UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

GENERADOR DE PROGRAMAS PARA FILTRADO Y VALIDACION DE ARCHIVOS

TESIS PROFESIONAL

ALEJANDRO LOPEZ CASTRO

MEXICO, D. F. 1984

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

I N D I C E

CAPITULO I

I N T R O D U C C I O N

Un problema inherente en todos los sistemas que se ven apoyados por procesos de cómputo es la correcta introducción de los datos a la computadora, misma que se efectua con intervención humana y que por tanto no puede ser perfecta. Se hace necesario entonces contar con un procedimiento capaz de detectar y reportar los errores cometidos en la captura de datos. Este procedimiento, que es parte obligada en casi todos los sistemas de cómputo, es comúnmente conocido como "Filtrado y Validación de Datos".

Para poder definir este procedimiento definamos un dato como el valor que toma una variable, y asociemos a cada variable un conjunto de valores permisibles para ser procesados por el sistema. Diremos entonces que un dato es correcto si es un valor permisible para la variable. Por otro lado un grupo de variables pueden establecer una relación definiendo una serie de condiciones que deben cumplir. Diremos que esta relación es correcta si los valores que toman estas variables satisfacen las condiciones impuestas. Una vez hechas las consideraciones anteriores podemos definir el "Filtrado de Datos" como el proceso que determina si un dato es correcto o no, y la "Validación de Datos" como el proceso que determina si las relaciones entre los mismos son correctas o no.

Dado que este proceso es de gran importancia para el correcto funcionamiento de un sistema de cómputo se hace conveniente automatizarlo para obtener las siguientes ventajas:

Suprimir la duplicidad de trabajo de programación proporcionando un método general para filtrar y validar los datos que puede ser utilizado por cualquier sistema

Detectar y reportar los errores cometidos en la captura de los datos lo más pronto posible

Proporcionar al usuario un procedimiento uniforme para la definición de las reglas que deben satisfacer las variables a filtrar y las relaciones a validar.

Este trabajo está orientado a proporcionar estas ventajas; consiste en la generación de un programa COBOL que efectue el filtrado y validación de datos. El programa COBOL es generado por un programa escrito en FORTRAN que recibe sus parámetros en tablas guardadas en archivos. Fue diseñado para manejarse principalmente en sistemas administrativos; su empleo no requiere de conocimiento alguno sobre algún lenguaje de programación y para su funcionamiento debe de proporcionarse solamente la forma en que están organizados los datos.

Es la intención de este trabajo cumplir con las ventajas ya señaladas con objeto de proporcionar tanto al usuario como al personal técnico facilidades para el mejor desarrollo del sistema.

CAPITULO II

SISTEMAS DE INFORMACION

El filtrado y la validación de datos son considerados como partes integrantes de un sistema de información cuyos objetivos son más amplios. Podemos definir un sistema de información como un sistema que provee información histórica, información sobre un estado presente o información proyectada hacia el futuro y que es capaz de resumirla con objeto de proporcionar bases para la toma de decisiones.

Un sistema de estas características para poder funcionar eficazmente debe de presentar los siguientes elementos dentro de su estructura:

- Una función objetivo (que sea cuantificable)
- Un banco de datos o alguna otra facilidad de almacenamiento
- Entradas al sistema
- Salidas del sistema
- Suficiente capacidad de proceso
- Un mecanismo de retroalimentación

Explicaré brevemente lo que representa cada uno de estos elementos, posteriormente enmarcaremos la función de filtrado y validación en este esquema.

Por función objetivo entenderemos lo que es la meta que persigue el sistema cuantificada sobre términos reales.

La meta que persigue un sistema puede ser por ejemplo, minimizar el número de asientos vacios para un sistema de reservaciones de una línea aerea, o bien, proporcionar el mejor servicio posible a los clientes en un sistema bancario de cuentas de ahorro. La interpretación de esta meta por una persona o un grupo de personas provee la función objetivo que puede ser maximizada o minimizada y contra la cual se puede medir el rendimiento del sistema.

Una vez definida la función objetivo, deben identificarse a los usuarios con el objeto de determinar la disponibilidad de la información que formará la entrada al sistema, así como la información que será devuelta a los usuarios y que constituye la salida del sistema.

Se requiere que el sistema tenga capacidad de procesamiento, para transformar la información de entrada en información de salida, de acuerdo a las necesidades del usuario. Para lograr este fin, el sistema debe poder almacenar la información que será modificada.

El estudio del comportamiento de un sistema real, siempre se hace sobre un modelo que lo representa, este modelo es una abstracción del sistema ya que no puede incluir todas las variables o factores que afectan al sistema real, sin embargo en la medida que resulte una buena representación, permite predecir el comportamiento futuro del sistema, así como probar diferentes alternativas para lograr una mejor función objetivo.

Si consideramos a nuestro sistema como un modelo de un sistema real, éste deberá tener información sobre el comportamiento pasado del sistema real que permita predecir acciones futuras. Esa información sobre el pasado del sistema constituye la base de datos histórica del modelo que será modificada posteriormente con la información que ha sido comunicada al usuario.

Podemos señalar que una tarea fundamental con el modelo en operación es la captura de la información que sirve como entrada al modelo, ya que si ésta no es correcta puede llevar a resultados erróneos del modelo, sin que éste sea una representación falsa del sistema, de aquí que la tarea de validación y filtrado de datos resulte esencial en todo sistema.

Finalmente, la retroalimentación del sistema es un elemento de tipo correctivo, cuya función consiste en permitir tomar una acción tendiente a prevenir una ruptura en el funcionamiento del sistema.

Los medios utilizados para estudiar el funcionamiento de un sistema, dependen de características del mismo, por ejemplo, si se está analizando el comportamiento de un nuevo modelo de avión, se puede recurrir a un laboratorio que simule el medio ambiente que encontrará el avión en operación y probar en el laboratorio todas las mejoras de diseño que lleven a obtener el modelo óptimo.

Sin embargo, existen muchos otros sistemas para los cuáles la computadora digital resulta un medio adecuado para su estudio, ya que un sistema de cómputo permite:

1) Gran capacidad de procesamiento rápido y correcto de la información
2) Manejo de grandes volúmenes de información.

Una computadora digital, debe concebirse como constituida por dos grandes partes, lo que representa el equipo en si, ya que cada vez resulta con más capacidad de procesamiento, más veloz debido al desarrollo tecnológico, y los programas de apoyo que provee todo sistema de cómputo.

Sobre este último punto, es importante mencionar que los equipos de cómputo pueden disponer de productos desarrollados para realizar operaciones de carácter general sobre los datos que se pueden almacenar, así por ejemplo, encontraremos programas conocidos como EDITOR que nos permiten reunir datos en registros que a su vez agrupados forman un archivo; programas que nos permiten ordenar los registros de un archivo de acuerdo a una secuencia lógica dada a través de un campo (dato) de un registro que se llaman SORT; programas que pueden obtener un archivo ordenado uniendo varios archivos ordenados que se nombran MERGE, o bien, programas que tienen la capacidad de efectuar operaciones generales sobre archivos, como son: obtener una copia, cambiar el nombre, borrarlos, juntar varios en uno, etc. etc.

Junto a estos productos de caracter general, existen también paquetes orientados a la introducción de datos en la computadora cuyas funciones varían en un amplio rango y van desde una captura simple que no efectua ningún filtrado hasta programas complejos que permiten al usuario definir los formatos que deben seguir sus datos en el filtrado y la validación. Ejemplos de distintos productos que cubren el objetivo señalado se citan a continuación.

Un EDITOR es el medio más simple para capturar datos. Como se mencionó anteriormente, ningún filtrado se efectura sobre los datos y es responsabilidad del usuario que estos se proporcionen correctamente, debido a esto, rara vez se emplea para captura de datos, y cuando este es el caso, se utiliza más bien para archivos pequeños. Su empleo es mucho más frecuente en la captura de programas. Existen programas que fueron desarrollados para facilitar la captura a través de una terminal en forma simple y permitiendo al usuario definir caracteristicas de sus datos para poderlos filtrar.

Un ejemplo es un producto conocido como DFU (DEFINE FILE UTILITY) que opera en los equipos IBM/34 en el que el usuario define los datos que quiere capturar dando a cada tipo de dato un nombre y una clasificación sobre si es numérico o no. Cuando el usuario desea capturar sus datos, DFU le presenta una pantalla con los nombres de los datos que fueron definidos y efectua el filtrado sobre las variables numéricas reportando el error y no registrando el dato hasta que se corrige.

Una herramienta más poderosa es el WSU (WORK STATION UTILITY) producto que opera también en el equipo IBM/34. El utilitario WSU contiene un lenguaje orientado a la captura, filtrado y validación de datos que permite diseñar formatos para presentarse en la pantalla el momento de la captura, definir características de los datos a capturarse introduciendo rangos de valores permitidos para los distintos tipos de dato, verificar contra otro archivo la validez de un grupo de datos (por ejemplo, en un sistema de nómina, una afectación a un empleado solo es válida si el empleado está dado de alta en algún archivo) y construir un algoritmo utilizando las proposiciones más comúnes que se encuentran en un lenguaje de programación con objeto de elaborar reglas propias para el filtrado y la validación. Desafortunadamente, su empleo requiere de conocimientos de programación en un lenguaje que el usuario se ve obligado a aprender. Finalmente citaré un paquete que opera sobre la serie 3000 de HP conocido como V3000 que proporciona una gran flexibilidad y fuerza para realizar las funciones de captura, filtrado y validación de datos. V3000 puede operarse en dos formas, la primera está orientada al usuario no programador, en ella se definen los formatos de las pantallas para la captura, las características de los datos que se van a capturar, proporcionando inclusive, la facilidad de definir un formato (i. e. A9 implica un campo de dos posiciones, donde la primera es un caracter alfabético y la segunda un dígito), y las validaciones que se deben de realizar sobre los datos; una vez hecha esta definición, el proceso de captura, filtrado y validación es

realizado en forma interactiva al momento de digitar en la pantalla. La otra forma de empleo, proporciona una serie de subrutinas que pueden ser llamadas desde un programa escrito en otro lenguaje, y como en el caso de WSU, se utiliza para desarrollar una validación más extensa y completa. La versatilidad que ofrece este paquete ha permitido que goce de gran demanda entre los usuarios de los equipos HP3000.

Como podemos ver, el problema de verificar los datos antes de registrarse en una computadora ha proporcionado un fuerte desarrollo de programas orientados a cumplir este objetivo, y esto se debe principalmente a dos razones:

Cualquier sistema de información debe de garantizar que sus entradas sean correctos para permitir el normal funcionamiento del mismo en sus etapas posteriores

El problema de verificar la validez de los datos puede programarse en muy buena parte en forma independiente de la aplicación que se este tratando, evitando así los continuos programas de filtrado y validación realizados exprofeso.

Esta tesis fue motivada también por estas razones, pero su orientación es distinta a la de los productos antes presentados. La principal diferencia es que estos fueron diseñados para operar en forma interactiva con un usuario, mientras que el generador propuesto filtra y valida los datos en lote. En el primer caso se presupone que se puede contar con una terminal a

través de la cual se van introduciendo los datos y por lo tanto, lo más deseable es que los errores se reporten inmediatamente. En el segundo caso, el empleo del generador se realiza sobre archivos que pueden estar ya capturados sin verificar, y este caso es frecuente de aplicar en archivos que pasan de un equipo de captura a un computador, como puede ser, por ejemplo, el caso de cintas capturadas en equipo especializado para grabación en este tipo de dispositivo.

A continuación, se dará una explicación general del paquete que posteriormente será más detallada en los siguientes capítulos.

En términos generales, el paquete recibirá como entrada las características de las variables a filtrar y las relaciones a validar y proporcionará como salida un programa COBOL para filtrar las variables y validar las relaciones. A este programa le llamaremos indistintamente programa o archivo resultante. El paquete constará de tres programas, uno de los cuales es el SORT que se emplea en el equipo PDP serie 11. Para facilitar la descripción de los programas se llamara "constructor" al primero, "ordenador" al segundo y "editor" al tercero. El programa "constructor" iniciará el proceso, recibirá las entradas recien descritas y construirá un archivo con instrucciones para efectuar las funciones de filtrado y validación, el "constructor" será un programa FORTRAN y proporcionará instrucciones de dos tipos, el primero contendrá instrucciones propias del lenguaje COBOL y el segundo instrucciones muy semejantes a las empleadas por COBOL. Para

formar el programa resultante se requiere entonces tomar las siguientes acciones; corregir las instrucciones del segundo tipo para que sean instrucciones COBOL y ordenar las instrucciones para que formen un programa. Estas funciones serán realizadas por los programas 'editor' y 'clasificador' respectivamente. Como ya fue mencionado, 'clasificador' será en este caso, el utilitario 'SORT' del sistema. 'editor' será un programa escrito en un editor programable llamado TECO que es muy poderoso para manejar textos. El primero en procesarse deberá ser el 'clasificador' que recibirá las instrucciones generadas por el 'constructor' y las dejará clasificadas en el archivo resultante. Finalmente, el 'editor' tomará este archivo y lo modificará a su estado definitivo, listo para compilarse y producir el programa objeto.

Ahora bien, todo este procedimiento será desarrollado para procesarse en un equipo PDP serie 11, que tan transportable puede ser a otro tipo de máquina? Transportar un producto de un equipo a otro sin realizar ningún trabajo adicional es una tarea que rara vez se llega a conseguir, son muchos los factores que obligan a tomar medidas correctivas para poder implantar un producto de una computadora en otra. Sin embargo, existen técnicas utilizadas para facilitar los traslados, basandose en el desarrollo de SOFTWARE trasportable. Las principales técnicas empleadas para desarrollar SOFTWARE transportable son:

Utilizar herramienta ampliamente disponibles como los lenguajes de programación de alto nivel

Usar un método lo suficientemente flexible como para poder ser soportado por una gran variedad de herramientas

Hacer uso de una herramienta especializada capaz de-generar código para una gran variedad de máquinas

Usar generación telescópica, esto es, construir una base transportable o simple de codificar sobre la - cual se pueda generar el resto del trabajo.

