98

2EJ

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

GENERADOR DE REPORTES

TESIS PROFESIONAL

Que para obtener el Título de
INGENIERO EN COMPUTACION
p r e s e n t a:

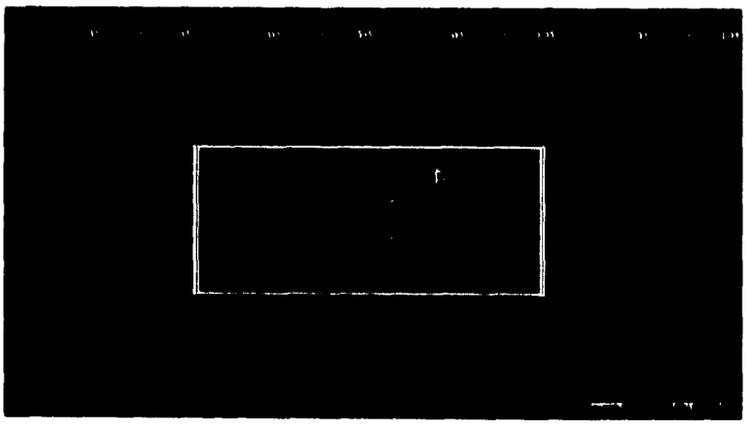
EUNICE ROSALES CONTRERAS

DIRECTOR DE TESIS:
ING. ADOLFO MILLAN NAJERA

MEXICO, D.F. 1995

FALLA DE ORIGEN

GENERADOR DE REPORTES



Pantalla principal del "Generador de Reportes"

DEDICATORIAS

a Ian
por compartir cada área de mi vida

a mis padres
por su ejemplo, su esfuerzo constante y apoyo incondicional.

al Ingeniero Adolfo Millán
por su guía y dedicación en la supervisión de esta tesis

a mis profesores y compañeros
por su experiencia, conocimiento y amistad

ÍNDICE TEMÁTICO

PRÓLOGO

Cada día el mercado de paquetes de software disponibles para PC's, que corren bajo el sistema operativo MS-DOS, se multiplica y gracias a ello los usuarios finales cuentan para cada necesidad de procesamiento de información con un paquete especializado. Es por ello que los proveedores de estos paquetes se han dado cuenta de la gran importancia de proporcionar las herramientas que faciliten a los usuarios la interacción con otros paquetes y ofrezcan así versatilidad y un ambiente de interoperabilidad a los usuarios en el uso de sus productos. Con este tipo de herramientas, un programador, al desarrollar una aplicación, podrá optar en ciertos módulos por utilizar un lenguaje de alto nivel, el cual permita interactuar al mismo tiempo con la base de datos y con un paquete especializado, para obtener los beneficios de ambos y lograr así aplicaciones más atractivas para los usuarios finales.

El objetivo principal de la tesis titulada "Generador de Reportes" es crear una herramienta que genere en forma automatizada el código, en un lenguaje de alto nivel, de cualquier reporte diseñado por el usuario mediante una interfaz sencilla y completa. Lo cual facilite a los usuarios, la interoperabilidad entre una base de datos comercial y otros paquetes de software o hardware bajo el sistema operativo MS-DOS.

Las Características de este Generador de Reportes son:

- Proporcionar a los usuarios de los manejadores de bases de datos más populares por su costo, una herramienta que sea tan fácil de usar como las herramientas que el manejador de archivos proporciona. Esto se logra a través de una interfaz sencilla con el usuario; se diseña en pantalla el formato del reporte y mediante una opción del menú se selecciona la generación de código.

- Diseño modular, lo cual permite añadir nuevos módulos que permitan el acceso y generación de código para cualquier base de datos comercial.

- Poseer las funciones características de los reporteadores de los manejadores de bases de datos comerciales (definición de grupos, ligar tablas, variables de resumen, etc.).

- Generar automáticamente código en un lenguaje de alto nivel para poder incluir las funciones que el reporteador del manejador de bases de datos no posee.

Se pretende que esta herramienta además de servir a los usuarios finales, sea una gran ayuda a los desarrolladores de sistemas, que brinde una buena base para la elaboración de reportes más complejos y para la interoperabilidad entre diferentes paquetes de software.

En el *Capítulo I* se presenta una introducción al "Generador de Reportes" y un resumen de las expectativas de los usuarios al contar con una herramienta de este tipo.

En el *Capítulo II* se presenta el resultado del análisis funcional realizado entre diferentes bases de datos comerciales para determinar la funcionalidad del "Generador de Reportes", y en en base a este análisis previo se presenta el modelo lógico del "Generador de Reportes" y en general la especificación estructurada.

En los *Capítulos III, IV y V* se presentan las diferentes fases de desarrollo del "Generador de Reportes" a partir del Diseño.

En el *Capítulo VI* se presentan las conclusiones acerca de esta tesis y en general del desarrollo de sistemas.

En esta primera versión del sistema "Generador de Reportes" que se desarrolló como resultado de esta tesis, se ha programado la interfaz que permite tener el acceso a la información almacenada en bases de datos compatibles con "Paradox" y la generación de código en lenguaje de programación "C". En versiones posteriores, el diseño del sistema permitirá añadir fácilmente nuevos módulos que accesen información de otras bases de datos comerciales y que permitan la generación de código en otros lenguajes de programación, utilizando la misma interfaz hacia el usuario final.

Por último debo mencionar que junto con el objetivo de desarrollar un Generador de Reportes como herramienta complementaria a los manejadores de bases de datos comerciales, es mi propósito el realizar este tema de tesis siguiendo los pasos que marca la Ingeniería de Programación y en general aplicando los conocimientos que adquirí al ser una alumna de la Facultad de Ingeniería, para que este trabajo posea las bases y documentación necesaria que respaldan a un sistema planeado.

Eunice Rosales Contreras

CAPÍTULO I

INTRODUCCIÓN

Al desarrollar sistemas de cómputo bajo el Sistema Operativo MS-DOS en sistemas PC , es común, en el área de Sistemas, realizar una interfaz entre un manejador de bases de datos comercial con un lenguaje de programación de alto nivel. Esto no quiere decir que se haga una selección entre un lenguaje de alto nivel y el lenguaje propio del manejador de bases de datos, sino que cada uno de ellos tiene sus propias esferas de funcionalidad y hay que conocer las principales características de cada uno para realizar una división apropiada de tareas entre ellos.

En general los lenguajes propios del manejador de bases de datos están estrechamente ligados al medio ambiente del manejador, por lo que proporcionan un manejo eficiente de la base de datos y permiten el uso de herramientas muy poderosas como son los generadores de consultas, formas, etc. Tareas que tomarían posiblemente años en desarrollar en un lenguaje de alto nivel; sin embargo, al obtener estas ventajas se sacrifica flexibilidad y control.

El uso de un lenguaje de alto nivel en combinación con un manejador de bases de datos permite:

- Prescindir del manejador de bases de datos y accesar las bases de datos mediante el lenguaje de alto nivel.
- Reducir el tiempo de respuesta y tamaño de las aplicaciones, ya que el correr una aplicación utilizando todos los beneficios de un manejador de bases de datos ocasiona una gran demanda de memoria en la estación de trabajo.
- Un alto nivel de control.
- Lograr una mayor versatilidad en la interfaz con el usuario.
- Realizar importación de datos de otros paquetes o sistemas.
- Accesar el hardware y el sistema operativo.
- Hacer uso de rutinas ya desarrolladas.
- Realizar funciones que el propio lenguaje del manejador de bases de datos no posee.

El desarrollo de los sistemas que dan mantenimiento a la base de datos, se realizan casi siempre con el lenguaje propio de la base de datos y los programas que realizan la explotación de información almacenada en la base de datos son los que más frecuentemente se realizan en un lenguaje de alto nivel que interactue con la base de datos, básicamente porque:

- Después de la liberación de cualquier sistema, el mantenimiento se centra básicamente en la generación de reportes que hagan uso de la información.

- En la elaboración de reportes, utilizando los reporteadores que incluyen los paquetes comerciales, no siempre se pueden realizar todos los reportes que son necesarios en un sistema.

- Es común la necesidad de generar reportes que combinan información de diferentes paquetes, sean estos otros manejadores de bases de datos, paquetes estadísticos, hojas de cálculo, etc.

Estas son las principales razones por las que se requiere de una herramienta que permita diseñar en forma rápida un reporte y que sea la misma herramienta la que genere el código del reporte en un lenguaje de alto nivel interactuando con la base de datos, lo cual permita acelerar la creación de reportes más complicados modificando el código generado, incluir funciones que manejen información de otros paquetes de software y hacer uso de las ventajas que representa el uso de un lenguaje de alto nivel.

CAPÍTULO II

ANÁLISIS Y ESPECIFICACIÓN ESTRUCTURADA

INTRODUCCIÓN

Para realizar el análisis del "Generador de Reportes" se siguieron los siguientes pasos los cuales se presentan en este capítulo.

- Revisión de la metodología del análisis estructurado.
- Análisis funcional de tres generadores de reportes de bases de datos comerciales, con el fin de determinar la funcionalidad del "Generador de Reportes".
- Elaboración del modelo lógico para el módulo *Editor* del "Generador de Reportes", como resultado del análisis funcional.
- Análisis de la generación de código.
- Elaboración del modelo lógico para el módulo Generador de Código del "Generador de Reportes", como resultado del análisis del punto anterior.
- Presentación de la Especificación Estructurada del "Generador de Reportes"

II.1. ANÁLISIS ESTRUCTURADO

II.1.1 INTRODUCCIÓN

El análisis de requerimientos puede dividirse en cuatro áreas:

1) Reconocimiento del problema
2) Evaluación del problema y síntesis de la solución
3) Especificación de requerimientos para definir las características y atributos del software y
4) Revisión.

El producto principal del análisis es el documento de especificaciones o documento de objetivos.

Para realizar el análisis de requerimientos existen varias metodologías de análisis. Estas son un conjunto de heurísticas para subdividir el problema y definir una forma de representación. Dichas metodologías facilitan al analista la aplicación de los principios fundamentales del análisis de una manera sistemática para realizar la construcción de "una descripción precisa e independiente" del elemento software de un sistema basado en computadora.

Todos los métodos de análisis están relacionados por un conjunto de principios fundamentales. Las características comunes son:

1) Mecanismos para el análisis del dominio de la información
2) Método de representación funcional del problema
3) Definición de interfaces
4) Mecanismos para subdividir el problema de forma que se descubran los detalles de una manera progresiva (o jerárquica)
5) Soporte de la abstracción
6) Representación de los modelos físicos y lógicos

La mayoría de los métodos de análisis de requerimientos son conducidos por la información. Esto es, el método suministra un mecanismo para representar el dominio de la información. De esta representación, se deriva la función y se desarrollan otras características de los programas. El dominio de la información contiene tres

representaciones diferentes de los datos que se procesan por los programas de computadoras: flujo de datos, contenido de los datos y estructura de datos.

II.1.2 OBJETIVOS DEL ANÁLISIS ESTRUCTURADO

Los métodos de flujo de datos se enfocan hacia el establecimiento de cómo los datos pasan a través del dominio de la información, y suministran procedimientos para aislar el contenido de los datos y las funciones de procesamiento.

De los métodos de flujo de datos se eligió para el desarrollo del "Generador de Reportes" el Análisis Estructurado. Este método hace uso de las siguientes herramientas:

- Diagrama de Flujo de Datos (DFD)
- Diccionario de Datos (DD)
- Tablas de decisión, árboles de decisión, diagramas de acción, etc.
- Normalización
- Miniespecificaciones

para construir el documento de objetivos, o especificaciones estructuradas, que son el principal producto del Análisis.

La especificación estructurada es un modelo en papel del nuevo sistema. El modelo se construye usando DFD(s) que muestran la partición y las interfaces. El diccionario de datos y la descripciones de transformación (mini-especificaciones) que documentan el interior de los procesos.

II.1.3 PROBLEMAS DEL ANÁLISIS

Un análisis deficiente puede ocasionar resultados no deseados, como lo son:

- Fallas frecuentes en la operación posterior del sistema.
- Excesivo mantenimiento de las aplicaciones.
- Incremento en su costo.
- Concepción equivocada de los requerimientos del usuario.

La metodología del Análisis Estructurado nos permite minimizar la probabilidad de un error crítico que pueda repercutir gravemente en el resto del proyecto. Para ello, se utilizan herramientas bien definidas como el Diagrama de Flujo de Datos, las Miniespecificaciones y el Diccionario de Datos. Estas herramientas eliminan los problemas de comunicación más comunes entre el usuario y el grupo de desarrollo, obteniendo productos fácilmente mantenibles, particionables y gráficos.

II.1.4 ETAPAS DEL ANÁLISIS ESTRUCTURADO

El análisis estructurado consta de las siguientes etapas, que son:

- Estudio del sistema físico actual
- Derivación de un modelo lógico
- Empacar las especificaciones estructuradas

En el caso del Generador de Reportes el sistema físico actual se enfoca a los Generadores de Reportes que pertenecen a las Bases de Datos Comerciales existentes y a la serie de reglas que sigue un programador, basándose en una gramática de un lenguaje de programación, para generar el código de un reporte.

Debido a que no se cuenta con los programas o documentación técnica relacionada con el diseño y programación de los manejadores de bases de datos a analizar, sólo se realizará un análisis funcional que sirva de base para proponer un nuevo diseño lógico de la interfaz que permita al usuario definir el reporte. Por otra parte se realizará el análisis de la serie de reglas que sigue un programador, basándose en una gramática de un lenguaje de programación, para generar el código de un reporte, lo cual permite proponer un modelo lógico. El diagrama de flujo de datos, el diccionario de datos y las miniespecificaciones, se crearán junto con la propuesta del modelo lógico del "Generador de Reportes".

II.2 OBJETIVOS Y ALCANCES DEL SISTEMA

El objetivo general del Generador de Reportes será:

- Crear un Generador de Reportes, utilizando los principios de la Ingeniería de Software, que genere automáticamente código en lenguaje de programación "C", el cual sea una herramienta que facilite la interoperabilidad entre una base de datos comercial y otros paquetes de software o hardware bajo el sistema operativo MS-DOS, en lo que se refiere a la explotación de la información.

Los objetivos particulares que deberá cumplir el Generador de Reportes como un producto de software son:

- Ser una herramienta fácil de usar que complemente los beneficios que ofrecen las bases de datos comerciales.
- Proporcionar las principales funciones que ofrecen los generadores de reportes comerciales.
- Generar el código del reporte en un lenguaje de programación que facilite la portabilidad del mismo y permita la programación de funciones para la integración de información de otros paquetes de software.
- El código generado del reporte debe ser estructurado, para facilitar al programador su modificación.
- Manejar información almacenada en una Base de Datos comercial.

II.3 ANÁLISIS FUNCIONAL DE TRES GENERADORES DE REPORTES DE BASES DE DATOS COMERCIALES

II.3.1 CONTEXTO DEL ESTUDIO

Para obtener las funciones características de los generadores de reportes que corren bajo el sistema operativo MS-DOS se realizó el análisis de tres de los manejadores de bases de datos comerciales más frecuentemente usados: Paradox 3.5, FoxPro Lan 2.0 y DBase 1.1.

El objetivo de este análisis fue:

- Para cada manejador de bases de datos, realizar un registro de la funcionalidad e interfaz con el usuario con que cuenta cada generador de reportes, para incluirlos en el nuevo desarrollo. El registro de la funcionalidad se tomará como parte de la lista de requerimientos

II.3.2. PARADOX VERSIÓN 3.5

II.3.2.1 Características Generales de PARADOX 3.5

Paradox es un manejador de base de datos relacional que puede ser usado por un solo usuario en una PC o por varios usuarios en una red. Las principales características de este manejador son:

- **Tablas**. Paradox es la única base de datos, entre las analizadas, donde las validaciones y el índice primario quedan asociadas a la tabla y son llamadas automáticamente al hacer uso de ella. Esta característica ayuda mucho al programador y evita inconsistencias en la definición de validaciones.

- **Formas**. Se pueden crear hasta 14 formas asociadas a cada tabla. Cada forma puede incluir varias tablas, haciendo diferentes relaciones: 1 a 1 y 1 a muchos. La definición de las relaciones en la creación de las formas permite que Paradox maneje automáticamente la integridad referencial de los datos al hacer uso de la forma.

- **Consultas**. El "Querie By Example (QBE)" es una poderosa herramienta para consultar una o más tablas, la cual de una manera sencilla permite:
  - Buscar o seleccionar información de una tabla.
  - Combinar información de más de una tabla.
  - Realizar cálculos de los datos en una tabla.
  - Insertar y borrar datos en una tabla.
  - Cambiar datos seleccionados de una tabla.
  - Definir grupos y conjuntos de información en los cuales se realicen cálculos y comparaciones.
  - Los "queries" se pueden almacenar y usar posteriormente a nivel de comando o incluir en el desarrollo de una aplicación.

- **Reportes**. Se pueden crear hasta 14 reportes asociados a una tabla, o reportes que incluyan varias tablas. Se tiene la facilidad para crear etiquetas y cartas.

- **Gráficas**. La información existente en las tablas o en tablas que sean el resultado de un "querie" se puede graficar en los siguientes tipos: gráfica de barra, gráfica de barra con más de una serie, gráficas de barra apiladas, gráficas de barra en 3ª dimensión, gráficas de línea, gráficas de puntos, gráficas de área, gráficas XY, y gráficas tipo pastel.

- **Generador de Aplicaciones PAL** (Paradox Personal Programmer).

- **Lenguaje de programación PAL** (Paradox Application Language). Este lenguaje permite crear aplicaciones de acuerdo a necesidades específicas y hacer uso de todas las características de Paradox.

II.3.2.2 Características del Generador de Reportes

Para usar el Generador de Reportes de Paradox y explotar todas sus opciones, se deben tener claros los siguientes conceptos: tablas, llaves de acceso a las tablas, relaciones, grupos, funciones aritméticas del PAL y en general las convenciones de teclas y manejo del ambiente de Paradox.

El Generador de Reportes de Paradox permite:

- **Crear reportes en forma tabular**
- **Crear reportes en formato libre**, lo cual también permite la creación de etiquetas y cartas.
- **Crear grupos.-** Esta opción permite organizar los datos para una mejor claridad y comprensión en su lectura. Se pueden establecer hasta 16 grupos para cada reporte, basándose en los valores de los campos o en campos calculados.
- **Incluir campos de tablas relacionadas.-** Como máximo, se puede relacionar una tabla por cada campo que exista en la tabla maestra (tabla para la cual se definió el reporte).
- **Insertar texto**.
- **Cambiar el nombre y formato de columnas**.
- **Imprimir etiquetas**.
- **Formatear campos y páginas**.
- **Operadores aritméticos** para resumen de columnas (SUM, AVERAGE, COUNT, HIGH y LOW). El resultado se puede definir para un grupo (subtotales) o para todo el reporte (totales).
- **Definir campos calculados y usar funciones del** lenguaje procedural de Paradox (ABS, ASC, LEN, SUM, etc.)
- **Direccionar el reporte** generado a impresora, pantalla o a un archivo.

II.3.2.3 Manejo del Generador de Reportes

A continuación se describe, en forma muy breve, el uso del generador de reportes.

Al seleccionar la opción de reportes se comienza a interactuar con una serie de menús que son los que manejan al generador de reportes. Por medio de estos menús se selecciona la tabla maestra, el reporte y si es reporte tabular o con formato libre.

Estando dentro del generador de reportes la idea principal es: que el diseño en la pantalla sea casi lo mismo a lo obtenido en el reporte. En la pantalla se presenta una regla horizontal que indica la columna, y varias divisiones verticales que indican en qué área se encuentra uno: encabezado del reporte, encabezado de un grupo, totales de un grupo, detalle, totales del reporte.

El movimiento dentro de esta pantalla es similar al de un editor. Cuando se requiere una función del generador de reportes se oprime [F10] y se comienza a interactuar con el menú. Para insertar un campo se pueden seleccionar los campos de la tabla maestra, campos de las tablas ligadas o bien crear un campo calculado, mediante el menú, y después en el editor se puede mover ese campo hasta la posición deseada.

Si se entienden los conceptos de ligar tablas, creación de grupos, creación de campos calculados y las reglas que define Paradox para su uso; la creación y uso del generador de reportes resulta bastante sencilla.

II.3.2.4 Comentarios

Como usuario de Paradox puedo decir que el aprender a manejar el ambiente de este paquete y sus utilerías resulta muy sencillo, debido a la simplicidad de sus pantallas y al fácil manejo de sus utilerías. Al manejar Paradox uno experimenta los beneficios de un auténtico Administrador de Bases de Datos Relacional, a través de la definición de formas con varias tablas asociadas (que mantienen automáticamente la integridad de los datos), haciendo uso del generador de consultas (Query By Example) y del generador de reportes.

A mi parecer, esta es la base de datos perfecta para el usuario final que no desea programar y cuyo principal interés es el análisis de datos. Además de ser la única que, como parte del paquete original, permite elaborar gráficas directamente de la base de datos.

Por otra parte, la programación en PAL (Paradox Application Language) para desarrollar alguna aplicación, no resulta tan sencilla, ya que la lógica que se utiliza es diferente a lo convencional. Para poder programar en PAL es sumamente necesario conocer a fondo Paradox, sus menús, funciones y comandos, y cómo funcionan éstos en los diferentes estados de Paradox (pantallas, tablas, menús), ya que PAL es un lenguaje que trata de automatizar el medio ambiente de Paradox.

Un artículo muy acertadamente dice[1]: "lo que hace difícil la programación en PAL, es que semeja el enviar comandos a un robot que esta sentado en el teclado corriendo Paradox".

Una de las utilerías más poderosa de programación en PAL, llamada DOWAIT() permite mayor flexibilidad en la programación, mas sin embargo es la única parte que no está lo suficientemente documentada y sólo tiene una aplicación de ejemplo.

Con respecto a la programación en PAL se puede decir que comprender el estilo y programar una aplicación con la certeza de lo que realizará, toma tiempo.

No obstante, de los Administradores de Bases de Datos relacionales que he manejado en PC, este paquete ha sido el más poderoso para realizar análisis de datos sin tener que programar.

Además de las características mencionadas Paradox cuenta con otra herramienta llamada Paradox Engine, la cual permite el uso de la Base de Datos por medio del lenguaje de programación C o Pascal.

Por otra parte Borland International adquirió a finales de 1991 la empresa Ashton-Tate y su producto DBASE, anunciando que la tendencia en un futuro es que exista una fusión de tecnologías y los archivos de Paradox o DBASE se puedan manejar indistintamente con cualquiera de los dos paquetes. Esta situación da a Paradox una perspectiva favorable de crecimiento a futuro.

II.3.3 FOXPRO/LAN VERSIÓN 2.0

II.3.3.1 Características Generales de FoxPro/Lan 2.0

FoxPro es un manejador de Base de Datos relacional que ha evolucionado a partir del paquete FoxBase. FoxPro tiene 200 comandos más que FoxBASE+ y 140 comandos más que DBASEIV; además los programas de FoxBASE+ y dBASE III PLUS pueden correr sin ningún cambio bajo FoxPro. Este paquete puede ser utilizado por un sólo usuario en una PC o por varios usuarios en una red. Las principales características de este manejador son:

[1] "DOS Data Bases at work" por Rick Grehan y Stanford Dielhl. revista BYTE, enero de 1992

- **Tablas**. En Foxpro además de poder definir los tipos de campos más comunes (caracter, fecha, numéricos, float y lógicos), se pueden definir campos tipo Memo. Al llamar una tabla no se asocia ningún índice, el usuario debe indicar qué índice desea usar o crearlo si no existe.

- **Formas y Generador de Código**. FoxPro es el único paquete de los analizados, que permite la definición de ventanas al crear formas.

  Además de poder definir campos de una o más bases de datos, se pueden definir campos de control del programa, de consulta y de selección; como son Push Buttons, Radio Button, Check Box, Popups y Listas. Este tipo de campos tiene como fin el mejorar la interfaz con el usuario final y dar ventajas al uso del "mouse".

  Las validaciones para la captura de los campos se especifican en el diseño de pantallas, lo cual, a mi parecer, puede causar una inconsistencia al definir diferentes validaciones en diferentes pantallas que usen la misma tabla.

  En cuanto al desarrollo de sistemas, este generador es la herramienta más fuerte, ya que permite la generación automática del código que maneja la pantalla. En la definición de la forma a cada campo, ventana o pantalla se le pueden asociar una serie de funciones que permitan el desarrollo total de una aplicación.

- **Generador de Menús**. Permite diseñar menús para manejo de las aplicaciones.

- **Lenguaje SQL y Query By Example**. FoxPro soporta el comando SQL SELECT, el cual permite desarrollar y ejecutar consultas a una o más bases de datos. El RQBE permite crear de manera interactiva consultas a una o más bases de datos, el resultado de una consulta ("querie") se puede ver en pantalla, almacenar en una tabla o utilizar para la impresión de reportes o etiquetas. Todos los "queries" se interpretan como comandos SQL SELECT, y estos comandos pueden incluirse en la programación de aplicaciones para obtener el mismo resultado.

- **Reportes**. Permite generar reportes de una o más tablas.

- **Etiquetas**. Permite el diseño de etiquetas con información de una o más tablas.

- **Programación.** FoxPro permite la programación de aplicaciones haciendo uso de todos sus comandos, junto con otras instrucciones y funciones de apoyo.

II.3.3.2 Características del Generador de Reportes

Para usar el Generador de Reportes de FoxPro y explotar todas sus opciones, se deben tener claros los siguientes conceptos: tablas, índices de las tablas, relaciones, grupos, funciones aritméticas de FoxPro y en general las convenciones de teclas y manejo del ambiente de FoxPro.

