja perfectamente en la forma a como se ha trabajado, pues desde que se expusieron los antecedentes nos hemos abocado a establecer diferentes soluciones, refinándolas cada vez más, yendo de lo preliminar al diseño detallado.

### 3.1.1.5 Estructuras básicas de control

Son aquellas estructuras que le indican al microprocesador la acción que deberá realizar sobre los datos en cuestión, mismos que están organizados de maneras diversas y que forman las estructuras de datos.

Dentro de la programación estructurada, estas estructuras pueden ser de tres tipos:

- Secuencial: Ejecuta las acciones sucesivamente, una tras otra, sin posibilidad de omitir alguna instrucción.

- Alternativas o Condicionales: La acción a realizar está en función de una o varias condiciones.

- Repetitivas o Iterativas: Son aquellas en las que las acciones se ejecutan un número determinado de veces y dependen de un valor predefinido o el cumplimiento de una determinada condición.

## 3.2 Módulo de comunicación

El enlace entre la CSU y la LAPTOP se realiza a través del módulo de comunicación, sus funciones no se limitan únicamente a recibir datos, sino también a transferir la información del usuario a la CSU, a fin de que ésta última lleve a cabo los servicios que el usuario necesita.

Para la recepción y transmisión de datos debemos escoger de entre dos métodos disponibles. El primero recibe el nombre de *polling*, que consiste en ciclos de examinación del puerto serie tanto para transmitir como para recibir datos.

Usando las técnicas de polling se tienen dos desventajas. El primer problema con éste método es el poder de procesamiento derrochado por una computadora que continuamente verifica el estado del puerto serie. Si la

computadora está dedicada solamente al manejo de tareas de comunicación, entonces ésto no es un problema. Sin embargo, las computadoras realizan varias tareas en un lapso de tiempo corto. En éstos casos se desea usar un método que evite polling innecesario.

El segundo problema con las técnicas de polling es que un nuevo caracter puede perderse mientras el primero está siendo procesado. Esto depende de dos factores: la velocidad a la cual los caracteres están siendo recibidos y a la velocidad en que están siendo procesados. Tipicamente, los caracteres de entrada son desplegados en una pantalla, salvados en un archivo e impresos. Algunas veces éstas necesitan ser interpretadas. Estos procesos requieren tiempo " *cuando una computadora típica usa técnicas de polling los caracteres que entran pueden fácilmente perderse si la velocidad excede las 1200 baudios aún si los accesos al disco no son requeridos* " [GOF 86].

La segunda opción es mediante interrupciones hardware. La interrupción hardware es generada por un chip del puerto serie denominado *UART* (Universal Asynchronous Receiver and Transmitter). Un UART programado para enviar interrupciones envía una señal cuando un evento importante ocurre. Por eso la computadora que recibe sabe cuando algo ha sucedido y no tiene que derrochar tiempo en checar.

Aunque el uso de interrupciones puede ahorrarnos mucho tiempo de procesamiento, no está exento de escollos. Es importante tomar en cuenta que una interrupción puede ser recibida a cualquier hora durante el ciclo de procesamiento. La computadora o cualquier programa que corra en ésta, debe estar diseñada para

que cualquier tarea se suspenda cuando una interrupción ocurra y reanudar después de que el evento que causó la interrupción haya sido atendido.

Dada las condiciones de trabajo de la CSU con su KPU, el tipo de interrupción a implementar será hardware.

El manejador de la interrupción hardware, como se dió a conocer en el capítulo anterior, hace uso de un espacio de memoria en donde se almacenan todos aquellos caracteres que arrivan al puerto serie de la LAPTOP. Ese espacio de memoria o cola circular se crea mediante la declaración de un arreglo. Para detallar el funcionamiento de la cola, veamos los siguientes códigos, los cuales nos ilustran la forma en que se introducen y se sacan los caracteres que llegan al puerto.

```
int metercar( char c)
{
  int bien;
  cola[end] = c;  /* almacenando caracteres en la cola circular */
  if (count < size)
  {
    end++;
    count++;
    bien = 1;
  }
```

```
    else
    {
       count = size;
       end++;
       start++;
       bien = 0;  /* overflow */
    }
    if(end >= size) end = 0;
    if (start >= size) start = 0;
    return bien;
}

unsigned char obtcar()
{
    unsigned char c;
    c = 0;
    if ( count <= 0) { count = 0; return 0; }
    c = cola[start];  /* sacando caracteres de la cola */
    start++;
    count--;
    if(start >= size) start = 0;
    return c;
}
```

Para que ambas funciones efectúen sus tareas, las variables *start*, *count* y *end*, deben ser globales con un valor inicial de 0. El valor de *size* también es global y debe inicializarse con alguna cantidad entera a criterio del programador.

Cada vez que un caracter arriva al puerto se graba en cola[end], incrementándose posteriormente el valor de end. Estas operaciones se pueden repetir tantas veces como lo permita size, que es el límite de caracteres que puede almacenar la cola circular.

Como podemos observar, la función *obtcar* sólo funciona en los casos de que existan caracteres que sacar de la cola. Ambas funciones contemplan los casos cuando la capacidad del arreglo es rebasado, para lo cual es necesario reestablecer a sus valores iniciales tanto a end como a start.

Otro punto de mucho interés que es necesario mencionar es que obtcar, es la misma función a la que se hace refernecia en el pseudocódigo 2 de la siguiente sección. De hecho, ésta es la relación entre los módulos de comunicación y editor, mostrada en la fig. 9.

### 3.3 Módulo editor

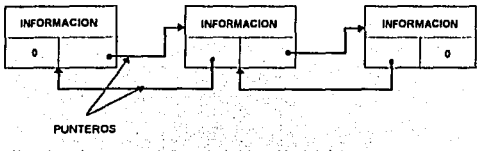
El módulo editor constituye la herramienta a través de la cual el usuario puede efectuar cambios a la información enviada por la CSU y almacenada en la LAPTOP. Los cambios, que en sí conforman la edición, van desde la modificación de un simple caracter hasta el borrado e inserción de renglones completos de información desplegada en pantalla.

Para el almacenamiento correcto de la información en LAPTOP se hace imprescindible la estructuración de los caracteres componentes de la información.