Considerando las técnicas anteriores y la forma en que será programado el paquete, se puede decir que solo el programa "constructor" cumplirá con tener un buen grado de transportabilidad ya que FORTRAN es una herramienta ampliamente disponible y suficientemente estándar. No obstante, debido a que este programa realizará la mayor parte del trabajo se puede considerar que la parte medular del sistema es bastante transportable. "Clasificador" será enteramente dependiente del SORT del sistema que se este utilizando, aunque regularmente el empleo de un SORT es muy simple, por lo que esta parte no debe significar mayor problema en un traslado. "Editor" será sin duda la parte más dificil de transportar, ya que TECO es un editor programable que corre en pocas máquinas.

Se empleará en esta parte del paquete para aprovechar las grandes ventajas que ofrece en el manejo de textos y considerando

que el problema que resuelve puede ser también programado en algún lenguaje estándar. Para este efecto, y en general, para facilidad de traslado entre dos equipos, los programas que constituirán el paquete serán documentados.

Pero trasladar el paquete no es la única alternativa con la que se puede resolver un problema de filtrado y validación, se puede también optar por trasladar el programa resultante al equipo deseado, aprovechando que se obtendrá escrito en COBOL utilizando solamente proposiciones comunes a cualquier compilador, por lo que resulta ser bastante transportable. Esta es de hecho la orientación bajo la cual se desarrollará el paquete en el aspecto de transportabilidad.

Resumiendo, podemos mencionar que; el problema de filtrado y validación de datos es común a cualquier sistema de información, ya que es indispensable para verificar que las entradas sean correctas; que los objetivos que cumple son los mismos en cualquier sistema por lo cual se puede generalizar su empleo a través de paquetes que se avoquen a este fin y que; la mayoría de los paquetes desarrollados a este respecto son de tipo interactivo lo cual implica que no pueden filtrar y validar archivos capturados fuera de una pantalla. Debido a lo anterior, la orientación de este trabajo es para efectuar el filtrado y la validación sobre archivos capturados en forma no interactiva.

En los siguientes capítulos se verá con más detalle la forma en que los programas realizarán sus funciones y los elementos que necesitarán para su correcto accionar.

CAPITULO III

DISEÑO DEL PAQUETE

El objetivo de este paquete es filtrar los datos y validar las relaciones existentes en un mismo registro de un archivo dado.

Entre las alternativas posibles, dos fueron las que me parecieron más viables, realizar un intérprete que pudiera cumplir con este objetivo en forma independiente del archivo a procesar o bien elaborar un generador de programas COBOL, en donde cada programa generado dependería de las características particulares del archivo a filtrar y validar. Ambas alternativas fueron enmarcadas por la característica especial de que debían programarse en FORTRAN, dado que era el único lenguaje de propósito general con que contaba el equipo donde realicé la programación. De esta particularidad se desprendio el hecho de que el trabajo posee ciertas características de transportabilidad.

La alternativa de realizar un Intérprete tenía la ventaja de la independencia sobre los archivos, sin embargo presentaba la limitante de tener que trabajar con tablas de dimensión fija si es que la programación no se deseaba complicar en demasia, lo cual acarrearía como consecuencia limitaciones en los archivos a procesar.

Por otro lado, un programa de caracter tan general debe de tomar una serie de previsiones que posiblemente no serán utilizadas con varios archivos, de aquí que en estos casos se tendra probablemente una pérdida de eficiencia a costa de generalidad.

La segunda posibilidad consistia en realizar un programa cuyo objetivo fuera obtener el programa que hiciera las funciones de filtrado y validación sobre un archivo dado, esto es, un generador de programas de filtrado y validación.

La alternativa de elaborar un Generador de Programas COBOL ofrecia ciertas ventajas que me decidieron a escogerla. Entre estas se encontraban las de obtener el programa en el lenguaje que mejor conozco, aprovechar el filtrado natural de COBOL, realizar una validación simple con ciertas reglas que el usuario debe seguir y generar solamente las instrucciones necesarias de acuerdo a las caracteristicas del archivo.

Ahora bien, una vez escogida la alternativa del Generador, se presentaban a su vez varias alternativas para poder recabar la información concerniente al archivo de datos a filtrar y validar. Estas fueron, crear un lenguaje de propósito especial que cumpliera este objetivo, establecer un diálogo con el usuario a través de una terminal, o bien, proporcionar esta información en forma tabular.

Preferi esta última opción debido a que la información que el usuario debe proporcionar es repetitiva, siendo las tablas un buen elemento para proporcionar este tipo de información.

Así el usuario proporciona cuatro tablas divididas en tres archivos con objeto de aprovechar la estructura de un programa COBOL.(*) Las otras alternativas no aprovechan tan bien esta característica ya que, en el caso del diálogo se hubiera caido en un proceso tedioso al tener que responder siempre el mismo cuestionario, y en el caso del lenguaje hubiera sido poco práctico aumentar la complejidad de la programación para que, en última instancia, la información a proporcionar siguiera siempre el mismo esquema.

Una vez seleccionadas las alternativas anteriores, se hizo necesario incluir dos procesos posteriores a la generación del programa.

El primero consistio en emplear el utilitario SORT con objeto de ordenar el programa que se obtiene en su primera fase.

El segundo consistio en elaborar un programa que modificara ciertas instrucciones que se obtienen bajo una edición tipo FORTRAN, necesitando cambiar al formato COBOL. Este programa fue desarrollado en TECO, un editor programable, con objeto de aprovechar los recursos del equipo en que trabaje y simplificar la programación.

El programa puede sustituirse por un programa en FORTRAN o COBOL si se desea que el paquete conserve caracteristicas de transportabilidad.

(*) Una de estas tablas constituye un archivo del paquete, y contiene instrucciones COBOL que se utilizán por el programa generador de acuerdo a las caracteristicas del archivo a filtrar y validar. A esta tabla le llamaremos de aqui en adelante archivo o programa esquema.

Con este diseño, se espera proporcionar al usuario un método simple y eficiente para efectuar el correcto filtrado y validación de sus archivos que tan necesarios son en el funcionamiento de un sistema.

CAPITULO IV

DESCRIECION DEL PAQUEIE

El paquete generador de programas realiza dos funciones principales sobre un archivo de datos, el filtrado y la validación. Por filtrado de información entenderemos aquella parte que revise que cada variable cumpla las características deseadas. Por validación de la información entenderemos aquella parte que revise que las relaciones deseadas entre variables se satisfagan.

El generador de programas de filtrado y validación esta dividido en tres partes, todas estas partes fueron programadas y compiladas en un equipo PDP/11 bajo el sistema operativo RSX-11M. La primera obtiene las instrucciones propias del programa, la segunda clasifica dichas instrucciones de tal forma que el programa queda ordenado; en esta parte se aprovecha el utilitario SORT común a casi todos los equipos de cómputo, la tercera parte efectua un reformateo de las instrucciones dadas. Las tres partes del generador se detallan a continuación.

I) MODULO CONSIRUCTOR.

El programa generador produce como resultado un conjunto de proposiciones en COBOL identificadas por un número de secuencia que ocupa las posiciones 1 a 6 de cada registro generado y que se

van obteniendo en un orden diferente al de COBOL(*). A continuación explico en forma general la manera en que el programa generador produce el programa resultante.

Toda instrucción del programa resultante proviene de una instrucción que se encuentra en un programa esquema o bien es generada por el programa generador. Podemos decir entonces que el programa resultante es la unión de una parte fija con una parte variable.

La parte fija del programa forma un archivo en el que se encuentran instrucciones que son de caracter general a los programas COBOL de filtrado y validación . Dichas instrucciones a su vez quedan divididas en dos grupos; en el primero se encuentran aquellas que formaran parte del programa resultante incondicionalmente, tales como las proposiciones fijas en la IDENTIFICATION DIVISION, ENVIRONMENT DIVISION y DATA DIVISION, o bien aquellas proposiciones de la PROCEDURE DIVISION que permanecen inalterables en el diagrama de flujo del programa. El segundo grupo esta integrado por aquellas proposiciones que pueden o no ser seleccionadas de acuerdo a las caracteristicas del archivo a filtrar y validar, y en algunos casos la inclusión de ciertas instrucciones trae como resultado la exclusión de otras.

(*) En un programa COBOL, las proposiciones deben codificarse en un orden preestablecido por el lenguaje.

A su vez las instrucciones de los grupos pueden ser completas o incompletas, siendo estas últimas instrucciones aquellas que quedan formadas con la inclusión de un argumento.

La parte variable del programa resultante está formada por instrucciones generadas por el programa generador de acuerdo a las caracteristicas de las variables a filtrar y validar.

Tanto las instrucciones obtenidas de la parte fija como las generadas en la parte variable llevan la identificación inicial citada que les permitirá en el proceso de clasificación quedar en forma ascendente formando un programa COBOL bien definido. Dicha identificación forma la secuencia del programa COBOL, los registros de la parte fija traen su propia secuencia y la forma de asignarsela a una instrucción generada aprovecha la estructura del programa COBOL que clasifica a una instrucción en una de sus cuatro divisiones, y a su vez dentro de cada división en una sección particular. Esta clasificación permite obtener números de secuencia fijos a partir de los cuales las instrucciónes generadas obtendrán su secuencia particular por medio de un incremento constante conforme se vayan generando, tomando como base la secuencia fija que les corresponde.

Hemos dicho que la parte variable y la parte fija combinadas en forma adecuada proporcionan el programa resultante. Ahora bien, para que la parte variable y la parte fija en sus instrucciones que requieran de argumentos para completarse puedan formarse, se necesita información relacionada con los datos a filtrar y validar. Dicha información queda organizada en

archivos de la siguiente manera:

i) Archivo que contiene los datos generales del archivo a filtrar y validar

ii) Archivo que contiene la información de las variables a filtrar

iii) Archivo que contiene las relaciones a validar.

Una vez que el usuario ha recabado la información que requieren estos archivos (y que se detalla en el Manual del Usuario) se puede proceder a la generación de su programa filtrador y validador.

Veamos con un poco más de detalle como esta información es utilizada por el programa generador para obtener el programa resultante. Si el programa esta trabajando con una instrucción de la parte fija puede tomar una de las siguientes acciones:

i) Copiarla tal como se encuentra

ii) Copiarla habiendole insertado antes un valor en una posición especifica

iii) No copiarla.

La decisión de copiar o no una instrucción depende de las características del archivo a filtrar y validar y que el usuario ha proporcionado en los archivos mencionados anteriormente. Por ejemplo, si un archivo no tiene declarada una llave (*) para poder detectar registros duplicados, las instrucciones del programa esquema que se ocupan de la duplicidad serán ignoradas y en caso contrario serán copiadas. Observemos que en algunos casos podemos determinar si un grupo de instrucciones debe ser omitido o copiado. Para poder identificar aquellas instrucciones en las que debemos hacer esta decisión, o bien, aquellas que necesitan completarse con algún valor, las instrucciones del programa esquema reservan una posición que identifica estos casos si su valor es diferente a un espacio. Si una instrucción en particular requiere completarse con un valor, además de venir marcada traerá codificada la posición a partir de la cual empieza a copiar el valor. Estos datos, que solo dan información al programa generador no forman parte de la instrucción en el programa resultante en caso de que la instrucción sea copiada.

Si el programa está generando instrucciones, la forma de hacerlo dependerá de la información que este tratando.

(*) Una llave es un dato del registro que contiene el valor suficiente para poder accesarlo.

Básicamente podemos hacer la siguiente distinción de esta información:

i) La que se refiere a las variables a filtrar

ii) La que se refiere a las relaciones a validar.

En el primer caso el programa generará instrucciones para la parte que describe los datos (DATA DIVISION) y para la parte que filtra los datos (PROCEDURE DIVISION). En lo referente a la parte de la descripción de los datos se genera la entrada obligatoria que describe el formato del dato y opcionalmente la clausula que declara los valores condicionales en caso que estos existan.

En la PROCEDURE DIVISION se genera la instrucción IF particular para cada dato de acuerdo a las características del mismo en la que se incluye el llamado a un parrafo común que se invoca si la prueba determina que el dato es erroneo.

En el segundo caso el programa genera instrucciones solamente para la parte que valida las relaciones (PROCEDURE DIVISION) y su funcionamiento consiste en generar una instrucción IF que revisa que la negación de la relación se cumpla (el usuario proporciona la relación que se debe cumplir), y el llamado a un parrafo común que se invoca si la relación que se esta validando es incorrecta.

En ambos casos las instrucciones se construyen a base de movimientos de constantes alfanuméricas o de valores de arreglos

alfanuméricos a posiciones determinadas de un registro de 80 posiciones que representa la instrucción COBOL.

Dado que el archivo esquema es completamente procesado en esta parte del sistema es conveniente revisar con mayor detalle el diseño de sus registros.

Las primeras seis posiciones se reservan para asociar a cada instrucción un número de secuencias que posteriormente será utilizado como llave en el proceso de clasificación del programa generado; la séptima posición es ocupada por un identificador que puede tomar uno de tres valores: espacio, '*' ó '@'. La función de este indicador estriba en servir como delimitador cuando su valor es diferente al espacio, para saltar o copiar instrucciones, o bien, para identificar una instrucción incompleta en la que deba introducirse un argumento. La acción particular que se tome en un momento determinado es controlada por el programa generador y depende de las características del archivo a filtrar y validar. El programa puede saltar instrucciones hasta encontrar una '@', o bien copiar instrucciones hasta encontrar un caracter diferente al espacio, '*' ó '@', o bien copiar instrucciones hasta encontrar un '*' y ahi completar la instrucción con un argumento.

Las posiciones 8 a 72 contienen la instrucción COBOL en forma completa o incompleta que puede pasar a formar parte del programa generado. En caso de que la instrucción sea incompleta las posiciones 77 y 78 serán ocupadas por un número que indica la posición del registro a partir de la cual debe introducirse

el argumento.

Una vez descrito el archivo esquema podemos representar gráficamente la primera parte del sistema por medio del diagrama de bloque mostrado en la figura 1.

Por construcción del programa el archivo de validación es opcional. En caso de no proporcionarse no se obtienen las secciones destinadas a validar la información y el programa COBOL obtendrá solamente el reporte de errores del filtrado. En caso de proporcionarse se debe dar también el archivo de filtrado ya que es condición del programa validar un registro solamente si ha sido filtrado y no se detectaron errores.

