UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Escuela Nacional de Estudios Profesionales Aragón

Desarrollo del Sistema de Estados de Resultados Financieros (S.E.R.F.)

T E S I S

QUE PRESENTA

ARTURO LEON JUAREZ

PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

MEXICO, D. F. **1995**

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ARAGON

SISTEMA DE ESTADOS DE RESULTADOS FINANCIEROS (S.E.R.F.)

Alumno:

ARTURO LEON JUAREZ

Fecha:

01/MAYO/1995

INDICE

	Página
INTRODUCCION.	
1 CONSECUENCIAS DE LA SEPARACION DE PEMEX.	
1.1 Introducción.	1
1.2 La separación de Pemex Gas y Petroquímica Básica.	2
1.3 Conceptos para el manejo del presupuesto.	3
2 UTILERIAS UTILIZADAS PARA LA APLICACION.	
2.1 Intruducción.	8
2.2 El precompilador Pro*C.	8
2.2.1 Introducción al precompilador Pro*C.	9
2.2.2 La Base de Datos Relacional ORACLE (RDBMS ORALE).	9
2.2.3 Características y beneficios del precompilador Pro*C.	9
2.2.4 Conceptos generales.	9
2.2.5 Mezclando comandos de C con instrucciones de SQL.	10
2.2.6 El comando EXEC SQL y EXEC ORACLE como prefijo.	10
2.2.7 Ejecución y declaración de las instrucciones de SQL.	11
2.2.8 Partes de un programa en PRO*C.	11
2.3 Introducción a la herramienta SQL*FORMS.	12
2.3.1 Desarrollo de aplicaciones.	12
2.3.2 Objetos de SQL*FORMS.	13
2.3.3 Un enfoque acerca de objetos.	14
2.3.4 Jerarquía de objetos.	14
2.3.5 Bloques.	15
2.3.6 Relación entre bloques.	16
2.3.7 Los componentes de SQL*FORMS.	16
2.3.7.1 Campos.	16
2.3.7.2 Páginas.	17
2.3.7.3 Triggers.	18
2.3.8 Modelos de procesamiento.	19
2.3.8.1 Eventos y funciones.	19
2.3.8.2. El punto de un trigger.	19
2.3.9 Componentes de SQL*FORMS.	19
2.3.10 Términos de SQL*FORMS y Base de Datos.	20
2.4 Introducción al lenguaje C.	24
2.4.1 Variables y Constantes.	25
2.4.2 Operadores.	28
2.4.2.1 Operadores aritméticos.	29
2.4.2.2 Operadores relacionales y lógicos.	29
2.4.3 Control de flujo de datos.	32
2.4.3.1 Proposiciones y bloques (Sentencias Primitivas).	32
2.4.3.2 Loops.	32
2.4.3.3 Sentencias de opción multiple.	34
2.4.4 Estructura de las funciones.	37
2.4.4.1 Recursividad.	39
2.4.4.2 Llamada de funciones.	39
2.4.5 Apuntadores y arreglos.	40
2.4.5.1. Arreglos de apuntadores.	41

INDICE

	Página
INTRODUCCION.	
1 CONSECUENCIAS DE LA SEPARACION DE PEMEX.	
1.1 Introducción.	1
1.2 La separación de Pemex Gas y Petroquímica Básica.	2
1.3 Conceptos para el manejo del presupuesto.	3
2 UTILERIAS UTILIZADAS PARA LA APLICACION.	
2.1 Intruducción.	8
2.2 El precompilador Pro*C.	8
2.2.1 Introducción al precompilador Pro*C.	9
2.2.2 La Base de Datos Relacional ORACLE (RDBMS ORALE).	9
2.2.3 Características y beneficios del precompilador Pro*C.	9
2.2.4 Conceptos generales.	9
2.2.5 Mezclando comandos de C con instrucciones de SQL.	10
2.2.6 El comando EXEC SQL y EXEC ORACLE como prefijo.	10
2.2.7 Ejecución y declaración de las instrucciones de SQL.	11
2.2.8 Partes de un programa en PRO*C.	11
2.3 Introducción a la herramienta SQL*FORMS.	12
2.3.1 Desarrollo de aplicaciones.	12
2.3.2 Objetos de SQL*FORMS.	13
2.3.3 Un enfoque acerca de objetos.	14
2.3.4 Jerarquía de objetos.	14
2.3.5 Bloques.	15
2.3.6 Relación entre bloques.	16
2.3.7 Los componentes de SQL*FORMS.	16
2.3.7.1 Campos.	16
2.3.7.2 Páginas.	17
2.3.7.3 Triggers.	18
2.3.8 Modelos de procesamiento.	19
2.3.8.1 Eventos y funciones.	19
2.3.8.2. El punto de un trigger.	19
2.3.9 Componentes de SQL*FORMS.	19
2.3.10 Términos de SQL*FORMS y Base de Datos.	20
2.4 Introducción al lenguaje C.	24
2.4.1 Variables y Constantes.	25
2.4.2 Operadores.	28
2.4.2.1 Operadores aritméticos.	29
2.4.2.2 Operadores relacionales y lógicos.	29
2.4.3 Control de flujo de datos.	32
2.4.3.1 Proposiciones y bloques (Sentencias Primitivas).	32
2.4.3.2 Loops.	32
2.4.3.3 Sentencias de opción multiple.	34
2.4.4 Estructura de las funciones.	37
2.4.4.1 Recursividad.	39
2.4.4.2 Llamada de funciones.	39
2.4.5 Apuntadores y arreglos.	40
2.4.5.1. Arreglos de apuntadores.	41

Página

2.4.6 Estructuras. 43
2.4.7 Archivos entrada y salidas. 44
2.5 Introducción a shells. 45
2.5.1 Uso de los comando del shell. 46
2.5.2 Secuencia de procesamiento. 46
2.5.3 La utilización de pipes. 47
2.5.4 Programación básica en shell. 47
2.5.5 Parámetros. 48
2.5.5.1 Empleo de parámetros en programas shell. 49
2.5.5.2 Substitución de parámetros. 49
2.5.5.3 Parámetros posicionales. 50
2.5.6 Programación avanzada. 52
3 AMBIENTE DE DESARROLLO.
3.1 Introducción. 55
3.2 Manejador de la Base de Datos Relacional Oracle. 55
3.2.1 Arquitectura de Oracle. 59
3.2.2 Segmentos de Rollback. 63
3.2.3 Redo Log. 63
3.2.4 El proceso DBWR. 64
3.2.5 Transacción con los log's. 64
3.2.6 El proceso LGWR. 66
3.2.7 Qué es un checkpoint y cómo se utiliza. 67
3.2.8 El proceso de archivar (ARCH). 68
3.2.9 Los procesos PMON y SMON. 69
3.3 Estructuras lógicas y físicas. 71
3.3.1 La herramienta SQL*DBA. 72
3.4 Creación de la estructura del sistema en ORACLE. 72
3.5 Introducción al sistema operativo HP-UX (UNIX). 81
3.5.1 Herramientas de programación. 81
3.5.2 Librerías gráficas. 81
3.5.3 Manejador de Base de Datos. 81
3.5.4 Ambiente de aplicación de utilerías. 81
3.5.5 Estandar de HP-UX. 82
3.6 El shell. 84
3.6.1 Utilerías. 84
3.7 Los file system. 85
3.7.1 Directorios. 87
3.7.2 Nombre de las trayectorias (Pathnames). 88
3.8 Las redes de trabajo de HP-UX (Networking). 90
3.8.1 Enlace. 91
3.9 Plataforma de desarrollo (Hardware). 91
3.9.1 Modelo de discos 670 FL y 1.3 FL. 91
3.9.2 Dos Arquitecturas de hardware HP-PB y CIO. 92
4 DESARROLLO DE MENUS.
4.1 Introducción. 95
4.2 Programas y pantallas de menús. 96

Página

5 DESARROLLO DE PANTALLAS.
5.1 Introducción. 142
5.2 Pantallas de Captura. 144
5.3 Pantallas de Consultas. 147
5.4 Pantallas de Monitoreo. 151
6 DESARROLLO DE REPORTES.
6.1 Introducción. 155
6.2 Programas y Salida de los Reportes. 155
6.3 Programa de Actualización de Saldos. 175
6.4 Programa para Respaldar Saldos. 185
6.5 Programa utilizado como librería. 188
7 APLICACION EN FORMA INTERACTIVA.
7.1 Introducción. 191
7.2 Espacio Utilizado por la Aplicación. 191
7.3 Trayectoria de los programas. 192
7.4 Shells utilizados para lanzar otros programas. 195
7.5 Ambiente para su ejecución interactiva. 199
7.6 Ejecución de la aplicación en forma interactiva. 201
CONCLUSIONES.
BIBLIOGRAFIA.

INTRODUCCION

Los cambios que en la actualidad se presentan dentro del país y por consiguiente en la estructura empresarial de este, nos conduce a que la dinámica de las empresas tengan un enfoque diferente, principalmente en las empresas paraestatales y de las cuales depende en gran medida la estructura económica de nuestro país. Como es la industria petrolera.

Petróleos Mexicanos sufre cambios muy importantes en su organigrama y por lo tanto en todo lo que se refiere a la administración financiera que actualmente es uno de los factores que mayor importancia toma dentro de Petróleos Mexicanos.

Los normas y reglamentaciones para la relación que Petróleos Mexicanos presenta al dividirse en cuatro empresas (Pemex Exploración y Producción; Pemex Gas y Petroquímica Secundaria; Pemex Refinación; Pemex Gas y Petroquímica Básica) hace que la administración sea mucho más compleja. Esta división busca que cada una de las empresas tengan un mejor aprovechamiento de los recursos, esto con la finalidad de tener una mayor productividad y por consiguiente ser más rentable para el país.

De las cuatro empresas de PEMEX solo haremos referencia a Pemex Gas y Petroquímica Básica, específicamente en la manera en que se puede tener un mayor control del presupuesto en el centro de trabajo (Subgerencia de Operación de Ductos y Terminales de Gas Natural y Azufre). Para lo cual se desarrolló un sistema en el que nos apoyaremos para tener un mejor aprovechamiento de la asignación del presupuesto y control de los gastos que se realicen dentro de este Centro de Trabajo.

Anteriormente el presupuesto se controlaba con diferentes Departamentos para cada uno de los Centros de Trabajo que existían, pero con la separación de las empresas se crea una nueva estructura, la cual a su vez se ve afectada en todo su organigrama teniendo este una mayor diversificación en todos sus niveles.

A nivel central se sigue controlando como un solo Centro de Trabajo con tres Departamentos y es de esta forma como es suministrado el presupuesto a nuestro Centro de Trabajo. La realidad es que el Centro de Trabajo esta compuestos por 12 Sectores y varios departamentos por lo que para la ministración de fondos de cada uno de estos es necesario un control. La problemática era: cómo manejar y administrar sus gastos así, como la asignación presupuestal a cada uno de estos, por este motivo nace el Sistema de Estados de Resultados Financieros (S.E.R.F.).

Los programas de los que esta compuesto el S.E.R.F., la plataforma en la que fue desarrollada la funcionalidad de la aplicación y la presentación de esta en forma interactiva en la parte central del tema de tesis, y que se describirá a lo largo de esta.

CONSECUENCIAS DE LA SEPARACION DE PEMEX

I.I Introducción

El presente capítulo tiene como objetivo principal dar a conocer a todas las áreas usuarias de los cambios presentados en los sistemas informáticos y administrativos, tanto del organismo de Petróleos Mexicanos como a sus empresas subsidiarias, el decreto que crea una nueva estructura de PEMEX en nuestro país, la reglamentación interna para el manejo de cambio y convivencia de las nuevas cuatro empresas en las que Petróleos Mexicanos se divide (Pemex Exploración y Producción; Pemex Gas y Petroquímica Secundaria; Pemex Refinación; y Pemex Gas y Petroquímica Básica), principios generales para su aplicación, premisas para su manejo y estrategias para su instrumentación a través de los diferentes sistemas de información en PEMEX. Todo esto conjuntado, nos lleva a tener un control más preciso de los gastos que en cada una de las 4 empresas se generen.

Debido a la diversificación de PEMEX y en particular a la del Centro de Trabajo (Subgerencia de Ductos y Terminales de Gas Natural y Azufre) se vio la necesidad de desarrollar un sistema para controlar a cada uno de los Departamentos y Sectores con los que cuenta el Centro de Trabajo esto con la finalidad de conocer las erogaciones de cada uno de estos y ver su rentabilidad que para el Departamento de Presupuesto es la única forma de poder controlar la solicitud de fondos y saber la cantidad que se les proporcionará mensualmente a cada uno de estos Departamentos dependiendo de su erogación mensual o presupuesto ejercido, también sirve como parámetro de referencia para la solicitud de fondos mensuales por Sector, para la solicitud de fondos por Centro de Trabajo, se basa en los datos de un sistema institucional. Los sistemas institucionales sólo se mencionarán para saber como se liga el Sistemas de Estados de Resultados Financieros (S.E.R.F.) con estos y cual es la importancia de los mismos para nuestro sistema.

De acuerdo al decreto presidencial, en donde se determina la separación de PEMEX en cuatro empresas. Se planteó la elaboración de los planes, lineamientos, procedimientos, cambios a los Sistemas de la Institución y estrategias de implantación. Se solicito la participación de las diversas áreas normativas de Petróleos Mexicanos tales como Contabilidad, Presupuestos, Proveeduría y Almacenes, Recursos Humanos y Tesorería, con quien se definieron de acuerdo a lo que marca el decreto, la interpretación e instrumentación del mismo dentro del organismo y sus empresas subsidiarias.

Derivado de los trabajos realizados en sesiones multidisciplinarias del grupo que se conformó, se determino el impacto que se tendrá respecto a los procedimientos y normatividad en materia

monetaria, las concertaciones con dependencias externas para el intercambio de información financiera, y los cambios que serán necesarios realizar en todos los sistemas informáticos (institucionales, especificados en rama y en pc's).

Los trabajos efectuados por el grupo, determinan las acciones y responsabilidades que tendrán tanto las áreas normativas del corporativo como la preparación operativa que le corresponda aplicar a cada una de las empresas de Petróleos Mexicanos en los diferentes Centros de Trabajo, tanto foráneos como a nivel central.

1.2 La separación de Pemex Gas y Petroquímica Básica

La creación de una nueva empresa llamada Pemex Gas y Petroquímica Básica crea un cambio radical en su estructura general que por tal motivo sufre una reorganización que a la fecha aun no termina, lo que por consecuencia trae una falta de estructura definitiva de muchos Centros de Trabajo entre los cuales se encuentra el nuestro. Esto provoca que en el Centro de Trabajo exista a nivel central como solo un Departamento (Subgerencia de Ductos y Terminales Zona Centro) y no como varios departamentos como son Presupuestos, Tesorería, Contaduría, Informática, etc., y un conjunto de sectores que lo componen como son Salamanca, Poza Rica, Madero etc. los cuales a su vez también tienen un conjunto de departamentos, por todo esto se vio la necesidad de desarrollar un sistema que controlara el gasto de cada Sector y sus Departamentos para poder administrar los fondos necesarios y controlar sus gastos.

Cabe señalar que a partir de la evaluación realizada de todo lo anterior se definió la complejidad que esto representa para el Centro de Trabajo y la dificultad para controlar la información por lo que conjuntamente con Contaduría, Presupuestos, Tesorería, Recursos Humanos e Informática se definieron las normas y políticas a seguir para el control de las erogaciones por Departamento y Sector, a su vez representar estas en forma global para entregar información de los gasto a nivel central como un solo Centro de Trabajo Departamento de manera institucional.

La políticas y normas para solucionar la problemática se definirán de forma breve ya que no es el tema de esta tesis aunque son importantes para entender la manera en que opera el Sistema y su funcionalidad además de la dependencia con otro Sistemas.

1) Para PEMEX cada Factura o documento contable tiene una clave de autorización la cual esta conformada de 10 dígitos y una literal un ejemplo.

F2444111240

El primero es la literal contando de izquierda a derecha es una literal que indica de acuerdo a la siguiente tabla el mes.

A	Enero	G	Julio
B	Febrero	H	Agosto
C	Marzo	J	Septiembre
D	Abril	K	Octubre
E	Mayo	L	Noviembre
F	Junio	M	Diciembre

Los siguientes dos campos representan el día en que se autoriza la operación del documento en este caso 24

El tercer dígito es el último dígito del año, para este como ejemplo es 4, que es el último dígito de 1994

Los siguientes tres campos es el número de la unidad de control responsable de la operación que para Gas y Petroquímica Básica es 411.

Los últimos cuatro campos son numéricos y es un consecutivo que le asigna el Centro de Trabajo y que es la base de nuestro sistema, es por medio de estos 4 dígitos que al conformar un rango determinado este le asigna un Departamento el cual no existe en los catálogos institucionales y con el cual podemos ubicar a cada uno de los Departamentos aquí existentes.

Nuestro segundo lineamiento es separar los gastos que corresponden al Centro de Trabajo de los gastos de la zona sur, a la cual se le esta dando apoyo, esto se debe a que es un Centro de Trabajo de nueva creación por lo que no cuenta con el personal capacitado.

Basándonos en las claves de autorización el séptimo dígito representa cada una de las zonas si es un 2 representa la zona sur y si es un 4 a la zona centro, que somos nosotros.

Por último el sistema se alimenta de lo capturado en los sistemas institucionales esto con la finalidad de evitar la doble captura y solo se anexa un campo para nuestro sistema que es el departamento económico el cual como se verá es la base del Sistema de Estados de Resultados Financieros SERF.

1.3 Conceptos para el manejo del presupuesto

El Sistema denominado Sistema de Estados de Resultados Financieros (SERF) tiene como principal objetivo controlar los gastos realizados por el Centro de Trabajo y acumular sus saldos mes por mes durante los 12 meses del año lo que nos sirve para comparar si hubo saldo a favor o en contra durante todo el año, nos muestra también donde fue realizado el gasto, quien lo realiza y en que fue utilizado el dinero, además los reportes entregan comparativos de las erogaciones por Departamento contra las ministraciones o solicitud de fondos que mes con mes se le asigna a cada Departamento, esto nos da como resultado los remanentes por departamento con lo que se podrá saber a cual departamento se le asigna un mayor presupuesto o disminuye según sea el caso y en que renglón del gasto, así se podrá controlar mejor la distribución del presupuesto.

Para PEMEX en general los gastos los controlan de la siguiente forma: existen lo que se conoce como concepto de origen que es la representación mínima de la realización de un gasto en PEMEX, el cual es numérico de 6 y que como su nombre lo indica este nos dice en que se realizó el gasto, si fue mano de obra, compra de un material, pago a un proveedor etc., y una descripción específica del gasto un ejemplo es el siguiente.

10-00-00	Planta sindicalizado salario tabulado.
27-00-00	Transitorio confianza salario tabulado
36-01-00	Fletes y arrastres ferrocarril pagados a terceros

Para nuestro caso solo nos interesa saber como opera esto, y no como se estructuró debido a que este concepto es manejado por PEMEX desde su inicio y el cual es normado a nivel central por el corporativo del área de Presupuestos y Finanzas.

Además del concepto de origen en PEMEX se maneja otro tipo de agrupación del gasto y este es el renglón del gasto que ya fue mencionado con anterioridad pero del cual no sea definido su formato. El renglón del gasto como en el caso del concepto de origen es institucional eso quiere decir que en todo

CONSECUENCIAS DE LA SEPARACION DE PEMEX

PEMEX se opera de la misma forma, el renglón del gasto esta compuesto de 3 dígitos, estos contienen una agrupación de los concepto de origen dependiendo de la relación que estos tengan entre sí para tener una idea de la diferencia entre la cantidad de concepto origen que se maneja y renglones del gasto el primero es un total 2324 y del segundo 110 como se puede observar esta es una agrupación más general de nuestras erogaciones, un ejemplo.

Renglón del Gasto	Concepto de Origen
	10-00-00
201	12-00-00
	27-00-00

201 Sueldos, Salarios y Prestaciones Normales y Extraordinarias.

Esto nos ayuda a tener la información agrupada y consultarla de una manera más rápida, por ejemplo, para conocer el monto del pago de salarios en esta catorcena no se necesita sumar cada uno de los conceptos, sólo se revisa cuando es el total del renglón del gasto y listo. Teniendo en cuenta que además el detalle del gasto se conserva por si existe algo que no concuerde o para verificar si fuese necesario.

De esto se desprende la agrupación por renglón del gasto y Departamento, con lo que tendremos un control de cada gasto realizado por Departamento, en qué renglón y porqué concepto, la información a nivel de Departamento es la parte que más importa para el Departamento de Presupuesto y por tanto para el sistema, esto por lo que ya se dijo, que el Departamento solo existe a nivel local y no en forma institucional.

Existen otras agrupaciones que a nivel presupuestal son de interés y que tampoco existen a nivel institucional que son los Sectores. Los sectores son dependencias foráneas del Centro de Trabajo que operan y dependen presupuestal y administrativamente de la Subgerencia de Venta de Carpio. Se cuenta con un total de 12 Sectores, los cuales son:

Veracruz
Cárdenas
Minatitlán
Ciudad Mendoza
Poza Rica
Ciudad Madero
Venta de Carpio
Apizaco
Valle de México
Salamanca
Guadalajara
San Juan Ixhuastepec
Abasolo

Además se maquila información a otras dependencias, de las cuales es importante saber su gastos para determinar cuánto nos esta costando estar dándole mantenimiento y poder cobrar el servicio que se les esta proporcionando a cada dependencia, la complejidad de PEMEX es muy grande, por lo que, controlarlo de forma manual es difícil esto hace la necesidad de la creación de un sistema que controle todo lo anterior, así nace el SERF (ya mencionado), el sistema entrega los reportes de las erogaciones por Departamento, Sector, por subgerencia, por dependencia y un global por todo el Centro de

Trabajo; Nos muestra la solicitud de fondos en forma automática de lo ministrado en el Centro de Trabajo de manera global y desglosada por Sector.

De todo lo anterior al corporativo solo le interesa las erogaciones en forma general del Centro de Trabajo por lo que lo anterior se consolida en reporte a nivel institucional, los cuales son comparados con el Sistema Institucional del Control del Ejercicio Presupuestal (SICEP).

El diagrama siguiente muestra la estructura del Centro de Trabajo y que debido a esto se vio en la necesidad de desarrollar el sistema SERF el cual es el tema de este libro y mi tema de tesis.

Venta de Carpio → Subsidiarias
↓
Sectores
↓
Departamentos
↓
Renglón del Gasto → Concepto de Origen

El SERF es un sistema que obtiene la información de las siguientes dos formas: la primera, el sistema extrae información del Sistema de Contabilidad el momento contable 14 (fase del devengado), previamente capturado y la segunda forma de obtener información es la captura que realiza el departamento de presupuestos de la ministración de fondos, la solicitud y el pasivo.

Como se han venido explicando algunos términos de la forma en que contablemente opera PEMEX para entender más el manejo y operación del sistema cabe aclarar que, PEMEX utiliza para agrupar en contabilidad cuentas de mayor como en toda contabilidad pero para presupuestos esta cuentas de mayor la representa por momentos contables por ejemplo.

Cuenta de Mayor	Momento Contable
7314	14
7315	15
7316	16

Para presupuestos la cuenta de mayor 7314 es momento contable 14 que es nuestro devengado, para contaduría existen la cuenta de mayor 7316 y para presupuestos el momento contable 16 que es el pago o flujo de efectivo así sucede con muchas cuentas, no con todas.

Para PEMEX hablar de devengado es todo aquello que ya tiene una factura pero que aún no se ha pagado. Esta puede pagarse a los 7 días o al mes dependiendo del proveedor o contratista, este devengado es lo que nos interesa saber y que muestra el sistema para tomar una decisión y saber si lo que se tiene es suficiente o hay que solicitar una ampliación.

Brevemente se explicará como se encuentra la información en el Sistema Institucional de Contabilidad (SIC) y como es utilizada por el sistema (SERF). Para la contaduría solo existen tres Departamentos 12246, 12247 y 12331 y dos Centros de Trabajo 475 y 476 zona sur y centro respectivamente. Dentro de los catálogos institucionales del (SIC), como se describe a continuación:

Centro	Departamento	Descripción del Departamento
475	12246	Subgerencia de Ductos y Terminales Zona Sur
476	12247	Subgerencia de Ductos y Terminales Zona Centro
476	12231	Subgerencia de Gas L.P.G

CONSECUENCIAS DE LA SEPARACION DE PEMEX

Como se ha podido observar este es el problema al que nos enfrentamos, cómo integrar la información no solo los gastos de cada zona sino de cada Sector y cada Departamento que de estas dependen. La zona sur es una Gerencia de nueva creación, su estructura está en proyecto para que a principios del siguiente año se aplique su autorización por lo que su contabilidad y presupuesto se maneja en Venta de Carpio hasta que esto no suceda.

La separación de los gastos por cada una de las zonas no es tan complicado como la separación de los gastos de los Sectores por cada uno de sus Departamentos y conocer en que realizo el gasto, la separación se realiza con las últimas cuatro cifras de la clave de autorización como ya se mencionó.

La clave de autorización es la que identifica a cada factura o documento contable que produce un gasto.

La asignación de un rango en las últimas cuatro cifras representa para nuestro sistema un Departamento en particular este le es asignado en base a un catálogo que conjuntamente con los Departamentos de Presupuestos, Contabilidad, Recursos Humanos e Informática se elaboró para el desarrollo del sistema.

Informáticamente se crearón 4 catálogos uno para identificar el Departamento local, otro para los sectores que se manejan en el Centro de Trabajo, uno más para las dos zonas y el último para la asignación de los rangos, un ejemplo de como se le asigna por medio de la clave de autorización un Departamento utilizando un rango con las últimas cuatro cifras se muestra a continuación :

Clave de Autorización	Rango	Departamento Institucional	Departamento Local ó Económico
D0844114807	4801-4825	12247	49000
D1144110576	0576-0600	12247	46000
D0844114826	4826-4950	12247	48200

Departamento	Descripción
49000	Informática
46000	Contaduría Venta de Carpio
48200	Sistemas y Costos

La descripción de los catálogos, las especificaciones técnicas del sistema, la estructura de las tablas y la forma en que opera este se verán con mayor detalle durante el desarrollo de la tesis por lo que esto sólo es una descripción general de la operación del sistema y las consecuencias que se dieron. Los cuales originaron la necesidad de desarrollar el Sistema de Estados de Resultados Financieros (SERF).

Se describieron algunos aspectos contables que nos sirven de base para saber en una forma global como opera PEMEX, sus registros contables, que son base importante para el desarrollo de pantallas de captura y generación de reportes, que sean de utilidad para el usuario final.

Para terminar con nuestro panorama general diremos que el sistema como se ha podido ver, no captura información de las erogaciones realizadas en el Centro de Trabajo, estas son extraídas de el sistema institucional de contabilidad (para evitar la doble captura), a esta información tomada de otro sistema se le asigna el Departamento local en base a su clave de autorización, para entregar al usuario final la información desglosada y este la pueda consultar en pantallas y reportes, para que, apoyados en estos realice la presentación de sus gastos en forma separada, de esta manera puede controlar mejor los gastos realizados por cada uno de los departamentos y sectores, asignándoles a cada uno un presupuesto más real, basado en los gastos que estos realizan.

A manera de resumen, el SERF es un sistema basado en otros, para controlar mejor los gastos y separar mejor la información del presupuesto, así como una presentación a nivel gerencial verídica y bien fundamentada

UTILERIAS UTILIZADAS PARA LA APLICACION

2.1 Introducción

El desarrollo de este capítulo proporciona una breve descripción de las utilerías utilizadas para el desarrollo del sistema, aunque, También se hablará de la Base de Datos ORACLE y del sistema operativo **UNIX**. Una explicación más amplia de estos dos se llevará a cabo en el capítulo III. Este capítulo cubre los términos, conceptos y estructuras principales de las utilerías empleadas para el desarrollo del sistema, el capítulo sirve como guía de muchos tópicos relacionados con ORACLE y **UNIX**.

La primer parte describe la estructura completa de un programa en Pro*C y presenta la sintaxis de las instrucciones de Pro*C, además cubre los términos, y conceptos relacionados con SQL*forms, todo esto basado en la Base de Datos relacional ORACLE. Para nuestra segunda parte mencionaremos al Lenguaje "C", su estructura, la utilización de las variables, sus palabras reservadas y algunos otros conceptos, junto con este hablaremos de la programación con **shell**, que son comandos del sistema operativo ejecutados en forma interactiva, qué es un **shell** y su aplicación, todo esto como utilerías de lo que es el sistema operativo **UNIX**.

Comenzaremos con el precompilador Pro*C y SQL*forms como parte primera de nuestro capítulo.

2.2 El precompilador Pro*C

2.2.1 Introducción al precompilador Pro*C

Este capítulo es una introducción general a Pro*C. Este describe como puedes usar conceptos generales y definiciones, partiendo de un programa en Pro*C, y tipos de declaraciones basadas en un programa desarrollado en Pro*C.

El lenguaje de datos SQL (Struct Query Lenguaje) es un lenguaje no procedural, por lo que la mayoría de sus instrucciones se ejecutan independientemente de su precedencia, o de la instrucción siguiente. Comparando este lenguaje de programación contra lenguajes semejantes, como "C", Fortran, Cobol, ó PL/1. Estos lenguajes llamados procedurales, son basados sobre construcciones semejantes como "loops", "branches" e "if else then".

Como se podrá observar conforme avancemos el lenguaje SQL es un lenguaje muy poderoso, no obstante este tiene alguna limitación, y no contiene la capacidad del procedural.

Teniendo identificado específicamente a SQL como un lenguaje no procedural y de esta manera entendendiendo las limitaciones de lenguajes semejantes, los creadores de SQL también diseñan explícitamente a SQL como constructor. Esto es, realizar programas en lenguaje procedural, semejante a "C" en sus construcciones, los programas pueden diseñar aplicaciones, que combinen las mejores características de SQL y las mejores características del lenguaje de programación procedural (El lenguaje del host). Estas aplicaciones son más poderosas y flexibles que las aplicaciones basadas sólo sobre "C" o SQL.

2.2.2 La Base de Datos Relacional ORACLE (RDBMS ORALE)

El administrador de la Base de Datos ORACLE incluye algunas herramientas que permiten a los programadores escribir programas en un lenguaje de host para accesar datos en una Base de Datos ORACLE.Muy comúnmente algunas herramientas son provistas por los lenguajes de programación como "C", Fortran, Cobol, Pascal y PL/1.

2.2.3 Características y beneficios del precompilador Pro*C

La herramienta de Pro*C viene provista con el Sistema Manejador de la Base de Datos Relacional ORALE (ORACLE RDBMS). La cual esta diseñada para convertir un programa en "C" que incluye instrucciones de SQL y poder accesar y manipular datos en la Base de Datos ORACLE.

Como precompilador Pro*C convierte las instrucciones encontradas de EXEC SQL, en el archivo de entrada para poder realizar llamadas apropiadas a ORACLE. con el archivo de salida. El archivo de salida puede subsecuentemente estar compilado, conectado, y ejecutado de la manera normal por un programa en "C".

Comparar Pro*C (o productos similares como son Pro*fortran, Pro*Cobol y Pro*PL/1) con la llamada ORACLE Call Interface (OCI) (previamente llamada Pro*SQL). Que es la Interface para llamar a la Base de Datos ORACLE, por todos aquellos usuario que realizan llamadas a ORACLE directamente desde un lenguaje de alto nivel semejante al C, FORTRAN, o COBOL.

Algunas transacciones son acompañadas por múltiples llamadas realizadas por medio de un cursor. Estas son algunas características de el precompilador Pro*C.

1) Las llamadas a Pro*C son más conceptuales, y más sencillas de entender que las llamadas a Pro*SQL (OCI).
2) Una llamada a Pro*C (OCI) es trasladada automáticamente a el equivalente de algunas llamadas a librerías "run-time", reduciendo el tiempo de programación.
3) Un programa puede ser usado varias veces en diferentes Base de Datos.
4) Múltiples programas pueden estar separadamente precompilados y ejecutarse juntos.

2.2.4 Conceptos generales.

Usando las herramientas de Pro*C, adiciona algunos otros pasos dentro del programa normal durante el proceso de programación; de esta manera estos pasos adicionales causan que la herramienta de

Pro*C, haga una justa suma de trabajo en favor del programador. La secuencia normal de eventos en escribiendo y corriendo un programa en "C" se muestra en la figura 2.1.

1.- Escribe un programa en "C"
2.- Compila el programa, obtiene una salida de un archivo objeto
3.- (Link-edit) El archivo objeto, obteniendo un archivo ejecutable
4.- Corre el programa, ejecutando el trabajo deseado

Figura 2.1 Pasos para escribir un programa en Lenguaje C.

El programador debe de incluir instrucciones en Pro*C en el programa original (de **lenguaje C**), como se observa sólo se adiciona un paso al principio, de la figura 2.1 como se muestra en la figura 2.2.

1.- Escribe un programa en Pro*C
2.- Precompila el programa usando Pro*C obteniendo una salida del archivo
3.- Compilar el programa de salida, obteniendo como resultado un programa objeto.
4.- (Link-edit) Conectándose y editando el archivo objeto obteniendo un archivo ejecutable.
5.- Correr el programa, ejecutando el programa deseado

Figura 2.2.- Pasos en la Estructura de un programa en Pro*C

2.2.5 Mezclando comandos de C con instrucciones de SQL.

La sintaxis y opciones para precompilar un programa en Pro*C son múltiples, por lo que sólo mencionaremos al comando que realiza esta función. El comando "cmpc" que es un **shell,** que tiene integrado el comando pcc y sus utilerias para generar el archivo de salida en "C" y obtener el programa objeto integrando aquí las librerías del lenguaje "C" y SQL, ejemplo:

$cmpc nombre.pc. La terminación de los programas en Pro*C es *.pc

Algunas instrucciones validas de SQL pueden ser ejecutadas desde un programa en "C". Mientras estos son algunos "pats" requeridos o instrucciones de un programa en Pro*C, y en un orden básico de apariencias, notar que las líneas de programación en "C" pueden aparecer en algún punto del programa (Siguiendo los estándares de la programación en "C", de cursores), Los pasos requeridos son introducidas en "Partes del programa en "C"

2.2.6 El comando EXEC SQL y EXEC ORACLE como prefijo

Para minimizar las dificultades que podrían causar por interpretación las instrucciones de SQL dentro de diferentes lenguajes de programación, todas las instrucciones de SQL son prefijadas con las palabras EXEC SQL.

Un propósito del precompilador consiste en trasladar todas las instrucciones que comienzan con EXEC SQL dentro de códigos apropiado del lenguaje fuente, para llamar a la Base de Datos (Asumiendo que esta en lenguaje "C").

Mientras unas instrucciones son prefijadas por la palabra EXEC SQL, algunas otras instrucciones son prefijadas por la palabra EXEC ORACLE, mientras un gran número de instrucciones de SQL requieren el prefijo EXEC SQL, hay un número menor de comandos que requieren del prefijo EXEC ORACLE. Que es utilizada junto con algunas opciones para mostrar si existe algún error y su número de error dentro de ORACLE.

Estas instrucciones no son compatibles con SQL, y son únicamente para el precompilador ORACLE.

2.2.7 Ejecución y declaración de las instrucciones de SQL

Las instrucciones de SQL que son incluidas en un programa en Pro*C, caen dentro de uno de estos dos grupos ejecutables ó declarativos, todas las sentencias ejecutables son declaradas con el prefijo de la palabra EXEC SQL.

Las instrucciones ejecutables de SQL son instrucciones que generalmente realizan llamadas hacia la Base de Datos, ellas son incluidas, pero no limitadas a:

(DML) Lenguaje de Manipulación de Datos.
(DDL) Lenguaje de Definición de Datos
(DCL) Lenguaje de Control de Datos.

Después de que una instrucción ejecutable de SQLCAL (SQL Communications Area), la cual contiene un conjunto de código de regreso, se ha ejecutado, esta regresa.

Una unidad lógica de trabajo es iniciada con la ejecución de la primera instrucción ejecutable de SQL, excepto para CONNECT en donde la primera instrucción ejecutable de SQL encontrada después de un CONNECT, COMMIT ó ROLLBACK WORK, inician con una nueva unidad lógica de trabajo.

Las instrucciones declarativas no generan código y no tienen efecto en unidades lógicas de trabajo. El SQLCAL no es afectado por las instrucciones declarativas. Las instrucciones declarativas SQL son mostradas en la figura 2.3.

SQL Instrucciones Declarativas
BEGIN DECLARE SECTION
END DECLARE SECTION
WHENEVER ...
DECLARE CURSOR ...
INCLUDE ...

Figura 2.3 Instrucciones Declarativas

2.2.8 Partes de un programa en PRO*C

Un programa en Pro*C contiene dos partes, ambas son requeridas por el procesador del precompilador Pro*C.

La aplicación prólogo.
La aplicación body.

La aplicación prólogo define las variables y realiza la preparación general del programa en Pro*C, la aplicación body, básicamente contiene las llamadas que se realizan en Pro*C a la Base de Datos, incluyendo instrucciones de SQL semejantes como INSERT, DELETE, ó UPDATE, para la manipulación de los datos en la Base de Datos ORACLE. El código del lenguaje "C" se ubica en las secciones que el código del procesador Pro*C requiere.

Básicamente este es un panorama general de las instrucciones y relaciones que existen entre el lenguaje "C" y SQL en relación al precompilador Pro*C que realiza la interface entre "C" y Pro*C. También conceptualizamos la estructura que presenta un programa en Pro*C y su principales comandos.

2.3 Introducción a la herramienta SQL*FORMS.

La herramienta SQL*forms es de propósito general para ejecutar y desarrollar formas basadas en aplicaciones interactivas. Los componentes de esta herramienta, son diseñados especialmente para desarrolladores de aplicaciones, programadores y analistas. Esta herramienta nos permite llevar a cabo tareas como las siguientes.

Definir transacciones que combinen datos de múltiples tablas, en forma individual.

Crear rápidamente formas por default usando una ubicación predefinida, incluyendo relaciones de forma automática de tipo maestro-detalle de una forma personalizada de manera que uno puede definir todos los aspectos deseados en una aplicación.

SQL*forms emplea el estándar de la corporación ORACLE fill-in-the-form, empleando esta interface para enriquecer el desarrollo de la productividad y reducir el tiempo de aprendizaje.

Los componentes de ejecución de SQL*forms permiten ejecutar o correr las aplicaciones diseñadas por el programador.

Este parte del capítulo muestra los conceptos y términos básicos de SQL*forms por lo que tenemos que estar relacionados con los conceptos de las Bases de Datos Relaciónales (ORACLE), para su mejor comprensión.

Dentro de este enfoque de SQL*forms hablaremos en forma general de los siguientes tópicos.

Desarrollo de aplicaciones.
Objetos de SQL*forms.
Los componentes de SQL*forms.
Modelo de procesamiento de SQL*forms.
SQL*forms y términos de la Base de Datos.

2.3.1 Desarrollo de aplicaciones

En la pasada década la tecnología empleada para construir aplicaciones era lenta, poco tiempo después la evolución comenzó a desarrollar un complejo conjunto de ensambladores con los que se desplazan a algunos lenguajes de tercera generación (3GL) tales como Fortran y Cobol. Para la Cuarta generación de lenguajes (4GL) la tendencia de la evolución ha sido para proporcionar más y mejor funcionalidad

de una manera no-procedural, los desarrollos de estas aplicaciones especifican como se deben de realizar estas.

La herramienta SQL*forms promueve la tendencia, proporcionándonos una gran cantidad de valores funcionales por default, en completas aplicaciones de trabajo. SQL*forms nos proporciona una completa estructura de aplicaciones y una estructura de funcionalidad para todos los objetos que componen a una aplicación, dando como resultado que la creación de la más sencilla forma de SQL*forms y aún la más simple aplicación tiene ciertos "detalles funcionales", como pueden ser la validación de tipos de datos, la exploración a través de una forma, el acceso a la Base de Datos, etc.

Una vez que se ha creado una aplicación, puedes modificar varias de sus características funcionales y la presentación o apariencia de esta. La interface para el usuarios "fill-in-then-form" te permite definir las especificaciones de una aplicación, por medio de la descripción de información dentro de una área del texto en una pantalla o por "turning off" apoyándonos en catálogos definidos en una lista.

Por ejemplo, asumimos que tu aplicación define una área, o campo, la pantalla de la aplicación nos muestra datos de una columna de la Base de Datos, que permite se realice la consulta de ese dato. Si se desea modificar la definición de ese campo de modo que un operador no pueda alterar los datos mostrados simplemente se desactiva la entrada a caracteres permitidos para ese elemento. Como se observa en el ejemplo se pueden llevar a cabo modificaciones sin tener que programar.

Además de la interface "fill-in-then-form", SQL*forms proporciona un poderoso diseñador de pantallas para editar imágenes de una pantalla y agregar textos, promts y elementos gráficos.

Si una aplicación requiere características especiales o aplicaciones lógicas se pueden agregar comandos usando PL*SQL, que en la mayoría de las aplicaciones desarrolladas es necesario utilizarlo, la integración de un lenguaje procedural dentro de la interface, provienen de llamadas de un lenguaje de programación de tercera generación.

Esto nos muestra la funcionalidad del default de la interface del usuario "fill-in-then-form" llevando a cabo ganancias significativas de productividad.

Además de la facilidad para crear rápidamente poderosos aplicaciones, SQL*forms proporciona una aplicación de desarrollo de ambiente portátil. En el pasado la elaboración de aplicaciones para poder ser ejecutadas o corridas en plataformas o ambientes diferentes no era posible, se tenían que escribir varias aplicaciones diferentes una para cada plataforma. Las aplicaciones de SQL*forms son portátiles para diferentes softwares, sistemas operativos, hardware y sistemas en general.

2.3.2 Objetos de SQL*FORMS

Las aplicaciones de SQL*forms son realizadas con objetos y por objetos. Estos objetos contienen toda la información que se necesita para manipular y construir aplicaciones en SQL*forms.

El primer objeto se compone de una aplicación de SQL*forms, que es la forma. Este objeto puede incluir una aplicación completa, o pude conectarse con otra forma, o grupo de formas, menús, reportes y otros componentes que forman una aplicación más compleja.

Dentro de SQL*forms se pueden crear formas y dentro de ellas algunas conexiones con otros componentes.

Una forma es realizada con objetos adicionales. Estos objetos conectan a la forma con elementos de la Base de Datos, semejantes entre si, como columnas y tablas que nos proporcionan un control de la ejecución al desarrollarse.

Blocks Describe cada sección o subsección de la forma, y sirve como un default de la tabla base dentro de nuestra Base de Datos
Fields Representa columnas datos entre áreas, y describe como los datos deben ser desplegados y validados, y como un operador debe interactuar con los datos, mientras estos se insertan.
Pages Es una colección de información desplegada, semejante a textos constantes y gráficas. Todos los campos de la aplicación son desplegados en alguna página.
Trigger Es un conjunto de comandos de procesamiento asociados con la ocurrencia de puntos, parecidos a una tecla de función en particular precedida por el operador
Form-Level Es un conjunto de comandos de procesamiento que pueden ser llamados por algún **trigger** o un procedimiento (from-level), a nivel de forma, y pueden formar argumentos

Usualmente cada forma contiene mínimo un block, una página, y uno ó más campos. Cada objeto de una forma tienen un conjunto de atributos o características, las cuales proporcionan información a cerca del objeto. Los campos son llamados por una instancia. Estos pueden hacer referencia a una computadora o la manipulación de la Base de Datos.

Se pueden crear uno ó más objetos por default y usualmente todas las características del default son suficientes. Cuando se modifican las características del default de los objetos usando comandos de procesamiento, se utilizan elementos designados por la interface de SQL*forms.

Cuando creas una aplicación, usualmente creas objetos al crear está. Sin embargo, desde ese momento los objetos son elementos separadamente, además tú puedes fabricar una aplicación copiando objetos existentes dentro de la misma aplicación o por medio de la referencia de objetos existentes en la definición. (Al hacer referencia a la copia, SQL*forms la ejecuta modificando los límites de referencia del objeto). Aquí se muestra porque los objetos son "reusables", puedes crear modelos. Comúnmente utilizan los **trigger**, estos nos dan como resultado la reducción de la forma, reduce las posibilidades de errores, y proporciona un buen camino que garantiza al diseñador un eficiente programación en los **triggers**

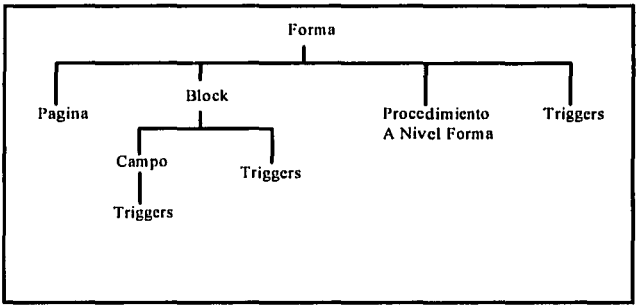
2.3.3 Un enfoque a cerca de objetos

Los inciso previos describen como objetos son la base para construir bloques de una aplicación y se introducen rápidamente objetos como: formas, bloques, campos, páginas, **triggers** y procedimientos a nivel de forma. Esta sección describe objetos de SQL*forms con mayor detalle y explica como se relacionan entre si. Aunque no se describirá a detalle las características de cada uno de estos objetos, ni como se definen por medio de la interfase del diseño.

2.3.4 Jerarquía de objetos

Los objetos en SQL*forms existen en orden jerárquico, La figura 2.4 Ilustra esta jerarquía y muestra los objetos de más alto nivel en la misma, así como sus objetos de más bajo nivel.

Figura 2.4 Jerarquia de Objetos en SQL*forms



La forma es el objeto primario o componente principal de una aplicación de SQL*forms, por consiguiente la forma reside en el más alto nivel. Dentro de una aplicación, un objeto de la forma puede poseer todos los objetos. La excepción son los objetos que se encuentran referenciados para una forma, pero que pertenecen a otra forma.

Mostrar que un objeto puede compartirse con varias y diferentes formas a través de la referencia de objetos (pero con un sólo diseño), bloques individuales pueden poseer campos. **Triggers** y campos individuales pueden poseer **triggers**. La jerarquía de los **trigger** depende de el nivel en el que se define el **trigger** y por lo tanto el alcance que puede tener su acción.

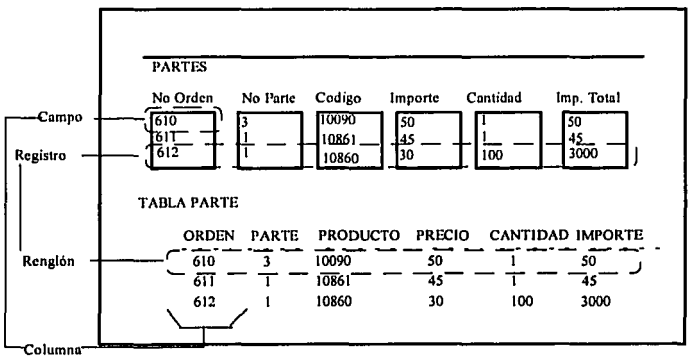
2.3.5 Bloques

Una forma contiene uno ó más bloques. Cada bloque puede estar relacionado directamente con una tabla o vista de la Base de Datos individualmente.

Lo anterior nos indica como se muestra en la figura 2.5 que cada campo del bloque esta asociado con una columna de la tabla (Conocida como tabla base) o vista. Por default esta relación nos permite directamente operar la forma para consultar, actualizar, insertar y borrar datos de la tabla base ó vistas. SQL*forms ejecuta estas operaciones automáticamente cuando una tabla base es especificada, alternamente, un block puede relacionarse indirectamente, o contener información de muchos tablas de la Base de Datos o vistas.

Un bloque muestra o despliega los datos en registros, como se muestra en la figura 2.5, estos registros son agrupados en campos que muestran la descripción de la información, la cual corresponde a renglones de alguna tabla.

Figura 2.5 Registros, Renglones, Campos y Columnas



Los bloques tienen características que tú puedes definir. Estas características determinan atributos, semejantes a recuperar o traer información de SQL*forms, dentro de un bloque al realizar una consulta, el número de registro de los datos que están desplegados en un bloque, son validados por estos atributos.

2.3.6 Relación entre bloques

Puedes relacionar unos bloques con otros a través de la relación maestro-detalle. Una relación maestro-detalle corresponde a una relación entre una llave primaria y una llave foránea entre tablas base de los bloques, SQL*forms, permite que automáticamente puedas establecer una relación maestro-detalle cuando creas los bloques, esta característica permite crear una forma que muestra todos los renglones dentro del bloque de detalle cuando se asocia a un renglón dentro del bloque maestro desplegado, adicionando manualmente algunos comandos de procesamiento especial.

2.3.7 Los componentes de SQL*FORMS

2.3.7.1 Campos

Los campos tienen el nivel más básico, estos nos muestran información contenida dentro de la forma. Los valores en un campo pueden ser manipulados por el operador, cuando inserta un valor dentro del campo se ejecuta un **trigger** que cálcula el importe total de una orden y despliega este total. Un campo siempre esta asociado dentro de un bloque. Cada bloque normalmente admite uno o más campos.

Los campos de un bloque usualmente corresponden a columnas de datos en la tabla base del bloque que contiene estos campos. Cuando en este caso entran datos de un campo, pueden afectar los valores en un renglón de la Base de Datos y el campo puede ser llenado al realizar una consulta de la tabla base. Sin embargo, los campos no siempre corresponden a columnas de la tabla base del bloque, algunos de ellos a veces contienen valores calculados, despliegan información relacionada con otra tabla y aceptan una entrada de operador por voluntad del proceso.

Una forma puede desplegar algún número de "instancias" de el mismo campo donde esta definido un bloque de la pantalla de manera que muestra más de un registro de información en un tiempo. Cada ilustración del registro contiene una instancia de cada campo en el bloque. Cada instancia potencialmente corresponde a un renglón de forma individual de la Base de Datos, pero todas las partes de los campos de la instancia tienen las mismas características.

Los campos tienen un número de características que los describen. Estas características especifican, también otras cosas, que benefician al operador y que ya contiene el campo, como son:

- Los datos deben ser desplegados en el campo,
- Validan que tipos de datos son insertados en el campo.

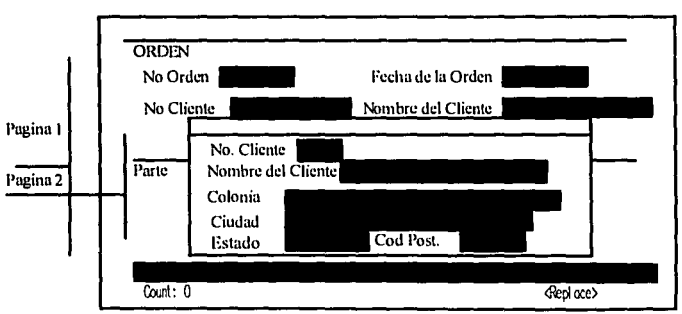
2.3.7.2 Páginas

Las páginas son un conjunto de pantallas de información similar en concepto a una proyección, una página despliega campos y texto de operadores al ser ejecutada la forma. Nótese que las páginas únicamente sirven para desplegar información. Ellas poseen las constantes de los textos (e.g.), campos y etiquetas de bloques que estos despliegan, pero ellas no poseen los campos que se despliegan, ni los bloques, que son en algunas ocasiones asociados a ellas. Mientras todos los campos de un sólo bloque frecuentemente son desplegados en conjunto por una sola página. Estos bloques pueden aparecer en algún número diferente de páginas y por lo consiguiente en diferentes campos de los bloques. Lógicamente ligando campos a bloques, para poder desplegar la liga de campos a página.

Las páginas tienen un número de características semejantes, como la dimensión y en toda una página se despliega la información como una ventana **pop-up**.

No confundir páginas con pantalla. Pantalla es una terminal o monitor que despliega información y no tiene una descripción como la tiene una página. Por ejemplo, múltiples páginas pueden aparecer en la pantalla en un tiempo, como se muestra en la figura 2.6. Como una forma más amplia de diferenciarlas una página puede tener grandes magnitudes que la pantalla no puede desplegar.