Esta estructuración la facilitan los lenguajes de programación de maneras muy propias. Una estructura de datos es una colección de elementos que tienen relación entre sí. La forma en que se relacionan unos elementos con otros y los valores de los mismos determinan el tipo de estructura de datos. Para nuestros propósitos, la estructura a usar deberá almacenar caracteres, pues si recordamos, toda la información que envía la CSU a su unidad KPU está conformada por caracteres ASCII. Por otro lado, ésta misma estructura deberá proporcionarnos un medio por el cual podamos tener acceso a cualquier parte de la información almacenada en RAM. El acceso a la información se consigue organizando los datos en líneas de texto, cada una con "n" caracteres, tantos como nos lo permita el ancho de la pantalla en donde se visualizarán. Las líneas deben estar enlazadas entre sí, para que se pueda acceder a cualquier parte de la información

A la estructura que nos proporciona tales facilidades se le denomina *Lista Doblemente Enlazada*. Una lista de este tipo es aquella que tiene tanto un primer como un último nodo, existiendo enlaces entre ellos. Cada nodo está relacionado con el anterior y el siguiente ( excepto el primero -sólo con el segundo y el último -sólo con el anterior ).

La ventaja de las listas doblemente enlazadas es que se pueden recorrer en ambas direcciones. La exploración en ambos sentidos puede realizarse con dos punteros por nodo. De éste modo se puede utilizar un puntero para localizar el registro precedente y otro para el siguiente (ver fig 19).




*Nota : Los números ceros indican el principio y el fin de la lista.*

FIGURA 19. UNA LISTA DOBLEMENTE

La ventaja de transitar en la lista en dos direcciones permitirá al usuario recorrer toda la información en RAM, de arriba hacia abajo y viceversa, únicamente presionando las teclas de desplazamiento vertical de la LAPTOP (flecha arriba "↑", flecha abajo "↓") ó las teclas de cambio de página (Pg Up, Pg Dw).

Cada lenguaje de programación tiene su propio estilo para implementar estructuras. Por ejemplo Turbo C, que es el lenguaje a utilizar, construye las listas doblemente ligadas como a continuación se describe.

### 3.3.1 Estructuras Autorreferenciadoras y su utilización en las listas doblemente ligadas.

Una estructura es un conjunto de elementos relacionados entre sí. Un *arreglo*, una *unión* constituyen estructuras de datos. Otra estructura de datos es la denominada *struct* (en lenguaje C), que a diferencia de los arreglos cuyos

elementos son todos del mismo tipo, puede contener elementos enteros, de punto flotante, caracteres, arreglos, punteros e incluso otra estructura.

La declaración de las estructuras struct debe ser definida en términos de sus miembros individuales. En general, la composición de una estructura struct puede ser definida como:

```
struct marca {
        miembro 1;
        miembro 2;
        ...
        miembro n;
};
```

En esta declaración struct es la palabra clave requerida, marca un nombre que identifica estructuras de éste tipo y miembro 1, miembro 2, ..., miembro n la declaración de los miembros individuales [BYR 91]. Para obtener una estructura autorreferenciadora, basta poner en algún miembro individual, algún puntero que apunte a otra estructura del mismo tipo, es decir, una estructura con los mismos miembros.

```
typedef struct lista {
        char texto[83];
        struct lista *siguiente;
        struct lista *anterior;
} LISTA;
LISTA *primero, *ultimo, *info, *punaux;
```

Para empezar a construir el editor, en la declaración anterior hemos formado una estructura llamada *lista* que contiene tres miembros: el primero es una estructura de tipo arreglo el cual almacenará la información de una línea de texto. El arreglo tendrá 83 celdas: 80 celdas para los caracteres a desplegar en pantalla, dos celdas para CR (Carry Return) y LF (Line Feed) y otra más para el caracter nulo ('\0'). El segundo y el tercer miembro, son punteros a estructuras, que como se observa en su declaración, apunta a estructuras del mismo tipo a la que forman parte. Haciendo una analogía con la fig. 19, el miembro texto[83] corresponde al elemento información, el miembro lista *siguiente corresponde a la flecha que apunta hacia la derecha y el miembro lista *anterior corresponde a la flecha que apunta hacia la izquierda. La palabra typedef es utilizada para permitir la declaración de estructuras del tipo struct lista. Así, *primero, *ultimo, *info y *punaux son estructuras del tipo LISTA que se definió previamente. La creación de *primero, *ultimo, *info y *panaux obedece a la necesidad de transferencia de direcciones, lo cual se realizará más adelante.

Esta es la forma de estructurar la información en nuestro editor. A continuación veremos la manera de trabajar con las lista doblemente ligada.

**PSEUDOCODIGO 1**

si (existe información en buffer del teclado de la LAPTOP)

Realizar tareas de edición

Enviar información a CSU si es un comando

si (existe información proveniente de la CSU)

Desplegar en monitor la información

Guardar la información en estructura de datos

El pseudocódigo 1 nos expresa que para efectuar cambios en la información almacenada en LAPTOP es necesario hacer uso del teclado. Pero no todo lo que se introduce en teclado constituye tareas de edición. Por ejemplo, el usuario debe comunicarle a la CSU por medio de comandos, lo que desea que la CSU realice. Por lo tanto la LAPTOP debe "saber" diferenciar entre una tarea de edición y una tarea de comunicación. Para resolver éste problema podríamos implementar una función que analizará toda la información, pero la implementación de ésta función necesariamente implicaría primero, tiempo para investigar todos comandos que acepta la CSU, y segundo mayor tiempo de programación. Una forma más sencilla es a través de la utilización de una sentencia caso (case en C) que contenga como opciones las tareas de edición y como opción default, el envío de la información a CSU. De ésta manera, si lo que se teclea no pertenece a ninguna de las tareas de edición, automáticamente se da por hecho de que se trata de un comando, sin importar de que verdaderamente lo sea, pues de no ser un comando, la CSU no lo interpretará como tal y mandará un mensaje de error a la LAPTOP. La LAPTOP a su vez visualizará el mensaje, guardándolo también en la estructura de datos (ver pseudocódigo 2).

**PSEUDOCODIGO 2**

```
Para (número indeterminado de veces)   ; Iteración infinita [10]
  si ( TECLAPRES( ) )                  ; ¿ existe información en buffer del teclado ?
    CARTEC = LEETECLA( )  ; se lee información
      caso ( CARTEC )                  ; selecciona opción
        ALT_X: Aborta sistema
        FL_ARRIBA: Posiciona cursor en línea anterior
        FL_ABAJO: Posiciona cursor en línea siguiente
        FL_IZQ: Posiciona cursor en la columna anterior
        FL_DER:Posiciona cursor en la columna siguiente
        INSERT: Activa edición
        PAG_ARRIBA: Cambia la página actual por la anterior
        PAG_ABAJO: Cambia la página actual por la siguiente
        CTRL_I: Inserta línea
        CTRL_B: Borra línea
        F10: Activa menú
        DEFAULT: Enviar datos a CSU ó editar información

  si ( CARPUERTO )     ; ¿ Existe información del puerto ?
    CARPTO = OBTCAR( )     ; se lee información del puerto
    Crear lista doblemente enlazada
    Visualizar CARPTO en pantalla
```

[10] En Turbo C, un bucle infinito se crea a través de la sentencia for(;;). En ella, debido a que ninguna de las tres expresiones que forman el bucle for se dan, se crea un bucle sin fin. Las maneras de salir del bucle es mediante la sentencia break y la función exit(). Esta función es precisamente la que se usa en la opción ALT_X para abortar el sistema.