El generador de reportes de FoxPro 2.0 cuenta con las siguientes opciones:

- **Generador rápido de reportes**. Las opciones que se pueden elegir son: Despliegue de campos en columnas o renglones, definición de títulos y selección de campos. El reporte generado puede modificarse.
- **Selección de campos**. Se pueden incluir en el reporte los campos de todas las bases de datos abiertas al crear el reporte.
- **Variables**. Permite crear o hacer uso de variables definidas para el reporte.
- **Campos calculados**. Esta opción contiene funciones y operadores para construir expresiones que utilicen, si se desea, los campos de las bases de datos o variables. El resultado de las expresiones aparece como un campo en el reporte.
- **Grupos de datos**. Esta opción permite crear hasta 20 grupos de datos.
- **Títulos o resumen**. Permite especificar si una banda de título o resumen se incluye en el reporte.
- **Texto**. Permite insertar texto al reporte.
- **Cuadros**. Se pueden dibujar cuadros con diferentes bordes.
- **Formato**. Se presentan los tipos de formatos disponibles para cada campo, de acuerdo a su tipo.

- **Organización de la página.** Mediante esta opción se modifican los márgenes y medidas de la página de impresión.

- **Estilo.** Permite seleccionar un estilo de impresión o se puede capturar un Código que sea soportado por la impresora.

II.3.3.3 Manejo del Generador de Reportes

Antes de entrar al generador de reportes se deben especificar las relaciones y los tipos de ordenamiento. Al guardar el reporte creado, se almacena también su ambiente, es decir las Bases de datos abiertas y sus relaciones. Lo más recomendado para crear reportes de varias tablas relacionadas, es generar un reporte como resultado de un querie. Este reporte se puede modificar desde el generador de reportes y su definición queda aunada al query.

Una vez dentro del generador de reportes, se presenta una pantalla que indica en la parte superior, el renglón y columna, y el área del reporte donde uno se encuentra; estas áreas también se especifican en el inicio de cada renglón y pueden ser: PgHead (Encabezado de la Página), Detail (Detalle), PgFoot (Pie de Página), Encabezado del Grupo (para los grupos definidos) y Pie de Grupo (para los grupos definidos). En la parte superior del menú aparece una nueva opción que es Report, mediante la cual se pueden llamar todas los opciones para crear o modificar el reporte (listados en el punto anterior).

Dentro de la pantalla para elaboración de reportes uno se mueve como dentro de un editor, y cabe mencionar que el soporte de Mouse facilita mucho más la edición, ya que permite seleccionar rápidamente opciones y mover campos y texto por toda la pantalla.

En general el uso del manejador de reportes resulta bastante sencillo y cuando se usa en combinación con el creador de consultas RQBE, la creación de los reportes más complejos se simplifica.

II.3.3.4 Uso del Generador de Etiquetas

El Generador de etiquetas de FoxPro es otra herramienta sencilla de usar. Como en el caso de la creación de reportes, primero se deben seleccionar la o las tablas y establecer las relaciones y después llamar el generador de etiquetas. El generador de etiquetas presenta una pantalla pequeña para diseñar la etiqueta. El tamaño de la etiqueta, el número de etiquetas a través del ancho de la hoja y el tipo de impresión

de los campos se selecciona a través del menú. Por medio de la opción Expression Builder se pueden agregar campos o expresiones a la etiqueta.

II.3.3.5 Conclusiones

Después de haber realizado el desarrollo de un sistema en FoxPro puedo decir que la interacción como usuario final con FoxPro no es tan intuitiva y sencilla como en Paradox, debido principalmente a que las pantallas con las que uno interactua se encuentran saturadas de opciones, mas sin embargo, después del uso constante del paquete y de obtener familiaridad con las pantallas, uno llega a acostumbrarse y el manejo se facilita.

A diferencia de Paradox, el usuario debe de encargarse de la creación y uso de los índices y de su regeneración en caso de una falla del sistema, por lo tanto, antes de hacer uso de una utilería uno debe llamar a las tablas y establecer las relaciones. Para lograr mayor flexibilidad, el usuario en FoxPro, debe de realizar más funciones de control de integridad que en Paradox.

Fuera de los comentarios anteriores debe mencionarse que las herramientas con las que cuenta FoxPro como el Generador de Reportes, el Generador de etiquetas, el Lenguaje SQL y Query by Example cuentan con una serie de caracteristicas que permiten una gran flexibilidad para obtener reportes y consultas, además de que es la única Base de Datos que permite el uso del mouse, tanto en modo interactivo como en la programación.

En cuanto a la programación FoxPro cuenta con una serie de comandos y funciones que le permiten ser la Base de Datos en la cual se pueda desarrollar la mejor interfaz para el usuario; ya que la creación de ventanas, Push Buttons, Radio Buttons, Check Boxes, Pop-up y Listas es sumamente sencilla para el programador. Si uno quiere que el usuario pueda usar el mouse sólo basta incluir la instrucción SET MOUSE ON. El único costo de todas estas características es el gran consumo de memoria. En equipos que no cuentan principalmente con memoria expandida, el rendimiento de los programas se ve sumamente disminuido.

La teoría de la programación de aplicaciones en FoxPro 2.0, utilizando el generador de pantallas, es que la pantalla y el código estén sumamente ligados; la selección de una pantalla, ventana o campo

origina la ejecución de una serie de funciones y validaciones. En el diseño, a la pantalla y a cada campo de control o captura, se le asocian una serie de funciones de control, mensajes informativos y de error, así como los validadores y características necesarias para la captura, en el caso de campos. Usando el generador de código del generador de pantallas se crea el código que maneja la pantalla y el código asignado a los campos y a la pantalla se agrupa dentro de funciones. Cabe mencionar que los nombres de las funciones y en algunos casos de las variables, creados por FoxPro son combinaciones de números y letras, lo cual no es claro para el programador, por lo que se puede optar por usar el editor de FoxPro para modificar el código generado.

El tipo de programación que se sigue en FoxPro sería la mejor dentro de las Bases de Datos analizadas de no ser por el compilador, que algunas veces dá la sensación de sólo eliminar comentarios. En el tiempo de prueba es cuando se descubren uno a uno los errores y en cada uno de ellos se puede perder mucho tiempo debido a que el error que señala el paquete es demasiado general y no ayuda al programador.

II.3.4 DBASE IV Versión 1.1

II.3.4.1 Características Generales de dBase IV

dBase IV es un manejador de Bases de Datos relacional que puede ser usado por un solo usuario en una PC o por varios usuarios en una red. Las principales características de este manejador son:

- **Bases de Datos**. Al igual que en Paradox, al definir una tabla se definen los campos índice, mas sin embargo la definición de validaciones se realiza en el diseño de cada pantalla que la use y esto puede originar inconsistencia en la captura.

- **Consultas** (Queries). Esta herramienta de consulta permite:
  - Ligar dos o más Bases de Datos.
  - Seleccionar sólo ciertos campos y/o registros.
  - Organizar los registros de uno o más archivos (Sort).
  - Sumarizar un grupo de registros.
    El resultado de un Query es una vista (View) o una actualización a una Base de Datos.

- **Formas**. Mediante esta utileria se pueden diseñar "n" formas asociadas a una Base de Datos; modificando el código generado para la forma se pueden incluir varias tablas relacionadas (no se realiza automáticamente y no se señala en la documentación).

- **Reportes**. Esta es una herramienta que permite crear reportes de una base de datos o de una vista (View, resultado de un Query). El editor del generador de reportes permite crear cartas y proporciona una serie de funciones para este fin.

- **Etiquetas**. Esta es una herramienta muy parecida al generador de reportes que permite la creación de etiquetas.

- **Generador de Aplicaciones**. El generador de aplicaciones permite la creación automática de procedimientos de alta, edición, consulta y de impresión de reportes (creados previamente).

- **Programación**. dBase IV permite la programación de aplicaciones haciendo uso del lenguaje de programación y del conjunto de comandos y funciones de dBase IV.

- **Utilerías.** Las utilerías de las cuales se dispone se relacionan con la especificación de la impresora y transferencia de datos.

II.3.4.2 Características del Generador de Reportes

Como en los generadores de reportes de los paquetes anteriormente analizados, se debe tener un conocimiento previo sobre tablas e índices para aprovechar al máximo las características de esta herramienta. En dBase IV, cuando se desea elaborar reportes de tablas relacionadas se debe crear primero la vista deseada (View), por medio de un query y posteriormente crear el reporte para esta vista; por lo tanto el conocimiento del generador de Queries y del ambiente y funciones de dBase IV es básico.

Las opciones que presenta el menú del generador de reportes son las siguientes:

- **Presentación**. Esta opción permite la selección del tipo de reporte instantáneo: con los campos en columnas o con los campos en renglones. Además del dibujo de líneas y cuadros en cualquier área del reporte.

- **Campos**. Permite añadir, borrar y modificar los siguientes tipos de campos:
  - Campos del archivo o vista seleccionada.
  - Campos calculados. Campos que son el resultado de una expresión válida de dBase IV. En los campos calculados se pueden definir funciones de formato (Picture Functions) para cada tipo de campo y plantillas, mediante las cuales se especifica cómo se representará un campo en el reporte.
  - Campos de resumen.
  - Campos predefinidos.
- **Bandas**. Esta opción permite la selección de características para las bandas (áreas en la pantalla que especifican a qué parte del reporte pertenece la información que agrupa, como: encabezado, detalle, resumen, etc.) y en especial la definición y modificación de grupos.
- **Palabras**. Esta opción permite:
  - Definir un estilo de impresión.
  - Incluir saltos de página.
  - Habilitar o deshabilitar la regla que se presenta en pantalla, así como el indentado.
- **Impresión**. Esta opción permite:
  - Direccionar la salida de un reporte a pantalla, a un archivo o a impresora.
  - Controlar la impresora. Se pueden definir códigos de control de inicio y fin, seleccionar impresión continua o por página y seleccionar la calidad de impresión.
  - Modificar el tamaño de página, márgenes y espacio entre líneas.
  - Seleccionar opciones de salida. Estas opciones son página de inicio o final, número de la primera página y número de copias.

II.3.4.3 Manejo del Generador de Reportes

Para crear un reporte se debe seleccionar previamente en el menú principal la Base de Datos o la Vista que se usará. Al entrar al Generador de Reportes se presenta una pantalla que esta dividida en diferentes áreas, delimitadas por líneas horizontales llamadas bandas.

Como en los demás generadores de reportes, se trata de diseñar en pantalla el formato de salida del reporte casi como se desea ver.

Para crear un reporte se puede seleccionar la opción de un reporte instantáneo, o si no, se puede comenzar a añadir campos en la banda que se desee. Los campos que sean calculados o de resumen se deben definir antes de incluirse.

Si se desea escribir texto, en el caso de cartas, se debe seleccionar en la opción de Banda el editor de palabras (Word Wrap), el cual distribuye las palabras, si es necesario en más de un renglón.

En general se puede decir que el uso del generador no es complicado, pero si involucra el uso de más teclas que en los otros paquetes.

II.3.4.4 Uso del Generador de Etiquetas

El uso del generador de etiquetas es casi idéntico al manejo del generador de reportes. La diferencia es que no existen bandas y por lo tanto grupos. Estando dentro de la pantalla, que simula el tamaño de la etiqueta, se insertan los campos deseados y texto, si es necesario. En el menú del generador de etiquetas se puede seleccionar alguno de los tamaños de etiquetas predefinidos o crear una nueva definición.

II.3.4.4 Conclusiones

Esta versión de dBase ofrece una interfaz un poco más amigable para el usuario final, que las versiones anteriores, llamada el "Centro de Control", esta interfaz utiliza menús tipo Lotus para permitir al usuario seleccionar las opciones que presenta cada una de las utilerías, aunque el gran inconveniente es que siempre se requiere de confirmación cuando se cambia de operación. En comparación con FoxPro, se extraña el uso del Mouse y de la facilidad para redimensionar las ventanas que se presentan y con respecto a Paradox, se extraña la sencillez de las pantallas y menús.

Una de las mejores herramientas de este paquete es el "Query Builder", que de una manera fácil permite la relación de varias bases de datos para realizar consultas de las tablas.

En lo que se refiere al Generador de Formas se puede decir que no soporta entradas de diferentes tablas relacionadas, a menos que manualmente se modifique el código que genera. Aparentemente si es

posible, porque antes de entrar al generador se llaman las bases de datos y se establecen las relaciones y al incluir campos en la forma de cualquiera de las tablas relacionadas no existe ningún problema; sin embargo, al correr la forma el paquete señala que no encuentra los campos.

II.3.5 CONCLUSIONES

II.3.5.1 FUNCIONALIDAD

En forma de resumen se ha elaborado la siguiente tabla comparativa, que presenta la funcionalidad de los generadores de reportes analizados y una última columna que se refiere a las funciones a implementar en el Generador de Reportes.

TABLA COMPARATIVA DE FUNCIONALIDAD

DESCRIPCIÓN DE LA FUNCION	PARADOX 3.5	FOXPRO/LAN 2.0	DBASE IV 1.1	GENERADOR DE REPORTES
Reportes tabulares instantáneos	X	X	X	
Reportes en formato libre	X	X	X	X
Añadir texto	X	X	X	X
Añadir campos				
Regulares	X	X	X	X
De resumen				
COUNT	X	X	X	X
HIGH	X	X	X	X
LOW	X	X	X	X
SUM	X	X	X	X
AVERAGE	X	X	X	X
STD.DEVIATION		X	X	
VARIANCE		X	X	
Calculados (Expresion)	X	X	X	
Calculados (Expresion)/ Variables	X	X	X	
Fecha	X	X	X	X
Tiempo	X	X	X	X
Número de página	X	X	X	X

DESCRIPCIÓN DE LA FUNCION	PARADOX 3.5	FOXPRO/LAN 2.0	DBASE IV 1.1	GENERADOR DE REPORTES
Número de registro	X	X	X	
Borrar campos	X	X	X	X
Dar formato a campos				
Fecha	X	X	X	X
Tiempo	X	X	X	
Alfanuméricos	X	X	X	X
Numéricos				
Dígitos	X	X	X	X
Negativos	X	X	X	
Comas	X	X	X	X
Justificación	X	X	X	
WordWrap	X		X	
Ligar otra tabla	X	[2]	[3]	X
Definir grupos				
Insertar				
Por un campo	X	X	X	X
Por rango alfanumérico	X	X		
Por rango numérico	X	X	X	
Por fecha	X	X		
Por un número de registro	X		X	
Por una expresión		X	X	
Borrar grupos	X	X	X	X
Encabezados por grupo	X	X	X	X
Ordenar registros de un grupo	X	[2]	[3]	X
Dibujar cajas		X	X	X
Imprimir				

[2] Antes de entrar al generador de reportes se deben especificar las relaciones y los tipos de ordenamiento. Al guardar el reporte creado se almacena también su ambiente, es decir, las bases de datos abiertas y sus relaciones.

[3] Para crear un reporte se debe seleccionar previamente en el menú principal la base de datos o la vista que se usará.

DESCRIPCIÓN DE LA FUNCION	PARADOX 3.5	FOXPRO/LAN 2.0	DBASE IV 1.1	GENERADOR DE REPORTES
Impresora	X	X	X	X
Pantalla	X	X	X	
Archivo	X	X	X	X
Definiciones				
Formato de página	X	X	X	
Márgenes	X	X	X	
Tipo de impresora	X	X	X	
Ayuda	X	X	X	X
Generación de código en C				X

Como se puede observar en la tabla anterior la funcionalidad de los tres generadores de reportes analizados es muy similar, la mayor diferencia se centra en la forma en que se deben de relacionar las tablas, en la definición de variables o campos calculados y en el manejo del editor. En la última columna se ha indicado la funcionalidad que abarcará el generador de reportes, la cual cubre la mayoría de los puntos. Por último debo mencionar que al realizar el análisis y diseño del generador de reportes, tomaré en cuenta lo que me pareció más claro para los usuarios de cada uno de los generadores de reportes analizados, en cuanto a la interfaz con el usuario e implementación de la funcionalidad.

II.3.5.2 ANTECEDENTES NECESARIOS PARA LOS USUARIOS

De manera general puedo decir que cualquier usuario final que desee hacer uso de las herramientas de las bases de datos analizadas (básicamente para la creación de una base de datos y el análisis de la información que captura mediante el generador de reportes) y aprovechar al máximo sus facilidades, debe contar con los conocimientos básicos de bases de datos como son tablas, índices y relaciones ya que son conceptos que forman parte de los menús y que se manejan constantemente. Para un programador ese hecho es irrelevante porque tiene este conocimiento, pero para un usuario de cualquier otra área estos conceptos muchas veces son un gran misterio aún después de leer los manuales y hacer uso del paquete.

Por lo que concluyo que se debe de introducir al usuario en estos conceptos como parte de un paquete de esta naturaleza, con el fin de

que los usuarios puedan aprovechar al máximo los beneficios de las bases de datos. Indudablemente la sencillez de la interfaz del paquete, la claridad de la documentación y el tiempo de respuesta serán la principal característica que un usuario final busque en el paquete a seleccionar.

II.3.5.3 SELECCIÓN DE LA BASE DE DATOS, DE LA QUE SE PODRÁN OBTENER REPORTES USANDO EL GENERADOR DE REPORTES

En una comparación entre las dos bases de datos más recomendadas como son FoxPro y Paradox puedo decir que Paradox trata de simplificar al máximo su uso a los usuarios finales, lo cual a mi parecer logra bastante bien, sin embargo, este hecho reduce la flexibilidad para los programadores quienes deben de conocer a fondo Paradox antes de intentar programar alguna aplicación más compleja y, aún contando con la experiencia, el desarrollo es más lento que en cualquier otro paquete. Por otra parte FoxPro permite una mayor flexibilidad en su manejo, lo cual implica mayor número de opciones al interactuar con los usuarios y sí permite una mayor flexibilidad en la programación (con excepciones como las pantallas de múltiples registros "browse").

Después de realizar este breve análisis, se decidió desarrollar el Generador de Reportes en el lenguaje de programación "C" e interactuar con la base de datos Paradox haciendo uso del paquete Paradox Engine. Básicamente porque:

- Paradox resulta una Base de Datos realmente atractiva para los usuarios finales.
- Paradox es una base de datos orientada a usuarios finales cuyo fin es ofrecer una interfaz sencilla y una serie de herramientas que automáticamente realicen el control de integridad. Estos beneficios se pagan con un sacrificio de flexibilidad al querer desarrollar aplicaciones usando el lenguaje de programación, los comandos y funciones de Paradox.
- Las condiciones de programación haciendo uso del lenguaje de programación de Paradox (PAL-Programming Application Language) no son tan favorables para el programador y el uso de un lenguaje de programación como "C" si representa una ventaja.
- FoxPro y dBase permiten mayor flexibilidad en la programación.

Finalmente concluyo reafirmando que el objetivo de desarrollar un Generador de Reportes, no es el de sustituir al generador de reportes de Paradox, ni a cualquier otro, sino el de ampliar su interoperatividad con otros paquetes, el sistema operativo y ampliar la flexibilidad, en este caso, para la explotación de la información.

II.4 MODELO LÓGICO DEL EDITOR DEL GENERADOR DE REPORTES

Una vez que se definió la funcionalidad, se concluyó que el generador de reportes estaría dividido en dos módulos principales:

- **El editor**.- Mediante el cual el usuario tenga una interfaz con el sistema para definir un reporte, en el cual pueda incluir la funcionalidad especificada.

- **El generador de código**.- El cual sea un proceso automático, que en base a la definición del reporte llevada a cabo por el usuario en el módulo del editor, realice la generación de código en lenguaje de programación "C".

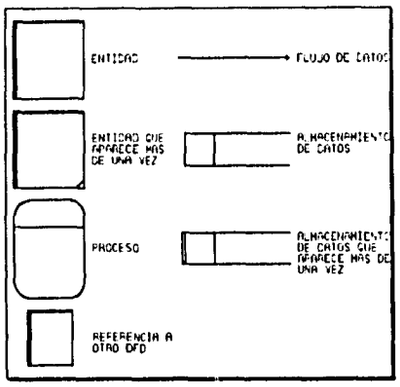
En esta parte del capítulo se realiza en primer lugar una breve revisión de los fundamentos que se siguieron para la creación de los Diagramas de Flujo de Datos (DFD´s) por medio de los cuales se representa en forma esquemática el generador de reportes. Tanto para el módulo del editor como para el módulo del generador de reportes, se presentan únicamente los niveles superiores, todos los niveles se incluyen en el documento *Anexo Apéndice "B"*.

II.4.1 DIAGRAMAS DE FLUJO DE DATOS (DFD)

Conforme la información se mueve a través del software, se modifica mediante una serie de transformaciones. Un diagrama de flujo de datos (DFD), es una técnica gráfica que describe el flujo de información y las transformaciones que se aplican a los datos, conforme se mueven de la entrada a la salida. El DFD puede usarse para representar un sistema de software a cualquier nivel de abstracción. De hecho, los DFD pueden particionarse en niveles que representan flujo incremental de información y detalle funcional.

Un DFD cuenta con los siguientes elementos:

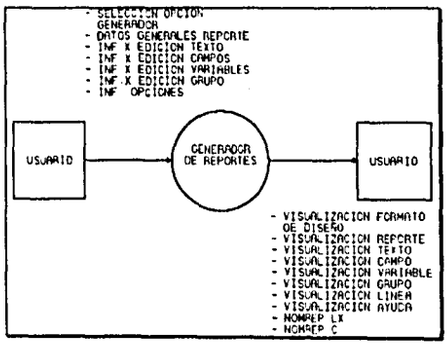
- Flujo de datos: representado por un vector (flecha).
- Procesos: representados por una "burbuja".
- Archivos: representados por rectángulos abiertos.
- Entidades: representados por rectángulos.



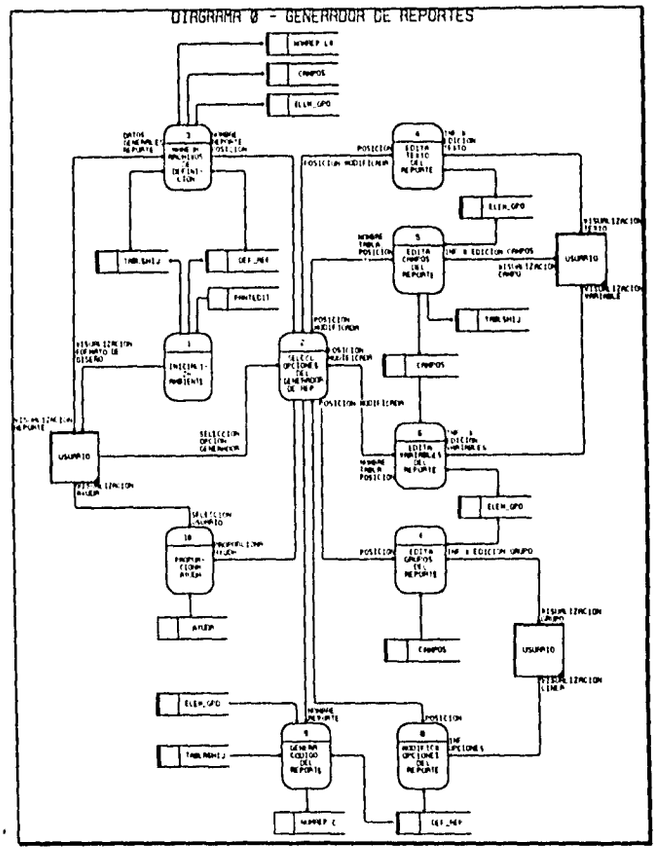
A continuación se listan los criterios que se siguieron para la elaboración de los DFD's:

1) El nivel 01 del diagrama de flujo de datos debe describir al software/sistema como una burbuja sencilla;
2) Los archivos de entrada/salida principales deben ser anotados cuidadosamente;
3) Todas las flechas y burbujas deben estar etiquetadas (con nombres con significado)
4) La continuidad del flujo de información debe ser mantenida
5) Cada vez debe refinarse una burbuja.

DIAGRAMA DE CONTEXTO DEL GENERADOR DE REPORTES

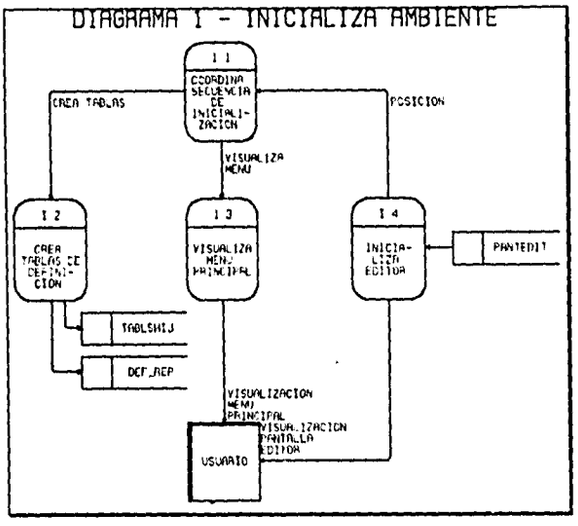


II.4.2 DIAGRAMAS DE FLUJO DE DATOS DEL EDITOR



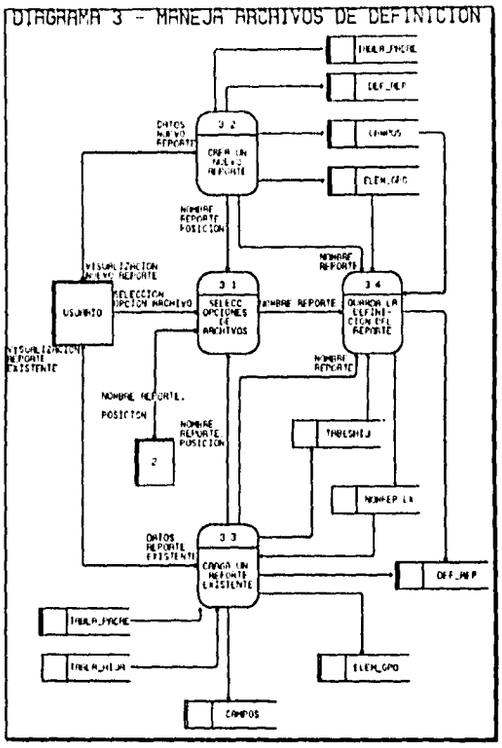
GENERADOR DE REPORTES - EDITOR

Permite al usuario final diseñar el formato de salida de un reporte, a través de una serie de menús y de un editor de texto sencillo. Haciendo uso de las opciones del editor, el usuario puede explotar la funcionalidad que brinda el generador de reportes (incluir, campos de diferentes tablas de Paradox, definir grupos, definir variables, etc.) y aplicarla en la definición del reporte. El editor va almacenando la definición del reporte (en ELEM_GPO), la cual puede guardarse/leerse en/de un archivo ASCII (NOMREP.LX) y servir como base para la generación de código.