Figura 2.6 Multipágina dentro de una Forma



2.3.7.3 Triggers

Los **triggers** son un conjunto de comandos de procesamiento que se pueden escribir. Diferentes tipos de **triggers** son asociados con diferentes eventos o puntos en el procesamiento de SQL*forms. Cuando un evento, de un **trigger** es asociado al **trigger**, se ejecutan los comandos que este contiene. Por ejemplo, cuando un operador [NEXT-FILES] se desplaza de un campo a otro, se ejecuta la tecla "nxtfld" del **trigger**.

Muchos de los **trigger** que tu escribes sin considerar tipo, serán **trigger** versión 3.
Este estilo de **trigger** consisten en comandos escritos en PL/SQL. Este es un lenguaje procedural que esta basado en el lenguaje estándar de la Base de Datos SQL, con el cual se puede poner diferentes tipos de comandos en **trigger** versión 3.

Instrucciones de SQL.
Declaración lógica-procedural
Formas de comandos de procedimiento.

Nota: para versión 2 los **trigger** son escritos en pasos, "ORACLE corporation" mantiene la compatibilidad con otras versiones de SQL*forms, pero para versión 7 de ORACLE sólo acepta formas versión 3.0.

Instrucciones de SQL en Trigger

El Lenguaje de Consulta Estructural (SQL) es un conjunto de instrucciones que pueden ser incluidas en un **trigger** utilizando las restricciones apropiadas ya existentes. Se puede definir en que tipo de **trigger** se debe ejecutar esta instrucción.

Las instrucciones lógicas-procedurales

Instrucciones lógicas procedurales, semejantes a la lógica condicional, enlazando manipulando errores, las cuales pueden ser utilizadas dentro de **triggers**. Estas instrucciones pueden cambiar valores de campos, variables del sistema y el control de información con continuos procedimientos. En adición, estás instrucciones pueden cambiar valores de campo.

Comandos de procesamiento de un trigger en una forma.

Los comandos adicionales se conocen como empaquetamiento de procedimiento, los cuales pueden afectar objetos de la forma y son extensiones de PL/SQL que estan disponibles únicamente en SQL*forms. Estos comandos permiten que ejecutes acciones semejantes al movimiento del cursor, llamado a otra forma, o cambiando características del campo, conjuntamente estos procedimientos se pueden ejecutar con estos comandos bajo un control programado basado en eventos de la aplicación.

Procedimientos a nivel forma

Los procedimientos a nivel forma son un conjunto de comandos que se pueden invocar. Estos procedimientos pueden usar algunos comandos que un **trigger** también puede utilizar. Ellos también toman argumentos y regresan valores, justamente como las subrutinas que contiene un lenguaje de tercera generación como COBOL.

Los procedimientos a nivel forma en un campo local, pueden ser llamados por algún otro procedimiento a nivel forma o un **trigger** de la forma en el momento en que es definido o referenciado.

Todas estas características permiten que se puedan crear rutinas, las cuales pueden ser usadas varias veces. Esto reduce el aumento de lógica que tengas que escribir para realizar algunas tareas y aumenta la eficiencia de las aplicaciones.

2.3.8 Modelos de procesamiento.

Cuando una forma es ejecutada, SQL*forms sigue un conjunto de reglas pre-definidas según estas vanyan ocurriendo. Estas acciones incluyen una navegación, con los modelos de procesamiento, puedes automatizar el comportamiento por default de SQL*forms, dependiendo de las necesidades de tus aplicaciones.

2.3.8.1 Eventos y funciones

Todos los procesamientos son centrados en torno a eventos, simplemente como una introducción. Eventos es un conjunto de objetos que ocurren cuando una forma es ejecutada SQL*forms muestra el manejo acerca de eventos y ejecución de funciones.

Un ejemplo de un evento es el operador precedido del [next-field], cuando este evento ocurre, SQL*forms ejecuta un comportamiento predefinido, puede que se comporte como la ejecución del default de la función [next-field] esto es que el cursor se mueva a la próxima definición o campo, o simplemente como la ejecución de un mensaje de la función, la función [next-field] nos sirve en este caso para desplegar un mensaje al operador antes de que se mueva el cursor.

Notese que durante el procesamiento, son ahorrados usualmente eventos. Que es la ocurrencia de un evento llamado función que llaman a otros eventos.

El incremento de funciones que llamen a eventos y que sean llamados por **triggers** nos dan como resultado los procesos de navegación y validación.

2.3.8.2. El punto de un trigger

Algunas funciones que son llamadas pueden tener uno o más puntos de un **trigger** asociados con el mismo.

Un punto de un **trigger** es un "lugar" temporal en un evento con un tipo de **trigger** asociado. Cuando SQL*forms procesa un evento en algún punto de un **trigger**, este ejecuta el **trigger** asociado que tienes definido como **trigger** de este tipo. Por ejemplo en la ejecución de la función [next-field] hay un punto del **trigger** definido por el **trigger** post-field. El **trigger** post-field es ejecutado primero antes del punto de proceso de SQL*forms.

El punto de un **trigger** y los **triggers** son las primeras herramientas para poder modificar en SQL*forms las formas, los procesos, o para un evento en particular. Notar que no todos los eventos tienen **trigger** o puntos de un **triggers**. Por ejemplo, cuando un caracter es representado de alguna manera en una terminal ocurre un evento de caracteres que son procesados por SQL*forms junto con las teclas de funciones, y de hecho, los campos. Sin embargo no hay un punto en un **trigger**, ni tampoco se puede fijar como un **trigger** dentro de ese evento.

2.3.9 Componentes de SQL*FORMS

SQL*forms consiste en los siguientes programas o componentes, que se pueden ejecutar independientemente con comandos en línea. Así mismo, en diseño utilizamos más, frecuentemente

componentes de SQL*forms (Design), para que se accese a algún otro componente tu requieres de las siguientes herramientas:

SQL*forms (Design)

Permite que utilices formas usando menús, "fill-in-the-form", pantallas, y un comprensivo sistema de ayuda en línea. SQL*forms (Design) almacena objetos definidos como una tabla de la Base de Datos de ORACLE.

SQL*forms (Run-form)

Permite ejecutar u operar formas predefinidas de una manera interactiva, además también puedes utilizar este componente para pruebas y formas en debug.

SQL*forms (Generate)

Permite que puedas crear una forma, definiendo sus archivos desde SQL*forms (Run-form) puedes ejecutar un archivo de texto, o representación de un archivo flotante

SQL*forms

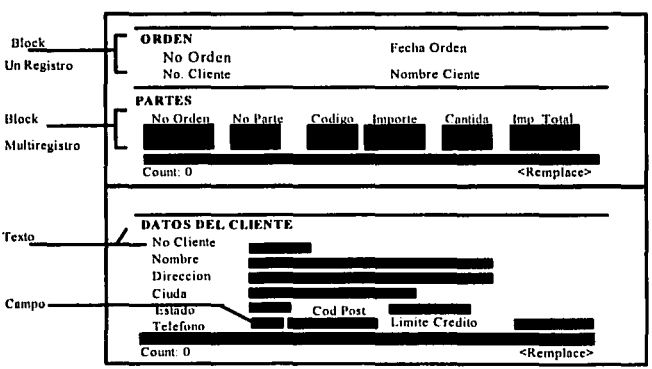
Permite que elijas entre varias versiones de una forma:

1) Crear archivos de texto para la definición de espacios en tablas de la Base de Datos
2) Insertar definiciones de formas (para SQL*forms (design)) y archivos de textos dentro de las tablas de la Base de Datos
3) Crear archivos de texto en el formato de versión 3.0, de archivos de textos creados en anteriores versiones de SQL*forma.

2.3.10 Términos de SQL*FORMS y Base de Datos

Hacer uso efectivo de las referencias, puedes trabajar, teniendo conocimiento de los siguientes términos de SQL*forms y términos de Base de Datos. La figura 6 y 7 muestran algunos términos básicos de SQL*forms

Figura 6 y 7 Términos básicos de SQL*forms



Blocks Anónimos

Son Blocks de PL/SQL sin un nombre y no se requieren que sea expresado explícitamente el BEGIN y END, que las teclas de trabajo encierran en la ejecución de las sentencias. Cuando no uses blocks anónimos en el texto del **trigger** estos deben ser escritos en PL/SQL.

Aplicación

Una forma o una serie de formas deben de satisfacer las necesidades de la empresa. Por ejemplo. Se debe poder construir una aplicación de servicio de órdenes dentro del sistema. Esta aplicación debe de tener contenido menús, reportes y otros componentes

Tabla Base

Es la tabla de la Base de Datos que es propia del block

Campo de la Tabla Base

El campo que corresponde a una columna de la tabla de la Base de Datos de el block propio del campo.

Block

Colección lógica de campos en una forma. Un block puede corresponder a una tabla de la Base de Datos o a ninguna. Colección de comandos de PL/SQL.

Constantes de Texto

El texto que aparece en la forma cuando esta es ejecutada, es donde se incluye, los títulos de los campos, direcciones y gráficas para desplegarlos en pantalla

Contexto

Un concepto que puedes usar para determinar, las partes de la definición de la forma que puedes accesar con la interface de SQL*forms (Design).

Diseñar

Una aplicación desarrollada, que utiliza SQL*froms para crear y modificar la forma.

Campo

Es una área en una página que puede desplegar datos y aceptar entradas de operadores. Los campos usualmente aparecen en video inverso o subrayados. El campo que aparece puede corresponder a una columna de la tabla de la Base de Datos.

Forma

Una colección lógica de blocks, campos, **triggers** y procedimientos a nivel de forma. Aquí determinas como deseas que trabaje una aplicación y como quieres que aparezcan los operadores. Cuando un usuario ejecuta una forma, la forma despliega estos elementos en la pantalla de la terminal.

Procedimientos a nivel-forma

Un block en PL/SQL tiene un nombre que requiere una sintaxis completa de PL/SQL. Se puede hacer referencia a procedimiento a nivel forma en el texto del **trigger**, dentro de los **triggers** en una forma V3

Variables globales

Almacenamiento de un texto, el cual existe a través de la forma. Cuando en una forma se crea una variable global, SQL*froms mantiene la variable hasta que la aplicación deja de existir o hasta que explícitamente la forma remueve está.

Login o clave del usuario

Un "user name" y un "password" se usan en ORACLE RDBMS. Esta cuenta es usualmente diferente a la cuenta que tu tienes para operar el sistema operativo aunque podría ser la misma.

Block multiregistro

Es un block que despliega más de un registro en un mismo tiempo.

Objeto

Es un conjunto de datos, semejantes a una forma, block, campo o **trigger**, que tú puedes copiar, mover o borrar en una sola operación. Es el nombre que se le asigna a un conjunto de datos en la BD ORACLE que pueden ser tablas, índices, vistas, etc.

Usuario operativo

Es el usuario que utiliza una aplicación del sistema.

Paquete de procedimientos

Es un procedimiento integral de PL/SQL que es habilitado en SQL*forms. Cada paquete de procedimientos ejecuta una función en SQL*forms, que puede ser: limpiar un campo o ejecutar una consulta.

Página

Es una colección de información desplegada. Similar en concepto a una proyección deslizada. Una página despliega campos y constantes de un texto en pantalla para el operador de la computadora o terminal, que ejecuta la forma. Una forma regularmente contiene una o más páginas y puede tener el número de páginas que desees crear.

Ventanas Pop-up

Es un objeto de SQL*forms que puede extenderse en una área de ventanas de la pantalla actual. SQL*forms despliega una ventana en respuesta a un evento ó acción del usuario. En una ventana puede aparecer una página, una lista de valores, o editar un archivo.

Registros

Datos de un renglón en una tabla o vista de la Base de Datos, representado en una forma

Alcance

Es el campo, o rango en el cual el **trigger** es operado. Nosotros determinamos el nivel de operación del campo en el **Trigger** (forma, blocks o campo).

Dibujo de la pantalla

En SQL*forms es el área de trabajo, en la cual, puedes modificar la pantalla y la distribución de la forma. El pintado de la pantalla despliega una área de la pantalla en un sólo tiempo.

Blocks de un sólo registro

Un block que puede desplegar un sólo registro en un tiempo.

Variables del Sistema

Son indicadores de información o del estado actual del campo, que nos suministran información del sistema acerca de formas u objetos que contenga la aplicación o forma. Por ejemplo, una variable del sistema la cual contiene el nombre del actual campo o su valor.

Trigger

Una pieza lógica que ejecuta, o "dispara" un suceso en SQL*forms (Al ser ejecutada la forma).

La siguiente tabla muestra dos términos de la Base de Datos Relacional.

Orden	Detalle	Producto	Precio por Unid	Cantidad	Imp. Tot.
610	3	10080	20	1	20
611	1	11100	30	4	120
612	1	18018	15	5	75 ⇐ Renglón
613	2	13013	18	1	18
614	1	14013	10	2	20
⇓					
Columna					

La Base de Datos y sus funciones son descritos en el capítulo siguientes.

Columna

En una tabla de la Base de Datos, es el grupo de celdas "vertical" que nos representa un mismo tipo de datos.

Limitaciones

Reglas o restricciones concernientes a un dato (puede ser una restricción a una columna el NOT NULL), este es respetado a nivel de dato, objeto o aplicación.

Diccionario de datos

Tablas o vistas las cuales son propiedad del administrador de la Base de Datos.

Base de Datos

Es una colección de tablas controladas por un Diccionario de Datos.

Administrador de la Base de Datos (DBA)

Es el usuario supervisor de la administración de la Base de Datos en ORACLE. El Database Administrator (DBA), está autorizado para dar y quitar permisos a los usuarios que accesen a la Base de Datos. Modifica en ORACLE opciones que afectan a todos los usuarios, y desempeña otras funciones administrativas.

Llave Foránea

Es un valor o columna de una tabla que referencia a la llave primaria de alguna otra tabla.

Indice

Una opción estructurada asociada con una tabla que es usada en el System Manager Database Relation (RDBMS) para acelerar la localización de los registros en la Base de Datos y (opcionalmente) garantiza que el registro sea único.

Candado

Una restricción que se asigna a la posición temporal, o control de los recursos de la Base de Datos (tablas o renglones) de el usuario. Un candado puede prevenir que otros usuarios cambién o modifiquen los datos del usuario dueño. Los candados nos ayudan, a no permitir que se realicen actualizaciones por fuera que comprometan la integridad de la Base de Datos.

Llave primaria

Información utilizada para identificar renglónes en la tabla primaria o tabla base. También para saber si es una llave única.

Renglón

Es el valor de un grupo de columnas en forma "Horizontal" dentro de una tabla de la Base de Datos.

SQL (Structure Query Languge)

Así como el idioma español, este, es el lenguaje básico para manipular los datos, y es la interface para que el usuario pueda almacenar y traer información de la Base de Datos ORACLE.

Tabla

Es la unidad básica de información en el sistema manejador de una Base de Datos Relacional (RDBMS). Una tabla tiene dos dimensiones, que esta formado como una cuadrícula entre renglones y columnas.

Transacción

Unidad lógica de trabajo. Específicamente, una transacción es el grupo de eventos que ocurren entre cualquiera de los siguientes eventos, como: el usuario al conectarse a ORACLE, al desconectarse, cuando se salva un cambio en la Base de Datos, o al regresar los cambios

2.4 Introducción al Lenguaje C.

Comencemos con una introducción rápida a "C". Este es un lenguaje de programación de empleo general, no esta ligado a ningún Sistema Operativo aunque muy frecuentemente lo asocian con el Sistema Operativo **UNIX** y, aunque se le ha llamado "Lenguaje de Programación de Sistemas", ha sido utilizado con el mismo éxito para programas numéricos, programas de procesamiento de texto, Base de Datos y muchos otros.

El lenguaje "C" es de relativo "bajo nivel" esto significa que "C" trabaja con la misma clase de objetos que la mayoría de las computadoras: caracteres, números y direcciones que se combinan a su vez con los operadores aritméticos y lógicos, utilizados normalmente en las computadoras.

El lenguaje "C" no contiene operaciones para trabajar directamente con objetos compuestos tales como cadenas de caracteres, conjuntos, listas, arreglos o vectores considerados como un todo. Este lenguaje no tiene definida ninguna posibilidad de realizar asignación de memoria. Aparte de las definiciones estáticas y el manejo de pilas para las variable locales de las funciones, "C" no cuenta con operaciones de entrada salida, ni métodos propios para el acceso a archivos. todos estos mecanismos de alto nivel deben ser aportados por funciones llamadas explícitamente.

De la misma forma, el lenguaje de programación "C" sólo ofrece proporciones de control de flujo sencillo, secuenciales, de selección, de interacción de bloques y subprogramas, pero no multiprogramación, paralelismo, sincronización ó corrutinas. Aunque la ausencia de alguna de estas características podrían parecer una grave deficiencia, el mantener el lenguaje dentro de dimensiones modestas produce beneficios reales. Ya que "C" es relativamente pequeño y puede aprenderse rápidamente. Su compilador es sencillo y práctico, además que posee un alto grado de portabilidad.

Conviene mencionar algunos aspectos históricos, Técnicos y filosóficos de "C" a fin de destacar sus detalles peculiares.

Muchas de las ideas de "C" proviene de un lenguaje aún vigente el BCPL desarrollado por Martin Richards. La influencia de BCPL le llega de forma indirecta a través del lenguaje B. Escrito por Kan Thompson para **UNIX** en una PDP-7.

El lenguaje "C" poseé las construcciones fundamentales de control de flujo sin las cuales no es posible escribir programas bien estructurados. Agrupamiento de sentencias, toma de decisiones **if**, ciclos **Bucles**, (con comprobación de la condición de terminación al principio **while, for** o al final **do while**), y selección entre un conjunto de casos posible **switch**. "C" incluye apuntadores y capacidad aritmética de direcciones, que lo hacen un lenguaje de programación estructurado.

El lenguaje "C" no es un lenguaje "Fundamentalmente Estructurado", en el sentido del PASCAL ó ALGOL68. Es relativamente flexible en la conversión de datos.

Lenguaje de programación "C" esta compuesto de uno o más archivos fuentes. cada archivo fuente contiene una o más funciones de referencia posible con uno o más encabezados de archivos.

Sintaxis para compilar un programa en " C "

$cc -O nombre.c -o nombre de salida.

Sintaxis para ejecutar un programa en " C "

$nombre del programa

El lenguaje de programación se fundamenta en funciones y llamadas a funciones, para escribir un programa en "C" primero hay que crear las funciones y después unirlas. La estructura estándar para la construcción de una función es:

1.- La función llevará paréntesis después del nombre de la función.
2.- Dentro de los paréntesis van los argumentos de la función.
3.- Declaración de la lista de argumentos.
4.- Se abre una llave para iniciar la función.
5.- Dentro de las llaves van las sentencias que define lo que se desea realizar.

La estructura de la función "**main()**".

```
#include <stdio.h>
main()
  {
  printf("Iniciamos \n");
  }
```

#include <stdio.h>	Librería estándar de entradas y salidas de datos.
main()	La función **main()** es la primer función que se llama cuando el programa se ejecuta. Indica el comienzo del programa.
{	Abertura de la Función.
printf("Iniciamos \n");	Cuerpo de la función (función **printf()**).
}	Indica la finalización del programa y salida al Sistema Operativo.

2.4.1 Variables y Constantes

Nombre de las variables. El nombre de las variables es una secuencia de no más de 255 caracteres. Debe iniciar con una letra, mayúscula o minúscula, los letreros son indistintos. En seguida pondrémos una lista de palabras reservadas que no pueden ser usados como nombres de variables.

auto	else	extern	switch
break	entry	for	typedef
case	enum	register	union
char	float	unsigned	void
continue	goto	return	while
default	if	size	
do	int	static	
long	struct	double	

Características.

- Inicia con una letra.
- Tipo de escritura: mayúsculas ó minúsculas.
- Tamaño no mayor a 255 caracteres.
- Letras, dígitos y/o subrayado.
- No se puede iniciar con el nombre de una variable.

Tipo de Variables

Tipo	Declaración	Ejemplo
Caracter	char	char c:
Entero	int, long, short unsigned	int x; short z; unsigned y;
Número Real	float, double	float r; double s;

Valores de las constantes en "C".
- Si empieza con 0x es hexadecimal entero.
- Si empieza con 0 es octal entero.
- Si termina con L es entero largo.
- Si la constante tiene punto decimal es tipo doble o de punto flotante
- Si la constante está entre apóstrofes es caracter.

Todos los programas que producen salida por la pantalla utilizan la función **printf()**. Esta función es una salida genérica por la pantalla. La forma genérica de la función **printf()** es,

printf ("cadena de control", lista de argumentos)

Los comandos de formateo y los argumentos se hacen coincidir de izquierda a derecha. El número de comandos de formateo en la cadena de control le dicen a la función **printf()**, el número de argumentos que debe contener esta.

Las variables y constantes son manipuladas por los operadores, para formar expresiones.

Nombre de las variables. Los nombres de las variables en "C" se construyen con letras y números o el carácter underline, el nombre de la variable debe empezar con una letra. Una variable no puede tener el mismo nombre que una palabra reservada, las mayúsculas y minúsculas son diferentes en "C"; utilizarémos mayúsculas para nombre de variables, y minúsculas para nombre de constantes, sólo son significativos los ocho primeros caracteres de un nombre interno. Aunque se pueden emplear más. Estas son dos clases básicas de Operandos.

Variable
Constantes

Las variables también se pueden "inicializar" en su declaración, aunque existen algunas restricciones. Si el valor va seguido por un signo "=" y una constante, esto constituye una inicialización, ejemplo:

```
int  i = 0;
char c = 'A';
```

En "C", todas las variables deben haber sido declaradas antes de usarlas. El nombre de una variable no tiene nada que ver con su tipo. La sintaxis para la declaración de cada tipo, se muestra en los siguientes ejemplos.

```
int x;          unsigned int ux;    float y;
short int sx;   long int lx;        double yy;
char p;
```

UTILERIAS UTILIZADAS PARA LA APLICACION

Las variables pueden ser declaradas; dentro de las funciones, en la definición de los parámetros de la función, o fuera de todas las funciones. Estas variables son llamadas, variables locales, parámetros formales y variables globales respectivamente.

Variables locales. Sólo se pueden referenciar a las sentencias que hay dentro de la función en donde las variables han sido declaradas.

Las variables locales no son conocidas por las funciones externas, estas sólo son creadas cuando la función es llamada, y se destruyen cuando se salen de la función, por lo que la utilización de la memoria para éstas es de una manera dinámica

Al contrario de las variables locales, las globales mantienen sus valores en todo el programa, mientras este se ejecute. Estas son creadas declarándose fuera de toda función. Pueden ser accesadas por cualquier expresión, independientemente de en que función se encuentre.

Si la variable es externa o estática, la inicialización se realiza una sola vez, conceptualmente antes de que comience la ejecución del programa.

Las variables estáticas son aquellas que mantienen permanentente su valor en su función entre llamadas sucesivas.

Las variables locales y de registro, se inicializan cada vez que se entra a la función, las variables estáticas y globales se inicializan a cero si no se especifica otro valor.

Las variables registros son aplicadas exclusivamente a los tipos "int" y "char". El modificador "register" mantiene el valor de la variable en un registro del CPU en lugar de la memoria que es normalmente donde se almacena, esto hace que las operaciones sobre las variables "register" sean mucho más rápidas, ya que no se requiere de un acceso a memoria. Esta característica las hace ideales para el control de ciclos. Aqui, tenemos un ejemplo de una variable registro.

La declaración de un arreglo es similar a la declaración de una variable simple.

```
int x[100];
```

Los arreglos pueden ser de cualquier tipo de variables. La forma general de la declaración de un arreglo es:

Tipo nombre variable [número de elementos]

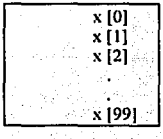
Un arreglo numérico

Declaración

```
int x[100];
```

Inicialización

```
int x[5] = {0, 3, 6, 9, 12, };
x [0] = 0, x [1] = 3, x [2] = 6
x [3] = 9, x [4] = 12
```



Los arreglos empiezan con el elemento inicial cero (el número entre paréntesis es el número de elementos del arreglo), ejemplo de un arreglo "**char**" es

Declaración

char c [4] ;

c [0]	c [1]	c [2]	c [3]

Inicialización

char c [ ] = " hola" ;
char d [10] = "hola";

'h'	'o'	'l'	'a'	'\0'

'h'	'o'	'l'	'a'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'

c [0] = h, c [1] = o, c [2] = l, c [3] = a

Todas las cadenas de caracteres en "C" terminan con un nulo que se añade automáticamente por el compilador, si no se especifica el número de elementos, el compilador cuenta el número de elementos hasta encontrar un nulo.

Las constantes en "C" se refieren a valores fijos que no pueden ser alterados por un programa. Pueden ser de cualquier tipo como se muestra.

TIPO DE DATOS	EJEMPLO
char	'a','', '9'
int	1, 123, -5
long int	38000, -34
short int	10, -5, 90
unsigned int	200, 3000
float	123.25, 4.34e-3
double	123.23, 23123, -0.98786

Una constante de caracter es un único carácter entre apóstrofos. Una expresión constante es una expresión formada únicamente por constantes. Esta expresión se evalúa en tiempo de compilación y, por consiguiente se emplea en cualquier posición en que se aplique una constante, como en:

```
#define MAXLINE 10
char line ( MAXLINE + 1);
```

ó bien

```
seg = 60 * 60 * hora ;
```

Técnicamente, una cadena, es un vector cuyos elementos son caracteres. El compilador coloca automáticamente el caracter nulo '\0' al final de cada cadena para que los programas puedan encontrar el fin. La siguiente función **string(s)**, devuelve la longitud de una cadena de caracteres, excluyendo el final '\0'.

2.4.2 Operadores

El lenguaje "C" es muy rico en operadores incorporados. Un operador es un símbolo que indica al compilador que se esta llevando a cabo manipulaciones matemáticas ó lógicas. El lenguaje "C" tiene los siguientes tipos de operadores.

Aritméticos
Relacionados
Lógicos
A nivel bit
De asignación
Especiales.

2.4.2.1 Operadores aritméticos

Existen dos tipos de operadores aritméticos; unarios y binarios, los operadores unarios sólo usan un operando, mientras que los operadores binarios requieren de dos. Existe sólo un operador unario; - (menos unario), y 5 operadores aritméticos binarios; * (multiplicación), / (división), % (modulo), + (adición) y - (sustracción).

Los operadores + , - son llamados de adición
Los operadores * , / , % se llaman multiplicativos.

OPERADORES	
Unarios	**Binarios**
-	-, +, *, /, %

La división entera trunca cualquier parte fraccionaria, la expresión siguiente produce el resto en la división, por lo tanto es cero cuando el residuo es cero. Por ello, el operador % no puede aplicarse a los tipos "float" ni "double" esto se debe a que trabajan con fracciones.

2.4.2.2 Operadores relacionales y lógicos

Estos operadores se refieren a la relación entre unos valores con otros, y lógicos a la forma en que éstas relaciones pueden efectuarse entre si.

Estos son seis operadores relacionales en "C"

Operadores Relacionales	Acción
<	menor que
>	mayor que
<=	menor o igual que
=>	mayor o igual que
==	igual a
!=	diferente a

Evaluación de los operadores relacionales:

(expr1) es igual a 1 si expr1 es verdadero
(expr1) es igual a 0 si expr1 es falso

Sintaxis.

Variable = (expresión); /* se válida */

Ejemplo:

x = (a < 3); /* x= 1 si a < 3 */
/* x= 0 si a > = 3 */

Los operadores relacionales tienen menor procedencia que los aritméticos y su procedencia es la siguiente:

Precedencia	Operador
mayor	>, >=, <, <=
	==, !=
	&&
menor	||

Los conectivos lógicos "&&" y "||" serán evaluados de izquierda a derecha, y la evaluación se interrumpe cuando se conoce el resultado falso o verdadero. El operador unario de negación es "!" el cual convierte a uno los ceros y viceversa.

TABLA DE OPERADORES LOGICOS y UNARIO

Operadores Lógicos	
&&	"y" Lógica
||	"o" Lógica
!	negación

La procedencia de "&&" es mayor que la de "||" , y ambos operadores tienen menor precedencia que los operadores relaciónales.

A diferencia que otros muchos lenguajes, el "C" cuenta con un conjunto de operadores de bits , los operadores lógicos a nivel bit son usados para la manipulación de bits de una expresión. Estos operadores no se pueden utilizar sobre variable de tipo "float" o "double".

TABLA DE OPERADORES BINARIOS

Operadores	Acción
~	Negación (Complemento a 1)
>>	Corrimiento a la derecha (Llena de ceros los bit's)
<<	Corrimiento a la izquierda (Llenar de ceros bit's)
&	AND ó "y" lógica
|	OR ó "o" lógica
^	OR Exclusiva lógica

Con el operador AND se pone en ceros un conjunto de bits.
Con el operador OR se ponen a unos un conjunto de bit

Este es un ejemplo que muestra como trabajan los operadores de bits

Si x = 0 0 1 1 y y = 1 0 1 0

Entonces:

```
~x  = 1 1 0 0      x & y = 0 0 1 1
x << 2 = 1 1 0 0   x | y = 1 0 1 1
y >> 2 = 0 0 1 0   x ~ y = 1 0 0 1
   = 1 1 1 0
```

Al desplazar a la derecha una cantidad "unsigned", la posición desplazada se rellena con 0. Al desplazar a la derecha una cantidad con signo, se llena las posiciones desplazadas con el bit del signo.

TABLA DE PRECEDENCIA Y ORDEN DE EVALUACIÓN

Operador	Precedencia	Valuación
	Máxima	
( ) [ ]		De izq. a derecha
! ~ ++ - - type * & sizeof		De derecha a izq.
* / %		De izq. a derecha
+ -		De izq. a derecha
<< >>		De izq. a derecha
< <= > >=		De izq. a derecha
== !=		De izq. a derecha
&		De izq. a derecha
~		De izq. a derecha
|		De izq. a derecha
&&		De izq. a derecha
||		De izq. a derecha
? :		De derecha a izq.
= += -= *= /=		De derecha a izq.
^		De izq. a derecha
	Mínima	

Los apuntadores, son apuntadores a direcciones de memoria o de registros como se verá más adelante, que se utilizan al definir una variable.

Los operadores, constantes y variables constituyen las expresiones. Una expresión en "C" es una combinación válida de estos elementos.

La conversión de tipo, de las expresiones. Cuando en una expresión se mezclan constantes y variables de distintos tipos, el compilador de "C" convierte todos los operadores al tipo de mayor precedencia

Estos son tres tipos de género de conversión que hace "C":

Automáticamente
char ⇒ int, short ⇒ int, float ⇒ double
Requiere evaluar la expresión. Utilizando el siguiente esquema.
int ⇒ unsigned ⇒ long ⇒ double
Necesitamos convertir (forzar la expresión al tipo deseado).

2.4.3 Control de flujo de datos

Las sentencia o proposiciones de control de flujo de datos que soporta "C". son; clasificados en 4 tipos de sentencias diferentes:

Primitivas
Loops
De opción múltiple
Llamado a funciones.

2.4.3.1 Proposiciones y bloques (Sentencias Primitivas)

Una expresión x+=x ó **getchar()**, se convierte en una proposición cuando va seguida por ";" que es un terminado de una sentencia.

Con las llaves se agrupan declaraciones y sentencias en proposiciones, en bloque que es equivalente a una proposición simple.

Tipo	Sintaxis
Nula	;
Simple	expresión;
Compuesta	{ sentencia; sentencia; }

La sentencia nula es realizada sobre una expresión nula seguida de ";".

2.4.3.2 Loops.

Estas sentencias son utilizadas en "C" para la ejecución de loops, que son diferentes;

while
for
do/while

El Lenguaje "C" permite la ejecución, de (conjunto de instrucciones), loops el programa ciclara hasta que se cumpla una cierta condición, la cual es definida por el programador, en ese momento terminará el loops. Esta condición puede ser predefinida como en el loop **for**, o no puede ser definida como en los loops **while** o **do/while**.

El loop **while** es usado en la ejecución de la sentencia, mientras la condición de la expresión no regrese un valor diferente de cero.

El loop **while**, es usado generalmente para la ejecución de una serie de sentencias mientras la condición sea verdadero. Cuándo la condición sea falsa, el control en el programa pasa a la línea siguiente después de la sentencia.

Diagrama	Sintaxis
	while (expresión) sentencia;
Expresión	
	Ejemplo
	While [x < 10]
Sentencia	{
	printf (Inicio \n");
	x+=x;
	}

La sentencia **for**. El loops **for** es usada como un loop interactivo (contador), que consta de tres partes separadas por punto y coma. El **for** es apropiado para loops en donde la inicialización y la reinicialización son sentencias simples y relacionadas lógicamente.

Diagrama General

Inicialización	Sintaxis para bloques
Condición	**for** (inicio;condición;reinicio);
	{
Sentencia	Sentencia1;
	Sentencia2;
Reinicialización	}

La inicialización normalmente es una expresión de asignación que se evalúa una y sólo una vez antes de que alguna otra parte del loop sea evaluada o ejecutada. La condición normalmente es una expresión relacional que determina cuándo saldrá del loop. Si la expresión de condición entrega un cero el loop es terminado, si ese resultado es diferente de cero indica que el loop se esta ejecutando.

La reinicialización, es una expresión que evalúa cómo cambiará la variable de control del loop cada que se repita este. La expresión es evaluada después de cada interacción del loop.

A diferencia de loop **for** o del **while**, que prueban la condición de terminación al principio en lugar de hacerlo al final. El loop **do/while** hace la comprobación al final de loop. Esta significa que por lo menos se ejecuta una vez el loop.

Esto es su formato general así como su diagrama del **do/while**.

Diagrama	Sintaxis
	do { Sentencia;
Sentencia	} **while** (expresión);
Expresión	Ejemplo:
	do {
	printf ("Escribe una mayúscula \n");
	c = **getchar**();
	} **while** (c < 'A' || c > 'Z')

La primera que se realiza en un **do/while** es la sentencia y posteriormente es evaluada la expresión. En caso de ser cierta, se ejecuta de nuevo la sentencia y así sucesivamente, si la expresión al ser evaluado es verdadera, el loop continua, de lo contrario el loop termina.

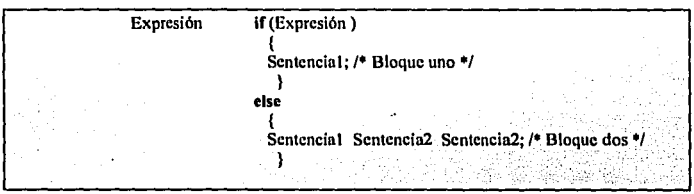
2.4.3.3 Sentencias de opción multiple

Existen dos tipos de sentencias de opción múltiple, condicionales e incondicionales. El lenguaje "C" soporta dos diferentes tipos de sentencias condicionales y tres incondicionales.

Condicionales	Incondicionales
if/else	**goto/label**
switch/case	**break**
	continue

La sentencia **if/else** sirve para tomar decisiones. La sentencia **if** es condicional. Esta expresión es evaluada, si la expresión regresa un valor verdadero (diferente de cero), la sentencia se ejecuta, si no, la sentencia no se ejecutará. La sentencia **if** tiene una parte opcional el **else**, en donde puede ejecutar una u otra opción. Este diagrama nos muestra como funciona.

Diagrama **Sintaxis**



Aquí, si la expresión es verdadera se ejecuta la sentencia1, de lo contrario se ejecuta la sentencia2. Esto también puede ser utilizado para sentencias en bloque.

La sentencia **switch/case** es una utilería especial para decisiones con opciones múltiples, la cual comprueba si una expresión es igual a una lista de valores constantes. Cuando se obtiene una igualdad se ejecuta una sentencia o bloque de sentencias, el **default** se ejecuta si no existe alguna igualdad.

El **default** es opcional y si se preside de esté, cuando la opción es incorrecta no se realiza ninguna acción.

La escritura del **case** y del **default** se realiza en cualquier orden. Algo que debe tomarse en cuenta es lo siguiente: que todas la opciones deben ser diferentes, llevar la sentencia **break** en forma simultánea. La sentencia **break** utilizada dentro de cada **case** de un **switch** hace que se produzca una salida inmediata de la instrucción **switch**, y que continue con la sentencia siguiente del **switch**. Esto se debe a que **case** actúa como una etiqueta que indica donde debe seguir el programa después de leer una opción. El **break** es necesario para la ejecución, si se excluye la ejecución continuará en el siguiente **case** hasta ejecutarlos a todos.

Diagrama	Sintaxis
Expresión	**switch** (expresión)
	{
Sentencia A	**case** (A) : Sentencia A;
	break;
Sentencia B	**case** (B) : Sentencia B;
	break;
	:
Sentencia N	.
	default: Sentencia N;
	break;
	}

La sentencia **goto/label** normalmente no es muy necesaria en el lenguaje "C" ya que este contiene un conjunto muy amplio en estructuras de control y de sentencias de control incondicional como el **break** y el continue. El **goto** necesita una etiqueta **(label)** la cual va seguida de dos puntos.

El uso más común del **goto** es para salir de un proceso en alguna estructura profundamente anidada o la cual este muy confusa, una sentencia **break** no serviría ya que sólo saldría del loop más interno.

Sintaxis	**Ejemplo**
goto label;	**goto** final;
Sentencia;	x = 1
:	x += x
label : Sentencia;	final: x = 3

El principal defecto del **goto** es que tiende a enmascarar los programas y dejarlos casi ilegible.

Por último, veremos las proposiciones incondicionales **break** y continue. Las sentencias **break** y **continue** son usadas para la interrupción del flujo normal de loops y en la sentencia **switch.**

Si la sentencia **break** es encontrada en el cuerpo del loops está provoca que el control del flujo salte a la primer sentencia que sigue a el loops por lo tanto el **break** es como una salida del loops.

Si la sentencia continue es encontrada en el cuerpo del loops **while** o **do/while** esta provoca que el control del flujo salte a la expresión condicional y de que luego continúe el proceso del loops. En el caso de **for**, se ejecuta la parte de reinicialización del loops, la condición y el loops continuará. El **continue** no será utilizado por la sentencia **switch.**

Sintaxis break y continue

```
while (expresión) {
        Sentencia1;
        Sentencia2;
        if (condicional)
          break;
        else
        continue;
        Sentencia3;
              }
```

```
do {
        Sentencia1;
        Sentencia2;
        if (condicional)
          break;
        else
          continue;
        Sentencia3;
        } while (expresión);
```

```
for (exp1; exp2; exp3) {
        Sentencia1;
        Sentencia2;
        if (condicional)
          break;
        else
          continue;
        Sentencia;
        }
```

Otra forma de determinar la salida de un loops es utilizar la función **exit()** que se encuentra en las librerías estándares de "C". La función **exit()** origina una terminación inmediata del programa y una salida al sistema operativo, por lo que es muy utilizada.

Sintaxis de una Función
[return_var =] función_nombre (arg1,arg2,... argn);

Estos son varios tipos de sentencias disponible en lenguaje "C", y ellos caen dentro de una de estas categorías.

Tipos de Sentencias

Primitivas	Loops	Opción Múltiple	Llamado de Funciones
Nulas	**while**	**if/else**	
Simples	**for**	**switch/case**	
Compuestas	**do/while**	**goto/label**	
		break	
		continue	

El procesador de "C" corre automáticamente en el tiempo de compilación. El procesador satisface algunas direcciones #. Estos son dos que con mayor frecuencia se utilizan.

#include
#define
#include < nombre_archivo >

El efecto que produce esta sentencia es copiar físicamente el archivo /usr/include/nombre_archivo dentro del archivo fuente.

#include "nombre_archivo"

El efecto que produce es copiar el archivo nombre_archivo dentro del archivo fuente

A continuación ciertas rutinas utilizadas en los encabezados de los programas, mostrados en orden decreciente dependiendo de su uso.

#include < stdio.h >
#include < string.h >
#include < math.h >
#include < ctype.h >
#define string1 string2

El efecto que realiza el procesador es sustituir el string2 por el string1 en cada ocurrencia (después de la dirección #) en el programa fuente.

#define macros

El **#define** también puede ser utilizado para definir macros, esto obliga al compilador a que realiza una macro instrucción.

Sintaxis

#define función(x) (expresión suponiendo (x))
Ejemplo:

Antes de correr el proceso
#define TEST (x) ((x) == ' A ' || (x) == ' B ')
char c;
if (TEST(c))
Después de correr el proceso
if ((c) == ' A ' || (x) == ' B '))

El **#include** y **#define** son necesarios para utilizar las funciones sobre ficheros en memoria estos son suministrados por el compilador. Cada fichero que utilice E/S por disco requiere que se lea un fichero de cabecera denominado stdio.h .

2.4.4 Estructura de las funciones

El lenguaje "C" se diseño para hacer que las funciones fueran eficientes y fáciles de usar. Los programas escritos en "C" normalmente constan de una gran cantidad de pequeñas funciones en lugar de pocas y grandes. Un programa puede residir en uno o más archivos fuente, de cualquier forma, según convenga; los archivos fuente pueden compilarse por separado y enlazarse junto con funciones de biblioteca compiladas con anterioridad.

Las funciones son bloques con los que se construyen programas en "C", y en ellos se lleva a cabo toda actividad del programa. Una vez que una función se ha depurado, puede utilizarse siempre.

Formato General de una Función

[tipo_dato] nombre (parámetros)
declaración de parámetros.
{
declaración de variables locales.
sentencia
sentencia
sentencia
return expresión;
}

Tipo_dato - Define el tipo de la función (por default es **int**).
Nombre - Es el nombre de la función. Este sólo puede ser llamada una vez **main**.
Parámetros - Una lista de expresiones, Las cuáles pasan los valores a la función.
Return - La expresión es evaluada y el valor que devuelve **return** será el valor de la función.

Todas las funciones, devuelven un valor, esto puede ser explícitamente especificado por la sentencia **return()**, ó bien puede ser "0" si no se especifica ningún otro valor. Por default las funciones regresan

valores enteros, sin embargo una función no puede ser objeto de una asignación, en este caso el compilador mandará un error y no compilara el programa.

Funciones que devuelven un valor como resultado, **sqrt(), mux().**
Funciones que solo indican el éxito o fracaso, **write(), fopen().**
Funciones que solo indica alguna clasificación de datos y no devuelve ningún valor **clasif().**

Estas son dos maneras de pasar los parámetros de las funciones. Ellas son conocidas generalmente como llamada por referencia y llamada por valor. Diferentes lenguajes de programación utilizan una (o ambas) de estás técnicas.

Llamada por referencia.

Con este método, la dirección de cada argumento se copia en los parámetros de la función, esto significa que no cambia el parámetro en la función llamada, si no cambia el valor de el parámetro correspondiente en la llamada de la función.

Llamada por valor.

Este método copia el valor de cada uno de los argumentos en los parámetros formales de la función, de está manera no cambia el parámetro en la función llamada. Esta no afecta el valor del correspondiente parámetro en la llamada de la función.

El ámbito de las variables

Locales

auto (default)
static

Global

auto (default)
static
extern

Variables externas.

Las variables externas se definen fuera de cualquier función y, por tanto, son potencialmente utilizables por muchas funciones. También estás son siempre externas, pues "C" no permite definir funciones dentro de otras. Por definición las variables externas son también "globales" por lo que a todas las referencias de tales variables del mismo nombre les corresponde el mismo dato.

Características.

1. Las variables externas son preferibles a una gran lista de argumentos.
2. En particular los arreglos externos se puede inicializar, pero los automáticos no.
3. Su campo de validez y su duración se mantienen entre una invocación de la función y la siguiente.

Variable estáticas.

Las variables **static** pueden ser internas o externas. La variable **static** interna es local a la misma función, en la misma forma que las automáticas pero, a diferencia de ellas, su existencia es permanente, en lugar de aparecer y desaparecer cada vez que se llama a la función.

Una variable **static** externa es accesible en el resto del archivo fuente en el que está declarada, pero no en otro. Por lo tanto, el almacenamiento **static** externo proporciona un medio para ocultar nombres.

El almacenamiento estático, interno o externo, se especifica al prefijar la declaración normal con la palabra **static**. La variable es externa si se define fuera de las funciones e interna si se define dentro de una función.

Variables automáticas (dinámicas)

Se crean cuando la función se ejecuta y se destruyen cuando se acaba la función, sólo es conocida por la función donde se declara.

Variables registro.

Una declaración **register** avisa al compilador que la variable en cuestión será muy usada. Cuándo es posible, las variables **register** se colocan en los registros de la computadora, lo que produce programas más cortos y más rápidos.

Estas son las cuatro clases de almacenamiento en "C": **auto**(default), **static**, **extern**, **register**(default).

2.4.4.1 Recursividad.

Las funciones de "C" pueden utilizarse en forma recursiva. Esto significa que una función puede llamarse asimismo directa o indirectamente.

La recursividad generalmente no ahorra memoria pues hace mantener una pila con los valores que están siendo procesados. Tampoco será más rápida, pero el código recursivo es más compacto y a menudo más sencillo de escribir y comprender. Está especialmente diseñado para estructura de tipo árbol.

Las funciones recursivas deben tener 2 características principales.

1. La condición para detenerse es, algunas veces, llamado el límite de la condición.
2. Debe decrementarse para excluir procesos recursivos infinitos.

Muchas de las facilidades en "C" no son parte de las sentencias fundamentales del lenguaje. Pero son una parte del estándar en "C" **Run-time Library**. Estas facilidades son referenciadas a las funciones de las librerías.

Las librerías estándares en "C" contienen funciones para un número de proposiciones.

Operación de caracteres
Operación de strings
Operación matemáticas
Proceso de asignación de almacenamiento
Manejo de archivos de entrada/salida
(Comunicación con el Sistema Operativo) depende del equipo.

2.4.4.2 Llamada de funciones.

Se puede utilizar el nombre de un arreglo de caracteres sin ningún índice, si se quiere llamar a una función con una cadena como argumento. Lo mismo es válido para todos los arreglos que se pasan como argumentos a las funciones.

Cuando se llama a una función con el nombre de un arreglo, la dirección del primer elemento en el arreglo se pasa a la función. Esto significa que la declaración de parámetros tiene que ser un puntero. Esto es debido a que en "C" los arreglos realmente son punteros a una región de memoria y la pareja "[]" es un operador que encuentra el valor de los datos que es un índice de arreglos especificado entre corchetes.

2.4.5 Apuntadores y arreglos

Un apuntador es una variable que contiene la dirección de otra variable. Puesto que un apuntador contiene la dirección de un puntero se puede accesar al objeto "indirectamente" a través de ellos.

Un apuntador es una variable (localidad de memoria) que contiene la dirección de alguna otra localidad de memoria.

Los apuntadores especiales sobre punteros son "&" y "*".

& .- Accesa a la dirección de memoria que contiene la variable.
* .- Contiene el valor de la localidad de memoria.

EJEMPLO

p = &x se lee como asigna a "p" la dirección de "x"
y = *p se lee como asigna a "y" el contenido de la dirección en "p".

También es necesario declarar las variables. La variable *p equivale a declarar una variable de tipo **int**. La sintaxis de la declaración de una variable limita a la sintaxis de la expresión en la que aparece la variable.

```
int x,y
int *p
```

Se debe observar que en la declaración de un apuntador se restringe el tipo de datos a los que apuntará.

Como en cualquier variable, un puntero se puede o no utilizar en el lado derecho de la sentencia de asignación para asignar su valor a otro puntero, como se muestra en el ejemplo

Código	**Resultado**			
int x,y,*p	X	Y	P	
	?	?	?	
	1024	1026	1120	<------- dirección
x= -7	-7	?	?	| de
	1024	1026	1120	<------- memoria
p=&x;	-7	?	1024	|
	1024	1026	1120	<-------
y= *p;	-7	-7	1024	|
	1024	1026	1120	<-------

Esto es lo que realiza

```
int x,y,*p;   /* declara dos enteros y un apuntador entero (p) */
  x=-7;       /* asigna el valor -7 a la variable x (localidad de memoria 1024) */
  p=&x        /* inicializa p y le asigna la dirección de la variable x al apuntador p con valor 1024 */
```

```
y=*p      /* Se pasa a y el contenido de la localidad de p donde la dirección de p es 1024 el */
          /* contenido de la localidad de memoria es -7 */
```

Ya que "C" pasa "por valor" los argumentos de las funciones, no hay forma directa de que la función llamada altere una variable de la función que la llama.

Por efecto de la llamada por valor, la función **cambio()** modifica los argumentos **a** y **b** (llamada por referencia), que devuelven la dirección de la variable, ejemplo:

```
                    cambio(&a, &b);
    cambio(px, py)
        int *px, *py
        {
            int temp,
            temp= *px
            *px=*py
            *py=temp
}
```

Los apuntadores como argumentos suelen emplearse en el caso de funciones que devuelven más de un valor simple. Sólo hay dos operadores aritméticos que se pueden utilizar con los punteros + y -

x= ++(*p) El contenido de la localidad cuya dirección esta en **p** es incrementado. Donde **x** es igual al contenido de la localidad cuya dirección esta en **p**. La variable **x** se le asigna el nuevo valor incrementado. El resto del apuntador **p** no cambio.

Las condiciones anteriores se cumplen para el operador "--" sólo que en lugar de incrementar el valor o la dirección estas son decrementadas.

2.4.5.1 Arreglos de apuntadores

El lenguaje "C" dispone de arreglos rectángulares bidimensionales. Para "C", un arreglo bidimensional es realmente un arreglo unidimensional por definición, cada elemento es un arreglo por lo que los subíndices se escriben como:

char strings [i] [j] En lugar de **char** strings [i,j]

Los elementos se almacenan por fila, es decir el subíndice situado más a la derecha varia más rápido.

La inicialización de un arreglo se efectúa mediante una lista de valores situados entre llaves, cada fila de un arreglo bidimensional es inicializada mediante una correspondiente sublista, veremos en forma gráfica como trabajo.

char strings [5] [10];

strings [0] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
[1] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
[2] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
[3] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
[4] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
0 1 2 3 4 5 6 7 8 9

Esta es alguna otra forma de obtener e introducir un dato dentro de "C". Utilizando comandos en línea para leer un dato.

Cuando es invoca **main()**, al iniciar la ejecución se llaman dos argumentos (**argc, argv**), el primero cuenta el número de argumentos en la línea de comandos y el segundo es un apuntador a un arreglo de cadenas de caracteres que contienen los argumentos, uno por cadena, su sintaxis es la siguiente:

```
main(argc,argv)
{
        int argc;
        char *argv[];
}
```

El elemento **argv** es una constante que apunta al inicio de un arreglo de apuntadores, cada elemento de el arreglo apunta al inicio de un arreglo de caracteres.

Cada argumento en línea de comandos almacena uno de estos arreglos a caracteres. Cada arreglo de caracteres es terminado con "\0" (elemento nulo). Este es un ejemplo

```
        Sclave juan pepe x
                                [ 'c' 'l' 'a' 'v' 'e' '\0' ]
argc [4]        argv    [ 'j' 'u' 'a' 'n' '\0' ]
                        [ 'p' 'e' 'p' 'e' '\0' ]
                        [ 'x' '\0' ]
```

Esta es alguna otra forma de escribir apuntadores a arreglos **argv[0][2]** y es lo mismo que ***(argv[0]+2)**, que es del tipo caracter y su valor es "a" .

En "C", una función no es una variable, pero se puede definir un apuntador a una función que puede ser: manipulada, pasada a otra función, colocada en un arreglo etc. Para entender lo que significa un puntero a funciones hay que entender un poco sobre como se compila una función y se le llama en "C". Primero el código fuente se transforma en código objeto, que ejecuta las actividades de la función; segundo, durante el enlace la dirección en donde empieza el código de la función, es conocida. Cuando se hace una llamada a una función, esta se realiza en lenguaje de máquina hacia la dirección de memoria de la función llamada.