En ésta refinación de pseudocódigo 1 se han añadido varias líneas necesarias para el desplazamiento del cursor en la pantalla y para tareas propias de edición. También se han añadido dos líneas que no tienen relación directa con la edición, esas líneas son las referentes a ALT_X que es la opción para salir del sistema de adquisición de datos, y la línea que contiene a F10, que será la que active el menú principal.

Otra de las cosas que podemos apreciar en el pseudocódigo 2, es el uso de funciones, como expresiones a evaluar. C permite evaluar cualquier expresión tales como variables, funciones, etc., que contengan ó que den como resultado valores booleanos. Así, por ejemplo si TECLAPRES( ) da un valor de uno cuando el buffer del teclado tiene información, las sentencias inmediatas siguientes se ejecutan.

Resulta lógico que para editar información es necesario contar con la información a editar. Esto obliga a detallar el pseudocódigo 2, comenzando con la rutina que crea la lista doblemente ligada.

### 3.3.2 Guardando información en lista y visualizándola en pantalla

```
if (!ultimo)   /* al principio ultimo=NULL */
{
   j=1;
   info=obt_mem_nodo( );  /* se asigna memoria al nodo info */
   info->siguiente=NULL;  /* se asigna NULL al elemento sig. */
   info->anterior=NULL;   /* se asigna NULL al elemento ant. */
   ultimo=primero=info;  /* ultimo y primero tienen misma dir. */
}
info->texto[i]=CARPTO;   /* almacenando inf. en la lista */
i++;   /* se incrementa i para grabar nuevo caracter */
info->texto[i]='/0';  /* grabando caracter nulo */
if(CARPTO!=LF&&j<=19)
{
   gotoxy(i,j);    putch(CARPTO);   /* visualización en pantalla */
}
if(CARPTO==LF)
{
   j++;
   info=obt_mem_nodo( );   /* memoria para nvo. nodo */
   ultimo->siguiente=info;   /* apuntando al nvo. nodo */
   info->anterior=ultimo;   /* apuntando al primer elemento */
   punaux=ultimo=info;   /* ultimo es el nvo. nodo */
   ultimo->siguiente=NULL;   /* cerrando lista */
   if(j>19)   /* línea 19, límite de ventana de trabajo */
   {
      j=19;   gotoxy(1,1);
      delline( );   /* borrando la línea superior */
      gotoxy(1,j);
   }
}
```

Inicialmente para introducir información en la lista debemos crear el primer nodo de dicha lista, para lo cual debemos solicitar memoria. Una vez que el sistema operativo asigna memoria, lo que sigue es asignarle a los punteros siguiente y anterior una dirección de memoria nula, ya que antes y después del nodo que se acaba de crear no existen otros nodos a los cuales apuntar. Es entonces cuando ya podemos introducir caracteres en la cadena texto[ ] y presentarlos en pantalla. La visualización de caracteres se consigue a través de la función putch( ). Para llevar a cabo la visualización es necesario que se cumplan dos condiciones, que CARPTO sea diferente de LF(line feed) y que j sea diferente o igual a 19. El número 19 es debido a que desde un principio (en otra parte del programa que no se presenta) se definió la ventana de trabajo con 19 líneas de texto.

Es un hecho que cuando el usuario teclea ENTER, una nueva línea debe ser creada. Es por esto que en una de las expresiones del programa anterior se verifica si el caracter en el puerto es LF (recuérdese la secuencia de caracteres que entrega la CSU cuando el usuario presiona ENTER en la LAPTOP). En caso afirmativo; es decir, que CARPTO sea igual a LF, j se debe decrementar inmediatamente, logrando que los siguientes caracteres se visualicen en la siguiente línea. La creación de una nueva línea conlleva necesariamente a la creación de un nuevo nodo y por consiguiente una nueva asignación de memoria.

En la creación del segundo nodo debemos hacer que el elemento siguiente del primer nodo ya no apunte a NULL, sino a la dirección de memoria que ocupa el segundo nodo. Para enlazar la lista y poderla recorrer en ambos sentidos, es necesario que el segundo nodo quede "unido" con el primero, lo cual se logra haciendo que el puntero anterior del segundo nodo apunte a la dirección de

memoria que ocupa el primer nodo. Como éste segundo nodo es el último creado, su puntero siguiente deberá apuntar a NULL. Este proceso se repite una y otra vez hasta finalizar el registro de la caverna.

Aprovechando la explicación expuesta, podemos decir que las operaciones de inserción y borrado de líneas, se basan en actualizaciones de las direcciones de los punteros anterior y siguiente con la diferencia de que en uno se asigna memoria y en el otro proceso se libera.

Aún quedan algunas líneas del código anterior por explicar. Observemos que existe una sentencia definida como if(j>19); en ella si j cumple con el requisito de ser mayor que 19, entonces debemos obligar a que j sea 19 y que se mantenga en ese valor, con el fin de que la nueva información por llegar se visualice en la línea 19 de la ventana de trabajo. Por ejemplo, supongamos que en la ventana de trabajo se han usado las líneas que van de la 1 a la 19. Cuando se teclea ENTER, j se incrementa en 1, conteniendo el valor de 20. Como no podemos desbordarnos del área de trabajo debemos desaparecer la línea de texto superior y subir las restantes, a fin de dejar espacio para la nueva línea que debe desplegar en el renglón 19. Esto explica el porqué se mantiene en 19 a j. Para nuestra fortuna no necesitamos implementar una rutina que elimine y al mismo tiempo suba las líneas por debajo de la borrada, pues existe una función en Turbo C, llamada delline( ) que efectúa ambas operaciones. Para su correcta utilización en el editor basta colocar el cursor en las coordenadas 1,1 de la ventana activa y entonces eliminar la línea del cursor. Para continuar, debemos regresar a la línea inferior y entonces desplegar la nueva información.

### 3.3.3 Acceso a la información almacenada en LAPTOP

Ahora sí estamos en condiciones de consultar toda la información almacenada en RAM. El acceso a esa información se lleva a cabo a través del uso de la flechas Pg Up, Pg Dw, flecha arriba "↑", flecha abajo "↓". Dado que la programación para las teclas Pg Up, Pg Dw se basa en la repetición de los procesos para flecha arriba y flecha abajo respectivamente, sólo se explicarán los códigos correspondientes a las últimas dos teclas.