1.0 INICIALIZA AMBIENTE

Este proceso se realiza en forma automática una vez que se entra al "Generador de Reportes". Su fin es el de crear las tablas donde se guardará parte de la definición del reporte, como la relación entre tablas (TABLSHIJ) y los datos generales de definición del reporte (DEF_REP), así como el de presentar al usuario final el menú con las opciones generales del generador de reportes y el formato inicial de diseño.



3.0 MANEJA ARCHIVOS DE DEFINICIÓN

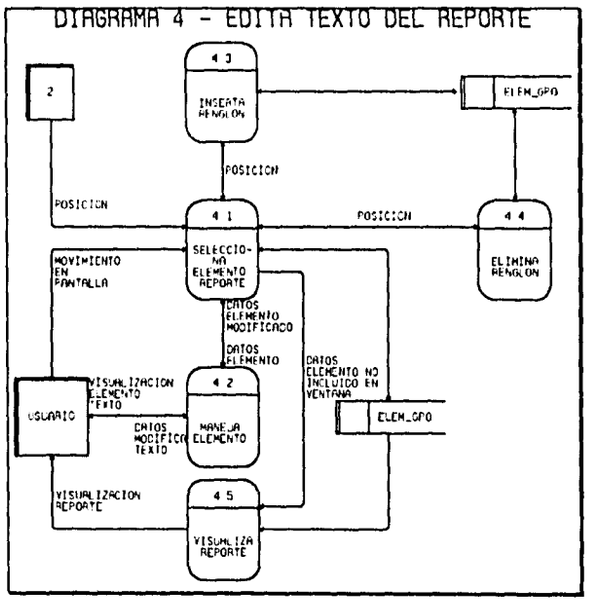
El usuario al elegir la opción de Archivos puede:

- Crear un nuevo reporte.-Proceso que solicitará al usuario el nombre del nuevo reporte y los datos descriptivos del reporte. En forma automática leerá en CAMPOS la información de los campos de la tabla seleccionada, leerá en ELEM_GPO una definición por default para comenzar un reporte, y finalmente permitirá al usuario visualizar un nuevo reporte.

- Cargar un reporte existente.- El usuario selecciona el archivo de definición del reporte que desea cargar y con esta información se carga, en los diferentes archivos del generador de reportes, un reporte hecho con anterioridad el cual se visualiza al final de esta

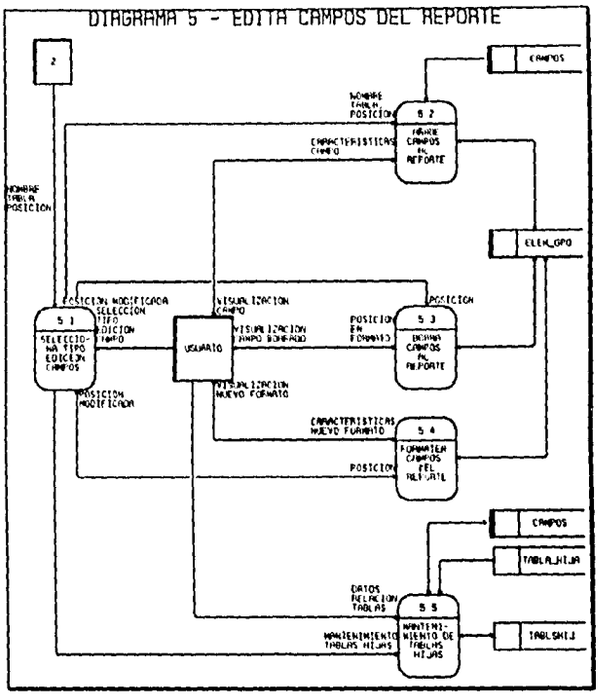
opción. El usuario puede continuar la definición de un reporte o generar código según lo desee.

- Guardar la definición del reporte.-Esta opción almacena la definición completa del reporte generado por el usuario en el módulo del editor, en un archivo ASCII (NOMREP.LX).



4.0 EDITA TEXTO DEL REPORTE

Esta opción permite al usuario desplazarse a través de la definición del reporte (ELEM_GPO) e insertar o modificar texto, así mismo se pueden insertar o eliminar renglones.



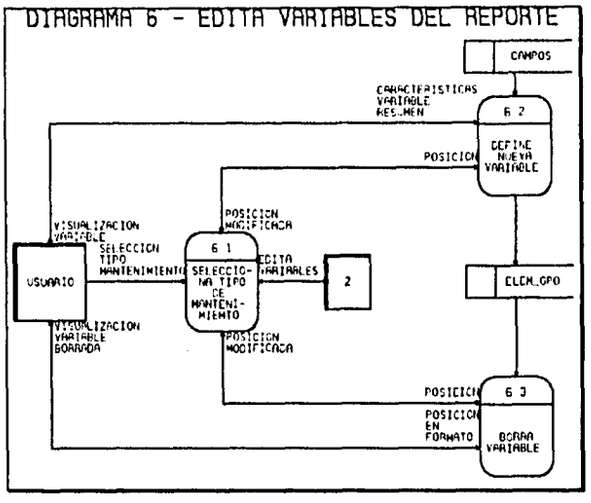
5.0 EDITA CAMPOS DEL REPORTE

Mediante esta opción el usuario puede:

- Incluir campos de la tabla que haya seleccionado originalmente, seleccionándolos del contenido de CAMPOS e indicando la posición en la definición del reporte.

- Borrar o dar formato a los campos seleccionándolos en la definición del reporte.

- Ligar otras tablas a la tabla original, indicando la relación de los campos llave de la nueva tabla (TABLA_HIJA) con campos de la tabla original (la relación se almacena en TABLSHIJ). Los campos de la TABLA_HIJA son leídos en CAMPOS.



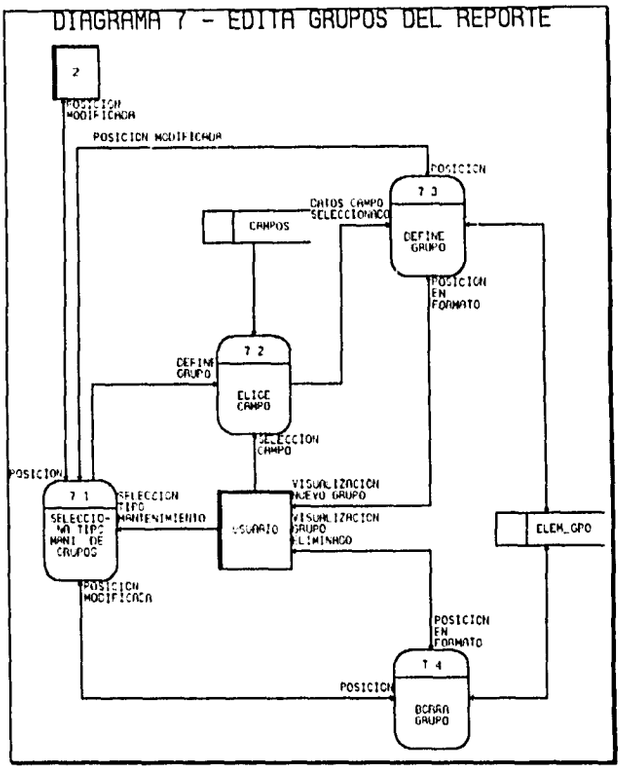
6.0 EDITA VARIABLES DEL REPORTE

El usuario puede elegir insertar en la definición del reporte alguno de los siguientes tipos de variable:

Variable fecha.- la cual imprimirá la fecha que tenga el sistema.

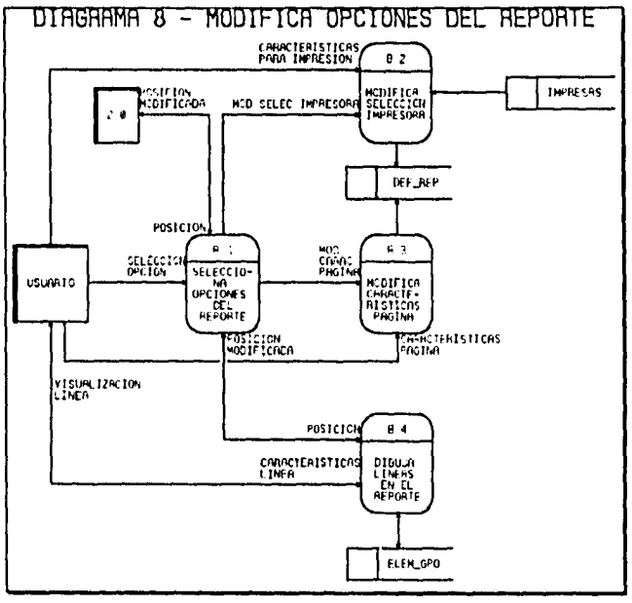
Variable Número de Página

Variable de resumen.- El usuario podrá seleccionar un campo y aplicar alguna de las siguientes funciones de resumen: SUM, AVG, COUNT, MAX y MIN.



7.0 EDITA GRUPOS DEL REPORTE

Al iniciar un nuevo reporte, éste contiene tres diferentes áreas: encabezado, detalle y total. El usuario puede insertar grupos mediante esta opción eligiendo un campo de la tabla original, en consecuencia, se crearán automáticamente dos nuevas áreas en el reporte: encabezado del grupo y total del grupo donde el usuario podrá expandir la definición del reporte. Si el usuario elige eliminar un grupo, las áreas de encabezado y total del grupo, junto con todo el trabajo que el usuario hubiera realizado en ellos, se elimina de la definición del reporte.



8.0 MODIFICA OPCIONES DEL REPORTE

Esta opción permite al usuario modificar las principales características del reporte, por otro lado permite dibujar líneas o cajas en la definición del reporte.

II.5 ANÁLISIS DE LA GENERACIÓN DE CÓDIGO

Una vez que se definió la funcionalidad del "Generador de Reportes" y se analizó la forma en que las podría definir un usuario en el modelo lógico del editor; se procedió a la programación de varios reportes en lenguaje "C" que abarcarán todos los puntos de la funcionalidad, el objetivo fue el analizar la sintaxis de dichos reportes para poderla reproducir subsecuentemente en forma automática. El resultado fue el análisis de la sintaxis para un sublenguaje de "C". El cual se presenta en esta sección como base para la elaboración del modelo lógico de la generación de código.

II.5.1 LA CONSTRUCCIÓN DE UN PROGRAMA

Un programa es una secuencia de instrucciones. En el lenguaje "C", existe una serie de reglas precisas para construir programas completos, las cuales dictan la posibilidad de secuencia de las instrucciones. Algunas características de los programas son:

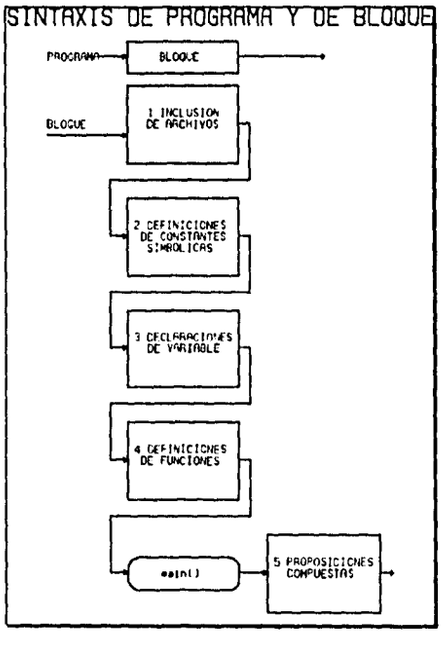
- Los programas tienen una estructura.
- El programa opera sobre ciertos objetos.
- Las instrucciones son precedidas por una declaración de los objetos sobre los cuales aquellas operan.

En este caso nos enfocaremos a la construcción de código para un sublenguaje de "C" para un reporte que cumpla con la funcionalidad especificada en la *Tabla Comparativa de Funcionalidad* para el "Generador de Reportes" (presentada en la sección II.3.5.1). Las reglas para construir un reporte las resumiremos por medio de los *diagramas de sintaxis*. En los cuales un rectángulo es un diagrama de sintaxis que representa una referencia a otro diagrama de sintaxis y una caja con finales redondeados contiene un símbolo que debe corresponderse exactamente.

II.5.2 DIAGRAMAS DE SINTAXIS

SINTAXIS DE PROGRAMA Y DE BLOQUE

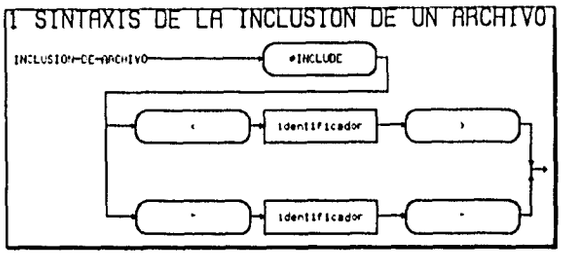
En primer lugar diremos que un programa consiste en un bloque.



Las proposiciones en el bloque son opcionales, mas sin embargo deben seguir el orden que se presenta en el diagrama de sintaxis de bloque.

SINTAXIS DE LA INCLUSIÓN DE UN ARCHIVO

Para facilitar el manejo de declaraciones, junto con otras cosas, el lenguaje "C" posee las dos posibilidades de incluir archivos, representadas por el siguiente diagrama:



La primera se refiere a archivos que contienen declaraciones de funciones de la biblioteca estándar y la segunda a archivos definidos por el usuario.

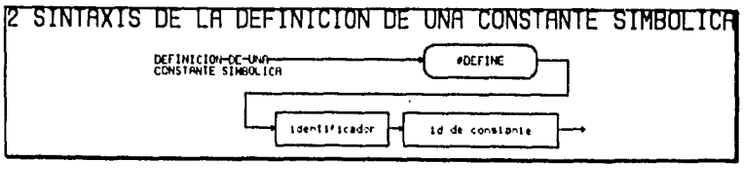
Inclusión de archivos en el código del reporte:

1. En nuestro caso identificamos las siguientes inclusiones de archivo que permanecen constantes:

```
#include <stdio.h>
#include "pxengine.h"
#include "cadenas.h"
#include "sistema.h"
#include "tiempo.h"
#include "pxfunc.h"
#include "impresor.h"
```

SINTAXIS DE LA DEFINICIÓN DE UNA CONSTANTE SIMBÓLICA

Mediante la construcción #define, se puede definir un nombre simbólico o constante simbólica como determinada cadena de caracteres. Posteriormente el compilador reemplazará todas las apariciones no entrecomilladas del nombre por su cadena correspondiente.



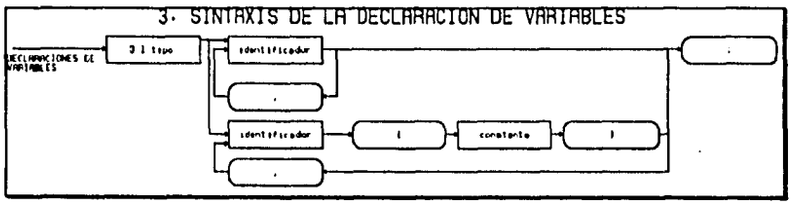
Definición de constantes simbólicas en el código del reporte:

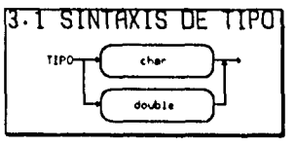
1. En Paradox Engine al realizar una operación referente a una tabla en especifico se identifica con un "TableHandle" y una operación referente al contenido de un registro en particular se identifica con un "Record Handle", para este efecto hemos pensado definir un arreglo en el cual cada uno de sus elemento se pueda accesar con una constante simbólica ("KN" + nombre de la tabla) cuyo valor sea progresivo (1..knMaxTableHndls).

2. Otra variable simbólica a definir será el directorio donde se encuentran los datos.

SINTAXIS DE LA DECLARACIÓN DE VARIABLES

Cada objeto de los datos en un programa es una constante o una variable, la diferencia está en que el valor de una variable puede cambiar durante la ejecución del programa. Cada variable en un programa tiene un tipo asociado a ella, y el tipo determina a la vez los valores que la variable puede tomar y las operaciones que pueden realizarse con ésta.





Declaraciones de variables en el código del reporte:

1. En la sección de la declaración de variables tenemos las siguientes declaraciones que permanecen en todos los reportes:

```
RECORDHANDLE aRecHandle [ knMaxRecBufs ];
TABLEHANDLE aTblHandle [ knMaxTableHndls ];
```

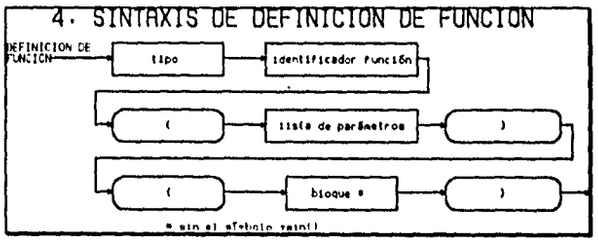
2. Se tendrá que definir una variable para cada una de las variables calculadas.

3. Se tendrá que definir una variable para cada uno de los campos utilizados en el reporte, para que en cada uno de ellos se lea el valor del campo de un registro de la tabla correspondiente.

4. Se tendrá que definir una variable para cada uno de los campos que formen parte de la llave con la que se relacionan los archivos.

SINTAXIS DE LA DEFINICIÓN DE FUNCIÓN

Las funciones dividen grandes trabajos de computación en partes más breves. Unas funciones apropiadas a menudo consiguen abstraer los detalles de operación en aquellas partes del programa que no necesitan conocerlas, clarificando por tanto el conjunto y facilitando la tarea de realizar cambios.



Definición de funciones en el código del reporte:

1. Abre archivos

2. Cierra Archivos

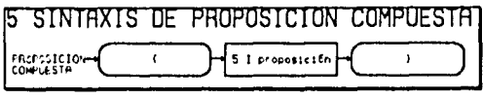
3. Una función de Lectura del Registro para cada una de las tablas

4. Cálculo de variables

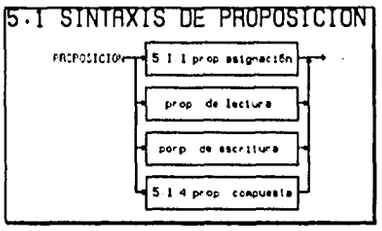
5. Una función que imprima el valor de los campos y variables seleccionadas por el usuario para cada una de las áreas del reporte. El área del reporte se divide en encabezado, detalle y total, al insertar un grupo se incluyen dos nuevas áreas encabezado y total por grupo.

PROPOSICIONES

Una proposición compuesta es una secuencia de proposiciones introducida por { y terminada por }.

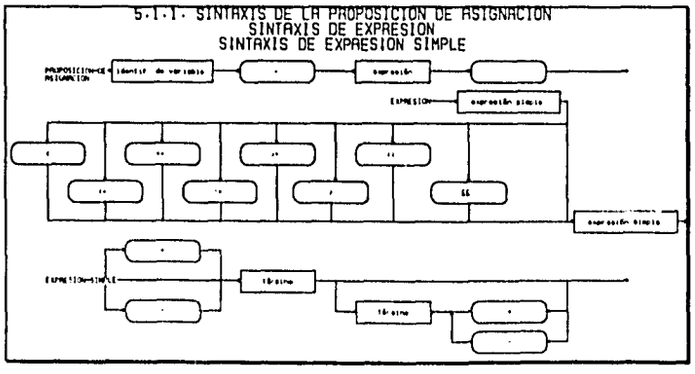


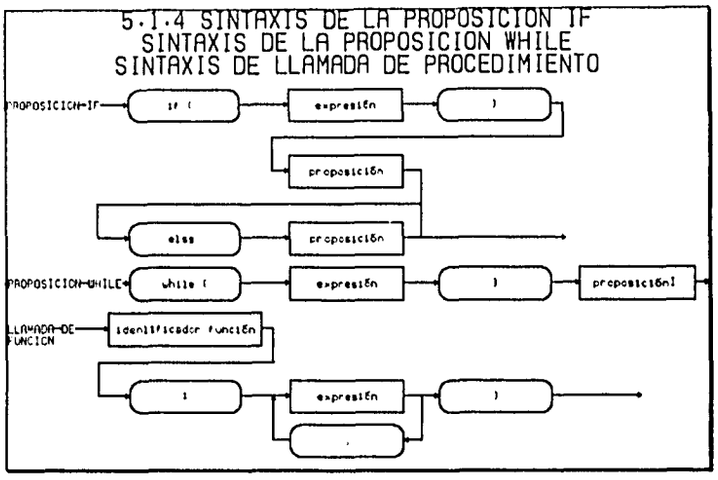
El siguiente diagrama muestra que una proposición simple puede ser una proposición compuesta.



Esta es una construcción de suma importancia ya que siempre que una proposición sea escrita en un programa, una proposición compuesta puede ser usada en su lugar.

A continuación se presentan los diagramas de sintaxis para la proposición if y la proposición while las cuales serán las únicas proposiciones de control de flujo a utilizar en el código de los reportes, seguidas por el diagrama de sintaxis para llamada a una función.





PROPOSICIONES COMPUESTAS DE main()

main() es el nombre de una función especial ya que el programa comienza a ejecutarse al principio de main. Esto significa que todos los programas deben tener la función main. Esta función principal hará uso de las funciones definidas anteriormente para realizar así la impresión del reporte que se desea generar, o en segunda alternativa hará uso de las funciones que están definidas en las bibliotecas de funciones definidas al inicio del programa.

Proposiciones compuestas para la función main en el código del reporte:

1. Definición de una variable para cada campo por medio del cual se haya definido un grupo.
2. Incluir el nombre de la tabla padre en un if que verifica si la tabla tiene o no registros
3. Incluir llamadas a las funciones de lectura de registro para cada una de las tablas incluidas en el reporte
4. Con el fin de controlar los cortes de información por grupos, incluir una comparación para determinar si se trata o no del mismo grupo
5. Incluir llamada a la función de impresión del área de detalle del reporte

6. Incluir llamadas a las funciones que imprimen los datos contenidos en el área de totales para el reporte y para cada uno de los grupos definidos.

7. Así por medio de este diagrama de sintaxis podremos identificar los pasos en los que el generador de código debe de sustituir una declaración de objeto, por el objeto definido por el usuario y podremos automatizar este proceso.

II.5.3 TABLA DE SÍMBOLOS

Las tablas de símbolos (también llamadas tablas de identificadores y tablas de nombres) asisten en dos funciones importantes en el proceso de traducción de un compilador: en verificar la semántica y en auxiliar en la generación de código. Ambas funciones se realizan al insertar y traer atributos de la tabla de símbolos, de las variables usadas en el programa fuente. Los atributos incluyen el nombre, tipo, dimensión de la variable, etc.

En el caso del módulo generador de código, no se construye un compilador ya que lo que genera el usuario a través del módulo editor es por decirlo así una lista de identificadores (campos, variables, grupos, y texto) y no una lista completa de tokens (palabras reservadas de un lenguaje de programación en particular y símbolos) los cuales tengamos que determinar si pertenecen o no al lenguaje.

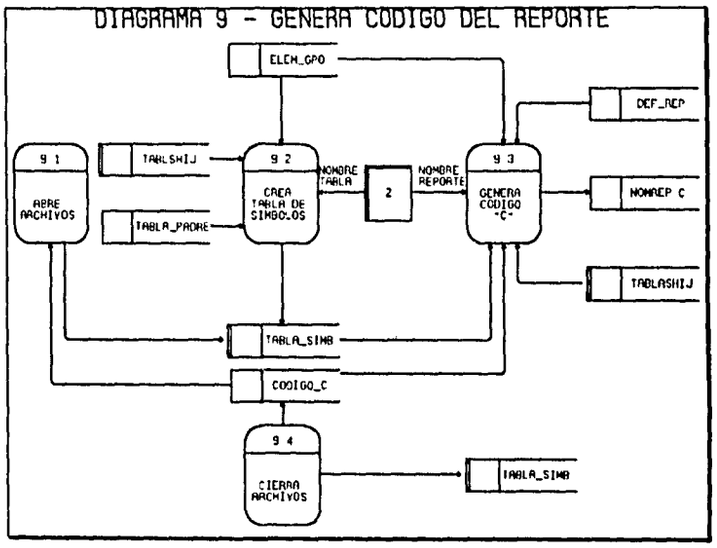
El módulo generador de código se guiará por la sintaxis descrita anteriormente, mas sin embargo, como se puede apreciar en el análisis de los diagramas de sintaxis, el elemento mínimo son los identificadores, por lo que será necesario crear en primer lugar una tabla de símbolos, con el fin de auxiliar en la generación de código.

II.6 MODELO LÓGICO DEL GENERADOR DE CÓDIGO

Con base en el análisis para la Generación de Código realizada en el sección anterior y en el modelo lógico del módulo del Editor, se elaboró el modelo lógico del módulo Generador de Código que se presenta en esta sección.

II.6.1 DIAGRAMAS DE FLUJO DE DATOS DEL GENERADOR DE CÓDIGO

A continuación se muestran los niveles superiores de DFD's para el módulo Generador de Código.