Los datos que deben proporcionarse en los archivos de entrada se especifican en el Manual del Usuario.

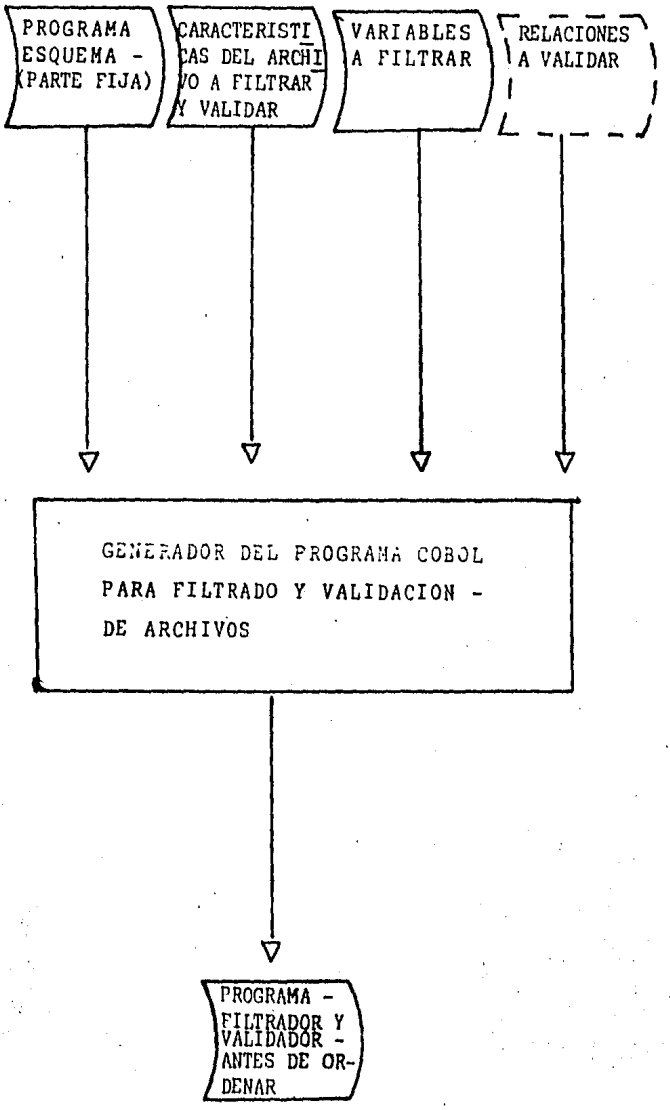

PROGRAMA ESQUEMA - (PARTE FIJA)
CARACTERISTICAS DEL ARCHIVO A FILTRAR Y VALIDAR
VARIABLES A FILTRAR
RELACIONES A VALIDAR
GENERADOR DEL PROGRAMA COBOL PARA FILTRADO Y VALIDACION - DE ARCHIVOS
PROGRAMA - FILTRADOR Y VALIDADOR - ANTES DE ORDENAR

FIGURA 1

II) MODULO CLASIFICADOR

Una vez que se han obtenido las instrucciones COBOL del programa resultante se debe proceder a la ordenación de las mismas. Esta ordenación se lleva a cabo en forma ascendente por las primeras seis posiciones de los registros, y su proceso se efectua por el utilitario correspondiente del equipo en que se este procesando. Este utilitario es comunmente conocido como SORT, su modo de empleo depende de la computadora para la cual fué hecho. En el caso de la PDP-11 bajo sistema operativo RSX-11M este utilitario permite varias alternativas de ordenación que serán seleccionadas por el usuario. En este trabajo se tomo la forma conocida como RECORD SORT en la cual se utiliza el registro completo durante la ejecución del proceso. Dicha forma es apropiada considerando que la longitud del registro es pequeña (80 caracteres). Una vez efectuada la clasificación se tiene un programa COBOL ordenado, aunque con formato incompleto en ciertas instrucciones, estas instrucciones son completadas en la tercera etapa. Un diagrama de bloque del proceso de ordenamiento se presenta en la figura 2.

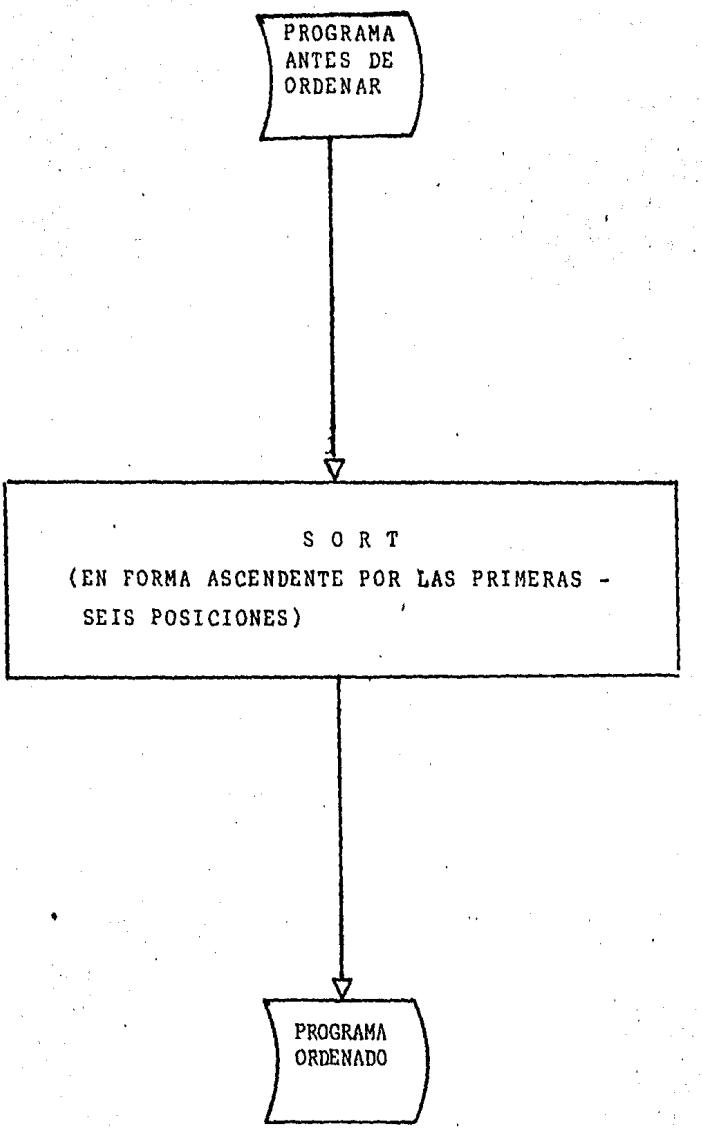

PROGRAMA
ANTES DE
ORDENAR
S O R T
(EN FORMA ASCENDENTE POR LAS PRIMERAS -
SEIS POSICIONES)
PROGRAMA
ORDENADO

FIGURA 2

III) MODULO EDIIOR

Debido a la forma en que fué construido el programa COBOL en la primera fase y a las características de edición de FORTRAN, ciertas instrucciones no siguen el formato establecido por COBOL haciendose necesaria su modificación. En términos generales, estas modificaciones implican la sustitución de ciertos caracteres por otros (por ejemplo espacios por ceros) lo cual puede realizarse facilmente por un editor de textos (utilitario que como el SORT es común a casi todos los equipos de cómputo). En este trabajo se utilizó un editor de textos que puede emplearse en las series PDP 8, 10, 11 y VAX de Digital y que recibe el nombre de TECO (Text Editor and Corrector). Este es un editor sumamente poderoso ya que es programable y por lo tanto la ejecución de esta última fase del trabajo se debe a un programa escrito en TECO que formatea correctamente el programa generado. Un diagrama de bloque del proceso de edición se muestra en la figura 3.

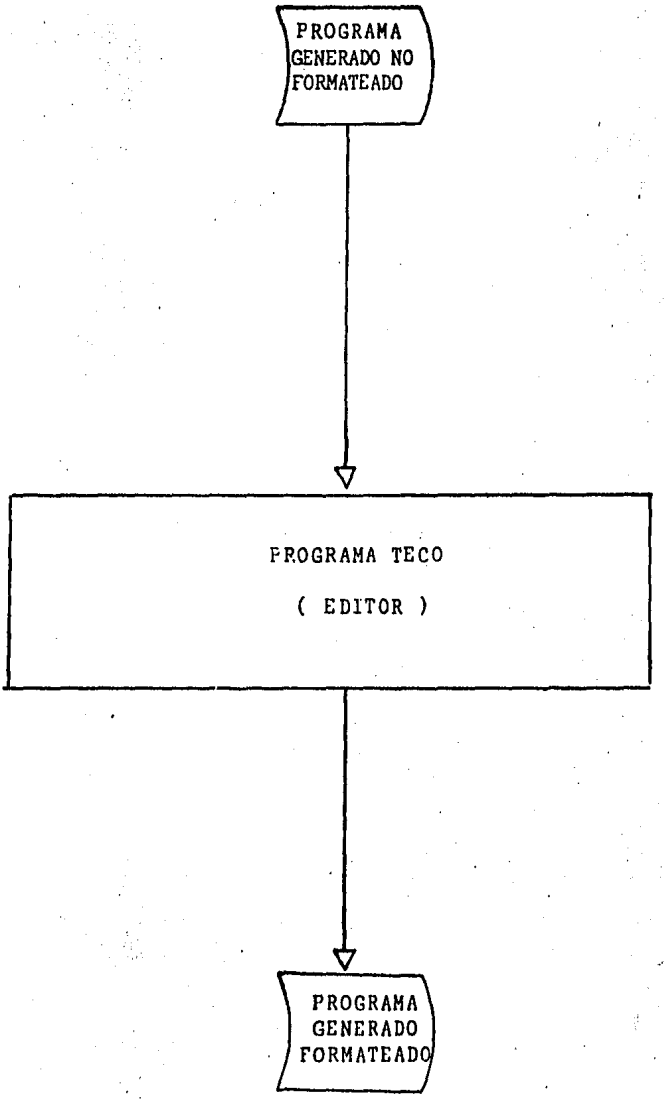

PROGRAMA GENERADO NO FORMATEADO
PROGRAMA TECO
(EDITOR)
PROGRAMA GENERADO FORMATEADO

FIGURA 3

IV) CONSIDERACIONES GENERALES

El programa generado por el sistema imprimirá del archivo de datos del usuario un reporte referente al filtrado de sus datos y opcionalmente un reporte de la validación de sus relaciones en caso que se hayan proporcionado relaciones a validar.

Los reportes impresos del filtrado y en su caso de la validación comparten características generales en sus encabezados, en donde aparecen el nombre del archivo tratado y el número de la página, en ambos aparecen también cifras de control en donde se indican los registros tratados, los correctos y los erroneos. Se reportan solamente los registros erroneos y los formatos de los reportes se muestran en el manual del usuario.

Una vez obtenidos estos reportes y revisadas las cifras de control el usuario podrá determinar si prosigue con el flujo normal de su sistema o en su caso corrige los errores reportados.

El diagrama de bloque de la figura 4 representa el proceso de filtrado y validación de un archivo de datos.

Las tres fases descritas llevarán a la generación de un programa correcto siempre y cuando el usuario observe las reglas establecidas.

DESCRIPCION DEL PROCESO

GENERACION DE DATOS POR EL USUARIO

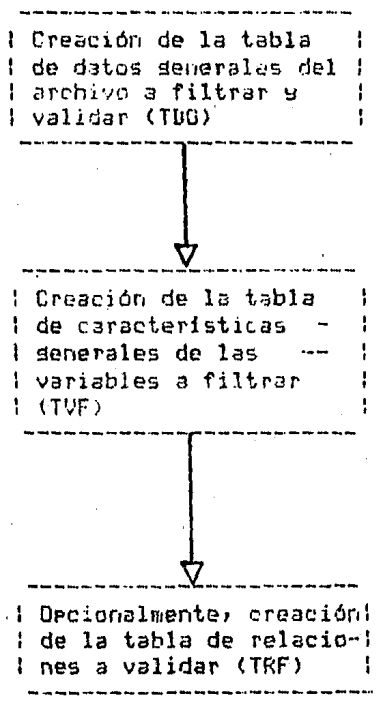


FIGURA 4

CONSTRUCCION DE LA PRIMERA VERSION DEL PROGRAMA RESULTANTE.

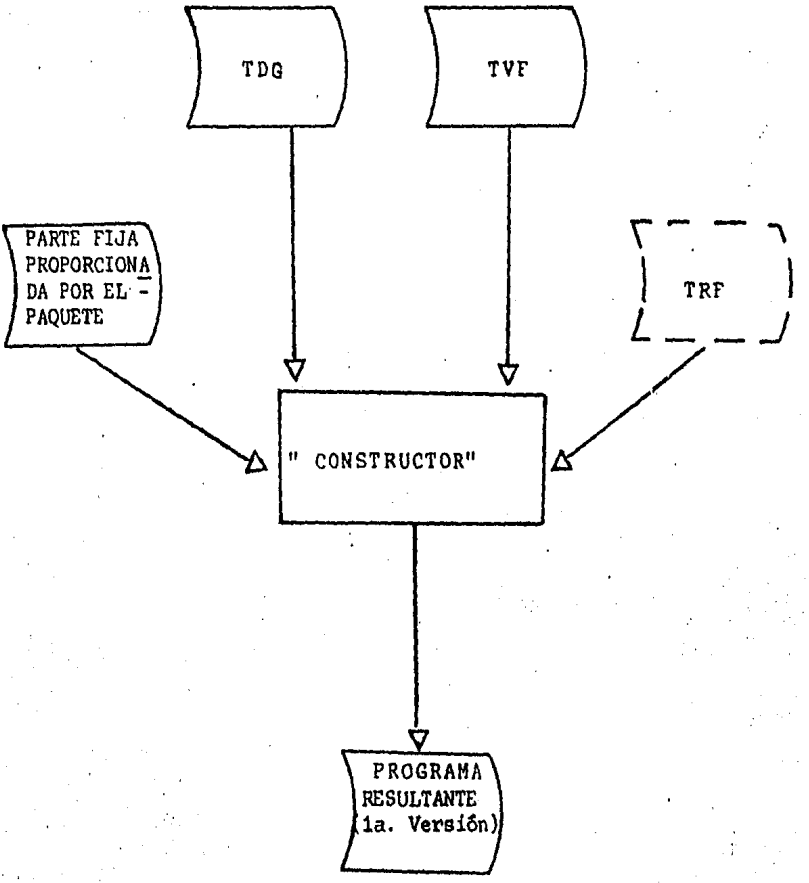


FIGURA 4

CLASIFICACION DEL PROGRAMA RESULTANTE

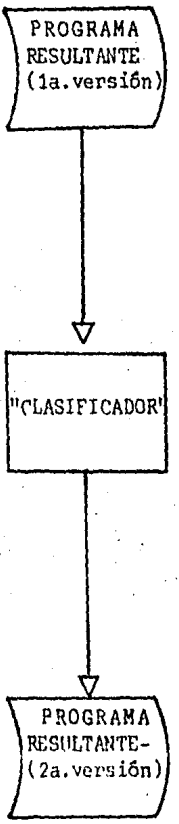


FIGURA 4

EDICION DEL PROGRAMA RESULTANTE.