Como primer paso en la codificación de la tecla flecha arriba, checaremos si existe información previa al nodo en que estamos situados al momento de empezar a recorrer la lista. Para que esto sea posible necesitamos un puntero que nos ayudará a efectuar el recorrido. Este puntero debe tener como dirección inicial la misma que la del último nodo creado. Observemos en el último programa presentado, que existe una variable que cumple con esos requisitos y que se llama punaux. En realidad ésta es la razón por la que aparece en el código.

Con un poco de análisis caemos en la cuenta que cada vez que se presiona la tecla flecha arriba, retrocedemos en la lista; es decir, utilizamos los elementos anteriores para recorrer la lista hacia atrás. Veamos el siguiente listado:

```
case FL_ARRIBA: if(punaux->anterior!=NULL)
                {
                 /* retrocediendo en la lista */
                 punaux=punaux->anterior;
                 j--;
                }
                else putch(BEEP);
                if(j<=0)
                {
                 /* insertando una linea */
                 insline( );
                 j=1;
                 gotoxy(1,j);
                 /* visualizando la linea accesada */
                 for(cont=0,cont<80,cont++)
                   putch((int)punaux->texto[cont]);
                 gotoxy(1,j);
                }
                break;
```

Dado que j es la variable que guarda el número de renglón de la línea de texto, toda vez que se accesa a una línea anterior, j debe ser decrementada en una unidad. Cuando j se ha decrementado hasta quedarse con un valor de 1, el cursor se encontrará en el primer renglón. Si el usuario desea seguir consultando más información y presiona tecla arriba, j se decrementa a 0. Como nos hallamos en el límite superior de la ventana de trabajo, tenemos que insertar una línea en blanco y mover las líneas que se encuentran por debajo, a fin de que la nueva información se despliegue en el renglón 1. Eso es precisamente la tarea de la función *insline( )* de la cual podemos decir que es la función inversa a *delline( )*. Una vez que contamos con espacio vacío en la parte superior, podemos escribir el texto

correspondiente al nodo accesado. Si el usuario continua su consulta, llegará un momento en que alcance al primer nodo y no pueda accesar más información. Para cuando ésto suceda se dispone de una señal audible que alertará al usuario.

El recorrido de la lista en sentido contrario, o sea, hacia adelante se lleva a cabo en forma muy similar al recorrido hacia atrás, solo que en vez de utilizar el puntero anterior se utilizará el puntero siguiente, con lo que j deberá incrementarse cada vez que "↓" sea presionado. La función que sustituirá a insline( ) será deline( ), la cual funcionará cuando hallamos llegado al límite inferior de la ventana de trabajo. El código correspondiente a flecha abajo es el que a continuación se describe.

```
case  FL_ABAJO : if(punaux->siguiente!=NULL)
                 {
                  /* avanzando en la lista */
                  punaux=punaux->siguiente;
                  j++;
                 }
                 else putch(BEEP);
                 if(j>19)
                 {
                  /* borrando una linea */
                  delline( );
                  j=19;
                  gotoxy(1,j);
                   /* visualizando la linea accesada */
                  for(cont=0,cont<80,cont++)
                    putch((int)punaux->texto[cont]);
                  gotoxy(1,j);
                 }
                 break;
```

Las tareas de edición requieren que el cursor se pueda mover tanto en forma vertical como horizontal. Aún falta explicar como es que se consigue "navegar" a través de la información en RAM, pero ésta vez con el uso de las teclas flecha derecha "→" y flecha izquierda "←". Los códigos para estas teclas son más sencillas que para los de desplazamiento vertical. Las teclas "→" y "←" están relacionadas con las operaciones de edición como podemos apreciar en los siguientes listados.

```
case FL_IZQ:  if(inserta)
              {
                gotoxy(--x,j);
                puntedic=&(texto[x-1]);
              }
              else gotoxy(--x,j);
              break;

case FL_DER:  if(inserta)
              {
                gotoxy(++x,j);
                puntedic=&(texto[x-1]);
              }
              else gotoxy(++x,j);
              break;
```

*Inserta* es una variable cuyo valor cambia de 0 a 1 y viceversa cuando la tecla *insert* es presionada. La transición de un valor a otro se efectúa en la opción INSERTA mostrada en el pseudocódigo 2. Si la variable inserta tiene el valor de 1, será porque el usuario presionó la tecla insert, por lo que todo lo que tecleará en lo sucesivo, cambiará la información original de la línea en donde está situado el

cursor. x es el número de la columna en que se sitúa, por lo que debe ser decrementado en una unidad cada vez que "←" sea presionado. La expresión puntedic=&(texto[x-1]), es muy significativa pues provee el medio por el cual se puede efectuar la edición. Observemos que hay una relación entre la posoción del cursor (x) y el índice del arreglo texto[x-1] que almacena los caracteres de la línea de texto a la cual accesamos. La unidad que se sustrae a x se debe a que los índices en los arreglos comienzan en 0. Por ejemplo, supongamos que tenemos escrito enla pantalla la palabra SCAN y que el cursor está posicionado en la letra C por lo que x equivale a 2. Los caracteres en el arreglo de esa línea está dispuesta de la siguiente forma:

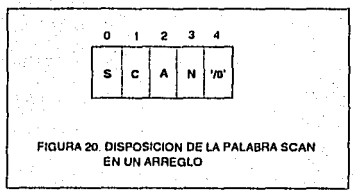



FIGURA 20. DISPOSICION DE LA PALABRA SCAN EN UN ARREGLO

Como podemos ver la letra C está en la celda 1 y no en la 2 como podríamos suponer. Restando 1 al valor de x en el índice del arreglo logramos accesar el caracter C, consiguiendo que exista una relación directa entre lo que se accesa en pantalla y lo que se accesa en RAM a través del arreglo texto[x-1].

Volviendo al código de FL_IZQ podemos ver que cuando la variable inserta es 0, la expresión a ejecutar es: gotoxy(--x,j), la cual regresa el cursor una posición

atrás y decrementa en 1 a x. Con relación al código de FL_DER diremos que es muy parecido al de FL_IZQ, salvo que, en ésta ocasión x se incrementará en 1 cada vez que la tecla "→" sea presionada.