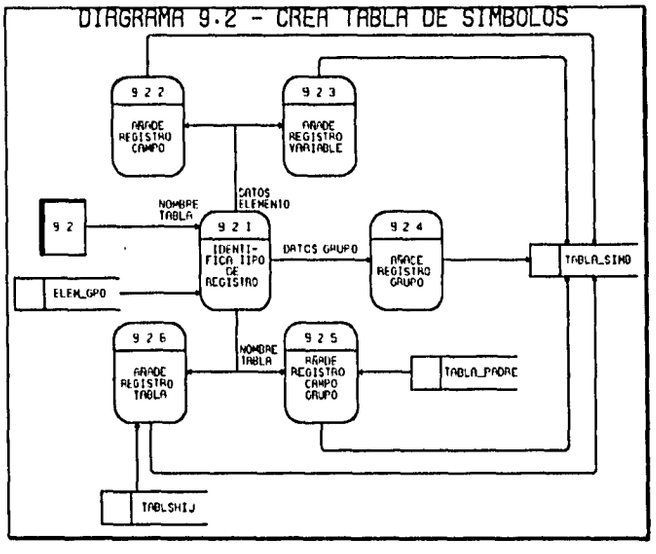


GENERADOR DE CÓDIGO

Con base en la definición del reporte realizada por el usuario en el editor, se genera automáticamente el código requerido en lenguaje de programación "C". El generador de código tiene dos fases:

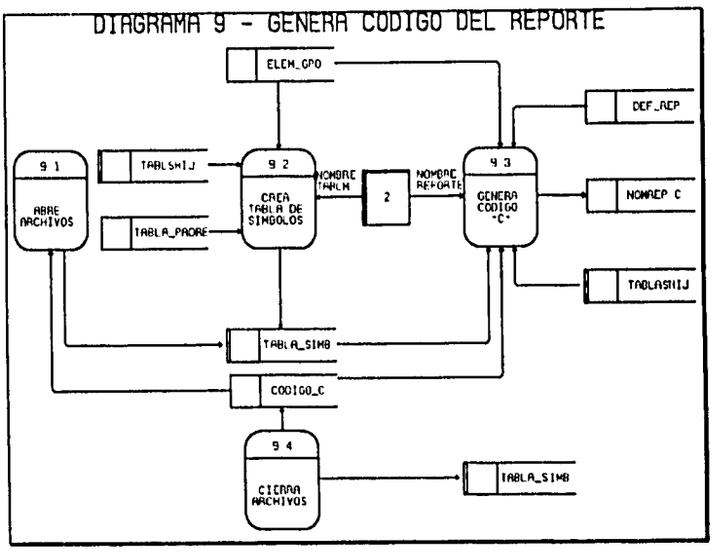
Fase I.- Con base a la definición del reporte se genera una tabla de campos, variables, grupos y tablas usados en el reporte (tabla de símbolos).

Fase II.- Siguiendo la definición del *diagrama de sintaxis* se va generando el código en lenguaje de programación "C", los procedimientos que se llaman hacen uso de la tabla de símbolos y de la definición del reporte para realizar la generación especifica de código.



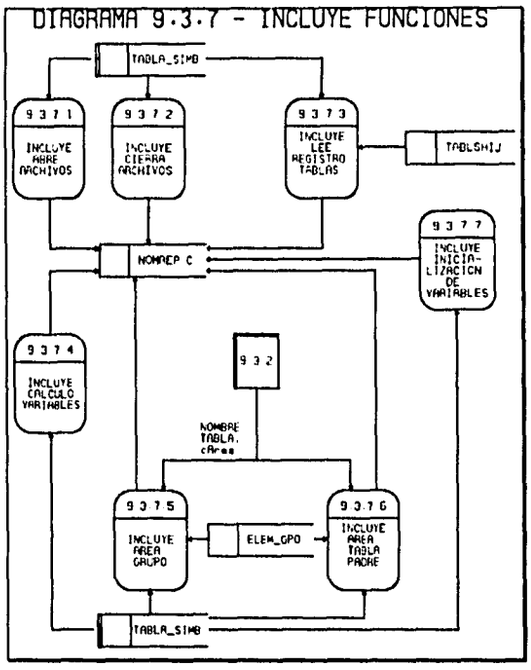
9.2 CREA TABLA DE SÍMBOLOS

El proceso que crea la tabla de símbolos lleva a cabo una revisión completa de la definición del reporte realizada por el usuario en ELEM_GPO; al ir identificando diferentes tipos de elemento (campo, variable, grupo), llama a una función la cual crea un registro en la tabla de símbolos para cada elemento. Finalmente realiza una revisión de TABLASHIJ con el fin de incluir en la tabla de símbolos cada una de las tablas utilizadas en la definición del reporte.



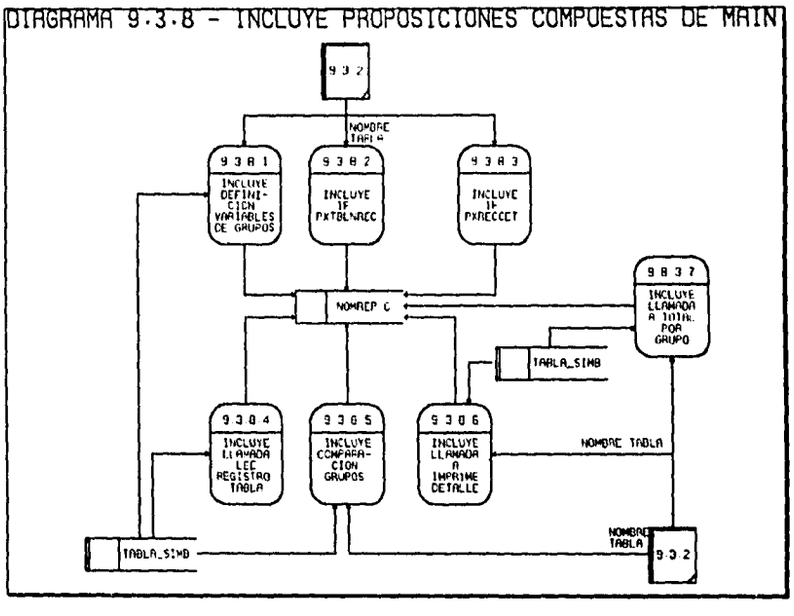
9.3 GENERA CÓDIGO "C"

El archivo CODIGO_C contiene la secuencia que se debe de seguir en la generación de código, la cual se basa en el análisis realizado en el punto II.5. de este capítulo. La función 9.3.2. se encarga de leer este archivo y de esta manera controlar la secuencia para la generación de código. Al leer un registro de CODIGO_C, éste indica si se tiene que realizar una definición de constantes simbólicas, una proposición específica, etc., con lo cual la función llama a otra función para que genere el código deseado, basándose principalmente en la tabla de símbolos TABLA_SIMB y en la definición del reporte realizada por el usuario en ELEM_GPO.



9.3.7 INCLUYE FUNCIONES

Cada una de las funciones de este submódulo es llamada por la función 9.3.2 dependiendo de si el usuario incluyó o no en la definición del reporte los elementos en las que se basan para generar código, sus características comunes son el incluir en el código del reporte las principales funciones que serán llamadas por la función main.



9.3.8. INCLUYE PROPOSICIONES COMPUESTAS DE MAIN

Este segundo grupo de funciones también son llamadas por la función 9.3.2 dependiendo de si el usuario incluyó o no en la definición del reporte los elementos en las que se basan para generar código, sus características comunes son el incluir en el código del reporte las principales proposiciones compuestas que forman parte de la función main.

II.7 DICCIONARIO DE DATOS

Un análisis del dominio de la información puede ser incompleto si sólo se considera el flujo de datos mostrados en los DFD's. Cada flecha de un diagrama de flujo de datos representa uno o más elementos de información.

Se ha propuesto el diccionario de datos como una gramática casi-formal para describir el contenido de los elementos de información y ha sido definido de la siguiente forma:

"El diccionario de datos contiene las definiciones de todos los datos mencionados en el DFD, en una especificación del proceso y en el propio diccionario de datos. Los datos compuestos (datos que pueden ser además divididos) se definen en términos de sus componentes; los datos elementales (datos que no pueden ser divididos) se definen en términos del significado de cada uno de los valores que puede asumir. Por tanto, el diccionario de datos está compuesto de definiciones de flujo de datos, archivos [datos almacenados] y datos usados en los procesos [transformaciones]..."[1]

Los siguientes puntos de esta sección describen las definiciones de todos los datos mencionados en los módulos del editor y del generador de código del "Generador de Reportes".

II.7.1 NOTACIÓN DEL DICCIONARIO DE DATOS

- = está compuesto de
- \+ y
- () optativo (puede estar presente o ausente)
- { } iteración ({ } seguido de una constante numérica especifica el número de veces que se repite la iteración)
- [ ] seleccionar una de varias alternativas
- ** comentario
- @ identificador (campo clave) para un almacén
- | separa opciones alternativas en la construcción

[1] Page-Jones, M., *The Practical Guide to Structured Systems Design*, Yourdon Press, New York, 1980, p.75

NOTAS:

1. En la sección II.7.2 se podrá observar que los atributos que han sido utilizados en la definción de una estructura comienzan con una letra minúscula, la cual tiene el siguiente significado:

   c indica que es un atributo con tipo Alfanumérico

   n Indica un atributo con tipo Numérico

   d Indica un atributo con tipo Fecha

1. Por otro lado las definiciones acerca de características específicas de los atributos se presentan en MAYÚSCULAS.

2. En la sección II.7.3 se observará que la declaración de una estructura comienza con las letras minúsculas **st**.

3. Todos los flujos de datos están escritos en MAYÚSCULAS, igual que en los DFD's .

II.7.2 ATRIBUTOS

AÑO = [00..99]

cAccion = { CARACTER VALIDO ALFANUMERICO }10

CAMPO = [CAMPO ALFANUMERICO | CAMPO NUMERICO | CAMPO FECHA]

CARACTER VALIDO ALFANUMERICO = [a..z | A..Z | 0..9]

CARACTER VALIDO NOMBRE ARCHIVO = [a..z | A..Z | - | 0..9]

CARACTER VALIDO NOMBRE CAMPO = [a..z | A..Z | _ | 0..9]]

CARACTER VALIDO TEXTO = [a..z | A..Z | á | é | í | ó | ú | | !../ | :..? | 0..9]]

cArea	= [Encabezado | Detalle | Totales]
cCampo	= cNomCampo
cCampoLlaveHija	= cNomCampo
cCampoLlavePadre	= cNomCampo
cCodigo	= { CARACTER VALIDO ALFANUMERICO }100
cExpresion	= FUNCION RESUMEN
cFormato	= [{A}| {(,)9} | 9.{9} | {D}(-){M}(-)({A})]
cIdArea	= [D*etalle* | E*ncabezado* | T*otal*]

cIdClaseDato = [C*campo* | G*grupo*+ SECUENCIAL | T*abla* | V*ariable*]

cIdTipoTabla = [P*adre* | H*ija*]

cImpCadenaConfig = { CARACTER VALIDO ALFANUMERICO } 25

cImpPuerto = [LPT1 | LPT2]

cJustificacion = [Derecha | Izquierda | Centrado | Justificado]

cNomAutor = { CARACTER VALIDO ALFANUMERICO }35

cNombre = cNomCampo

cNomCampo = 1 { CARACTER VALIDO NOMBRE CAMPO } 25

cNomDato = [(cNomTabla + _) + cNomCampo | cNomGrupo | cNomTabla | n + SECUENCIAL + [Adi | Cue | Max | Min | Pro] + cNomCampo]

cNomGrupo = cNomCampo

cNomTabla = 1 { CARACTER VALIDO NOMBRE ARCHIVO } 8

cNomTablaHija = cNomTabla

cPagFormaContinua = [SI | NO]

cTexto = {CARACTER VALIDO TEXTO} 80

cTipo = cTipoCampo

cTipoCampo = [ALFANUMERICO | NUMBYTE | NUMDOBLE | NUMDINERO | FECHA]

cTipoDato = cTipoCampo

cTipoGrupo = cTipoCampo

cTipoUnion = [T*exto* | I*nformativa* | C*ampo* | *campo tabla *H*ija* | V*ariable* | v*ariable tabla hija*]

cTituloReporte = { CARACTER VALIDO ALFANUMERICO } 50

dFechaActualizacion = DIA + / + MES + / + AÑO

dFechaCreacion = DIA + / + MES + / + AÑO

DIA = [1..30]

DIGITOS NUMERO = [{(,)9} | 9.{9}]

EXTENSION REPORTE = 'L' + NUMERO REPORTE

FORMATO ALFANUMERICO = cJustificacion

FORMATO FECHAS = [DD/MMMMMMMMMM/AAAA | DD/MMM/AA | DD/MM/AA]

FORMATO NUMERICO= [FCOMAS | FDINERO]

FORMATO SELECCIONADO = [FORMATO ALFANUMERICO | FORMATO NUMERICO | FORMATO FECHAS]

FUNCION RESUMEN = [Cuenta | mAximo | mInimo | Promedio | Suma]

LIMITE GRUPO = cNomGrupo + {-} 80

LONGITUD A MODIFICAR = * LONGITUD EN QUE DISMINUYE O AUMENTA EL CAMPO DEBIDO AL CAMBIO DE FORMATO *

LONGITUD A REDUCIR = * LONGITUD QUE SE REDUCIRA EL TEXTO DEL SIGUIENTE ELEMENTO, COMO CONSECUENCIA DE HABER INSERTADO UN NUEVO ELEMENTO *

LONGITUD CAMPO = *MAXIMA LONGITUD DEFINIDA PARA EL CAMPO*

MES = [1..12]

MOVIMIENTO EN FORMATO REPORTE = [Arriba | Abajo | Izquierda | Derecha]

nImpresora = [1..N]

nLongitud = *TAMAÑO DEL ELEMENTO*

NOMBRE CAMPO = cNomCampo

nPagEspacLineas = [1 | 2]

nPagEstiloImpresion = [NEGRITAS | ITALICO | CONDENSADO]

nPagNoColumnas = [1..132]

nPagNoRenglones = [1..N] * N SIGNIFICA SIN LIMITE *

nPosX = nX

nPosY = nY

nPrimerRenglon = *INDICA EL NUMERO DE LINEA DE LA DEFINICIÓN DEL REPORTE QUE APARECE EN EL PRIMER RENGLÓN DE LA PANTALLA*

NUEVA POSICION = POSICION MODIFICADA

NUMERO REPORTE = [1..99]

nX = * COORDENADA POSICION EN X*

nY = * COORDENADA POSICION EN Y*

OPCIONES GENERADOR DE REPORTES = [ARCHIVO | EDITOR | CAMPOS | VARIABLES | GRUPOS | OPCIONES | AYUDA]

SECUENCIAL = *NUMERO PROGRESIVO*

II.7.3 ESTRUCTURAS DE DATOS

stCampo = cNomTabla + cNombre + cTipo

stCampoVar = cNomCampo + cNomTabla + cExpresion + cTipoCampo + cFormato + cJustificacion]

stCodigoC = cAccion + cCodigo

stDefRep = cTituloReporte + cNomAutor + dFechaCreacion + cNomModifico + dFechaActualizacion

stElemGpo = nX + nY + nLongitud + cNomGrupo + cTipoGrupo + cArea + cTipoUnion + [cTexto | stCampoVar]

stPosicion = nPrimerRenglon + nPosX + nPosY

stTablaSimb = @cIdClaseDato + @cNomTabla + @cNomDato + @cIdArea + cIdTipoTabla + cTipoDato + cNomGrupo + cCampo + cExpresion + nLongitud

stTablsHij = @cNomTablaHija + @cCampoLlavePadre + cCampoLlaveHija

II.7.4 TABLAS

CAMPOS = { stCampo }

DEF_REP = { stDefRep }

DEFAULT.L = { stElemGpo }

ELEM_GPO = { stElemGpo }

NOMREP.LX = stDefRep + { stElemGpo } + { stTablsHij }

TBLSHIJ = { stTablsHij }

DATOSREP = { stDatosRep }

CODIGO_C = { stCodigoC }

TABLA_HIJA = { CAMPO }

TABLA_PADRE = { CAMPO}

TABLA_SIMB = { stTablaSimb }

II.7.5 FLUJOS DE DATOS

CAMPO ALFANUMERICO = DATOS CAMPO

CAMPO FECHA = DATOS CAMPO

CAMPO NUMERICO = DATOS CAMPO

CAMPO TABLA PADRE = CAMPO

CARACTERISTICAS CAMPO = SELECCION CAMPO + MOVIMIENTO EN FORMATO REPORTE

CARACTERISTICAS LINEA = POSICION EN FORMATO

CARACTERISTICAS NUEVO FORMATO = POSICION EN FORMATO + SELECCION FORMATO CAMPO

CARACTERISTICAS PAGINA = nPagNoColumnas + nPagNoRenglones + nPagEspcLineas

CARACTERISTICAS PARA IMPRESION = SELECCION IMPRESORA + DATOS PARA IMPRESION

CARACTERISTICAS VARIABLE = CARACTERISTICAS VARIABLE RESUMEN + CARACTERISTICAS VARIABLE FECHA + CARACTERISTICAS VARIABLE NO. PAGINA

CARACTERISTICAS VARIABLE FECHA = POSICION EN FORMATO +FORMATO FECHAS

CARACTERISTICAS VARIABLE NO. PAGINA = POSICION EN FORMATO + FORMATO NUMERICO

CARACTERISTICAS VARIABLE RESUMEN = SELECCION CAMPO + MOVIMIENTO EN FORMATO REPORTE

DATOS CAMPO = NOMBRE CAMPO + LONGITUD CAMPO + TIPO CAMPO

DATOS CAMPO SELECCIONADO = CAMPO + TIPO CAMPO + NOMBRE TABLA

DATOS DESCRIPTIVOS MODIFICADOS = DATOS DESCRIPTIVOS

DATOS ELEMENTO = ELEMENTO GRUPO

DATOS ELEMENTO CARACTER BORRADO = cTexto + POSICION MODIFICADA

DATOS ELEMENTO ENCABEZADO = [LIMITE GRUPO | { cTexto }] + cNomGrupo + cArea

DATOS ELEMENTO MODIFICADO = DATOS ELEMENTO CARACTER BORRADO + NUEVA POSICION + DATOS TEXTO MODIFICADO

DATOS ELEMENTO NO INCLUIDO EN VENTANA = ELEMENTO GRUPO

DATOS ELEMENTO SELECCIONADO = POSICION + ELEMENTO GRUPO

DATOS ELEMENTO TOTALES = [LIMITE GRUPO | { cTexto }] + cNomGrupo + cArea

DATOS GENERALES REPORTE = DATOS NUEVO REPORTE + OPCION REPORTE EXISTENTE

DATOS MODIFICA TEXTO = cTexto + MOVIMIENTO EN FORMATO REPORTE

DATOS NUEVO ELEMENTO = ELEMENTO GRUPO

DATOS NUEVO REPORTE = SELECCION NOMBRE REPORTE + DATOS DESCRIPTIVOS

DATOS PARA IMPRESION = nPagEstiloImpresion + nPagFormaContinua + cImpPuerto + cImpCadena Config

DATOS PRIMER ELEMENTO NUEVO GRUPO = ELEMENTO GRUPO

DATOS PRIMER ELEMENTO REPORTE = ELEMENTO GRUPO *DONDE X = 1 y Y = 1 *

DATOS RELACION TABLA HIJA = SELECCION TABLA HIJA + SELECCION CAMPO TABLA PADRE

DATOS RELACION TABLAS = DATOS RELACION TABLA HIJA + SELECCION TABLA HIJA

DATOS REPORTE EXISTENTE = SELECCION REPORTE EXISTENTE + DATOS DESCRIPTIVOS MODIFICADOS

DATOS TEXTO MODIFICADO = cTexto + POSICION MODIFICADA

DEFINICION CAMPO = nNoCampos + cNomTabla + { NOMBRE Y TIPO DE CAMPOS }

DEFINICION REPORTE = stDefRep

ELEMENTO GRUPO = stAtribElem

ELEMENTO MODIFICADO = ELEMENTO GRUPO

FORMATO DE DISEÑO = OPCIONES GENERADOR DE REPORTES + FORMATO EDITOR

FORMATO EDITOR = REGLA POSICION + AREA DE DISEÑO + BARRA ESTADO

FUNCION SELECCIONADA = FUNCION RESUMEN

INFORMACION OPCIONES = CARACTERISTICAS PARA IMPRESION + CARACTERISTICAS PAGINA + CARACTERISTICAS LINEA

INFORMACION PARA EDICION DE CAMPO = CARACTERISTICAS CAMPO + POSICION EN FORMATO + CARACTERISTICAS NUEVO FORMATO + DATOS RELACION TABLAS

INFORMACION PARA EDICION DE GRUPO = SELECCION CAMPO + POSICION EN FORMATO

INFORMACION PARA EDICION DE VARIABLES = CARACTERISTICAS VARIABLE + POSICION EN FORMATO

INFORMACION PARA EDICION DE TEXTO = MOVIMIENTO EN FORMATO REPORTE + DATOS MODIFICA TEXTO

NOMBRE REPORTE = NOMBRE TABLA + EXTENSION REPORTE

NOMBRE TABLA SELECCIONADA = cNomTabla

NOMBRE TABLA = cNomTabla

NOMBRE Y TIPO DE CAMPOS = cNombre + cTipo

NUEVA POSICION = POSICION

NUEVO ELEMENTO ALFANUMERICO = ELEMENTO GRUPO

NUEVO ELEMENTO FECHA = ELEMENTO GRUPO

PLANTILLA CAMPO = cFormato

PLANTILLA VARIABLE= cFormato

POSICION = nX + nY

POSICION EN FORMATO = POSICION

POSICION MODIFICADA = POSICION

REPORTE EXISTENTE = { NOMBRE REPORTE }

SELECCION CAMPO = cNomCampo

SELECCION CAMPO TABLA PADRE = cNomCampo

SELECCION IMPRESORA = nImpresora

SELECCION NOMBRE REPORTE = NOMBRE REPORTE

SELECCION REPORTE EXISTENTE = NOMBRE REPORTE

SELECCION TABLA HIJA = NOMBRE TABLA

TIPO CAMPO = cTipoCampo

VISUALIZACION CAMPO = VISUALIZACION NUEVO CAMPO + VISUALIZACION CAMPO BORRADO + VISUALIZACION NUEVO FORMATO

VISUALIZACION CAMPO AÑADIDO = VISUALIZACION NUEVO CAMPO + VISUALIZACION CAMPO MODIFICADO

VISUALIZACION DATOS DESCRIPTIVOS = DATOS DESCRIPTIVOS

VISUALIZACION DIGITOS NUMERO = DIGITOS NUMERO

VISUALIZACION ELEMENTO MODIFICADO = cFormato

VISUALIZACION ELEMENTO TEXTO = cTexto

VISUALIZACION FORMATO CAMPO = cFormato

VISUALIZACION GRUPO = { LIMITE GRUPO } + { cTexto }

VISUALIZACION GRUPO = VISUALIZACION NUEVO GRUPO + VISUALIZACION GRUPO ELIMINADO

VISUALIZACION GRUPO ELIMINADO = { cTexto } *TEXTO EN BLANCO*

VISUALIZACION NUEVA VARIABLE = DIGITOS NUMERO

VISUALIZACION NUEVO CAMPO = cFormato

VISUALIZACION NUEVO GRUPO = VISUALIZACION GRUPO

VISUALIZACION REPORTE = {cTexto} + {PLANTILLA CAMPO} + {PLANTILLA VARIABLE} + {LIMITE GRUPO}

VISUALIZACION REPORTE EXISTENTE = VISUALIZACION DATOS DESCRIPTIVOS + VISUALIZACION REPORTE

VISUALIZACION TEXTO = VISUALIZACION ELEMENTO TEXTO + VISUALZACION REPORTE

VISUALIZACION VARIABLE = VISUALIZACION NUEVA VARIABLE + VISUALIZACION VARIABLE BORRADA

VISUALIZACION VARIABLE BORRADA = cTexto *TEXTO EN BLANCO*

II.8 MINIESPECIFICACIONES

Una vez que ha sido representado el dominio de la información del "Generador de Reportes" (usando DFD's y el diccionario de datos), se describe cada función (transformación) representada, adecuando la definición de las funciones a estructuras fácilmente programables. Las miniespecificaciones consisten en verbos imperativos y términos definidos en el diccionario de datos. Su sintaxis está limitada a oraciones declarativas, estructuras de edición y estructuras de iteración.

Las estructuras básicas de las miniespecificaciones son:

- Estructura SI-ENTONCES-OTRO

```
SI condición -1
  frase-1
OTRO
  frase-2
FIN SI
```

- Estructura CASO

```
HACER CASO
  CASO variable = valor-1
        frase-1
  .
  .
  CASO variable = valor-n
        frase-n
  OTRO
        frase-n+1
FIN CASO
```

- Estructura HACER-MIENTRAS

```
HACER-MIENTRAS condición-1
  frase-1
FIN HACER
```

- Estructura REPITE-HASTA

```
REPITE
  frase-1
HASTA condición-1
```

A continuación se presentan algunos ejemplos de las miniespecificaciones desarrolladas para el "Generador de Reportes".