FIGURA 4

Estas son las siguientes:

Se filtra y valida 1 solo archivo por corrida
Se permite un máximo de 9 tipos de registro
En caso de venir más de 1 tipo de registro estos se numeran
En caso de tener llave, los registros deben de venir ordenados por el valor de la llave, y dentro de esta por número de tipo de registro si es que el archivo tiene más de uno
Los nombres utilizados por el usuario deberán de consistir - unicamente de letras y números
El nombre externo del archivo puede contener un máximo de 20 caracteres, el del dispositivo de almacenamiento 10 y el de - una variable 16
La longitud máxima del registro es de 9999 caracteres
El bloqueaje máximo del archivo es de 9999 registros
Las validaciones se harán solamente con variables de un mismo registro.

Este conJunto de restricciones son necesarias para el correcto funcionamiento del programa generado y su observación facilita el uso del paquete. Debido a que el procesamiento es secuencial se pide que en su caso el archivo se ordene por llave y tipo de registro con objeto de poder detectar completez y duplicidad en los registros, lo cual se puede conseguir facilmente empleando el utilitario SORT. Esta restricción está relacionada con el correcto procesamiento de la información a filtrar y validar, el resto de las restricciones tienen relación

con la correcta generación del programa y su cumplimiento es enteramente responsabilidad del usuario.

Este paquete fue realizado con objeto de simplificar la intervención del usuario teniendo en mente que normalmente es un leso en la materia. El siguiente capítulo tratará del Manual del Usuario, un documento que proporciona al interesado la información necesaria para que pueda usar el generador.

CAPITULO V

MANUAL DEL USUARIO

El presente capítulo tiene por objetivo instruir al usuario en la forma en que debe de introducir los datos para generar el programa filtrador y validador de su archivo, la manera de obtenerlo y la interpretación que debe dar a los reportes obtenidos del mismo. Supone que el usuario conoce las características generales del archivo a filtrar y validar.

El manual se encuentra dividido en cuatro secciones, la primera de ellas contiene antecedentes generales que serán de utilidad para el usuario que no está familiarizado con la organización de los datos para su procesamiento en una computadora, la segunda presenta el diseño de los registros en los diferentes archivos que tiene que proporcionar, la tercera parte da una breve explicación de las variables contenidas en cada archivo y la última proporciona ejemplos individuales que pueden auxiliar al usuario.

I) ANTECEDENTES GENERALES

El usuario del generador tendrá que proporcionar información concerniente a las características de sus datos. Esta información debe ser almacenada en archivos que se encuentran en dispositivos propios donde la computadora puede tener acceso. En este caso, se pueden utilizar como dispositivos de almacenamiento tarjetas ó disco magnético siendo la perforación o el tecleo a través de una terminal los medios de grabación. Un archivo se organiza por medio de registros en donde cada registro tiene una serie particular de datos, mismos que pueden ser capturados en formato libre ó fijo, en el primer caso la separación de los datos se señala explícitamente por medio de un caracter separador, en el segundo los datos guardan siempre una posición fija en el registro. En este trabajo los datos deben proporcionarse en formato fijo que se datalla en su lugar. El empleo del formato fijo fué escogido con objeto de facilitar la programación del paquete. Al mismo tiempo, los archivos generados por el usuario deben de poder reconocerse por un nombre que permita identificarlos completamente.

Se deberán proporcionar dos archivos obligatorios y en su caso uno opcional para poder obtener el programa que filtre y valide el archivo de datos, uno de ellos proporciona información general sobre las características del mismo (obligatorio), otro sobre las características de sus variables (obligatorio), y el último sobre las relaciones a validar si es que estas existen (opcional).

Después de haber obtenido el programa y procesado el archivo de datos del usuario se genera un reporte del filtrado y opcionalmente un reporte de la validación.

En el diseño de los registros se utilizarán las siguientes abreviaturas para la descripción del tipo de variable:

A .- Alfanumérica

I .- Entera

L .- Lógica.

Una variable alfanumérica es una cadena de caracteres cualesquiera.

Una variable entera esta formada solamente por números.

Una variable lógica puede tomar dos valores, verdadero o falso. En el primer caso se codificará la letra T y en el segundo la letra F.

Los datos numéricos deben proporcionarse justificados a la derecha, esto es; el último digito del dato debe coincidir con la última posición del campo; inversamente un dato alfanumérico se justifica a la izquierda, es decir; el primer caracter del dato debe coincidir con la primera posición del campo.

A una variable que no puede subdividirse le llamaremos variable elemental. Las variables que pueden ser subdivididas, como en el caso por ejemplo, de un nombre que se puede

descomponer en nombre de pila, apellido paterno y apellido materno, le llamaremos variables de grupo.

Por facilidad llamaré indicador a la variable numérica que toma los valores cero y uno. Cuando el valor del indicador sea 0 nos estaremos refiriendo a la negación del concepto que representa, y cuando tome el valor 1 a su afirmación. De la misma forma, si la variable utilizada es una variable lógica, el valor T (TRUE) indicará la afirmación del concepto al que se hace referencia, y el valor F (FALSE) la negación del mismo. La información concerniente a algún tipo de registro en particular de los varios que puede utilizar el usuario puede ser representada por un indicador, con objeto de poder referirse al indicador asociado a un tipo de registro por número en vez de nombre el diseño de los registros emplea tablas (colecciones de datos en las que a cada elemento se le referencia por un índice).

Es muy común en el procesamiento de datos el empleo de un conjunto de valores permitidos para una variable en particular, Estos valores toman dos formas principales, una de ellas es aquella en la que los valores se muestran explícitamente, como por ejemplo, en el sexo de una persona los valores se representan por M ó F, la categoría de un empleado en el seguro social tiene por valores permitidos R, S, T, U, V ó W, etc. La otra forma de representar valores permitidos consiste en proporcionar un rango mediante un límite inferior y uno superior, este es el caso de las fechas en las que el día debe estar comprendido entre los valores 1 y 31 inclusive, y el mes entre los valores 1

y 12 inclusive.

El usuario tiene la facilidad de proporcionar este tipo de valores asociados a sus variables. Para poder proporcionar estos valores necesita emplear constantes numéricas y alfanuméricas, las primeras se forman con una serie de números y opcionalmente un punto decimal, las segundas se representan como una serie de caracteres encerrados entre comillas. Cuando el usuario proporciona este tipo de datos se hace necesario que el generador maneje dos tipos de registro para el archivo que contiene la información de las variables; el primer tipo de registro se ocupa para la codificación de las características de la variable y el segundo para la codificación de los valores permitidos de la variable. En caso de que una variable tenga los dos tipos de registro, los registros del tipo 2 deben ser colocados inmediatamente después del registro tipo 1.

Cuando un archivo de datos contiene más de un tipo de registro es usual que registros de diferentes tipos proporcionen información referente al mismo ente. Tal puede ser el caso en un sistema de inventarios, de un producto que contiene una serie de datos repartidos en varios registros. Con objeto de poder identificar el ente a que hace referencia el registro y el tipo de registro del mismo es conveniente que ciertas variables del archivo se destinen a este propósito. A la variable que se utiliza para identificar a un registro se le suele llamar llave del registro, esta variable por su naturaleza puede ser de grupo, en tal caso, debe ser proporcionada al generador como el conjunto

de variables elementales que la forman, en donde cada variable elemental deberá tener el atributo que la identifica como llave del registro. También debe de considerarse que si la llave es una variable de grupo, las variables en que se subdivide deben ocupar posiciones contiguas en el registro.

El empleo de una llave en el registro se utiliza principalmente con objeto de evitar duplicidad en los registros y es opcional, en caso de proporcionarse el programa generado reportará aquellos registros que sean duplicados. (Cuando el archivo tiene llave y un solo tipo de registro, dos registros son duplicados si tienen la misma llave, si el archivo tiene llave y más de un tipo de registro, dos registros son duplicados si tienen la misma llave y el mismo número de tipo de registro).

El usuario debe reservar una variable de su archivo para identificar el tipo de registro en caso que exista más de uno.

Si el usuario proporciona relaciones a validar, estas deberán estar codificadas de acuerdo a la forma que se emplea en COBOL.

Estas consideraciones deberán ser tomadas en cuenta para la correcta elaboración de los archivos que se deben proporcionar al generador y cuyo diseño se presenta a continuación.

II) DISEÑO_DE_LOS_REGISTROS__

ARCHIVO QUE CONTIENE LAS CARACTERISTICAS DEL ARCHIVO A FILTRAR Y VALIDAR. (ARCHIVO 1)

VARIABLE	COLUMNAS	TIPO
Nombre externo del archivo	1-20	A
Dispositivo de almacenamiento	21-30	A
Longitud del registro	31-34	I
Bloqueaje del registro	35-38	I
Tipos de registro	39-39	I
Tabla de nueve indicadores utilizada cuando los registros tienen llave y existe más de un tipo de registro. Determina los tipos de registro que deben ser obligatorios para que la - información relacionada a la - llave se considere completa. (tabla 1)	40-48	I
Tabla de nueve indicadores en - donde cada uno determina si el- tipo de registro en cuestión tiene ó no relaciones a validar. (Tabla 2)	49-57	I

ARCHIVO QUE CONTIENE LAS CARACTERISTICAS DE LAS VARIABLES A FILTRAR. (ARCHIVO 2)

TIPO DE REGISTRO 1

VARIABLE	COLUMNAS	TIPO
Numero de tipo de registro	1-1	I
Nombre de la variable	2-17	A
Tipo de la variable	18-18	L
Indicador que determina si la Variable debe tomar un valor diferente de espacios. (Indicador de Estado)	19-19	L
Indicador que determina si el campo es identificador de tipo de registro.(Indicador de Registro).	20-20	L
Indicador que determina si el campo es llave del registro. (Indicador de Llave)	21-21	L
Posición inicial de la variable	22-25	I
Longitud de la variable	26-29	I
Posiciones para decimales si el campo es numérico.	30-31	I
Campo que determina si la variable posee un conjunto de valores permitidos (Campo de Valores Permitidos)	32-32	I

TIPO DE REGISTRO 2

VARIABLE	COLUMNAS	TIPO
Valores permitidos que puede tomar la variable.	1-40	A
Campo que determina si continuan los valores permitidos (Campo de Continuación).	41-41	I

ARCHIVO QUE CONTIENE LAS RELACIONES A VALIDAR. (ARCHIVO 3)

VARIABLE	COLUMNA	TIPO
Numero de tipo de registro	1-1	I
Relación a validar	2-51	A
Indicador que determina si continua la relación a validar (Indicador de Continuación).	52-52	I

III) DESCRIPCION DE LAS VARIABLES

ARCHIVO 1

Nombre externo del archivo.

Es la identificación por medio de la cual el usuario reconoce a su archivo.

Dispositivo de almacenamiento.

Este dato indica al programa el medio físico en que se encuentra el archivo del usuario. Los dispositivos más comúnes son las tarjetas perforadas, la cinta magnética, el diskette y el disco magnético; estos últimos dos son ahora los más utilizados, el disco en equipos medianos y grandes y el diskette en equipos pequeños que no tienen configurado disco magnético.

Longitud del registro.

Indica el número de caracteres que contiene un registro. Puede tomar un valor de 1 a 9999. (La máxima longitud que puede tener un registro depende también del equipo que se use).

Bloqueaje del registro.

Indica el número de registros que contiene un bloque. Esta medida es utilizada por el sistema operativo para sus operaciones de lectura y escritura. Puede tomar un valor de 1 a 9999.

Tipos de registro.

Indica el número de tipos de registro que tiene el archivo. Puede tomar un valor del 1 al 9.

Tabla 1.

Como se indicó anteriormente, esta tabla es de utilidad cuando el usuario tiene llave en su archivo y más de un tipo de registro. En este caso, una serie de datos asociados a una llave vienen distribuidos en varios registros, donde cada registro se identifica por un número. Los indicadores están asociados a los tipos de los registros y determinan los registros que deben existir para cada llave.

1 Indica que el registro debe existir

0 Indica que el registro puede ó no existir.

Tabla 2.

Esta tabla es necesaria cuando el usuario tiene relaciones a validar. Cada indicador determina si los registros pertenecientes al tipo de registro en cuestión deben validarse o no.

1 Indica que el registro debe validarse

0 Indica que el registro no debe validarse

ARCHIVO 2.

Número de tipo de registro.

Puede tomar un valor de 1 a 9. Si el archivo tiene un solo tipo de registro el valor de este debe ser 1.

Nombre de la variable.

Consiste en un conjunto de letras y números en donde el primer caracter debe ser una letra.

Tipo de la variable

T Indica que el campo es numérico

F Indica que el campo es alfanumérico

Indicador de Estado.

T Indica que la variable debe de tomar un valor diferente a espacios

F Indica en una variable alfanumérica que puede tomar cualquier valor (inclusive espacios), y en una variable numérica que su valor debe ser numérico o bien espacios.

Indicador de Registro.

T Indica que el campo en cuestión es el identificador de registro

F Indica que el campo no es un identificador de registro. En caso de que exista un solo tipo de registro este campo debe tomar siempre el valor F, en caso contrario los registros del archivo del usuario deben de tener un campo que identifique al registro en cuestión y que debe de guardar la misma posición en todos los formatos. Este campo es único, común a todos los registros y debe ser indicado en un solo tipo de registro.