Ahora que ya podemos movernos a través de la información en dirección a los 4 puntos cardinales, estamos en condiciones de hacerle cambios a la información, lo cual se consigue, primero presionando la tecla insert y después introduciendo los cambios. Al oprimir la tecla insert se da por hecho que el usuario editará y por tanto debemos preparar al arreglo texto[i] para aceptar esos cambios. En la expresión puntedic=&(texto[x-1]) que también pertenece al código de la opción INSERTA, copiamos a través de la variable puntedic (puntero de edición), la dirección del caracter en la que el usuario está posicionado, así si el usuario empieza a editar, esa edición se registrará de inmediato. Como cada tecla presionada en LAPTOP es comparada con las opciones del módulo editor (ver pseudocódigo 2), se hace necesario que el almacenamiento de la nueva información se codifique en la opción default de la siguiente manera:

```
case DEFAULT :  if(inserta)
                {
                  putch(CARTEC);
                  x++;
                  *pedic=(char) CARTEC;
                  pedic++;
                }
                .
                .
                .
                break;
```

El caso DEFAULT, además de grabar la edición de usuario, también despliega en pantalla dicha edición mediante el uso de la función putch. Esta función después de desplegar el caracter de edición, mueve el cursor a la posición siguiente, por lo que es necesario incrementar x y mover el puntero pedic también una posición adelante con el objeto de que la edición continúe y se lleve a cabo correctamente.

## 3.4 Módulo de Interfaz de Usuario

En el capítulo 2, cuando se hizo el análisis de éste módulo se mencionó la forma de trabajar de la interfaz de usuario. Dijimos que para un mejor control de las operaciones a realizar dentro del programa adquisitor se iban a utilizar los menús de selección y de acceso de información. También se mencionó la forma de trabajo de esos menús. Toca turno ahora al diseño de la interfaz de usuario para lo cual debemos recordar algunas cosas referentes a las características de los menús citados. En el apartado 2.6.2, mencionamos que un menú de selección era aquel que presentaba varias opciones a elegir, mientras que el de acceso de información presenta varias preguntas, las cuales deben ser contestadas exaustivamente por el usuario. También se dijo que ambos menús son similares en la forma de activarse y desactivarse, siendo una de las diferencias la forma de procesar la información, pues uno ejecuta una acción en base a la opción elegida y el otro en función a la información introducida desde el teclado.

Comenzaremos el diseño de la interfaz de usuario programando el menú de selección.

### 3.4.1 Menú de Selección

Para crear un menú de éste tipo debemos pasar una lista de opciones a la rutina encargada de desplegar el menú de selección, lo cual se hará mediante el uso de un puntero a un arreglo bi-dimensional que contendrá las cadenas u opciones. Como se mencionó en 2.6.2, la opción del menú puede ser seleccionada mediante su sobre-iluminación y presionando ENTER o presionando una tecla asociada a esa opción. Para que la función del menú reconozca que operación se debe realizar, los caracteres asociados a cada opción le deben ser pasados también. La forma de hacer esto es pasando una cadena que contenga los nombres de todos los caracteres clave en el mismo orden que las cadenas del menú.

En general la función de despliegue y proceso del menú de selección ( que para efectos de explicación se llamará menu_select( ) ) debe realizar 5 tareas:

1) Salvar la parte de la pantalla usada por el menú.
2) Visualizar el borde si es requerido.
3) Visualizar el menú.
4) Procesar respuesta del usuario.
5) Restaurar la pantalla a su posición original.

#### 3.4.1.1 Salvar y Restaurar la pantalla a su condición original

Las tareas 1 y 5 se llevan a cabo mediante la utilización de dos funciones que forman parte de las librerías de Turbo C. Esas funciones reciben el nombre de gettext( ) y puttext( ). La primera almacena la información que se encuentra en el lugar que ocupará el menú. Su formato es el siguiente:

int gettext(int izquierda, int arriba, int derecha, int abajo, void *destino).
Izquierda, arriba, derecha y abajo son las coordenadas de la información a almacenar, mientras que *destino debe ser un buffer en donde se va a guardar esa información.

La función puttext( ) es la función inversa de gettext( ), pues copia la información que se almacenará previamente en *destino a la pantalla; su formato es:

int puttext(int izquierda, int arriba, int derecha, int abajo, void *origen).
De igual forma izquierda, arriba, derecha y abajo son las coordenadas en donde se colocará de nueva cuenta lo almacenado en *destino u *origen.

### 3.4.1.2 Visualización del borde

La visualización del borde se consigue mediante dos iteraciones ( en las cuales se dibujan las cuatro líneas que dibujan un marco ) y cuatro instrucciones más para dibujar las esquinas. En la primera iteración se trazan la parte superior e inferior del marco, en la otra iteración se trazan los lados, para posteriormente dibujar las cuatro esquinas. Por ser una rutina sencilla, no se abundará más en detalles al respecto.

### 3.4.1.3 Visualización del menú

La clave para la tarea 3 es que a la función menú_selec( ) le sea pasado un puntero a un arreglo cadenas. Para visualizar una cadena individual, sólo hay que indexar el puntero como un arreglo. La siguiente rutina escrita, llamada visualiza_menú( ), visualiza cada opción del menú utilizando éste método.

```
/*visualiza el menu en su posicion*/
void visualiza_menu(menu,teclas,x,y,contador,pos)
char *menu[];
char *teclas;
int x, y, contador;
char pos;
{
  register int y;   int numenu=0;
  if(pos=='h') /* si el menu es principal o horizontal */
  {
   window(1,2,80,2);
   for(i=0;i<contador;i++,x+=10)
   {
       gotoxy(x,y);
       escribe_video(x,y,menu[numenu],teclas,NORM_VID);
       teclas++;     x=strlen(menu[numenu])+x;
       numenu++;
   }
  }
  else   /* el menu es vertical */
  {
   window(1,3,14,9);
   for(i=0;i<contador;i++,y++)
   {
       gotoxy(x,y);
       escribe_video(x,y,menu[numenu],teclas,NORM_VID);
       teclas++;
       numenu++;
   }
  }
}
```

Como podemos ver, aparte del puntero menú, también es nesario pasar como parámetros las letras claves ó *teclas calientes*, las coordenadas x,y (desde donde comenzarán a visualizarse las opciones), el número de opciones del menú y la posición del menú en la pantalla.