Por lo tanto, un apuntador a una función realmente contiene la dirección en memoria donde inició el código de la función. La forma general de escribir un apuntador a funciones es la siguiente:

Sintaxis

```
        type (*variable) ();   /* apuntador a una función */
```

Usamos

```
        int(*p) ();
        int func(),x;
        p = func;
        x = (*p) (arg1, arg2);
        p() = es el nombre de la función
        (*p) (x) = es el apuntador a la función
        int = es el tipo de  do de la función
```

2.4.6 Estructuras

El "C" permite crear tipos de datos nuevos de dos formas: primero, combinando mucha variables en una variable denominada estructura; y segundo, utiliza una unión para permitir que muchas variables compartan la misma memoria.

La estructura es un conjunto de variables, que pueden ser de diferentes tipo, los cuales están referenciados bajo un mismo nombre. "C" utiliza estructuras para mantener en un sitio conveniente la información la cual está relacionada.

La estructura relaciona datos complicados, en programas grandes, ya que permite tratar como una sola variable a un conjunto de estas, en lugar de tratarlas como entidades independientes. Una estructura se puede manejar como una función. El siguiente fragmento de programa declara una estructura, la cual es el registro de la nómina.

```
struct nomi{
        char nombre [30];
        int salario [12];
        char ciudad [20];
        char fec_nac [09];
        unsigned long int cp;
};
```

El nombre de la estructura es "nomi", la cual se termina con un punto y, debido a que esta es una sentencia, sus elementos o variables dentro de la estructura se le denominan miembros.

La estructura puede ir seguida de una o varias variables, para declarar una variable en una estructura se puede realizar de dos maneras.

```
        struct dir cap
```

Aquí declaramos una variable "cap" del tipo **dir**, también se puede declarar una o más variables cuando declara la estructura ejemplo.

```
struct dir {
        char nombre [30];
        int salario [12];
        char ciudad [20];
        char  fec_nac [9];
        unsigned long int cp;
} cap, anexo, inf;
```

Aquí definimos una estructura llamada **dir**, la cual tiene declarada las variables "cap", "anexo" e "inf" que son de tipo **dir**. El formato general de una estructura es el siguiente.

```
struct nombre_estructura
{
   type nombre_variable;
   type  nombre_variable;
        .
        .
} variable_estructura,...;
```

En donde se puede omitir el nombre de la estructura o la variable de la estructura. El operador punto "." en el lenguaje "C" nos indica que queremos accesar a un elemento de la estructura el cual nosotros definimos. Su sintaxis es la siguiente.

nomb_estruc.nomb_elem

2.4.7 Archivos entrada y salidas.

El propósito en este capítulo es dar una breve explicación de las librerías necesarias para abrir, cerrar y accesar archivos en "C".

Los archivos en "C" son fáciles de implementar. **FILE** es una estructura de tipo de datos declarada en **typedef** que aparece en la cabecera de la librería **<stdio.h>**. Se pueden accesar archivos en "C" usando punteros para el tipo **FILE**. Para lo cual deberán usarse las direcciones de compilación.

#include <stdio.h>

El archivo **stdio.h** define ciertas macros y variables empleadas por la biblioteca estándar de E/S, el uso de los signos "<>" en lugar de las comillas lo indica el compilados que busque el archivo en la librería estándar /usr/include.

Para hacer que el encabezado de la librería **stdio.h** sea incluido en cualquier programa que accesa archivos se debe realizar la siguiente operación (La declaración del puntero se muestra en el ejemplo).

```
#include <stdio.h>
   :
FILE *prt
```

En el ejemplo "prt" es declarado como puntero a **FILE**. Recordar que la declaración de un puntero no se inicializa automáticamente.

Se inicializa el puntero cuando se abre el archivo. Para abrir el archivo. Se utiliza la función **fopen**(abrir archivo) de la librería de "C". La función **fopen** se regresa al puntero para abrir el archivo. Los argumentos para **fopen** especifican el nombre del archivo que se va a abrir y el modo con el que los archivos serán abiertos.

Los modos con que los archivos pueden abrirse son los siguientes.

r - abrir para leer
w - abrir para escribir
a - abrir para adicionar a un archivo ya existente

La función **fclose** de la librería estándar de "C" se utiliza para cerrar archivos. Sólo un argumento es pasada para **fclose**. El puntero que será obtenido con la llamada de la función **fopen**. Ahora que los archivos están abiertos. ¿Qué desea hacer con ellos?

Leer de ellos
Escribir en ellos
Moverse alrededor de ellos

Leer un archivo se puede hacer de varias formas,a saber:

```
FILE *fp
fp = fopen ("archivo","r");
/* Formato de entrada */
fscanf (fp,"con_string",Rarg1,Rarg2..),
```

La función **fscanf** se comporta exactamente igual que **scanf,** excepto que deberá especificar el puntero de archivos para **fscanf.**

Existen números as maneras para escribir en un archivo Para las opciones posteriores se asume lo siguiente:

```
/* formato de salida */
fprintf (fp,"cont-string",arg1,arg2..);
```

fprintf es exactamente igual a **printf,** excepto que deberá de especificarse el apuntador al archivo de salida en **fprintf**

2.5 Introducción a shells

Empezaremos definiendo que es un **shell,** este es la interface entre el sistema operativo y el usuario, el **shell** interpreta el texto que se escribe, y las teclas presionadas, en el orden de la dirección del sistema operativo **(hp-ux)** ejecutando la acción apropiada de la tarea a realizar, un **shell** puede servir además como un lenguaje de programación.

Bourne shell (sh) es el lenguaje más viejo del **shell.** Este fue escrito por Stephen Bourne en los laboratorios Bell. El **Bourne shell** era el default para los usuarios del HP-UX y era un factor estándar dentro de la industria. El **Bourne shell** no tiene las características de interactividad, y no existe una completa construcción de programación en "C" como el **Korn shells**

C shell (csh) es el desarrollo de Bill Joy en la Universidad de California y Berkeley, su sintaxis es muy parecida a la del lenguaje de programación "C". Este tiene el poder de la interactividad, característica de guardar la historia de los comandos y el nombre dentro de un archivo.

Korn shell (ksh) es un nuevo desarrollo de David Korn en los laboratorios Bell, y es compatible en cuanto a las características de **Bourne shell,** pero muy por encima en cuanto a sus ejecución. Este es interactivo, se caracteriza por guardar la historia como **C shell,** pero además es fácil de ejecutar y puede trabajar con comandos en línea, compatible con el editor.

Key shell (Keysh) es una interface para el **Korn shell** y es un desarrollo de Hewlett Packard Company. Está provista de menús y de la asistencia de ayuda en línea de los comandos dentro de grupos. Nos muestra el performance del las tareas por medio de la visualización de archivos, también cuenta con monitoreo de impresión de archivos, y una lista del contenido de los directorios.

PAM Manejador Personal de Aplicaciones es diseñado para correr aplicaciones de una manera más cómoda y poder visualizar su ejecución; "function-key-drive" es la interface de navegación dentro del archivo del sistema que nos permite realizar ejecuciones de programas.

2.5.1 Uso de los comando del shell

Aquí podemos ver algunos métodos para la ejecución de los comandos en **shell**, es necesario que nos familiarizemos con la ejecución de algunos comandos, como cuando deseamos correr el comando **date**, esto es, la adición de un sólo comando en la línea de comandos seguido por un retorno de carro (return), con esto es habilitada la ejecución del comando además se pueden incluir opciones y parámetros a este.

Todos los comandos en **UNIX** tiene múltiples opciones por lo que no mencionaremos más que algunos comandos con alguna de sus opciones. Estas opciones son usualmente precedidas por un ion (-) y son separadas del nombre del comando, algunas otras opciones, y parámetros sólo son separados por un espacio en blanco. Parámetros ó variables, son datos que no propiamente son necesarios en la función del comando. Si tu haz omitido algún parámetro para el comando **ls**, el actual directorio es listado. Si se incluye el nombre de un directorio (o el nombre de la trayectoria) como un parámetro, se esta listando la impresión en pantalla del directorio seleccionado. La sintaxis de los comandos usualmente tomada es de la siguiente forma

comando [opción] [parámetros]

2.5.2 Secuencia de procesamiento

Cuando se introducen comandos línea por línea (presionando **return** después de cada comando), estos son validos en el sistema hasta que se completa el comando (o programa) antes de que el siguiente comando pueda ser ejecutado. Ejemplo de la ejecución de algunos comandos:

$date <return>
$ps -ef <return>
$who -u <return>

Para poder ejecutar cada uno de los anteriores comandos tienes que completar el comando terminándolo con un **return**. Ahora escribiremos todos los comandos dentro de una sola línea, pero utilizaremos un separador ";" para cada uno de los comando escrito. Por ejemplo.

$date; ps -ef; who -u

Antes de completar el comando con un **return**, ahí mismo se escriben todos los comandos sólo utilizando el separador de línea ";". Este proceso es llamado proceso secuencial.

El nuevo programa o comando no se puede estar utilizando o ejecutando hasta que el procesamiento del programa o comando anterior se haya completado.

Si algunos parámetros son requeridos en el comando o programa, estos son escritos como usualmente se hace, precedido por un "-" y separado del nombre del comando. El ";"es desplazado después de los parámetros.

Cuando un programa ejecuta un proceso de manera secuencial, en este momento el teclado no responde, sino hasta después de que el programa ha sido completado (las opciones o comandos tecleados en ese momento se postergan en el **buffer** para posteriormente ejecutarse).

2.5.1 Uso de los comando del shell

Aquí podemos ver algunos métodos para la ejecución de los comandos en **shell**, es necesario que nos familiarizemos con la ejecución de algunos comandos, como cuando deseamos correr el comando **date**, esto es, la adición de un sólo comando en la línea de comandos seguido por un retorno de carro (return), con esto es habilitada la ejecución del comando además se pueden incluir opciones y parámetros a este.

Todos los comandos en **UNIX** tiene múltiples opciones por lo que no mencionaremos más que algunos comandos con alguna de sus opciones. Estas opciones son usualmente precedidas por un ion (-) y son separadas del nombre del comando, algunas otras opciones, y parámetros sólo son separados por un espacio en blanco. Parámetros ó variables, son datos que no propiamente son necesarios en la función del comando. Si tu haz omitido algún parámetro para el comando **ls**, el actual directorio es listado. Si se incluye el nombre de un directorio (o el nombre de la trayectoria) como un parámetro, se esta listando la impresión en pantalla del directorio seleccionado. La sintaxis de los comandos usualmente tomada es de la siguiente forma

comando [opción] [parámetros]

2.5.2 Secuencia de procesamiento

Cuando se introducen comandos línea por línea (presionando **return** después de cada comando), estos son validos en el sistema hasta que se completa el comando (o programa) antes de que el siguiente comando pueda ser ejecutado. Ejemplo de la ejecución de algunos comandos:

$date <return>
$ps -ef <return>
$who -u <return>

Para poder ejecutar cada uno de los anteriores comandos tienes que completar el comando terminándolo con un **return**. Ahora escribiremos todos los comandos dentro de una sola línea, pero utilizaremos un separador ";" para cada uno de los comando escrito. Por ejemplo.

$date; ps -ef; who -u

Antes de completar el comando con un **return,** ahí mismo se escriben todos los comandos sólo utilizando el separador de línea ";". Este proceso es llamado proceso secuencial.

El nuevo programa o comando no se puede estar utilizando o ejecutando hasta que el procesamiento del programa o comando anterior se haya completado.

Si algunos parámetros son requeridos en el comando o programa, estos son escritos como usualmente se hace, precedido por un "-" y separado del nombre del comando. El ";"es desplazado después de los parámetros.

Cuando un programa ejecuta un proceso de manera secuencial, en este momento el teclado no responde, sino hasta después de que el programa ha sido completado (las opciones o comandos tecleados en ese momento se postergan en el **buffer** para posteriormente ejecutarse).

Programas en proceso, cuando un programa en proceso secuencial es ejecutado continúa corriendo como usualmente lo hace. Mientras un programa es corrido de esta forma, no existe otra opción más, esperar hasta que el programa finalice.

2.5.3 La utilización de pipes

Dos ó más comandos conectados en una salida estándar de un programa puede ser usados en la entrada de algún otro programa, el dato de la trayectoria que enlaza los programas es llamado **pipe**. Con los **pipes** se puede redireccionar la entrada y salida de un programa al mismo tiempo. Estos utilizan archivos temporales.

Cuando los programas son concatenados con **pipes**, estos **shell** coordinan la entrada y salida entre los programas. Los **pipes** sólo transfieren datos de alguna entrada estándar hacia algunos otros programas.

Cómo es la concatenación de los programas con **pipes**. La barra vertical (|) es el símbolo del **pipe**. Los parámetros de el programa son listados después del nombre del programa, pero después del símbolo "|". El espacio entre el nombre del programa y la barra vertical es opcional. La sintaxis para la concatenación de programas con **pipes** es la siguiente.

programa_a | programa_b | programa_c

No importa si es un programa o un comando ejecutable. Por ejemplo, el programa_a forzosamente requiere que se teclee un dato de entrada, el programa_a toma este dato y lo direcciona a la salida **stdout** (estándar de salida), esta salida es pasada a través del primer **pipe** para ser recibido en la entrada del programa_b, el programa_b forzosamente comprueba que el dato sea válido y lo procesa necesariamente en algún momento del programa, quizá solo lo sortea. En el segundo **pipe** se recibe en la entrada del programa_c el dato de salida del programa_b, el programa_c formatea la entrada y nos entrega el resultado deseado ya sea en pantalla o en un reporte impreso.

Como muestra he aquí unos ejemplos:

```
$ls -l|wc
```

Imprime el número de archivos en el actual directorio

```
$ls |more
```

Imprime una lista de archivos en el actual directorio y página de forma conveniente para la vista de la pantalla.

```
$cat file |pr | lp
```

Manda el contenido del archivo a **pr**, que lo formatea y lo pasa hacia **lp** para que este lo mande imprimir en la impresora rápida en línea.

2.5.4 Programación básica en shell

Después de haber dominado algunos comandos simples de **shell**, pueden moverse dentro de los aspectos de programación de **shell** (**shell** scrips y programas **shell** son lo mismo, excepto que algunas veces los programas **shell** son, elaborados para contener más de programas estructurados que sólo líneas de comandos).

Este capítulo muestra la manera para pasar información a un programa **shell**, también como ejecutar comandos condicionales, como obtener datos provenientes desde el teclado durante la ejecución de un programa **shell**.

Todos los programas elaborados en **shell** pueden ejecutarse de dos maneras: puedes teclear los comandos dentro de un archivo, así, estos serán ejecutados cuando el nombre del archivo sea teclado, o puede introducirse el comando directamente sobre el **shell**.

Cuando se introduce un **shell** estructurado directamente dentro de otro **shell**, se puede teclear en la misma línea (y presionar **return** para ejecutarlos) o puede teclearse sobre varias líneas. Por ejemplo, podemos teclear las siguientes estructuras de dos formas.

Primero, en una línea:

```
if test -d/d1; then echo "/d1 es un directorio";fi
```

y en varias líneas:

```
if test -d/d1
then
   echo "/d1 es un directorio"
fi
```

Teclear el comando en una línea es simple para hacer un **shell**. Si tecleas el comando en varias líneas, recibirás un **prompt** secundario (el cual puedes definir en la variable PS2).

El **prompt** secundario es usualmente un ">". Así, al escribir el comando anterior en varias líneas, la pantalla podría verse como:

```
$ if test -d/d1
> then
> echo "/d1 es un directorio"
> fi
/d1 es un directorio
$
```

en donde "$" es el **prompt** primario y ">" es el **prompt** secundario del sistema.

Lo que es más, se pueden crear **shells** que provienen de programas tales como "note", "mail" y muchos editores tales como "vi", y ejecutar comandos **shell** dentro de esos comandos (**shell**). Ejecutar un **shell** dentro de otro programa es usualmente llamado "Forking" a **shell**. Esto puede ser útil si se escribe un programa y se desea probarlo, es necesario editarlo con el comando "vi", y levantar o ejecutar este **shell**, dentro del "vi" con la instrucción "sh" se sale, posteriormente ejecutar el programa para determinar si este trabaja correctamente, para abandonar o salir del **shell** sólo tecleamos "exit" o "ctrl-d", con esto regresamos al editor para realizar cualquier cambio.

2.5.5 Parámetros

Además de los parámetros del **shell**, se pueden crear parámetros propios. El formato para crear parámetros propios (user-created) es :

```
parameter=value.
```

Note que no debe haber espacios en blanco entre el parámetro, el signo "=" y el valor. Podemos crear estos parámetros mientras nos mantendremos en el **shell**, y estos parámetros serán de ayuda para salvar escrituras. Por ejemplo:

```
x=phantom.
```

Cuando se teclea en la instrucción anterior, se crea la variable "x" y se asigna el valor "phantom". Para accesar a la variable "x", necesitaremos preceder el nombre de la variable con un signo pesos ($). Así:

```
$echo $x
phantom
$
```

El comando **echo** escribe el valor de "x" en la pantalla. Un uso posible de parámetros es para signar una ruta larga a una variable, así no tendremos que teclear toda la ruta cada vez que se desee emplearla. Por ejemplo:

```
dir1=/users/hpux/davek/proyects/shell
```

así, para listar el contenido de este directorio, teclee:

```
ls $dir1
```

2.5.5.1 Empleo de parámetros en programas shell

Se pueden emplear parámetros dentro de los programas en **shell** de la misma forma. En una línea, define la variable con el formato anteriormente visto. Así cuando se quiera referir al valor del parámetro, se hará precediendo al nombre del parámetro con un signo "$".

Una ventaja del empleo de parámetro en un programa es que puede concatenar el parámetro fácilmente. Por ejemplo define un parámetro para que contenga la ruta del directorio.

```
dir2=/users/hpux/dave/proyects/memos
```

Al imprimir el contenido de un archivo en el directorio anterior, puede emplearse el comando **cat** como se muestra:

```
cat ${dir2}/junememo
```

Donde las llaves diferencian entre el parámetro y los caracteres siguientes a este y "junememo" es el nombre de un archivo (note que tenemos que incluir un slash antes del nombre del archivo o, "junememo" podría haber sido concatenado directamente a "memos" y podríamos haber recibido un mensaje de error).

Lo que ha ocurrido es llamado substitución de parámetros y será discutido posteriormente.

2.5.5.2 Substitución de parámetros

Cuando se quiera incluir el valor de un parámetro dentro de una línea ó instrucción, debes preceder al parámetro con un signo de pesos $. También seguir las convenciones:

${parameter}

El valor del parámetro entre llaves es sustituido por el valor de la variable. Se utilizan las llaves {} cuando el parámetro es seguido por una letra, un dígito, o alguna marca, la cual no es parte del parámetro. Ejemplo:

${dir1}123_FILE será sustituido el valor que contiene "dir1" y se le agregarán los caracteres 123_FILE.

${parameter:-word}

Si el parámetro es colocado, y no es nulo, el valor será sustituido. De otra forma será la palabra (word) la que lo sustituira. Ejemplo: ${dir1:-/usr/bin} si "dir1" es nulo, entonces "/usr/bin" será sustituido.

${paramenter:=word}

Si el valor del parámetro no es colocado, o es nulo, entonces coloca el valor a "word" y sustituye ese valor. Ejemplo: ${dir1:=/usr/bin} si "dir1" es nulo su nuevo valor es "/usr/bin".

${parameter:?word}

Hace lo mismo que "-" excepto que el programa **shell** será expulsado si el parámetro es nulo. y aparecerá, el mensaje "parameter null or not set" impreso en la pantalla. Ejemplo: ${dir1:?/usr/bin} llevará a cabo la sustitución con "/usr/bin" si "dir1" es nulo, y entonces se expulsará al **shell**.

${parameter:+word}

Si el parámetro es colocado y no nulo, entonces sustituye "word". De otra forma no sustituye nada.

2.5.5.3 Parámetros posicionales.

Cuando ejecutes un programa de un **shell**, puedes incluir parámetros en la línea de comandos. Cuando lo hagas, cada parámetro deberá ser separado por un blanco, como se muestra:

```
scopy file1 file2 file3
```

Donde "scopy" es un programa con tres parámetros. Cuando corre o ejecuta el programa en **shell**, se puede accesar el valor de esos parámetros (cada uno separado por un blanco) con parámetros posicionales llamados $0,$1,$2....$9, si la lista de valores excede 9 parámetros, los valores son colocados en el buffer, y accesar los valores con el comando **shift** (discutido posteriormente)

```
scopy personal archivoA
```

Tiene parámetros posicionales **$1** igual a "personal" y **$2** igual a "archivoA". El parámetro posicional "$0", es siempre el nombre del comando, para el ejemplo anterior es "scopy"

- Para conocer el número de parámetros posicionales emplear **$#**
- Si necesitas un parámetro el cual contenga todos los parámetros posicionales separado por blancos, emplear **$***.

Los parámetros posicionales son accesados dentro del cuerpo del programa. Cuando el programa es ejecutado, se van asignando los parámetros a valores sólo para el programa en ejecución. Aquí un ejemplo del programa empleando parámetros posicionales:

```
echo "searching for $1 in $2"
grep $1 $2
echo "done"
```

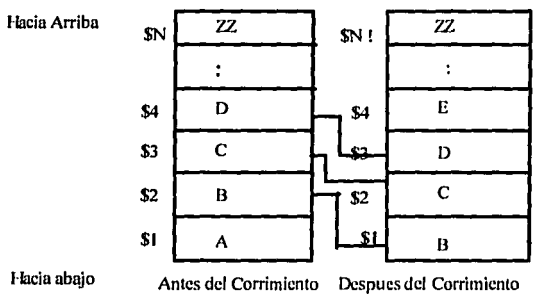
Este programa en **shell** espera dos parámetros posicionales. El primer parámetro es un comando en línea, y el segundo es un archivo. El comando **grep** busca en el archivo la ocurrencia de la cadena dentro de las líneas de caracteres que se encuentran en este.

En esta última sección vimos como accesar los parámetros posicionales mediante el empleo de números del $1 hasta $9. Sin embargo, si accesamos estos por nombre, debemos saber lo que se espera. En otras palabras, no podemos tener el parámetro posicional en un orden arbitrario ni más de nueve parámetros.

La operación **shift** auxilia para resolver los problemas con parámetros posicionales. Piensa en los parámetros posicionales como una pila con $1 en la base y $9 en la parte superior (si hay más de nueve parámetros, el resto deberá ser almacenado en la parte superior)

El **shift** removerá el valor de "$1", y reemplazará este con el valor de "$2", mueve el valor de "$3" a "$2" y así sucesivamente. Esto e como remover la base de la entrada de la pila y permitir a los valores caer una posición hacia abajo. Obsérvese la gráfica que representa la idea:

FIG 2.5 Corrimiento de parámetros posicionales.



Podemos emplear corrimientos en forma cíclica, lo cual discutiremos posteriormente o, puedes emplear este corrimiento secuencial como en el siguiente ejemplo llamado "list".

```
if[ $1=si ]
then
   shift
   cat $1
   exit
else
   shift
   echo "Archivo llamado $1 fue rechazado"
fi
```

Si el primer parámetro posicional "$1" es igual a "si", entonces el contenido del nombre del archivo (el segundo parámetro posicional) será listado. La primera ocasión "$1" es usado para la prueba, este

podría tener el valor "**si**". Después del corrimiento, el valor que era "**$2**" es desplazado a "**$1**", y podría ser el nombre del archivo. Para ejecutar este programa, teclear: **list** "**si** nombre_de_archivo" o, **list** "**no** nombre_de_archivo"

2.5.6 Programación avanzada.

Muchas veces el procesamiento secuencial en un programa no es suficiente. Necesitamos un mecanismo que nos permitirá repetir el mismo arreglo de comandos empleando un arreglo diferente de valores de cada uno de los parámetros. Para llevar a cabo esto con programas en **shell** puedes escoger entre 3 ciclos iterativos: (estructuras cíclicas): **for**, **while**, y until

El **for**. La estructura **for** permite ejecutar un arreglo de comandos, una vez por cada nuevo valor asignado a un parámetro. Observemos el siguiente formato

```
for     parámetro [in lista-palabras]
do      lista-comando
done
```

Donde el parámetro es cualquier nombre de "parámetro", "lista-palabras" es un arreglo de uno o más valores que son asignados a "parámetro", y "lista-comando" es un arreglo de comandos a ser ejecutados cada vez que el ciclo se lleve a cabo. Si la "lista-palabras" se omite (y también "**in**"), entonces se asigna al "parámetro" el valor de cada parámetro posicional.

El arreglo "lista-palabras" es una cantidad versátil en la estructura **for**. Esta puede ser una lista que se teclea específicamente (separada con blancos), o puede ser un comando **shell** (empleando apóstrofes), lo cual genera una lista. Ejemplo:

```
for i in `ls`
do
  cp $i /users/validar/$i
  echo "El Archivo $i fue copiado"
done
```

Este ejemplo asignará un archivo a la vez, procedente del directorio actual (se generan los valores a través del comando **ls**) para el parámetro "i". El comando **cp** del ciclo **for**, los copia en otro directorio, entonces reportará el éxito de la copia.

Empleando el "*" para generar una lista de todos los archivos e igualar directorios, en lugar de la primera línea del ciclo anterior que es "**for i in `ls`**", podrías usar; "**for i in ***"

```
for direc in /dev /usr /users/bin /lib
do
  num =`ls $direc|wc -l`
  echo "$num files in $direc"
done
```

Este ejemplo lista los valores que serán dados a "direc" en el ciclo. El comando **ls** lista cada uno de los directorios (parámetros) y asigna a el parámetro "num" el valor de las líneas contadas por el comando **wc -l**, para reportar el número de archivos en cada uno de los directorio listados respectivamente.

```
for;
do
  sort -d -o ${i}.srt $i
done
```

Por último, este ejemplo asignará en la variable "i" cada uno de los parámetro posicionales, respectivamente (la sentencia **in** fue omitida). Si los parámetros posicionales son nombre de archivos, el programa clasificará el archivo y colocará el resultado en un archivo que tiene el mismo nombre. Posteriormente un archivo no clasificado con extensión ".srt" es agregado a este. El mismo será entonces colocado en el siguiente parámetro posicional hasta que todos hallan sido accesado.

El **While.** La estructura **while** ejecuta repetidamente una secuencia de comandos bajo el siguiente format;

```
while comand-list1
  do comand-list2
done
```

Todos los comandos en "comand-list1" son ejecutados. Si el último de la lista es exitoso (indicando por una condición de salida u originada por el comando), entonces, se ejecutan posteriormente los comandos de "command-lis2". Así nuestro ciclo retornará para ejecutar "command-lis1" hasta que el último comando en la lista sea no exitoso, y entonces el ciclo **while** termina.

```
while [ -r "$1"]
do
 cat $1 >> composite
 shift
done
```

Este ejemplo, prueba el parámetro posicional, para saber si existe y es leible un archivo. Si este existe, el parámetro agregará el contenido de el archivo a "composite", corriendo los parámetros posicionales, lo que era "$2" es ahora "$1" y prueba el nuevo archivo. Cuando este no es leible, no hay más parámetros, o "$1" es nulo el ciclo **while** es terminado. (Nota: realizar la prueba sin dobles comillas (")) en el parametro de "$1", esta prueba responderá con un argumento de error dé sintaxis cuando "$1=**null**").

El Until. La estructura **until** es básicamente la misma que la estructura **while** excepto que los comandos son ejecutados en el ciclo hasta que la condición sea verdadera (en lugar de falsa como en el **while**), He aquí un ejemplo.

```
Until Comando_1
  do Comando_2
done
```

Si el "comando_1" no es exitoso, entonces se ejecuta el "comando_2". Cuando el "comando_1" es exitoso, el ciclo **until** termina. Vamos a utilizar la misma operación que en la estructura **while,** para ejemplificarlo:

```
until [ ! -r $1 ]
        do
```

```
    cat $1 >> archivo1
done
```

Notese que la condición es diferente en sintaxis a la del ciclo **while**. El signo de negación **!** niega la condición. Se ejecutará el ciclo hasta que la condición sea verdadera (o exitosa). En el **while** se ejecutan los comandos mientras la condición sea verdadera.

El **Case**. La estructura **case** es una expansión de la estructura **if**. Se tiene una condición, la cual podría tener varias respuestas posibles, pueden concatenarse muchos **if's** o usar la estructura **case**

```
case parámetro in
        valor1 [ | valor2 .. ]) comando_1;;
        valor2 [ | valor3 ...]) comando_2;;
        .
        .
esac
```

La primer línea pregunta si "parámetro" es igual a una de las siguientes condiciones de la lista que nos muestra todos los posibles valores que puede tomar "parámetro". Cada una de las líneas contienen una constante (o valor de parámetro). Los corchetes "[]" son para referenciarnos a otros valores que pueden ser válidos, y la barra vertical es una "o". Por último los valores son seguidos por ")", y después el o los comandos a ejecutar, estos terminados con doble ";", con la instrucción **esac** se finaliza la estructura del **case**. Un ejemplo nos puede ilustrar mejor su estructura

```
case $i in
        -d | -r ) rmdir $dir1
                echo " opción -d o -r ";;
        -o)     echo " opción -o ";;
        -*)     echo " respuesta incorrecta ";;
esac
```

El primer parámetro posicional es comparado con varias constantes, si "$i" es "-d" o "-r", entonces es borrado "$dir", y se manda un mensaje a la pantalla, si "$i" es "-o" sólo se manda un mensaje. El último valor empleado es el caracter comodín "*" en donde cualquier otra condición que no sea alguna de las anteriores entraría aquí, ejecutando un mensaje de error.

Existen muchos otros comandos pero sólo se mencionaron los de mayor importancia para nuestro desarrollo junto con las características que comprenden estos.

El capítulo muestra las características más importantes de las utilerías usadas en el desarrollo del sistema (**S.E.R.F.**) junto con su estructura y su mejor aprovechamiento aunado a ésto la importancia que éstas herramientas tienen. En los capítulos siguientes se verá la utilización de los mismos por el momento ya tenemos un panorama general de estos tópicos. Como complemento en el capítulo siguiente veremos la **BASE DE DATOS ORACLE** y el **SISTEMA OPERATIVO UNIX**.

AMBIENTE DE DESARROLLO

3.1 Introducción.

Este capítulo nos introduce al Sistema Manejador de la Base de Datos Relacional **ORACLE (RDBMS ORACLE)** y al Sistema Operativo **UNIX**, para conocer de manera general la estructura de estos y la forma en que interactúan. Conoceremos también la plataforma donde el **SERF** fue desarrollado y donde actualmente es operado (hardware).

3.2 Manejador de la Base de Datos Relacional Oracle

Conforme pasan los años los sistemas administradores de Base de Datos Relacionales han llegado a ser los más aceptados para manejar datos. Los sistemas Relacionales ofrecen beneficios tales como:

Fácil acceso hacia los datos.
Flexibilidad en el modelo de datos.
Reducción en el almacenaje y redundancia de datos.
Un alto nivel de manipulación de datos.

Como la tecnología asociada con los sistemas administradores de Base de Datos Relacionales han tendido a crecer rápidamente, en recientes años la ayuda de las Bases de Datos Relacionales se han vuelto evidentes y con demasiada audiencia.

El crecimiento de las tecnologías relacionales ha tenido más demanda en los **RDBMS** (Sistema Manejador de Base de Datos Relacional) para ambientes entre PC's y CPU's de alta seguridad y en ocasiones muy sofisticados.

ORACLE Corporation fue la primer compañía en brindar un verdadero sistema relacional **DBMS** comercialmente, y ha continuado mostrando innovaciones en el campo de los **RDBMS**. Las estrategias de **ORACLE** Corporation es ofrecer un **RDBMS** que sea portátil, compatible y conectable. Resultando ser una exitosa herramienta para los usuarios.

El **RDBMS** de **ORACLE** es de una alta funcionalidad, un sistema tolerante de fallas en la Base de Datos, especialmente diseñado para transacciones de procesos en línea y amplias aplicaciones en la Base de Datos.

El lenguaje "SQL" como manajedor de **ORACLE RDBMS** llamado "Lenguaje Secuencial de Datos" **(SQL)**. Es usado para mejorar las actividades en la Base de Datos. El lenguaje "SQL" es simple, permite al usuario accesar a los datos fácil y rápidamente, es muy poderoso, ofreciendo la capacidad y flexibilidad que se requiere.

El lenguaje "SQL" fue desarrollado y definido por investigaciones IBM y ha sido reconocido por la American Standars Institute (ANSI) como el lenguaje estándar para sistemas administradores de Base de Datos Relacionales.

El lenguaje "SQL" implementado por **ORACLE** Corporation es el lenguaje estándar de "SQL" con algunas mejoras **(SQL*PLUS)**.

Las declaraciones de "SQL", son declaraciones que manipulan datos de la **BD**. Las declaraciones "SQL" (comandos) pueden ser usadas en muchos ambientes, incluyendo los productos de **ORACLE** como "SQL*forms" utilerías de "SQL*dba" y programas en Pro*C.

Las declaraciones en el lenguaje "SQL" son divididas en 4 categorías.

Consultas.
Lenguaje de Manipulación de Datos (DML).
Lenguaje de Definición de Datos (DDL).
Lenguaje de Control de Datos (DCL).

Estas 4 categorías son algunas veces simplificadas en dos (Las dos primeras como **DML** y las dos últimas como **DDL**). Cada categoría tiene diferentes aplicaciones sobre la operación de la **BD**.

Las consultas son declaraciones que recuperan datos, de la forma deseada. Las consultas usualmente empiezan con la palabra reservada **SELECT**, y no modifican los datos.

Las declaraciones **DML** son usadas para cambiar los datos de la **BD** con los siguientes comandos **INSERT, UPDATE** y **DELETE**; insertar, modificar y borrar datos respectivamente.

Las declaraciones **DDL** son usadas para definir y mantener objetos de la **BD**, con los comandos **CREATE, ALTER** y **DROP**; crear, modificar, y borrar la estructura de un objeto de la **BD** en ese orden.

Las declaraciones **DCL** son usadas para controlar los accesos a la **BD** con los comandos **GRANT** y **REVOKE**; para otorgar y suprimir permisos al usuario.

El sistema administrador de la Base de Datos Relacional **ORACLE RDBMS** es el producto central de **ORACLE**. Este incluye el manejador de la **BD** y algunas herramientas con el propósito de asistir a usuarios administradores **(DBA's)** en el mantenimiento, monitoreo y uso de datos.

La esencia de **RDBMS** es el "kernel", el cual maneja las siguientes tareas:

Manejo del almacenamiento y definición de datos.
Control y limitaciones de acceso de datos y concurrencias.
Permite respaldar y recuperar datos
Interprete de "SQL" y "PL/SQL".

AMBIENTE DE DESARROLLO

Una función del "kernel" es optimizar los recursos de la **BD**. Para lo cual examina la ruta de accesos del usuario y localiza la ruta más corta, optimizando las ejecución de la consulta y los recursos del equipo.

Las utilerías y aplicaciones incluídas en el producto **ORACLE RDBMS**, se muestran en el siguiente esquema.

Herramientas para Desarrollo de Aplicaciones	**Aplicaciones Financieras**	**Aplicaciones de Prod. Empresarial.**
CASE*Dictionary	ORACLE Gereral Leger	
CASE*Designer	ORACLE Purchasing	ORACLE Casting
CASE*Generator	ORACLE Payables	ORACLE Bill of Material
ORACLE Graphics	ORACLE Assets	ORACLE Engineering
ORACLE*Terminal		ORACLE Master Scheduling
SQL*Plus		
SQL*Forms		Herramienta de Soporte
SQL*Menu		para Toma de Decisiones
SQL*ReportWriter		Easy*SQL
SQL*TextRetrieval		SQL*Calc
Pro*(Precompiladores)		SQL*Qmx
ORACLE Card		Data Query

Utileria de la Base de Datos Oracle	**SQL Oracle Servicios y Integridad del Diccionario de Datos**	**Herramientas para la Integración de Productos**
SQL*DBA		ORACLE for 1-2-3
Export/Import		ORACLE for dBASE
SQL*Loader		ORACLE for 4th Dimension

Oracle Valor-Agregado Herramientas de Diseño	**Productos de Conectividad**	**Herramientas Automatizadas para Oficina**
A I	SQL*Net	ORACLE*Mail
CAD/CAM/CIM/CAE	SQL*Connect	ORACLE*Aler
Retail	ORACLE Palm link	ORACLE Post Card
Dristribution		ORACLE CoAuthor
Project Manager		

A Continuación la figura 3.1 denota la estructura organizacional del **RDBMS** y la figura 3.2 las tareas que son responsabilidad del administrador de la Base de Datos, necesarias para tener un mejor control de estas.

Figura 3.1 Arquitectura de Oracle

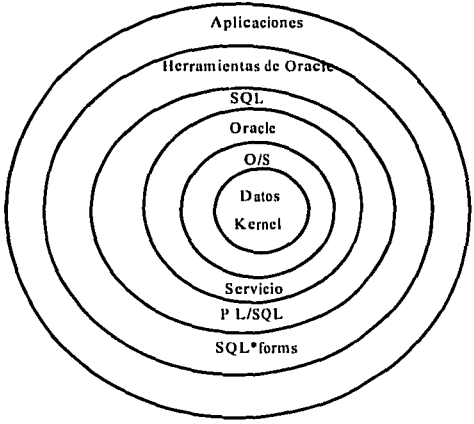


Figura 3.2 Administración de la Base de Datos.

3.2.1 Arquitectura de Oracle

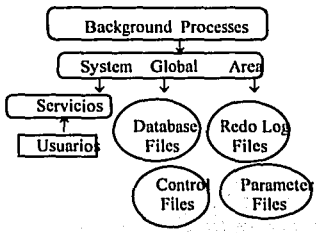
Aquí veremos la arquitectura física y de proceso que maneja **ORACLE,** como son:

Descripción de la estructura de los archivos físicos en la Base de Datos **ORACLE**

Descripción de las funciones que realiza la estructura de la memoria, que incluye el **System Global Area (SGA)** y el **Program Global Area (PGA).**

Y una breve explicación del concepto del proceso de una "INSTANCIA" en **ORACLE.**

Arquitectura de **ORACLE** (Estructura, Memoria y Procesos)



Una Base de Datos esta compuesta de **control files, data files** y **redo log files,** estos nombrados dentro de los **parameter files.**

Data Files
Contienen todos los datos de la Base de Datos, estructuras lógicas, semejantes a tablas e índices, que son almacenados físicamente en los **data files.**

Redo Log Files
Transacción que se realiza con muchos registros; contienen información de todo cambio hecho en la Base de Datos para su recuperación

Control Files
Registros que guardan la estructura física de la Base de Datos (mantiene una imágen de la Base de Datos)

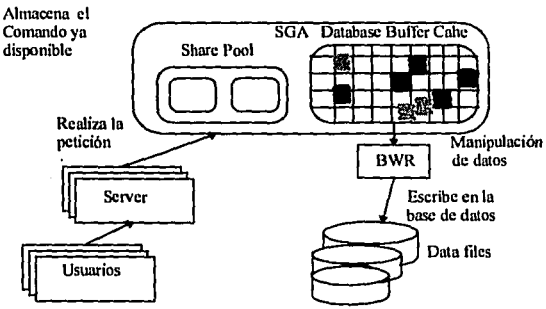
Cada vez que en **ORACLE** se inicializa una "INSTANCIA", los Procesos de **backgroud** localizan al **System Global Area (SGA),** estos procesos son inicializados por **ORACLE.** La combinación de **buffers** de memoria y procesos de **background** son llamados en **ORACLE** "INSTANCIA".

El **SGA** es un grupo de particiones de **buffers** de memoria compartidos en **ORACLE** por una sóla "INSTANCIA". Los procesos de **background** asincronamente realizan distintas tareas en nombre de todos los usuarios de la Base de Datos. Los parámetros de los archivos determinan las características de la "INSTANCIA" estos localizados en el parameter file.

El acceso a los datos. Antes que se pueda accesar a los datos, un proceso de server debe colocar los datos en el **database buffer cache.** Cuando se modifica un block de datos estos son escritos sobre el disco por un proceso de **background,** llamado **Database Writer** (DBWR). El proceso de

confirmación de "SQL", es un proceso del server que usa parte de la memoria del **SGA**, como se muestra en la figura 3.3

Figura 3.3 Procesos de background



El proceso de un usuario es creado cuando este corre un programa en una aplicación de **ORACLE**, y a su vez, crea un proceso de "server" que manda un requerimiento para conectar el proceso del usuario.

El proceso del "server" comunica al usuario. Esto se produce cuando el proceso realiza el requerimiento para conectar en **ORACLE** el proceso del usuario. Estos son los pasos que ejecuta el proceso del "server":

1) Checa la sintaxis y ejecuta las "INSTANCIA" de "SQL".
2) Lee los datos del block del disco dentro de la parte del **database buffers**
3) Regresa el resultado de las sentencias de "SQL" que realizan los procesos del usuario.

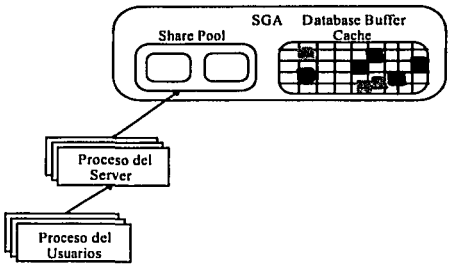
Todas las sentencias de "SQL" son realizadas implícitamente por el proceso del "server", este utiliza tres fases.

- Comprueba la sintaxis
- Consulta en el diccionario de datos, para la localización de los objetos.
- Seguridad, privilegios, y trayectoria de acceso.
- Determina sus fases, ó plan de ejecución.

SQL > select nombre from empleado;

Las tres fases son almacenadas en el share "SQL" area. Múltiples procesos del "server", pueden ejecutar al mismo tiempo las tres fases. El tamaño del **share pool** es determinado por el parámetro **shared_pool_size.** La figura 3.4 muestra como se ejecutan estos procesos en la sentencia **SELECT. ORACLE** usa cursores dentro de los registros para confirmar el estado en que se encuentra la información.

Figura 3.4 Ejecución de la Sentencia SELECT



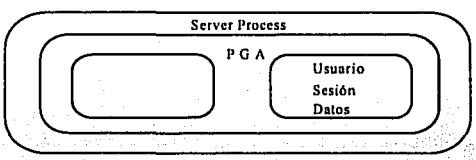
Como se ejecuta la sentencia **SELECT**. Primero se aplica las tres fases sobre el **data buffers**, posteriormente ejecuta la lectura física o lógica de entrada/salida, y finalmente trae los renglones que contienen los datos que fueron seleccionados por la sentencia **SELECT**.

El **shared sql area** contiene información usada para la ejecución especifica de las sentencias de "SQL" la ejecución de este proceso identifica la sentencia **SELECT** de "SQL" como parte de la información. Este forma parte del **shared pool**, que a su vez es parte del **SGA**. El contenido del **shared pool** es:

- El texto de las sentencias de "SQL" o PL/SQL.
- Información de la sintaxis de las sentencias de "SQL" o de PL/SQL.
- Programa de ejecución para sentencias de "SQL" y PL/SQL
- Diccionario de datos. El cache contiene información de los renglones en el diccionario de datos

El **Program Global Area (PGA)** es una región de memoria que contiene datos e información de control para un solo usuario o proceso del server. El **PGA** es localizado por **ORACLE** cuando un usuario realiza un proceso de conexión a la Base de Datos **ORACLE** y esta sesión es estabilizada, la figura 3.5 muestra la configuración del **PGA**.

Figura 3.5 Configuración del PGA



En el **PGA** se crea un espacio en forma de pila que guarda en memoria para tener sesiones variables en forma de arreglos.

- La sesión del usuario es un dato de memoria adicional en la misma.
- El **PGA** es reescribible y no-reusable.

El **database buffer cache** tiene copia de los blocks de datos leídos desde el disco. Los **buffers** en el **database buffer cache** forman parte de cada uno de los procesos que efectúen los usuarios de **ORACLE**, conectados al mismo tiempo en una "INSTANCIA".

El tamaño de los blocks es determinado por el parámetro **db_block_size**. El número de blocks del cache en memoria es determinado por el parámetro **db_block_buffers**.

El **database buffer cache** es organizado en dos listas: la lista usada y la recién usada **(LRU)**. La lista usada contiene las modificaciones dentro del **buffer** que nos sirve para no tener que rescribirlas en disco

El contenido de la lista **LRU** (la cual es un algoritmo)

Buffers Libres
Buffers en uso
Buffers Usados

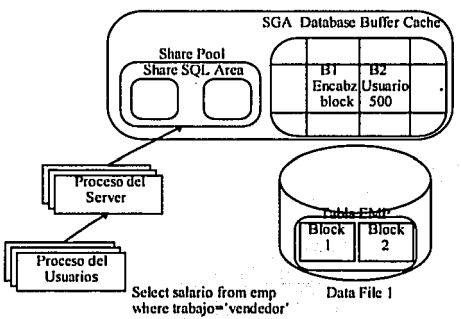
Cuando un proceso del "server" necesita leer un block de datos desde el disco, este lo inserta en el **database buffer cache** como se muestra en la figura 3.6, esto realiza lo siguiente:

Busca en la lista **LRU**
Cierra los **buffers** libres
Mueve los **buffers** usados hacia las listas usadas

El proceso del "server" se detiene cuando:

- Encuentra un **buffer** libre,
- Cuando él ha reconocido un número específico de **buffers** sin encontrar un **buffer** libre.

Figura 3.6 Almacenamiento de Blocks



3.2.2 Segmentos de Rollback

Un **segmento de rollback** es una porción de la **BD** para la ejecución de los registros en una transacción, esto los pueden regresar a su estado anterior, o deshacer los cambios realizados.

Cada Base de Datos contiene uno o más **segmentos de rollback**, el uso de los **segmentos de rollback** nos sirven para:

- Comprobar consistencia en la lectura
- Recuperación de transacciones por medio de la sentencia de "SQL" **rollback**
- Recuperación de transacciones en la Base de Datos

En que momento se ejecuta o se necesita ejecutar un **rollback**

- Cuando se regresa una sentencia por alguna otra sentencia o error de interrupción
- Se regresa hacia una sentencia ejecutada o hacia un **savepoint.**
- Se regresa debido a una terminación anormal en un proceso
- Regresar todas las transacciones pendientes durante la recuperación automática de una "INSTANCIA".

Cada transacción debe de estar asignada a un **segmento de rollback.** Una transacción puede estar asignada automáticamente a una **segmento de rollback,** esto se basa en el criterio de seleccionar al **segmento de rollback** de menor uso en ese momento.

3.2.3 Redo Log

Su principal función es mantener las transacciones confirmadas hacia la BD. Y mantener información que probablemente cambiará en la **BD.**

Los blocks de **rollback** contienen la identificación de las transacciones, la dirección de la columna, cual es la forma del archivo, su block, renglón y columna, del valor anterior del dato.

- Un segmento de **rollback** es un objeto que es usado para salvar datos anteriores
- **ORACLE** provee una lectura consistente de esta manera, muestra una imagen del dato, para todos los usuarios.

Consistencia en la lectura

Durante todo el tiempo del procesamiento de una sentencia,, **ORACLE,** regresa una fotografía de los datos de la tabla, esto consiste en tomar en un tiempo dado una copia de la tabla en el momento en que la sentencia se ejecuta. El **RDBMS** garantiza que por encima de la ejecución de una sentencia, todos los usuarios pueden ver el mismo valor mientras algún otro usuario simultáneamente salva los cambios realizados en la tabla que se esta observando.

ORACLE solamente regresa un dato salvado o cambio previamente realizado por el usuario actual o el que realizó la modificación.

3.2.4 El proceso DBWR

El **DBWR** administra el **database buffer cache,** de esta manera los procesos de los usuarios siempre pueden encontrar buffer libres estas son las funciones del **DBWR**:

- Escribe todos los cambios de los buffer hacia los **data files**
- Utiliza el algoritmo **LRU** para guardar la mayor parte de los blocks recien usados en memoria
- Difiere la escritura para optimizar las entradas y salidad (I/O)

El **DBWR** escribe en los buffers usados del disco cuando:

- Cuando la lista usada alcance su mayor umbral.
- Un proceso explora un número especifico de buffers en la lista del **LRU** sin encontrar un buffer libre.
- Ocurre un tiempo fuera.
- Sucede un **checkpoint.**

Procesos de **background**

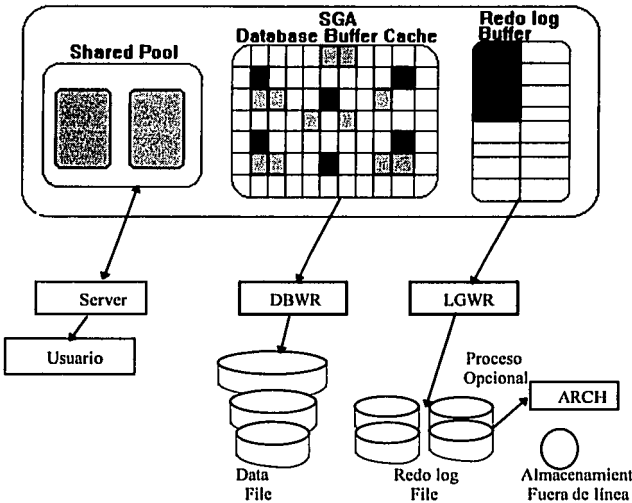
	DBWR Database Writes	(Se encarga de escribir en la **BD**)
SGA	**LGWR** Log Write	(Se encarga de escribir en los **Redo Log's**)
	SMON System Monitor	(Monitorea la ejecución de transacciones **BD**)
	PMON Proces Monitor	(Monitorea la respuesta de la **BD**)

3.2.5 Transacciones con los log

En **ORACLE** todos los cambios ralizados se registran en los **redo log buffer** de la **BD.**

Un proceso de **background (LGWR)** escribe información en los **redo log buffer** dentro del disco. Otro proceso de **background**, el Archiver **(ARCH)**, si se desea puede ser inicializado para respaldar información en linea de los **redo log**, un ejemplo es la figura 3.7.

Figura 3.7 La Base de Datos y el Proceso del LOG WRITE



Redo log buffer es un cojunto de buffers en forma circular que contienen información acerca de los cambios realizados en la Base de Datos. Esta información es almacenada dentro de los **redo**. Los **redo log buffers** almacenan todos los cambios ralizados dentro de los **redo log** de la **BD**.

Los **redo log's** son usados para reconstruir o regresar los cambios hechos en la **BD** cuando necesita realizar una recuperación.

Pasos importantes en la ejecución de los **redo log buffers**, dentro de su operación de actualización.

1.- Forma blocks de la **BD** y los introduce en los **database buffer cache**
2.- Adquiere blocks de rollback y los introduce en los **database buffer cache.**
3.- Poner candados en forma exclusiva en cada renglón cuando estos son cambiados.
4.- Adiciona una imagen anterior y una imagen posterior de los blocks dentro de los **redo log buffer.**
5.- Salva los datos recuperados dentro de los blocks de los segmentos de **rollback.**
6.- Ejecuta los cambios en los blocks de datos.

El tamaño de los **redo log buffer** son determinados en el parámetro **"log_buffer"**.

3.2.6 El proceso LGWR.

El **Log Writer (LGWR)** proceso que escribe en los **redo log** dentro del disco.

El proceso **LGWR** escribe de los **redo log buffer** a los **redo log files** cuando:

- Ocurre un **commit** (salvar los cambios).
- El **redo log buffer** alcanza un almacenamiento mayor a una tercera parte de su umbral.
- Cuando el **DBWR** no ha limpiado los blocks de los buffers por un **checkpoint**.
- Porque ha ocurrido un tiempo fuera (tirar la Base de Datos).

La escritura de una "INSTANCIA" es llevada a cabo por un sólo **redo log** La confirmación de un **commit** se lleva a cabo, hasta el último registro que tiene la transacción en los **redo log files**.

Muchos **commits** son implícitos en un archivo, y en un porcentaje muy bajo son **commits** por E/S. La transacción **commit** es utilizada para hacer cambios permanentes en la **BD**.