Indicador de Llave.

T Indica que la variable es llave o parte de la llave de un registro.

F Indica que la variable no es llave ni parte de la llave de un registro.
En caso de que el archivo contenga más de un tipo de registro las variables que cumplan con este atributo deben ser declaradas en un solo tipo.

Posición inicial de la variable.

Indica la posición en la que empieza el campo. Puede tomar un valor de 1 a 9999.

Longitud de la variable.

Indica el número de caracteres que tiene la variable. El valor que puede tomar depende de la computadora en cuestión.

Decimales de la variable.

Este dato es proporcionado para las variables numéricas que no sean enteras. Indica el número de dígitos contando de derecha a izquierda que deben de considerarse como decimales. El máximo número de decimales que puede declararse depende del equipo en cuestión. En un campo alfanumérico este campo debe dejarse con espacios.

Campo de Valores Permitidos.

Puede tomar 1 de los siguientes 3 valores:

0.- Indica que la variable no tiene asociado un conjunto de valores permitidos.

1.- Indica que el siguiente registro contiene una serie de valores discretos (esto es, - dados de uno en uno) asociados a la variable.

2.- Indica que el siguiente registro contiene un rango de valores permitidos asociados a la variable.

TIPO DE REGISTRO 2

Valores permitidos que puede tomar la variable.

En el caso en que los valores se den uno a uno, estos se codificaran separados cuando menos por un espacio y podrán introducirse en un registro tantos como este pueda contener. Si los valores se dan por rango el registro debe contener el límite inferior, un guión y el límite superior en el orden descrito, y se permitirá un solo rango por registro. Observese que el límite inferior y el límite superior forman parte del rango de valores, y que en ningún caso es posible mezclar las dos formas en que se pueden proporcionar estos valores en un mismo registro, no obstante es posible asociar ambas formas a una variable proporcionandolas en distintos registros. Ejemplos de posibles formas de representar estos valores se encuentran en la última sección del capítulo.

Campo de Continuación.

0 - Indica el fin de valores permitidos en la variable.

1 - Indica que el siguiente registro contendrá valores expresados en forma discreta.

2 - Indica que el siguiente registro contendrá valores expresados por rango.

Este campo proporciona al usuario una forma más agil de introducir sus valores ya que le permite mezclar las dos formas de representación descritas en la primera sección del capítulo.

ARCHIVO 3.

Número de tipo de registro.

Este dato indica el tipo al que pertenece el registro. Puede tomar un valor de 1 a 9 y en caso de que el archivo tenga un solo tipo de registro debe de codificarse un 1.

Relación a validar.

El usuario proporciona aquí la relación que quiere que se valide considerando que todas las variables que intervengan en la misma deben de pertenecer al mismo registro. Las relaciones a validar son de tipo aritmético.

Indicador de Continuación.

Este indicador permite al usuario extender su relación en varios registros. Los valores son:

0.- Indica el principio de una relación

1.- Indica que el contenido del registro
es continuación de la relación anterior

REPORTE DEL FILTRADO DEL ARCHIVO.

Se imprime el registro en divisiones de 100 caracteres, los campos erroneos se subrayan con dos tipos de caracteres (*,@) en forma alternada y a la derecha de la división se señala con claves los tipos de errores que se cometieron.

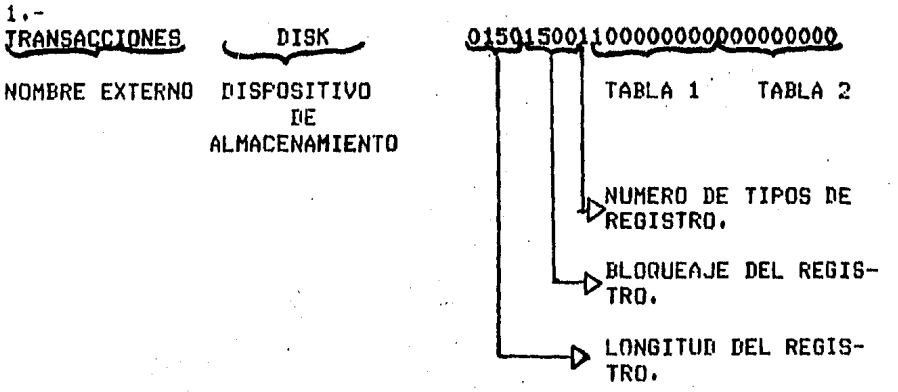
REPORTE DE LA VALIDACION DEL ARCHIVO.

Como en el caso anterior, los registros se imprimen en divisiones de 100 caracteres, una vez que el registro se ha terminado de imprimir, se imprimirá una línea en la que aparecerán los números de las relaciones que se detectaron con error. Las relaciones se numeran dentro de cada registro en el orden en que el usuario las va introduciendo.

IV) EJEMPLOS

ARCHIVO 1

1.-



En este ejemplo, el usuario proporciona un archivo que se llama TRANSACCIONES y se encuentra en disco (en este caso, la forma de representar al disco como medio de almacenamiento fué tomada del COBOL de Burroughs 6700). La longitud del registro es de 150 caracteres y su bloqueaje es de 1500 caracteres. El archivo tiene un tipo de registro único. Los valores 2 al 9 de las dos tablas toman el valor 0 debido a que el archivo tiene un solo tipo de registro. El primer elemento de la tabla 1 toma el valor 1 indicando que el primer tipo de registro (y en este caso único) del archivo debe de existir. Dado que el registro no tiene relaciones a validar, el primer elemento de la tabla 2 toma el valor 0.

2.-

MULTIPLE DISK 005010003101000000000000000

El archivo MULTIPLE tiene 3 tipos de registro, de los cuales el 1 y el 3 deberán de existir para todas las llaves. El registro 2 puede ó no venir. Los registros no tienen relaciones a validar.

3.-

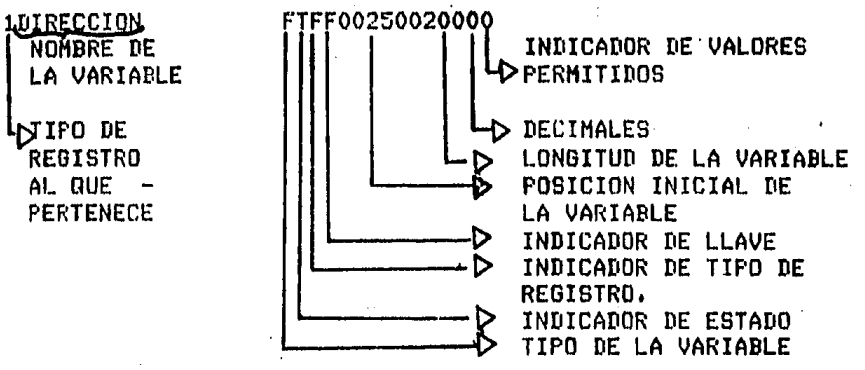
MUESTRA DISK 012525004010100000001100000

El archivo MUESTRA contiene 4 tipos de registro. Los registros de tipo 2 y 4 deben existir para todas las llaves, y los registros 1 y 3 pueden ó no existir. En este caso, los registros 3 y 4 tienen relaciones que se validarán.

ARCHIVO 2

TIPO DE REGISTRO 1

1.-



LA VARIABLE DIRECCION pertenece al registro tipo 1, es alfanumérica y debe contener un valor diferente a espacios, no es llave ni indicador de registro, empieza en la columna 25 con una longitud de 20 caracteres.

2.-

2ESTATURA TFFF0F00500003020

La variable ESTATURA pertenece al registro tipo 2, es numérica de 2 decimales y su valor puede ser espacios. No es indicador de tipo de registro ni llave, empieza en la posición 50.

3.-

```
3REFACCION          FTFT00010005000
```

La variable REFACCION pertenece al tipo de registro 3, es alfanumérica y llave de registro. Esta variable debe tener forzosamente un valor y no puede ser declarada en otro tipo de registro.

4.-

En este ejemplo, consideremos el caso de una variable de grupo que esta formada por dos campos, el primero es una clave de procedencia de un artículo, de dos posiciones y de tipo alfanumérico, el segundo es un identificador del artículo, numérico y de 5 posiciones. Supondremos además que esta variable es la llave del registro. La forma de codificar estos datos es la siguiente.

```
1PROCEDENCIA        FTFT00010002000
1CLAVE              TTFT00030005000
```

Observese que los dos campos ocupan posiciones contiguas en el registro.

La codificación de una variable que identifique el tipo de registro al que pertenece el registro es similar a la presentada en el siguiente ejemplo.

5.-

```
1TIPO                TTTF00100001000
```

Como en el caso de las variables declaradas como llaves de registro, esta variable debe ser codificada en un solo tipo de registro, su presencia en el registro es obligatoria y debe ser, además, de tipo numérico y de una sola posición. En este caso, esta posicion se localiza en la columna 10.

TIPO DE REGISTRO 2

En los siguientes ejemplos, se presentarán variables que tienen un conjunto de valores permitidos asociados. En primer término se presenta la descripción de la variable tal como se vio en los ejemplos anteriores, inmediatamente se muestra la forma de proporcionar sus valores permitidos. Notese que en la descripción de la variable, la última posición del registro tiene codificado el valor 1 ó 2, indicando que los siguientes registros contienen los valores asociados a la variable.

6.-

```
1BINARIA             TFFF00150001001
  0 1                                  0
  VALORES PERMITIDOS                     INDICADOR DE
  PARA LA VARIABLE                       CONTINUACION
                                         DE VALORES
                                         PERMITIDOS.
                                         EN ESTE CASO
                                         LA VARIABLE
                                         NO TIENE MAS
                                         VALORES PERMI-
                                         TIDOS.
```

Los valores permitidos para esta variable son 0 y 1

7.-

```
1GRUPO              FTFF00200001001
  "R"   "S"   "T"   "W"              0
```

Los valores que puede tomar la variable GRUPO son "R", "S", "T", "W".

8.-

```
1RANGO              TTFF00850089022
50.0-100.0                         0
```

Los valores que puede tomar rango son del 50 al 100 inclusive.

9.-

Este ejemplo presenta una variable cuyos valores permitidos se proporcionan en las dos formas posibles y se extienden en varios registros.

```
1GLOBAL             TTFF00750007001
5 10 15                            1
20 50 100                          2
250 - 500                          2
800 - 1100                         1
10000 20000                        2
500000 - 1000000                   0
```

En este caso, los registros cuya última posición tiene el valor 1 indican que el siguiente registro tiene valores asociados dados en forma discreta, el valor 2 indica que el siguiente registro tiene valores asociados dados por rango y el valor 0 indica la terminación de valores asociados a la variable. La variable GLOBAL puede tomar por lo tanto los valores 5,10,15,20,50,100,10000,20000, y los rangos del 250 al 500,800 al 1100 y 500000 al 1000000.

ARCHIVO 3

Supondremos en los siguientes ejemplos que los nombres de las variables vienen dados por letras del alfabeto. El usuario debe proporcionar relaciones de variables contenidas en el mismo registro presentadas de acuerdo a las reglas de COBOL.

1.-

A = B	0	
RELACION		INDICADOR DE CONTINUACION.
A VALIDAR		EN ESTE CASO LA RELACION
		ESTA COMPLETA.

Se valida la igualdad entre A y B.

2.-

A + B > C - D 0

Se valida que la suma de A y B sea mayor que la resta de C y D.

3.-

A < B AND X=Y 0

Se valida que A sea menor que B y además que X sea igual a Y.

El usuario puede extender una relación en varios registros, en este caso el indicador de continuación debe tomar el valor 1 excepto en el último registro de la relación en el que debe indicar el fin de la relación codificando el valor 0. Este ejemplo se ilustra a continuación:

4.-

```
A + (B/C)NOT>X          1
AND D / E * F = Y       1
OR (K - L > M - N       1
    AND W > P - Q)      0
```

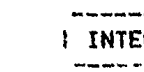
En este caso se valida que la suma de A con el cociente de B y C no sea mayor a X y que el producto del cociente de D y E por F sea igual a Y o que la resta de K y L sea mayor a la resta de M y N y W sea mayor a la resta de P y Q.

CAPITULO VI

PROGRAMACION DEL PAQUETE

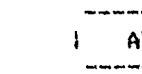
El paquete para filtrado y validación de archivos está formado por un programa FORTRAN que genera un programa COBOL de filtrado y validación, un conjunto de comandos que ejecutan el utilitario SORT con objeto de ordenar el programa COBOL y un archivo de comandos de edición que permite al programa COBOL ser compilado y producir un programa objeto. Las diferencias principales entre las dos partes programadas fueron el lenguaje utilizado y las estructuras de los mismos. En la primera parte del paquete se utilizo el lenguaje FORTRAN y el programa contiene una rutina principal y varias subrutinas.En la tercera parte se utilizo el editor TECO y el programa consiste de una rutina única. En ambos casos el método que se empleo para describir las rutinas o subrutinas consistio en definir las variables de entrada y salida que se utilizan, explicar la función que realiza y presentar un diagrama de flujo. En el caso del programa FORTRAN la presentación se hace "de abajo hacia arriba" con objeto de que en todo caso se encuentren completamente definidos todos los elementos que se usan.

En el diagrama de flujo de la rutina principal se encontrará la figura



esta figura representa un procedimiento que consiste en la selección de un conJunto de subrutinas y se describe una vez que se han definido estas.

De igual manera, se encontrara en varios diagramas de flujo la figura



que representa el proceso de añadir al programa resultante una o varias instrucciones, y que se define después de la descripción de la subrutina MOVE.

La mayor parte de las subrutinas reciben valores a través de instrucciones COMMON(*), las variables comprendidas en estas instrucciones son de caracter general y por tanto se repiten varias veces en diferentes subrutinas.

(*) Una proposición COMMON es utilizada en FORTRAN para declarar variables que serán compartidas por diferentes unidades del programa.

Dentro de la documentación de las subrutinas se definen solamente aquellas variables que son de importancia especial para el funcionamiento de las mismas, por separado se definen todas las variables utilizadas en las instrucciones COMMON indicando: el tipo de la variable, el mnemotécnico empleado y la posición que guardan en dicha instrucción.