La forma para crear el arreglo bi-dimensional que contenga las cadenas de selección del menú es crear variables usando ésta forma general:

```
char *<nombre del menú>[ ] =
{
        "primera selección"
        "segunda selección"
        .
        .
        "n-ésime selección"
};
```

Así, la siguiente declaración crea un arreglo llamado **archivo** que contiene operaciones relacionadas con los archivos que se obtendrán en el registro de las cavernas.

```
char *archivo[ ] =
{
        " Nombrar "
        " Grabar "
        " Editar INS "
        " Imprimir F1 "
        " Salir ALT_X "
};
```

### 3.4.1.4 Procesar respuesta del usuario

Esta es la cuarta tarea a realizar por el menú de selección. Dado que es la que procesa la selección del usuario, resulta muy interesante su diseño. Dicha selección puede ser elegida de las dos formas siguientes: la primera, mediante las teclas de desplazamiento vertical, moviendo la sobre-iluminación a la opción deseada y entonces pulsar ENTER. La segunda manera de seleccionar una opción es presionando su tecla asociada. Para efectos de explicación llamaremos a la función que realiza estas acciones *procesa_resp( )*. En el siguiente código podemos observar el mecanismo de funcionamiento de ésta función.

```
/*obtiene seleccion del usuario en menus verticales*/
procesa_resp(x,y,contador,menu,teclas)
int x,y,contador;
char *menu[];
char *teclas;
{
 union tecla_in
 {
  char ch[2];
  int i;
 }c;
 int flecha=0, tecla_elegida;
 char *teclave;
 y++;

  /*sobre_ilumina la primera seleccion*/
  gotoxy(x,y);
  escribe_video(x,y,menu[0],teclas,VID_INV);
```

```
                    break;

            case FL_ABAJO:
                    flecha++;
                    teclas++;
                    break;
        }
      }
      if(flecha==contador)
      {
        flecha=0;
        teclas=teclave;
      }
      if(flecha<0)
       {
        flecha=contador-1;
        teclas = teclave+4;
       }
       /* sobre-ilumina la siguiente seleccion */
       gotoxy(x,y+flecha);
       escribe_video(x,y+flecha,menu[flecha],teclas,VID_INV);
    } /* cierra for */
}
```

Cuando procesa_resp() comienza a ejecutarse, la primera opción del menú aparece en VID_INV (video inverso), mientras que las demás se muestran en VID_NORM (video normal). Posteriormente, la rutina entra en un bucle que espera la respuesta del usuario. Para tal propósito se dispone de una función que lee la tecla presionada. En el caso de que sea un caracter el que se teclee, éste se compara con las teclas asociadas a las opciones, sino sucede así, entonces puede tratarse de que se haya presionado las teclas de flecha, en ambos casos se genera

un número entero, el cual corresponderá a la opción deseada. Si se ha tecleado un caracter clave, el retorno es inmediato, mientras que si se ha presionado alguna flecha el retorno del número entero dependerá de si a continuación se presiona la tecla Enter.

Cada vez que una tecla de flecha es presionada, la opción que está sobre-iluminada es revisualizado en video normal y el siguiente se sobre-ilumina. Si se presiona la tecla flecha abajo cuando la opción sobre-iluminada es la última, la primera opción es la que se sobre-ilumina como si se girara en torno al menú. Lo mismo ocurrirá en el caso que nos encontremos en la primera opción y pulsemos la tecla flecha arriba.

Finalmente, la función escribe_video() es usada por procesa_resp() para escribir una cadena en la pantalla en la posición indicada por la coordenada x,y usando el atributo *atrib* especificado.

### 3.4.2 Menú de Acceso de Información

A diferencia de la forma de procesar la respuesta del usuario, este menú se activa y desactiva de la misma forma que el selección. En otras palabras, las tareas de salvar, visualizar menú y restaurar pantalla son similares en ambos menús. Es por esto que sólo nos concentraremos en la forma de procesar la respuesta de usuario.

Dado que en los menús de acceso de información las respuestas de usuario deberán estar correctas, es necesario validarlas. Tal validación se hace mediante el

uso de una rutina cuyo objetivo es precisamente verificar la validez de la información introducida. Esta rutina se proporcionará como parte de una estructura denominada blanco (refiriéndonos al área del menú en donde se introduce la respuesta del usuario - ver fig. 16). Esta estructura contiene 4 elementos (ver sig. listado). El elemento int loc_x, loc_y se refiere a la localización dentro de la ventana que alojará el menú, el elemento char *resp, es la variable que almacenará la información a introducir por el usuario, int long es la variable que almacenará la longitud de la cadena *resp, y finalmente verifica es la función que se encargará de checar la respuesta del usuario.

```
struct blanco
    {
        int loc_x, loc_y;
        char *resp;
        int long;
        int (*verifica) (char*);
    };
```

Para mostrar como se utilizó la estructura anterior, ejemplificaremos usando el menú para el establecimiento de los parámetros de comunicación. Los parámetros, tal como se comentó en 2.3 son la velocidad y el puerto serie de comunicación.

Como se trata de dos parámetros, debemos crear un arreglo de estructuras del tipo blanco tal como se muestra a continuación:

```
#define X_LOC  16
#define X_LOC2  11

struct blank blanks[] = {
{
        X_LOC, 1,       /* Localizacion del blanco */
        baudios,        /* respuesta */
        sizeof(baudios),  /* Longitud */
        verify_ok_bauds  /* Verifica respuesta del usuario */
}, {
        X_LOC, 2,       /* Localizacion del blanco */
        puerto,         /* respuesta */
        sizeof(puerto),   /* Longitud */
        verify_ok_pto    /* Verifica respuesta del usuario */
}, {
        -1, -1,          /* limites de captura */
        NULL,
        0,
        NULL
   }
};
```

La información almacenada en el arreglo anterior se pasa como parámetro a una rutina la cual se encarga de procesar la información tecleada por el usuario. A ésta rutina la denominaremos obt_cadena (obtener cadena) y trabajará de la siguiente forma:

```
int obt_cadena (struct blanco *blanco)
{
  int x_pos = 0;  /* posicion x dentro de la respuesta */
  int ch;         /* caracter con el que se está trabajando */

  while(1) {
        textatr(VID_INV);
        gotoxy(blanco->loc_x,blanco->loc_y);
        cprintf("%-*s", blanco->long,blanco->resp);
        textatr(NORM_VID);
        gotoxy(blanco->locx + x_pos,blanco->loc_y);
        ch = x_getch();
```

```
        switch(ch){
                .
                .
                .
                .
        case ESC:
                return(MENU_ABORT);
        case ENTER:
                if(x_pos!=0)
                 blanco->resp[x_pos]='\0';
                if(blanco->verifica(blanco->resp)==VERIFY_OK)
                 return(ch);
                break;
        default:
                if(isprintf(ch)){
                 if(x_pos==0)
                   blanco->resp[1]='\0';
                 if(blanco->resp[x_pos] =='\0');
                   blanco->resp[x_pos+1] = '\0';
                 blanco->resp[x_pos] = ch;
                 x_pos++;
                }
                break;
      }
     }
}
```

En el código anterior debemos observar cómo después de dibujar los blancos dentro de la ventana, se va leyendo caracter por caracter la respuesta del usuario. Los caracteres introducidos pueden ser de diferentes tipos, aunque por motivos de simplificación sólo se muestran los casos de cuando se teclea Esc, Enter y cualquier caracter imprimible (los cuales serán almacenados en resp).

Cuando se teclea Esc la computadora simplemente reacciona abandonando la ventana. El caso Enter funciona cuando el usuario introduce su información y desea procesarla. Como no se conoce la validez de tal información es necesario verificarla, acción realizada por la función ( *verifica ) asociada a la estructura *blanco* en proceso.