1) El usuario puede ejecutar un **commit.**
2) El **commit** en un registro es puesto en los **redo log buffer.**
3) El **LGWR** manda muchos **redo log buffers** a los actuales **log file** cuando utiliza para escribir multi-block. Si es posible.
4) El usuario es notificado cuando la transacción ejecuta un **commit.**
5) Los candados son utilizados en los datos y blocks de **rollback.**
6) Los blocks de datos son marcados como ya usados.
7) **DBWR** eventualmente realiza escritura en el disco de los block de la **BD.**

Los **redo log files** registran todos los cambios hechos en la Base de Datos, y los usa para recuperación de datos, los **redo log files** reflejan, la misma información que es escrita en los múltiples **redo log files** en línea.

Los **redo log files**, son escritos de una manera cíclica, y deben ser como mínimo dos grupos de **redo log.**

Se recomienda para los **redo log files** una configuración que sea como mínimo de dos miembros por grupo de **redo log**, para que se pueda hacer una copia o imagen de los **redo log files.**

Todos los miembros de los grupos de los **log files** contienen la misma información, estos son actualizados en forma simultánea. Cada grupo debe de contener el mismo número de miembros como los otros grupos, la imagen o copia de los **redo log files** protege contra perdidas de algún **redo log file.**

Qué son y cómo trabajan los **log switch.**

- Un **log switch** ocurre cuando en **ORACLE** cambias un **redo log** por otro
- Un **log switch** ocurre cuando el **LGWR** ha llenado un grupo de **redo log**
- Un **log switch** puede ser forzado por el **DBA** cuando sea necesario archivar al actual **redo log.**
- Un **log switch** se presenta cuando se ejecuta un shutdown de la **BD.**
- Automáticamente en un checkpoint ocurre un **log switch.**

En un **log switch** al actual **redo log** se le asigna un número secuencial de **log**, identificando la información almacenada en este **redo log** y además lo usa para su sincronización.

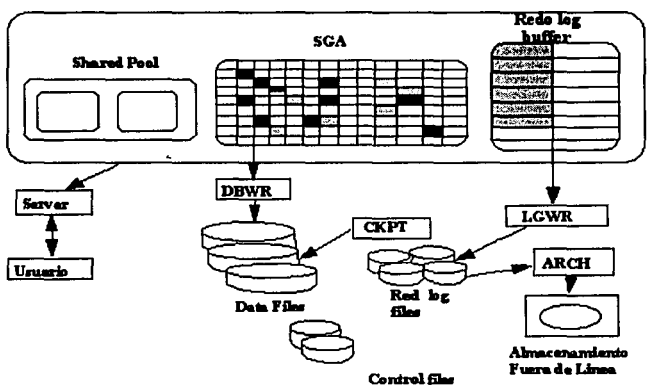
3.2.7 Qué es un checkpoint y cómo se utiliza.

Durante un **checkpoint DBWR** escribe todos los buffers utilizados en el **database buffer cache** hacia el disco. Esto garantiza que todos los blocks de datos modificados desde el previo checkpoint son en ese momento escritos en el disco.

Un **checkpoint** ocurre en cada **log switch**, en un especifico número de segundos antes del último **checkpoint** de la **BD**, esto sucede cuando un número predeterminado de blocks de los **redo log** tienen que ser escritos en el disco desde el último; **checkpoint** en el **shutdown** de una "INSTANCIA", cuando es forzado por el **DBA**, y cuando un **tablespaces** esta fuera de línea.

Durante un **checkpoint** y antes de un **log switch**, **LGWR** realiza la actualización de los headers de la **BD**, y control file, a menos que el proceso del **checkpoint** haya sido levantado con el parámetro **log_checkpoint_timeout** el cual determina el intervalo de tiempo antes de que ocurra otro **checkpoint**. El parámetro **log_checkpoint_intervale**, determina el número de blocks que nuevamente llenará los **redo log files** necesarios para inicializar un **checkpoint**, la figura 7.8 muestra la manera en que trabaja el **LGWR** y **DBWR**.

Figura 3.8 La Base de Datos y el proceso de checkpoint



En los **checkpoint** aseguramos que todas las modificaciones en los buffers de la **BD** sean escritos en los archivos de ésta. En los archivos de la **BD** se marca su actualización en el momento en que se integren estos y el **checkpoint** es registrado en el **control file**.

Para la recuperación de una "INSTANCIA" se requiere necesariamente sólo aplicar los cambios desde el último **checkpoint**.

Los **redo log files** en línea permiten ser rehusados para garantizar que todos estos cambios sean almacenados en los **redo log** file además son escritos en el apropiado **data file**.

Los **checkpoint** no necesitan parar su actividad, ni afectan transacciones actuales.

Los procesos de los **checkpoint**.

- La actualización de los "headers" de los datos y los **control file** debe ser completa antes de que ocurra un **checkpoint**.
- Una frecuencia mayor de los **checkpoint** nos ayuda a reducir el tiempo que se necesita para recuperar la falla de una "INSTANCIA", y a mejorar el **performances** del sistema.
- El proceso del **checkpoint** es habilitado a través del parámetro **checkpoint_process**.

3.2.8 El proceso de archivar (ARCH)

El proceso de archiver (ARCH) copia en línea los **redo log files** hacia un dispositivo de almacenamiento designado.

En el proceso de **archiver**, los **redo log files** son copiados hacia cintas o discos por medios de recuperación externos. El **ARCH** opera en línea y se ejecuta solo cuando en un grupo de **log files** ocurren cambios.

El **ARCH** es opcional y sólo se presenta cuando automáticamente **archiving** es habilitado o cuando se realiza en forma manual. El **ARCH** va a escribir hacia un dispositivo de almacenamiento que puede ser un disco o cinta magnética.

Para **ORACLE** una secuencia de declaraciones de "SQL" son una unidad lógica de trabajo. También son llamadas transacciones.

Comenzar una declaración, es inicializar una sesión o terminar una transacción previamente inicializada.

Finalizar una declaración se lleva a cabo por la declaración **commit** y la declaración **abort**.

Una declaración realizada, se ejecuta por la setencia **commit** de "SQL", o por las sentencias **DDL** "Lenguaje de Definición de Datos" (por ejemplo **DROP, CREATE, ALTER, GRANT**) las cuales causan un **commit** implícito, y normalmente también cuando el usuario sale del programa, con **logout** (salir) de **ORACLE**, el cual también ejecuta un **commit** implícito.

Una declaración abortada, se realiza cuando, la sentencia **rollback** de "SQL" se ejecuta, o cuando el usuario trunca un requerimiento, o alguna salida anormal de algún programa sin **logout** de **ORACLE**. También por el fracaso de un proceso, o falla de un disco.

Control de transacciones. Por default, si una declaración de **ORACLE** falla, los cambios junto con la declaración se recuperan, antes que se complete la transacción.

Tres ejemplos de Transacciones:

1. Crear un índice único en la tabla DEPTO.
2. Insertar renglones dentro de la tabla.
3. Consultar la tablas desplegando los renglones.

Ejemplo 1)

```
SQL> create unique undex ind_dept on depto (deptno);
     indeex created.
```

Ejemplo 2)

```
SQL> insert into dept values (10, 'INFORMATICA', 'CARPIO');
SQL> commit;
```

Ejemplo 3)

```
SQL> select * from dept:
     n_dept    nom_dept            lugar
     --------  ------------------  ----------
     10        INFORMATICA         CARPIO
SQL> exit
```

3.2.9 Los procesos PMON y SMON.

El **Process Monitor (PMON)** y el **System Monitor (SMON)** recuperan recursos de la **BD**, cuando ya no son necesitados por el usuario.

El **PMON** limpia cuando una conexión termina anormalmente, recupera las transacciones no realizadas o salvadas, realiza candados de ayuda para cuando se ejecuta un proceso, libera recursos del **SGA** que se utilizan por algún proceso que falló, detecta los **deablocks** y automáticamente ejecuta la recuperación de la transacción respaldada.

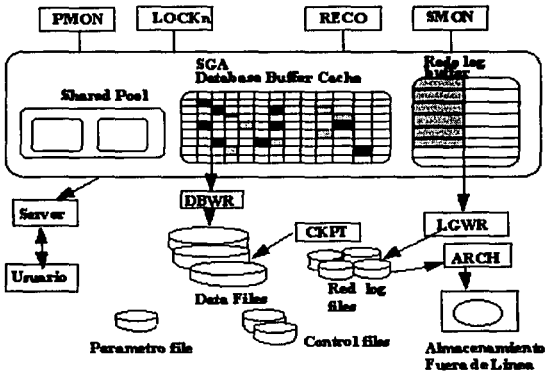
El **SMON** realiza automáticamente la recuperación de una "INSTANCIA", recupera el espacio de la tabla al realizar un **sort**, cuida todos los procesos de la Base de Datos y recupera los segmentos temporales que son utilizados, por una transacción.

Los **RECO** y **LCKn** son utilizados en la Base de Datos Distribuidas

El **Recoverer Process (RECO)** resuelve fallas envueltas en una transacción distribuida.

Los **Lock Process (LCKn)** realizan en una "INSTANCIA" bloqueos internos en paralelo al servidor del sistema, la figura 3.9. muestra la forma en que interactúa la **BD** y su proceso de **background**

Figura 3.9 Configuración global de ORACLE



El archivo de **control file,** es un archivo pequeño en binario que describe la estructura de la **BD.**

La finalidad del **control file.** Es contener la información necesaria de todos los archivos de la **BD,** y **log file,** así como el nombre de la Base de Datos, lo cual es necesario para poder abrir y accesar la Base de Datos, también nos ayuda a sincronizar la información necesaria para una recuperación. Toda esta información la almacena dentro del **control file.**

Es recomendado configurar como mínimo dos **control file** que estén en diferentes discos.

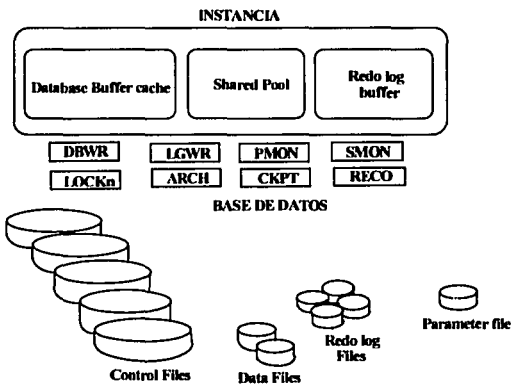
El parámetro "**control_file**" identifica los **control file** activos.

La arquitectura de **ORACLE** incluye estructura física, estructura de la memoria y procesos como se muestra en la figura 3.10.

En **ORACLE** una "INSTANCIA" contiene el **SGA** y los procesos de **background.**

En **ORACLE** la Base de Datos contiene todos los **data files, redo log files** y **control files.**

Figura 3.10 UNA INSTANCIA Y LA BASE DE DATOS



El **Sistema Manejador de Base de Datos Relacional (RDBMS)** de **ORACLE** requiere una asignación fija de espacio para los datos y programas.

Los recursos tales como **data files** son áreas de almacenamiento reservadas en disco para que solo las utilice **RDBMS** en almacenamiento de datos, así **RDBMS** solo reconoce el espacio.

3.3 Estructuras lógicas y físicas.

Estructura física. Los archivos dentro del Sistema Operativo son almacenados sobre áreas de Hardware (cintas magnéticas, disquetes, o discos duros) un archivo en el Sistema Operativo ocupa un espacio en disco. Algunos archivos son requeridos para que se ejecute el manejador de la **BD ORACLE.**

Ocasionalmente se puede tener interés acerca de las estructuras físicas, sólo para casos como por ejemplo: para mejorar el rendimiento del tiempo de E/S en la **BD** o para conocer el contenido del disco.

Las estructuras lógicas también son asignaciones de espacio en disco, pero sus límites son independientes al espacio físico. Un ejemplo de espacio son las tablas, las cuales se podrán extender hasta el tamaño definido para esta y no más allá. Junto con las tablas, los **tablespaces** también son estructuras lógicas.

La mayoría de los usuarios de la **BD** se interesa más por las estructuras lógicas que por las físicas, ya que estas tienen una mayor relación con el almacenamiento de los datos, por ejemplo: La rapidez para desplegar los datos seleccionados en una tabla. Esto depende de en que **tablespaces** este la tabla y si tiene o no creado un índice.

3.3.1 La herramienta SQL*DBA

La herramienta **SQL*dba** es usada para levantar y tirar la **BD** y dar mantenimiento a ésta.

A) Pasos para levantar la Base de Datos.
1.- Levantar la "INSTANCIA".
2.- Montar la Base de Datos.
3.- Abrir la Base de Datos.

B) Pasos para tirar la Base de Datos
1.- Cerrar la Base de Datos.
2.- Desmontar la Base de Datos.
3.- Tirar la "INSTANCIA".

Cada que se levanta una "INSTANCIA" es leído el archivo init.ora, el que como ya se vió contiene los parámetros de la **BD**, y nos crea el **SGA** que esta asociado con la memoria de **ORACLE**.

SQL*DBA es una herramienta interactiva utilizada normalmente por el Administrador de la Base de Datos (**DBA**).

Es una herramienta que nos sirve para administar los recursos de la **BD** y monitorear los procesos. Los comandos **startup** y **shutdown** son utilerías de **SQL*DBA** para levantar y tirar la **BD** respectivamente, estos comandos contienen múltiples opciones, las cuales utilizará el **DBA** según lo que se desee hacer.

SQLDBA> startup mount	Montar la **BD** sin abrirla
SQLDBA> shutdown immediate	Tira la **BD** sin cerrar procesos incompletos

Esta es una lista de los commandos más comunes en la manipulación de la **BD**.

CREATE	INSERT	SHOW	CONNECT	ALTER
UPDATE	IMP	SET	GRANT	SELECT
EXP	SPOOL	REVOKE	DELETE	STARTUP
ROLLBACK	DROP	COMMIT	SHUTDOWN	SAVE

A continuación veremos las tablas, sinónimos, permisos, índices, espacio y usuarios del Sistema de Estados de Resultados Financieros (**SERF**) y como un ejemplo la utilización de los comandos anteriores y estructura de la Base de Datos.

3.4 Creación de la estructura del sistema en ORACLE

Creación de tablespaces.

```
system/manager
drop tablespace PARTICION9 including contents
/
drop tablespace PARTCION19 including contents
/
create tablespace PARTICION9
datafile '/transfer/db9/dbd9.scf                size 30M
/
create tablespace PARTICION19
datafile '/transfer/db91/dbi9.scf               size 8M
/
```

Creación de las tablas

```
tj30071/op_serf
drop table pasivo;
drop table presp;
drop table pre_adpt;
drop table puesto;
drop table rsaldo;
drop table saldo;
drop table saldoi;
drop table saldol;
drop table sal_219i;
drop table sal_219l;
drop table sal_392i;
drop table sal_392l;
drop table solct;
drop table tab_temp;
drop table temp;
drop table tab_temp;
drop table grupo;
/
create table pasivo
(
CVE_CTRO            NUMBER(3),
 CVE_DPTO           NUMBER(5),
 CVE_ECOM           NUMBER(5),
 CVE_RENG           NUMBER(3),
 CVE_PROG           CHAR(2),
 ING_EGR            CHAR(1),
 IMP_PRE            NUMBER(15,2),
 CVE_ANIO           NUMBER(2),
 CVE_MESS           NUMBER(2))
tablespace PARTICION9
storage (initial 456k next 50k maxextents 25 pctincrease 0)
/


create table presp
(
 CVE_CTRO           NUMBER(3),
 CVE_DPTO           NUMBER(5),
 CVE_ECOM           NUMBER(5),
 CVE_RENG           NUMBER(3),
 CVE_PROG           CHAR(2),
 ING_EGR            CHAR(1),
 IMP_PRE            NUMBER(15,2),
 CVE_ANIO           NUMBER(2),
 CVE_MESS           NUMBER(2))
tablespace PARTICION9
storage (initial 686k next 74k maxextents 25 pctincrease 0)
/
```

```
create table pre_adpt
(
CVE_CTRO            NUMBER(3),
CVE_DPTO            NUMBER(5),
CVE_RENG            NUMBER(3),
CVE_PROG            CHAR(2),
ING_EGR             CHAR(1),
IMP_PRE             NUMBER(15,2),
CVE_ANIO            NUMBER(2))
tablespace PARTICION9
storage (initial 456k next 98k maxextents 25 pctincrease 0)
/
create table puesto
(
CLA_PUEST           NUMBER(1),
NOM_PERSO           CHAR(30),
DES_PUEST           CHAR(30))
tablespace PARTICION9
storage (initial 10k next 10k maxextents 25 pctincrease 0)
/
create table rsaldo
(
CVE_CTRO            NUMBER(3),
CVE_PROG            CHAR(2),
CVE_DPTO            NUMBER(5),
CVE_RENG            NUMBER(3),
IMP_16              NUMBER(18),
IMP_26              NUMBER(18),
IMP_SAL             NUMBER(20),
IMP_PRE             NUMBER(20),
EJE_ACUM            NUMBER(20),
PRE_ACUM            NUMBER(20),
ING_EGR             CHAR(1))
tablespace PARTICION9
storage (initial 980k next 100k maxextents 25 pctincrease 0)
/

create table saldo
(
CVE_CTRO            NUMBER(3),
CVE_DPTO            NUMBER(5),
CVE_PROG            CHAR(2),
CVE_RENG            NUMBER(3),
CVE_MESS            NUMBER(2),
CVE_ANIO            NUMBER(2),
ING_EGR             CHAR(1),
IMP_SAL             NUMBER(18,2),
SIGNO               CHAR(1),
EJE_ACUM            NUMBER(18),
PRE_ACUM            NUMBER(18))
tablespace PARTICION9
```

```
storage (initial 510k next 50k maxextents 25 pctincrease 0)
/
create table saldoi
(
CVE_CTRO            NUMBER(3),
CVE_DPTO            NUMBER(5),
CVE_PROG            CHAR(2),
CVE_RENG            NUMBER(3),
CVE_MESS            NUMBER(2),
CVE_ANIO            NUMBER(2),
ING_EGR             CHAR(1),
IMP_SAL             NUMBER(18),
SIGNO               CHAR(1),
EJE_ACUM            NUMBER(18),
PRE_ACUM            NUMBER(18))
tablespace PARTICION9
storage (initial 510k next 50k maxextents 25 pctincrease 0)
/
create table saldol
(
CVE_CTRO            NUMBER(3),
CVE_ECOM            NUMBER(5),
CVE_PROG            CHAR(2),
CVE_RENG            NUMBER(3),
CVE_MESS            NUMBER(2),
CVE_ANIO            NUMBER(2),
ING_EGR             CHAR(1),
IMP_SAL             NUMBER(18),
SIGNO               CHAR(1),
EJE_ACUM            NUMBER(18),
PRE_ACUM            NUMBER(18))
tablespace PARTICION9
storage (initial 510k next 50k maxextents 25 pctincrease 0)
/

create table sal_219i
(
CVE_CTRO            NUMBER(3),
CVE_DPTO            NUMBER(5),
CVE_PROG            CHAR(2),
CVE_RENG            NUMBER(3),
CVE_MESS            NUMBER(2),
CVE_ANIO            NUMBER(2),
ING_EGR             CHAR(1),
IMP_SAL             NUMBER(18),
SIGNO               CHAR(1),
EJE_ACUM            NUMBER(18),
PRE_ACUM            NUMBER(18))
tablespace PARTICION9
storage (initial 510k next 50k maxextents 25 pctincrease 0)
/
```

```
create table sal_2191
(
CVE_CTRO            NUMBER(3),
CVE_ECOM            NUMBER(5),
CVE_PROG            CHAR(2),
CVE_RENG            NUMBER(3),
CVE_MESS            NUMBER(2),
CVE_ANIO            NUMBER(2),
ING_EGR             CHAR(1),
IMP_SAL             NUMBER(18),
SIGNO               CHAR(1),
EJE_ACUM            NUMBER(18),
PRE_ACUM            NUMBER(18))
tablespace PARTICION9
storage (initial 510k next 50k maxextents 25 pctincrease 0)
/
create table sal_392i
(
CVE_CTRO            NUMBER(3),
CVE_DPTO            NUMBER(5),
CVE_PROG            CHAR(2),
CVE_RENG            NUMBER(3),
CVE_MESS            NUMBER(2),
CVE_ANIO            NUMBER(2),
ING_EGR             CHAR(1),
IMP_SAL             NUMBER(18),
SIGNO               CHAR(1),
EJE_ACUM            NUMBER(18),
PRE_ACUM            NUMBER(18))
tablespace PARTICION9
storage (initial 510k next 50k maxextents 25 pctincrease 0)
/


create table sal_3921
(
CVE_CTRO            NUMBER(3),
CVE_ECOM            NUMBER(5),
CVE_PROG            CHAR(2),
CVE_RENG            NUMBER(3),
CVE_MESS            NUMBER(2),
CVE_ANIO            NUMBER(2),
ING_EGR             CHAR(1),
IMP_SAL             NUMBER(18),
SIGNO               CHAR(1),
EJE_ACUM            NUMBER(18),
PRE_ACUM            NUMBER(18))
tablespace PARTICION9
storage (initial 510k next 50k maxextents 25 pctincrease 0)
/
```

```
create table solct
(
CVE_CTRO            NUMBER(3),
CVE_DPTO            NUMBER(5),
CVE_ECOM            NUMBER(5),
CVE_RENG            NUMBER(3),
CVE_PROG            CHAR(2),
ING_EGR             CHAR(1),
IMP_PRE             NUMBER(17,2),
CVE_ANIO            NUMBER(2),
CVE_MESS            NUMBER(2))
tablespace PARTICION9
storage (initial 510k next 50k maxextents 25 pctincrease 0)
/
create table temp
(
CVE_RENG            NUMBER(3),
IMP_CONT            NUMBER(15,2),
CVE_CTRO            NUMBER(3),
CVE_DPTO            NUMBER(5))
tablespace PARTICION9
storage (initial 100k next 20k maxextents 25 pctincrease 0)
/
create table tab_temp
( CVE_CTRO          NUMBER(3),
  CVE_PROG          CHAR(1),
  CVE_DPTO          NUMBER(5),
  CVE_RENG          NUMBER(3),
  ING_EGR           CHAR(1),
  IMP_CONT          NUMBER(17,2),
  IMP_PAS           NUMBER(17,2),
  IMP_SALD          NUMBER(17,2))
tablespace PARTICION9
storage (initial 50k next  20k maxextents 121 pctincrease 0)
/
create table grupo
( CVE_MOMC          NUMBER(4),
  CVE_CAFE          NUMBER(3),
  CVE_ECOM          NUMBER(5),
  CVE_DPTO          NUMBER(5),
  FEC_ELAB          DATE,
  CVE_RENG          NUMBER(3),
  IMP_CONT          NUMBER(17,2),
  CVE_CTRO          NUMBER(3),
  ING_EGR           CHAR(1),
  CVE_PROG          CHAR(1))
tablespace PARTICION9
storage (initial 10k next  10k maxextents 121 pctincrease 0)
/
```

```
create table DPTO
( CVE_CTRO          NUMBER(3) NOT NULL,
  CVE_ECOM          NUMBER(5) NOT NULL,
  CVE_CTRG          NUMBER(3) NOT NULL,
  CVE_GCIA          NUMBER(5) NOT NULL,
  CVE_UDCO          NUMBER(3) NOT NULL,
  NUM_CTAM          NUMBER(4) NOT NULL,
  NUM_CTAO          NUMBER(6) NOT NULL,
  DES_DPTO          CHAR(50) NOT NULL,
  CVE_NIVL          NUMBER(2) NOT NULL,
  TIP_REGI          CHAR(1) NOT NULL,
  CVE_OFIP          NUMBER(3) NOT NULL)
tablespace PARTICION9
storage (initial 120k next 10k maxextents 121 pctincrease 0)
/
exit
```

Creación de Indices.

```
tj30071/op_serf
REM ================================================
REM == IXD_SCF.sql Crea indices locales del SERF  ==
REM ================================================
spool idx_serf.lis
drop index ind_dpto1;
drop index ind_dpto2;
create unique index ind_dpto1
   on dpto (cve_ecom)
storage ( initial 5k next 5k maxextents 121 pctincrease 0)
tablespace PARTICION15
/
create unique index ind_dpto2
   on dpto (cve_ctro, cve_ecom)
storage ( initial 5k next 5k maxextents 121 pctincrease 0)
tablespace PARTICION15
/
commit;
exit
```

Creación de Sinonimos

```
tj30071/op_serf
set echo on
spool sy_serf.lis
drop synonym t_041;
drop synonym cctro;
drop synonym depto;
drop synonym renga;
drop synonym prog;
drop synonym dpto;
drop synonym gpo_ecom;
drop synonym rangos;
drop synonym regas;
```

```
drop synonym rp_00;
drop synonym sector;
drop synonym temp2;
drop synonym zona;
drop synonym cp_v1;

create synonym t_041      for tj30026.t_041;
create synonym depto      for tj30061.depto;
create synonym cetro      for tj30061.cetro;
create synonym renga      for tj30061.renga;
create synonym prog       for tj30061.prog;
create synonym dpto       for tj30048.dpto;
create synonym gpo_ecom for tj30048.gpo_ecom;
create synonym rangos     for tj30048.rangos;
create synonym regas      for tj30048.regas;
create synonym rp_00      for tj30048.rp_00;
create synonym sector     for tj30048.sector;
create synonym temp2      for tj30048.temp2;
create synonym zona       for tj30048.zona;
create synonym cp_v1      for tj30048.cp_v1;
commit;
exit
```

Permisos sobre Sinonimos

```
tj30048/adm_scf
set echo on
spool gr_serf.lis
revoke all on dpto        from tj30071;
revoke all on gpo_ ecom   from tj30071;
revoke all on rangos      from tj30071;
revoke all on regas       from tj30071;
revoke all on rp_00       from tj30071;
revoke all on sector      from tj30071;
revoke all on temp2       from tj30071;
revoke all on zona        from tj30071;
revoke all on cp_v1       from tj30071;
grant select on dpto      to tj30071;
grant select on gpo_ecom  to tj30071;
grant select on rangos    to tj30071;
grant select on regas     to tj30071;
grant select on rp_00     to tj30071;
grant select on sector    to tj30071;
grant select on temp2     to tj30071;
grant select on zona      to tj30071;
grant select on cp_v1     to tj30071;
conn tj30061/pemex
revoke all on cetro   from tj30071;
revoke all on depto   from tj30071;
revoke all on renga   from tj30071;
revoke all on prog    from tj30071;
```

```
grant select on cetro   to tj30071;
grant select on depto   to tj30071;
grant select on renga   to tj30071;
grant select on prog    to tj30071;
conn tj30026/adm_vta
revoke all on t_041     from tj30071;
grant select on t_041   to tj30071;
commit;
exit
```

Estructura de la Base de Datos del SERF.

Nombre del objeto	Tipo de Objeto
CETRO	SYNONYM
CP_V1	SYNONYM
CP_V2	SYNONYM
DEPTO	SYNONYM
DPTO	SYNONYM
FE_00	SYNONYM
FL_EFC	SYNONYM
GPO_ECOM	SYNONYM
GRUPO	SYNONYM
PASIVO	TABLE
PRESP	TABLE
PRE_ADPT	TABLE
PROG	SYNONYM
PUESTO	TABLE
RANGOS	SYNONYM
REGAS	SYNONYM
RENGA	SYNONYM
RP_00	SYNONYM
RSALDO	TABLE
SALDO	TABLE
SALDOI	TABLE
SALDOL	TABLE
SAL_219I	TABLE
SAL_219L	TABLE
SAL_392I	TABLE
SAL_392L	TABLE
SECTOR	SYNONYM
SOLCT	TABLE
TAB_TEMP	TABLE
TEMP	TABLE
TEMP2	SYNONYM
T_041	SYNONYM
T_082	SYNONYM
ZONA	SYNONYM
IND_DPTO1	INDEX
IND_DPTO2	INDEX

3.5 Introducción al sistema operativo HP-UX (UNIX).

HP-UX es un Sistema Operativo basado en estándares, excepcionalmente poderoso. Es una implementación del Sistema Operativo **UNIX** (marca registrada por AT&T en los EE.UU. y otros países) con mejoras en el incremento en tiempo real.

HP-UX esta provisto de un ambiente interactivo de trabajo, el cual incluye.

Un poderoso intérprete de comandos en línea (**shell**)
Un rico lenguaje de comandos (la programación en lenguaje **shell**)
Un conveniente sistema de archivo (el sistema de directorios).
Un poderoso lenguaje de programación "C".

HP-UX es completamente compatible con la definición de interfase del sistema V de AT&T (SVID). **HP-UX** también incluye muchas características de la Universidad de California en Berkeley, versión 4.2 BSD (Distribución de Sotfware en Berkeley). Esta adición de características de soporte aumenta la compatibilidad de **HP-UX**. Además **HP-UX** incluye muchas innovaciones que aumentan la capacidad del Sistema Operativo.

Características de **HP-UX** incluídas:

- Enlace en tiempo real
- Alta funcionalidad en el acceso a archivos
- Librerías de Dispositivos de Entrada/Salida (E/S)
- Soporte de Lenguaje Nativo (NLS)

Características estándares de **HP-UX** incluídas

- Múltiples tareas
- Compatible de modo multiusuario o monousuario
- Ambiente de soporte flexible.
- Comunicación entre usuarios, incluyendo correo electrónico
- Librerías de herramientas de edición, compilación y eliminación de errores.
- Redireccionamiento de los usuarios de E/S.
- Capacidad para jerarquizar los archivos del sistema
- Capacidad para ejecutar comandos seguidos de otro.

En adición a las características estándares de el Sistema Operativo **HP-UX** enlace en tiempo real incluye.

- Prioridad en tiempo real.
- Programar en base a tiempo real.
- Sincronización de dispositivos de E/S.
- Control de usuarios que accesan.
- Señales de sotfware (interrupción y bloqueo).
- Control de acceso para privilegios en tiempo real.
- Control de usuarios para accesar a archivos del sistema.
- Comunicación entre interprocesos.
- Candados de proceso.
- Candados de archivos.

Una gran variedad de herramientas están disponibles dentro del Sistema Operativo, las cuales son soportadas por el Sistema Operativo **HP-UX**:

- Programación y migración de herramientas
- Manejador de Base de Datos.
- Ambiente de Aplicación de Utilerías.
- Lenguaje Nativo.
- Aplicaciones para redes de trabajo.

El siguiente párrafo describe algunas de estas aplicaciones.

3.5.1 Herramientas de programación.

Las herramientas de programación incluyen servicio de compiladores y significativos mensajes de error (symbolic debugger). Lenguajes de programación actualmente ofrecidos

Ensamblador
HP C
HP Fortan 77
HP Pascal

El **symbolic debugger** de HP, es un poderoso y flexible, programa interactivo. Diseñado para mejorar la productividad del desarrollo del "software".

PORT/HP-UX es un grupo de herramientas de emigración que HP9000 permite usar para modificar programas así estos pueden correr en el Sistema **HP-UX.**

3.5.2 Librerías gráficas.

Starbase y DGL/AGP son las actuales gráficas que existen en **HP-UX.** La librería gráfica "starbase" es de un nivel bajo de dos dimensiones y librerías gráficas de tres dimensiones para **HP-UX.** El **Device-indepent Graphic Library (DGL)** y el **Advanced Graphics Package (AGP)** son dos librerías gráficas de soporte.

3.5.3 Manejador de Base de Datos.

ALLBASE es un sistema manejador de Base de Datos que nos permite elegir el apropiado modelo de datos en una aplicación (aplicación base). **ALLBASE** ofrece un compresivo grupo de características para sus dos modelos **HPSQL, interface** del modelo relacional, o **HPIMAGE interface** para el modelo de red. **ALLBASE** es construida en base a una funcionalidad solida de comandos internos que son diseñados especificamente para la realización de la explotación de una manera precisa con respecto a la arquitectura de HP.

3.5.4 Ambiente de aplicación de utilerías.

Hptoday es un lenguaje de cuarta generación que consiste en paquetes de programas asistidos por computadora para ambiente de datos o relacionar transacciones de procesos con aplicaciones.

Native Lenguaje Support (NLS). Es un conjunto de herramientas disponibles para producir aplicaciones locales. Las herramientas **NLS** permiten escribir programas con algún lenguaje independiente por medio de una interfase. Esta interfase permite hacer cambios de un lenguaje local sin realizar modificaciones, hacia un programa ejecutable. Actualmente **HP-UX NLS** incluye soporte de caracteres, mensajes y comandos para 25 diferentes lenguajes.

Aplicación para redes de trabajo

Network Services (NS), Local Area Network (LAN), Advanced Research Projects Agency (ARPA), y Systems Network Arquitecture (SNA) Network File System (NFS). Estas herramientas nos facilitan la transferencia de archivos entre sistemas, sin preocuparse por todos los detalles técnicos del chequeo de errores y mensaje de **run-time.**

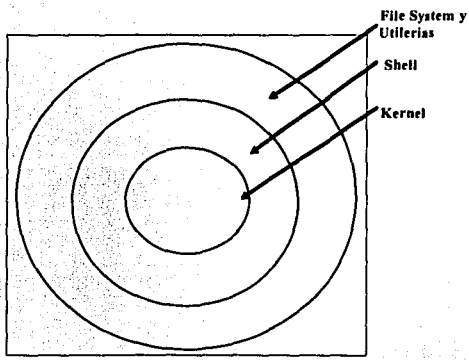
3.5.5 Estandar de HP-UX.

El Sistema Operativo (SO) **HP-UX** esta compuesto.

- Kernel
- Shell
- Utilerías
- File System (particiones lógicas de los discos).

La figura 3.11 muestra la estructura básica del sistema **HP-UX.**

Figura 3.11 Estructura básica del sistema HP-UX.



El **kernel** controla los recursos de la computadora. Esto permite realizá tareas en la computadora sin poner atención a los detalles de "hardware". Por ejemplo para obtener copia de un archivo en la impresora. No necesitamos preocuparnos acerca de cuándo el dato es enviado o del control de cómo

opera la impresora. No se necesita esperar a que termine la impresión para que continuar el trabajo, simplemente se teclea el comando con el nombre del archivo y el sistema se encarga de realizar esta tarea,lo que es más, se puede teclear otro comando inmediatamente para realizar la siguiente tarea y posteriormente recoger la impresión.

Las tareas principales del **kernel** son:

1) Administración de los recursos del sistema incluyendo dispositivos físicos semejantes a terminales, impresoras y discos para dejar **haring of resources**.
2) Administración de la memoria del **CPU**, para mantener la mayor eficiencia del uso de la memoria.
3) Ejecutar programas a través de comandos, tecleándolos en la terminal o dentro de un programa.
4) Control de operación multi-terminal para permitir a los usuarios que en diferentes terminales realicen sus tareas al mismo tiempo.
5) Control de dispositivos de Entrada/Salida (E/S) para facilitar la comunicación con diferentes dispositivos. (E/S).
6) Respuesta para interrupciones externas del **scheduler** y correr programas de aplicación especial para administración o monitoreo del **SO**.

3.6 El shell

El **shell** es un programa interactivo que interpreta los comandos que tecleamos en la terminal y los manda al **kernel** para realizar el requerimiento de la tarea.

El ambiente **shell** es donde se realizan muchos de los trabajos por medio de la terminal. Desde el **shell** se puede invocar otros programas semejantes como editores de texto.

Si ocurre un error durante la ejecución del comando, el **shell** despliega un diagnóstico de mensaje de error seguido por el **prompt** del **shell**. En este punto se puede reintegrar el comando correcto para la realización de la tarea requerida.

Tres programas de interfase son provistos por **HP-UX**, en el momento que tu cuenta es ejecutada, el administrador del sistema elige uno de estos tres programas para el usuario. Estos son el **Bourne Shell (sh)**, y **CShell (csh)**, o el **Korn shell (ksh)**. El ambiente del **Cshell** esta mucho más ligado con el **Bourne shell**, pero ofrece un mayor poder de ambiente con más características de uso interactivo. El **korn shell** incorpora las características de el **C shell** y la portabilidad de **Bourne shell**.

Características estándares del **shell**

Lenguaje compatible con archivos de programas de **batch**
Seleccionar la forma de ejecución **foreground** o **background**.
Redireccionamiento E/S.
Entubamiento entre varios comandos.

3.6.1 Utilerías

Las utilerías forman una gran parte de el estándar del **SO HP-UX**. Estos son programas que realizan una variedad de funciones específicas. Muchos de estos programas son herramientas diseñadas para auxiliar en el desarrollo de aplicaciones que se deseen.

Econtraremos muchas herramientas en el **SO HP-UX**. Estas son utilerías para editar textos y formatearlos,desarrollo de programas, administración y mantenimiento del sistema. Utilerías estándares incluídas.

grep - Un programa para buscar dentro de archivos de texto.
sort - Un programa para sortear archivos de texto.
awk - Un lenguaje de programación para manipulación de datos y textos.
make - Un programa de mantenimiento para programadores.
link - Programa que checa y verifica el código fuente de "C".
ed - Un editor de textos interactivo orientado a trabajar por línea.
vi - Un editor de textos interactivo orientado a trabajar por pantalla.
sed - Un editor de texto no interactivo.
bc - Un programa aritmético.
dc - Una cálculadora de escritorio.
wc - Contador de palabras, líneas y caracteres.
mailx - Programa de correo electrónico.

3.7 Los file system

Los file system de **HP-UX** tienen una estructura organizada en forma jerárquica para almacenar información. A esta colección de información se le llama archivo y puede contener programas, cartas, memorandums, datos estadísticos, o descripciones del **shell**. Un directorio contiene archivos e información acerca de otros directorios.

Estructura de los file system, directorios y archivos.

Los file system en el **SO HP-UX,** contienen una estructura diseñada para almacenar datos. Ellos están hechos de archivos y directorios. Esta parte describe los atributos de almacenamiento (usualmente un disco) y son accesados por el nombre del archivo.

Estos son tres tipos de archivos de los file system

1) Archivos ordinarios
2) Directorios.
3) Device file (archivos especiales).

Un directorio es un archivo que contiene información de otros archivos. El sistema usa device file así de este modo puede accesar dispositivos periféricos (semejantes al de una cinta magnética).

Solo archivos de texto, directorios y archivos ejecutables son descritos. Los archivos especiales y **device file** no se verán.

Una serie de archivos estándares comunmente aparecen en cada cuenta de los usuarios. Una cuenta se establece por medio del administrador del sistema, así se puede tener acceso al sistema. El dueño de la cuenta puede o no alterar estos archivos que fueron determinados por el administrador del sistema cuando la cuenta fue creada, archivos que pueden o no aparecer en tu cuenta.

El archivo **.login** o **.cshrc**, en el momento de establecer una sesión de trabajo es ejecutado sus **script** de estos, cuando utiliza el **bin/cshell**, esto es cuando inicia tu logotipo de entrada. Este archivo levanta las variables de ambiente en el momento de ser ejecutado. Es necesario ejecutarlo en el inicio de cada sesión de **login**.

El archivo **.cshrc**, muchas veces es usado para adecuar el ambiente **cshell** en el **login** o cuando es ejecutado **csh**.

El archivo **.porfile** en el **bourne shell** adecúa el ambiente del **bourne shell** como el archivo **.cshrc** lo hace en el **cshell**.

El archivo **.mailrc** levanta las variables de recepción de correo. Este adecúa el ambiente de lectura de correo, controla los comandos **mailx** y puede proporcionar nombres cortos para el uso de direcciones comunes.

El archivo **.history** es utilizado por el **kshell** y contiene una historia de la mayor parte de los comandos recientemente ejecutados. Este archivo es usado para guardar la historia de los comandos y poderlos utilizar nuevamente sin teclearlos de nuevo, apoyados en el editor **vi.**

Todos estos son comandos ocultos dentro de cada directorio, los cuales contienen el **pathname** (el nombre de la trayectoria padre), del directorio y el directorio mismo. Estos directorios especiales son descritos en la siguiente sección.

El archivo que contiene el **pathname** para el directorio padre es "..". El archivo que contiene el **pathname** para el actual directorio es ".".

El nombre de un archivo en **HP-UX** puede tener 255 caracteres. Se puede usar una combinación de letras de la "a" a la "z" y de la "A" hasta la "Z", números del 0 hasta el 9 y caracteres como son el **underscore** "_", coma "," y el signo de menos "-". Los nombres de los archivos pueden permitir caracteres internacionales válidos de 8 y 16 bits.

Para **HP-UX** si existe diferencia entre mayúsculas y minúsculas. No debemos utilizar caracteres que tengan un significado especial. Estos caracteres son llamados matacaracteres.

La siguiente es una lista de caracteres que se deben evitar cuando se escribe el nombre de un archivo.

Slash	(/)
Mayor que	(>)
Menor que	(<)
Barra Vertical (pipe)	(|)
Signo de Interrogación	(?)
Corchete Izquierdo	([ )
Corchete Derecho	( ])
Asterisco	(*)
Llave Derecha	({)
Lave Izquierda	(})
Espacio	( )
Tilde	(~)
Apóstrofe	(')
Comillas	(")
Diagonal Invertida	(\ )
Grave	(`)
Signo de Admiración	(!)
Ampersand	(&)

Signo de Pesos	($)
Paréntesis Izquierdo	(( )
Paréntesis Derecho	( ))
Punto y Coma	(;)

El punto (.) en un archivo es incondicional se usa de dos formas. Primero, cuando el nombre de un archivo inicia con punto es normalmente para ocultar el archivo en el momento en que tu invoques la lista de archivos de un directorio con el comando (ls -l) estos no se verán. La segunda forma de utilizar el punto es generalmente para preceder la extensión de un archivo. Algunos comandos (o programas) esperan ciertas convenciones como las siguientes: Por ejemplo los archivos fuentes en "C" usualmente terminan con extensión **.c**, los archivos fuentes de pascal terminan con **.p** y las librerías terminan con **.a**. Si se necesita se puede asignar alguna extensión para archivos que sean similares. Por ejemplo, si se tienen archivos que contiene textos se puede dar la extensión **.txt**, o se puede dar a archivos temporales la extensión **.tmp**. Estos son algunos ejemplos. Se puede dar al nombre del archivo la extensión que se desee.

Ejemplos de nombres de archivos con y sin extensión

mensaje	tmp2	paso.p	reporte	100884
memo.qut1	carta	fruta	Arturo3	Tmp
temporal	a.out	prueba.007	jose	magi,c
prueba.102	tmp1	tunt.-f	.profile	TEMPORAL
aves@palab	una_ayuda	lost+found	_exre	arch.nombre.Z

3.7.1 Directorios

Un directorio es un archivo que contiene descripción sobre algunos otros archivos. Estos archivos deben estar contenidos en el directorio. Se puede tener uno ó más directorios que contienen algunos archivos u otros directorios. Un directorio contiene el nombre e **inodo** que es el identificador del archivo contenido dentro de él. El **inodo** de un archivo contiene información como el tipo de archivo, el tamaño del archivo y la localización de esté. Un directorio contenido dentro de algún otro directorio es llamado subdirectorio. Esta capacidad de anidar directorios nos muestra que la arquitectura en la que se ordenan los archivos es de tipo jerárquica Si pensamos que los archivos son carpetas de información, entonces los directorios son archiveros en donde se almacenan carpetas que se relacionan entre si.

Se pueden crear y utilizar directorios para almacenar información. Por ejemplo, tener todos los memorándums en un directorio, toda la información acerca de clientes en otro, y toda la información acerca de manuales en algún otro directorio

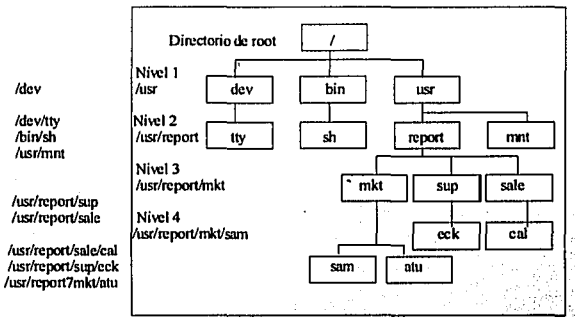
El nombre de un archivo puede estar sobre los 255 caracteres, y consiste en la combinación de letras mayúsculas, dígitos y otros caracteres del grupo de caracteres de HP (Hewlett Packard). De cualquier manera la misma restricción es aplicada para nombres de directorios como para nombre de archivos, haciendo referencia en este capítulo a la lista de las restricciones a caracteres específicos.

El **slash** (/) es un medio para los **files system** y no es permitido para los nombres de directorios. Los nombre de los directorios pueden permitir caracteres internacionales validos de 8 y 16 bits.

3.7.2 Nombre de las trayectorias (Pathnames)

El **pathname** es la localización de los archivos o directorios dentro de los **files system.** Este **pathname** es la trayectoria que nos muestra la estructura jerárquica de los directorios. El **pathname** esta compuesto de una serie de nombres de directorios separados por el slash (/) y termina cuando el nombre del directorio o archivo es localizado. Por ejemplo, la figura 3.12 muestra algunos **pathnames**

Figura 3.12 Trayectorias de localización (Pathnames)



El **pathname** de un directorio puede ser especificado por dos rutas, absoluta o relativa. La ruta utilizada en el **pathname** depende de donde se esta trabajando en el momento, y hacia donde queremos ir, para poder accesar al archivo o directorio deseado.

El **pathname** absoluto describe la localización de un archivo o directorio en relación de el **file system** de **root.** Un **pathname** absoluto inicia con un **slash** (/), esto significa que es el directorio de **root.** Un **pathname** absoluto se inicia con toda la trayectoria del file system. Por ejemplo para llamar al directorio "cal" se especifica el **pathname** absoluto.

/usr/report/sale/cal

El **pathname** relativo define la localización de un directorio en relación al directorio de trabajo. El **pathname** relativo inicia con el nombre del directorio o archivo dentro del actual directorio. Por ejemplo

sal/cal report/sup/eck mkt/sam mkt/atu

Estos son varios directorios especiales en la estructura jerárquica de los archivos. Los cuatro directorios especiales son, el directorio de root, el directorio casa, el directorio de trabajo y el directorio padre.

El directorio de **root** es el más alto en la estructura de los archivos y es designado por el **slash** (/) es el primer caracter de un **pathname.** Todos los **pathnames** absolutos inician con el directorio de **root.**

El directorio casa (**HOME**) es el directorio que es asignado en tu **login**. Este directorio es asignado por el administrador del sistema cuando crea algún cargo. Y es el directorio de trabajo hasta que no se cambie de directorio.

El directorio de trabajo es nuestro actual directorio. El directorio de trabajo y nuestro directorio casa son el mismo cuando accesamos al sistema y no se cambia de este.

El directorio padre. Cada directorio tiene un directorio padre, incluyendo el directorio de **root**. El directorio padre de **root**, es el mismo directorio de **root**. Por ejemplo el directorio / (**root**) es el directorio padre de dev, bin, y usr como se muestra en la figura 3.13

Estándares de directorios

/	El directorio de root requerido para todos los **files system**.
/usr	Contiene usuarios y directorios de soporte de sistemas.
/bin	Contiene subdirectorios para el uso del sistema
/usr/bin	Contiene utilerías y programas.
/dev	Contine archivos de dispositivos.
/etc	Contiene miscelánceas del administrador del sistema, archivos como el **passwd**, el **inittab** y utilerías de instalación.
/tmp	Contiene archivos temporales.
/lib	Contiene librerías (subrutinas).
/usr/lib	Contine más librerías (manuales).
/mnt	Contiene el directorio casa de los usuarios, aunque no siempre.

Estructura de los file system de la HP9000/842.

PATHNAMES	FILE SYSTEM	BLOCKS LIBRES	NUMERO DE INODOS
/usr	(/dev/dsk/c1000d0s4):	15908 blocks	26227 i-nodes
/tmp	(/dev/dsk/c1001d0s0):	41088 blocks	8160 i-nodes
/spr	(/dev/dsk/c1001d0s1):	52512 blocks	13962 i-nodes
/dbroll1	(/dev/dsk/c1001d0s4):	64192 blocks	954 i-nodes
/rdbms	(/dev/dsk/c1001d0s5):	43920 blocks	622 i-nodes
/dbroll2	(/dev/dsk/c2000d0s7):	496 blocks	698 i-nodes
/transfer	(/dev/dsk/c2000d0s10):	53824 blocks	367 i-nodes
/admon	(/dev/dsk/c2000d0s4):	8928 blocks	946 i-nodes
/prd/serf	(/dev/dsk/c1000d0s14):	14656 blocks	920 i-nodes
/prd/flujo	(/dev/dsk/c1000d0s3):	52224 blocks	1276 i-nodes
/prd	(/dev/dsk/c2000d0s5):	119952 blocks	4111 i-nodes
/dbroll3	(/dev/dsk/c2001d0s7):	496 blocks	698 i-nodes
/db3	(/dev/dsk/c2001d0s10):	15248 blocks	1145 i-nodes
/db5	(/dev/dsk/c2001d0s3):	256 blocks	314 i-nodes
/dbs	(/dev/dsk/c2001d0s4):	10064 blocks	954 i-nodes
/base1	(/dev/dsk/c2001d0s5):	235680 blocks	6967 i-nodes
/bck	(/dev/dsk/c1000d0s5):	526320 blocks	19233 i-nodes
/dsr/inst	(/dev/dsk/c1001d0s3):	7136 blocks	1125 i-nodes
/dsr/dsrloc	(/dev/dsk/c2000d0s3):	23488 blocks	892 i-nodes
/dsr	(/dev/dsk/c1001d0s10):	108432 blocks	2383 i-nodes
/	(/dev/dsk/c1000d0s15):	5248 blocks	10720 i-nodes

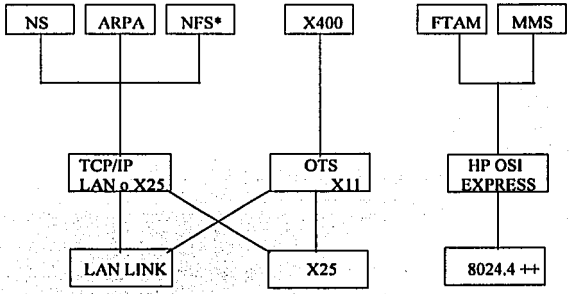
3.8 Las redes de trabajo de HP-UX (Networking).

Las funciones del software de trabajo son frecuentemente divididas en tres catagorías: servicio, transporte y enlace. Cada categoría generalmente corresponde a un nivel de OSI (Organización Internacional de Standares).

1) Este servicio corresponde al nivel 7.
2) Transporte a los niveles 3 a 6.
3) Enlace a los niveles 1 y 2.

HP ofrece productos en cada una de estas categorías, la fila superior de la figura 3.13 muestra los productos ofrecidos por **HP-UX**, la fila central lista los productos de transporte y la última los productos de enlace.

Figura 3.13 Servicios que ofrece HP-UX



* El transporte **TCP/IP** esta enlazado con los productos **LAN y X25**
^ **NFS** no esta apoyado sobre **X25**
++ Productos enlazados,que estan incluidos **HP OSI Express** y **MAP 3.0**

Estos son los seis productos de servicio de **HP-UX**

* Servicio ARPA
* Servicio NFS
* Servicio NS
* X.400
* MMS
* FTAM

El transporte define como trabaja una red de datos y asegura su integridad, generalmente corresponde a las etapas 3 a 6 del modelo **OSI**. Estos son tres productos de transporte que ofrecen **HP-UX**.

* Enlace LAN/900 y el transporte para TCPI/IP network.
* OTS/900 (con XTI)
* HP OSI Express

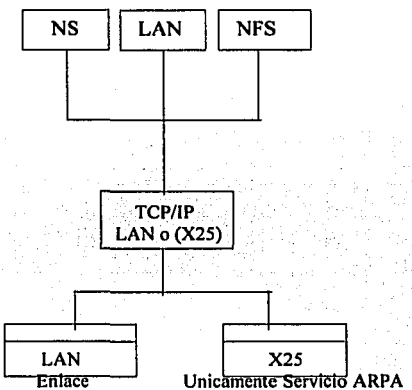
3.8.1 Enlace

Links define como se transmite los datos físicamente desde un punto a otro. Ello incluye el hardware y sotfware para las capas inferiores de **OSI**. El **hardware** es el medio físico de transmisión, y el sotfware define la trayectoria de la red, selecciona los datos y los carga para transmitirlos. generalmente corresponde a los niveles 1 y 2 de **OSI**. Estos son tres productos de enlace que ofrece **HP-UX**.