Para definir las variables de entrada y salida a las subrutinas se utilizara la siguiente nomenclatura:

I) Variable de entrada

O) Variable de salida

I-O) Variable de entrada y salida.

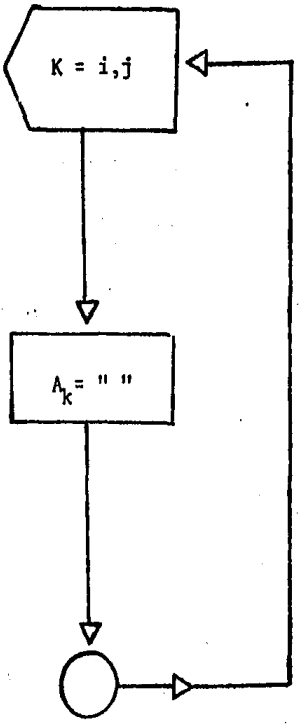
SUBRUTINA CLNREC (A, i, J)

ARGUMENTOS

I-O) Arreglo (A)

I) Indice inferior (i)

I) Indice superior (J)



Esta subrutina pone blancos en un arreglo del elemento i al J. Se utiliza en todos aquellos arreglos que reciben cadenas de número variable de caracteres y se emplea en casi todas las subrutinas.

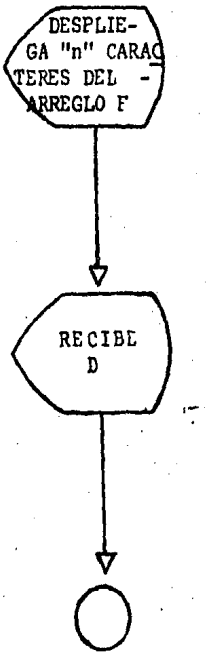
SUBRUTINA DISP (n, F, D)

ARGUMENTOS

I) Número de caracteres que se despliegan en pantalla (n)

I) Arreglo que contiene los caracteres a desplegar (F)

O) Arreglo en que se recibe una cadena de caracteres (D)

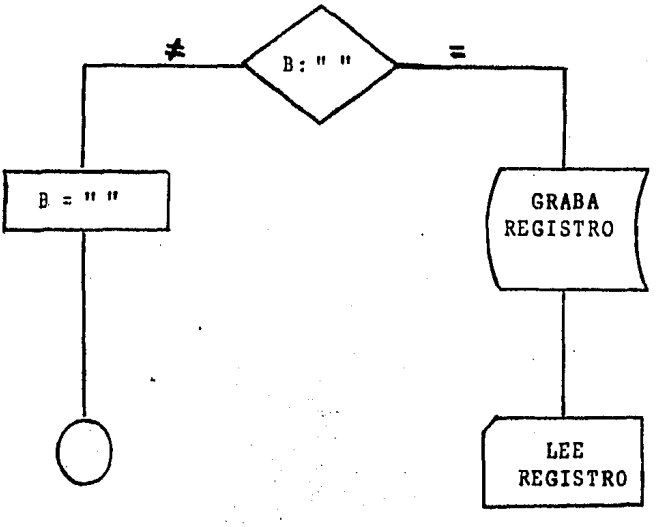


La subrutina despliega en pantalla "n" caracteres del arreglo F en donde se pide al usuario que proporcione el nombre de un archivo (que según el caso puede ser cualquiera de los descritos). Este nombre se almacena en el arreglo D que posteriormente se utiliza para abrir el archivo.

SUBRUTINA CPYFL

ARGUMENTOS

I-O) Bandera puesta en COMMON que indica en cada registro si este debe ser copiado o no al programa resultante. (B)



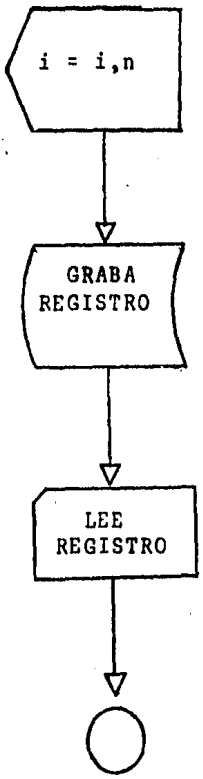
La subrutina copia registros del archivo esquema al programa resultante hasta que la bandera(*) de algún registro sea diferente al espacio.

(*) Una bandera es una variable que un programa utiliza para tomar una acción dependiendo del valor que tenga.

SUBRUTINA CPYRCS(n)

ARGUMENTOS

I) Número de registros a copiar (n)



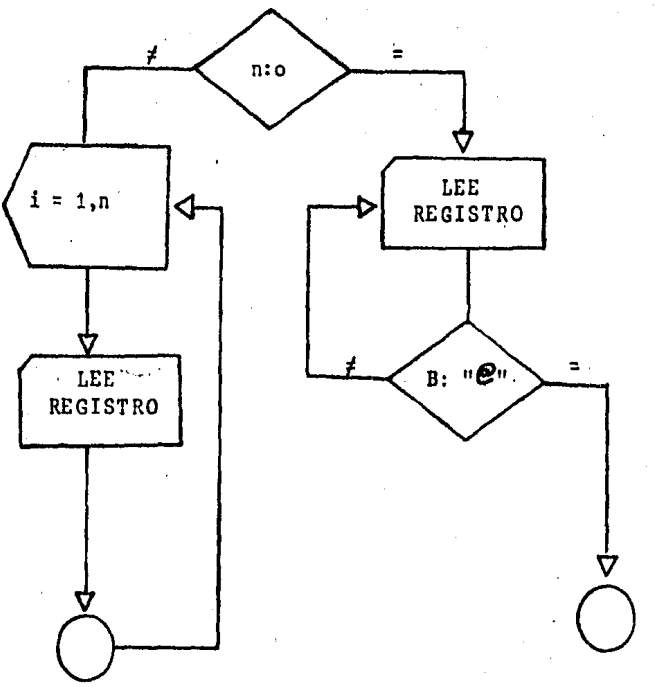
Esta subrutina copia un número de registros "n" del archivo esquema al programa resultante.

SUBRUTINA PSSRCS(n)

ARGUMENTOS

I) Número de registros a saltar (n)

I) Bandera del registro pasada en COMMON (B)



La subrutina salta registros en el archivo esquema. Si n=0 saltara registros hasta encontrar el caracter "@" en una bandera. En caso contrario, saltara un número fijo "n" de registros.

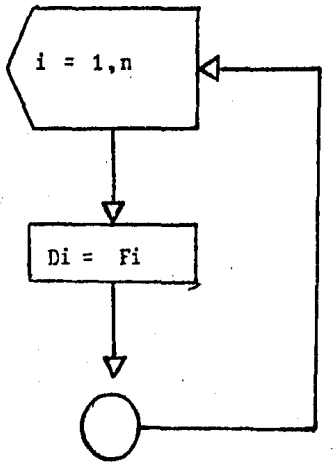
SUBRUTINA MOVE (n, F, D)

ARGUMENTOS

I) Número de caracteres a mover (n)

I) Arreglo donde se encuentran los caracteres (F)

O) Arreglo donde se dejan los caracteres (D)



La subrutina copia caracteres de un arreglo a otro. Dado que los arreglos pueden pasarse con índice, se puede copiar o recibir desde cualquier posición del arreglo.

PROCESO DE ADICION

Este proceso anexa instrucciones al programa resultante cuyo contenido depende del punto en el que se encuentre el programa. La instrucción se graba desde un arreglo en memoria de 80 posiciones que representa el registro COBOL. En las primeras 6 posiciones se proporciona a este arreglo un número de secuencia que se utiliza posteriormente para ordenar el programa COBOL. El resto del registro toma valores de algún otro arreglo en memoria a través de la subrutina MOVE. Estos arreglos son de propósito especial y representan instrucciones COBOL que se utilizan en el programa, y que contienen una parte fija y una parte variable cuyo valor se obtiene con la inserción de un argumento a través de la subrutina MOVE o de la instrucción ENCODE.

PROCESO DE INTEGRACION

Utiliza las subrutinas descritas anteriormente y su función consiste en integrar o dejar fuera del programa resultante las instrucciones del programa esquema. La función de integrar instrucciones se efectua con las subrutinas CPYFL ó CPYRCS(n) y la de saltarlas con la subrutina PGSRCS(n). En caso de que se tenga que integrar una instrucción que se ecuentra incompleta, esta se completara primero con la inserción del argumento correspondiente a través de la subrutina MOVE.

La forma en que las subrutinas se combinen darán un proceso de integración particular, y dependerá del punto en que se encuentre el programa.

SUBRUTINA FLSVAL

ARGUMENTOS (pasados en COMMON)

I-O) Variables y arreglos de uso general en el programa.

Esta subrutina se llama una sola vez al inicio del programa y asigna valores a variables y arreglos de uso general en el programa. Las variables y los arreglos que toman el mismo valor en todos sus elementos se inicializan con instrucciones DATA. Los arreglos que no están en este caso se inicializan con la subrutina MOVE.

SUBRUTINA INTL

ARGUMENTOS (Pasados en COMMON)

I-O) Arreglos de uso general en el programa

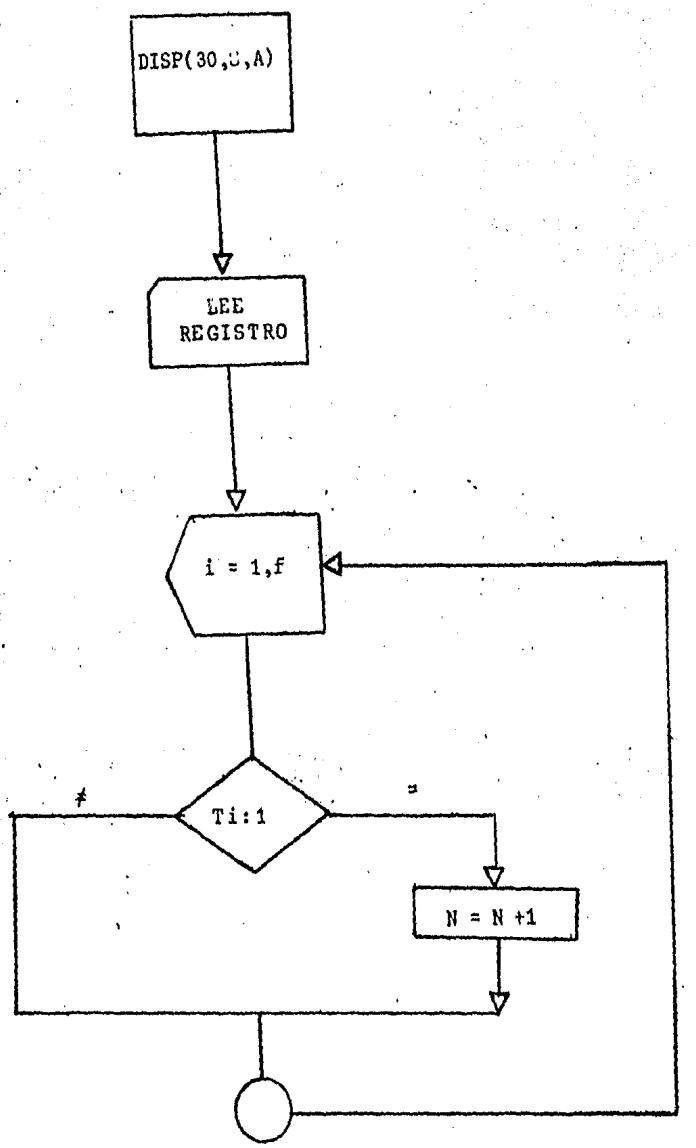
I) Número de tipos de registro que contiene el archivo (f)

I) Tabla asociada a los tipos de registro que indica cuales de ellos contienen relacions a validar (T)

O) Número de tipos de registro que tienen relaciones a - validar (n)

I) Cadena de caracteres en donde se pide al usuario el - nombre del archivo que contiene los datos generales del archivo a filtrar y validar (S)

O) Arreglo donde recibe este nombre (A)


DISP(30,C,A)
LEE REGISTRO
i = 1,f
Ti:1
≠
=
N = N +1

Esta subrutina abre y lee el único registro del archivo que contiene los parámetros generales del archivo a filtrar y validar. Con estos datos determina el número de tipos de registro que tienen relaciones a validar y posteriormente, en el proceso de INTEGRACION completa clausulas para el programa resultante.

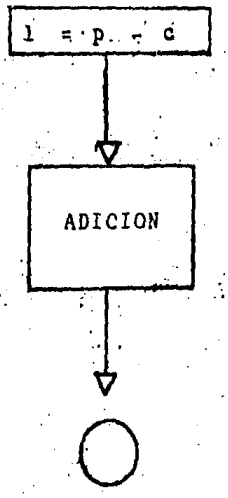
SUBRUTINA GENFLR

ARGUMENTOS (pasados en COMMON)

I) Posición en la que principia la variable (p)

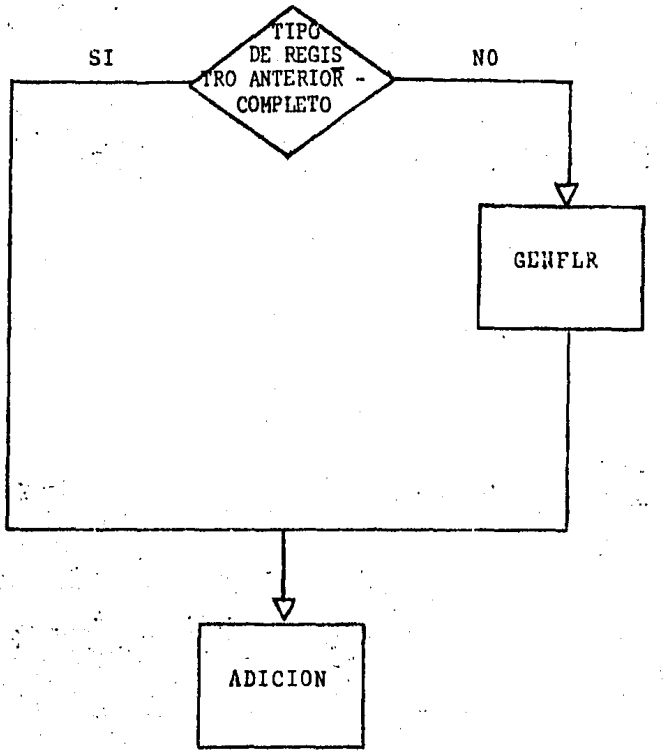
I) Posición en la que termina la variable anterior (c)

O) Espacio que existe entre ambas variables (l)



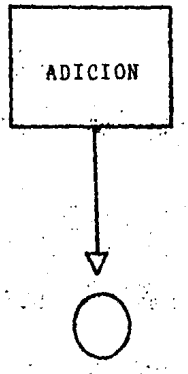
Calcula el espacio que existe entre dos variables que no son contiguas. En este caso, el programa añade una instrucción 'FILLER' al programa resultante.