Las macros VERIFY_OK y MENU_ABORT fueron definidas previamente y nos ayudan a llevar un control adecuado de los diferentes caminos que toma el flujo de control dentro del sistema adquisitor.

## 3.5 Módulo de Conversión

El proceso de conversión del archivo de registro a un formato compatible con el programa de interpretación, conforma la última etapa en el sistema adquisitor. Se supone que el usuario para llegar a ésta etapa debió hacer el registro de la caverna y posteriormente haber editado la información almacenada. Estas actividades previas garantizan que el archivo a convertir exista y que por lo tanto se pueda efectuar la conversión.

En forma general el módulo de conversión realiza 5 actividades básicas. La forma en que se ejecutan se detalla en el siguiente pseudocódigo.

**PSEUDOCODIGO**

- ABRIR archivo a convertir
- LEER línea del archivo a convertir
- SELECCIONAR información
- CREAR archivo de interpretación
- Si (fin de archivo a convertir)

    cerrar archivo a convertir y salir del módulo

  Si no

    ir a proceso de LEER línea del archivo a convertir

La tarea más importante y por tanto la que implica mayor programación se centra en la actividad que selecciona la información para crear el archivo de interpretación. Esta es la razón por la que el diseño del módulo de conversión girará en torno a la actividad mencionada.

Para explicar el funcionamiento del pseudocódigo anterior debemos entender que el archivo a convertir será leído línea por línea hasta que el archivo se termine. En la lectura de cada línea debemos preguntar si las palabras contenidas en ella corresponde a alguna de las palabras reservadas que nos interesan y que nos darán la pauta para seleccionar información y generar el archivo de interpretación. Una inspección minuciosa en los listados impresos en KPU, indican que:

1) Antes de cada despliegue de datos(azimuth, radio, etc.), siempre aparece la frase SCAN DEPTH:xxxx.xx M, indicando la profundidad a la que se hace el barrido.

2) Posterior a la palabra SCAN DEPTH y a una linea en blanco se imprimen los nombres correspondientes a los datos de la caverna (p. ej.: ETIM - tiempo que tarda en registrarse el eco en el transductor-, AZIV ó AZIH - Azimuth vertical u horizontal, HRAD ó PVR - Radio para transductor horizontal ó vertical respectivamente -).

3) El final de cada barrido de estación se indica con las palabras en inglés LOG STOPPED ( ver figs. 4 y 12).

SCAN DEPTH, ETIM, AZIV, AZIH, HRAD, PVR, PVRD y LOG STOPPED constituyen el conjunto de palabras claves que debemos buscar en el archivo a convertir, pues seguramente contendrán los valores de los datos necesarios para crear el archivo de interpretación según el formato de la sección 2.5. Esta filosofía de búsqueda es utilizada por el programa que a continuación se detalla:

```
fprintf(fconv,"%s\n",fech);
fprintf(fconv,"%s\n",pozo);
fprintf(fconv,"%s\n",oper);

/* lee linea a linea el archivo */
while(fgets(linea,83,fp))
{
 aplin=linea;
 switch(estado)
 {
  /* caso de que la frase sea SCAN DEPTH: */
  case 'A' :
                if(obten_simbolo()==SCAN && obten_simbolo()==DEPTH
                  && *aplin==':')
                {
                 aplin++;
                 if(obten_simbolo()==NUMERO)
                 {
                  strcpy(profun,objeto);
                  fprintf(fconv,"%s\n",profun);
                 }
                 else
                 {
                  strcpy(profun,"0.0000");
                  fprintf(fconv,"profun");
                 }
                 fprintf(fconv,"%s\n",fluido);
                 estado='B';
                }
                break;
```

```
/* seleccionando datos */
    case 'B' :
                    if(obten_simbolo()!=ETIM)
                     break;
                    columazim=0; columphr=0;
                    columhrad=0; columpvrd=0;
                    for(col=1;(cod=obten_simbolo())!=FIN_LINEA;col++)
                    {
                     if(cod==AZIH || cod==AZIV) columazim=col;
                     if(cod==PVR)   columphr=col;
                     if(cod==HRAD || cod==VRAD) columhrad=col;
                     if(cod==PHRD || cod==PVRD) columvhrd=col;
                    }
                    estado='C';
                    break;

    /* almacena parámetros de la caverna hasta alcanzar LOG STOPPED */
    case 'C' :
                    if(cod=obten_simbolo())==NUMERO)
                    {
                     for(col=1;obten_simbolo()!=FIN_LINEA;col++)
                      strcpy(variables[col],objeto);
                     if(columazim)
                      fprintf(fconv,"%s\n",variables[columazim]);
                     if(columphr)
                      fprintf(fconv,"%s\n",variables[columphr]);
                     else
                      fprintf(fconv,"%s\n",variables[columhrad]);
                     if(columpvrd)
                      fprintf(fconv,"%s\n",variables[columpvrd]);
                     else    fprintf(fconv,"%s\n",profun);
                    }
```

```
                if(cod==LOG && obten_simbolo()==STOPPED)
                {
                 fprintf(fconv,"-999\n");
                 estado='A';
                }
                break;
    default :
                estado='A';
                break;
   }
  }
  fclose(fp);
  fclose(fconv);
}
```

El programa comienza grabando los valores correspondientes a la fecha, nombre del pozo y número de operación en el archivo de conversión "fconv", los cuales fueron introducidos en la ventana de la opción de CONVERSION. Desde luego, es necesario que las variables que almacenan esos valores sean del tipo *extern* para que se pueda manipular en éste módulo.

La función fgets es la encargada de leer línea a línea el archivo a convertir (*fp*), almacenando la información de cada línea en el arreglo de caracteres llamado *línea*. Para recorrer la línea leída se dispone de un apuntador a caracter denominado *aplin*. Aplin será utilizado por la función *obten_simbolo* que se encargará a su vez de buscar las palabras claves y eliminar aquellas que no lo sean. Así pues la primera frase a encontrar es la de SCAN DEPTH:. Si la búsqueda es exitosa se espera a continuación un número que corresponde a la profundidad del registro y que por lo tanto debe ser almacenado como tal. Para ser congruentes con

el formato especificado en la sección 2.5, a continuación del valor de la profundidad debe escribirse la velocidad del fluido. A fin de continuar la búsqueda de información debemos cambiar el estado del flujo del sistema a B.