- LAN/900 link.
- X25/900 link.
- HP OSI Express y MAP 3.0

La figura 3.15 muestra de manera más ilustrativa la estrecha relación que hay entre estos productos.

Figura 3.14 Interrelación entre productos de enlace.



3.9 Plataforma de desarrollo (Hardware)

3.9.1 Modelo de discos 670 FL y 1.3 FL

Sistema de almacenamiento sobre discos en los modelos **670 FL** y **1.34 FL**.

Los sistemas de almacenamiento en disco de la serie HP9000 son sistemas de disco de alta funcionalidad, diseñados para sistemas computacionales medios hasta completos. Cada disco combina un mecanismo de discos de 5.25 pulgadas, un controlador **HP-FL** y una fuente de almacenamiento en un gabinete, el modelo de **1.34 FL** contiene **1.34 gigabytes** y el **670 FL 670 kilobytes** de espacio en disco.

Todos los modelos son equipos con un controlador inteligente **HP-FL.** El controlador **HP-FL** es una interfase de fibra óptica de alta funcionalidad. Para mayor confiabilidad, cuenta con un diagnóstico de autoprueba el cual es diseñado dentro del controlador. El sistema de almacenamiento en discos pude ser operado como una unidad o montarse en un **Rack** de cuatro dimensiones.

Más de 8 discos pueden ser instalados en un mismo gabinete (el **IBG-DELETE**, el circuito de interfase de fibra óptica de **HP-FL** y el controlador **PCA**).

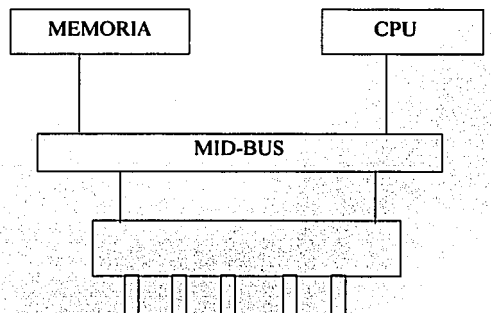
3.9.2 Dos Arquitecturas de hardware HP-PB y CIO

Existen dos tipos de arquitectura de hardware **HP-PB** y **CIO** para equipo HP9000 series 808, 815, 832, 842 y 852.

Como se puede observar en la figura 3.15, la Unidad de Procesamiento Central **(CPU)**, la memoria y el canal adaptador esta sobre el **MID-BUS** (el bus es la ruta de comunicación).

La figura siguiente (figura 3.15) muestra la arquitectura básica de una configuración **CIO** para computadoras HP9000 serie 600/800.

Figura 3.15 Arquitectura CIO



En el interior de la Unidad de Proceso de Servicio **(SPU)** hay ranuras sobre el **MID-BUS** en las cuales se pueden insertar tarjetas de circuitos. Estas tarjetas **(MID-BUS-CARD)** pueden comunicarse a través de el **MID-BUS**, dos ejemplos de **MID_BUS_CARD** son la tarjeta de memoria y el canal adaptador.

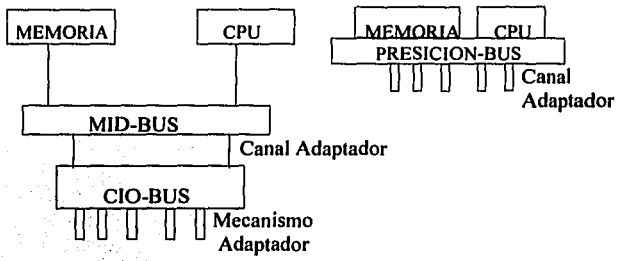
El Canal de Entrada/Salida **(CIO-BUS)** en computadoras de serie 600/800 son de propósito general para todas, estás utilizan una arquitectura **CIO**. Dentro del **SPU**, hay ranuras en las cuales, se pueden insertar tarjetas de **CIO** estas ranuras del **SPU** son especiales para tarjetas **CIO**. Estas tarjetas se comunican a través de el canal adaptador al **MID-BUS**. Algunos ejemplos de tarjetas **CIO** son:

- Tarjeta **HP-IB** para mecanismos tales como, **HP-IB** disk drive, tape drive, **CD-ROM** drive y **HP-IB** impresoras.

- Tarjeta **HP-FL** para mecanismos los cuales usan fibra óptica como enlace, tales como **HP-FL** disk drive.
- Tarjeta de **SCSI** para mecanismos que usan **SCSI** bus, tales como disk drive, tape drive, CD-ROM drive, magnetos ópticos drive.
- Tarjeta **MUX** para periféricos o dispositivos seriales tales como terminales, modems, e impresoras en serie.
- Tarjeta **LAN** (o **LANIC**) para redes LAN (Redes de Area Local).

Los equipos HP9000 serie 800 de las familias 808, 815, 822, 832, 842, 852 utilizan la arquitectura de **hardware** conocida como **HP-PB** (Hewlett Packard Precisión Bus)

La figura 3.17 Muestra la comparación entre la Arquitectura CIO y HP-PB



Como se puede observar en la Arquitectura **HP-PB**, el **CPU**, la memoria y las tarjetas de E/S, se comunican a través del **PRESICION-BUS** (esta es la ruta de comunicaciones entre los módulos de hardware). Dentro del **SPU** existen ranuras en el **PRECISION-BUS**, en las cuales se insertan tarjetas de E/S.

Las tarjetas de E/S se comunican a través del **PRECISION-BUS** con el **CPU** y la memoria, algunas tarjetas de E/S de la arquitectura **HP-PB** son:

- Tarjeta **HP-IB** para mecanismos **HP-IB**.
- Tarjetas **PBA-FL** para mecanismos **HP-FL**.
- Tarjetas **SCSI** para mecanismos los cuales usan **BUS SCSI**.
- Tarjetas **MUX** para mecanismos en serie
- Tarjetas **LAN** (o **LANIC**) para redes locales.
- Tarjetas de acceso remoto (**AP**) para soporte remoto.

Dispositivos con los que cuenta el equipo HP9000/842

1	HP9000 Modelo 842	A1154A
1	28650A HP-IB Interface	28650A
1	HP-PB 8 Channel Asinchronous Mux	40299A
1	Models 8425	A1154X
1	32 MB ECC RAM Memory	A1437A
1	HP 700/92 Terminal Green Disp	C1001G
1	Embedded Digital Data Storage	C1501A
1	Integrated 670 Mb Mechanism/Con	C2282A
1	LAN link	A1154A
1	LAN 9000 Series 800 for Model 8	98189A
1	Increase memory 96 MB	A1154A
2	32 Mb ECC RAM Memory	A1437A
1	Increase disk 1.3 Gb	A1154A
1	Integrated 670 Mb Mechanism/Con	C2282A
1	Add 5 Mux Card	A1154A
5	HA-PB 8 Channel Asinchronous Mux	40299A
1	Rack mount kit for 20 ADP's	D2350A
1	Model 1.34FL Formatted fixed Di	C2204A
1	Model 1.34FL Formatted fixed Di	C2204A
1	Mini Rack Bundle for HP6000 Dis	92211y
1	PBA FL Interface	A1749A
1	PBA FL Interface	7980A
1	NetAssure Support Service	36960A+16B
1	X.25/9000 Link for the Series	36960A
1	X.25/800 Link for the Model 840	36942A
1	NetAssure Service	36942A+16B
1	1200 LPM Line Impact Printer	2556C
1	RS-232-C Interface	2566C-049

DESARROLLO DE MENUS

4.1 Introducción

El Sistema de Estado de Resultados Financieros (SERF) fue desarrollado para la implementación de un programa de control local y seguimiento de las adquisiciones que se originan en le centro de trabajo.

El sistema consta de 9 menús y dos librerías, todos estos desarrollados en lenguaje "C", como se observara durante este capítulo. Aquí podremos ver los programas fuentes utilizados para el desarrollo de menús, y una descripción de lo que realizan, así como las pantallas que éstos generan. Los menús se encuentran en la trayectoria siguiente:

/dsr/dsrloc/serf/menús

El nombre de todos los programas fuentes que estan dentro de esta trayectoria es el siguiente:

menú_sasl.c	Menú de Actualización de Saldos
menú_grpt.c	Menú de Generación de Reporte
menú_impr.c	Menú de Impresión de Reporte
menú_mtor.c	Menú de Monitoreo de Procesos
menú_pcap.c	Menú de Pantallas de Captura
menú_pcon.c	Menú de Pantallas de Consultas
menú-princ.c	Menú de Principal
menú_rsal.c	Menú de Respaldo de Saldos
menú_solc.c	Menú de Solicitud de Fondos
menu2	Librería que integra rutinas para generar el recuadro, la fecha, la estructura y ubicación del cursor.
error.c	Librería para establecer los mensajes de error y el modo **raw**.

Aquí también incluiremos algunos programas, que aunque no son propiamente menús, forman parte del sistema y como los menús también fueron desarrollados en lenguaje " C".

Estos programas se encuentran en diferentes trayectorias, como se describe a continuación.

/dsr/dsrloc/serf

marco.c	Programa que genera el marco para lanzar algunos procesos.
cuadro.c	Librería utilizada en los proceso para limpiar y mandar los mensajes que generan algunos procesos en pantalla.

4.2 Programas y pantallas de menús.

Los programas y la pantalla que despliega cada uno se muestran en seguida, estos tiene la finalidad de que el usuario explote el sistema en forma interactiva apoyado en los siguientes menús.

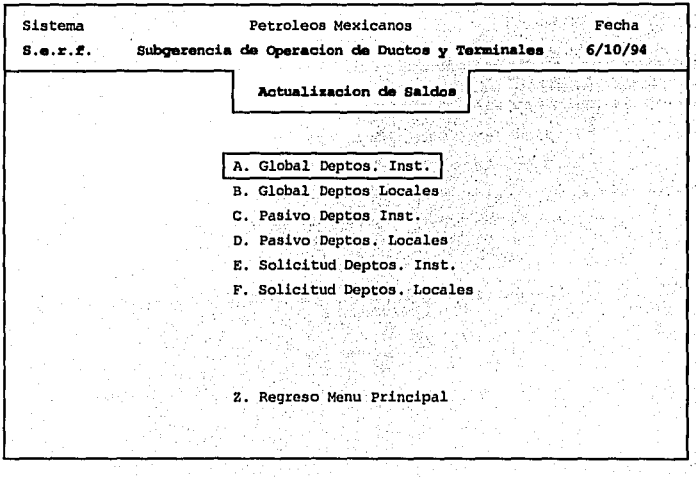
```
/*************************************************************
 *          Sistema de Estados de Resultados Financieros     *
 *                                                           *
 *          Libreria que Contiene las Rutinas de Mensajes    *
 *         de Errores y  Manejo del Modo Raw                 *
 *                                                           *
 * Autor:  Arturo Leon Juarez              Programa: error   *
 * Fecha: 01/01/94                        Equipo  : hp9000/842 *
 *************************************************************/

/*****************************
 *      rutinas   syserr()     *
 *****************************/

void syserr(msg)  /* impresion del mensaje de error del sistema y aborta */
char *msg;
{
  extern int errno, sys_nerr;
  extern char *sys_errlist[];

  gotoxy(21,7);
  fprintf(stderr, "Error : %s ( %d", msg, errno);

  if (errno > 0 && errno < sys_nerr)
     {
     gotoxy(21,20);
     fprintf(stderr,"; %s ) \n", sys_errlist[errno]);
     }
  else
     {
     gotoxy(21,20);
     fprintf(stderr,"; ) \n");
     }
  exit(1);
}

/*****************************************
 *      rutinas   setraw()       *
 *****************************************/
```

```
static struct termio tbufsave;

void setraw()
{
  struct termio tbuf;

  if (ioctl(1, TCGETA, &tbuf) == -1)      /* se guarda estado actual */
     syserr("ioctl ");
  if (ioctl(1, TCGETA, &tbufsave) == -1)  /* se guarda estado actual */
     syserr("ioctl ");

      /* se respalda estado actual y se cambia a modo raw */

   tbuf.c_iflag &= ~(INLCR | ICRNL);
   tbuf.c_lflag &= ~(ICANON | ECHO);
   tbuf.c_cc[4] = 1;                  /* Minutos */
   tbuf.c_cc[5] = 0;                  /* Tiempo */

  if (ioctl(0, TCSETAF, &tbuf) == -1)  /* Se establecen modos raw */
     syserr("ioctl2 ");
}

/*****************************************
 *      rutinas   restore()        *
 *****************************************/

void restore()                       /* Regresa la bandera de fin */
{
  if (ioctl(0, TCSETAF, &tbufsave) == -1)
     syserr("ioctl3 ");
}
```

```
/*****************************************************************
 *          Sistema de Estados de Resultados Financieros         *
 *                                                               *
 *              Programa que Contiene el Menú de                 *
 *              Actualizacion de Saldos                          *
 *                                                               *
 * Autor: Arturo Leon Juarez                Programa: menú_asal  *
 * Fecha: 01/01/94                          Equipo  : hp9000/842 *
 *****************************************************************/

#include "menu2.c"
#include "error.c"

main()
{
  while(1)
   {
   menu001();
  }
}

menu001()
  {
     setraw();
     for (i=1;i<=2;i++)
        for (j=1;j<=12;j++)
           matriz[i][j] = 0;
     dimx = 1;            /*** numero de columnas  de la matriz ***/
     dimy = 12;           /*** numero de renglones de la matriz ***/

/*************************************
 * Posicion de cordenadas iniciales  *
 *************************************/

     renglon = 1;
     columna = 1;
     funcion = 'A';

/***********************************************
 * Posicion de cordenadas columna con renglon  *
 ***********************************************/


     matriz[1][1] = 'A';
     matriz[1][2] = 'B';
     matriz[1][3] = 'C';
     matriz[1][4] = 'D';
     matriz[1][5] = 'E';
     matriz[1][6] = 'F';
     matriz[1][12] = 'Z';
```

```
/*************************
 * Posicion de del texto *
 *************************/

 strcpy(texto[1].c,"A. Global Deptos. Inst.");
 strcpy(texto[2].c,"B. Global Deptos Locales");
 strcpy(texto[3].c,"C. Pasivo Deptos Inst.");
 strcpy(texto[4].c,"D. Pasivo Deptos Locales");
 strcpy(texto[5].c,"E. Solicitud Deptos Inst.");
 strcpy(texto[6].c,"F. Solicitud Deptos Locales");
 strcpy(texto[12].c,"Z. Regreso Menú Principal");

 titulos();
 descripcion(" Actualizacion de Saldos");
 gotoxy(9,27);printf(texto[1].c);
 gotoxy(10,27);printf(texto[2].c);
 gotoxy(11,27);printf(texto[3].c);
 gotoxy(12,27);printf(texto[4].c);
 gotoxy(13,27);printf(texto[5].c);
 gotoxy(14,27);printf(texto[6].c);
 gotoxy(20,27);printf(texto[12].c);

/*******************************************************
* cursor (posicion columna 1 "27", renglon  inicial "9"      *
* cursor  posicion columna 2 "" ,  renglon inicial "9"       *
* cursor  numero de ubicacion de la letra en alfa "27=Z")    *
*******************************************************/
   cursor(27,0,9,27);
   switch (funcion)
     {
       case 'A':
          restore();
          {
          system("nohup exe/actsali.exe >>log.err &");
          }
          break;
       case 'B':
          restore();
          {
          system("nohup exe/actsall.exe >>log.err &");
          }
          break;
       case 'C':
          restore();
          {
          system("nohup exe/actsal2i.exe >>log.err &");
          }
          break;
       case 'D':
          restore();
          {
```

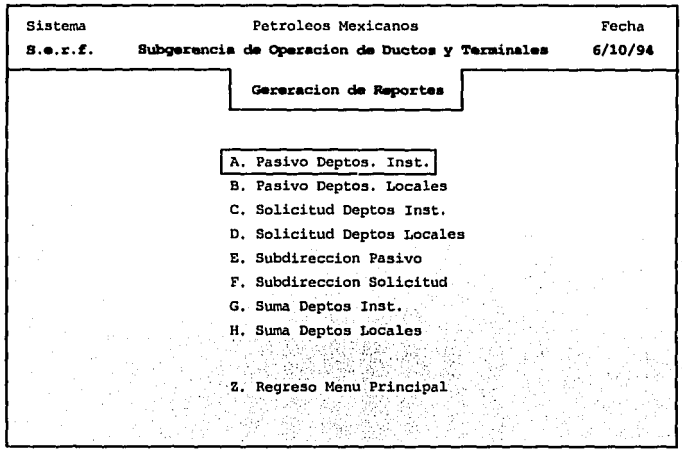
```
            system("nohup exe/actsal2l.exe >>log.err &");
            }
            break;
         case 'E':
            restore();
            {
            system("nohup exe/actsal3i.exe >>log.err &");
            }
            break;
         case 'F':
            restore();
            {
            system("nohup exe/actsal3l.exe >>log.err &");
            }
            break;
         case 'Z':
            limpia();
            restore();
            gotoxy(1,1);
            exit(0);
      }
   }
```



```
/*********************************************************
 *       Sistema de Estados de Resultados Financieros    *
 *                                                       *
 *          Programa que Contiene el Menú de             *
 *             Generacion de Reportes                    *
 *                                                       *
 * Autor: Arturo Leon Juarez          Programa: menú_grpt *
 * Fecha: 01/01/94                    Equipo : hp9000/842 *
 *********************************************************/

#include "menu2.c"
#include "error.c"

main()
{
  while(1)
   {
   menu001();
  }
}

menu001()
  {
    setraw();
    for (i=1;i<=2;i++)
      for (j=1;j<=12;j++)
         matriz[i][j] = 0;
    dimx = 1;            /*** numero de columnas  de la matriz ***/
    dimy = 12;           /*** numero de renglones de la matriz ***/

/*********************************
 * Posicion de cordenadas iniciales *
 *********************************/

    renglon = 1;
    columna = 1;
    funcion = 'A';

/*****************************************
 * Posicion de cordenadas columna con renglon *
 *****************************************/

    matriz[1][1] = 'A';
    matriz[1][2] = 'B';
    matriz[1][3] = 'C';
    matriz[1][4] = 'D';
    matriz[1][5] = 'E';
    matriz[1][6] = 'F';
    matriz[1][7] = 'G';
    matriz[1][8] = 'H';
    matriz[1][12] = 'Z';
```

```
/*********************
 * Posicion de el texto *
 *********************/

    strcpy(texto[1].c,"A. Pasivo  Deptos.  Inst. ");
    strcpy(texto[2].c,"B. Pasivo  Deptos.  Locales ");
    strcpy(texto[3].c,"C. Solicitud Deptos Inst. ");
    strcpy(texto[4].c,"D. Solicitud Deptos Locales ");
    strcpy(texto[5].c,"E. Subdireccion Pasivo");
    strcpy(texto[6].c,"F. Subdireccion Solicitud");
    strcpy(texto[7].c,"G. Suma Deptos Inst.");
    strcpy(texto[8].c,"H. Suma Deptos Locales");
    strcpy(texto[12].c,"Z. Regreso Menú Principal");

    titulos();
    descripcion("Generacion de Reportes");

    gotoxy(9,27);printf(texto[1].c);
    gotoxy(10,27);printf(texto[2].c);
    gotoxy(11,27);printf(texto[3].c);
    gotoxy(12,27);printf(texto[4].c);
    gotoxy(13,27);printf(texto[5].c);
    gotoxy(14,27);printf(texto[6].c);
    gotoxy(15,27);printf(texto[7].c);
    gotoxy(16,27);printf(texto[8].c);
    gotoxy(20,27);printf(texto[12].c);

/*********************************************************
 * cursor (posicion columna 1 "27", renglon  inicial "9"      *
 * cursor  posicion columna 2 ""  ,  renglon inicial "9"      *
 * cursor  numero de ubicacion de la letra en alfa "27=Z")    *
 *********************************************************/

    cursor(27,0,9,27);
    switch (funcion)
      {
        case 'A':
          restore();
          {
          system("exe/rep219i.exe ");
          }
          break;
        case 'B':
          restore();
          {
          system("exe/rep219l.exe ");
          }
          break;
        case 'C':
          restore();
          {
```

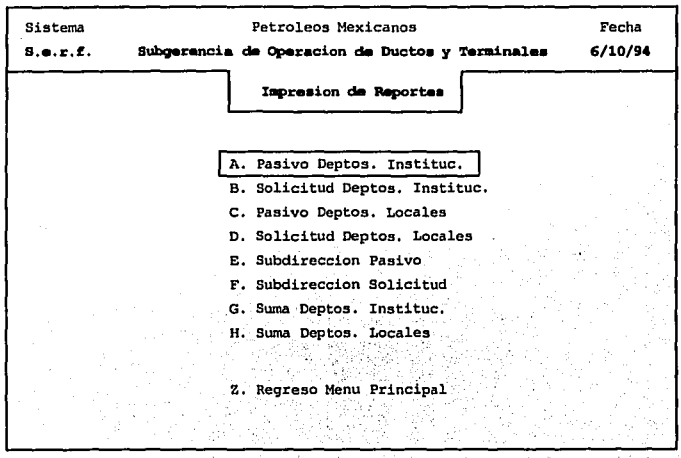
```
            system("exe/rep392i.exe ");
            }
            break;
         case 'D':
            restore();
            {
            system("exe/rep392l.exe ");
            }
            break;
         case 'E':
            restore();
            {
            system("exe/repsub2.exe ");
            }
            break;
         case 'F':
            restore();
            {
            system("exe/repsub3.exe ");
            }
            break;
         case 'G':
            restore();
            {
            system("exe/repsumi.exe ");
            }
            break;
         case 'H':
            restore();
            {
            system("exe/repsuml.exe ");
            }
            break;
         case 'Z':
            limpia();
            restore();
            gotoxy(1,1);
            exit(0);
   }
}
```


Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Gereracion de Reportes
A. Pasivo Deptos. Inst.
B. Pasivo Deptos. Locales
C. Solicitud Deptos Inst.
D. Solicitud Deptos Locales
E. Subdireccion Pasivo
F. Subdireccion Solicitud
G. Suma Deptos Inst.
H. Suma Deptos Locales
Z. Regreso Menu Principal

```
/*********************************************************
 *        Sistema de Estados de Resultados Financieros   *
 *                                                       *
 *           Programa que Contiene el Menú de            *
 *              Impresion de Reporte                     *
 *                                                       *
 * Autor: Arturo Leon Juarez          Programa: menú_impr *
 * Fecha: 01/01/94                     Equipo : hp9000/842 *
 *********************************************************/

#include "menu2.c"
#include "error.c"

main()
{
  while(1)
   {
   menu001();
   }
}

menu001()
  {
     setraw();
     for (i=1;i<=2;i++)
        for (j=1;j<=12;j++)
           matriz[i][j] = 0;
     dimx = 2;              /*** numero de columnas  de la matriz ***/
     dimy = 12;             /*** numero de renglones de la matriz ***/

/*********************************
 * Posicion de cordenadas iniciales *
 *********************************/

     renglon = 1;
     columna = 1;
     funcion = 'A';

/*****************************************
 * Posicion de cordenadas columna con renglon *
 *****************************************/

    matriz[1][1] = 'A';
    matriz[1][2] = 'B';
    matriz[1][3] = 'C';
    matriz[1][4] = 'D';
    matriz[1][5] = 'E';
    matriz[1][6] = 'F';
    matriz[1][7] = 'G';
    matriz[1][8] = 'H';
    matriz[1][12] = 'Z';
```

```
/**************************
 * Posicion de el texto *
 **************************/

   strcpy(texto[1].c,"A. Pasivo Deptos. Instituc.");
   strcpy(texto[2].c,"B. Solicitud Deptos. Instituc.");
   strcpy(texto[3].c,"C. Pasivo Deptos. Locales");
   strcpy(texto[4].c,"D. Solicitud Deptos. Locales ");
   strcpy(texto[5].c,"E. Subdireccion Pasivo");
   strcpy(texto[6].c,"F. Subdireccion Solicitud");
   strcpy(texto[7].c,"G. Suma Deptos. Instituc.");
   strcpy(texto[8].c,"H. Suma Deptos. Locales");
   strcpy(texto[12].c,"Z. Regreso Menú Principal");

   titulos();
   descripcion(" Impresion de Reportes");

   gotoxy(9,27);printf(texto[1].c);
   gotoxy(10,27);printf(texto[2].c);
   gotoxy(11,27);printf(texto[3].c);
   gotoxy(12,27);printf(texto[4].c);
   gotoxy(13,27);printf(texto[5].c);
   gotoxy(14,27);printf(texto[6].c);
   gotoxy(15,27);printf(texto[7].c);
   gotoxy(16,27);printf(texto[8].c);
   gotoxy(20,27);printf(texto[12].c);

/*****************************************************************
 * cursor (posicion columna 1 "27",  renglon inicial "9"      *
 * cursor  posicion columna 2 "0",   renglon inicial "9"      *
 * cursor  numero de ubicacion de la letra en alfa "27=Z")    *
 *****************************************************************/

   cursor(27,0,9,27);
   switch (funcion)
     {
       case 'A':
          restore();
          {
          system(".lp_imp tmp/rep219i.lis");
          }
          break;
       case 'B':
          restore();
          {
          system(".lp_imp tmp/rep392i.lis");
          }
          break;
       case 'C':
          restore();
```

```
            {
            system(".lp_imp tmp/rep219l.lis");
            }
            break;
         case 'D':
            restore();
            {
            system(".lp_imp tmp/rep392l.lis");
            }
            break;
         case 'E':
            restore();
            {
            system(".lp_imp tmp/repsub2.lis");
            }
            break;
         case 'F':
            restore();
            {
            system(".lp_imp tmp/repsub3.lis");
            }
            break;
         case 'G':
            restore();
            {
            system(".lp_imp tmp/repsumi.lis");
            }
            break;
         case 'H':
            restore();
            {
            system(".lp_imp tmp/repsuml.lis");
            }
            break;
         case 'Z':
            limpia();
            gotoxy(1,1);
            restore();
            exit(0);
   }
}
```


Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Impresion de Reportes
A. Pasivo Deptos. Instituc.
B. Solicitud Deptos. Instituc.
C. Pasivo Deptos. Locales
D. Solicitud Deptos. Locales
E. Subdireccion Pasivo
F. Subdireccion Solicitud
G. Suma Deptos. Instituc.
H. Suma Deptos. Locales
Z. Regreso Menu Principal

```
/*************************************************************
 *        Sistema de Estados de Resultados Financieros       *
 *                                                           *
 *           Programa que Contiene el Menú de                *
 *              Monitoreo de Procesos                        *
 *                                                           *
 * Autor: Arturo Leon Juarez          Programa: menú_mtor    *
 * Fecha: 01/01/94                     Equipo  : hp9000/842  *
 *************************************************************/

#include "menu2.c"
#include "error.c"

main()
{
  while(1)
   {
   menu001();
  }
}

menu001()
  {
     setraw();
     for (i=1;i<=2;i++)
        for (j=1;j<=12;j++)
           matriz[i][j] = 0;
     dimx = 1;              /*** numero de columnas  de la matriz ***/
     dimy = 12;             /*** numero de renglones de la matriz ***/

/*********************************
 * Posicion de cordenadas iniciales *
 *********************************/

     renglon = 1;
     columna = 1;
     funcion = 'A';

/*****************************************
 * Posicion de cordenadas columna con renglon *
 *****************************************/

     matriz[1][1] = 'A';
     matriz[1][2] = 'B';
     matriz[1][3] = 'C';
     matriz[1][4] = 'D';
     matriz[1][5] = 'E';
     matriz[1][6] = 'F';
     matriz[1][12] = 'Z';
```

```
/**************************
 * Posicion de  el texto *
 **************************/

 strcpy(texto[1].c,"A. Actualizacion Saldos Inst.");
 strcpy(texto[2].c,"B. Actualizacion Saldos Locales");
 strcpy(texto[3].c,"C. Actualizacion Pasivo  Inst.");
 strcpy(texto[4].c,"D. Actualizacion Solctd. Inst.");
 strcpy(texto[5].c,"E. Actualizacion Pasivo  Locales");
 strcpy(texto[6].c,"F. Actualizacion Solctd. Locales");
 strcpy(texto[12].c,"Z. Salida al Menú Principal");

  titulos();
  descripcion("Menú de Monitoreo");

  gotoxy(9,27);printf(texto[1].c);
  gotoxy(10,27);printf(texto[2].c);
  gotoxy(11,27);printf(texto[3].c);
  gotoxy(12,27);printf(texto[4].c);
  gotoxy(13,27);printf(texto[5].c);
  gotoxy(14,27);printf(texto[6].c);
  gotoxy(20,27);printf(texto[12].c);

/*********************************************************************
 * cursor (posicion columna 1 "27", renglon  inicial "9"       *
 * cursor  posicion columna 2 ""  ,  renglon inicial "9"       *
 * cursor  numero de ubicacion de la letra en alfa "27=Z")     *
 *********************************************************************/

  cursor(27,0,9,27);
  switch (funcion)
    {
      case 'A':
        restore();
        {
        system("clear;pg -p 'Teclear <Retorn>' bit/actsall.bit");
        }
        break;
      case 'B':
        restore();
        {
        system("clear; pg -p 'Teclear <Retorn>' bit/actsall.bit");
        }
        break;
      case 'C':
        restore();
        {
        system("clear; pg -p 'Teclear <Retorn>' bit/actsal2i.bit");
        }
        break;
      case 'D':
```

```
            restore();
            {
            system("clear; pg -p 'Teclear <Retorn>' bit/actsal3i.bit");
            }
            break;
         case 'E':
            restore();
            {
            system("clear; pg -p 'Teclear <Retorn>' bit/actsal2l.bit");
            }
            break;
         case 'F':
            restore();
            {
            system("clear; pg -p 'Teclear <Retorn>' bit/actsal3l.bit");
            }
            break;
         case 'Z':
            limpia();
            gotoxy(1,1);
            restore();
            exit(0);
      }
   }
```

Sistema S.e.r.f. | Petroleos Mexicanos | Subgerencia de Operacion de Ductos y Terminales | Fecha 6/10/94

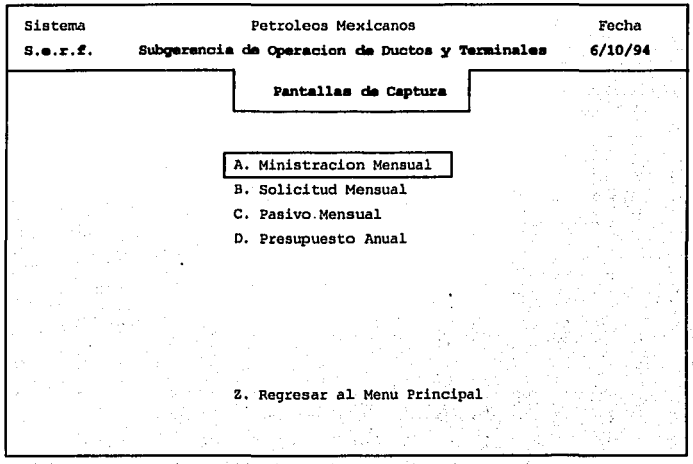
Menu de Monitoreo

A. Actualizacion de Saldos Inst.
B. Actualizacion de Saldos Locales
C. Actualizacion Pasivo Inst.
D. Actualizacion Solctd. Inst.
E. Actualizacion Pasivo Locales
F. Actualizacion Solctd. Locales

Z. Salir al Menu Principal

```
/*************************************************************/
 *        Sistema de Estados de Resultados Financieros       *
 *                                                           *
 *           Programa que Contiene el Menú de                *
 *              Pantallas de Captura                         *
 *                                                           *
 * Autor: Arturo Leon Juarez          Programa: menú_pcap    *
 * Fecha: 01/01/94                    Equipo : hp9000/842    *
 *************************************************************/
#include "menu2.c"
#include "error.c"

main()
{
  while(1)
   {
   menu001();
   }
}

menu001()
  {
     setraw();
     for (i=1;i<=2;i++)
        for (j=1;j<=12;j++)
           matriz[i][j] = 0;
     dimx = 1;              /*** numero de columnas  de la matriz ***/
     dimy = 12;             /*** numero de renglones de la matriz ***/

/*********************************
 * Posicion de cordenadas iniciales *
 *********************************/
     renglon = 1;
     columna = 1;
     funcion = 'A';
/*****************************************
 * Posicion de cordenadas columna con renglon *
 *****************************************/
     matriz[1][1] = 'A';
     matriz[1][2] = 'B';
     matriz[1][3] = 'C';
     matriz[1][4] = 'D';
     matriz[1][12] = 'Z';
   /*************************
    * Posicion de  el texto *
    *************************/
     strcpy(texto[1].c,"A. Ministracion Mensual ");
     strcpy(texto[2].c,"B. Solicitud Mensual ");
     strcpy(texto[3].c,"C. Pasivo Mensual ");
     strcpy(texto[4].c,"D. Presupuesto Anual ");
     strcpy(texto[12].c,"Z. Regreso Menú Principal");
```

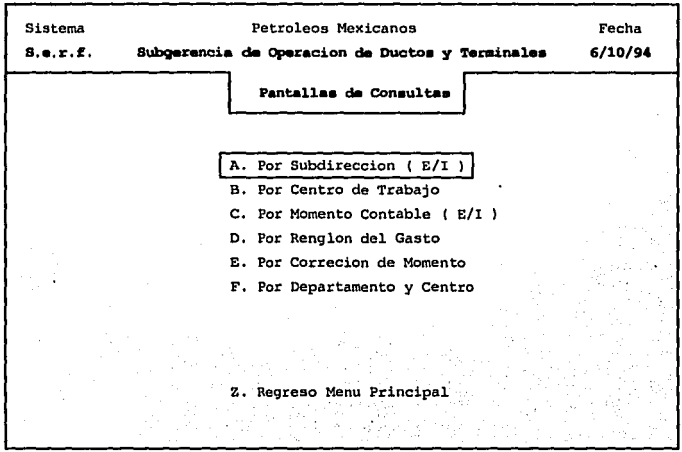
```
    titulos();
    descripcion(" Pantallas de Captura");
    gotoxy(9,27);printf(texto[1].c);
    gotoxy(10,27);printf(texto[2].c);
    gotoxy(11,27);printf(texto[3].c);
    gotoxy(12,27);printf(texto[4].c);
    gotoxy(20,27);printf(texto[12].c);

/*****************************************************
 * cursor (posicion columna 1 "27", renglon  inicial "9"      *
 * cursor (posicion columna 2 "",    renglon  inicial "9"     *
 * cursor numero de ubicacion de la letra en alfa "27=Z")    *
 *****************************************************/

    cursor(27,0,9,27);
    switch (funcion)
     {
       case 'A':
          restore();
          {
           system("runform30  cap/PRESP $SI_USR/$SI_PWD");
          }
          break;
       case 'B':
          restore();
          {
           system("runform30  cap/SOLCT $SI_USR/$SI_PWD");
          }
          break;
       case 'C':
          restore();
          {
           system("runform30  cap/PASIVO $SI_USR/$SI_PWD");
          }
          break;
       case 'D':
          restore();
          {
           system("runform30  cap/PRE_ADPT $SI_USR/$SI_PWD");
          }
          break;
       case 'Z':
           limpiu();
           gotoxy(1,1);
           restore();
           exit(0);
     }
  }
```


Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Pantallas de Captura
A. Ministracion Mensual
B. Solicitud Mensual
C. Pasivo Mensual
D. Presupuesto Anual
Z. Regresar al Menu Principal

```
/*************************************************************
 *        Sistema de Estados de Resultados Financieros       *
 *                                                           *
 *           Programa que Contiene el Menú de                *
 *              Pantallas de Consulta                        *
 *                                                           *
 * Autor: Arturo Leon Juarez          Programa: menú_pcon    *
 * Fecha: 01/01/94                     Equipo : hp9000/842   *
 *************************************************************/

#include "menu2.c"
#include "error.c"

main()
{
  while(1)
   {
   menu001();
  }
}

menu001()
  {
     setraw();
     for (i=1;i<=2;i++)
        for (j=1;j<=12;j++)
           matriz[i][j] = 0;
     dimx = 1;              /*** numero de columnas  de la matriz ***/
     dimy = 12;             /*** numero de renglones de la matriz ***/

/*********************************
 * Posicion de cordenadas iniciales *
 *********************************/

     renglon = 1;
     columna = 1;
     funcion = 'A';

/*****************************************
 * Posicion de cordenadas columna con renglon *
 *****************************************/

     matriz[1][1] = 'A';
     matriz[1][2] = 'B';
     matriz[1][3] = 'C';
     matriz[1][4] = 'D';
     matriz[1][5] = 'E';
     matriz[1][6] = 'F';
     matriz[1][12] = 'Z';
```

```
/*********************
 * Posicion de  el texto *
 *********************/

   strcpy(texto[1].c,"A. Por Subdireccion (E/I)");
   strcpy(texto[2].c,"B. Por Centro de Trabajo");
   strcpy(texto[3].c,"C. Por Momento Contable (E/I)");
   strcpy(texto[4].c,"D. Por Renglon del Gasto");
   strcpy(texto[5].c,"E. Por Correccion de Momento");
   strcpy(texto[6].c,"F. Por Departamento y Centro ");
   strcpy(texto[12].c,"Z. Regreso Menú Principal");

   titulos();
   descripcion(" Pantallas de Consulta");
   gotoxy(9,27);printf(texto[1].c);
   gotoxy(10,27);printf(texto[2].c);
   gotoxy(11,27);printf(texto[3].c);
   gotoxy(12,27);printf(texto[4].c);
   gotoxy(13,27);printf(texto[5].c);
   gotoxy(14,27);printf(texto[6].c);
   gotoxy(20,27);printf(texto[12].c);

/*******************************************************
 * cursor (posicion columna 1 "27",  renglon inicial "9"      *
 * cursor  posicion columna 2 "" ,    renglon inicial "9"     *
 * cursor numero de ubicacion de la letra en alfa "27=Z")    *
 *******************************************************/

   cursor(27,0,9,27);
   switch (funcion)
     {
      case 'A':
         restore();
         {
          system("runform30  pant/CAPSUBDI $SI_USR/$SI_PWD");
         }
         break;
      case 'B':
         restore();
         {
          system("runform30 pant/CENTRO $SI_USR/$SI_PWD");
         }
         break;
      case 'C':
         restore();
         {
          system("runform30 pant/MOMENTO $SI_USR/$SI_PWD");
         }
         break;
      case 'D':
         restore();
```

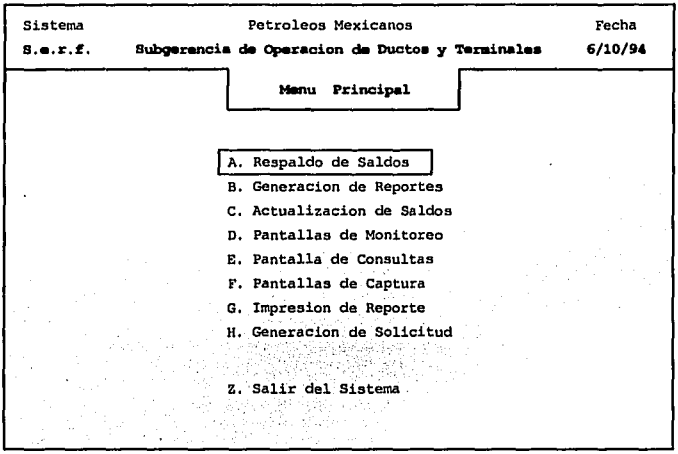
```
        {
         system("runform30 pant/RENGLON $SI_USR/$SI_PWD");
        }
        break;
      case 'E':
        restore();
        {
         system("runform30 pant/MOMCMAL $SI_USR/$SI_PWD");
        }
        break;
      case 'F':
        restore();
        {
         system("runform30 pant/SALDOA $SI_USR/$SI_PWD");
        }
        break;
      case 'Z':
        limpia();
        restore();
        gotoxy(1,1);
        exit(0);
    }
}
```


Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Pantallas de Consultas
A. Por Subdireccion (E/I)
B. Por Centro de Trabajo
C. Por Momento Contable (E/I)
D. Por Renglon del Gasto
E. Por Correcion de Momento
F. Por Departamento y Centro
Z. Regreso Menu Principal

```
/*************************************************************
 *        Sistema de Estados de Resultados Financieros       *
 *                                                           *
 *          Programa que Contiene el Menú de                 *
 *                Principal                                  *
 *                                                           *
 * Autor: Arturo Leon Juarez          Programa: menú_princ   *
 * Fecha: 01/01/94                    Equipo  : hp9000/842   *
 *************************************************************/

#include "menu2.c"
#include "error.c"

main()
  {
   while (1)
    {
     menu001();
    }
  }

menu001()
  {
    setraw();
    for (i=1;i<=3;i++)
       for (j=1;j<=8;j++)
          matriz[i][j] = 0;
    dimx = 1;              /*** numero de columnas  de la matriz ***/
    dimy = 12;             /*** numero de renglones de la matriz ***/

/*******************************
 * Posicion de cordenadas iniciales *
 *******************************/

    renglon = 1;
    columna = 1;
    funcion = 'A';

/***********************************************
 * Posicion de cordenadas columna con renglon *
 ***********************************************/

    matriz[1][1] = 'A';
    matriz[1][2] = 'B';
    matriz[1][3] = 'C';
    matriz[1][4] = 'D';
    matriz[1][5] = 'E';
    matriz[1][6] = 'F';
    matriz[1][7] = 'G';
    matriz[1][8] = 'H';
    matriz[1][12] = 'Z';
```

```
/*********************
 * Posicion de  el texto *
 *********************/

   strcpy(texto[1].c,"A. Respaldo de Saldos ");
   strcpy(texto[2].c,"B. Generacion de Reportes ");
   strcpy(texto[3].c,"C. Actualizacion de Saldos ");
   strcpy(texto[4].c,"D. Pantallas de Monitoreo ");
   strcpy(texto[5].c,"E. Pantallas de Consulta ");
   strcpy(texto[6].c,"F. Pantallas de Captura ");
   strcpy(texto[7].c,"G. Impresion de Reportes ");
   strcpy(texto[8].c,"H. Generacion de Solicitud");
   strcpy(texto[12].c,"Z. Salida del Sistema");

   titulos();
   descripcion("Menú Principal");
   gotoxy(9,27);printf(texto[1].c);
   gotoxy(10,27);printf(texto[2].c);
   gotoxy(11,27);printf(texto[3].c);
   gotoxy(12,27);printf(texto[4].c);
   gotoxy(13,27);printf(texto[5].c);
   gotoxy(14,27);printf(texto[6].c);
   gotoxy(15,27);printf(texto[7].c);
   gotoxy(16,27);printf(texto[8].c);
   gotoxy(20,27);printf(texto[12].c);

/*********************************************************
 * cursor (posicion columna 1 "27", renglon  inicial "9"      *
 * cursor  posicion columna 2 ""  ,  renglon inicial "9"      *
 * cursor numero de ubicacion de la letra en alfa "27=Z")     *
 *********************************************************/

   cursor(27,0,9,27);
   switch (funcion)
     {
       case 'A':
         restore();
         {
         system("menús/menú_rsal");
         }
         break;
       case 'B':
         restore();
         {
         system("menús/menú_grpt");
         }
         break;
       case 'C':
         restore();
         {
```

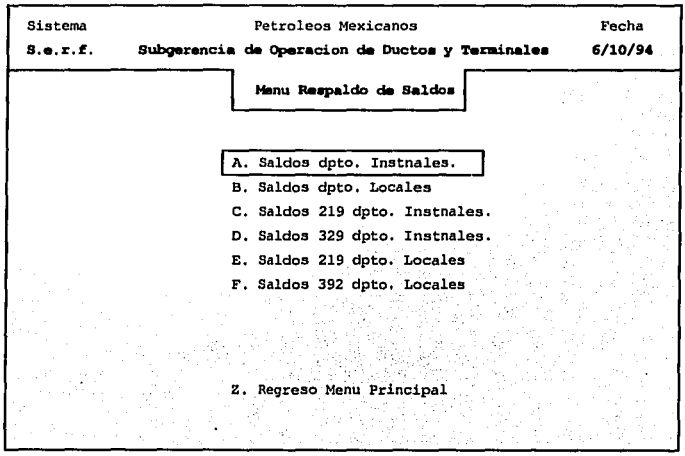
```
            system("menús/menú_asal");
            }
            break;
        case 'D':
            restore();
            {
            system("menús/menú_mtor");
            }
            break;
        case 'E':
            restore();
            {
            system("menús/menú_pcon");
            }
            break;
        case 'F':
            restore();
            {
            system("menús/menú_pcap");
            }
            break;
        case 'G':
            restore();
            {
            system("menús/menú_impr");
            }
            break;
        case 'H':
            restore();
            {
            system("menús/menú_solc");
            }
            break;
        case 'Z':
            limpia();
            restore();
            gotoxy(1,1);
            exit(0);
    }
}
```


Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Menu Principal
A. Respaldo de Saldos
B. Generacion de Reportes
C. Actualizacion de Saldos
D. Pantallas de Monitoreo
E. Pantalla de Consultas
F. Pantallas de Captura
G. Impresion de Reporte
H. Generacion de Solicitud
Z. Salir del Sistema

```
/*********************************************************
 *        Sistema de Estados de Resultados Financieros   *
 *                                                       *
 *           Programa que Contiene el Menú de            *
 *               Respaldo de Saldos                      *
 *                                                       *
 * Autor: Arturo Leon Juarez          Programa: menú_rsal *
 * Fecha: 01/01/94                    Equipo  : hp9000/842 *
 *********************************************************/

#include "menu2.c"
#include "error.c"

main()
{
  while(1)
   {
   menu001();
   }
}

menu001()
  {
     setraw();
     for (i=1;i<=2;i++)
        for (j=1;j<=12;j++)
            matriz[i][j] = 0;
     dimx = 1;            /*** numero de columnas  de la matriz ***/
     dimy = 12;           /*** numero de renglones de la matriz ***/

/*********************************
 * Posicion de cordenadas iniciales *
 *********************************/

     renglon = 1;
     columna = 1;
     funcion = 'A';

/*****************************************
 * Posicion de cordenadas columna con renglon *
 *****************************************/

     matriz[1][1] = 'A';
     matriz[1][2] = 'B';
     matriz[1][3] = 'C';
     matriz[1][4] = 'D';
     matriz[1][5] = 'E';
     matriz[1][6] = 'F';
     matriz[1][12] = 'Z';
```

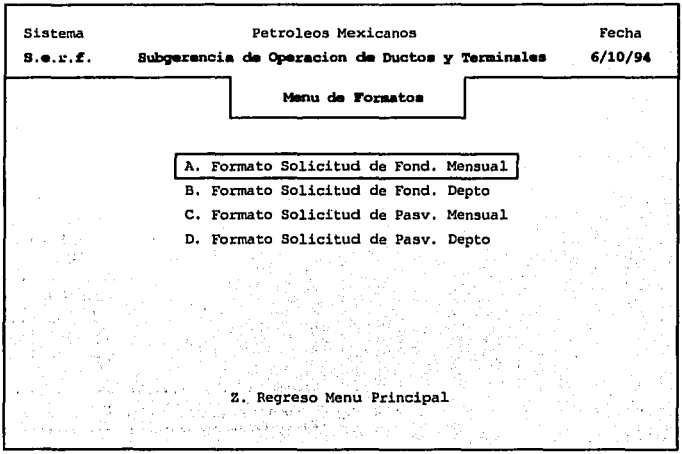
```
/*********************
 * Posicion de  el texto *
 *********************/

   strcpy(texto[1].c,"A. Saldos dpto. Instnales. ");
   strcpy(texto[2].c,"B. Saldos dpto. Locales ");
   strcpy(texto[3].c,"C. Saldos 219 dpto. Instnales.");
   strcpy(texto[4].c,"D. Saldos 392 dpto. Instnales.");
   strcpy(texto[5].c,"E. Saldos 219 dpto. Locales");
   strcpy(texto[6].c,"F. Saldos 392 dpto. Locales");
   strcpy(texto[12].c,"Z. Regreso Menú Principal");

   titulos();
   descripcion(" Menú Respaldo de Saldos");

   gotoxy(9,27);printf(texto[1].c);
   gotoxy(10,27);printf(texto[2].c);
   gotoxy(11,27);printf(texto[3].c);
   gotoxy(12,27);printf(texto[4].c);
   gotoxy(13,27);printf(texto[5].c);
   gotoxy(14,27);printf(texto[6].c);
   gotoxy(20,27);printf(texto[12].c);

/*****************************************************
 * cursor (posicion columna 1 "27", renglon  inicial "9"      *
 * cursor posicion columna 2 ""  ,   renglon inicial "9"      *
 * cursor numero de ubicacion de la letra en alfa "27=Z")     *
 *****************************************************/

   cursor(27,0,9,27);
   switch (funcion)
     {
       case 'A':
          restore();
          {
           system("bin/ressali $SI_USR $SI_PWD");
          }
          break;
       case 'B':
          restore();
          {
           system("bin/ressali $SI_USR $SI_PWD");
          }
          break;
       case 'C':
          restore();
          {
           system("bin/ressal2i $SI_USR $SI_PWD ");
          }
          break;
       case 'D':
```

```
            restore();
            {
             system("bin/ressal3i $SI_USR $SI_PWD");
            }
            break;
          case 'E':
            restore();
            {
             system("bin/ressal2i $SI_USR $SI_PWD");
            }
            break;
          case 'F':
            restore();
            {
             system("bin/ressal3i $SI_USR $SI_PWD");
            }
            break;
          case 'Z':
            limpia();
            gotoxy(1,1);
            restore();
            exit(0);
    }
}
```


Sistema
S.e.r.f.
Petroleos Mexicanos
Subgerencia de Operacion de Ductos y Terminales
Fecha
6/10/94
Menu Respaldo de Saldos
A. Saldos dpto. Instnales.
B. Saldos dpto. Locales
C. Saldos 219 dpto. Instnales.
D. Saldos 329 dpto. Instnales.
E. Saldos 219 dpto. Locales
F. Saldos 392 dpto. Locales
Z. Regreso Menu Principal

```
/*************************************************************
 *        Sistema de Estados de Resultados Financieros       *
 *                                                           *
 *          Programa que Contiene el Menú para               *
 *            el Formato de solc. de Fondh.                  *
 *                                                           *
 * Autor: Arturo Leon Juarez          Programa: menú_solc    *
 * Fecha: 01/01/94                    Equipo  : hp9000/842   *
 *************************************************************/
#include "menu2.c"
#include "error.c"
main()
{
  while(1)
   {
   menu001();
  }
}

menu001()
  {
    setraw();
    for (i=1;i<=2;i++)
       for (j=1;j<=12;j++)
          matriz[i][j] = 0;
    dimx = 1;              /*** numero de columnas  de la matriz ***/
    dimy = 12;             /*** numero de renglones de la matriz ***/
/*********************************
 * Posicion de cordenadas iniciales *
 *********************************/
    renglon = 1;
    columna = 1;
    funcion = 'A';
/*****************************************
 * Posicion de cordenadas columna con renglon *
 *****************************************/
    matriz[1][1] = 'A';
    matriz[1][2] = 'B';
    matriz[1][3] = 'C';
    matriz[1][4] = 'D';
    matriz[1][12] = 'Z';
  /*********************
   * Posicion de el texto *
   *********************/
    strcpy(texto[1].c,"A. Formato Solicitud de Fond. Mensual");
    strcpy(texto[2].c,"B. Formato Solicitud de Fond. Depto");
    strcpy(texto[3].c,"C. Formato Solicitud de Pasv. Mensual ");
    strcpy(texto[4].c,"D. Formato Solicitud de Pasv. Depto.");
    strcpy(texto[12].c,"Z. Regresar al Menú Principal");
    titulos();
```

```
    descripcion(" Pantallas de Captura");
    gotoxy(9,22);printf(texto[1].c);
    gotoxy(10,22);printf(texto[2].c);
    gotoxy(11,22);printf(texto[3].c);
    gotoxy(12,22);printf(texto[4].c);
    gotoxy(20,22);printf(texto[12].c);
/*******************************************************
* cursor (posicion columna 1 "27", renglon  inicial "9"      *
* cursor (posicion columna 2 "",    renglon  inicial "9"      *
* cursor  numero de ubicacion de la letra en alfa "27=Z")    *
*******************************************************/
    cursor(22,0,9,27);
    switch (funcion)
     {
       case 'A':
          restore();
          {
           system("exe/mes392.exe ");
          }
          break;
       case 'B':
          restore();
          {
           system("exe/solc392.exe ");
          }
          break;
       case 'C':
          restore();
          {
           titulos();
           gotoxy(14,30);printf("Opcion Inactiva ");
           gotoxy(22,50);printf("Para Regresar Oprima RETURN ");
           getchar();
          }
          break;
       case 'D':
          restore();
          {
           titulos();
           gotoxy(14,30);printf("Opcion Inactiva ");
           gotoxy(22,50);printf("Para Regresar Oprima RETURN ");
           getchar();
          }
          break;
       case 'Z':
           limpia();
           gotoxy(1,1);
           restore();
           exit(0);
     }
  }
```


Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Menu de Formatos
A. Formato Solicitud de Fond. Mensual
B. Formato Solicitud de Fond. Depto
C. Formato Solicitud de Pasv. Mensual
D. Formato Solicitud de Pasv. Depto
Z. Regreso Menu Principal

```
/*********************************************************
 *        Sistema de Estados de Resultados Financieros   *
 *                                                       *
 *       Libreria que Contiene las Rutinas del Tiempo,   *
 *         de el Recuadro y Posicion del Cursosr         *
 *                                                       *
 * Autor: Arturo Leon Juarez              Programa: menu2 *
 * Fecha: 01/01/94                      Equipo : hp9000/842 *
 *********************************************************/

#define INV printf("\033[7m")
#define NOR printf("\033[0m")
#define BLI printf("\033[5m")
#define BRI printf("\033[1m")
#include <termio.h>
#include <errno.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>

/*****************************************
 * variables globales del sistema de menú's  *
 *****************************************/

struct tm *ptime, *localtime();
time_t time(), nseg;
int dia,mes,ano;

int dimx, dimy;
int renglon;
int columna;
int t, l, j;
int opcion;
int matriz[5][15];
int funcion;
int x, y;

  struct ren {
         char  c[40];
         };
  struct ren texto[];

/************************************
 * procedimiento para limpiar la pantalla *
 ************************************/

limpia()
  {
     printf("\033[2J");        /* borra pantalla */
  }
```

```
/*********************************************
 * procedimiento para posicionarse en la pantalla *
 * parametros:                                    *
 *          x - renglon de la pantalla            *
 *          y - columna de la pantalla            *
 *********************************************/
gotoxy(x,y)
  int x,y;
   {
     char xx[3],yy[3];
     sprintf(xx,"%d\0",x);
     sprintf(yy,"%d\0",y);
     printf("\033[%s;%sH",xx,yy);
   }

/*************************************************
 * procedimiento para dibujar un rectangulo en pantalla  *
 * parametros:                                           *
 *          x1 - renglon de la esquina superior          *
 *               izquierda del rectangulo                *
 *          y1 - columna de la esquina superior          *
 *               izquierda del rectangulo                *
 *          x2 - renglon de la esquina inferior          *
 *               derecha del rectangulo                  *
 *          y2 - columna de la esquina inferior          *
 *               derecha del rectangulo                  *
 *************************************************/
box(x1,y1,x2,y2,x3)          /* procedimiento que hace el rectangulo  */
 int x1,y1,x2,y2,x3;         /* declaro las cordenadas de los 2 puntos */
 {
 register int j;
 printf("\033(0");            /* inicio modo grafico              */
gotoxy(x1,y1);                /* me posesiono en el punto de inicio  */
    printf("l\033[D\033[B");  /* pongo la esquina superior derecha  */
for (j=1;j<x2-x1;j++)         /* pinto la linea derecha           */
    printf("x\033[B\033[D");
    printf("m");              /* pongo la esquina inferior derecha  */
for (j=1;j<y2-y1;j++)         /* pinto la linea inferior          */
    printf("q");
    printf("j");              /* pinto la esquina inferior izquierda */
for (j=1;j<x2-x1;j++)         /* pinto la linea izquierda         */
    printf("\033[A\033[Dx");
    printf("\033[A\033[Dk");  /* pinto la esquina superior izquierda */
for (j=1;j<y2-y1;j++)         /* pinto la linea superior          */
    printf("\033[2Dq");
 gotoxy(x3,y1);
    printf("t");              /* union izquierda de lineas  */
 for (j=1;j<26;j++)
    printf("q");              /* linea horizontal         */
    printf("k\033[D\033[B");  /* esquina superior derecha  */
 for (j=1;j<2;j++)
```

```
    printf("x\033[B\033[D");   /* line vertical          */
    printf("m");               /* esquina inferior derecha  */
  for (j=1;j<26;j++)
    printf("q");               /* linea horixontal       */
    printf("j");               /* esquina inferior izquierda */
  for (j=1;j<2;j++)
    printf("\033[A\033[Dx");   /* linea vertical         */
    printf("\033[A\033[Dl");   /* esquina superior izquierda */
  for (j=1;j<26;j++)
    printf("q");               /* linea horizontal       */
    printf("u");               /* union derecha de lineas  */
    printf("\033(B");          /* salir del modo grafico   */
 }
/************************************************
 * procedimiento para desplegar los titulos en pantalla *
 ************************************************/
titulos()
 {
  int j;
  limpia();
  box(1,1,24,79,4);
  gotoxy(2,30);printf("Petroleos Mexicanos");
  gotoxy(3,15);printf("Subgerencia de Operacion de Ductos y Terminales");
  gotoxy(2,4);printf("Sistema");
  gotoxy(3,4);printf("\033[1mS.e.r.f\033[0m");
  fecha();
  gotoxy(2,70);printf(" Fecha ");
  gotoxy(3,68);printf("\033[1m %2d/%02d/%02d \033[0m",dia,mes,ano);
 }
/***********************************
 * Rutina que nos estrega la ubicacion  *
 * del enacbezado principal con letras  *
 * mas brillosasa                       *
 ***********************************/
descripcion(letrero)
char letrero[70];
{
 int columna,longitud;
 longitud=strlen(letrero);
 columna=(80 - longitud) / 2;
 gotoxy(5,columna);
 BRI;
 printf(letrero);
 NOR;
}

/************************************************************
 * procedimiento para encontrar la posicion en pantalla (x,y)      *
 * en donde inicia la opcion deseada, ademas de calcular la        *
 * posicion del texto dentro de la matriz de textos de opciones    *
 * parametros:                                                     *
```

```
*          pc1 - columna de la pantalla en donde se encuentra     *
*              la opcion de la primera columna de opciones.       *
*          pc2 - columna de la pantalla en donde se encuentra     *
*              la opcion de la segunda columna de opciones.       *
*          pr  - renglon de la pantalla en donde inicia las       *
*              opciones.                                          *
*************************************************************/
transforma(pc1,pc2,pr)
    int pc1,pc2,pr;
  {
    if (columna == 1)
       y = pc1;
    if (columna == 2)
       y = pc2;

    x = renglon + pr - 1;
    t = columna * dimy - dimy + renglon;
  }

/*************************************************
 *  Rutina para moverse en el menú con las flechas o  *
 *  tecleando la letra a de la opcion deseada.        *
 *                                                    *
 * Parametros:                                        *
 *                                                    *
 *   poscol1 - Se posiciona en el primer renglon de la *
 *         primer columna del menú.                   *
 *                                                    *
 *   poscol1 - Se posiciona en el primer renglon de la *
 *         segunda columna del menú.                  *
 *                                                    *
 *   posren  - numero de renglon de inicio            *
 *                                                    *
 *   nopcion - numero asignado a la letras a seleccionar *
 *************************************************/
cursor(poscol1,poscol2,posren,nopcion)
  int poscol1,poscol2,posren,nopcion;
  {
    transforma(poscol1,poscol2,posren,nopcion);
    gotoxy(x,y);
    printf("\033[7m");
    printf(texto[t]);
    printf("\033[0m");
    while((opcion = getchar())!=13)
      {
        opcion = toupper(opcion);
        if (opcion == 88)
          {
           funcion=opcion;
           break;
          }
```

```
if (opcion == 27)
  {
    opcion = getchar();
    if (opcion == 91)
      {
        transforma(poscol1,poscol2,posren);
        gotoxy(x,y);
        printf(texto[t]);
        opcion = getchar();
        switch (opcion)
          {
            case 65:
             renglon--;
             if (renglon <= 0)
               renglon = dimy;
               while (matriz[columna][renglon] == 0)
                 {
                 if (renglon <= 0)
                   renglon=dimy;
                 else
                   renglon--;
                 }
            break;
            case 66:
             renglon++;
             if (renglon > dimy)
               renglon = 1;
               while (matriz[columna][renglon] == 0)
                 {
                 if (renglon > dimy)
                   renglon = 1;
                 else
                   renglon++;
                 }
            break;
            case 68:
             columna++;
             if (columna > dimx)
               columna = 1;
               while (matriz[columna][renglon] == 0)
                 {
                 if (columna > dimx)
                   columna = 1;
                 else
                   columna++;
                 }
            break;
            case 67:
             columna--;
             if (columna <= 0)
               columna = dimx;
```

```
                        while (matriz[columna][renglon] == 0)
                          {
                          if (columna <= 0)
                            columna=dimx;
                          else
                            columna--;
                          }
                      break;
                    }
                    funcion = matriz[columna][renglon];
                  }
            }
          else
            {
            if (((opcion > 64) && (opcion < 65 + nopcion)) ||
              (opcion == 88) || (opcion ==90))
              {
                transforma(poscol1,poscol2,posren);
                gotoxy(x,y);
                printf(texto[t]);
                funcion = toupper(opcion);
                funcion = opcion;
                for (i=1;i<=dimx;i++)
                  for (j=1;j<=dimy;j++)
                    if (matriz[i][j] == funcion)
                      {
                        columna = i;
                        renglon = j;
                        goto found;
                      }
              }
            }
found:
    transforma(poscol1,poscol2,posren);
    gotoxy(x,y);
    printf("\033[7m");
    printf(texto[t]);
    printf("\033[0m");
  }
 }
fecha()
{
  dia=mes=ano=0;
  nseg=time(NULL);
  ptime=localtime(&nseg);

  ano = ptime->tm_year;
  mes += (ptime->tm_mon+1);
  dia += (ptime->tm_mday);
}
```

```
/*************************************************************
 *        Sistema de Estados de Resultados Financieros       *
 *         Programa que Genera el Marco Fecha actual         *
 *                                                           *
 *                                                           *
 * Autor:. Arturo Leon Juarez          Programa: marco       *
 * Fecha: 01/01/94                   Equipo  : hp9000/842 *
 *************************************************************/
#include <stdio.h>
#include <time.h>
struct tm *ptime, *localtime();
time_t time(), nseg;

int dia;
int mes;
int ano;

main()
{
  int j;
  limpia();
  box(1,1,24,79,4);
  ubica(2,30);printf("Petroleos Mexicanos");
  ubica(3,15);printf("Subgerencia de Operacion de Ductos y Terminales");
  ubica(2,4);printf("Sistema");
  ubica(3,4);printf("\033[1mS.e.r.f\033[0m");
  fecha();
  ubica(2,70);printf(" Fecha ");
  ubica(3,68);printf("\033[1m %2d/%02d/%02d \033[0m",dia,mes,ano);
}

limpia()
{
  printf("\033[2J");
}

box(regi,coli,regf,colf,regm)
 int regi,coli,regf,colf,regm;
 {
 char lin_sup[82];
 char lin_inf[82];
 char lin_med[82];
 int va;

 if (regi<1 || regf>24 || coli<1 || colf>80)
    {
    fprintf ( stderr, "marco : parametros fuera de rango\n");
    system ( "sleep 2");
    exit (-1);
    }
 printf("\033(0");        /* Entrada Modo Grafico */
```

```
  strcpy (lin_sup, "l");
  strcpy (lin_inf, "m");
  strcpy (lin_med, "t");
  for (va=1; va<colf-coli; va++)
    {
    strcat (lin_sup, "q");
    strcat (lin_inf, "q");
    strcat (lin_med, "q");
    }
  strcat (lin_sup, "k");
  strcat (lin_inf, "j");
  strcat (lin_med, "u");

  for (va=regi+1; va<regf; va++)
    {
    ubica (va,coli);
      printf("%s","x");
    ubica (va,colf);
      printf("%s","x");
    }

  ubica (regi,coli);
    printf("%s",lin_sup);
  ubica (regm,coli);
    printf("%s",lin_med);
  ubica (regf,coli);
   printf("%s",lin_inf);

  printf("\033(B");       /* Salir del Modo Grafico */
  fflush ( stdout );
}

ubica(x,y)
int x,y;
{
 char xx[3],yy[3];
 sprintf(xx,"%d\0",x);
 sprintf(yy,"%d\0",y);
 printf("\033[%s;%sH",xx,yy);
}

fecha()
{
  dia=mes=ano=0;
  nseg=time(NULL);
  ptime=localtime(&nseg);

  ano = ptime->tm_year;
  mes += (ptime->tm_mon+1);
  dia += (ptime->tm_mday);
}
```

Sistema	Petroleos Mexicanos	Fecha
S.e.r.f.	Subgerencia de Operacion de Ductos y Terminales	6/10/94

```
/*********************************************************
 *        Sistema de Estados de Resultados Financieros    *
 *         Libreria Para Genera el Marco Fecha actual     *
 *                                                        *
 *                                                        *
 * Autor:  Arturo Leon Juarez          Programa: cuadro   *
 * Fecha: 01/01/94                   Equipo : hp9000/842  *
 *********************************************************/
#include <stdio.h>
#include <time.h>

struct tm *ptime, *localtime();
time_t time(), nseg;

int diam;
int mesm;
int anom;

titulos()
 {
  int j;
  limpia();
  box(1,1,24,79,4);
  gotoxy(2,30);printf("Petroleos Mexicanos");
  gotoxy(3,15);printf("Subgerencia de Operacion de Ductos y Terminales");
  gotoxy(2,4);printf("Sistema");
  gotoxy(3,4);printf("\033[1mS.e.r.f\033[0m");
  fecha();
  gotoxy(2,70);printf(" Fecha ");
  gotoxy(3,68);printf("\033[1m %2d/%02d/%02d \033[0m",diam,mesm,anom);
  gotoxy(6,16);printf("Sistema de Estados de Resultados Financieros");
  gotoxy(8,28);printf("Respaldo de Saldos");
}
gotoxy(x,y)
int x,y;
{
  char xx[3],yy[3];
  sprintf(xx,"%d\0",x);
  sprintf(yy,"%d\0",y);
  printf("\033[%s;%sH",xx,yy);
}
box(x1,y1,x2,y2,x3)
int x1,y1,x2,y2,x3;
{
 register int j;
 printf("\033(0");
 gotoxy(x1,y1);
    printf("l\033[D\033[B");
 for (j=1;j<x2-x1;j++)
    printf("x\033[B\033[D");
    printf("m");
```

```
 for (j=1;j<y2-y1;j++)
    printf("q");
    printf("j");
 for (j=1;j<x2-x1;j++)
    printf("\033[A\033[Dx");
    printf("\033[A\033[Dk");
 for (j=1;j<y2-y1;j++)
    printf("\033[2Dq");

  gotoxy(x3,y1);
    printf("t");
 for (j=1;j<y2-y1;j++)
    printf("q");
  printf("u");
  printf("\033(B");
}
limpia()
{
   printf("\033[2J");
}
fecha()
{
  diam=mesm=anom=0;
  nseg=time(NULL);
  ptime=localtime(&nseg);

  anom = ptime->tm_year;
  mesm += (ptime->tm_mon+1);
  diam += (ptime->tm_mday);
}
```

DESARROLLO DE PANTALLAS

5.1 Introducción

El siguiente capítulo tiene como finalidad mostrar las pantallas desarrolladas en Sistema de Estados de Resultados Financieros (SERF), las cuales fueró desarrolladas en **SQL*forms V. 3.0** como sea visto en el capítulo dos, esta es una utilería de **ORACLE**. El desarrollo de las pantallas es por medio de pantallas pop-up. La interfase de **SQL*froms** es una utilería que sirve de interfase entre el sistema operativo **UNIX** y **ORACLE**. La cual nos sive para que el programador desarrolle por medio de menús y ayuda en línea , y poder explotar la información, de una manera más fácil apoyado en estas herramientas.

La utilización de la herramienta **SQL*froms** genera automáticamente código por lo que el código generado es mucho y muy poco comprensible para el programador, en resumen la interfase de **SQL*froms** genera autocódigo por lo que solo se mostrarán las pantallas que se desarrollaron y no el código que generan estas pantallas, pero si se dará una explicación de lo que realiza cada una de estas pantallas, así como la ubicación de los programas fuente y la cantidad de pantallas.

En este capítulo incluiremos las pantallas que nos sirven de monitoreo para los procesos de actualización de saldos, las cuales nos muestran en que momento terminan estos ya que se pueden ejecutar todos los programas en un instante y revisar las pantallas de monitoreo para saber el momento en que terminan estos procesos.

Las pantallas de monitoreo son algunos **shells** que nos sirven para revisar archivos en donde se va monitoreando el avance que tiene el proceso, en que momento empieza y el momento en que este termina. También se verá la ubicación de los programas ejecutables, cuantos son y a diferencia de las formas aquí si se incluirá el código.

Empezaremos por la ubicación de las pantallas, cuántas y cuáles son estas, también se verá la ubicación de sus programas ejecutables.

Trayectoria de las pantallas de captura
/dsr/dsrloc/serf/cap
El nombre de las pantallas de captura y su tamaño es:

NOMBRE	TAMAÑO EN BYTS	
PASIVO.inp	41434	Captura de pasivo.
PRESP.inp	38632	Captura del presupuesto.

PRE_ADPT.inp	35910	Captura del presupuesto anual.
SOLCT.inp	41495	Captura de la solicitud de fondos.

Trayectoria de las pantallas de consulta
/dsr/dsrloc/serf/pant
El nombre de las pantallas de consulta y su tamaño es:

NOMBRE	TAMAÑO EN BYTS	
CAPSUBDI.in	24434	Consulta por subdirección.
CENTRO.inp	17019	Consulta por centro de trabajo.
MOMCMAL.inp	14225	Consulta por momento mal codificado.
MOMENTO.inp	15118	Consulta por momento contable.
RENGLON.inp	36344	Consulta por renglón del gasto.
SALDOA.inp	12534	Consulta departamento economico.

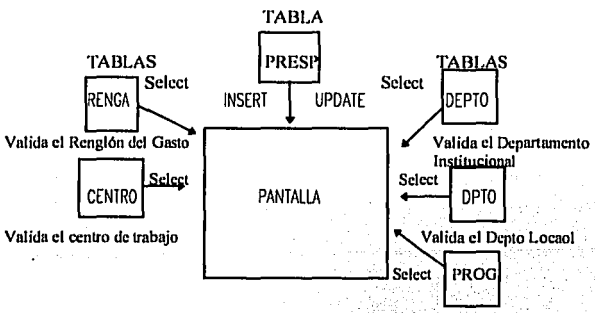
Trayectoria de los shells de monitoreo
/dsr/dsrloc/serf/exe
El nombre de los shells de monitoreo y su tamaño es:

NOMBRE	TAMAÑO EN BYTS	
actsal2i.exe	1129	Actualización de pasivo por Depto. Inst.
actsal2l.exe	1109	Actualización de pasivo por Depto. Local.
actsal3i.exe	1150	Actualización de pago inmediato Depto. Inst.
actsal3l.exe	1137	Actualización de pago inmediato Depto Local
actsall.exe	1282	Actualización global por Depto. Local
actsali.exe	1100	Actualización global por Depto. Inst.

Nota. Estos programas ejecutan programas en **Pro*C** los cuales mandan mensajes hacia algunos archivos de salida en donde se puede checar la hora en que inicia su ejecución y la hora de finalización del programa, para que sean monitoreados por el usuario. Como se observa, es un conjunto de instrucciones que interactúan en conjunto de una manera secuencial. Existen algunas otros **shells,** los cuales se verán conjuntamente con los reportes y algunos otros programas desarrollados en **Pro*C**

Este es un esquema de como interactuan las pantallas con la Base de Datos de una manera general.

Esquema de la interrelación de las pantallas con la BD.



PRE_ADPT.inp	35910	Captura del presupuesto anual.
SOLCT.inp	41495	Captura de la solicitud de fondos.

Trayectoria de las pantallas de consulta
/dsr/dsrloc/serf/pant
El nombre de las pantallas de consulta y su tamaño es:

NOMBRE	TAMAÑO EN BYTS	
CAPSUBDI.in	24434	Consulta por subdirección.
CENTRO.inp	17019	Consulta por centro de trabajo.
MOMCMAL.inp	14225	Consulta por momento mal codificado.
MOMENTO.inp	15118	Consulta por momento contable.
RENGLON.inp	36344	Consulta por renglón del gasto.
SALDOA.inp	12534	Consulta departamento economico.

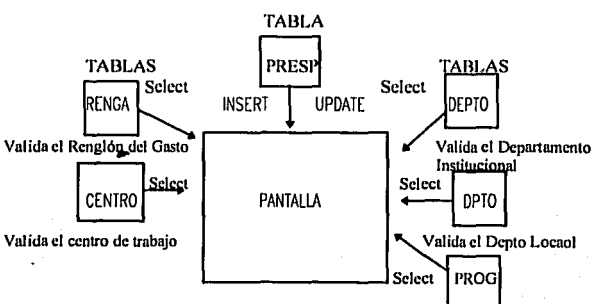
Trayectoria de los shells de monitoreo
/dsr/dsrloc/serf/exe
El nombre de los shells de monitoreo y su tamaño es:

NOMBRE	TAMAÑO EN BYTS	
actsal2i.exe	1129	Actualización de pasivo por Depto. Inst.
actsal2l.exe	1109	Actualización de pasivo por Depto. Local.
actsal3i.exe	1150	Actualización de pago inmediato Depto. Inst.
actsal3l.exe	1137	Actualización de pago inmediato Depto Local
actsall.exe	1282	Actualización global por Depto. Local
actsali.exe	1100	Actualización global por Depto. Inst.

Nota. Estos programas ejecutan programas en **Pro*C** los cuales mandan mensajes hacia algunos archivos de salida en donde se puede checar la hora en que inicia su ejecución y la hora de finalización del programa, para que sean monitoreados por el usuario. Como se observa, es un conjunto de instrucciones que interactúan en conjunto de una manera secuencial. Existen algunas otros **shells,** los cuales se verán conjuntamente con los reportes y algunos otros programas desarrollados en **Pro*C**

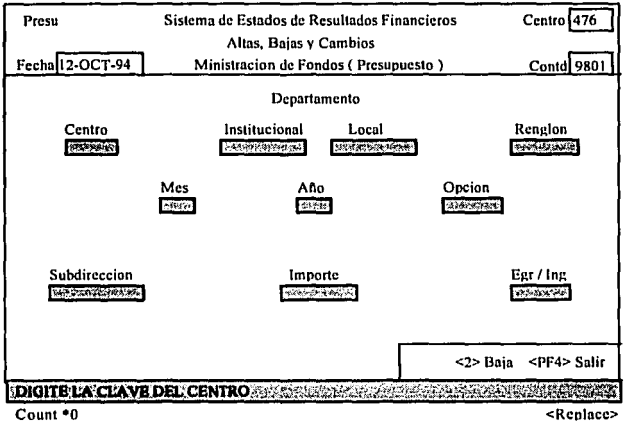
Este es un esquema de como interactuan las pantallas con la Base de Datos de una manera general.

Esquema de la interrelación de las pantallas con la BD.

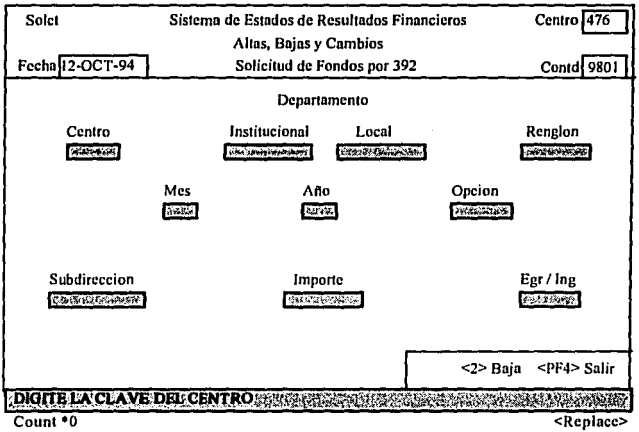


5.2 Pantallas de Captura

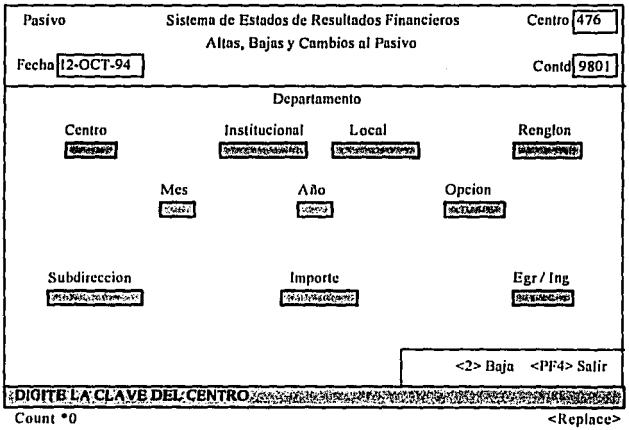
La siguiente pantalla es donde se realiza la captura, borrado y actualización de la tabla (**PRESP**) del presupesto autorizado a el centro de trabajo por mes, los campos capturados son: centro de trabajo, departamento institucional, departamento local, mes y año de la asignación del presupuesto, renglón del gasto, si se desea dar de alta el registro, la subdirección a la que pertenece el departamento , importe y si es un egreso o un ingreso. La pantalla valida el centro de trabajo, el departamento y la subdirección en los catálogos institucionales y el departamento local en catálogos propios del centro de trabajo.



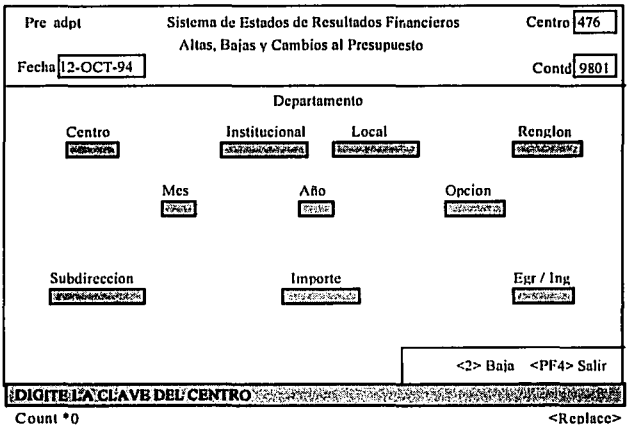
La siguiente pantalla realiza las mismas funciones que la anterior a excepción de que esta lo realiza sobre la tabla de solicitud de fondos (**SOLCT**), la validación es sobre los mismos catálogos.



La siguiente pantalla realiza las mismas funciones que la primera descrita, a excepción de que esta lo realiza sobre la tabla de la solicitud de pasivo programado por mes **(PASIVO)**, la validación es sobre los mismos catalogos.



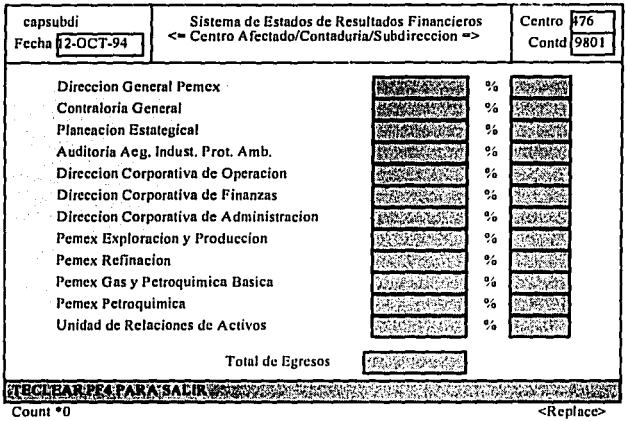
Finalmente, dentro de las pantallas de captura esta la forma que nos sirve para capturar, borrar y actualizar el presupuesto asignado anualmente. Esta contiene los mismos campos que las anteriores sólo sin el campo del mes. Y lo anterior lo realiza sobre la tabla del presupuesto, anula **(PRE_ADPT)**, la validación es sobre los mismos catálogos.



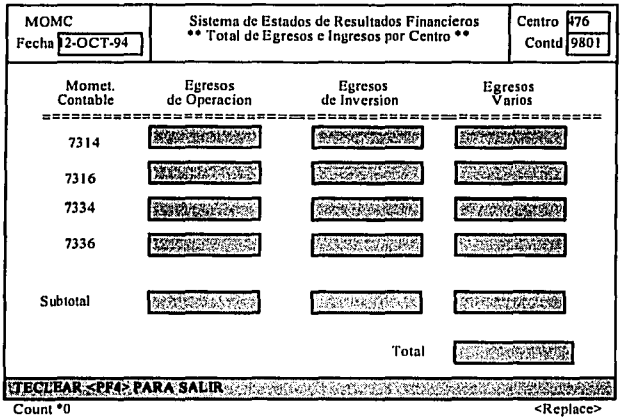
5.3 Pantallas de Consultas

La siguiente pantalla es donde se realiza la consulta,de la información cargada del Sistema Institucional de Contabilidad (**SIC**) hacia el sistema (**SERF**) ejecutando los proceso de inserción y actualización de la información en la tabla **REGAS , RP_00** y **GRUPO**, la información que contienen estas tablas es presentada en las pantallas que se muestran junto con la validacion que se realiza con algunos catálogos institucionales.

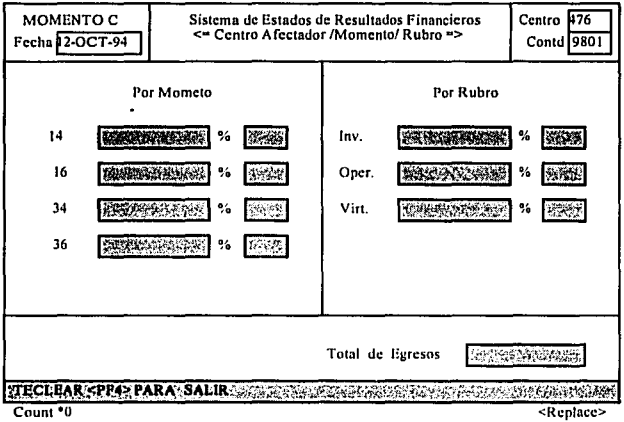
La siguiente pantalla muestra el importe de las erogaciones por subdirección y el porcentaje que cada una de las subdirecciones con respecto al gasto global del centro de trabajo, esta pantalla extrae y valida la información de las tablas mostradas en el esquema.



La siguente pantalla muestra los egresos e ingresos que se generan en el centro de trabajo en los diferentes momentos contables y nos entrega el total por centro, validando y tomando información de las tablas mostradas en el esquema:

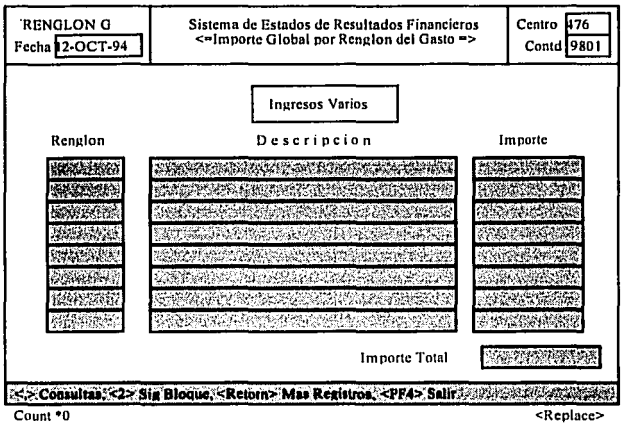


La siguiente pantalla muestra el gasto realizado en el centro de trabajo por mes en cada uno de los momentos contable, así como el procentaje que cada uno comprende del 100%, también el porcentaje y gasto de las diferentes erogaciones que comprende la suma total del gasto. La información es seleccionada de las tablas mostradas en el esquema.

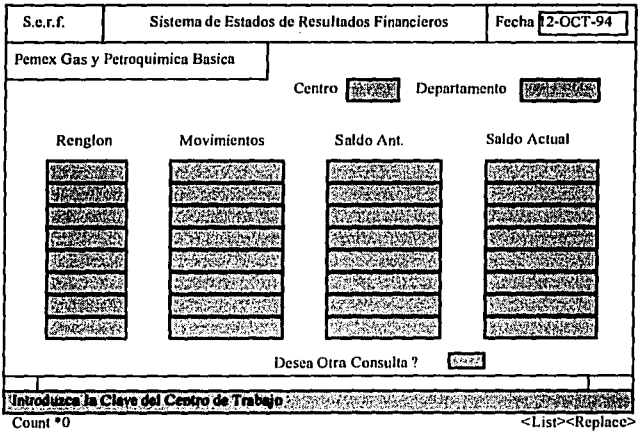


DESARROLLO DE PANTALLAS

La siguiente pantalla muestra el importe erogado en cada uno de los renglones del gasto y la descripción de este. Esta pantalla esta compuesta por cuatro páginas, las cuales son similares, sólo cambia el texto que dice Ingresos Varios por (Egresos de Operación, Egresos de Inversión o Egresos Varios), y la selección de información de cada una de ellas. Además selecciona por pantalla los renglones correspondientes a cada una de estas y nos muestra su total en pantalla. La información es seleccionada de las tablas mostradas en el esquema.



Esta pantalla nos pide que introduzcamos el centro de trabajo, posteriormente el departamento local y con la coma del teclado numerico la selección del renglón del gasto, así como sus movimientos del mes su saldo acumulado al mes anterior y su saldo del mes actual al final nos pregunta si se desea seguir consultando o salir. La información es extraida de las tablas **REGAS** y **TEMP.**


S.e.r.f.
Sistema de Estados de Resultados Financieros
Fecha 12-OCT-94
Pemex Gas y Petroquimica Basica
Centro
Departamento
Renglon
Movimientos
Saldo Ant.
Saldo Actual
Desea Otra Consulta ?
Introduzca la Clave del Centro de Trabajo
Count *0
<List><Replace>

5.4 Pantallas de Monitoreo

Todas las pantallas de monitoreo son **shells** que mandan información hacia un archivo, este a su vez es monitoreado para ver su avance y saber en que momento termina de actuálizar los saldos. Las pantallas y lo que muestran es similar, sólo cambian los mensajes para saber que tabla se esta actualizando, por lo que sólo se mostrará una pantalla. Lo que si se verá es el código de los **shells**, que ejecutan los programas en **Pro*C** y mandan el texto a un archivo para que se pueda seguir el avance del proceso y saber en que momento la información ha sido actualizada. Los programas al detectar un error mandan un mensaje para que se verifique y corrija este antes de seguir procesando.

Inicia Actualizacion de Saldos Locales

en Forma Global Suma (392 + 219)

FECHA 28/10/40 HORA 17:45

Termina Actualizacion de Saldos

FECHA 28/10/94 HORA 17:48

(EOF) Teclear <Retorn>

```
#####################################################
#  Shell que Lanza el programa de actualizacion de Saldos   #
#  de la Solicitud del Pasivo por departamento Institucional  #
#  Nombre del Programa : actsal2i.exe                  #
#####################################################
error='sum log.err|cut -c1-2'
if [ $error -eq 0 ]
  then
echo "    Inicia Actualizacion de Saldos Institucionales" >bit/actsal2i.bit
echo "       Solicitud del Pasivo Forma ( 219 ) "      >>bit/actsal2i.bit
fecha                                    >>bit/actsal2i.bit
echo                                     >>bit/actsal2i.bit
bin/actsal2i $usr $pwd
echo "    \033[7m  Termina Actualizacion de Saldos\033[0m">>bit/actsal2i.bit
fecha                                    >>bit/actsal2i.bit
else
echo "\033[10;15H Existe un Error en el Proceso Anterior "  >bit/actsal2i.bit
echo "\033[12;15H Favor de Avisar al Area de Informatica "  >>bit/actsal2i.bit
echo "\033[14;15H Antes de Continuar < Retorn > \c"       >>bit/actsal2i.bit
read nada
fi
```

```
################################################
# Shell que Lanza el programa de actualizacion de Saldos #
# de la Solicitud del Pasivo por departamento Local      #
# Nombre del Programa : actsal2l.exe                     #
################################################
error='sum log.err|cut -c1-2'
if [ $error -eq 0 ]
  then
echo "      Inicia Actualizacion de Saldos Locales"     >bit/actsal2l.bit
echo "       Solicitud del Pasivo Forma ( 219 ) "       >>bit/actsal2l.bit
fecha                                    >>bit/actsal2l.bit
echo                                     >>bit/actsal2l.bit
bin/actsal2l $usr $pwd
echo "     \033[7m Termina Actualizacion de Saldos\033[0m">>bit/actsal2l.bit
fecha                                    >>bit/actsal2l.bit
else
echo "\033[10;15H Existe un Error en el Proceso Anterior "  >bit/actsal2l.bit
echo "\033[12;15H Favor de Avisar al Area de Informatica "  >>bit/actsal2l.bit
echo "\033[14;15H Antes de Continuar < Retorn > \c"         >>bit/actsal2l.bit
read nada
fi



#######################################################
# Shell que Lanza el programa de actualizacion de Saldos de la  #
# Solicitud del Flujo de Efectivo por Depto. Institucional      #
# Nombre del Programa : actsal3i.exe                            #
#######################################################
error='sum log.err|cut -c1-2'
if [ $error -eq 0 ]
  then
echo "      Inicia Actualizacion de Saldos Institucionales" >bit/actsal3i.bit
echo "      Solicitud del Flujo de Efectivo Forma  ( 392 )" >>bit/actsal3i.bit
fecha                                    >>bit/actsal3i.bit
echo                                     >>bit/actsal3i.bit
bin/actsal3i $usr $pwd
echo "     \033[7m  Termina Actualizacion de Saldos\033[0m">>bit/actsal3i.bit
fecha                                    >>bit/actsal3i.bit
else
echo "\033[10;15H Existe un Error en el Proceso Anterior "  >bit/actsal3i.bit
echo "\033[12;15H Favor de Avisar al Area de Informatica "  >>bit/actsal3i.bit
echo "\033[14;15H Antes de Continuar < Retorn > \c"         >>bit/actsal3i.bit
read nada
fi
```

```
#####################################################
#  Shell que Lanza el programa de actualizacion de Saldos de #
#  la Solicitud del Flujo de Efectivo por Depto. Locales      #
#  Nombre del Programa : actsal31.exe                         #
#####################################################
error='sum log.err|cut -c1-2'
if [ $error -eq 0 ]
  then
echo "     Inicia Actualizacion de Saldos Locales"       >bit/actsal31.bit
echo "     Solicitud del Flujo de Efectivo Forma  ( 392 )" >>bit/actsal31.bit
fecha                                        >>bit/actsal31.bit
echo                                         >>bit/actsal31.bit
bin/actsal31 $usr $pwd
echo "     \033[7m Termina Actualizacion de Saldos\033[0m"  >>bit/actsal31.bit
fecha                                        >>bit/actsal31.bit
else
echo "\033[10;15H Existe un Error en el Proceso Anterior "   >bit/actsal31.bit
echo "\033[12;15H Favor de Avisar al Area de Informatica "   >>bit/actsal31.bit
echo "\033[14;15H Antes de Continuar < Retorn > \c"          >>bit/actsal31.bit
read nada
fi



#####################################################
#  Shell que Lanza el programa de actualizacion de Saldos   #
#  Por departamento Institucional en Forma Global           #
#  Nombre del Programa : actsali.exe                        #
#####################################################
error='sum log.err|cut -c1-2'
if [ $error -eq 0 ]
  then
echo "     Inicia Actualizacion de Saldos Institucionales">bit/actsali.bit
echo "          en Forma Global suma (392 + 219) "    >>bit/actsali.bit
fecha                                        >>bit/actsali.bit
echo                                         >>bit/actsali.bit
echo "               Inicia Respaldo              " >>bit/actsali.bit
fecha                                        >>bit/actsali.bit
bin/actsali $usr $pwd
echo "     \033[7m  Termina Actualizacion de Saldos\033[0m">>bit/actsali.bit
fecha                                        >>bit/actsali.bit
else
echo "\033[10;15H Existe un Error en el Proceso Anterior "  >bit/actsali.bit
echo "\033[12;15H Favor de Avisar al Area de Informatica "  >>bit/actsali.bit
echo "\033[14;15H Antes de Continuar < Retorn > \c"         >>bit/actsali.bit
read nada
fi
```

```
################################################
# Shell que Lanza el programa de actualizacion de Saldos #
# Por departamentos locales en Forma Global              #
# Nombre del Programa : actsall.exe                      #
################################################
error=`sum log.err|cut -c1-2`
if [ $error -eq 0 ]
  then
echo "     Inicia Actualizacion de Saldos Locales"     >bit/actsall.bit
echo "       en Forma Global suma (392 + 219) "        >>bit/actsall.bit
fecha                                    >>bit/actsall.bit
echo                                     >>bit/actsall.bit
bin/actsall $usr $pwd
echo "    \033[7m Termina Actualizacion de Saldos\033[0m">>bit/actsall.bit
fecha                                    >>bit/actsall.bit
else
echo "\033[10;15H Existe un Error en el Proceso Anterior "  >bit/actsall.bit
echo "\033[12;15H Favor de Avisar al Area de Informatica "  >>bit/actsall.bit
echo "\033[14;15H Antes de Continuar < Retorn > \c"         >>bit/actsall.bit
read nada
fi
```

Para complementar y conocer mejor la utilidad de los **shells,** en el capítulo siguiente mostraremos los programas en **Pro*C** y la operación de cada uno de estos conjuntamente con los reportes, los cuales son también **Pro*C.**

DESARROLLO DE REPORTES

6.1 Introducción

Como se podrá observar durante el capítulo, el código de los programas desarrollados en **Pro*C** para nuestra aplicación es muy extenso para cada programa, por lo que solo se escribirán completos aquellos que sean totalmente diferentes tanto en su estructura como en la presentación del programa y se mencionarán y explicaran diferencias de los demás programas.

Se dará una breve descripción de lo que realiza el programa y la salida o reporte que entrega para el caso de que el programa se escriba todo. Para el caso en que el programa sea muy similar sólo se explicarán las diferencias que existen entre estos y se describirán cuales son cada una de estas diferencias, conjuntamente se dará una breve explicación de qué realiza, omitiendo la salida que el programa proporciona.

Se incluirán los programas que actualizan saldos, así como todos aquellos programas que estén desarrollados en **Pro*C** y librerías de lenguaje "C" que estos utilizan. También como en los capítulos anteriores se verá la ubicación de los programas fuente cuántos son y el tamaño de cada uno de ellos.

6.2 Programas y Salida de los Reportes.

Empezaremos describiendo y mostrando la ubicación de los reportes, cuantas y cuales son estos.

Trayectoria de los Reportes

/dsr/dsrloc/serf/repo

El nombre de los reportes:

NOMBRE	TAMAÑO EN BYTS
rep219i.pc	14282
rep219l.pc	14294
rep392i.pc	14307
rep392l.pc	14296
repsub2.pc	14373

repsub3.pc	14385
repsumi.pc	14434
repsuml.pc	14445
mes392.pc	10606
solc392.pc	10386

Existen dos reportes "mes392.doc" y "solc392.pc" estos programas entregan un formato especial que en **PEMEX** se conoce como la solicitud de fondos o forma 392 y es en esta en donde se encuentra lo que cada Departamento solicita de presupuesto a nivel de renglón del gasto, y lo agrupa dependiendo de su naturaleza, por Inversión, Operación o Crédito, según sea el gasto realizado.

Este programa "rep219i.pc" entrega de salida un reporte llamado "rep219i.lis", que proporciona información agrupada por Centro de Trabajo, Departamento institucional y renglón del gasto, junto con su descripción de cada uno de estos. También muestra el saldo anterior, la ministración ó pasivo que mensualmente captura el Departamento de presupuestos, la diferencia entre el saldo anterior y la ministración del pasivo, los gastos que realizan durante el mes por cada Departamento y el remanente entre el pasivo ministrado y los gastos del mes. Selecciona información de las tablas **t_041, Regas, Sal_219i, Pasivo, Renga, Depto** y **Grupo** para generar de salida el reporte que posteriormente se verá.

Los reportes "rep219i.pc", "rep392i.pc" y "repsumi.pc", nos muestran los gastos por Departamento Institucional, los reportes "rep219l.pc", "rep392l.pc" y "repsuml.pc" agrupan la información por Departamento local. Los reportes "repsub2.pc" y "repsub3.pc" entregan información agrupada por subdirección, el primero por pago de pasivo y el segundo por pago inmediato.