SUBRUTINA CHFMT



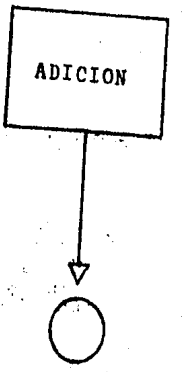
Se emplea cada vez que en el archivo de variables se detecta un cambio de tipo de registro. La subrutina genera una instrucción 'FILLER' para completar la longitud del registro en caso necesario y genera dos nuevas instrucciones; la primera contiene el encabezado de la descripción del nuevo tipo de registro, la segunda el nombre del párrafo en donde se harán los filtros de este tipo de registro.

SUBRUTINA TBLSVL



Genera las instrucciones MOVE necesarias para asignar valores a elementos de tablas tomando los valores numéricos del archivo de variables.

SUBRUTINA CPYID



Genera una instrución MOVE que copia el valor de la variable que es identificador de tipo de registro a una variable reservada.

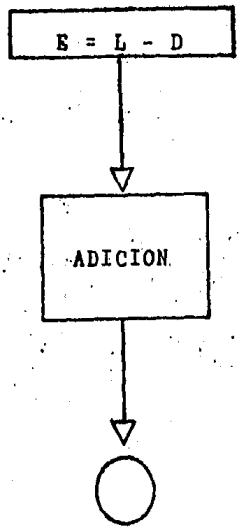
SUBRUTINA NUMPIC

ARGUMENTOS (pasados en common).

I) Longitud de la variable numérica (L)

I) Decimales que ocupa (D)

O) Longitud de la parte entera (E)



Calcula la longitud de la parte entera y completa el arreglo destinado a representar instrucciones PIC asociadas a cada variable en la DATA DIVISION.

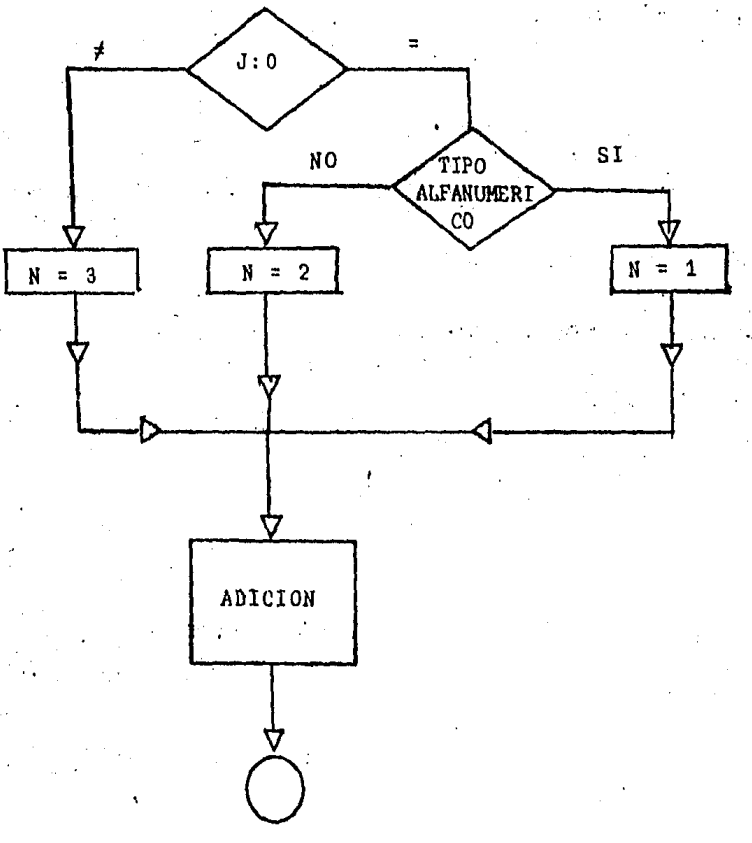
SUBRUTINA EDTFLD

ARGUMENTOS (pasados en COMMON)

I) Tipo del campo a filtrar (t)

I) Variable que indica si el campo tiene valores asociados (J) o no (0) (J)

O) Número de error en el filtro (n)



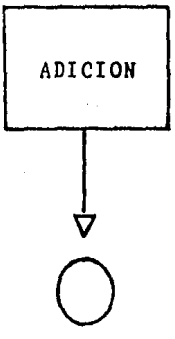
Genera instrucciones para manejar el tipo de error, que formarán parte de una instrucción IF. La condición que puede presentar una variable para su filtrado es una de las siguientes:

i) Es alfanumérica, no tiene valores asociados
y debe ser diferente a espacios

ii) Caso i para una variable numérica

iii) Tiene valores asociados.

SUBRUTINA RDFTST.

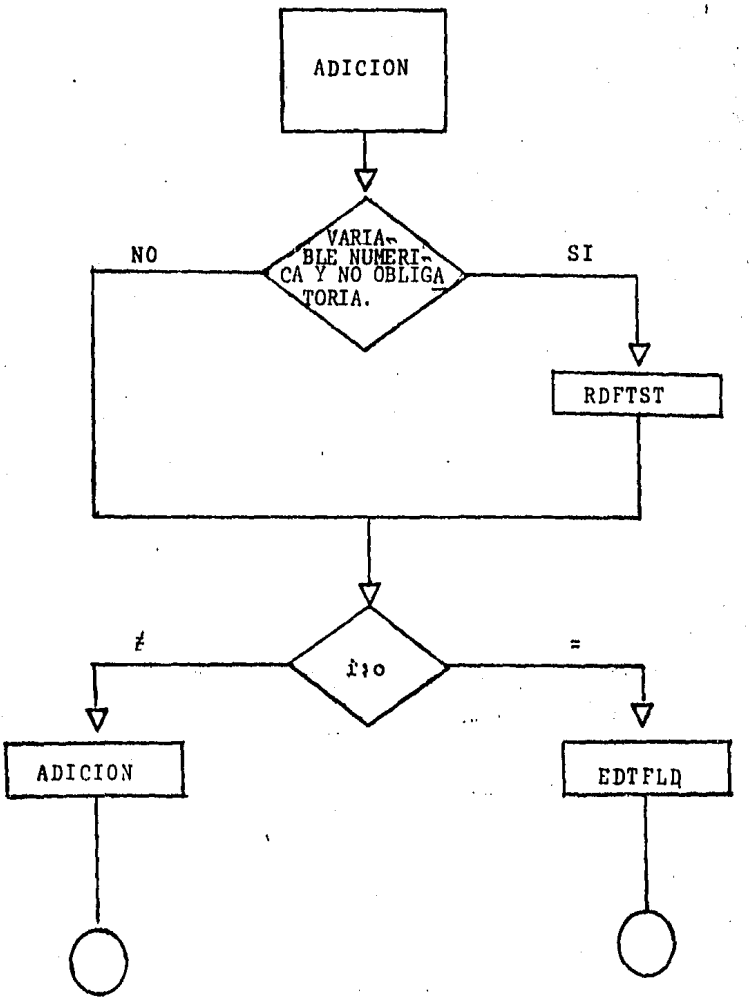


Genera una condición simple en la que se determina si un dato redefinido es diferente a espacios, y que forma parte de una condición compuesta en una instrucción IF.

SUBRUTINA RPTFLD.

ARGUMENTOS (pasados en COMMON.)

I) Variable que indica si el campo a filtrar tiene valores permitidos (1) ó no (0) (i).

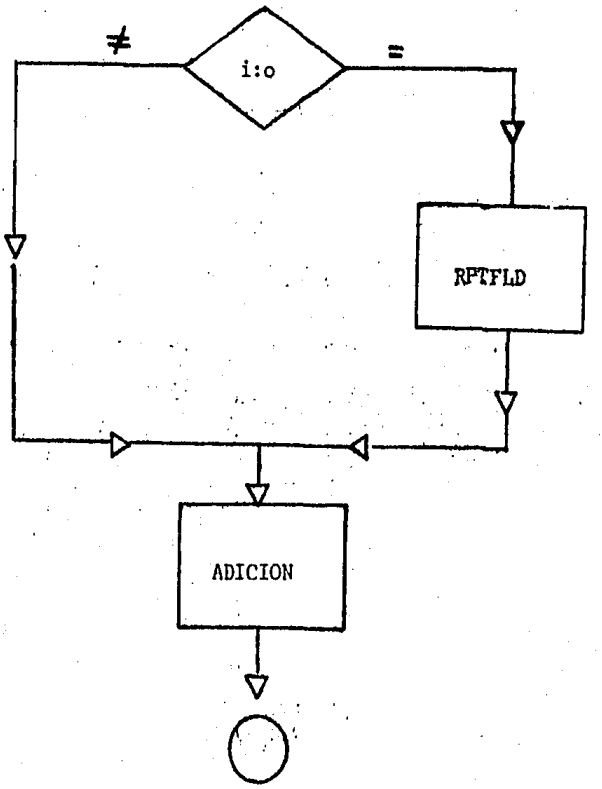


Construye la primera parte de la instrucción IF en la que se incluye la prueba para el filtrado, en el caso de una variable numérica no obligatoria, la prueba permite que la variable tome el valor de espacios; posteriormente genera instrucciones dependiendo de que la variable tenga valores asociados ó no. Si la variable no tiene valores asociados se termina de construir la instrucción IF, en caso contrario, se generan instrucciones que permiten pasar la variable sin filtrar si esta tiene espacios como valor ó encadenar en caso contrario a la prueba sobre el rango.

SUBRUTINA CNDVAL

ARGUMENTOS (pasados en COMMON)

I) Variable que identifica si el campo debe ser diferente a espacios (1) o no (0). (i).



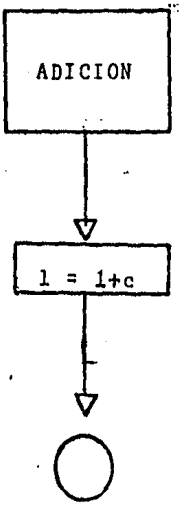
Genera instrucciones para filtrar una variable que tenga asociados valores permitidos. Si la variable puede ser igual a espacios genera primero instrucciones que permitan pasarla sin filtrar. Posteriormente, genera las instrucciones para filtrar sobre los valores permitidos.

SUBRUTINA KEYREC

ARGUMENTOS (pasados en COMMON)

I) Longitud del campo (c)

O) Longitud de la llave (l)

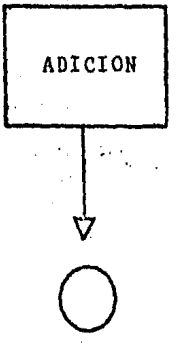


Dado que la llave del registro puede venir en forma particionada, esta subrutina genera instrucciones que copian estas partes a una variable de grupo con objeto de poder utilizarse posteriormente en el programa resultante. Así mismo, se obtiene la longitud de la llave que será utilizada por el programa generador.

SUBRUTINA GRFREC

ARGUMENTOS (Pasados en COMMON)

I) Longitud del campo (L)



Genera una redefinición de un campo numérico como alfanumérico de longitud L.

SUBRUTINA PRSMFL

ARGUMENTOS (Pasados en COMMON)

I) Número de tipo de registro de la variable (TR)

I) Número de tipo de registro de la variable anterior (TRA)

I) Número de tipos de registro que contiene el archivo a filtrar y validar (NTR)

I) Variable que indica si el campo es identificador de tipo (JTR)

I) Posición inicial de la variable (PIV)

I-O) Posición final de la variable anterior (PFV)

I) Tipo de la variable, numérico (1) o alfanumérico (0) (t)

I) Decimales de la variable (d)

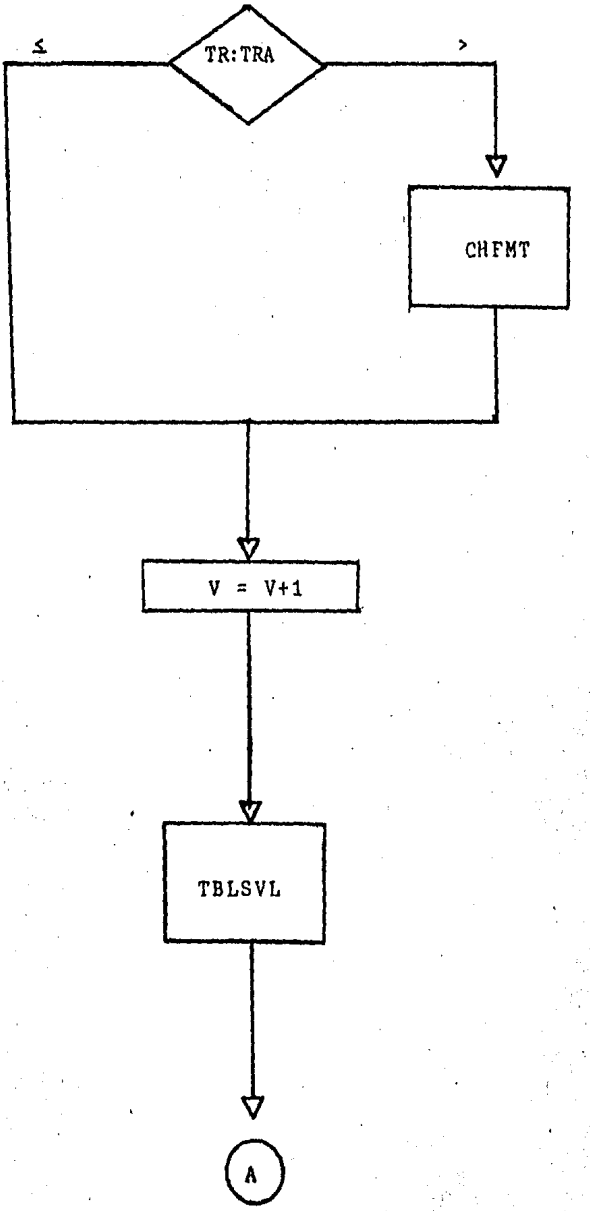
I) Variable que indica si el campo tiene valores asociados (1) o no (0) (FVA)