Es en el caso B en donde se realiza la selección de la información correspondiente a la azimuth, radio y profundidad. Como no se sabe en que orden éstos datos se encuentran dispuestos en el archivo a convertir lo primero que hay que hacer es verificar precisamente ese orden. Esto último se realiza identificando los encabezados AZIH, AZIV, PVR, HRAD, PHRD y PVRD y almacenando en una variable la posición en que aparecen. Conociendo ésta posición es fácil almacenar la información en el archivo de interpretación en su correcto orden. El caso c es el encargado de efectuar éstas últimas actividades. Su proceso comienza leyendo la línea que contiene los valores numéricos almacenándolos en un arreglo. Como sabemos previamente en que orden deben ser almacenados los datos en el archivo de interpretación, basta indexar el arreglo con el valor de la variable obtenida al identificar los encabezados y grabarlo en el archivo de interpretación en el orden conocido.

El proceso obliga a que el flujo de control no cambie a otro estado hasta que se alcance el fin de barrido de una estación, o en otras palabras hasta que se lea LOG STOPPED, almacenando entonces en fconv el valor de -999. Los estados A, B y C nos garantizan que se lean todos los datos importantes del archivo a convertir. Finalmente el proceso termina cuando se alcanza el fin de archivo.

El módulo de conversión constituye la última etapa del proceso dentro del sistema adquisitor, por medio de la cual se obtuvo un archivo de mucha importancia: *el archivo que contiene los datos para generar las gráficas que determinan el volumen y forma de la caverna subterránea en estudio.*

# CONCLUSIONES

Para evaluar los resultados obtenidos a través de la implementación y uso del sistema de adquisición de datos, creo conveniente en primer lugar, restablecer los objetivos propuestos al comienzo de la tesis y comprobarlos con los logros alcanzados.

En el capítulo 1, más específicamente, en el apartado 1.5, se mencionó que los objetivos primordiales consistirían en el análisis, diseño e implementación de un sistema de adquisición de datos, tendientes a modernizar el proceso del manejo de la información de cavernas subterráneas. La implementación de tal sistema daría entonces como resultados:

- menores tiempos de operación,
- mayor confiabilidad,
- menor costo de operación,
- mayor eficiencia.

Ahora que el proceso de elaboración del sistema ha concluido, podemos afirmar que en efecto, todas éstas ventajas se ofrecen. El hecho de prescindir de una unidad que presentaba la información en forma impresa y de haberla sustituida por una LAPTOP que almacena esa información en su memoria principal y secundaria, obliga al usuario a tener que familiarizarse con una forma de trabajo más sencilla. La sencillez del sistema adquisitor se basa en la integración de los procesos de adquisición, edición y conversión de información, todo bajo una misma plataforma de trabajo. Lo anterior redunda en menores tiempos de operación y por ende en

una mayor eficiencia, pues los procesos de edición y conversión dejan de realizarse en forma manual. Por otra parte la sustitución de la KPU por una LAPTOP trae como consecuencia inmediata el ahorro de papel termosensible, disminuyéndose así el costo de operación.

Todo desarrollo de software, una vez finalizado, es susceptible de mejorarse. El sistema adquisitor no es la excepción y dentro del trabajo a futuro, se contempla la incorporación del mouse como periférico de entrada a fin de agilizar la interacción con la LAPTOP.

Cabe señalar que el sistema adquisitor forma parte de un conjunto de proyectos que fueron propuesto por el departamento de Instrumentación y Control del IMP y que han sido aceptados por PEMEX y que además una versión del mismo sistema ya ha sido utilizado en cursos de capacitación sobre uso y manejo de la CSU, contando con muy buena aceptación por parte de los expositores del curso.

Es motivo de satisfacción personal el saber que el esfuerzo que implicó la elaboración del sistema de adquisición de datos se haya transformado en algo útil para el ejercicio de las labores de otras personas; así también la redacción del presente trabajo, pues siempre se me presentó como un reto a superar. De éste modo, el desarrollo completo de la tesis me aportó muchos conocimientos: primero, los relacionados a las técnicas de programación y segundo, los relacionados a la elaboración de trabajos de investigación.

# BIBLIOGRAFIA


**[BYRON 91]**

Byron, Gottfried S., 1991, **Programación en C**, Mc Graw-Hill, Madrid, España.

**[GOFTON 86]**

Gofton, Peter W., 1986, **Mastering Serial Communications**, SYBEX Inc., Berckley, California, U.S.A.

**[HALSALL 89]**

Halsall, Fred., 1989, **Data communications, computer network and open systems**, Addison-Wesley, U.S.A

**[JOYANES 87]**

Joyanes, Luis, 1987, **Metodología de la Programación**, Mc Graw-Hill, México, D.F.

**[PEREZ 92]**

Pérez, Tito L., 1992, *Respuesta de los Registros Geofísicos en Cavidades Subterráneas*, **Ingeniería Petrolera**, V. XXXII, No.4, Abril, PP 19-28.

**[PRESSMAN 92]**

Pressman, Roger S., 1992 , **Ingeniería de Software: Un enfoque práctico**, Mc GRAW-HILL, MEXICO, D.F.

**[SCHLUMBERGER 81]**

Schlumberger Well Services, 1981, **Schlumberger Well Services Maintenance Manual, Keyboard/Printer Unit**, U.S.A., Editado por Schlumberger Well Service, Información confidencial.

## REFERENCIAS ADICIONALES UTILIZADAS EN LA ELABORACION DEL SISTEMA ADQUISITOR


- The Peter Norton Computing Group, 1992, **Advanced Programming: The Accessible Guide To Profesional Programming**, BRADY Publishing, New York, N.Y, E.U.A.

- Schildt, Herbert, 1989, **Programación C: Guía para usuarios expertos**, Mc Graw-Hill/Interamericana, Madrid, España.

- Schildt, Herbert, 1990, **Programación avanzada**, Mc Graw-Hill, México, D.F.

- Schildt, Herbert, 1990, **Manual de Bolsillo. Turbo C**, Mc Graw-Hill, México, D.F.

- Que Corporation, 1989, **Manual de Bolsillo. Funciones del DOS y BIOS**, Addison-Wesley Iberoamericana.

# APENDICE A

| Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | NUL | 32 | 20 | | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 01 | SOH | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 02 | STX | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 03 | ETX | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 04 | EOT | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 05 | ENQ | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 06 | ACK | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 07 | BEL | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 08 | BS | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 09 | HT | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | 0A | LF | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | 0B | VT | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | 0C | FF | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | 0D | CR | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | 0E | SO | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | 0F | SI | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | DLE | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | DC1 | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | DC2 | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | DC3 | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | DC4 | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | NAK | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | SYN | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | ETB | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | CAN | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | EM | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | SUB | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | ESC | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | FS | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | GS | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | RS | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | US | 63 | 3F | ? | 95 | 5F | | 127 | 7F | □ |

CARACTERES ASCII QUE SE PUEDEN GENERAR CON SIETE BITS DE DATOS