Para el módulo del editor:

GENERADOR DE REPORTES

```
1.0 Inicializa ambiente
REPITE
   Pedir opción al usuario
   HACER CASO
      CASO: opción = Archivo
         3.0 Maneja archivos de definición, pasa NOMBRE REPORTE,
         POSICION
      CASO: opción = Editor
         4.0 Edita texto del reporte, pasa POSICION
      CASO: opción = Campos
         5.0 Edita campos del reporte, pasa NOMBRE TABLA,
      POSICION
      CASO: opción = Variables
         6.0 Edita variables del reporte, pasa NOMBRE TABLA,
      POSICION
      CASO: opción = Grupos
         7.0 Edita grupos del reporte, pasa POSICION
      CASO: opción = Opciones
         8.0 Modifica opciones del reporte, pasa POSICION
      CASO: opción = Código
         9.0 Genera código del reporte, pasa NOMBRE REPORTE
      CASO: opción = Ayuda
         10. Proporciona ayuda
   FIN CASO
HASTA opción = salir
Cerrar ambiente

1.0 INICIALIZA AMBIENTE
1.2 Crear tablas de definición
1.3 Visualiza menú principal
1.4 Inicializa editor

1.2 CREAR TABLAS DE DEFINICION
Crea tabla TABLSHIJ
Crea tabla DEF_REP
```

```
1.3 VISUALIZA MENÚ PRINCIPAL
Desplegar marco general del editor
Mostrar el menú princial con las opciones:
Archivo, Editor, Campos, Variable, Grupos, Opciones, Código y Ayuda

1.4 INICIALIZA EDITOR
Desplegar la pantalla contenida en PANTEDIT
```

Para el módulo del Generador de Código:

9.2 CREA TABLA DE SIMBÓLOS

```
9.2.1 IDENTIFICA TIPO DE REGISTRO
Leer el primer elemento de ELEM_GPO
Grupo = Grupo del elemento
REPITE
   HACER CASO
      CASO: cTipoUnion = Campo
      CASO: cTipoUnion = campo tabla Hija
         9.2.2 Añade Registro Campo pasa DATOS ELEMENTO
      CASO: cTipoUnion = Variable
      CASO: cTipo Union = variable tabla hija
         9.2.3 Añade Registro Variable pasa DATOS ELEMENTO
   FIN CASO
   SI Grupo del elemento es diferente de Grupo
      9.2.5. Añade Registro Campo Grupo pasa NOMBRE TABLA
      Grupo = Grupo del elemento
   FIN SI
   Leer el siguiente elemento de ELEM_GPO
HASTA haber recorrido todos los elementos de ELEM_GPO
9.2.5 Añade Registro Campo Grupo pasa NOMBRE TABLA
9.2.6 Añade Registro Tabla pasa NOMBRE TABLA

9.2.2 AÑADE REGISTRO CAMPO
Recibir DATOS ELEMENTO
Guardar DATOS ELEMENTO en TABLA_SIMB identificándolos como un -
campo

9.2.3 AÑADE REGISTRO VARIABLE
Recibir DATOS ELEMENTO
Guardar DATOS ELEMENTO en TABLA_SIMB identificándolos como una -
variable
```

```
9.2.4 AÑADE REGISTRO GRUPO
Recibir NOMBRE TABLA
Crear una llave para identificar el grupo
Guardar datos del grupo en TABLA_SIMB identificándolos como un grupo

9.2.5 AÑADE REGISTRO CAMPO GRUPO
Recibir NOMBRE TABLA
Buscar en TABLA_SIMB el primer grupo definido
REPITE
   SI existe
      Buscar en TABLA_PADRE ( NOMBRE TABLA) el campo por el cual se
      definió el grupo
      Guardar los datos del campo en TABLA_SIMB identificándolos como
      un campo
   FIN SI
   Leer el siguiente grupo de TABLA_SIMB
HASTA haber revisado todos los grupos definidos en TABLA_SIMB

9.2.6 AÑADE REGISTRO TABLA
Recibir NOMBRE TABLA
Guardar los datos de la tabla padre (NOMBRE TABLA) en TABLA_SIMB
identificándolos como una tabla
Buscar en TABLSHIJ la primera tabla hija definida
REPITE
   SI existe
      Guardar los datos de la tabla hija en TABLA_SIMB identificándolos
      como una tabla
   FIN SI
   Leer la siguiente tabla hija de TABLSHIJ
HASTA haber revisado todas las tablas hijas definidas en TBLSHIJ
```

CAPÍTULO III

DISEÑO ESTRUCTURADO

INTRODUCCIÓN

Para realizar el diseño del "Generador de Reportes" se siguieron los siguientes pasos los cuales se presentan en este capítulo.

- Revisión de la metodología del Diseño Estructurado.
- Aplicar la metodología del Diseño Estructurado para realizar el diseño del generador de reportes que cumpla con los objetivos presentados en el capítulo anterior.

II.1 DISEÑO ESTRUCTURADO

II.1.1 INTRODUCCIÓN

El diseño es el proceso iterativo de tomar el modelo lógico del nuevo sistema junto con los objetivos del mismo y producir la especificación de un sistema físico que cumpla con esos objetivos. Esto se logra mediante un conjunto de consideraciones generales y técnicas para diseño de programas, que facilitan la codificación, depuración y modificación de estos, a través, de un proceso que reduce la complejidad, mediante la separación de las funciones de un programa en módulos relativamente independientes.

II.1.2 LAS METAS DEL DISEÑO ESTRUCTURADO

Diseñar programas como:

- estructuras independientes
- módulos que realizan una sola función

Las características de los programas resultantes del diseño estructurado serán:

- los programas son más simples
- los módulos se pueden codificar independientemente
- el programa se puede entender pieza por pieza
- las pruebas se facilitan
- los efectos colaterales de un cambio se reducen
- se reduce la cantidad de errores

II.1.3 ACOPLAMIENTO Y COHESIVIDAD

Dos medidas para conseguir la independencia entre módulos son: acoplamiento y cohesividad.

El **acoplamiento** es una medida de que tan estrecha es una conexión entre dos módulos. Entre menos y más simples sean las conexiones más fácil será entender los módulos sin hacer referencia a otros. Un alto grado

de acoplamiento complica al sistema ya que un módulo es más difícil de entender, cambiar o corregir si esta altamente interrelacionado con otros.

La **cohesividad**, es la interrelación de los elementos de un mismo módulo, elementos, instrucciones y datos.

Para lograr máxima independencia entre módulos:

- minimizar acoplamiento
- maximizar cohesividad

A continuación se presenta la escala de cohesividad, de menor a mayor:

1. coincidental
2. lógica
3. temporal
4. por comunicación
5. secuencial
6. funcional

La escala no es lineal. La cohesividad funcional es más fuerte que el resto, y las dos primeras son mucho más débiles que las demás.

Coincidental: cuando no existe ninguna relación significativa entre los elementos de un módulo.

Lógica: existe alguna relación lógica entre los elementos de un módulo, esto es, hacen cierta clase de funciones, como por ejemplo, altas, bajas y cambios.

Temporal: es igual que la lógica, excepto que los elementos también están relacionados en el tiempo. Esto es, los elementos son ejecutados en un mismo periodo de tiempo.

Por Comunicación: tiene elementos que hacen referencia a los mismo datos de entrada/salida.

Secuencial: cuando los datos de salida de un elemento son usados como entrada para el siguiente elemento.

Funcional: todos los elementos están relacionados para realizar una sola función.

II.1.4 DIAGRAMAS DE ESTRUCTURA Y DESCRIPCIÓN DE ENTRADAS Y SALIDAS

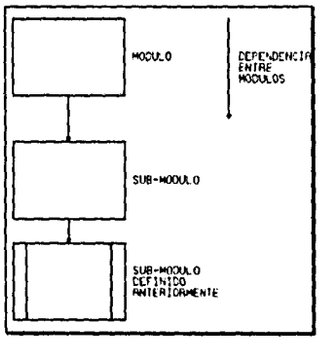
El objetivo del diseño estructurado es construir programas como estructuras de una sola función, compilados por separado, con nombres apropiados que estén acoplados por datos a través del menor número de parámetros posibles

La herramienta del diseño estructurado es el diagrama de estructura.

El proceso de diseño consiste en construir una estructura que cumpla con las especificaciones del sistema y usar los conceptos de acoplamiento y cohesividad para evaluar el diseño paso por paso.

Los elementos que forman parte de un diagrama de estructura son los que se muestran en la siguiente figura.

NOTACIÓN DE LOS DIAGRAMAS DE ESTRUCTURA



La descripción de datos de acoplamiento entre módulos (entradas y salidas) está íntimamente relacionada con los diagramas, por lo que a cada módulo o submódulo se le asocia un número con el que posteriormente se puedan especificar las entradas y salidas.

II.2 PLANTEAMIENTO DE LA ESTRUCTURA DE SOFTWARE

II.2.1 LIMITACIONES Y ALCANCES DEL DISEÑO

El diseño abarcará la creación de los dos módulos del Generador de Reportes: Editor y Generador de Código e incluirán los todos los objetivos descritos en el análisis.

El "Generador de Reportes" se desarrollará en el lenguaje de programación "C", y se utilizarán módulos de rutinas ya existentes para el manejo de menús y ventanas. Por lo tanto el diseño estará enfocado a aprovechar al máximo las librerías que nos ofrece el lenguaje de programación y las rutinas.

El análisis del módulo Editor, del "Generador de Reportes" fue realizado con el fin de que tenga la capacidad de incluir campos de una tabla sin especificar una base de datos particular, sin embargo en el diseño se desarrollan únicamente los módulos para interactuar con la base de datos Paradox a través de la funcionalidad permitida por el producto Paradox Engine 2.0.

El análisis del módulo generador de código, del "Generador de Reportes" fue enfocado al lenguaje de programación "C", por lo que consecuentemente, el diseño también lo estará.

II.2.2 REVISIÓN DE LAS ESTRUCTURAS DE DATOS

Al realizar una revisión de las estructuras de datos definidas en el análisis se llegaron a las siguientes conclusiones:

- El campo cTexto de la estructura stAtribElem será un apuntador tipo texto, con lo cual se le podrá asignar únicamente el espacio en memoria requerido (entre 1 a 80 caracteres).

  stAtribElem = nX + nY + nLongitud + cNomGrupo + cTipoGrupo + cArea + cTipoUnion + [*cTexto | stCampoVar]

- El almacenamiento de datos CAMPOS se definirá de la siguiente manera:

  - primero una estructura stCampo que contiene los datos de cada tabla abierta por el usuario la cual apunta a la estructura stNomTipo.

    stCampo = *stNomTipo + nNoCampos + nNoTabla + cNomTabla

- la estructura stNomTipo contiene los datos de cada campo de la tabla y es una estructura dinámica ya que apunta al siguiente campo, formando asi una lista.

  stNomTipo = nFldHndl + cNombre + cTipo + *apSiguiente

- finalmente apCampos será un arreglo de stCampo (en el DD era una tabla), con lo cual cada tabla leida será un elemento de apCampos y las caracteristicas de todos los campos que contengan quedarán almacenados en la lista formada por stNomTipo

  struct stCampo *apCampos[knMaxTbls]

- El almacenamiento de datos ELEM_GPO se definirá de la siguiente manera:

  - a la estructura stCampoVar se le agregará un nuevo campo nNoCampo

    stCampoVar= cNomCampo + cNomTabla + cExpresion + cTipoCampo + nNoCampo + cFormato + cJustificacion

  - en segunda lugar se declarará una estructura de datos que contiene toda la información del elemento (igual a la definida en el DD como stElemGpo)

    stAtribElem= nX + nY + nLongitud + cNomGrupo + cTipoGrupo + cArea + cTipoUnion + [cTexto | stCampoVar]

  - en consecuencia cambiará la definición de las siguientes tablas:

    DEFAULT.L = { stAtribElem }

    NOMREP.LX = stDefRep + { stAtribElem } + { stTabisHij }

  - finalmente se definirá stElemGpo como una estructura dinámica (en el DD se indicaba como una tabla), mediante la cual podamos formar una lista doblemente ligada en memoria y ahí se maneje en una manera óptima la definición del reporte que realice el usuario.

    stElemGpo = stAtribElem + *apAnterior + *apSiguiente

- A la estructura stPosicion se le agregará un nuevo campo que es un apuntador a la estructura stElemGpo, con el cual se apuntará al elemento de la lista que es el primer elemento que aparece en la pantalla. Además de el campo cInserta.

  stPosicion = *stElemGpo + nPrimerRenglon + nPosX + nPosY + cInserta

- A la estructura stTablaSimb se le agregará el campo nFldHndl

  stTablaSimb= @cIdClaseDato + @cNomTabla + @cNomDato + @cIdArea + cIdTipoTabla + cTipoDato + cNomGrupo + cCampo + cExpresion + nFldHndl + nLongitud

A la estructura stTablsHij se le agregará el campo nNumeroCampo

stTablsHij = @cNomTablaHija + @cCampoLlavePadre + cCampoLlaveHija + nNumeroCampo

- La definición de los **nuevos atributos** es la siguiente:

  cInserta = [SI | NO]

  nNoTabla = *NÚMERO PROGRESIVO ASIGNADO A CADA TABLA QUE EL USUARIO ABRE EN UN REPORTE*

  nFldHndl = *FIELD HANDLE QUE ASIGNA PARADOX A CADA CAMPO*

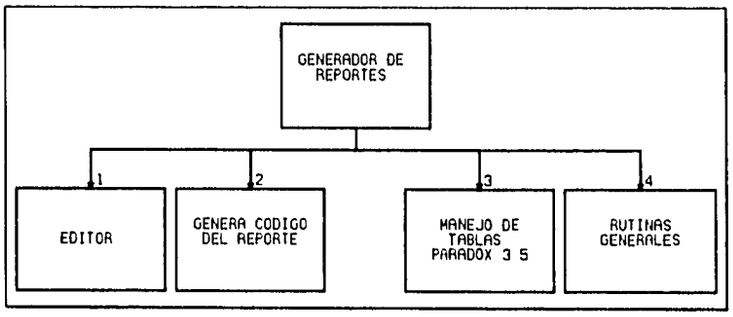
  nNoCampo = *FIELD HANDLE QUE ASIGNA PARADOX A CADA CAMPO*

  nNoCampos = *NÚMERO DE CAMPOS QUE CONTIENE LA TABLA*

  nNumeroCampo = *FIELD HANDLE QUE ASIGNA PARADOX Y QUE CORRESPONDE AL CAMPO cCampoLlaveHija*

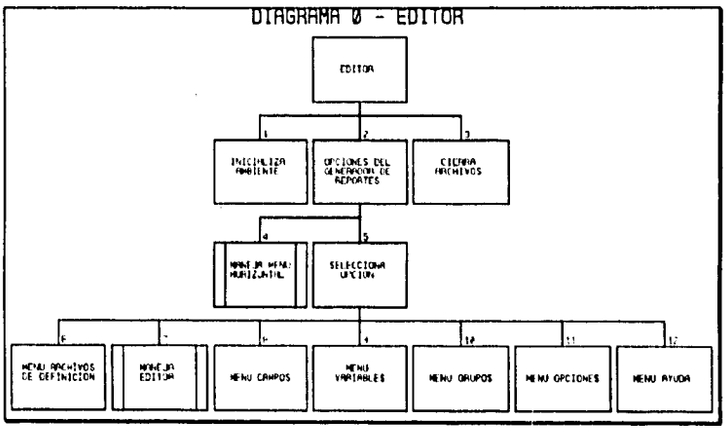
II.2.3 DIAGRAMAS DE ESTRUCTURA DEL "GENERADOR DE REPORTES"

A continuación se presentan los principales diagramas de estructura del "Generador de Reportes", junto con la descripción de entradas y salidas correspondientes. El conjunto completo se encuentra en el documento *Anexo Apéndice "C".*

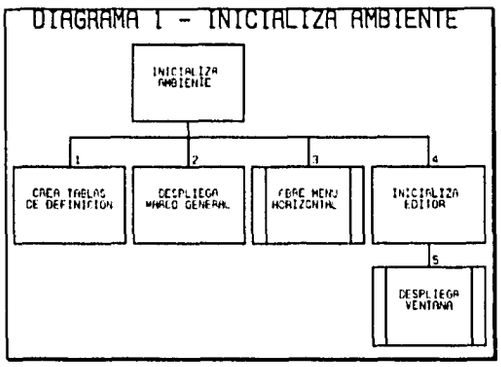


M.	ENTRADAS	SALIDAS
1.		LISTA ELEMENTO GRUPO, NOMBRE TABLA PADRE, NÚMERO DEL REPORTE
2	LISTA ELEMENTO GRUPO, NOMBRE TABLA PADRE, NÚMERO DEL REPORTE	
3		
4.		

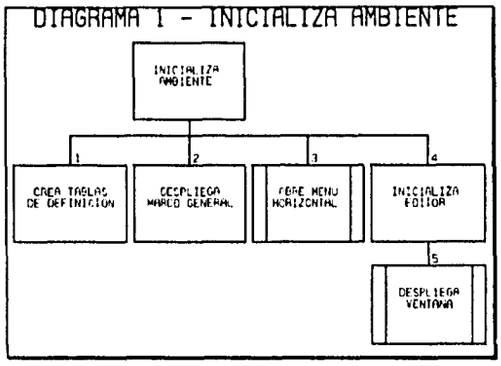
II.2.3.1 DIAGRAMAS DE ESTRUCTURA DEL MÓDULO EDITOR



M.	ENTRADAS	SALIDAS
1	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN INICIALIZADO
2.		
3.		
4.		OPCIÓN
5.		
6.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO, NOMBRE TABLA PADRE, NÚMERO DEL REPORTE
7.	REGISTRO DE POSICIÓN, ACCIÓN EDITOR	CÓDIGO TECLA, REGISTRO DE POSICIÓN ACTUALIZADO
8.	NOMBRE TABLA PADRE, REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
9.	NOMBRE TABLA PADRE, REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
10.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
11.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
12.		

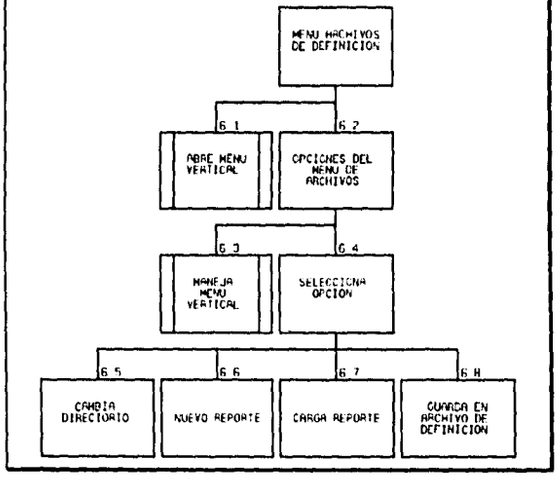


M.	ENTRADAS	SALIDAS
1.		
2.		
3.	OPCIONES DEL MENÚ DEL EDITOR	
4.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
5.	NOMBRE VENTANA, ACTIVA SOMBRA VENTANA	

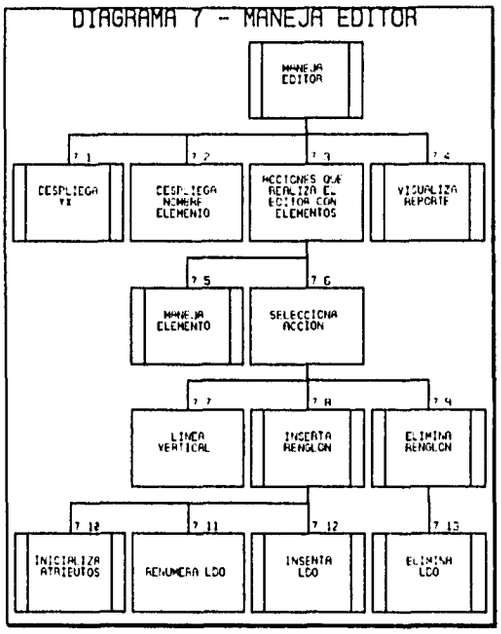


M.	ENTRADAS	SALIDAS
1.		
2.		
3.	OPCIONES DEL MENÚ DEL EDITOR	
4.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
5.	NOMBRE VENTANA, ACTIVA SOMBRA VENTANA	

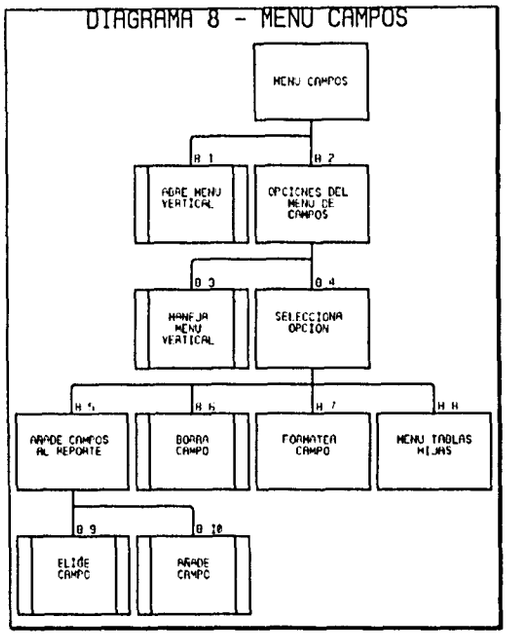
DIAGRAMA 6 - MENU ARCHIVOS DE DEFINICION



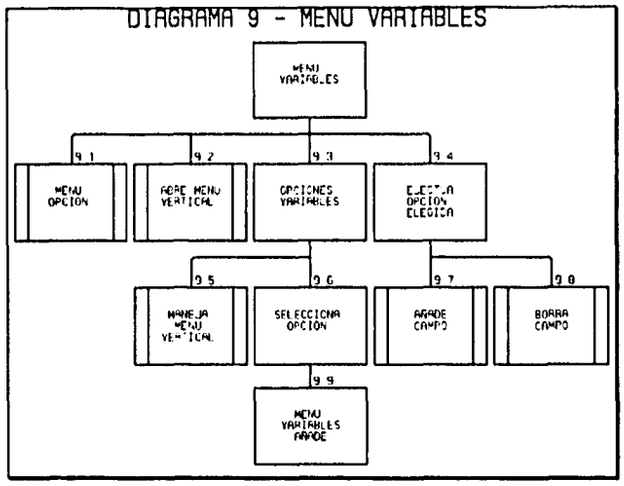
M.	ENTRADAS	SALIDAS
1.	OPCIONES DEL MENÚ ARCHIVOS	
2.		
3.		OPCIÓN
4.		
5.	NOMBRE DIRECTORIO	NOMBRE DIRECTORIO MODIFICADO
6.	REGISTRO DE POSICIÓN, NÚMERO TABLAS ABIERTAS	REGISTRO DE POSICIÓN INICIALIZADO, NOMBRE TABLA PADRE, NÚMERO REPORTE
7.	REGISTRO DE POSICIÓN, NÚMERO TABLAS ABIERTAS	REGISTRO DE POSICIÓN INICIALIZADO, NOMBRE TABLA PADRE, NÚMERO REPORTE
8.	NOMBRE TABLA PADRE, NÚMERO DEL REPORTE, NÚMERO TABLAS ABIERTAS	



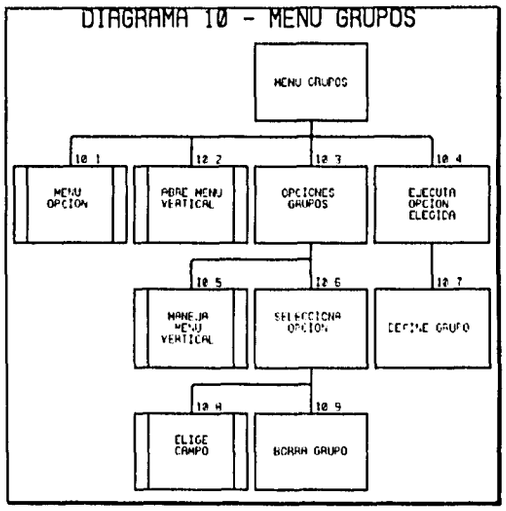
M.	ENTRADAS	SALIDAS
1.	POSICIÓN X,Y	
2.	REGISTRO ELEMENTO GRUPO ACTUAL	
3.		
4.	REGISTRO ELEMENTO GRUPO ACTUAL, REGISTRO DE POSICIÓN	
5.	REGISTRO ELEMENTO GRUPO ACTUAL, REGISTRO DE POSICIÓN, ACCIÓN	REGISTRO ELEMENTO GRUPO ACTUALIZADO, REGISTRO DE POSICIÓN ACTUALIZADO
6.		
7.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
8.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
9.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
10.	NUEVO REGISTROELEMENTO GRUPO	REGISTRO ELEMENTO GRUPO INICIALIZADO
11.	REGISTRO ELEMENTO GRUPO SIGUIENTE	
12.	REGISTRO ELEMENTO GRUPO INICIALIZADO	
13.		