Como se puede observar y de lo anterior los programas también se pueden agrupar por forma de pago, de la siguiente manera los reportes "rep219i.pc", "rep219l.pc" y "repsub2.pc" se integran por pago a pasivo, mientras que los reportes "rep392i.pc", "rep392l.pc" y "repsum3.pc" por pago inmediato, se puede observar que todos los programas son muy similares y por lo que solo se destacara la diferencia que existe en relación al programa que ejemplificaremos.

```
/*************************************************************
 *        Sistema de Estados de Resultados Financieros       *
 *         Reporte que Muestra la Diferencia entre el        *
 *           Ejercicio del Pasivo y su Ministrado            *
 *                                                           *
 * Tablas Accesadas: t_041 [S] Regas [S] Sal_219i [S] Pasivo [S]  *
 *             Renga [S] Depto [S] Grupo [S]                 *
 *                                                           *
 * Autor:  Arturo Leon Juarez              Programa: rep219i *
 * Fecha: 01/01/94                       Equipo  : hp9000/842 *
 *************************************************************/

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define BEGIN {
#define END   }
FILE *salida;

  /*****************************
   *    Declaracion de variables   *
   *****************************/

EXEC SQL BEGIN DECLARE SECTION;

VARCHAR usr[7];
VARCHAR pwd[7];
VARCHAR cve_mes[2];
VARCHAR nom_mes[12];
VARCHAR fec_elab[8];
VARCHAR fecha[8];
VARCHAR inger[2];
VARCHAR inger_ant[2];
VARCHAR t_des[50];
VARCHAR d_des[50];
VARCHAR cve_cont[4];
VARCHAR wimp_sald[16];
VARCHAR wimp_pas[16];
VARCHAR wimp_cont[16];
VARCHAR wsum_imp[16];
VARCHAR wdif_imp[16];
VARCHAR wtot_sald[16];
VARCHAR wtot_pas[16];
VARCHAR wtot_cont[16];
VARCHAR wtot_sum[16];
VARCHAR wtot_dif[16];
VARCHAR wisal_dep[16];
VARCHAR wipas_dep[16];
VARCHAR wicont_dep[16];
VARCHAR wisum_dep[16];
VARCHAR widif_dep[16];
```

```
int dia;
int ano;
int ctro;
int dpto;
int reng;
int mesg;
int ctro_ant;
int dpto_ant;
int reng_ant;
int num2;
int cont;
int mesal;
int mes;
int flag;

double imp_cont,imp_pas,imp_sald;
double sum_imp,dif_imp;
double tot_sald,tot_pas,tot_cont;
double tot_sum,tot_dif;
double isal_dep,ipas_dep,icont_dep;
double isum_dep,idif_dep;

  /*****************************
   * Inicializacion de variables *
   *****************************/

int conlin=58;
int hoja=0;
int sw=1;

EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE sqlca.h;

main(argc,argv)
   int argc;
   char *argv[];
   BEGIN
    strcpy (usr.arr,argv[1]);
    strcpy (pwd.arr,argv[2]);
    usr.len = strlen(usr.arr);
    pwd.len = strlen(pwd.arr);

EXEC SQL WHENEVER SQLERROR GOTO error_oracle;
EXEC SQL CONNECT :usr IDENTIFIED BY :pwd;
EXEC SQL WHENEVER NOT FOUND CONTINUE;
EXEC SQL SELECT n_cont INTO :cve_cont FROM t_041;
EXEC SQL SELECT max(to_char(fec_elab,'DD')),max(to_char(fec_elab,'MM')),
          decode(max(to_char(fec_elab,'MM')),01,'Enero',02,'Febrero',03,
          'Marzo',04,'Abril',05,'Mayo',06,'Junio',07,'Julio',08,'Agosto',
          09,'Septiembre',10,'Octubre',11,'Noviembre',12,'Diciembre'),
```

```
        max(to_char(fec_clab,'YYYY')),to_char(sysdate,'DD-MM-YY')
      INTO :dia, :mes, :nom_mes, :ano, :fecha FROM regas;
EXEC SQL SELECT max(cve_mess) INTO :mesal FROM sal_219i;

   if ((salida = fopen("tmp/rep219i.lis","w")) == NULL)
     BEGIN
       printf ("No Puedo Abrir Archivo de Salida rep219i.lis\n");
       goto error_oracle;
     END

EXEC SQL SELECT count(*) INTO :cont  FROM sal_219i;

num2 = mes-1;
if (num2 == mesal)
 BEGIN
 flag=0;
 END
else
  if ((num2 == 1) && (cont == 0))
    BEGIN
     flag=1;
    END
  else
     if ((num2 == 1) && (cont > 0))
       BEGIN
        flag=2;
       END
 if (flag == 2)
 BEGIN
  fprintf(salida,"\nError los Saldos no Corresponden al Mes");
  fprintf(salida,"\n Anterior que se va ha Procesar\n");
  goto lee_error;
 END
if (flag == 1)
 BEGIN
  fprintf(salida,"\nError los Saldos no Estan en Cero y el ");
  fprintf(salida,"\n Mes a Procesar es el Primero  \n");
  goto lee_error;
 END

  /*****************************
   *  Declaracion de cursores  *
   *****************************/

EXEC SQL DECLARE lee_regas CURSOR FOR
      SELECT cve_ctro,cve_dpto,cve_reng,ing_egr,sum(imp_cont),0,0
        FROM grupo
        WHERE cve_momc in (14,34)
     GROUP BY cve_ctro,cve_dpto,cve_reng,ing_egr
     UNION
      SELECT cve_ctro,cve_dpto,cve_reng,ing_egr,0,sum(imp_pre),0
```

```
      FROM pasivo
      WHERE cve_mess = :mes
    GROUP BY cve_ctro,cve_dpto,cve_reng,ing_egr
    UNION
     SELECT cve_ctro,cve_dpto,cve_reng,ing_egr,0,0,sum(imp_sal)
      FROM sal_219i
    GROUP BY cve_ctro,cve_dpto,cve_reng,ing_egr
    ORDER BY 1,2,3,4;

  /*****************************
   *  Inicializacion  de Proceso  *
   *****************************/

 EXEC SQL DELETE FROM TAB_TEMP;
 EXEC SQL COMMIT WORK;

EXEC SQL OPEN lee_regas;
EXEC SQL WHENEVER NOT FOUND GO TO regas_fin;
for( ; ; )
BEGIN

 EXEC SQL FETCH lee_regas INTO :ctro,:dpto,:reng,:inger,:imp_cont,:imp_pas,
                  :imp_sald;

 EXEC SQL INSERT INTO TAB_TEMP (cve_ctro,cve_dpto,cve_prog,cve_reng,ing_egr,
                    imp_cont,imp_pas,imp_sald)
                VALUES (:ctro, :dpto, null, :reng, :inger, :imp_cont,
                    :imp_pas, :imp_sald);
END

regas_fin:
 EXEC SQL CLOSE lee_regas;

EXEC SQL DECLARE lee_temp CURSOR FOR
     SELECT cve_ctro,cve_dpto,cve_reng,ing_egr,sum(imp_cont),
         sum(imp_pas),sum(imp_sald)
      FROM tab_temp
    GROUP BY cve_ctro,cve_dpto,ing_egr,cve_reng
    ORDER BY cve_ctro,cve_dpto,ing_egr,cve_reng;

EXEC SQL OPEN lee_temp;
EXEC SQL WHENEVER NOT FOUND GO TO temp_fin;
for( ; ; )
BEGIN

 imp_cont = 0;
 imp_pas  = 0;
 imp_sald = 0;

 EXEC SQL FETCH lee_temp INTO :ctro,:dpto,:reng,:inger,:imp_cont,:imp_pas,:imp_sald;
```

```
 inger.len = strlen(inger.arr);
 inger_ant.len = strlen(inger_ant.arr);

if (sw==1)
  BEGIN
   ctro_ant = ctro;
   dpto_ant = dpto;
   reng_ant = reng;
   strcpy(inger_ant.arr,inger.arr);
   sw=0;
   END

 if ((ctro_ant == ctro) && (dpto_ant == dpto))
   BEGIN
   if ((reng_ant >= 100) && (reng_ant < 200))
     BEGIN
      if ((reng < 100) || (reng > 199))
        BEGIN
         if (strcmp(inger_ant.arr,"I")==0)
           BEGIN
            mesg=1;
            titulo_r();
           END
        END
     END
    if ((reng_ant >= 200) && (reng_ant < 300))
      BEGIN
       if ((reng < 200) || (reng > 299))
         BEGIN
          mesg=3;
          titulo_r();
         END
      END
    if ((reng_ant >= 300) && (reng_ant < 400))
      BEGIN
       if ((reng < 300) || (reng > 399))
         BEGIN
          mesg=4;
          titulo_r();
         END
      END
    if ((reng_ant >= 100) && (reng_ant < 200))
      BEGIN
       if ((reng < 100) || (reng > 199))
         BEGIN
          if (strcmp(inger_ant.arr,"E")==0)
           BEGIN
             mesg=2;
             titulo_r();
           END
         END
```

```
      END
    saca_reng();
    sum_imp=0; dif_imp=0;
    sum_imp = imp_pas + imp_sald;
    dif_imp = imp_pas - imp_cont;
    reg_ant();
   END
  else
    BEGIN
    corte();
    saca_reng();
    titulos();
    reg_ant();
   END
END
temp_fin:

    /*************************************
     *  Fin del proceso exitoso o con error  *
     *************************************/

EXEC SQL CLOSE lee_temp;
corte();
fclose(salida);
EXEC SQL COMMIT WORK RELEASE;
return(0);
error_oracle:
     EXEC SQL whenever sqlerror continue;
     printf("═══════════════════════════════════════════════════════\n");
     printf("# Error :%s \n",sqlca.sqlerrm.sqlerrmc);
     printf("═══════════════════════════════════════════════════════\n");
     EXEC SQL rollback work release;
     fclose(salida);
     exit(0);
     lee_error:
     fclose(salida);
     exit(0);
END

reg_ant()
BEGIN
  detalle();
  totales();
  ctro_ant = ctro;
  dpto_ant = dpto;
  reng_ant = reng;
  strcpy(inger_ant.arr,inger.arr);
END

totales()
BEGIN
```

```
 tot_sald = tot_sald + imp_sald;
 tot_cont = tot_cont + imp_cont;
 tot_pas = tot_pas + imp_pas;
 tot_sum = tot_sum + sum_imp;
 tot_dif = tot_dif + dif_imp;
 isal_dep = isal_dep + imp_sald;
 icont_dep = icont_dep + imp_cont;
 ipas_dep  = ipas_dep + imp_pas;
 isum_dep = isum_dep + sum_imp;
 idif_dep = idif_dep + dif_imp;
END

saca_reng()
BEGIN
 strcpy(d_des.arr,"Sin Descripcion");
 EXEC SQL WHENEVER NOT FOUND CONTINUE;
 EXEC SQL SELECT des_reng INTO :t_des FROM renga
          WHERE cve_reng = :reng;
 EXEC SQL SELECT des_dpto INTO :d_des FROM depto
        WHERE cve_dpto = :dpto
          AND cve_ctro = :ctro;
 t_des.arr[t_des.len]='\0';
 d_des.arr[d_des.len]='\0';
END

   /*****************************************
    *  Inicio de funciones y cortes por regl y dpto  *
    *    para la generacion del reporte de salida    *
    *****************************************/

titulos()
BEGIN
 hoja++;
 fprintf(salida,"\f\n\n");
 fprintf(salida,"%-60s","Pasivo Dep Inst :");
 pemex();
 fprintf(salida,"%46s %3d \n","Hoja :",hoja);
 fprintf(salida,"%93s","Comparativo de Fondos Disponibles Contra Ejercicio");
 fprintf(salida,"\n%5s","Centro de Trabajo Afectado: ");
 fprintf(salida,"%-4d",ctro);
 fprintf(salida,"%21s%3d%3s%12s%3s%5d","al",dia,"de",nom_mes.arr,"de",ano);
 fprintf(salida,"%45s %-4s \n"," Contaduria: ",cve_cont.arr);
 fprintf(salida,"%-14s %-4d  %-75s","Departamento: ",dpto ,d_des.arr);
 fprintf(salida,"%23s %-9s ","Fecha : ",fecha.arr);
 lineas();
 fprintf(salida,"%21s %41s","Renglon del Gasto","Saldo del Mes");
 fprintf(salida,"%16s %16s %16s","Ministracion","Suma 1+2","Ejercicio");
 fprintf(salida," %14s \n","Variacion");
 fprintf(salida,"%4s %27s","Clave","D e s c r i p c i o n");
 fprintf(salida,"%29s %16s %14s","Anterio (1)","Mensual (2)","(3)");
 fprintf(salida,"%16s %15s \n","(4)","2-4=5");
```

```
 lineas();
 conlin=8;
END

corte_dep()
BEGIN
 convierte();
 lineas();
 fprintf (salida,"%-45s","Total por Departamento :  ");
 fprintf (salida,"%16s ",wisal_dep.arr);
 fprintf (salida,"%16s %16s ",wipas_dep.arr,wisum_dep.arr);
 fprintf (salida,"%16s %16s ",wicont_dep.arr,widif_dep.arr);
END

detalle()
BEGIN
 if (conlin > 57)
    titulos();
imprime();
END

imprime()
BEGIN
 convierte();
 fprintf (salida,"\n%-3d %-41.41s",reng,t_des.arr);
 fprintf (salida,"%16s ",wimp_sald.arr);
 fprintf (salida,"%16s %16s ",wimp_pas.arr,wsum_imp.arr);
 fprintf (salida,"%16s %16s ",wimp_cont.arr,wdif_imp.arr);
 conlin++;
END

corte()
BEGIN

  if ((reng_ant >= 100) && (reng_ant < 200))
    BEGIN
     if (strcmp(inger_ant.arr,"I")==0)
       BEGIN
         mesg=1;
         titulo_r();
       END
    END
   if ((reng_ant >= 200) && (reng_ant < 300))
     BEGIN
      mesg=3;
      titulo_r();
     END
   if ((reng_ant >= 300) && (reng_ant < 400))
     BEGIN
       mesg=4;
       titulo_r();
```

```
      END
    if ((reng_ant >= 100) && (reng_ant < 200))
      BEGIN
       if (strcmp(inger_ant.arr,"E")==0)
         BEGIN
           mesg=2;
           titulo_r();
         END
      END
   corte_dep();
    isal_dep=0;
    ipas_dep=0;
    icont_dep=0;
    isum_dep=0;
    idif_dep=0;
    sum_imp = imp_pas + imp_sald;
    dif_imp = imp_pas - imp_cont;
END

lineas()
BEGIN
 fprintf (salida,"\n=============================================================");
 fprintf (salida,"=============================================================");
 fprintf (salida,"=============================================================\n");
 conlin=conlin+2;
END

corte_reg()
BEGIN
 convierte();
 fprintf (salida,"%16s ",wtot_sald.arr);
 fprintf (salida,"%16s %16s ",wtot_pas.arr,wtot_sum.arr);
 fprintf (salida,"%16s %16s ",wtot_cont.arr,wtot_dif.arr);
 fprintf (salida,"\n");
 conlin++;
END

inicializa()
BEGIN
 tot_sald=0;
 tot_pas=0;
 tot_cont=0;
 tot_sum=0;
 tot_dif=0;
END

convierte()
BEGIN
  EXEC SQL whenever not found continue;
  EXEC SQL select to_char(:imp_sald, '$999,999,999.99'),
           to_char(:imp_pas , '$999,999,999.99'),
```

```
            to_char(:imp_cont, '$999,999,999.99'),
            to_char(:sum_imp , '$999,999,999.99'),
            to_char(:dif_imp , '$999,999,999.99'),
            to_char(:tot_sald, '$999,999,999.99'),
            to_char(:tot_pas , '$999,999,999.99'),
            to_char(:tot_cont, '$999,999,999.99'),
            to_char(:tot_sum , '$999,999,999.99'),
            to_char(:tot_dif , '$999,999,999.99'),
            to_char(:isal_dep , '$999,999,999.99'),
            to_char(:ipas_dep , '$999,999,999.99'),
            to_char(:icont_dep, '$999,999,999.99'),
            to_char(:isum_dep , '$999,999,999.99'),
            to_char(:idif_dep , '$999,999,999.99')
          into :wimp_sald,:wimp_pas,:wimp_cont,:wsum_imp,:wdif_imp,
             :wtot_sald,:wtot_pas,:wtot_cont,:wtot_sum,:wtot_dif,
             :wisal_dep,:wipas_dep,:wicont_dep,:wisum_dep,:widif_dep
          from dual;
END

pemex()
BEGIN
 fprintf(salida,"%s","P^HP^HP^HPe^He^He^Het^Ht^Ht^Htr^Hr^Hr^Hro^Ho^Ho^Ho");
 fprintf(salida,"%s","I^HI^HI^Hle^Hc^Hc^Hco^Ho^Ho^Hos^Hs^Hs^Hs M^HM^HM^HM");
 fprintf(salida,"%s","e^He^He^Hex^Hx^Hx^Hxi^Hi^HI^Hic^Hc^Hc^Hca^Ha^Ha^Ha");
 fprintf(salida,"%s","n^Hn^Hn^Hno^Ho^Ho^Hos^Hs^Hs^Hs");
END

titulo_r()
BEGIN
 lineas();
 mensaje();
 corte_reg();
 inicializa();
END

mensaje()
BEGIN
switch(mesg)
 BEGIN
  case 1: fprintf (salida,"%-45s","Total Ingresos Varios");
          break;
  case 2: fprintf (salida,"%-45s","Total Egresos");
          break;
  case 3: fprintf (salida,"%-45s","Total Egresos de Operacion");
          break;
  case 4: fprintf (salida,"%-45s","Total Egresos de Inversion");
          break;
 END
END
```

Pasivo Dep Inst : **Petroleos Mexicanos** Hoja : 1

Comparativo de Fondos Disponibles Contra Ejercicio

Centro de Trabajo Afectado: 475 al 31 de Enero de 1995 Contaduria: 9801

Departamento: 75710 CONSULTORIO PERIFERICO CD. MENDOZA, VER. Fecha : 01-02-95

englon del Gasto Clave Descripcion	Saldo del Mes Anterio (1)	Ministracion Mensual (2)	Suma 1+2 (3)	Ejercicio (4)	Variacion 2-4=5
118 I.V.A. ACREDITABLE POR PEMEX (EGRESOS)	$.00	$6,239.27	$6,239.27	$4,748.16	$1,491.11
Total Egresos	$.00	$6,239.27	$6,239.27	$4,748.16	$1,491.11
202 ADQUISICION DE MATERIALES, ACCES. Y ARTS.	$.00	$150,055.79	$150,055.79	$144,462.74	5,593.05
204 CONSERVACION Y MANTENIMIENTO DIVERSO POR	$.00	$91,573.69	$91,573.69	$52,599.60	$18,974.09
207 HONORARIOS Y GASTOS PAGADOS A TERCEROS	$.00	$5,836.82	$5,836.82	$. 00	$5,836.82
232 ABSORCION DE IMPUESTOS DEL PERSONAL	$.00	$.00	$.00	$2,355.63	-$2,355.63
235 GASTOS GENERALES PAGADOS A TERCEROS	$.00	$3,831.50	$3,831.50	$2,530.45	$1,301.05
Total Egresos de Operacion	$.00	$231,297.80	$231,297.80	$201,948.42	$29,349.38
310 REHABILITACION, MODIF. Y REACONDICIONAM.	$.00	$6,545.56	$6,545.56	$4,435.50	$2,110.06
Total Egresos de Inversion	$.00	$6,545.56	$6,545.56	$4,435.50	$2,110.06
Total por Departamento :	$.00	$244,082.63	$244,082.63	$211,132.08	32950.55

El programa **rep219l.pc** es similar al anterior en cuanto a su estructura y selección de la información , su principal diferencia es que, muestra la información por Departamento local, tomando información de la tabla **SAL_219L** y del catálogo de Departamentos locales **DPTO**. El programa **rep392l.pc** también toma información del catálogo **DPTO**, además la información que selecciona es del pago inmediato, tomando la información de saldos en tabla **SAL_392L** y el presupuesto cargado en la tabla **MINIS**, en lugar de la tabla **PASIVO** como lo hacen los anteriores programas descritos.

Por último el programa **repsuml.pc** realiza la suma del pago inmediato más el pasivo por Departamento local y su diferencia principal es que toma información de las tablas de **PASIVO**, **PRESUP** y **MINIS** y los saldos los selecciona de la tabla **SALDOL**, entregando saldos globales por mes, en cada uno de los Departamentos locales.

Para el caso de los programas **rep392i.pc** y **repsumi.pc**, el primero solo se diferencia del ejemplo en que selecciona información de pago inmediato y no de pasivo tomando los saldos y ministración de las tablas **MINIS** y **SAL_392L** para la generación del reporte, su estructura es idéntico, el segundo también agrupa la información por Departamento institucional pero este suma el gasto global y el presupuesto que se captura por pasivo y pago inmediato, tomando el saldo global de la tabla **SALDOI**, para entregar un saldo global o variación por mes en cada Departamento Institucional.

Solo falta por describir la funcionalidad y diferencia de los programas **repsub2.pc** y **repsub3.pc**, estos dos programas agrupan información por Subdirección, el primero por pasivo y el segundo por pago inmediato, su estructura y salida son muy similar a los anteriormente vistos, la principal diferencia es que agrupan información por Subdirección, seleccionando el campo de **CVE_PROG** (clave del programa) que contiene la Subdirección en cada una de las tablas que estas accesan. Estos son los programas que tienen similitud con el ejemplo aquí mostrado.

```
/*************************************************************
 *        Sistema de Estados de Resultados Financieros       *
 *                                                           *
 *       Reporte que Genera el Formato para la Solicitud     *
 *          de Fondos de Flujo de Efectivo 392               *
 *                                                           *
 * Tablas Accesadas: Solct [S] Dpto [S] Prog [S] Cetro [S]   *
 *             Renga [S] Puesto [S]                          *
 *                                                           *
 * Autor:  Arturo Leon Juarez              Programa: Solc392 *
 * Fecha: 01/01/94                      Equipo : hp9000/842 *
 *************************************************************/

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define BEGIN {
#define END }

FILE *salida;

/*************************************************************
 *                  DEFINICION DE VARIABLES                  *
 *************************************************************/

EXEC SQL BEGIN DECLARE SECTION;

varchar usr[8];
varchar pwd[8];
varchar capmes[3];
varchar centro[4];
varchar depto[6];
varchar nom_mes[11];
varchar prog[3];
varchar desdpto[50];
varchar desprog[35];
varchar desctr[50];
varchar dreng[50];
varchar wimpopr[20];
varchar wimpinv[20];
varchar wcrdins[20];
varchar wtotal[20];
varchar wsumopr[20];
varchar wsuminv[20];
varchar wsumcrd[20];
varchar wsumtot[20];
varchar puesto1[30];
varchar puesto2[30];
varchar nombre1[30];
varchar nombre2[30];
```

```
int dia_ini;
int dia_fin;
int anio4;
int cetro;
int ecom;
int ofipg;
int rengl;

/************************************
 *      Inicializacion de Variables      *
 ************************************/

int  lineas = 60;
int  hoja   = 0;
int  suma   = 0;
int  sw     = 1;

double  impopr = 0;
double  impinv = 0;
double  crdins = 0;
double  sumopr = 0;
double  suminv = 0;
double  sumcrd = 0;
double  sumtot = 0;
double  total = 0;

EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE sqlca.h;

main(argc,argv)
   int argc;
   char *argv[];
   BEGIN
    strcpy (usr.arr,argv[1]);
    strcpy (pwd.arr,argv[2]);
    strcpy (capmes.arr,argv[3]);
    strcpy (centro.arr,argv[4]);
    strcpy (depto.arr,argv[5]);

    usr.len = strlen(usr.arr);
    pwd.len = strlen(pwd.arr);
    capmes.len = strlen(capmes.arr);
    centro.len = strlen(centro.arr);
    depto.len = strlen(depto.arr);

/************************************
 *      Seleccion de Pramametros      *
 ************************************/

EXEC SQL whenever sqlerror goto error_oracle;
```

```
EXEC SQL connect :usr identified by :pwd;
EXEC SQL whenever not found continue;
EXEC SQL SELECT distinct to_char(sysdate,'YYYY'),1,31,
          decode(cve_mess,01,'ENERO',02,'FEBRERO',03,'MARZO',04,'ABRIL',
          05,'MAYO',06,'JUNIO',07,'JULIO',08,'AGOSTO',09,'SEPTIEMBRE',
          10,'OCTUBRE',11,'NOVIEMBRE',12,'DICIEMBRE')
        into :anio4, :dia_ini, :dia_fin, :nom_mes
        from solct
       where cve_mess = :capmes;

if ((salida = fopen("tmp/solc392.lis","w")) == NULL)
  BEGIN
   printf ("No se pudo generar el reporte solc392.lis \n");
   goto error_oracle;
  END

/***********************************
 *      Declaracion de Cursores      *
 ***********************************/

EXEC SQL whenever not found continue;
EXEC SQL select cve_ctro,cve_ecom,des_dpto,cve_ofip,cve_prog,des_prog
        into :cetro,:ecom,:desdpto,:ofipg,:prog,:desprog
        from dpto,prog
       where substr(cve_subd,1,2)=substr(cve_gcia,1,2)
         and cve_ctro = :centro
         and cve_ecom = :depto;

EXEC SQL whenever not found continue;
EXEC SQL select des_ctro
        into :desctr
        from cetro
       where cve_ctro = :centro;

 EXEC SQL declare C01 cursor for
     select des_reng,solct.cve_reng,imp_pre,imp_pre,imp_pre
       from solct,renga
      where cve_ctro = :centro
        and cve_ecom = :depto
        and cve_mess = :capmes
        and renga.cve_reng = solct.cve_reng
    order by solct.cve_reng;

/***********************************
 *      Lectura de los Cursores      *
 ***********************************/

 EXEC SQL open C01;
 EXEC SQL whenever not found go to finb;
 for( ; ; )
  BEGIN
```

```
    EXEC SQL fetch C01 into :dreng,:rengl,:impopr,:impinv,:crdins;
    if (lineas > 59 )
      BEGIN
        titulos();
        corte1();
      END
    else
      BEGIN
        corte1();
      END
  END
finb:
conta();
impri();
perso();
fclose(salida);
EXEC SQL close C01;
EXEC SQL commit work release;
return(0);
error_oracle:
   EXEC SQL whenever sqlerror continue;
   printf("=============================================================\n");
   printf("# Error : %s \n",sqlca.sqlerrm.sqlerrmc);
   printf("=============================================================\n");
   EXEC SQL rollback work release;
   fclose(salida);
   exit(0);
END

/*************************************
 *      Funcion de los encabezados     *
 *************************************/

corte1()
  BEGIN
  EXEC SQL whenever not found continue;
   EXEC SQL
     select to_char(:impopr, '$9,999,999.99'),
          to_char(:impinv, '$9,999,999.99'),
          to_char(:crdins, '$9,999,999.99'),
          to_char(:total, '$9,999,999.99')
      into :wimpopr,:wimpinv,:wcrdins,:wtotal
      from dual;

     fprintf(salida,"# %-54s|%6d  |",dreng.arr,rengl);
      if ( rengl > 199 && rengl < 300 )
        BEGIN
          strcpy(wtotal.arr,wimpopr.arr);
          fprintf(salida,"%14s|%14s|%14s|",wimpopr.arr," "," ");
          total=impopr;
          sumopr=sumopr+total;
```

```
        END
       if ( rengl > 299 && rengl < 400 )
        BEGIN
         strcpy(wtotal.arr,wimpinv.arr);
         fprintf(salida,"%14s|%14s|%14s|"," ",wimpinv.arr," ");
         total=impinv;
         suminv=suminv+total;
        END
       if ( rengl < 200 )
        BEGIN
         strcpy(wtotal.arr,wcrdins.arr);
         fprintf(salida,"%14s|%14s|%14s|"," "," ",wcrdins.arr);
         total=crdins;
         sumcrd=sumcrd+total;
        END
      fprintf(salida,"%14s #\n",wtotal.arr);

         lineas++;
      sumtot=sumtot+total;
      impopr = 0;
      impinv = 0;
      crdins = 0;
      total = 0;
   END

conta()
    BEGIN
     for(suma=50-lineas; suma>1; suma--)
       BEGIN
        fprintf(salida,"#%55s|%69s|%14s|%14s|"," "," "," "," ");
        fprintf(salida,"%14s|%15s#\n"," "," ");
       END
    END

titulos()
BEGIN
 hoja++;
 fprintf (salida,"\f\n");
 finaz();
 fprintf (salida,"%81s \n\n","Solicitud de Credito Presupuestal");
 fprintf (salida,"%14s %11s %2s %5d","Para el mes de",nom_mes.arr,"de",anio4);
 fprintf (salida,"%36s %-5d %-40s \n","Centro de Trabajo:",cetro,desctr.arr);
 fprintf (salida,"%67s %9d %-40s\n","Departamento :",ecom,desdpto.arr);
 fprintf (salida,"%22s %49s","Cubriendo el Periodo :","Oficina Pagadora : ");
 fprintf (salida,"%-5d %-40s\n",cetro,desctr.arr);
 fprintf (salida,"%3s %1d %2s %2d %3s","Del",dia_ini,"al",dia_fin,"de");
 fprintf (salida,"%11s %2s %4d %32s",nom_mes.arr,"de",anio4,"Subdireccion :");
 fprintf (salida,"%6s    %-30s\n\n",prog.arr,desprog.arr);
 fprintf (salida,"=============================================================");
 fprintf (salida,"=============================================================");
 fprintf (salida,"=============================================================\n");
```

```
fprintf (salida,"#%55s","D e s c r i p c i o n          ");
fprintf (salida,"|%9s|"," Renglon ");
fprintf (salida,"%14s|%14s|%14s|","Operacion ","Inversion ","Credito  ");
fprintf (salida,"%14s #\n","Total   ");
fprintf (salida,"#%55s|%9s|%14s|%14s|"," "," "," "," ");
fprintf (salida,"%14s|%14s #\n","Institucional"," ");
fprintf (salida,"═══════════════════════════════════════════════════════");
fprintf (salida,"═══════════════════════════════════════════════════════");
fprintf (salida,"═══════════════════════════════════════════════════════\n");
lineas = 15;
END

finaz()
BEGIN
fprintf(salida,"%78s","C^HCO^HOO^HOR^HRD^HDI^HIN^HNA^HAC^HCI^HI");
fprintf(salida,"%s","O^HON^HN E^HEJ^HJE^HEC^HCU^HUT^HTI^HIV^HV");
fprintf(salida,"%s \n","A^HA D^HDE^HE F^HFI^HIN^HNA^HAN^HNZ^HZA^HAS^HS");
fprintf(salida,"%83s","S^HSU^HUB^HBD^HDI^HIR^HRE^HEC^HCC^HCI^HI");
fprintf(salida,"%s \n","O^HON^HN D^HDE^HE F^HFI^HIN^HNA^HAN^HNZ^HZA^HAS^HS");
fprintf(salida,"%87s","G^HGE^HER^HRE^HEN^HNC^HCI^HIA^HA D^HDE^HE ");
fprintf(salida,"%s \n","F^HFI^HIN^HNA^HAN^HNZ^HZA^HAS^HS");
END

perso()
BEGIN
  EXEC SQL whenever not found continue;
  EXEC SQL  select nom_perso,des_puest
          into :nombre1,:puesto1
          from puesto
         where cla_puest = 1;
  EXEC SQL  select nom_perso,des_puest
          into :nombre2,:puesto2
          from puesto
         where cla_puest = 2;

fprintf (salida,"\n\n\n\n");
fprintf (salida,"      ----------------------------------------");
fprintf (salida,"      ----------------------------------------\n");
fprintf (salida,"%50s %50s \n",nombre2.arr,nombre1.arr);
fprintf (salida,"%49s %50s \n",puesto2.arr,puesto1.arr);
lineas += 7;
END

impri()
BEGIN
  EXEC SQL whenever not found continue;
  EXEC SQL
    select to_char(:sumopr, '$9,999,999.99'),
        to_char(:suminv, '$9,999,999.99'),
        to_char(:sumcrd, '$9,999,999.99'),
        to_char(:sumtot, '$9,999,999.99')
```

```
        into :wsumopr,:wsuminv,:wsumcrd,:wsumtot
        from dual;
 fprintf (salida,"=============================================================");
 fprintf (salida,"=============================================================");
 fprintf (salida,"=============================================================\n");
 fprintf (salida,"%16s","Nuevos Pesos N$ ");
 fprintf (salida,"%40s|%9s|%14s|%14s|","Total "," ",wsumopr.arr,wsuminv.arr);
 fprintf (salida,"%14s|%14s #\n",wsumcrd.arr,wsumtot.arr);
 fprintf (salida,"%56s");
 fprintf (salida,"=============================================");
 fprintf (salida,"=============================================\n");
 sumopr=0;
 suminv=0;
 sumcrd=0;
 sumtot=0;
 lineas += 3;
END
```

El programa **mes392.pc** es semejante al programa **solc392.pc** que es el programa anterior, sólo existe una mínima diferencia entre estos dos, el primero entrega información de todo lo presupuestado en un mes a todos los Departamentos, mientras que el segundo es más específico, nos muestra el reporte de salida por cada Centro de Trabajo, Departamento que se desee en el mes.

COORDINACION EJECUTIVA DE FINANZAS
SUBDIRECCION DE FINANZAS
GERENCIA DE FINANZAS
Solicitud de Credito Presupuestal

Para el mes de ENERO de 1995
Cubriendo el Periodo :
Del 1 al 31 de ENERO de 1995

Centro de Trabajo: 476 SIST. TRONCAL DUCTOS CENTRO (GAS) VENTA DE CARPIO
Departamento : 40000 SUBGERENCIA VENTA DE CARPIO GAS NATURAL
Oficina Pagadora : 476 SIST. TRONCAL DUCTOS CENTRO (GAS) VENTA DE CARPIO
Subdireccion : K PEMEX - GAS Y PETROQUIMICA BASICA

# # Descripcion	Renglon	Operacion	Inversion	Credito Institucional	Total # #
# DERECHOS E IMPUESTOS DIRECTOS	112			$150.00	$150.00 #
# I.V.A. ACREDITABLE POR PEMEX (EGRESOS)	118			$1,300.00	$1,300.00 #
# SUELDOS,SALA. Y PRESTACIONES NORMALES Y EXTRAORD.	201	$35,000.00			$35,000.00 #
# ADQUISICION DE MATERIALES, ACCES. Y ARTS. DE SERV.	202	$5,000.00			$5,000.00 #
# CONSERVACION Y MANTENIMIENTO DIVERSO POR CONTRATO	204	$1,000.00			$1,000.00 #
# SERVS. DE TRANSP. Y GASTOS CONEXOS PAGADOS A TERC.	212	$1,000.00			$1,000.00 #
# ARRENDAMIENTOS VARIOS	215	$100.00			$100.00 #
# GASTOS DE PREVISION SOCIAL PAGADOS AL PERSONAL	234	$12,500.00			$12,500.00 #
# GASTOS GENERALES PAGADOS A TERCEROS	235	$6,500.00			$6,500.00 #
#					#
#					#
Nuevos Pesos N$ Total		$61,100.00	$.00	$1,450.00	$62,550.00 #

Ing. Jose H. Garza Pe~a
Subgerente Zona Centro

Lic. Manuel J. Llanes Valdez
Subgerente Soporte Tecnico

6.3 Programa de Actualización de Saldos.

Los programas **actsal3i.pc, actsal3l.pc, actsal2i.pc, actsal2l.pc** y **actsali.pc** y **actsall.pc** son los programas en **Pro*C** que se encargan de actualizar los saldos en la Base de Datos, validando que exista un respaldo de estos hechos con anterioridad a nivel de Sistema Operativo. La tabla siguiente describe qué tablas son las que actualizan cada uno de estos programas y el nombre del archivo que validan que exista.

TABLA DE EN DONDE SE REALIZA LA ACTUALIZACION DE SALDOS

Programa	Tablas que Actualizan	Archivos que Validan
actsal3i.pc	SAL_392I	ressal3i.(mes)
actsal3l.pc	SAL_392L	ressal3l.(mes)
actsal2i.pc	SAL_219I	ressal2i.(mes)
actsal2l.pc	SAL_219L	ressal2l.(mes)
actsali.pc	SALDOL	ressall.(mes)
actsall.pc	SALDOI	ressali.(mes)

Trayectoria de los programas que actualizan y respaldan saldos

/dsr/dsrloc/serf/bin

El nombre de los programas que actualizan y respaldan saldos:

NOMBRE	TAMAÑO EN BYTS
actsal2i.pc	12984
actsal2l.pc	12984
actsal3i.pc	12983
actsal3l.pc	12983
actsall.pc	13125
actsali.pc	13124
ressal2i.pc	4761
ressal2l.pc	4772
ressal3i.pc	4772
ressal3l.pc	4771
ressall.pc	4838
ressali.pc	4838
cuadro.c	2262

El programa siguiente ejemplifica la estructura de los programas mencionados, así como la validación que estos realizan y la forma en que realizan la actualización de saldos; primero, seleccionan los movimientos del mes, para nuestro caso, los de pago inmediato de la tabla **REGAS**, luego selecciona la ministración que se le asigna a cada Departamento de las tablas **MINIS, PRESUP** y **PASIVO** según sea el caso, para el ejemplo de **MINIS**, y por último toma el saldo anterior de las tablas de saldos para el ejemplo es la tabla **SAL_392I** y actualiza en la misma. Resumiendo lo anterior en una expresión aritmética seria.

SAL_ACT = Saldo Anterior + (Presupuesto Asignado - La Suma de Movimientos).

Todos los programas realizan lo anterior pero cada uno en diferenten tabla como se vio.

```
/*********************************************************
 *        Sistema de Estados de Resultados Financieros   *
 *       Proceso que Realiza La Actualizacion de Saldos  *
 *         Del Pasivo Por Departamento Institucional     *
 *                                                       *
 * Tablas Acesadas: Grupo [S] Sal_219i [S,I] Pasivo [S]  *
 *            Regas [S] Rsaldo [S,I,U]                   *
 *                                                       *
 * Autor:  Arturo Leon Juarez          Programa: actsal2i *
 * Fecha: 01/01/94                   Equipo : hp9000/842 *
 *********************************************************/

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "cuadro.c"
#define BEGIN {
#define END }

FILE *mescp;

EXEC SQL BEGIN DECLARE SECTION;

VARCHAR cv_us[12];
VARCHAR passwd[12];
VARCHAR nomarch[17];
VARCHAR cve_dpto[6];
VARCHAR cve_ctro[4];
VARCHAR cve_reng[4];
VARCHAR cve_momc[3];
VARCHAR cve_prog[3];
VARCHAR fec_elab[10];
VARCHAR anio[5];
VARCHAR meas[4];
VARCHAR nom_mes[12];
VARCHAR ing_egr[2];

int anio_min;
int anio_max;
int rob;
int num;
int dia;
int mes;

double imp_cont,imp_sal,imp_pre;

char c;

 /*****************************
  *  Inicializacion de Variables  *
  *****************************/
```

```
double pre     = 0;
double sal     = 0;
double pre_acum = 0;
double ejc_acum = 0;
double imp_16  = 0;
double imp_26  = 0;
double sal3    = 0;
double pre3    = 0;
double tpre3   = 0;
double ejer3   = 0;
double tejer3  = 0;

int ban5=1;

EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE sqlca.h;

main(argc,argv)
  int argc;
 char *argv[];
 BEGIN

 strcpy(cv_us.arr,*++argv);
 strcpy(passwd.arr,*++argv);

 cv_us.len=strlen(cv_us.arr);
 passwd.len=strlen(passwd.arr);

EXEC SQL whenever sqlerror goto error_oracle;
EXEC SQL CONNECT :cv_us IDENTIFIED BY :passwd;
EXEC SQL SELECT max(to_char(fec_elab,'DD')),max(to_char(fec_elab,'MM')),
          decode(max(to_char(fec_elab,'MM')),01,'Enero',02,'Febrero',03,
          'Marzo',04,'Abril',05,'Mayo',06,'Junio',07,'Julio',08,'Agosto',
          09,'Septiembre',10,'Octubre',11,'Noviembre',12,'Diciembre'),
          max(to_char(fec_elab,'YYYY')),to_char(sysdate,'DD-MM-YY')
       INTO :dia, :mes, :nom_mes, :anio, :fec_elab FROM regas;

num=mes;
EXEC SQL SELECT max(cve_mess)
       INTO :meas
       FROM sal_219i;

/*********************************************************
 *    Rutina para identificar mes 01 y ponerlo como minimo   *
 *********************************************************/

EXEC SQL whenever not found continue;
EXEC SQL SELECT max(substr(fec_elab,8,9))
       INTO :anio_max
       FROM regas;
```

```
EXEC SQL whenever not found continue;
EXEC SQL SELECT min(substr(fec_elab,8,9))
        INTO :anio_min
        FROM regas;

if (strcmp (anio_min,anio_max) != 0)
   mes=num=1;

if (num == 1)
  BEGIN
   funcion();
  END
if (num > 1)
  BEGIN
   meas.arr[meas.len]='\0';
   rob=atoi(meas.arr);
   rob=rob +1;
  if ( num == rob )
    BEGIN
     strcpy(nomarch.arr,"resp/ressal2i.");
     strcat(nomarch.arr,meas.arr);

     if (( mescp = fopen(nomarch.arr,"r"))==NULL)
       BEGIN
        limpia();
        box(1,1,24,79,4);
        gotoxy(15,15);printf("Error, los Saldos del Mes : %9s ",meas.arr);
        gotoxy(17,15);printf("No Han Sido Respaldados");
        gotoxy(19,38);printf("Oprima <Retorno> para Continuar: ");
         c=getchar();
         goto salida;
       END
     else
       funcion();
    END
 else
  BEGIN
   limpia();
   box(1,1,24,79,4);
   gotoxy(10,15);printf("Mes en Existencias %d ",num);
   gotoxy(11,15);printf("Mes a  Actualizar  %d",rob-1);
   gotoxy(13,15);printf("No Se Genera La Actualizacion de Saldos Porque");
   gotoxy(15,15);printf("El Mes de Saldos No Corresponde al Mes Anterior");
   gotoxy(17,15);printf("Que Se Esta Procesando ");
   gotoxy(19,38);printf("Oprima <Retorno> para Continuar: ");
     c=getchar();
     goto salida;
  END
END

goto sale;
```

```
error_oracle:
  limpia();
   EXEC SQL whenever sqlerror continue;
    printf("\n\t=============================================================\n");
    printf("\t# Error :%s \n",sqlca.sqlerrm.sqlerrmc);
    printf("\t=============================================================\n");
    box(1,1,24,79,4);
    gotoxy(22,38);printf("Oprima <Retorno> para Continuar: ");
    c=getchar();
   EXEC SQL rollback work release;
   exit(0);

sale:
  EXEC SQL commit work release;
  exit(0);

salida:
  fclose (mescp);
  exit(0);
END

/*****************************************************
*    Inicia La Funcion uno  para Actualizar las Tablas  *
*    Saldos Institucionales                             *
*****************************************************/

funcion()

   /*********************************
    * Abro y Declaro el Primer Cursor *
    *********************************/

BEGIN
 EXEC SQL delete from sal_219i;
 EXEC SQL DECLARE lee_cpsic1 CURSOR FOR
       SELECT cve_ctro,cve_dpto,cve_prog,cve_reng,sum(imp_cont)
        FROM grupo
        WHERE cve_momc in (14,34)
      GROUP BY cve_ctro,cve_dpto,cve_prog,cve_reng
      ORDER BY cve_ctro,cve_dpto,cve_prog,cve_reng;

EXEC SQL OPEN lee_cpsic1;
EXEC SQL whenever not found goto fin_lee1;
for ( ; ; )
BEGIN
EXEC SQL FETCH lee_cpsic1 INTO :cve_ctro,:cve_dpto,:cve_prog,:cve_reng,
                  :imp_cont;
ini_var();

EXEC SQL INSERT INTO rsaldo (cve_ctro,cve_dpto,cve_prog,cve_reng,imp_16,ing_egr)
        VALUES (:cve_ctro,:cve_dpto,:cve_prog,:cve_reng,:imp_cont,'E');
```

```
END
fin_lee1:
EXEC SQL close lee_cpsic1;
EXEC SQL commit;

  /************************************
   * Abro y Declaro el Segundo Cursor *
   ************************************/

EXEC SQL DECLARE lee_cpsic2 CURSOR FOR
      SELECT cve_ctro,cve_dpto,cve_prog,cve_reng,sum(imp_cont)
       FROM grupo
       WHERE cve_momc = 26
       AND cve_reng >= 100
       AND cve_reng < 200
       AND cve_prog is not null
   GROUP BY cve_ctro,cve_dpto,cve_prog,cve_reng;

EXEC SQL OPEN lee_cpsic2;
EXEC SQL whenever not found goto fin_lee2;
for ( ; ; )
BEGIN
EXEC SQL FETCH lee_cpsic2 INTO :cve_ctro,:cve_dpto,:cve_prog,:cve_reng,
                  :imp_cont;
 ini_var();

 EXEC SQL whenever not found goto inserta26;
 EXEC SQL UPDATE rsaldo
        SET imp_26 = :imp_cont,ing_egr='I'
       WHERE cve_ctro=:cve_ctro
        AND cve_dpto=:cve_dpto
        AND cve_reng=:cve_reng
        AND cve_prog=:cve_prog
        AND ing_egr='I';

 ban5=0;
 inserta26:
 if ( ban5 == 1)
   BEGIN
   EXEC SQL INSERT INTO rsaldo (cve_ctro,cve_dpto,cve_prog,cve_reng,imp_26,
                  ing_egr)
        VALUES (:cve_ctro,:cve_dpto,:cve_prog,:cve_reng,:imp_cont,'I');
   END
END
fin_lee2:
EXEC SQL close lee_cpsic2;
EXEC SQL commit;

  /*********************************
   * Abro y Declaro el Tercer Cursor *
   *********************************/
```

```
  EXEC SQL DECLARE lee_cpsic3 CURSOR FOR
     SELECT cve_ctro,cve_dpto,cve_prog,,cve_reng,sum(imp_cont)
      FROM grupo
      WHERE cve_momc = 26
      AND cve_reng >= 200
      AND cve_reng < 400
      AND cve_prog is not null
   GROUP BY cve_ctro,cve_dpto,cve_prog,cve_reng;

EXEC SQL OPEN lee_cpsic3;
EXEC SQL whenever not found goto fin_lee3;
for ( ; ; )
BEGIN
EXEC SQL FETCH lee_cpsic3 INTO :cve_ctro,:cve_dpto,:cve_prog,:cve_reng,
                  :imp_cont;
ini_var();

EXEC SQL whenever not found goto inserta27;
EXEC SQL UPDATE rsaldo
       SET imp_26 = :imp_cont
      WHERE cve_ctro=:cve_ctro
       AND cve_dpto=:cve_dpto
       AND cve_reng=:cve_reng
       AND cve_prog=:cve_prog
       AND ing_egr='E';

ban5=0;
inserta27:
if ( ban5 == 1)
  BEGIN
   EXEC SQL INSERT INTO rsaldo (cve_ctro,cve_dpto,cve_prog,cve_reng,imp_26,
                  ing_egr)
       VALUES (:cve_ctro,:cve_dpto,:cve_prog,:cve_reng,:imp_cont,'E');
  END
END
fin_lee3:
EXEC SQL close lee_cpsic3;
EXEC SQL commit;

  /*********************************
   * Abro y Declaro el Cuarto Cursor *
   *********************************/

EXEC SQL DECLARE lee_sal CURSOR FOR
    SELECT cve_ctro,cve_dpto,cve_prog,cve_reng,nvl(sum(imp_sal),0),
       nvl(sum(pre_acum),0),nvl(sum(eje_acum),0),ing_egr
     FROM sal_219i
   GROUP BY cve_ctro,cve_dpto,cve_prog,cve_reng,ing_egr;

EXEC SQL OPEN lee_sal;
EXEC SQL WHENEVER NOT FOUND GOTO lee_fin1;
```

```
for ( ; ; )
BEGIN
EXEC SQL FETCH lee_sal INTO :cve_ctro,:cve_dpto,:cve_prog,:cve_reng,:imp_cont,
                  :pre_acum,:cje_acum,:ing_egr;

   EXEC SQL WHENEVER NOT FOUND GOTO inserta;
   EXEC SQL UPDATE rsaldo
        SET imp_sal = :imp_cont,
          cje_acum = :cje_acum,
          pre_acum = :pre_acum
        WHERE cve_ctro = :cve_ctro
         AND cve_dpto = :cve_dpto
         AND cve_prog = :cve_prog
         AND cve_reng = :cve_reng
         AND ing_egr = :ing_egr;
ban5=0;
inserta:
if (ban5==1)
  BEGIN
   EXEC SQL INSERT INTO rsaldo
              (cve_ctro,cve_dpto,cve_prog,cve_reng,imp_sal,
               pre_acum,cje_acum,ing_egr)
    VALUES(:cve_ctro,:cve_dpto,:cve_prog,:cve_reng,:imp_cont,:pre_acum,
          :cje_acum,:ing_egr);
  END
END
lee_fin1:
EXEC SQL CLOSE lee_sal;
EXEC SQL COMMIT;

  /*********************************
   * Abro y Declaro el Quinto Cursor *
   *********************************/

EXEC SQL DECLARE lee_minis CURSOR FOR
     SELECT cve_ctro,cve_prog,cve_dpto,cve_reng,ing_egr,sum(imp_pre)
      FROM pasivo
      WHERE cve_mess = :num
    GROUP BY cve_ctro,cve_prog,cve_dpto,cve_reng,ing_egr
    ORDER BY 1, 2, 3, 4;

EXEC SQL OPEN lee_minis;
EXEC SQL whenever not found goto fin_lee4;
for ( ; ; )
BEGIN
EXEC SQL FETCH lee_minis INTO :cve_ctro,:cve_prog,:cve_dpto,:cve_reng,:ing_egr,
                  :imp_cont;
ini_var();
EXEC SQL whenever not found goto inserta21;
EXEC SQL UPDATE rsaldo
       SET imp_pre = :imp_cont
```

```
      WHERE cve_ctro = :cve_ctro
       AND cve_dpto = :cve_dpto
       AND cve_reng = :cve_reng
       AND cve_prog = :cve_prog
       AND ing_egr = :ing_egr;
 ban5=0;
 inserta21:
 if (ban5 == 1)
  BEGIN
   EXEC SQL INSERT INTO rsaldo (cve_ctro,cve_prog,cve_dpto,cve_reng,imp_pre,
                ing_egr)
      VALUES (:cve_ctro,:cve_prog,:cve_dpto,:cve_reng,:imp_cont,:ing_egr);
  END
END
fin_lee4:
EXEC SQL close lee_minis;
EXEC SQL commit;

  /*********************************
   * Abro y Declaro el Sexto Cursor *
   *********************************/

EXEC SQL DECLARE lee_repsal CURSOR FOR
     SELECT cve_ctro,cve_prog,cve_dpto,cve_reng,imp_16,imp_26,imp_pre,
        imp_sal,eje_acum,pre_acum,ing_egr
      FROM rsaldo
    ORDER BY cve_ctro,cve_prog,cve_dpto,cve_reng,ing_egr;

EXEC SQL OPEN lee_repsal;
EXEC SQL whenever not found goto lee_act1;
for ( ; ; )
BEGIN
EXEC SQL FETCH lee_repsal INTO :cve_ctro,:cve_prog,:cve_dpto,:cve_reng,:imp_16,
                :imp_26,:pre,:sal,:eje_acum,:pre_acum,:ing_egr;
ini_var();

pre3=pre;
ejer3=imp_16-imp_26;
sal3=sal+(pre3-ejer3);
tpre3=pre_acum+pre3;
tejer3=eje_acum+ejer3;

EXEC SQL whenever sqlerror goto error_oracle2;
EXEC SQL INSERT INTO sal_219i (cve_ctro,cve_prog,cve_dpto,cve_reng,cve_mess,
                cve_anio,imp_sal,pre_acum,eje_acum,ing_egr)
     VALUES (:cve_ctro,:cve_prog,:cve_dpto,:cve_reng,:mes,:anio,:sal3,:pre3,
          :ejer3, :ing_egr);
END
lee_act1:
EXEC SQL close lee_repsal;
EXEC SQL delete rsaldo;
```

```
 EXEC SQL commit work release;
 exit(0);

error_oracle2:
  limpia();
    EXEC SQL close lee_repsal;
    EXEC SQL delete rsaldo;
    EXEC SQL whenever sqlerror continue;
      printf("\n\t=============================================================\n");
      printf("\t# Error :%s \n",sqlca.sqlerrm.sqlerrmc);
      printf("\t=============================================================\n");
      box(1,1,24,79,4);
      gotoxy(22,38);printf("Oprima <Retorno> para Continuar: ");
      c=getchar();
    EXEC SQL rollback work release;
    exit(0);
END

ini_var()
 BEGIN
   cve_ctro.arr[cve_ctro.len]='\0';
   cve_prog.arr[cve_prog.len]='\0';
   cve_dpto.arr[cve_dpto.len]='\0';
   cve_reng.arr[cve_reng.len]='\0';
   ing_egr.arr[ing_egr.len]='\0';
   ban5=1;
 END
```

6.4 Programa para Respaldar Saldos.

Para finalizar, están los programas **ressal2i.pc**, **ressal2l.pc**, **ressal3i.pc**, **ressal3l.pc**, **ressall.pc** y **ressali.pc**, estos son los programas que se encargan de seleccionar la información de las tablas **sal_219l**, **sal_219i**, **sal_392i**, sal_392l, **saldol** y **saldoi** respectivamente para realizar un respaldo de estas a nivel de Sistema Operativo, generando un archivo de salida con el mismo nombre del programa, sólo cambiando la extensión del archivo que será la del mes que se respalde, como ejemplo: el programa **ressal2i.pc** que deja de salida el archivo plano:

ressal2i.1 para el mes de Enero, así sucesivamente.

Todo los archivos que se respaldan quedan bajo el directorio **resp**

```
/*************************************************************
 *        Sistema de Estados de Resultados Financieros       *
 *          Proceso que Realiza el Respaldo de Saldos        *
 *          Del Pasivo Por Departamento Institucional        *
 *                                                           *
 * Tablas Acesadas: rp_00 [S] Sal_219i [S]                   *
 *                                                           *
 * Autor:  Arturo Leon Juarez          Programa: ressal2i    *
 * Fecha: 01/01/94                   Equipo  : hp9000/842    *
 *************************************************************/

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "cuadro.c"
#define BEGIN {
#define END }
FILE *mescp;

  /*****************************
   *  Declaracion de variables *
   *****************************/

EXEC SQL BEGIN DECLARE SECTION;
VARCHAR cve_dpto[06];
VARCHAR cve_ctro[04];
VARCHAR cve_prog[03];
VARCHAR cve_reng[04];
VARCHAR cve_mess[03];
VARCHAR cve_anio[03];
VARCHAR passwd[12];
VARCHAR cv_us[12];
VARCHAR nomarch[9];
VARCHAR mes1[03];
VARCHAR mes[03];
VARCHAR ano[05];
VARCHAR ing_egr[02];

int num2;
int num;
double imp_sal;
double eje_acum;
double pre_acum;
char c;

EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE sqlca.h;

main(argc,argv)
  int argc;
  char *argv[];
```

```
    BEGIN
    strcpy(cv_us.arr,argv[1]);
    strcpy(passwd.arr,argv[2]);
    cv_us.len=strlen(cv_us.arr);
    passwd.len=strlen(passwd.arr);
titulos();

EXEC SQL WHENEVER SQLERROR GOTO error_oracle;
EXEC SQL CONNECT :cv_us IDENTIFIED BY :passwd;
EXEC SQL SELECT max(to_char(fec_elab,'MM'))
        INTO :mes
        FROM rp_00;
EXEC SQL SELECT nvl(max(cve_mess),0),nvl(max(cve_anio),0)
        INTO :mes1, :ano
        FROM sal_219i;
  gotoxy(11,28);printf("del mes : %2s  del : %4s",mes1.arr,ano.arr);
  strcpy(nomarch.arr,"resp/ressal2i.");
  strcat(nomarch.arr,mes1.arr);
  num=atoi(mes.arr);
  num2=atoi(mes1.arr);
  num2=num2+1;
   if ((num == num2) && ((mescp = fopen(nomarch.arr,"r")) != NULL))
    BEGIN
     gotoxy(15,15);printf("Ya Existe el Respaldo y no Estan Actualizados");
              printf(" los Saldos ");
     goto salir;
    END
   fclose(mescp);
   if (( num < num2) && ((mescp = fopen(nomarch.arr,"r")) != NULL))
    BEGIN
     gotoxy(15,15);printf("Ya Existe el Respaldo y Estan Actualizados");
              printf(" los Saldos");
     goto salir;
    END
   fclose(mescp);
   if ((mescp = fopen(nomarch.arr,"w")) == NULL)
    BEGIN
     gotoxy(15,15);printf("No Puedo Abrir el Archivo de Salida ");
              printf(" %9s \n",nomarch.arr);
     goto salir;
    END
  gotoxy(15,19);printf("Se esta Generando el Respaldo de Saldos");

 /*****************************
  *  Declaracion de cursores  *
  *****************************/
  EXEC SQL DECLARE lee_saldos CURSOR FOR
  SELECT cve_ctro,cve_dpto,cve_prog,cve_reng,cve_mess,cve_anio,imp_sal,
      eje_acum,pre_acum,ing_egr
   FROM sal_219i;
```

```
EXEC SQL OPEN lee_saldos;
EXEC SQL WHENEVER NOT FOUND GOTO fin;
for ( ; ; )
  BEGIN
    EXEC SQL FETCH lee_saldos INTO :cve_ctro, :cve_dpto, :cve_prog, :cve_reng,
                        :cve_mess, :cve_anio, :imp_sal, :eje_acum,
                        :pre_acum, :ing_egr;
    fprintf(mescp,"%3s%5s%1s",cve_ctro.arr,cve_dpto.arr,cve_prog.arr);
    fprintf(mescp,"%3s%2s%2s",cve_reng.arr,cve_mess.arr,cve_anio.arr);
    fprintf(mescp,"%18g%18g%18g%1s\n",imp_sal,eje_acum,pre_acum,ing_egr.arr);
  END

   /*************************************
    *  Fin del proceso exitoso o con error  *
    *************************************/

 error_oracle:
    limpia();
    EXEC SQL whenever sqlerror continue;
    printf("\n\t=============================================================\n");
    printf("\t# Error :%s \n",sqlca.sqlerrm.sqlerrmc);
    printf("\t=============================================================\n");
    box(1,1,24,79,4);
    gotoxy(22,38);printf("Oprima <Retorno> para Continuar: ");
    c=getchar();
    EXEC SQL rollback work release;
    exit(0);
 fin:
    fclose(mescp);
    EXEC SQL commit work release;
    gotoxy(19,38);printf("Oprima <Retorno> para Salir: ");
    c=getchar();
    exit(0);
 salir:
    EXEC SQL rollback work release;
    gotoxy(19,38);printf("Oprima <Retorno> para Salir: ");
    c=getchar();
    exit(0);
END
```

6.5 Programa utilizado como libreria

Aquí, dentro del capítulo de reportes y en donde se incluyeron todos los programas en **Pro*C** también veremos una librería que es utilizada por los programas y es el programa **cuadro.c**, el cual nos muestra una carátula para cuando se piden los parámetros que el programa toma en línea, o para cuando sucede un error y se manda un mensaje en la pantallas. Con este programa se concluye lo referente al capítulo VI y en si, a la aplicación en lo que respecta a programación solo faltan por mencionar y describir algunos programas, los cuales se verán en el capítulo VII junto con la ejecución de la aplicación en forma interactiva.

```
/*************************************************************
 *        Sistema de Estados de Resultados Financieros       *
 *         Programa que Genera el Marco Fecha actual         *
 *                                                           *
 *                                                           *
 * Autor:  Arturo Leon Juarez          Programa: cuadro      *
 * Fecha: 01/01/94                   Equipo  : hp9000/842 *
 *************************************************************/
#include <stdio.h>
#include <time.h>

struct tm *ptime, *localtime();
time_t time(), nseg;

int diam;
int mesm;
int anom;

titulos()
 {
  int j;
  limpia();
  box(1,1,24,79,4);
  gotoxy(2,30);printf("Petroleos Mexicanos");
  gotoxy(3,15);printf("Subgerencia de Operacion de Ductos y Terminales");
  gotoxy(2,4);printf("Sistema");
  gotoxy(3,4);printf("\033[1mS.e.r.f\033[0m");
  fecha();
  gotoxy(2,70);printf(" Fecha ");
  gotoxy(3,68);printf("\033[1m %2d/%02d/%02d \033[0m",diam,mesm,anom);
  gotoxy(6,16);printf("Sistema de Estados de Resultados Financieros");
  gotoxy(8,28);printf("Respaldo de Saldos");
}
gotoxy(x,y)
int x,y;
{
  char xx[3],yy[3];
  sprintf(xx,"%d\0",x);
  sprintf(yy,"%d\0",y);
  printf("\033[%s;%sH",xx,yy);
}
box(x1,y1,x2,y2,x3)
int x1,y1,x2,y2,x3;
{
 register int j;
 printf("\033(0");
 gotoxy(x1,y1);
     printf("l\033[D\033[B");
 for (j=1;j<x2-x1;j++)
     printf("x\033[B\033[D");
     printf("m");
```

```
 for (j=1;j<y2-y1;j++)
    printf("q");
    printf("j");
 for (j=1;j<x2-x1;j++)
    printf("\033[A\033[Dx");
    printf("\033[A\033[Dk");
 for (j=1;j<y2-y1;j++)
    printf("\033[2Dq");

  gotoxy(x3,y1);
    printf("t");
 for (j=1;j<y2-y1;j++)
    printf("q");
  printf("u");
  printf("\033(B");
}
limpia()
{
   printf("\033[2J");
}
fecha()
{
  diam=mesm=anom=0;
  nseg=time(NULL);
  ptime=localtime(&nseg);

  anom = ptime->tm_year;
  mesm += (ptime->tm_mon+1);
  diam += (ptime->tm_mday);
}
```

APLICACION EN FORMA INTERACTIVA

7.1 Introducción

Desarrollo de la aplicación en forma interactiva. Dentro de este capítulo se muestra el espacio ocupado por la aplicación a nivel se Sistema Operativo y Base de Datos, tanto en los programas ejecutables como en sus programas fuente, la trayectoria donde estos estan ubicados y la manera en que el usuario entra al sistema para explotar la información que sea de su interes. Se dará un panorama general de cuántos programas son utilizados en la aplicación y en que lenguaje son desarrollados. Se describirán algunos **shell** que sólo son programas que se encargan de ejecutar a otro, o para mandar alguna impresión. Y que durante los capítulos anteriores no se vieron.