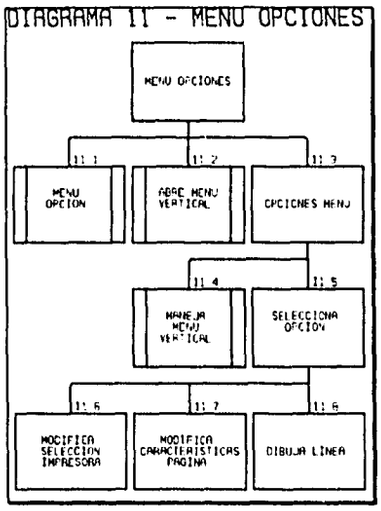
M.	ENTRADAS	SALIDAS
1.	OPCIONES DEL MENÚ CAMPOS	
2.		
3.		OPCIÓN
4.		
5.	NOMBRE TABLA PADRE, REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
6.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
7.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
8.		
9.	NÚMERO TABLAS ABIERTAS	DATOS DEL CAMPO SELECCIONADO
10.	NOMBRE TABLA PADRE, DATOS DEL CAMPO SELECCIONADO, REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO



M.	ENTRADAS	SALIDAS
1.		
2.	OPCIONES DEL MENÚ VARIABLES	
3.		
4.		
5.		OPCIÓN
6.		
7.	NOMBRE TABLA PADRE, DATOS NUEVA VARIABLE, REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
8.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO
9.		DATOS NUEVA VARIABLE

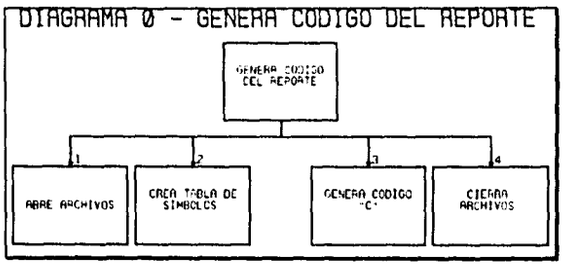


	ENTRADAS	SALIDAS
1.	OPCIONES DEL MENÚ GRUPOS	
2.		
3.		
4.		
5.		OPCIÓN
6.		
7.	REGISTRO DE POSICIÓN, DATOS DEL CAMPO SELECCIONADO	REGISTRO DE POSICIÓN ACTUALIZADO
8.		DATOS DEL CAMPO SELECCIONADO
9.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO

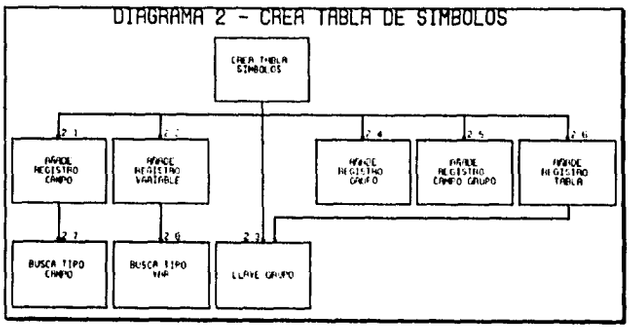


M.	ENTRADAS	SALIDAS
1.	OPCIONES MENÚ OPCIONES	
2.		
3.		
4.		OPCIÓN
5.		
6.		
7.		
8.	REGISTRO DE POSICIÓN	REGISTRO DE POSICIÓN ACTUALIZADO

DIAGRAMAS DE ESTRUCTURA DEL MÓDULO GENERADOR DE CÓDIGO



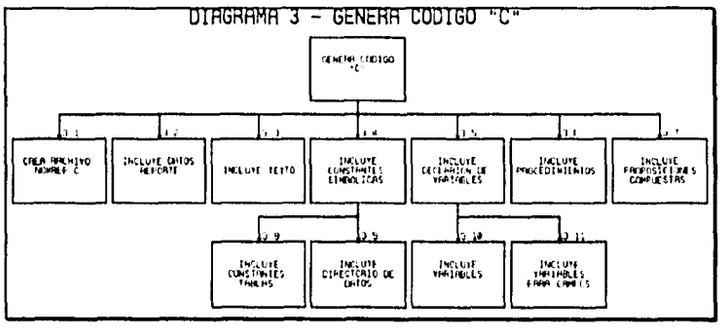
M.	ENTRADAS	SALIDAS
1.		
2.	NOMBRE TABLA PADRE	
3.	NOMBRE TABLA PADRE, NÚMERO DEL REPORTE	
4.		



M.	ENTRADAS	SALIDAS
1.	REGISTRO ELEMENTO GRUPO ACTUAL	
2.	REGISTRO ELEMENTO GRUPO ACTUAL, NÚMERO DE VARIABLE	
3.	NÚMERO GRUPO	NÚMERO GRUPO ACTUALIZADO, LLAVE GRUPO
4.	NOMBRE TABLA PADRE, LLAVE GRUPO, NOMBRE DEL GRUPO, AREA	
5.	NOMBRE TABLA PADRE	

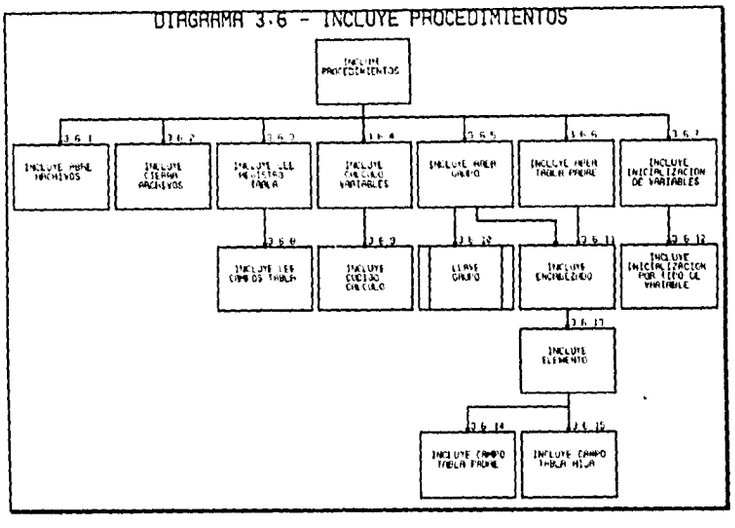
6.	NOMBRE TABLA PADRE	
7.	TIPO DEL CAMPO	DESCRIPCION DEL TIPO
8.	TIPO DE LA VARIABLE	DESCRIPCION DEL TIPO

DIAGRAMA 3 - GENERA CODIGO "C"



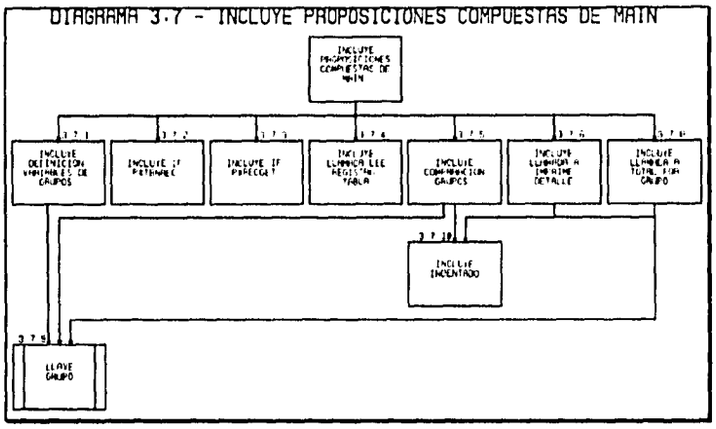
M.	ENTRADAS	SALIDAS
1.	NOMBRE TABLA PADRE, NÚMERO DEL REPORTE	
2.		
3.	CÓDIGO	
4.		
5.		
6.	NOMBRE TABLA PADRE, AREA	
7.	NOMBRE TABLA PADRE	
8.		
9.		
10.		
11		

DIAGRAMA 3.6 - INCLUYE PROCEDIMIENTOS



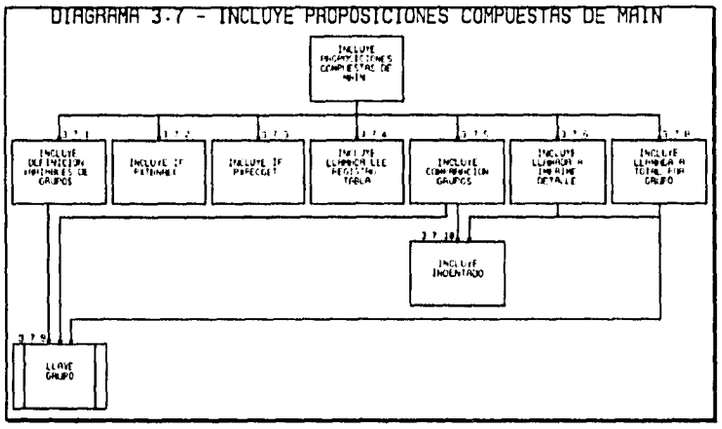
M.	ENTRADAS	SALIDAS
1.		
2.		
3.		
4.		
5.	NOMBRE TABLA PADRE, AREA	
6.	NOMBRE TABLA PADRE, AREA	
7.		
8.	NOMBRE TABLA	
9.	REGISTRO TABLA DE SIMBOLOS	
10.	NÚMERO GRUPO	NÚMERO GRUPO ACTUALIZADO, LLAVE GRUPO
11.	LLAVE GRUPO, AREA	
12.	REGISTRO TABLA DE SIMBOLOS	
13.	NÚMERO RENGLON	REGISTRO ELEMENTO GRUPO ACTUAL, NÚMERO RENGLON ACTUALIZADO
14.	REGISTRO ELEMENTO GRUPO ACTUAL	
15.	REGISTRO ELEMENTO GRUPO ACTUAL	

DIAGRAMA 3.7 - INCLUYE PROPOSICIONES COMPUESTAS DE MAIN



M.	ENTRADAS	SALIDAS
1.	NOMBRE TABLA PADRE	
2.	NOMBRE TABLA PADRE	
3.	NOMBRE TABLA PADRE	
4.		
5.	NOMBRE TABLA PADRE, NÚMERO GRUPO, NIVEL DE INDENTADO	NIVEL DE INDENTADO ACTUALIZADO
6.	NOMBRE TABLA PADRE, NIVEL DE INDENTADO	NIVEL DE INDENTADO ACTUALIZADO
7.		
8.	NOMBRE TABLA PADRE, NIVEL DE INDENTADO	NIVEL DE INDENTADO ACTUALIZADO
9.	NÚMERO GRUPO	NÚMERO GRUPO ACTUALIZADO, LLAVE GRUPO
10.	NIVEL DE INDENTADO	

DIAGRAMA 3.7 - INCLUYE PROPOSICIONES COMPUESTAS DE MAIN



M.	ENTRADAS	SALIDAS
1.	NOMBRE TABLA PADRE	
2.	NOMBRE TABLA PADRE	
3.	NOMBRE TABLA PADRE	
4.		
5.	NOMBRE TABLA PADRE, NÚMERO GRUPO, NIVEL DE INDENTADO	NIVEL DE INDENTADO ACTUALIZADO
6.	NOMBRE TABLA PADRE, NIVEL DE INDENTADO	NIVEL DE INDENTADO ACTUALIZADO
7.		
8.	NOMBRE TABLA PADRE, NIVEL DE INDENTADO	NIVEL DE INDENTADO ACTUALIZADO
9.	NÚMERO GRUPO	NÚMERO GRUPO ACTUALIZADO, LLAVE GRUPO
10.	NIVEL DE INDENTADO	

CAPÍTULO IV

PROGRAMACIÓN DEL SISTEMA

INTRODUCCIÓN

Todos los pasos de la ingeniería de software que se han presentado hasta ahora van dirigidos hacia un objetivo final: traducir las representaciones del software a un lenguaje de programación que, por último, es (automáticamente) transformado en instrucciones ejecutables por la computadora. Este es el paso de codificación - un proceso que transforma el diseño en un lenguaje de programación. Esta es la etapa en la que se conformará el "Generador de Reportes" y por lo tanto será la carta de presentación ante los usuarios.

El resultado de la programación es un producto tangible que nos podrá dar los parámetros para determinar si el análisis y diseño fueron adecuados, asimismo será relativamente fácil si un programa funciona correctamente.

En este capítulo se presentan las principales consideraciones que se tomaron para la programación del "Generador de Reportes" y algunos ejemplos de la codificación realizada.

IV.1 LENGUAJES DE PROGRAMACIÓN

Existen dos formas en las que se pueden clasificar los lenguajes de programación: por su nivel y por su aplicación.

Por su nivel podemos decir que existen cuatro grandes grupos: lenguajes de máquina, lenguaje ensamblador, lenguaje de alto nivel y lenguajes declarativos. Los lenguajes de alto nivel, en la actualidad, son los más ampliamente usados. Esto es debido principalmente a la característica de ser portátil, es decir, pueden ser transferidos de una máquina a otra. Por su aplicación existen cinco áreas: científica, procesamiento de datos, inteligencia artificial, procesamiento de texto y programación de sistemas.

El lenguaje seleccionado para el desarrollo del "Generador de Reportes" es C, clasificado como lenguaje de alto orden de propósito general. El lenguaje de programación C fue originalmente desarrollado como lenguaje para implementaciones de sistemas operativos. El sistema operativo Unix está implementado en C. Sin embargo, actualmente se ha construido una gran cantidad de productos de software, de aplicaciones empotradas y de software de sistemas, usando el lenguaje C.

Algunas de las razones específicas por las que se escogió C son las siguientes:

- *C soporta estructuras de datos sofisticas.*- Por ejemplo: la estructura dinámica que requerirá stAtribElem, en la cual se basa la lista en que se almacena la definición del reporte, es una estructura tipo unión la cual varia en longitud (dependiendo del tipo de elemento se conforma por la estructura T o C), y la cual se definió de la siguiente manera:

```
typedef struct {
        unsigned char  nX,nY  ;
        int            longitud;
        char           cNomGrupo [ mxTamNomCampo ],
                       cTipoGrupo,
                       cArea,
                       cTipoUnion;
                       ..... ☞
```

```
union {
        struct {
                char    *cTexto /*LA LONGITUD ES VARIABLE*/
                } T;
                /*ESTRUCTURA PARA ELEMENTOS TEXTO, GRUPO E
                INFORMATIVOS*/
        struct {
                char    cNomCampo [ mxTamNomCampo ],
                        cNomTabla [ mxTamNomTabla ],
                        cExpresion [ mxTamExpresion ];
                TipoCampo     cTipoCampo;
                int     nNoCampo;
                char    *cFormato, /*LA LONGITUD ES VARIABLE*/
                        cJustificacion;
                } C;
                /*ESTRUCTURA PARA ELEMENTOS CAMPO O
                VARIABLES */
        } EG
} stAtribElem;
/*ESTRUCTURA PARA ALMACENAR LOS ATRIBUTOS DE UN ELEMENTO*/
```

- tiene características de tipificación,
- *hace un uso intensivo de los apuntadores.-* los cuales se usan intensamente sobre todo en el módulo del editor, ya que con base a la estructura dinámica stElemGpo se forma una lista doblemente ligada para almacenar la definición del reporte.
- tiene un vasto conjunto de operadores para el cálculo y la manipulación de datos,
- permite al programador "acercarse a la máquina" al suministrar posibilidades similares al lenguaje ensamblador.

Particularmente se utilizará Borland C++ V.3.1. Es importante aclarar que no se hizo uso de las facilidades que da Borland C++ con respecto a la Programación Orientada a Objetos (OOP).

IV.2 ESTÁNDARES UTILIZADOS EN LA PROGRAMACIÓN

- Todas las palabras reservadas de C aparecen en letras minúsculas.
- Los comentarios aparecen en letras mayúsculas.
- Los nombres de los atributos siguen los estándares especificados en el diccionario de datos.
- Existen recuadros en los encabezados de cada módulo, declaraciones, procedimientos, funciones, etc., indicando en cada uno el nombre y uso de esa sección de código.
- Las sangrías son obligatorias y de cuatro espacios
- Las proposiciones de control de flujo son escritas de la siguiente manera

```
if ( expresión )
   proposición-1
else
   proposición-2
/*ENDIF*/

switch ( variable )
   {
   case valor-1:
          proposición-1
          break;
   .
   .
   case valor-n:
          proposición-n
          break;
   default:
          proposición-n+1
          break;
   }
/*ENDSWITCH*/

while ( expresión )
   proposición
/*ENDWHILE*/

for ( expresión-1;expresión-2;expresión-3)
   proposición
/*ENDFOR*/

do
   proposición
while ( expresión );
```

IV.4 PROGRAMAS

A continuación se presentan algunas secciones representativas de los dos módulos principales .Editor y Generador de Código, que componen el "Generador de Reportes".

FUNCIÓN PRINCIPAL main( )

```
/*-------------------------------------------------------------------------
     -----> FUNCION PRINCIPAL <-----
              GENERADOR DE REPORTES
 -------------------------------------------------------------------------*/
main()

  {
  char                    cCodigo,
                          cOpcion;
  struct stPosicion       stPosicion;
  char                    cTblPadre[13],
                          cNumRep[3];

  cCodigo                 = 0;
  cOpcion                 = '\0';
  nglNumTablas = 0;
  InicializaAmbiente();
  despliega_memoria();
  do
    {
    cOpcion= maneja_menu_horiz( cOpcion,TRUE,TRUE );
    switch ( cOpcion )
      {
      case '\0' :
        despliega_error( 25,"tecla invalida",125);
        break;
      case 'A':
        abre_menu_Archivo( cTblPadre,cNumRep,&stPosicion );
        break;
      case 'E':
        if ( nglNumTablas == 0 )
            despliega_mensaje( 24,"Utilice las opciones Archivo/Nuevo o Archivo/Cargar
antes de usar esta opción",TRUE,1500);
        else
          ManejaMovEditor( &cCodigo,&stPosicion,EDITATEXTO );
        /*ENDIF*/
        break;
      case 'C':
        abre_menu_Campos( cTblPadre,&stPosicion );
        break;
      case 'V':
```

```
            abre_menu_Variables( cTblPadre,&stPosicion );
            break;
          case 'G':
            abre_menu_Grupos( &stPosicion );
            break;
          case 'O':
            abre_menu_Opciones( &stPosicion );
            break;
          case 'Y':
            abre_menu_Ayuda();
            break;
          default :
            break;
          }
        /* ENDSWITCH */
        }
    while ( !fin);
    CierraAmbiente();
    }
/*FIN main*/
```

FUNCIONES DEL MÓDULO EDITOR

```
/*-------------------------------------------------------------------------------
    ****** RUTINAS RELACIONADAS CON LA LECTURA Y ESCRITURA DE ******
                LOS ARCHIVOS DE DEFINICION DE REPORTES
  -------------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------------
    ---> LEE EN UNA LDO LA DEFINICION DEL REPORTE QUE ESTA ALMACENADA
         EN EL ARCHIVO DE TEXTO DEL REPORTE
  -------------------------------------------------------------------------------*/
void LeeElemGpo(        FILE          *DefRep,      /*ENTRA*/
                        stAtribElem   *apAtrib      /*ENTRA*/ )

  {
  char clinea[ mxTamRenglon ],
       cformato[ 20 ],
       cNumero[  20 ];

  fgets( clinea,12,DefRep );
  sscanf( clinea,"%d %d %d",&(apAtrib->nX),&(apAtrib->nY),&(apAtrib->nLongitud) );
  fgets( clinea,26,DefRep );
  strcpy( apAtrib->cNomGrupo,clinea );
  fgets( clinea,6,DefRep );
  sscanf( clinea,"%c %c %c",&(apAtrib->cTipoGrupo),&(apAtrib->cArea),
          &(apAtrib->cTipoUnion) );
  switch ( apAtrib->cTipoUnion )
    {
    case 'T':
    case 'I':
```

```
        fgets( clinea,apAtrib->nLongitud,DefRep );
        if ( (apAtrib->EG.T.cTexto = farmalloc( apAtrib->nLongitud+1 )) == NULL )
           maneja_error_heap();
        /*ENDIF*/
        strcpy(apAtrib->EG.T.cTexto,clinea );
        break;
     case 'C':
     case 'H':
     case 'V':
     case 'v':
        fgets( clinea,mxTamNomCampo+1,DefRep );
        sscanf( clinea,"%s",apAtrib->EG.C.cNomCampo );
        fgets( clinea,mxTamNomTabla+1,DefRep );
        sscanf( clinea,"%s",apAtrib->EG.C.cNomTabla );
        fgets( clinea,mxTamExpresion+1,DefRep );
        sscanf( clinea,"%s",apAtrib->EG.C.cExpresion);
        fgets( clinea,2,DefRep );
        sscanf( clinea,"%d",&(apAtrib->EG.C.cTipoCampo) );
        fgets( clinea,4,DefRep );
        sscanf( clinea,"%d",&(apAtrib->EG.C.nNoCampo) );
        fgets( clinea,apAtrib->nLongitud+1,DefRep );
        if ( (apAtrib->EG.C.cFormato = farmalloc( apAtrib->nLongitud+1)) == NULL )
           maneja_error_heap();
        /*ENDIF*/
        sscanf( clinea,"%s",apAtrib->EG.C.cFormato );
        fgets( clinea,2,DefRep );
        sscanf( clinea,"%c",&(apAtrib->EG.C.cJustificacion) );
        break;
     }
  /*ENDSWITCH*/
  }
/*FIN LeeElemGpo */

/*-------------------------------------------------------------------------
   ---> LEE DEL ARCHIVO DE TEXTO LA DEFINICION DEL REPORTE
 -------------------------------------------------------------------------*/
char lLeeArchDef( char *cDirectorio,     /*ENTRA*/
                  char *cNomReporte, /*ENTRA */
                  int   *nNumTablas   /*ENTSAL*/ )
  {
  FILE          *DefRep;
  pathstr       cNomArchivo;
  struct stElemGpo  *apElemGpo    ,
                *apUltimoElem  ;

  int           nTipoReg      ;
  char          cTipoReg[ mxTamRenglon ]  ;
  char          cha[ mxTamRenglon ];

  strcpy( cNomArchivo,cDirectorio );
```

```
strcat( cNomArchivo,"\\");
strcat( cNomArchivo,cNomReporte );
DefRep = fopen( cNomArchivo,"rt" );
ngMaxRenglones = 0;
if ( DefRep != NULL )
   {
   (*nNumTablas)++;
   while ( !(feof( DefRep )) && ( fgets( cTipoReg,4,DefRep ) != NULL ) )
      {
      nTipoReg = atoi( cTipoReg );
      switch ( nTipoReg )
         {
         case ELEMGPO:
            apElemGpo = (struct stElemGpo *) farmalloc( sizeof( struct stElemGpo ) );
            if ( apElemGpo == NULL ) maneja_error_heap();
            LeeElemGpo( DefRep,&(apElemGpo->Atributos) );
            if ( (apElemGpo->Atributos.nX == 1 ) && ( apElemGpo->Atributos.nY == 1 ) )
               {
               apgPrimerElem = apElemGpo;
               apUltimoElem  = apElemGpo;
               }
            /*ENDIF*/
            apElemGpo->apSiguiente    = apgPrimerElem;
            apElemGpo->apAnterior     = apUltimoElem;
            apgPrimerElem->apAnterior = apElemGpo;
            apUltimoElem->apSiguiente = apElemGpo;
            apUltimoElem= apElemGpo;
            break;
         case DEFREP:
            break;
         case VARIABLS:
            break;
         case NOMTHIJS:
            LeeNomTblsHijs( DefRep,nNumTablas );
            break;
         case TABLSHIJ:
            lLeeTblsHijs( DefRep );
            break;
         }
      /*ENDSWITCH*/
      }
   /*ENDWHILE*/
   ngMaxRenglones = apUltimoElem->Atributos.nY;
   despliega_memoria();
   fclose( DefRep );
   return TRUE;
   }
else
   {
   despliega_error( 24,"El archivo de Definición No existe",1500 );
```

```
      return FALSE;
      }
   /*ENDIF*/
   }
/*FIN ILeeArchDef */

/*-------------------------------------------------------------------------------
    ----> ALMACENA LA LDO QUE CONTIENE LA DEFINICION DEL REPORTE
          EN EL ARCHIVO DE TEXTO DEL REPORTE
 -------------------------------------------------------------------------------*/
void GuardaElemGpo( FILE          *DefRep, /*ENTRA*/
                    stAtribElem   *apAtrib  /*ENTRA*/ )

   {
   fprintf( DefRep,"%3.3d",ELEMGPO );
   fprintf( DefRep,"%3.3d %3.3d %3.3d",apAtrib->nX,apAtrib->nY,apAtrib->nLongitud );
   fprintf( DefRep,"%25.25s",apAtrib->cNomGrupo );
   fprintf(DefRep,"%c %c %c",apAtrib->cTipoGrupo,apAtrib->cArea,apAtrib->cTipoUnion );
   switch ( apAtrib->cTipoUnion )
      {
      case 'T':
      case 'I':
          fprintf( DefRep,"%s",apAtrib->EG.T.cTexto );
          break;
      case 'C':
      case 'H':
      case 'V':
      case 'v':
          fprintf( DefRep,"%25.25s",apAtrib->EG.C.cNomCampo );
          fprintf( DefRep,"%9.9s",apAtrib->EG.C.cNomTabla );
          fprintf( DefRep,"%15.15s",apAtrib->EG.C.cExpresion );
          fprintf( DefRep,"%d",apAtrib->EG.C.cTipoCampo );
          fprintf( DefRep,"%3.3d",apAtrib->EG.C.nNoCampo );
          fprintf( DefRep,"%s",apAtrib->EG.C.cFormato );
          fprintf( DefRep,"%c",apAtrib->EG.C.cJustificacion );
          break;
      }
   /*ENDSWITCH*/
   }
/*FIN GuardaElemGpo */

/*-------------------------------------------------------------------------------
    ----> ALMACENA EN UN ARCHIVO DE TEXTO LA DEFINICION DEL REPORTE
 -------------------------------------------------------------------------------*/
void GuardaEnArchDef(   char *cNomTblPadre, /*ENTRA */
                        char *cNumReporte,  /*ENTRA */
                        int   nNumTablas    /*ENTRA*/ )

   {
   FILE                 *DefRep;
```

```
  pathstr                 cNomArchivo;
  struct stElemGpo        *apElemGpo;

  strcpy( cNomArchivo,cDirDatUsr );
  strcat( cNomArchivo,"\\");
  strcat( cNomArchivo,cNomTblPadre );
  strcat( cNomArchivo,".");
  strcat( cNomArchivo, cNumReporte );
  DefRep = fopen( cNomArchivo,"wt" );
  if ( DefRep != NULL )
     {
     apElemGpo = apgPrimerElem;
     do
        {
        GuardaElemGpo( DefRep,&(apElemGpo->Atributos) );
        apElemGpo = apElemGpo->apSiguiente;
        }
     while ( apElemGpo != apgPrimerElem );
     lGuardaTblsHijs( DefRep );
     GuardaNomTblsHijs( DefRep,nNumTablas );
     }
  else
     {
     despliega_error( 24,"No pude abrir el archivo de Definición",1500 );
     exit( 1 );
     }
  /*ENDIF*/
  fclose( DefRep );
  }
/*FIN GuardaEnArchDef */
```

```
  pathstr                 cNomArchivo;
  struct stElemGpo        *apElemGpo;

  strcpy( cNomArchivo,cDirDatUsr );
  strcat( cNomArchivo,"\\");
  strcat( cNomArchivo,cNomTblPadre );
  strcat( cNomArchivo,".");
  strcat( cNomArchivo, cNumReporte );
  DefRep = fopen( cNomArchivo,"wt" );
  if ( DefRep != NULL )
     {
     apElemGpo = apgPrimerElem;
     do
        {
        GuardaElemGpo( DefRep,&(apElemGpo->Atributos) );
        apElemGpo = apElemGpo->apSiguiente;
        }
     while ( apElemGpo != apgPrimerElem );
     lGuardaTblsHijs( DefRep );
     GuardaNomTblsHijs( DefRep,nNumTablas );
     }
  else
     {
     despliega_error( 24,"No pude abrir el archivo de Definición",1500 );
     exit( 1 );
     }
  /*ENDIF*/
  fclose( DefRep );
  }
/*FIN GuardaEnArchDef */
```