El capítulo también muestra el **.profile** del usuario que es el que se encarga de poner el ambiente propicio para el usuario y la manera en que dentro de este se ejecuta la aplicación. Por último la distribución de los programas objeto o ejecutable dentro del Sistema Operativo.

La secuencia que seguiremos para mostrar todo lo anteriormente expuesto será como sigue.

1. Espacio general de la apliación tanto en Base de Datos como en Sistema Operativo.
2. Cantidad de programas en general y cuántos por lenguajes.
3. Trayectoria de los programa fuente y programas ejecutables.
4. Descripción de los **shells** que no se mencionaron en los capítulos anteriores y que sólo sirven para lanzar a otro programa o para mandar a imprimir.
5. Breve explicación y descripción del **.profile** y las claves de los usuarios que operan el sistema.
6. Finalmente concluiremos con una explicación de la ejecución de la aplicación.

7.2 Espacio Utilizado por la Aplicación

El espacio utilizado por la aplicación esta dividido en tres grupos los programas fuente, los programas ejecutable y la Base de Datos

Espacio Base de Datos	10500 Kb
Espacio Programas Fuentes	1000 Kb
Espacio Programas Ejecutable	14000 Kb

Total del Sistema	25500 Kb

La totalidad de los programas los desglosaremos por el lenguaje o utilería en que estos fueron desarrollados.

Programas desarrollados en Lenguaje C	13
Programas desarrollados en Pro*C	22
Programas desarrollados en Sql*forms	10
Programas desarrollados en Shells	17

Programas en Total	62

7.3 Trayectoria de los programas.

Los directorios de los programas y su ubicación dentro del sistema operativo esta en dos **fail system** "/prd" para programas en producción (ejecutables), y "/dsr" para los programas desarrollados (fuente).

Trayectoria de todos los programas fuentes

Trayectoria	Directorios y Archivos
/dsr/dsrloc/serf/bin	actsal2i.pc
	actsal2l.pc
	actsal3i.pc
	actsal3l.pc
	actsali.pc
	actsall.pc
	cuadro.c
	ressal2i.pc
	ressal2l.pc
	ressal3i.pc
	ressal3l.pc
	ressali.pc
	ressall.pc
/dsr/dsrloc/serf/cap	PASIVO.inp
	PRESP.inp
	PRE_ADPT.inp
	SOLCT.inp
/dsr/dsrloc/serf/exe	actsal2i.exe
	actsal2l.exe
	actsal3i.exe
	actsal3l.exe
	actsali.exe
	actsall.exe
	mes392.exe
	rep219i.exe
	rep219l.exe
	rep392i.exe
	rep392l.exe
	repsub2.exe
	repsub3.exe
	repsumi.exe

	repsuml.exe
	solct392.exe
/dsr/dsrloc/serf/menu	error.c
	menu_asal.c
	menu_grpt.c
	menu_impr.c
	menu_mtor.c
	menu_pcap.c
	menu_pcon.c
	menu_princ.c
	menu_rsal.c
	menu_solc.c
	menu2.c
/dsr/dsrloc/serf/pant	CAPSUBDI.inp
	CENTRO.inp
	MOMCMAL.inp
	MOMENTO.inp
	RENGLON.inp
	SALDOA.inp
/dsr/dsrloc/serf/repo	mes392.pc
	rep219i.pc
	rep219l.pc
	rep392i.pc
	rep392l.pc
	repsub2.pc
	repsub3.pc
	repsumi.pc
	repsuml.pc
	solc392.pc
/dsr/dsrloc/serf/lp_imp	
/dsr/dsrloc/serf/marco.c	

Trayectoria de todos los programas ejecutables

Trayectoria	Directorios y Archivos
/prd/serf/bin	actsal2i
	actsal2l
	actsal3i
	actsal3l
	actsali
	actsall
	ressal2i
	ressal2l
	ressal3i
	ressal3l
	ressali
	ressall
/prd/serf/bit	actsal2i.bit
	actsal2l.bit
	actsal3i.bit

	actsal3l.bit
	actsali.bit
	actsall.bit
/prd/serf/cap	PASIVO.frm
	PRESP.frm
	PRE_ADPT.frm
	SOLCT.frm
/prd/serf/exe	actsal2i.exe
	actsal2l.exe
	actsal3i.exe
	actsal3l.exe
	actsali.exe
	actsall.exe
	mes392.exe
	rep219i.exe
	rep219l.exe
	rep392i.exe
	rep392l.exe
	repsub2.exe
	repsub3.exe
	repsumi.exe
	repsuml.exe
	solct392.exe
/dsr/dsrloc/serf/menu	menu_asal
	menu_grpt
	menu_impr
	menu_mtor
	menu_pcap
	menu_pcon
	menu_rsal
	menu_solc
/prd/serf/pant	CAPSUBDI.frm
	CENTRO.frm
	MOMCMAL.frm
	MOMENTO.frm
	RENGLON.frm
	SALDOA.frm
/prd/serf/repo	mes392
	rep219i
	rep219l
	rep392i
	rep392l
	repsub2
	repsub3
	repsumi
	repsuml
	solc392
/prd/serf/.menu_princ	
/prd/serf/.lp_imp	
/prd/serf/.profile	
/prd/serf/.marco	

	actsal3l.bit
	actsali.bit
	actsall.bit
/prd/serf/cap	PASIVO.frm
	PRESP.frm
	PRE_ADPT.frm
	SOLCT.frm
/prd/serf/exe	actsal2i.exe
	actsal2l.exe
	actsal3i.exe
	actsal3l.exe
	actsali.exe
	actsall.exe
	mes392.exe
	rep219i.exe
	rep219l.exe
	rep392i.exe
	rep392l.exe
	repsub2.exe
	repsub3.exe
	repsumi.exe
	repsuml.exe
	solct392.exe
/dsr/dsrloc/serf/menu	menu_asal
	menu_grpt
	menu_impr
	menu_mtor
	menu_pcap
	menu_pcon
	menu_rsal
	menu_solc
/prd/serf/pant	CAPSUBDI.frm
	CENTRO.frm
	MOMCMAL.frm
	MOMENTO.frm
	RENGLON.frm
	SALDOA.frm
/prd/serf/repo	mes392
	rep219i
	rep219l
	rep392i
	rep392l
	repsub2
	repsub3
	repsumi
	repsuml
	solc392
/prd/serf/.menu_princ	
/prd/serf/.lp_imp	
/prd/serf/.profile	
/prd/serf/.marco	

7.4 Shells utilizados para lanzar otros programas.

Los **shells** que no fueron descritos, son los que se muestran en la tabla siguente y los cuales se encargan de ejecutar algun pograma en **Pro*C** del mismo nombre y pedir los parámetros necesarios para ejecutar estos.

Estos **shells** son similares solo cambia el nombre del programa que mandan a ejecutar.

Shell	Programa en C	Shell	Programa C
mes392.exe	mes392	solc392.exe	solc392
rep392l.exe	rep392l	rep219i.exe	rep219i
rep392i.exe	rep392i	rep219l.exe	rep219l
repsub3.exe	repsub3	repsub2.exe	repsub2
repsumi.exe	repsumi	repsuml.exe	repsuml

Veremos un ejemplo y también el **shell** que manda a imprimir los reportes (**lp_imp**).

```
#############################################
#  Shell que Lanza el programa que Genera el reporte  #
#  de la Solicitud por departamento Institucional     #
#  Nombre del Programa : rep219i.exe                  #
#############################################
error='sum log.err|cut -c1-2`
if [ $error -eq 0 ]
  then
   .marco
   echo "\033[8;15H Inicia la Generacion del Reporte de Pasivo "
   echo "\033[10;20H Por Departamentos Institucionales"
   fecha2 12
   repo/rep219i $usr $pwd
   echo "\033[14;19H \033[7m Finaliza La Generacion del Reporte \033[0m"
   fecha2 16
   echo "\033[22;40H Para Continuar Oprima < Retorn > \c"
   read nada
else
   .marco
   echo "\033[10;15H Existe un Error en el Proceso Anterior "
   echo "\033[12;15H Favor de Avisar al Area de Informatica "
   echo "\033[14;22H Oprimir  < Retorn > \c"
   read nada
fi
```

Programa que manda a imprimir (lp_imp)

```
#############################################################
#   Impesc  : Shell para impresion de reportes, por impresora       #
#             rapida, esclava y monitoreo por pantalla.             #
#                                                           #
#  Parametros : $1 -- nombre del reporte                    #
#               $2 -- clave del usuario                     #
#               $3 -- passwd del usuario                          #
#                                                           #
#  Elaboro :  Arturo Leon Juarez                 11/Ene/92        #
#############################################################
.marco
rep=`echo $1|cut -f2 -d"/"`
echo "\033[12;15HEl nombre del reporte es ====>" $rep

if  [ $0 != .lp_imp ]
 then
    cd `expr $0 : "\(.*\)/.lp_imp"`
fi

#################################
##     Los parametros Iniciales      ##
#################################

banner "$2" " " "SERF" > SRF$$
cat $1 >> SRF$$

#################################
#   Solicita opciones para impresion  #
#################################
while true
do
 sleep 1
 echo '\033[16;22H--> Teclee una opcion <-- \033[0'
 sleep 1
 echo "\033[18;15HImpresora Rapida (R) == Impresora Esclava (E) "
 echo "\033[19;15HForma comprimida (C) == Inhibir Impresion (I) "
 echo "\033[20;22HDesplegar en Pantalla (D) "
 sleep 1
 echo "\033[22;40HDigite una opcion (R,E,C,D o I ) --> \c"
 read modo
 case $modo in
    [Rr])
          ######################################
          #  Imprime el reporte en la impresora rapida  #
          ######################################
     if [ "$modo" = R -o "$modo" = r ]
       then
         cat SRF$$ |lp
         break
```

```
 fi
 ;;
[Cc])
      ##############################################
      #       Imprime el reporte en la impresora esclava       #
      # Cambia el modo de printer normal a controller mode #
      ##############################################
 if [ "$modo" = C -o "$modo" = c ]
   then
     echo \\033[5i
                  ###############################
                  #    Cambia a modo comprimido    #
                  ###############################
     echo \\017
     cat SRF$$
    ##############################################
    # Cambia el modo de controller mode a normal printer #
    ##############################################
     echo \\033[4i
                  ########################
                  #   Cambia a modo normal   #
                  ########################
     echo \\033@
     echo \\014
     break
 fi
 ;;
[Ee])
      ##############################################
      #       Imprime el reporte en la impresora esclava       #
      # Cambia el modo de printer normal a controller mode #
      ##############################################
 if [ "$modo" = E -o "$modo" = e ]
   then
     echo \\033[5i
                  ########################
                  #  Imprime en modo normal  #
                  ########################
     cat SRF$$
    ##############################################
    # Cambia el modo de controller mode a printer normal #
    ##############################################
     echo \\033[4i
     break
 fi
 ;;
[Dd])
      ##########################################
      #          Despliega el reporte en la pantalla          #
      # Cambia el Tama&o de la pantalla a 132 columnas #
      ##########################################
```

```
    if [ "$modo" = D -o "$modo" = d ]
      then
        echo \\033[?3h
       echo "\033[2;15H\033[7mPara detener la pantalla utilize F1 \033[0m"
        sleep 2

                    ####################
                    #  Despliga el archivo #
                    ####################
        cat SRF$$
      ##############################################
      #  Cambia el tama&o de la pantalla  a 80 columnas  #
      ##############################################
        echo \\033[?3l
        break
    fi
    ;;
   [Ii])
         ########################
         #    Inhibir impresion      #
         ########################
     if [ "$modo" = I -o "$modo" = i ]
      then
        break
     fi
     ;;
   *)
   .marco
   echo '\033[16;21H  La opcion no existe fue mal tecleada '
   echo '\033[17;28H  Favor de rectificar '
   sleep 5
 esac
.marco
done
rm SRF*

################################
#  fin para del programa de impresion  #
################################
```

Los archivos con extensión **.bit** le sirven para saber al usuario en que momento terminó la ejecución de la actualización de saldos, si ha ocurrido un error genera un mensaje. Para el informático existe un archivo llamado **log.err**, en este archivo se guardan los mensaje que se generan cuando al ejecutar un proceso del sistema ocurre un error dentro de la Base de Datos o Sistema Operativo.

/prd/serf/log.err

7.5 Ambiente para su ejecución interactiva

El Sistema esta compuesto por dos cargos, el primero es para cargar información del Sistema Institucional de Contabilidad hacia el Sistema de Estados de Resultados Financieros y ponerle su Departamento local. El segundo cargo es el usuario que explota la información para generar sus programas de pagos para cada Departamento.

UNIX		ORALE	
Usuario	**Passowrd**	**Usuario**	**Password**
tj30048	adm_acf	tj30048	adm_scf
tj30072	op_serf	tj30072	pemex

El **.profile** del usuario **tj30072** el cual entra directamente a la aplicación, se muestra a continuación. El **.profile** exporta las variables y trayectoria utilizadas por el usuario ademas valida que este dado de alta en el "/etc/passwd" y que tenga permisos para accesar la Base de Datos, este sirve para que el usuario ejecute la aplicación desde que accesa.

```
# ****************************************************
# =  Profile    Prototipo, para Todos los Usuarios          =
# =  Autor:     Arturo Leon Juarez                          =
# ****************************************************
trap 'clear ;exit' 1 2 3 4 5 6 7 8 10 12 13
mesg n
PATH=$PATH:/usr/lbin:/etc:/usr/contrib/bin:/usr/local/bin:/etc/conf/machine:/usr/include/sys:.
TERM=vt100
export TERM PATH
stty intr '^c' kill '^u' erase '^h'
ORACLE_HOME=/rdbms/oracle
ORACLE_LPPROG="lp"
ORACLE_PAGER=more
ORACLE_LPSTAT=lpstat
ORACLE_LPARGS=""
ORAENV_ASK=NO
ORACLE_SID=A
SI_AMB=/prd/serf
SI_USR=tj30071
SI_PWD=op_serf
SI_IMP=lp_stk1t
PATH=$PATH:/usr/bin:$ORACLE_HOME/bin
ORAKITPATH=.:$ORACLE_HOME/oraterm/admin/resource:$ORACLE_HOME/forms30/admin/res
ource:$ORACLE_HOME/menu5/admin/resource
LANGUAGE=American_America.US7ASCII
export ORACLE_HOME ORACLE_SID ORACLE_LPPROG ORACLE_LPARGS ORAENV_ASK
export ORACLE_PAGER ORACLE_LPSTAT ORAKITPATH LANGUAGE
export SI_AMB SI_TTY SI_CON SI_USR SI_PWD SI_AF SI_IMP SI_MENU PATH
cont=0
if  [ $TERM = vt100 ]
then INV="\033[7m"
   NOR="\033[0m"
fi
```

```
while :
do
clear
echo "\033[2;10H\033#6PETROLEOS MEXICANOS
#            Sistema de Estados de Resultados Financieros              #
#                                                                      #
======================================================================

\n
                Usuario: ${INV}        \b\b\b\b\b\b\b\b\b\b\c"
read usr
if [ "$usr" = "$LOGNAME" ]
  then
stty -echo
echo "${NOR}
          Clave de Acceso: ${INV}        ${NOR}\b\b\b\b\b\b\b\b\b\b\c"
read pwd
stty echo
x=`sqlplus ${usr}/$pwd <<MMM| egrep 'ERROR|Enter'
MMM
`
if  [ -n "$x" ]
then echo "\033[17;25H Usuario y/o Clave de Acceso no validos.
          \033[19;46H Oprima <Retorno> \c"
cont=`expr $cont + 1`
   read nada
 if [ "$cont" = 3 ]
   then
     exit
 fi
else break
fi
fi
echo "${NOR}"
done
clear
if [ -f lock ]
then echo "\033[10;19H Se esta realizando un proceso especial
          \033[12;28H Favor de esperar
       \033[14;28H Oprima <Retorno> \c"
   read nada
exit
fi
clear
export usr pwd
.menu_princ
exit
```

Después de exportar todas la variables necesarias y de validar el usuario, el **.profile** ejecuta el programa **.menu_princ** presentando al usuario el menú principal.

7.6 Ejecución de la aplicación en forma interactiva

Se verá su ejecución paso por paso del menú principal y los submenús, mostrando solo una de las opciones de cada submenús y los mensajes o parámetros que estas pidan.

Estos son los parámetros que al usuario le aparecen al encender su terminal.

Primero el Sistema Operativo le pide su **login** y **password** para que tenga acceso a este.

login : tj30071
Password: serf02

Prosteriormente le aparece la pantalla de Seguridad que es ejecutada por el **.profile** la cual valida que el usuario tenga permisos para entrar a la Base de Datos y ejecuta el programa **.menu_princ**

```
#                  Petroleos Mexicanos                  #
#          Sistema de Estados de Resultados Financieros  #
```

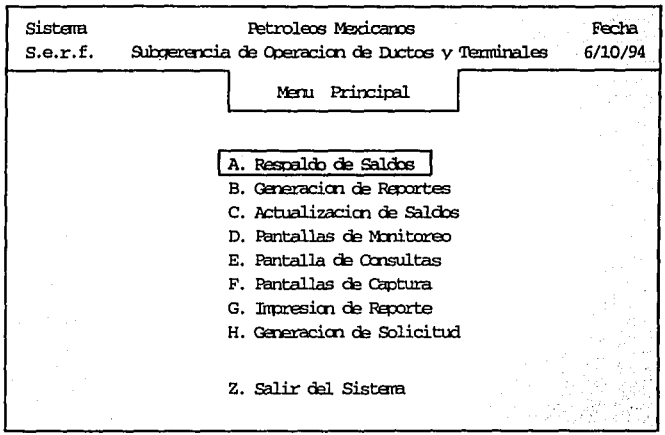
Usuario tj30071
Clave de Acceso op_serf

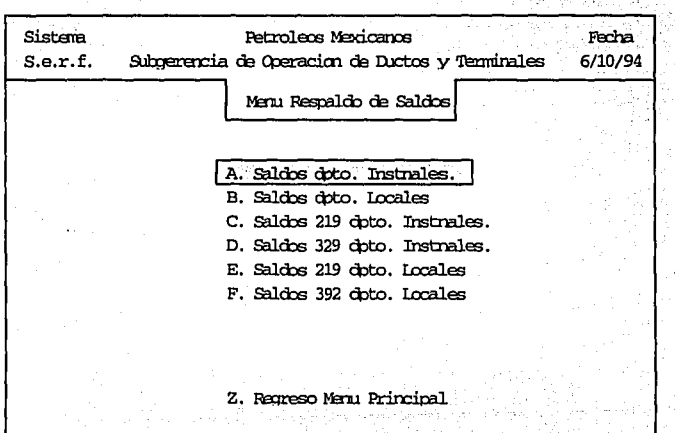
Después de teclear la clave de acceso correcta, el sistema limpia la pantalla y ejecuta la aplicación, de lo contrario manda el mensaje siguiente.

Usuario y/o Clave de Acceso no validos.
Oprima <Retorno>

El usuario tiene tres oportunidades de teclear su clave correcta, de lo contrario el sistema lo mandara al **login.**

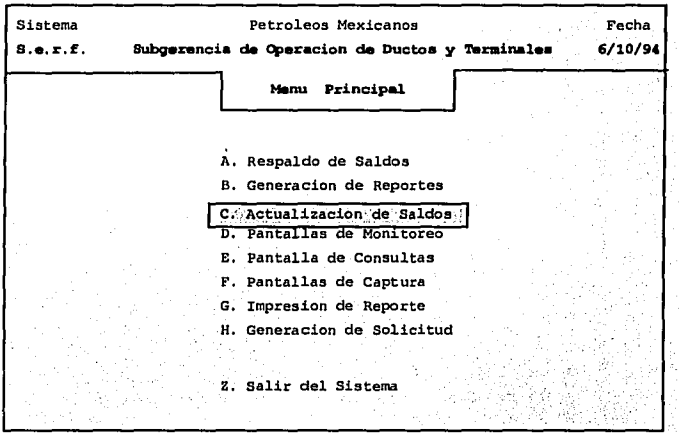
Después de accesar aparecerá el menú siguiente, la operación de cómo apareceran cada uno de los menús y las opciones que se despliegan se verá a continuación. La opción enmarcada es la que se esta seleccionando y la pantalla posterior su ejecución.

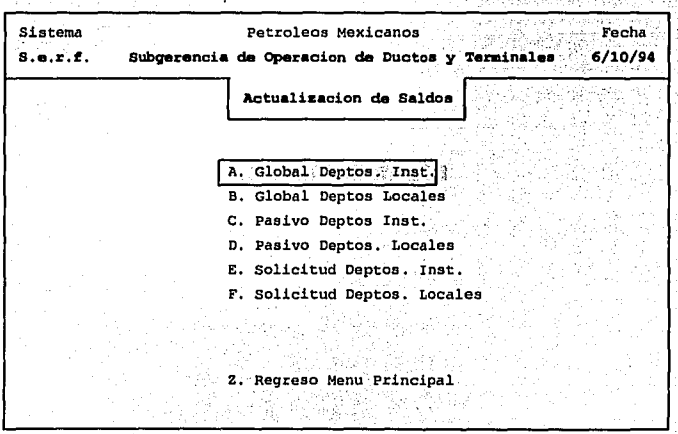

Sistema
S.e.r.f.
Petroleos Mexicanos
Subgerencia de Operacion de Ductos y Terminales
Fecha
6/10/94
Menu Principal
A. Respaldo de Saldos
B. Generacion de Reportes
C. Actualizacion de Saldos
D. Pantallas de Monitoreo
E. Pantalla de Consultas
F. Pantallas de Captura
G. Impresion de Reporte
H. Generacion de Solicitud
Z. Salir del Sistema


Sistema
S.e.r.f.
Petroleos Mexicanos
Subgerencia de Operacion de Ductos y Terminales
Fecha
6/10/94
Menu Respaldo de Saldos
A. Saldos dpto. Instnales.
B. Saldos dpto. Locales
C. Saldos 219 dpto. Instnales.
D. Saldos 329 dpto. Instnales.
E. Saldos 219 dpto. Locales
F. Saldos 392 dpto. Locales
Z. Regreso Menu Principal

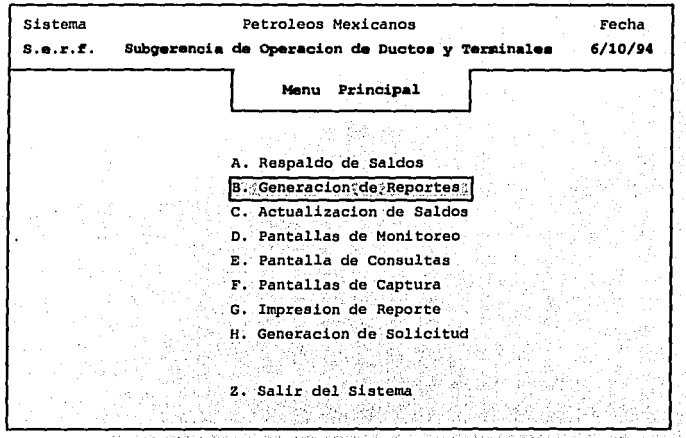
```
Sistema                    Petroleos Mexicanos                    Fecha
S.e.r.f.    Subgerencia de Operacion de Ductos y Terminales      6/10/94

          Sistema de Estados de Resultados Financieros

                        Respaldo de Saldos

                      del mes : 1      de : 94

            Se esta Generando el Respaldo de Saldos

                                  Oprima <Retorn> para Salir :
```

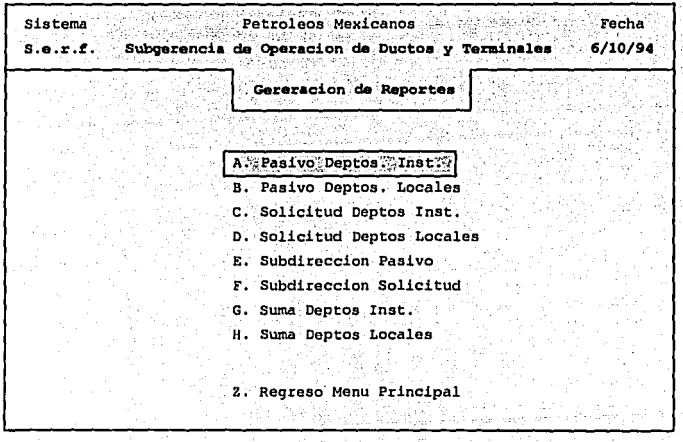
Finaliza la opción para el resplado de saldos.





Finaliza la actualización de saldos


Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Menu Principal
A. Respaldo de Saldos
B. Generacion de Reportes
C. Actualizacion de Saldos
D. Pantallas de Monitoreo
E. Pantalla de Consultas
F. Pantallas de Captura
G. Impresion de Reporte
H. Generacion de Solicitud
Z. Salir del Sistema


Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Gereracion de Reportes
A. Pasivo Deptos. Inst.
B. Pasivo Deptos. Locales
C. Solicitud Deptos Inst.
D. Solicitud Deptos Locales
E. Subdireccion Pasivo
F. Subdireccion Solicitud
G. Suma Deptos Inst.
H. Suma Deptos Locales
Z. Regreso Menu Principal

Sistema	Petroleos Mexicanos	Fecha
S.e.r.f.	Subgerencia de Operacion de Ductos y Terminales	6/10/94

Inicia la Generacion de Reportes de Pasivo

Por Departamento Institucional

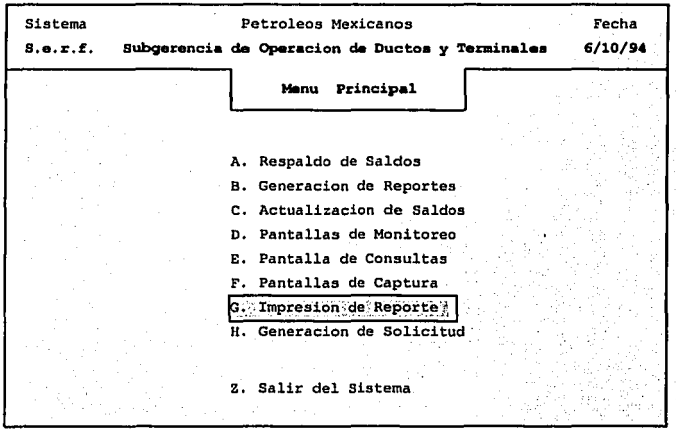
FECHA 06/10/94 HORA 14:28

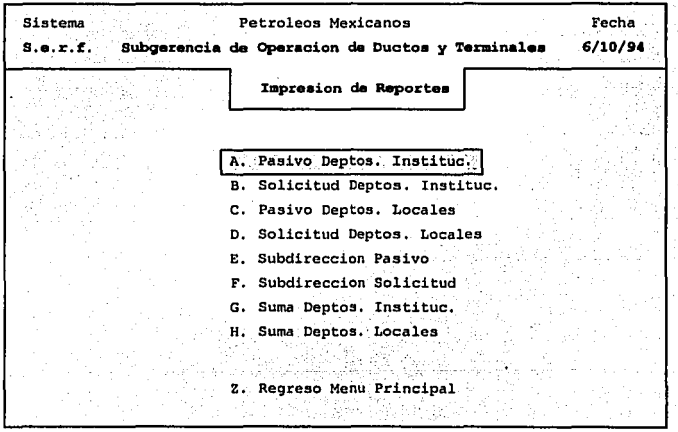
Finaliza la Generacion del Reporte

FECHA 06/10/94 HORA 14:29

Para Continuar < Return >

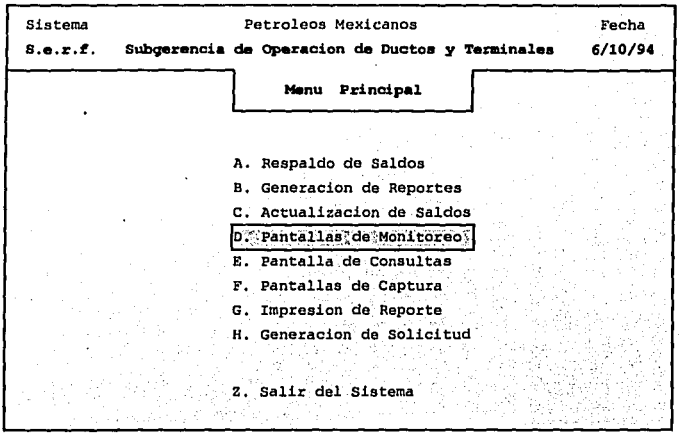
Finaliza la generación de Reportes

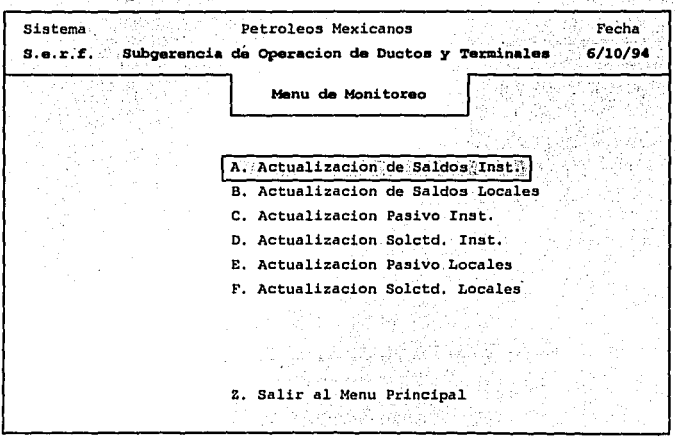

Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Menu Principal
A. Respaldo de Saldos
B. Generacion de Reportes
C. Actualizacion de Saldos
D. Pantallas de Monitoreo
E. Pantalla de Consultas
F. Pantallas de Captura
G. Impresion de Reportes
H. Generacion de Solicitud
Z. Salir del Sistema


Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Impresion de Reportes
A. Pasivo Deptos. Instituc.
B. Solicitud Deptos. Instituc.
C. Pasivo Deptos. Locales
D. Solicitud Deptos. Locales
E. Subdireccion Pasivo
F. Subdireccion Solicitud
G. Suma Deptos. Instituc.
H. Suma Deptos. Locales
Z. Regreso Menu Principal

```
Sistema          Petroleos Mexicanos                          Fecha
S.e.r.f.    Subgerencia de Operacion de Ductos y Terminales   6/10/94


                El nombre del reporte ===> rep219i.lis

                     -> Teclee una Opcion <-

          Impresora Rapida (R) == Impresora Esclava (E)
          Formato Comprimido (C) == Inhibir Impresion (I)
                    Desplegar en Pantalla (D)

                              Digite una Opcion (R,E,C,D o I) ->
```

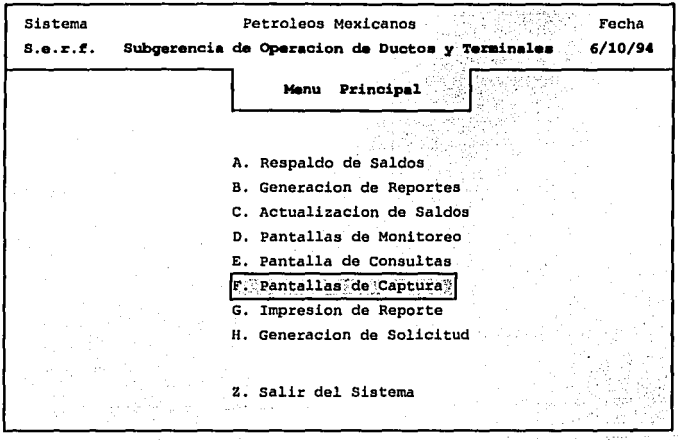
Finaliza la opción de impresión de reportes

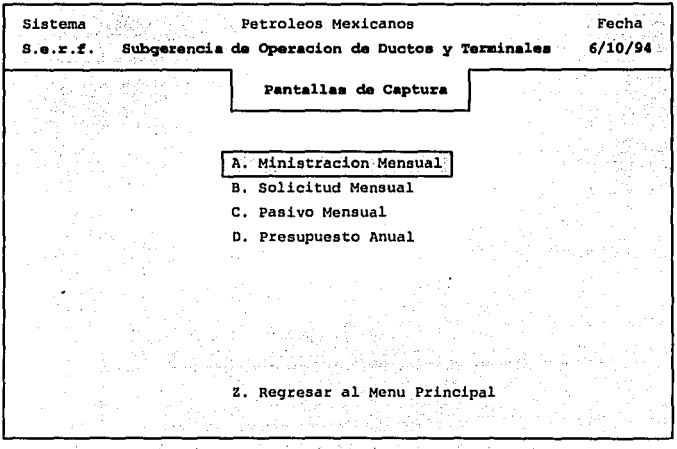

Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Menu Principal
A. Respaldo de Saldos
B. Generacion de Reportes
C. Actualizacion de Saldos
D. Pantallas de Monitoreo
E. Pantalla de Consultas
F. Pantallas de Captura
G. Impresion de Reporte
H. Generacion de Solicitud
Z. Salir del Sistema

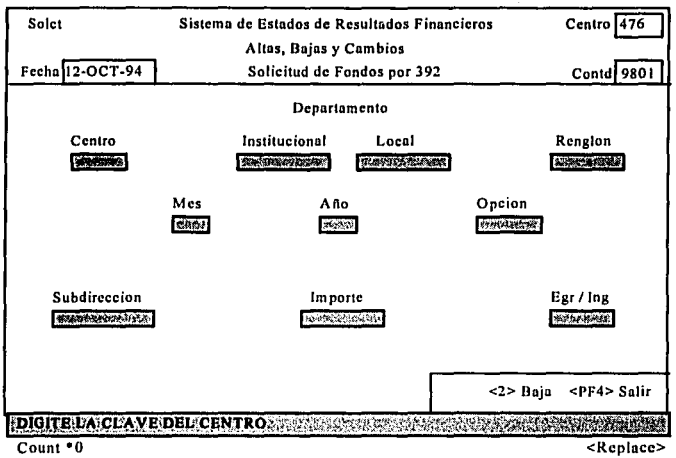

Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Menu de Monitoreo
A. Actualizacion de Saldos Inst.
B. Actualizacion de Saldos Locales
C. Actualizacion Pasivo Inst.
D. Actualizacion Solctd. Inst.
E. Actualizacion Pasivo Locales
F. Actualizacion Solctd. Locales
Z. Salir al Menu Principal

```
Sistema                    Petroleos Mexicanos                    Fecha
S.e.r.f.      Subgerencia de Operacion de Ductos y Terminales     6/10/94


                El nombre del reporte ===> rep219i.lis

                        -> Teclee una Opcion <-

          Impresora Rapida (R) == Impresora Esclava (E)
          Formato Comprimido (C) == Inhibir Impresion (I)
                    Desplegar en Pantalla (D)

                              Digite una Opcion (R,E,C,D o I) ->
```

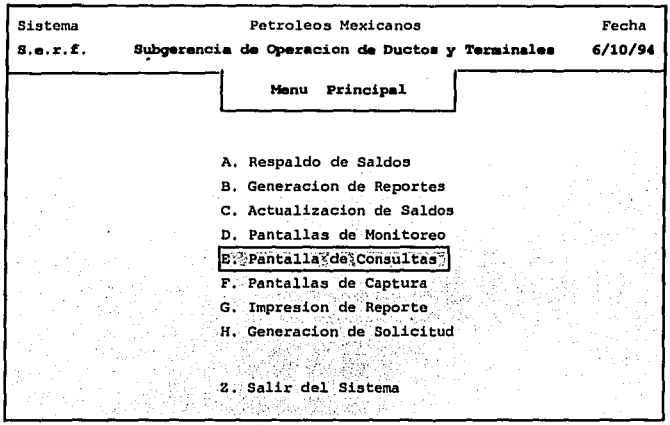
Finaliza la opción de actualización de saldos.

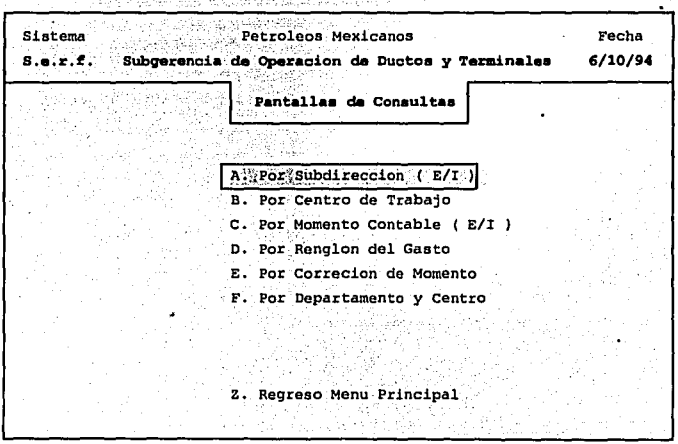

Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Menu Principal
A. Respaldo de Saldos
B. Generacion de Reportes
C. Actualizacion de Saldos
D. Pantallas de Monitoreo
E. Pantalla de Consultas
F. Pantallas de Captura
G. Impresion de Reporte
H. Generacion de Solicitud
Z. Salir del Sistema

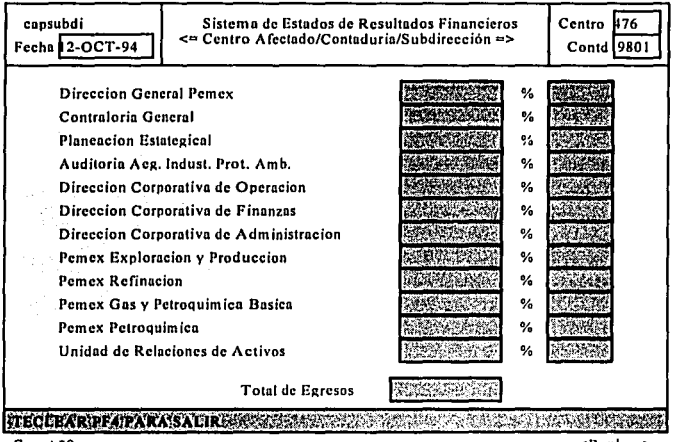

Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Pantallas de Captura
A. Ministracion Mensual
B. Solicitud Mensual
C. Pasivo Mensual
D. Presupuesto Anual
Z. Regresar al Menu Principal



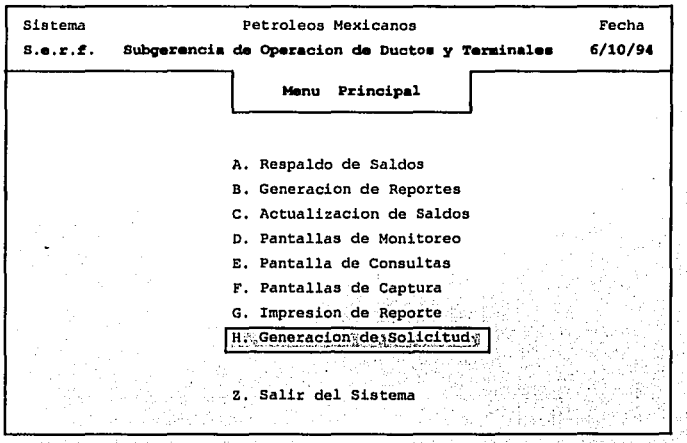
Finaliza la opción de pantallas de captura.

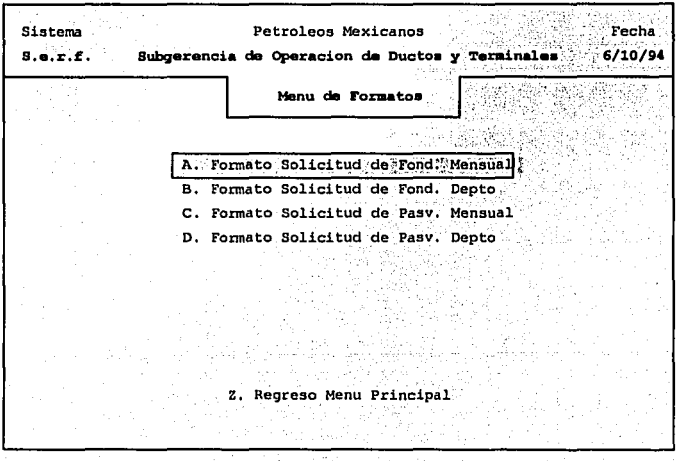

Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Menu Principal
A. Respaldo de Saldos
B. Generacion de Reportes
C. Actualizacion de Saldos
D. Pantallas de Monitoreo
E. Pantalla de Consultas
F. Pantallas de Captura
G. Impresion de Reporte
H. Generacion de Solicitud
Z. Salir del Sistema


Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Pantallas de Consultas
A. Por Subdireccion (E/I)
B. Por Centro de Trabajo
C. Por Momento Contable (E/I)
D. Por Renglon del Gasto
E. Por Correcion de Momento
F. Por Departamento y Centro
Z. Regreso Menu Principal



Finaliza la opción de pantallas de consultas

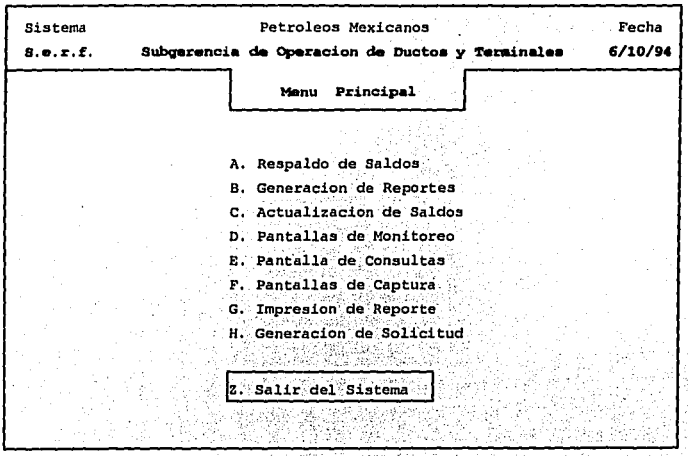

Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Menu Principal
A. Respaldo de Saldos
B. Generacion de Reportes
C. Actualizacion de Saldos
D. Pantallas de Monitoreo
E. Pantalla de Consultas
F. Pantallas de Captura
G. Impresion de Reporte
H. Generacion de Solicitud
Z. Salir del Sistema


Sistema
Petroleos Mexicanos
Fecha
S.e.r.f.
Subgerencia de Operacion de Ductos y Terminales
6/10/94
Menu de Formatos
A. Formato Solicitud de Fond. Mensual
B. Formato Solicitud de Fond. Depto
C. Formato Solicitud de Pasv. Mensual
D. Formato Solicitud de Pasv. Depto
Z. Regreso Menu Principal

```
Sistema          Petroleos Mexicanos                          Fecha
S.e.r.f.   Subgerencia de Operacion de Ductos y Terminales    6/10/94

Generando el Formato para la Solicitud de Fondos solc392.exe

        Solicitando los Parametros del Reporte


        Mes de la Solicitud de Fondos <DD> =>
        Teclear el Centro de Trabajo <DDD> =>


                                  Para Continuar Oprima <Return>
```

```
Sistema          Petroleos Mexicanos                          Fecha
S.e.r.f.   Subgerencia de Operacion de Ductos y Terminales    6/10/94


        El nombre del reporte ===> rep219i.lis

              -> Teclee una Opcion <-

      Impresora Rapida (R) == Impresora Esclava (E)
      Formato Comprimido (C) == Inhibir Impresion (I)
              Desplegar en Pantalla (D)

                        Digite una Opcion (R,E,C,D o I) ->
```

Finaliza la opción de generación de la forma 392



La opción de salir del Sistema de Estados de Resultados Finacieros (S:E:R:F) nos lleva de nuevo al login.

login :

CONCLUSIONES

El presente trabajo tiene como objetivo principal mostrar el desarrollo de la aplicación y la funcionalidad de está, la cual apoya al Departamento de presupuestos en el manejo de la información de una manera más veraz y oportuna. El desarrollo de un sistema que trabaje de una manera interactiva ayuda al Centro de Trabajo y por lo tanto a la empresa, para ajustar los gastos que se realizan por Departamento y Sector, teniendo un panorama de cada uno de estos gastos. Con el apoyo del sistema la información se obtiene de una manera más sencilla y con una mayor oportunidad.

Si se toma en cuenta que anteriormente el ajuste presupuestal se realizaba de una manera manual, y apoyados en prorrateos para la asignación del presupuesto a cada uno de los Sectores y Departamentos que estos contienen, y que para realizar algún ajuste este no se realizaba hasta tener en sus manos las facturas o comprobantes de gastos, esto hacía que se tuviera un retraso de un mes y en ocasiones mayor. La consecuencia de todo lo mencionado nos ocasionaba una mala distribución de las ministraciones y pérdidas que se reflejaban en ampliaciones continuas del presupuesto, esto aunado a la necesidad de contar con más personal y tiempo en la revisión de la documentación.

El presente trabajo no tiene como objetivo dar un curso de administración de ORACLE o UNIX, simplemente nos muestra un panorama general de cada uno de estos, que son la plataforma de desarrollo del sistema y por ende al hablar de estos y de las herramientas con las que realmente cuentan y que nos sirven de apoyo para el desarrollo del sistema que es realmente el objetivo. Estas solo se verán de una manera general ya que al hablar de cada una de las herramientas de una manera más detallada se podría realizar un tema de tesis de cada una de estás, lo cual nos desviaría de nuestro objetivo principal.

Por último, la finalidad principal es mostrar como esta desarrollada la aplicación a nivel de programación y mostrar sus funcionalidad y operación de esta de una manera interactiva. Los menus que son presentados y programas con los que cuenta el sistema ó aplicación. Es en gran parte el contenido de la tesis. No se trata de mostrar una metodología de desarrollo para esta, aunque todo sistema se base en alguno; para nuestro caso el sistema se apoyo en la metodología de James Martin.

Finalizando se resumiría como enfoque de la tesis tres fases importantes:

1. Mostrar las causa de porqué la necesidad de desarrollar la aplicación.
2. Dar a conocer un panorama de la plataforma de desarrollo de la aplicación.
3. El desarrollo de los programas y lo que nos entrega cada uno de estos a su salida.

BIBLIOGRAFIA

Titulo : Administer the ORACLE Database y Version 6
Editorial : Oracle Corporation, 1992, 1993
Autor : Oracle Corporation
Revisión : February 1993

Titulo : SQL*Forms Designer's Reference Version 3
Editorial : Oracle Corporation, 1989, 1991
Autor : Oracle Corporation
Revisión : Versión 3.0 1991

Titulo : Pro*C User's Guide Version 1
Editorial : Oracle Corporation, 1985, 1987
Autor : Oracle Corporation
Revisión : April 10th, 1987

Titulo : Database Administrator's Guide Version 6
Editorial : Oracle Corporation, 1987, 1990
Autor : Oracle Corporation
Revisión : October 1990

Titulo : SQL Lenguage Reference Manual Version 6.0
Editorial : Oracle Corporation, 1988, 1990
Autor : Oracle Corporation
Revisión : February 1990

Titulo : Installing and Updating HP-UX
Editorial : Hewlett Packard 1991
Autor : Hewlett Packard
Revisión : Second Edition January 1991

Titulo : HP C/HP-UX Reference Manual
Editorial : Hewlett Packard 1991
Autor : Hewlett Packard
Revisión : Second Edition January 1991

Titulo : HP-UX User's Guide
Editorial : Hewlett Packard 1988
Autor : Hewlett Packard
Revisión : Second Edition February 1988

Titulo : A Beginner's Guide to HP-UX
Editorial : Hewlett Packard 1990, 1991
Autor : Hewlett Packard
Revisión : Second Edition January 1991

Titulos :	Manual de Bolsillo
Editorial :	Osborne McGraw-Hill
Autor :	Herbert Schildt
Revisión :	Segunda Edición 1990

Titulo :	El Lenguaje de Programación
Editorial :	Printice Hall
Autor :	Brian W. Kernighan Dennis M. Ritchie
Revisión :	Segunda Edición 1991