FUNCIONES DEL MÓDULO GENERADOR DE CÓDIGO

```
/*-------------------------------------------------------------------------------
        PRIMERA PASADA: CREACION DE DATOS PARA EL ARCHIVO TBLSIMB.DB
 -------------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------
  ----> AÑADE UN REGISTRO TIPO GRUPO EN LA TABLA DE SIMBOLOS      |
 -------------------------------------------------------------------------*/
char lAddRegGrupo(      char *cNomTblPadre,     /*ENTRA*/
                        char *cCveGrupo,        /*ENTRA*/
                        har cArea,              /*ENTRA*/
                        har *cNomGrupo          /*ENTRA*/ )

  {
  int  nError;
  char cTemp[ 3 ];

  strcpy( cTemp," ");
  cTemp[0] = cArea;
  strip( cNomGrupo );
  if ( ErrorPX(PXRecBufEmpty( aRecHandle[ KNTBLSIMB ] )) )
     return FALSE;
  /* ENDIF */
  nGuardaAlfaNum(&aRecHandle[KNTBLSIMB],&aTblTSimb[knIdClaseDato-1],cCveGrupo );
  nGuardaAlfaNum(&aRecHandle[KNTBLSIMB],&aTblTSimb[knNomTabla-1],cNomTblPadre);
  if ( strcmp( cNomGrupo,"") == 0 )
     nGuardaAlfaNum(&aRecHandle[KNTBLSIMB],&aTblTSimb[knNomDato-1],cNomTblPadre);
  else
     nGuardaAlfaNum(&aRecHandle[KNTBLSIMB],&aTblTSimb[ knNomDato-1 ], cNomGrupo );
  /*ENDIF*/
  nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdArea-1 ], cTemp );
  nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdTipoTabla-1 ], "P" );
  if(ErrorPX(PXRecAppend(aTblHandle[KNTBLSIMB],aRecHandle[KNTBLSIMB]))) return FALSE;
  return TRUE;
  }
/*FIN lAddRegGrupo*/

/*-------------------------------------------------------------------------
  ----> AÑADE UN REGISTRO TIPO CAMPO EN LA TABLA DE SIMBOLOS
 -------------------------------------------------------------------------*/
char lAddRegCampo( stAtribElem *apAtrib /*ENTRA*/ )

  {
  int nError;
  char cTemp[ 80 ],
       cTipoDato[ 13 ];

  if ( ErrorPX(PXRecBufEmpty( aRecHandle[ KNTBLSIMB ] )) )
     return FALSE;
  /* ENDIF */
  nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdClaseDato-1 ],"C" );
```

```
   nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knNomTabla-1 ],
                        apAtrib->EG.C.cNomTabla );
   if ( apAtrib->cTipoUnion == 'C' )
      {
      nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knNomDato-1 ],
                           apAtrib->EG.C.cNomCampo  );
      nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdTipoTabla-1 ], "P" );
      }
   else
      {
      strcpy( cTemp,apAtrib->EG.C.cNomTabla );
      strcat( cTemp,"_");
      strcat( cTemp,apAtrib->EG.C.cNomCampo );
      nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knNomDato-1 ], cTemp );
      nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdTipoTabla-1 ], "H" );
      }
   /*ENDIF*/
   BuscaTipoCampo( cTipoDato,apAtrib->EG.C.cTipoCampo );
   nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knTipoDato-1 ],cTipoDato );
   nGuardaDouble( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knFldHndl-1 ],
                        apAtrib->EG.C.nNoCampo );
   nGuardaDouble( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knLongitud-1 ],
                        apAtrib->nLongitud );
   if( ErrorPX(PXRecAppend( aTblHandle[ KNTBLSIMB ], aRecHandle[ KNTBLSIMB ] )))
      return FALSE;
   else
      return TRUE;
/*ENDIF*/   }
/*FIN lAddRegCampo*/

/*-------------------------------------------------------------------------------
  ---->  AÑADE UN REGISTRO TIPO VARIABLE EN LA TABLA DE SIMBOLOS
 -------------------------------------------------------------------------------*/
char lAddRegVariable(    stAtribElem       *apAtrib, /*ENTRA*/
                         int               *nNoVar  /*ENTSAL*/ )

   {
   int nError;
   char cTemp[    3 ],
      cTipoDato[ 13 ],
      cNomDato[  33 ],
      cNoVar[    10 ],
      cNomGrupo[ mxTamNomCampo+1 ];

   if ( strcmp( apAtrib->EG.C.cExpresion,"FechaReporte") == 0 ||
      strcmp( apAtrib->EG.C.cExpresion,"No.Página") == 0 )
     return;  /*SE TRATA DE UNA VARIABLE NO CALCULADA */
   /*ENDIF*/

   strcpy( cTemp," ");
```

```
   cTemp[0]  = apAtrib->cArea;
   strcpy( cNomGrupo,apAtrib->cNomGrupo );
   strip( cNomGrupo );
   if ( ErrorPX(PXRecBufEmpty( aRecHandle[ KNTBLSIMB ] )) )
      return FALSE;
   /* ENDIF */
   nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdClaseDato-1 ],"V" );
   nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knNomTabla-1 ],
                        apAtrib->EG.C.cNomTabla );
   if ( apAtrib->cTipoUnion == 'V' )
      {
      strcpy( cNomDato,"n");
      (*nNoVar)++;
      itoa( *nNoVar,cNoVar, 10);
      strcat( cNomDato,cNoVar );
      BuscaTipoVar( apAtrib->EG.C.cExpresion,cNomDato );
      strcat( cNomDato,apAtrib->EG.C.cNomCampo );
      nGuardaAlfaNum(&aRecHandle[KNTBLSIMB],&aTblTSimb[knNomDato-1 ],cNomDato );
      nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdTipoTabla-1 ], "P" );
      }
   else
      {
      nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdTipoTabla-1 ], "H" );
      }
   /*ENDIF*/
   nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdArea-1 ],cTemp );
   BuscaTipoCampo( cTipoDato,apAtrib->EG.C.cTipoCampo );
   nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knTipoDato-1 ],cTipoDato );
   nGuardaAlfaNum(&aRecHandle[KNTBLSIMB],&aTblTSimb[knNomGrupo-1],cNomGrupo );
   nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knCampo-1 ],
                        apAtrib->EG.C.cNomCampo );
   nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knExpresion-1 ],
                        apAtrib->EG.C.cExpresion );
   nGuardaDouble( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knLongitud-1 ],
                        apAtrib->nLongitud );
   if( ErrorPX(PXRecAppend( aTblHandle[ KNTBLSIMB ], aRecHandle[ KNTBLSIMB ] )))
      return FALSE;
   else
      return TRUE;
   /*ENDIF*/
   }
/*FIN lAddRegVariable*/

/*-------------------------------------------------------------------------------
  ----->  AÑADE UN REGISTRO TIPO TABLA EN LA TABLA DE SIMBOLOS
  -------------------------------------------------------------------------------*/
char lAddRegsTabla( char *cNomTblPadre /*ENTRA*/ )

   {
   int    nError;
```

```
  char  cNomTblHija[ mxTamNomTabla+1 ],
        cNomTbl[    mxTamNomTabla+1 ];

  if ( ErrorPX(PXRecBufEmpty( aRecHandle[ KNTBLSIMB ] )) )
     return FALSE;
  /* ENDIF */
  nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdClaseDato-1 ],"T" );
  nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knNomTabla-1 ],
                  cNomTblPadre );
  nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knNomDato-1 ],
                  cNomTblPadre );
  nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdTipoTabla-1 ], "P" );
  if( ErrorPX(PXRecAppend( aTblHandle[ KNTBLSIMB ], aRecHandle[ KNTBLSIMB ] )))
     return FALSE;
  /*ENDIF*/
  strcpy( cNomTbl,cNomTblPadre );
  if (ErrorPX(PXRecFirst( aTblHandle[ KNTABLSHIJ ]))) return FALSE;
  do
     {
     if(ErrorPX(PXRecGet(aTblHandle[KNTABLSHIJ],aRecHandle[KNTABLSHIJ])))
        return FALSE;
     /*ENDIF*/
     nLeeAlfaNum( &aRecHandle[ KNTABLSHIJ ],&aTblTablsHij[ knNomTablaHija-1 ],
                  cNomTblHija );
     if ( strcmp( cNomTbl,cNomTblHija) != 0 )
        {
        if ( ErrorPX(PXRecBufEmpty( aRecHandle[ KNTBLSIMB ] )) )
           return FALSE;
        /* ENDIF */
        nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knIdClaseDato-1 ],"T" );
        nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knNomTabla-1 ],
                        cNomTblHija );
        nGuardaAlfaNum( &aRecHandle[ KNTBLSIMB ],&aTblTSimb[ knNomDato-1 ],
                        cNomTblHija );
        nGuardaAlfaNum(&aRecHandle[KNTBLSIMB ],&aTblTSimb[ knIdTipoTabla-1 ], "H" );
        if (ErrorPX(PXRecAppend(aTblHandle[ KNTBLSIMB ], aRecHandle[ KNTBLSIMB ] )))
           return FALSE;
        /*ENDIF*/
        strcpy( cNomTbl,cNomTblHija );
        }
     /*ENDIF*/
     }
  while ( PXRecNext( aTblHandle[ KNTABLSHIJ ]) == PXSUCCESS );
  return TRUE;
  }
/*FIN IAddRegsTabla*/


/*-----------------------------------------------------------------------
   -----> FASE - I CREACION DE LA TABLA DE SIMBOLOS
```

```
                FUNCION PRINCIPAL
 ------------------------------------------------------------------------*/
void CreaTblSimb( char *cNomTblPadre /*ENTRA */ )

  {
  struct stElemGpo      *apElemGpo;
  char                  cArea,
                        cNomGrupo[ mxTamNomCampo+1 ],
                        cCveGrupo[ 3 ];
  int                   lIncrementa,
                        nGrupo,
                        nNoVariable;

  apElemGpo  = apgPrimerElem;
  nGrupo     = 0;
  lIncrementa = TRUE;
  nNoVariable = 0;
  strcpy( cNomGrupo,apElemGpo->Atributos.cNomGrupo );
  cArea      = apElemGpo->Atributos.cArea;
  do
    {
    despliega_memoria();
    switch( apElemGpo->Atributos.cTipoUnion )
      {
      case 'C':
      case 'H':
         lAddRegCampo( &(apElemGpo->Atributos) );
         break;
      case 'V':
      case 'v':
         lAddRegVariable( &(apElemGpo->Atributos),&nNoVariable );
         break;
      }
    /*ENDSWITCH*/
    if ( ( strcmp( apElemGpo->Atributos.cNomGrupo,cNomGrupo ) != 0 ) )
      {
      if ( cArea == 'D' )
        {
        lIncrementa = FALSE;
        strcpy( cCveGrupo,"G1" );
        nGrupo++;
        }
      else
        LlaveGpo( &nGrupo,cCveGrupo,lIncrementa );
      /*ENDIF*/
      lAddRegGrupo( cNomTblPadre, cCveGrupo, cArea, cNomGrupo );
      strcpy( cNomGrupo,apElemGpo->Atributos.cNomGrupo );
      cArea = apElemGpo->Atributos.cArea;
      }
    apElemGpo = apElemGpo->apSiguiente;
```

```
    }
  while ( apElemGpo != apgPrimerElem );
  LlaveGpo( &nGrupo,cCveGrupo,lIncrementa );
  lAddRegGrupo( cNomTblPadre, cCveGrupo, cArea, cNomGrupo );
  lAddRegCampoGpo( cNomTblPadre );
  lAddRegsTabla( cNomTblPadre );
  }
/*FIN CreaTblSimb */
```

CAPÍTULO V

PRUEBAS AL GENERADOR DE REPORTES

INTRODUCCIÓN

La secuencia que se ha seguido en la realización de las pruebas y que se describe en el presente capítulo es la siguiente:

- Pruebas de cada módulo por función en forma individual, asegurando que funcionarán apropiadamente como unidad.
- Después de que se integraron los módulos para formar el "Generador de Reportes", se realizaron pruebas de integración, con la cual se verificó el funcionamiento en conjunto de todas las funciones desarrolladas.
- A continuación, se verificaron los requerimientos de validación que se establecieron durante la fase del análisis, a través de la especificación del sistema.
- Finalmente se realizó la prueba del sistema, la cual verifica que todos los elementos engranen apropiadamente y se alcance el rendimiento y funcionalidad adecuados.

V.1 PRUEBAS AL SISTEMA

Antes de liberar un sistema es necesario realizar una serie de pruebas exhaustivas que nos aseguren el buen funcionamiento del sistema en su operación posterior. El éxito del sistema depende de la honestidad con que se aplique esta importante etapa del desarrollo de sistemas. La prueba de software es un elemento crítico para la garantía de calidad del software y representa un último repaso de las especificaciones, del diseño y de la codificación.

V.1.1 OBJETIVO DE LAS PRUEBAS

El propósito de la aplicación de pruebas al sistema, es evitar las sorpresas cuando dicho sistema pase a ser operado. Las pruebas al sistema evalúan el trabajo del análisis, no el de implantación. Glen Myers[1] establece una serie de reglas que sirven acertadamente como objetivos de prueba:

1. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.
2. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
3. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

V.1.2 FASES DE PRUEBA

Las fases que se siguieron en las pruebas del "Generador de Reportes" fueron las siguientes:

- **Pruebas de Unidad**.- Centran el proceso de verificación en la menor unidad del diseño del software, el módulo. Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo.

[1] Myers, G., *The Art of Software Testing*, Wiley, 1979.

Pruebas de Integración .- Es una técnica sistemática para construir la estructura del programa mientras que al mismo tiempo se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo es tomar los módulos probados en unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

- **Prueba de validación**.- Esta prueba se logra cuando el software funciona de acuerdo a la especificación de requerimientos establecida durante el análisis, es decir, funciona de acuerdo con las expectativas del usuario.

- **Prueba del Sistema**.- Esta constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora (hardware, información, etc.).

V.1.3 TIPOS DE PRUEBAS

Para la evaluación de este sistema, en las diferentes fases antes mencionadas, se usarán los tipos de pruebas que a continuación se mencionan:

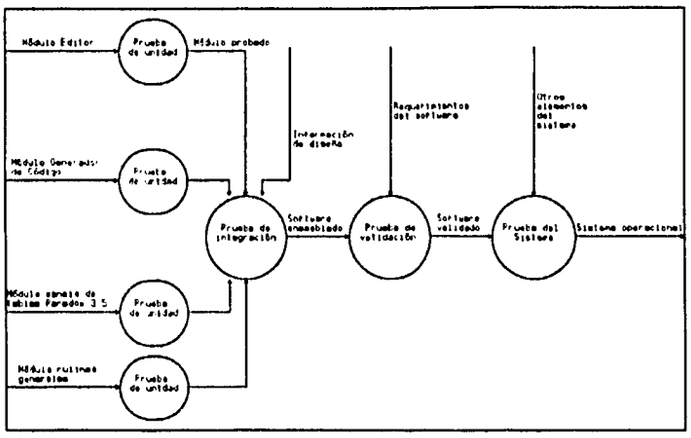
- **Prueba de Trayectoria Normal.-** Esta prueba se deriva, principalmente, del diccionario de datos, y consiste en examinar cada flujo de datos que entre al sistema (lo que ve el usuario). Cada prueba se hace con un número válido de entradas que se conformen a las definiciones del diccionario de datos.

- **Pruebas de Excepción**.- En esta prueba se genera un conjunto de entradas para evaluar la forma del manejo de errores, esto es, se seleccionan valores inválidos para los atributos.

- **Pruebas Especiales**.- Son todas aquellas que no estando dentro de las anteriormente descritas, dependen de la naturaleza particular del sistema, por ejemplo, puntos de reinicio y reprocesos.

V.2 PRUEBAS AL "GENERADOR DE REPORTES"

El siguiente diagrama muestra en forma esquemática los pasos que se siguieron en la prueba del editor.



A continuación se presenta un ejemplo, bastante completo, de pruebas realizadas al "Generador de Reportes" durante la fase de pruebas de validación:

Prueba de Trayectoria Normal

1. Procedimiento para realizar la prueba

Se seleccionó un número válido de datos para examinar el flujo de los mismo, a partir de su entrada al sistema.

Para el módulo editor se seleccionaron tres tablas de Paradox y se copiaron en un directorio de datos, posteriormente se ejecutó el editor del "Generador de Reportes" y se probaron cada una de las opciones del editor (es decir toda la funcionalidad), definiendo un reporte que utilizará campos de las tres tablas. La definición se almacenó en un archivo, posterior a lo cual se salió del generador y se volvió a entrar para cargar la definición y comprobar que fuera correcta.

Para el módulo generador de código, se utilizó la definición del reporte elaborada en el módulo del editor. Una vez que se seleccionó la opción de Código, se comprobó que no hubiera ninguna falla en el proceso de generación de código, y posteriormente se verificó que el código del

reporte generado compilara sin ningún error. Finalmente se ejecutó el nuevo reporte y se comprobó que la salida generada cumpliera con la especificación diseñada mediante el módulo del editor.

2. Datos de prueba al módulo editor

Para nuestro fin, mostraremos un ejemplo de los datos con los que se realizó la prueba del módulo editor.

DATOS DE ENTRADA GENERALES

ATRIBUTO	VALOR DE PRUEBA
Directorio de Datos	C:\GREPORTE\DATOS
Nombre Tabla	INFECNOS
Extensión Reporte	L1
Tabla Hija	PACIENTE
Campo de relación	NUM_REG_PACIENTE
Tabla Hija	SECTORES
Campo de Relación	NUM_SECTOR
Grupo basado en	NUM_SECTOR
Grupo basado en	NUM_REG_PACIENTE

DATOS DE ENTRADA PARA EL ENCABEZADO DEL REPORTE

ATRIBUTO	VALOR DE PRUEBA
Fecha del Reporte	DD/MM/AA
Nó.Página	NNNN
Texto del Encabezado	HOSPITAL DEL VALLE, Listado de Infecciones Nosocomiales, (enmarcado) NUM.REG. PACIENTE, APELLIDOS, PATERNO, MATERNO, NOMBRE(S), EPISODIO, INGRESO, EGRESO

DATOS DE ENTRADA PARA EL ENCABEZADO DEL GRUPO NUM_SECTOR

ATRIBUTO	VALOR DE PRUEBA
Texto del Encabezado	NUMERO DEL SECTOR:
Campos	
	NUM_SECTOR, SECTORES-LUGAR

DATOS DE ENTRADA PARA EL ENCABEZADO DEL GRUPO NUM_REG_PACIENTE

ATRIBUTO	VALOR DE PRUEBA
Campos	NUM_REG_PACIENTE,PACIENTE-APELLIDO_PATERNO, PACIENTE-APELLIDO_MATERNO, PACIENTE-NOMBRES

DATOS DE ENTRADA PARA EL DETALLE DEL REPORTE

ATRIBUTO	VALOR DE PRUEBA
Campos	NUM_VECES_INFEC, FECHA_INGRESO, FECHA_EGRESO

DATOS DE ENTRADA PARA EL TOTAL DEL GRUPO NUM_SECTOR

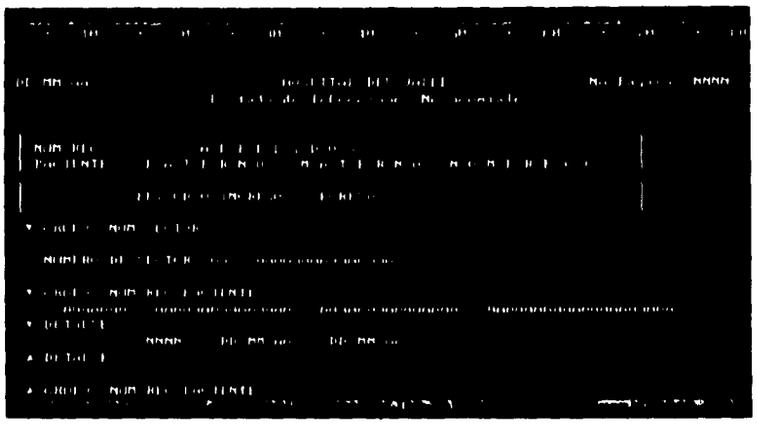
ATRIBUTO	VALOR DE PRUEBA
Texto del total	(una linea para indicar total ---------), TOTAL DE INFECCIONES DEL SECTOR:
Variable	Cuenta del campo NUM_VECES_INFEC

DATOS DE ENTRADA PARA EL TOTAL DEL REPORTE

ATRIBUTO	VALOR DE PRUEBA
Texto del total	(una linea para indicar total ---------), TOTAL DE INFECCIONES EN EL HOSPITAL:, ELABORO: EUNICE ROSALES CONTRERAS
Variable	Cuenta del campo NUM_VECES_INFEC

3. Resultado de la prueba al módulo editor

A continuación se muestran las pantallas del editor, con el formato final del diseño del reporte.



4. Datos de prueba al módulo generador de código

Los datos de entrada a este módulo serán la definición del reporte generada en la prueba del módulo editor, la cual está almacenada en el archivo INFECNOS.L1

5. Resultados de la prueba al módulo generador de código

Como resultado de elegir esta opción se obtuvo:

a) **El código del reporte** creado automáticamente a través del generador de código, con base en la definición realizada mediante el módulo editor, se almacenó en el archivo INFECL1.C, el cual se muestra a continuación:

```
/*-----------------------------------------------------------------
            Generador de Reportes Versión 1.0
   PROGRAMA:  Listado de Infecciones Nosocomiales
   PROGRAMADOR: Eunice Rosales Contreras
   FECHA DE CREACION    : 21/11/1994
   FECHA DE ACTUALIZACION:
 -----------------------------------------------------------------*/

#include <stdio.h>

#include "pxengine.h"
#include "cadenas.h"
#include "sistema.h"
#include "tiempo.h"
#include "pxfunc.h"
#include "impresor.h"

/*DEFINICION DE CONSTANTES*/
#define knMaxLineas 48
#define KNINFECNOS 0
#define KNSECTORES 1
#define KNPACIENTE 2
#define cDirDatos "C:\\GREPORTE\\DATOS"

/*DEFINICION DE VARIABLES*/
   RECORDHANDLE aRecHandle[ knMaxRecBufs ];
   TABLEHANDLE  aTblHandle[ knMaxTableHndls ];

   est_sistema  sist;
   FILE         *ArchSalida;
   double       n1CueNUM_VECES_INFEC,
                n2CueNUM_VECES_INFEC;
   char         FECHA_EGRESO[ 8+1 ],
                FECHA_INGRESO[ 8+1 ],
                NUM_REG_PACIENTE[ 7+1 ],
                NUM_SECTOR[ 2+1 ];
   double       NUM_VECES_INFEC;
   char         PACIENTE_APELLIDO_MATERNO[ 15+1 ],
                PACIENTE_APELLIDO_PATERNO[ 15+1 ],
                PACIENTE_NOMBRES[ 20+1 ],
```

```
/*DEFINICION DE PROCEDIMIENTOS*/
int lAbreArchivos( char *cSalida /*SALE*/ )

   {
   pathstr cTemp;
   char    cExito;

   strcpy( cTemp,cDirDatos );
   strcat( cTemp,"\\INFECNOS" );
   cExito = AbreTabla( cTemp,&aTblHandle[KNINFECNOS],&aRecHandle[KNINFECNOS ] );

   strcpy( cTemp,cDirDatos );
   strcat( cTemp,"\\SECTORES" );
   cExito = AbreTabla( cTemp,&aTblHandle[KNSECTORES],&aRecHandle[KNSECTORES]);

   strcpy( cTemp,cDirDatos );
   strcat( cTemp,"\\PACIENTE" );
   cExito = AbreTabla( cTemp,&aTblHandle[ KNPACIENTE],&aRecHandle[ KNPACIENTE] );

   strcpy( cTemp,cDirDatos );
   strcat( cTemp,"\\INFECL1.TXT" );
   ArchSalida = fopen( cTemp,"wt" );
   strcpy( cSalida,cTemp );
}
/*FIN lAbreArchivos*/

void CierraArchivos( void )

   {
   CierraTabla( &aTblHandle[ KNINFECNOS ],&aRecHandle[ KNINFECNOS ] );
   CierraTabla( &aTblHandle[ KNSECTORES ],&aRecHandle[ KNSECTORES ] );
   CierraTabla( &aTblHandle[ KNPACIENTE ],&aRecHandle[ KNPACIENTE ] );
   fclose( ArchSalida );
}
/*FIN CierraArchivos*/

void lLeeRegINFECNOS( void )
   {
   nLeeFecha( &aRecHandle[ KNINFECNOS ],4,FECHA_EGRESO );
   nLeeFecha( &aRecHandle[ KNINFECNOS ],3,FECHA_INGRESO );
   nLeeAlfaNum( &aRecHandle[ KNINFECNOS ],1,NUM_REG_PACIENTE );
   nLeeAlfaNum( &aRecHandle[ KNINFECNOS ],9,NUM_SECTOR );
   nLeeDouble( &aRecHandle[ KNINFECNOS ],2,&NUM_VECES_INFEC );
   }
/*FIN lLeeRegINFECNOS */

void lLeeRegPACIENTE( void )
   {
   int nOk;

   PXPutAlpha( aRecHandle[ KNPACIENTE ],1, NUM_REG_PACIENTE );
```

```
  if ( nOk = ErrorPX( PXSrchKey( aTblHandle[ KNPACIENTE ], aRecHandle[KNPACIENTE],
  1,SEARCHFIRST ))) return;
  if ( nOk = ErrorPX( PXRecGet( aTblHandle[ KNPACIENTE ], aRecHandle[ KNPACIENTE ]
  ))) return;
  nLeeAlfaNum( &aRecHandle[ KNPACIENTE ],3,PACIENTE_APELLIDO_MATERNO );
  nLeeAlfaNum( &aRecHandle[ KNPACIENTE ],2,PACIENTE_APELLIDO_PATERNO );
  nLeeAlfaNum( &aRecHandle[ KNPACIENTE ],4,PACIENTE_NOMBRES );
  }
/*FIN lLeeRegPACIENTE */

void lLeeRegSECTORES( void )
  {
  int nOk;

  PXPutAlpha( aRecHandle[ KNSECTORES ],1, NUM_SECTOR );
  if (nOk=ErrorPX(PXSrchKey(aTblHandle[ KNSECTORES ], aRecHandle[ KNSECTORES ],
  1,SEARCHFIRST ))) return;
  if (nOk=ErrorPX( PXRecGet( aTblHandle[ KNSECTORES ], aRecHandle[ KNSECTORES ]
  ))) return;
  nLeeAlfaNum( &aRecHandle[ KNSECTORES ],2,SECTORES_LUGAR );
  }
/*FIN lLeeRegSECTORES */

void CalculaVariables()

  {
  n1CueNUM_VECES_INFEC++;
  n2CueNUM_VECES_INFEC++;
  }
/*FIN CalculaVariables*/

void ImpEncabezado( int *nNumLinea, /*ENTSAL*/
                    int *nNumPag    /*ENTSAL*/ )
  {
  est_fecha stFecha;

  if ( *nNumLinea > knMaxLineas || *nNumPag == 1 )
    {
    if ( *nNumPag > 1 )
      fprintf( ArchSalida,"%c",FF );
    /*ENDIF*/
    trae_fecha( &stFecha );
    fprintf( ArchSalida,"%s","                                                    ");
    fprintf( ArchSalida,"\n");
    fprintf( ArchSalida,"%s","                                                    ");
    fprintf( ArchSalida,"\n");
    fprintf( ArchSalida,"%s"," ");
    fprintf( ArchSalida,"%s",form_fecha( "DD/MM/AA",&stFecha ));
    fprintf( ArchSalida,"%s","              HOSPITAL DEL VALLE          No.Página: ");
    fprintf( ArchSalida,"%s",rform("NNNN",*nNumPag));
    fprintf( ArchSalida,"%s"," ");
```

```
      fprintf( ArchSalida,"\n");
      fprintf( ArchSalida,"%s","              Listado de Infecciones Nosocomiales                  ");
      fprintf( ArchSalida,"\n");
      fprintf( ArchSalida,"%s","                                                                   ");
      fprintf( ArchSalida,"\n");
      fprintf( ArchSalida,"%s"," +-------------------------------------------------------------+        ");
      fprintf( ArchSalida,"\n");
       fprintf( ArchSalida,"%s","| NUM.REG.          A P E L L I D O S                          |
");
      fprintf( ArchSalida,"\n");
      fprintf( ArchSalida,"%s","| PACIENTE   P A T E R N O   M A T E R N O   N O M B R E
      (S)   |         ");
      fprintf( ArchSalida,"\n");
      fprintf( ArchSalida,"%s"," +-------------------------------------------------------------+        ");
      fprintf( ArchSalida,"\n");
      fprintf( ArchSalida,"%s","|          EPISODIO INGRESO   EGRESO                |        ");
      fprintf( ArchSalida,"\n");
      fprintf( ArchSalida,"%s"," +-------------------------------------------------------------+        ");
      fprintf( ArchSalida,"\n");
      *nNumLinea += 11;
      (*nNumPag)++;
      }
    /*ENDIF*/
    }
/*FIN ImpEncabezado*/

void IniVarsNUM_SECTOR()

    {
    n1CueNUM_VECES_INFEC = 0;
    }
/*FIN IniVarsNUM_SECTOR*/

void IniVars()

    {
    n2CueNUM_VECES_INFEC = 0;
    }
/*FIN IniVars*/

void ImpEncGNUM_SECTOR( int *nNumLinea, /*ENTSAL*/
                        int *nNumPag    /*ENTSAL*/ )
    {
      fprintf( ArchSalida,"\n");
      fprintf( ArchSalida,"%s","    NUMERO DE SECTOR: ");
      fprintf( ArchSalida,"%2.2s ",NUM_SECTOR);
      fprintf( ArchSalida,"%s","  ");
      fprintf( ArchSalida,"%15.15s ",SECTORES_LUGAR);
      fprintf( ArchSalida,"%s","                              ");
      fprintf( ArchSalida,"\n");
```

```
    fprintf( ArchSalida,"\n");
    *nNumLinea += 3;
  }
/*FIN ImpEncGNUM_SECTOR */

void ImpEncGNUM_REG_PACIENTE( int *nNumLinea, /*ENTSAL*/
          int *nNumPag   /*ENTSAL*/ )
  {
    fprintf( ArchSalida,"%s","     ");
    fprintf( ArchSalida,"%7.7s ",NUM_REG_PACIENTE);
    fprintf( ArchSalida,"%s","  ");
    fprintf( ArchSalida,"%15.15s ",PACIENTE_APELLIDO_PATERNO);
    fprintf( ArchSalida,"%s","  ");
    fprintf( ArchSalida,"%15.15s ",PACIENTE_APELLIDO_MATERNO);
    fprintf( ArchSalida,"%s","  ");
    fprintf( ArchSalida,"%20.20s ",PACIENTE_NOMBRES);
    fprintf( ArchSalida,"%s","          ");
    fprintf( ArchSalida,"\n");
    *nNumLinea += 1;
  }
/*FIN ImpEncGNUM_REG_PACIENTE */

void ImpDetalle( int *nNumLinea, /*ENTSAL*/
          int *nNumPag   /*ENTSAL*/ )
  {
    fprintf( ArchSalida,"%s","             ");
    fprintf( ArchSalida,"%s",rform("NNNN",NUM_VECES_INFEC));
    fprintf( ArchSalida,"%s","    ");
    fprintf( ArchSalida,"%8.8s ",FECHA_INGRESO);
    fprintf( ArchSalida,"%s","   ");
    fprintf( ArchSalida,"%8.8s ",FECHA_EGRESO);
    fprintf( ArchSalida,"%s","                                  ");
    fprintf( ArchSalida,"\n");
    *nNumLinea += 1;
  }
/*FIN ImpDetalle */

void ImpTotGNUM_SECTOR( int *nNumLinea, /*ENTSAL*/
          int *nNumPag   /*ENTSAL*/ )
  {
    fprintf( ArchSalida,"%s","  ------------------------------------------------------------------------  ");
    fprintf( ArchSalida,"\n");
    fprintf( ArchSalida,"%s","         TOTAL DE INFECCIONES DEL SECTOR: ");
    fprintf( ArchSalida,"%s",rform("NNNN",n1CueNUM_VECES_INFEC));
    fprintf( ArchSalida,"%s","                         ");
    fprintf( ArchSalida,"\n");
    fprintf( ArchSalida,"\n");
    *nNumLinea += 3;
  }
/*FIN ImpTotGNUM_SECTOR */
```

```
void ImpTotGNUM_REG_PACIENTE( int *nNumLinea, /*ENTSAL*/
          int *nNumPag   /*ENTSAL*/ )
  {
  fprintf( ArchSalida,"\n");
  *nNumLinea += 1;
  }
/*FIN ImpTotGNUM_REG_PACIENTE */

void ImpTotReporte( int *nNumLinea, /*ENTSAL*/
          int *nNumPag   /*ENTSAL*/ )
  {
     fprintf( ArchSalida,"%s","                                                    ");
     fprintf( ArchSalida,"\n");
     fprintf( ArchSalida,"%s","  ------------------------------------------------------------  ");
     fprintf( ArchSalida,"\n");
     fprintf( ArchSalida,"%s","        TOTAL DE INFECCIONES EN EL HOSPITAL: ");
     fprintf( ArchSalida,"%s",rform("NNNN",n2CueNUM_VECES_INFEC));
     fprintf( ArchSalida,"%s","                  ");
     fprintf( ArchSalida,"\n");
     fprintf( ArchSalida,"%s","                                                    ");
     fprintf( ArchSalida,"\n");
     fprintf( ArchSalida,"%s","                                                    ");
     fprintf( ArchSalida,"\n");
     fprintf( ArchSalida,"%s","   ELABORO: EUNICE ROSALES CONTRERAS                ");
     fprintf( ArchSalida,"\n");
     fprintf( ArchSalida,"%s","                                                    ");
     fprintf( ArchSalida,"\n");
     fprintf( ArchSalida,"%s","                                                    ");
     fprintf( ArchSalida,"\n");
     fprintf( ArchSalida,"%s","                                                    ");
     fprintf( ArchSalida,"\n");
     *nNumLinea += 9;
  }
/*FIN ImpTotReporte */

void ImprimeSalida( char *cSalida /*ENTRA*/ )

  {
  char    cRespuesta;
  pathstr cTemp;

  do
     {
     printf( "¿Desea mandar el reporte a impresora? < S/N > ");
     cRespuesta = toupper( getche() );
     }
  while ( cRespuesta != 'S'&& cRespuesta != 'N' );
  if ( cRespuesta == 'S' )
     {
     printf( "%s\n","Aliste la IMPRESORA y oprima el <ENTER>" );
```

```
    if ( !inic_impresora() )
       {
       printf( "%s\n","La impresora no está lista para recibir datos...");
       return;
       }
     else
       {
       strcpy( cTemp,"type " );
       strcat( cTemp,cSalida );
       strcat( cTemp," >lpt1 ");
       system( cTemp );
       }
     /*ENDIF*/
     }
   /*ENDIF*/
   }
/*FIN ImprimeSalida */

/*FUNCION PRINCIPAL*/
  main()

  {
   int        nOk,
              dummy,
              nPagina,
              nLinea;
   pathstr   cSalida;
   RECORDNUMBER nRecs;
   char       cGpoNUM_SECTOR[ 2+1 ];
   char       cGpoNUM_REG_PACIENTE[ 7+1 ];

   impresora_ok = TRUE;
   nOk          = TRUE;
   dummy        = FALSE;
   nPagina      = 1;
   nLinea       = 0;
   sist.num_impresora = 1;
   InicializaPXFunc();
     if ( !AbreArchivos( cSalida ) )
      {
      printf( "La salida se está enviando al archivo %s\n",cSalida );
      if ( ErrorPX( PXTblNRecs( aTblHandle[ KNINFECNOS ],&nRecs )))
         return FALSE;
      else
         if ( nRecs < 1 )
            return FALSE;
         /*ENDIF*/
      /*ENDIF*/
      if (ErrorPX(PXRecGet(aTblHandle[KNINFECNOS ],aRecHandle[ KNINFECNOS ] ))) return;
      !LeeRegINFECNOS();
      !LeeRegSECTORES();
```

```
    lLeeRegPACIENTE();
    ImpEncabezado( &nLinea,&nPagina );
    IniVars();
    while( nOk && impresora_ok )
      {
      strcpy( cGpoNUM_SECTOR,NUM_SECTOR );
      ImpEncGNUM_SECTOR( &nLinea,&nPagina );
      IniVarsNUM_SECTOR();
      while( strcmp( cGpoNUM_SECTOR,NUM_SECTOR ) == 0 && nOk && impresora_ok )
        {
        strcpy( cGpoNUM_REG_PACIENTE,NUM_REG_PACIENTE );
        ImpEncGNUM_REG_PACIENTE( &nLinea,&nPagina );
        while( strcmp( cGpoNUM_REG_PACIENTE,NUM_REG_PACIENTE ) == 0 &&
              nOk && impresora_ok )
          {
          ImpDetalle( &nLinea,&nPagina );
          CalculaVariables();
          impresora_ok = TRUE;
          if ( ErrorPX( PXRecNext( aTblHandle[ KNINFECNOS ] ))) nOk = FALSE;
          if ( ErrorPX( PXRecGet( aTblHandle[ KNINFECNOS ],
              aRecHandle[ KNINFECNOS ] ))) nOk = FALSE;
          if ( nOk )
            {
            lLeeRegINFECNOS();
            lLeeRegSECTORES();
            lLeeRegPACIENTE();
            }
          /*ENDIF*/
          }
        /*ENDWHILE*/
        ImpTotGNUM_REG_PACIENTE( &nLinea,&nPagina );
        }
      /*ENDWHILE*/
      ImpTotGNUM_SECTOR( &nLinea,&nPagina );
      }
    /*ENDWHILE*/
    ImpTotReporte( &nLinea,&nPagina );
    if (impresora_ok ) fprintf( ArchSalida,"%c",FF);
    CierraArchivos();
    ImprimeSalida( cSalida );
    }
  /*ENDIF*/
  }
/*FIN PROGRAMA*/
```

El código fue compilado y ligado con éxito por el módulo generador de código, el cual hace una llamada al compilador de línea de comandos de Borland C++ 3.1, creando como resultado el archivo ejecutable INFECL1.EXE.

b)El archivo de salida en donde se almacenó el resultado de ejecutar el nuevo reporte INFECL1 fue INFECL1.TXT el cual también se mando a impresión. A continuación se muestra el contenido del archivo de salida (al importarlo a Word 6.0, se perdió un poco el formato), el cual cumple con las especificaciones realizadas por el usuario en el diseño del reporte mediante el módulo editor.

```
21/11/94              HOSPITAL DEL VALLE              No.Página: 1
                  Listado de Infecciones Nosocomiales

NUM.REG.      APELLIDOS
PACIENTE  PATERNO  MATERNO  NOMBRE(S)
      EPISODIO INGRESO  EGRESO

 NUMERO DE SECTOR: 01      U.T.I.

  0000001        DIAZ        PEREZ       JOSE ALFREDO
        2      25/3/93    16/4/93

  0000002      MENDEZ       ROJAS          IRMA
        1      12/10/93   22/10/93

TOTAL DE INFECCIONES DEL SECTOR:  2

 NUMERO DE SECTOR: 02      URGENCIAS

  0000003     ALVARADO       ACUNA        RODRIGO
        1      15/1/93     5/2/93

  0000004    RODRIGUEZ       ZUNIGA       PATRICIA
        1      15/1/93    30/1/93

TOTAL DE INFECCIONES DEL SECTOR:  2

TOTAL DE INFECCIONES EN EL HOSPITAL:  4

 ELABORO: EUNICE ROSALES CONTRERAS
```

CAPÍTULO VI

CONCLUSIONES

El propósito de este último capítulo es el de recopilar los pensamientos y conclusiones que surgieron durante el desarrollo de esta tesis.

La Ingeniería de Software

La Ingeniería de Software tiene como objetivo el brindar métodos, herramientas y procedimientos para el desarrollo de software, que facilitan controlar el proceso de desarrollo del software y suministran las bases para construir software de alta calidad de una forma productiva, por lo cual fue una disciplina fundamental que sirvió de guía durante el desarrollo de esta tesis. El Generador de Reportes desarrollado como tema de tesis, es el resultado de seguir una de las metodologías que forman parte de la Ingeniería de Software: la metodología del Análisis y Diseño Estructurado y de hacer uso de los procedimientos y herramientas que ésta utiliza. El Análisis Estructurado dio la guía para la formulación de un documento claro de objetivos para el Generador de Reportes, el cual fue la base en la fase de diseño, programación y pruebas. El resultado final es un sistema que cumple con lo plasmado en el documento de objetivos, el cual ha sido diseñado y consecuentemente programado en forma sencilla y modular, lo cual facilitará el mantenimiento y la realización de mejoras al sistema.

La interfaz con el usuario final

La interfaz con el usuario final, se consideró uno de los puntos críticos a considerar en las diferentes fases del desarrollo del Generador de Reportes. No sólo la funcionalidad determina el éxito de un sistema, sino que dependiendo de lo amigable que se considere un sistema es que un usuario lo usa o lo deja de usar. Una clara muestra es la tendencia actual a mostrar una interfaz gráfica con el usuario y el auge del sistema operativo Windows en el mundo de las PC´s. Simplificar la interfaz entre el sistema y el usuario final, lograr que una pantalla amigable cree la ilusión de simplicidad representa un reto para el analista-programador, la idea de sencillez implica una mayor complejidad en el desarrollo del sistema. Por otra parte debemos enfocar nuestra atención a el hecho de que estamos creando un sistema para ser usado por humanos y que por lo tanto no debemos subestimarlo en el desarrollo de sistemas.

Análisis

La fase del análisis fue la más importante y la más difícil de evaluar ya que el objetivo de esta fase fue la de crear el documento de objetivos que fuera la base para las siguientes fases y en sí para el Generador de Reportes. Al crear una nueva herramienta como el Generador de Reportes se pretendió proporcionar a los usuarios una herramienta que proporcionara una funcionalidad adicional a la del generador de reportes de una base de datos comercial, en este caso Paradox. Por este motivo fue fundamental realizar un análisis comparativo de diferentes generadores de reportes de bases de datos comerciales y llevar a cabo no solo un análisis teórico, revisando manuales y artículos, sino también utilizándolos como un usuario final, con el fin de percibir más claramente las ventajas de las diferentes interfaces con el usuario, diferencias en la funcionalidad, principios para definición de la base de datos y procedimientos para definición de reportes. Con este análisis comparativo se obtuvo una base para establecer la funcionalidad y las principales características para la interfaz con el usuario del Generador de Reportes. El módulo del editor es el que refleja el resultado de esta primera parte del análisis. La segunda parte del análisis tuvo como objetivo la definición de la generación de código. Para lograr este objetivo se revisó bibliografía relacionada con la sintaxis de los lenguajes de programación y con la teoría de compiladores. Con esta información se fundamentó el documento de objetivos del módulo genereador de código. En general el análisis, fue la fase en la que se requirió de una mayor investigación de productos y bibliografía, con el fin de obtener una herramienta más competitiva y atractiva a los usuarios finales. Fue la fase en la que se invirtió más tiempo ya que como se mencionó en un inicio se considera como la más importante porque determina la mayor o menor probabilidad de éxito de un proyecto.

Diseño

El objetivo del diseño fue el de obtener un sistema simple, con el fin de reducir el tiempo de depuración de errores y de modificaciones. En esta fase se trató de invertir el tiempo necesario para identificar las diferentes funciones del sistema y también para identificar aquellas funciones que fueran constantemente usadas y pudieran integrarse en un módulo de rutinas especificas. Por otra parte el enfoque que se tuvo al realizar la revisión de las estructuras de datos fue la del rendimiento del sistema, las estructuras con menor frecuencia de uso se manejaron como

estáticas y la estructuras de datos en las que se almacena la definición del reporte se definió como una estructura dinámica (una lista doblemente ligada), con lo cual el usuario final obtuviera una mejor respuesta del sistema. En conclusión durante la fase de diseño se tuvo muy presente que de la calidad del diseño depende la facilidad de la programación y en si el rendimiento futuro del sistema.

Programación

Esta fase concretó todo el trabajo desarrollado en las fases anteriores y, por decirlo así, es en la que se pudo evaluar de una manera más objetiva la calidad del análisis y diseño realizado. El mayor esfuerzo en esta fase se concentró en concretar el punto que se consideraba clave para el éxito del sistema: la interfaz con el usuario. Esta implicó la facilidad de uso, la simplicidad de aprendizaje, la mejora en fiabilidad, la reducción de la recurrencia de error y el aumento de satisfacción del usuario, al mismo tiempo que se mantuvo la garantía de eficiencia de la máquina, de la capacidad del software y de las restricciones del hardware.

El Generador de Reportes

El Generador de Reportes, es una herramienta que permite generar reportes de información depositada en tablas con el formato de la base de datos Paradox, sin que sea necesario contar con el administrador de la base de datos, y que por otra parte facilita la interoperabilidad entre dicha base de datos y otros paquetes de software o hardware. Esta interoperabilidad se logra al realizar la generación automática de código en lenguaje C, libre de errores sintácticos, el cual incluye las funciones necesarias de Paradox Engine que permiten extraer información de la base de datos, y el código necesario para obtener el reporte diseñado por el usuario final mediante el módulo del editor. El código generado puede modificarse para combinar información de diferentes bases de datos, paquetes, etc.

La interfaz con el usuario final se lleva a cabo mediante el módulo editor, con el cual el usuario final crea la definición del reporte. En la pantalla del editor el usuario diseña el formato de salida del reporte, auxiliándose por una serie de menús tipo Lotus. En esta pantalla se utiliza un estándar de colores para indicar el significado de cada uno de los elementos incluidos en la definición del reporte (campo de una tabla, texto y texto informativo), en la parte inferior de la pantalla se proporciona información sobre el nombre del campo y tabla a la que pertenece, renglón y columna etc. Cabe mencionar que el editor de

reportes es muy similar a los generadores de reportes que se incluyen junto con los administradores de bases de datos comerciales, por lo que su uso es similar y sencillo de aprender.

Por las características antes mencionadas el Generador de Reportes está orientado a los siguientes tipos de usuarios:

- Usuarios finales de diferentes áreas, que no cuentan con el administrador de Paradox y que requieren elaborar en forma sencilla reportes para explotar información depositada en tablas con el formato de Paradox. Este tipo de usuarios no requiere tener conocimientos de programación para utilizar el Generador de Reportes.

- Desarrolladores de sistemas con conocimiento de programación en lenguaje C, para los cuales el Generador de Reportes proporciona el código fuente del reporte diseñado, el cual puede ser modificado.

Líneas futuras

Día con día aparecen en el mercado de las computadoras personales nuevas tecnologías, herramientas, estándares, etc. Por lo que los productos de software se renuevan constantemente para integrar los nuevos beneficios que estas traen consigo y así brindar al usuario final un producto que aproveche al máximo su inversión. En el caso del Generador de Reportes se han identificado los siguientes puntos a integrar en versiones futuras del sistema.

- *Interfaz con el usuario*.- Como se mencionó anteriormente uno de los puntos que se consideran de mayor importancia es la interfaz con el usuario final. En este aspecto, el sistema operativo Windows y las aplicaciones para este ambiente se están convirtiendo en un nuevo estándar en el mercado, por lo que uno de los principales cambios a realizar en una siguiente versión sería el modificar el Generador de reportes para correr en este ambiente.

- *Campos calculados*.- Integrar una mayor flexibilidad en la definición de campos calculados, al poder definir expresiones en las cuales se puedan incluir funciones (subcadenas, desviación estándar, etc.) , y operaciones (+,-,*,/) que operen sobre uno o más campos.

- *Bases de datos accesadas.*- Accesar información almacenada en los manejadores de base de datos dBase y FoxPro. El usuario final elegiría el manejador de base de datos a accesar y haciendo uso de la misma interfaz y funciones que presenta el módulo editor, elaboraría el reporte deseado.

Es importante señalar que en el diseño del Generador de Reportes se aplicaron dos conceptos fundamentales para conseguir la independencia entre módulos: el acoplamiento y la cohesividad; con el objetivo de que en las versiones futuras al sistema, se reduzcan los efectos colaterales al agregar nuevas funciones en módulos existentes y nuevos módulos, como los que permitan accesar información de dBase y FoxPro.

Desarrollo profesional

Durante el desarrollo de este trabajo de tesis pude comprobar que en nuestra formación como Ingenieros en Computación se nos dan las bases para desarrollar todo tipo de soluciones. Los conocimientos que adquirimos nos dan un punto de partida y un método para un análisis e investigación más profundo.

Nuestra responsabilidad es seguir actualizándonos con nuevas tecnologías, para crear mejores soluciones y colaborar de esta manera al progreso de nuestro país.

Conclusión General

Finalmente se concluye que en el desarrollo de la tesis titulada "Generador de Reportes" se han cumplido con todos los objetivos establecidos; estos se detallaron de manera formal en el documento de objetivos. El documento de diseño muestra la arquitectura del sistema que cumple con los objetivos y el sistema "Generador de Reportes" es el resultado final y objetivo del trabajo realizado durante el análisis, diseño, programación y pruebas.

Por otra parte, esta tesis representa el logro de un reto personal, que involucró creatividad, uso del conocimiento adquirido en la universidad, estudio, planeación, constancia y dedicación. Esta actitud fue la que permitió lograr el objetivo perseguido en la tesis. Como experiencia aplicada en el ejercicio profesional, permite reflexionar en nuevas metas, y en todas las oportunidades existentes al aplicar y profundizar en el conocimiento adquirido durante la formación académica.

BIBLIOGRAFÍA

Borland International Inc.
Borland C++ v. 3.1
Library Reference
1991,1992

Borland International Inc.
Borland C++ v. 3.1
Programmer's Guide
1991,1992

Borland International Inc.
Borland C++ v. 3.1
Tools and Utilities Guide
1991,1992

Borland International Inc.
Paradox Engine 2.0
C Reference Guide
1990,1991

Borland International Inc.
Paradox Engine 2.0
User's Guide
1990,1991

Borland International Inc.
Paradox 3.5
User's Guide
1985,1990

Borland International Inc.
Paradox 3.5
Introduction
1985,1990

Date, C.J.
Introducción a los Sistemas de Bases de Datos
1986 (trad. Tercera Edición en Inglés 1986)
Sistemas Técnicos de Edición, .S.A. de C.V.

Grogorno, Peter
Programación en PASCAL,
Edición revisada 1984.
Fondo Educativo Interamericano

IBM de México
Análisis y Diseño de Sistemas
IBM Latin American Education Center (IBM-LANEC)
1989

Kernighan, Brian W.
Ritchie, Dennis M.
El lenguaje de programación C
Primera edición 1985 (trad. Primera Edición en Inglés 1978)
Prentice Hall

Pressman, Roger S.
Ingeniería del Software - Un enfoque práctico
Segunda Edición (trad. Segunda Edición en Inglés), 1990
McGraw-Hill
España

Tremblay, Jean-Paul
Sorenson, Paul G.
The Theory and Practice of Compiler Writing
McGraw-Hill
1985

Yourdon, Edward
Análisis Estructurado Moderno
Primera edición 1993 (trad. Primera Edición en Inglés 1989)
PRENTICE-HALL HISPANOAMERICANA, S.A.