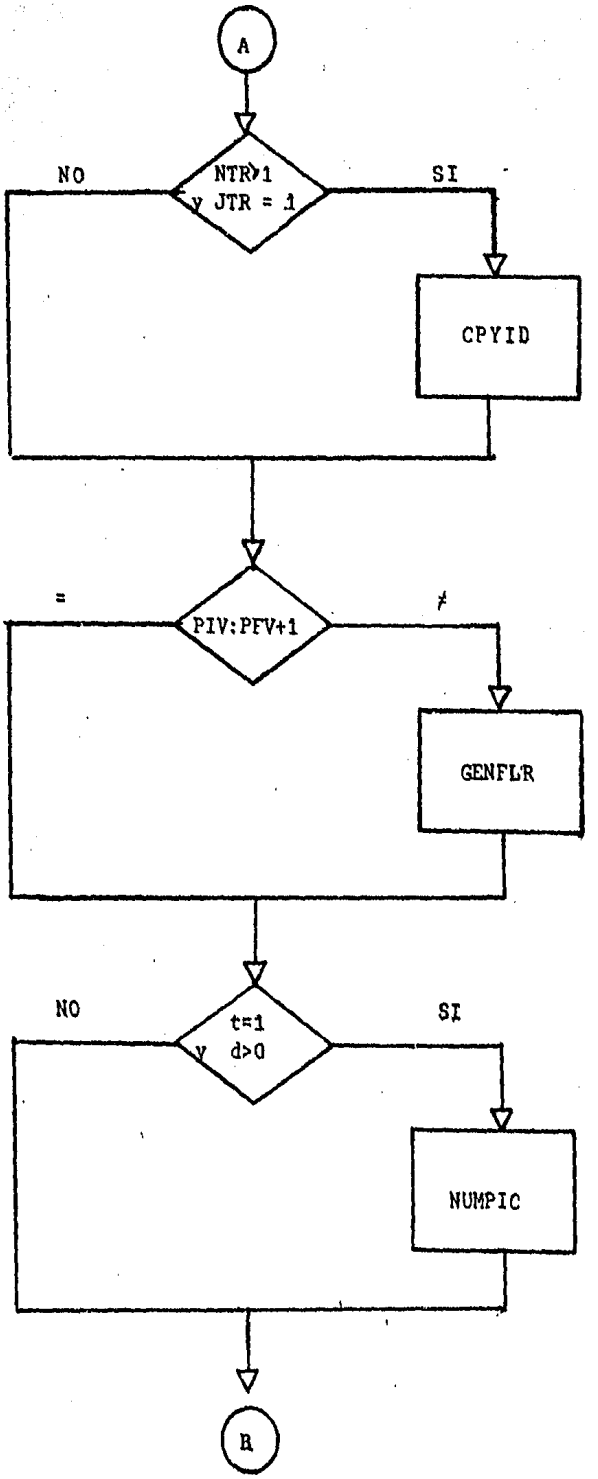
I) Variable que indica si el campo forma parte de la llave (1) o no (0) (IL)

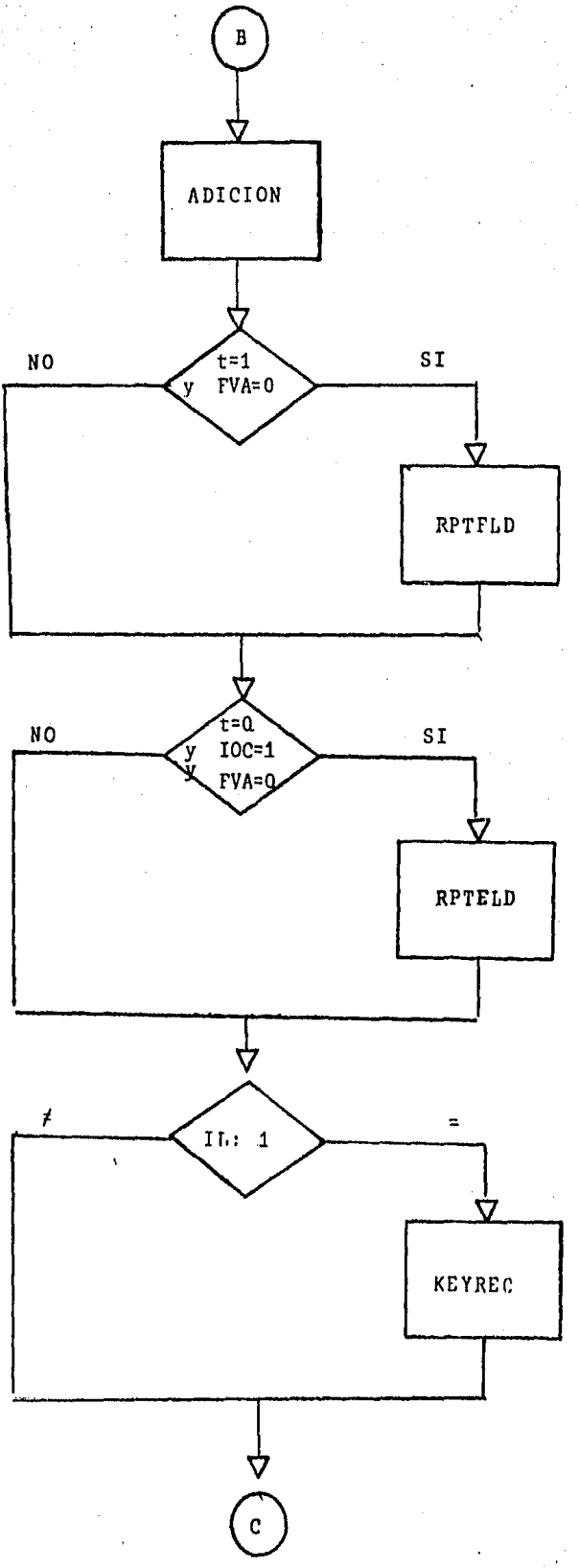
I) Longitud de la varialble (l)

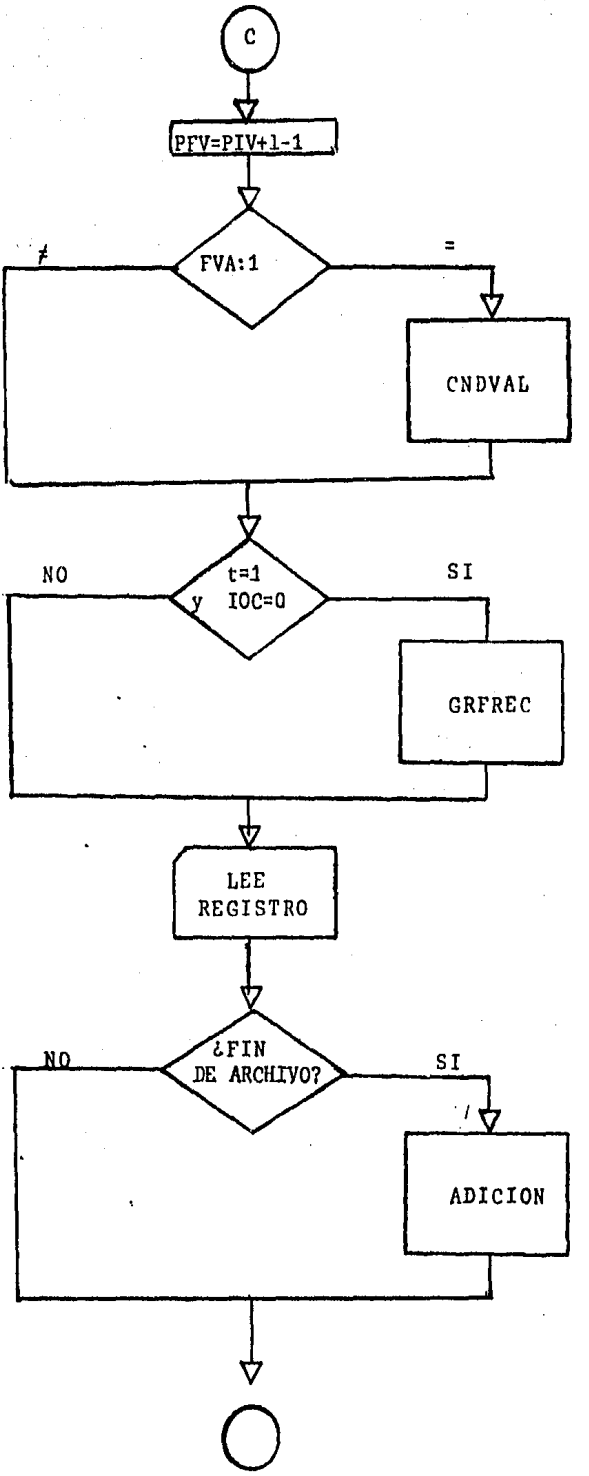
I-O) Número de variable dentro del tipo de registro (v)

I) Variable que indica si el campo puede ser igual a espacios (0) o no (1) (IOC)


TR:TRA
≤
>
CHFMT
V = V+1
TBLSVL
A


A
NO
NTR>1
y JTR = 1
SI
CPYID
=
PIV:PFV+1
≠
GENFLR
NO
t=1
y d>0
SI
NUMPIC
B


B
ADICION
NO
t=1
y FVA=0
SI
RPTFLD
NO
t=0
y IOC=1
y FVA=0
SI
RPTELD
≠
=
KEYREC
C


C
PFV=PIV+1-1
FVA:1
≠
=
CNDVAL
t=1
y IOC=0
NO
SI
GRFREC
LEE
REGISTRO
¿FIN
DE ARCHIVO?
NO
SI
ADICION

Efectua las llamadas a todas las subrutinas que intervienen en el filtrado de las variables con objeto de generar las instrucciones adecuadas. Las variables deben venir ordenadas por tipo de registro y, dentro de este, por posición en el registro. La subrutina numera las variables por tipo de registro con propósito de utilizarlas como indices en diversos puntos. Al alcanzar el fin de archivo se efectua un proceso de adición de instrucciones. Cuando existe más de un tipo de registro el programa resultante trabaja con una tabla en la que cada elemento indica con un "1" si el tipo de registro es obligatorio y con un "0" si no lo es. Análogamente, se trabaja con una tabla en la que un "1" indica que el tipo de registro contiene relaciones a validar y un "0" que carece de ellas. En este proceso de adición se generan las instrucciones necesarias para que los elementos de estas tablas tomen los valores debidos.

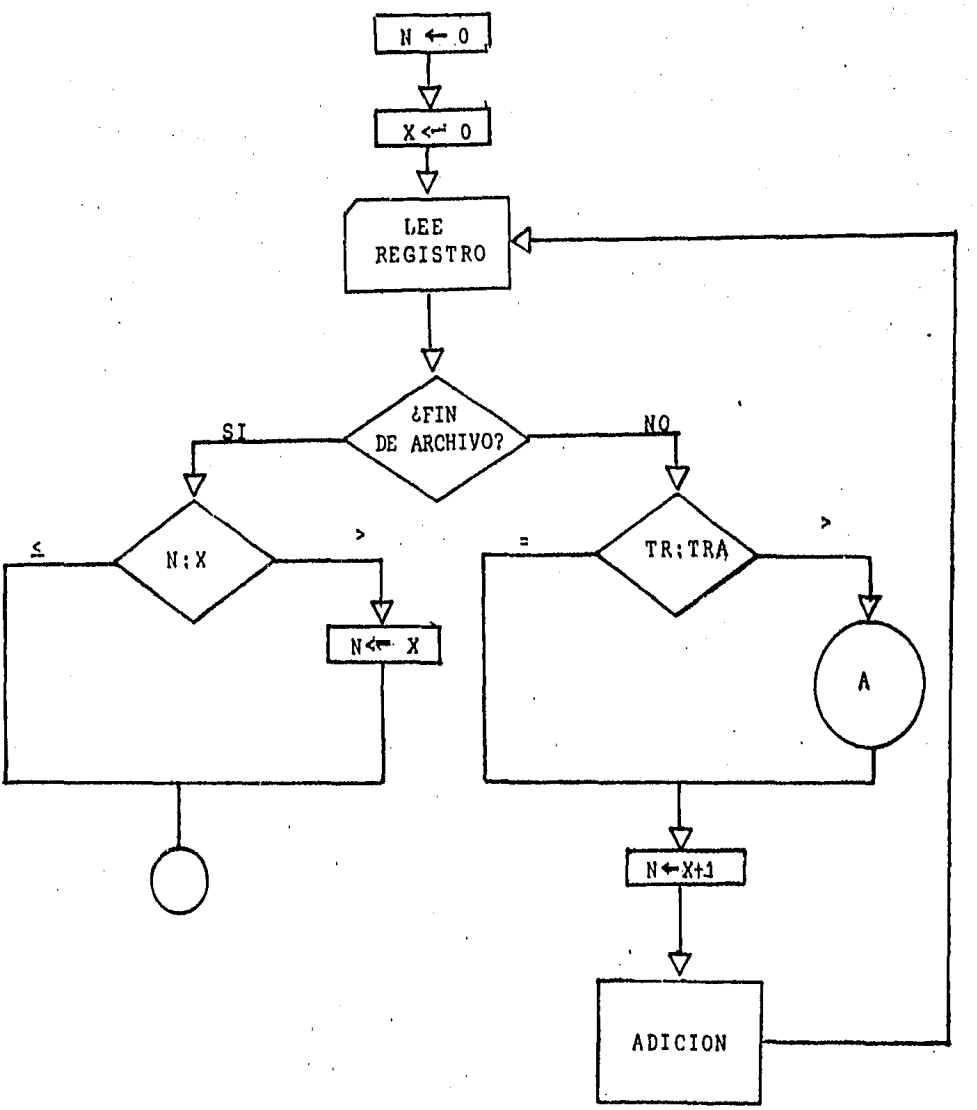
SUBRUTINA PRVAL

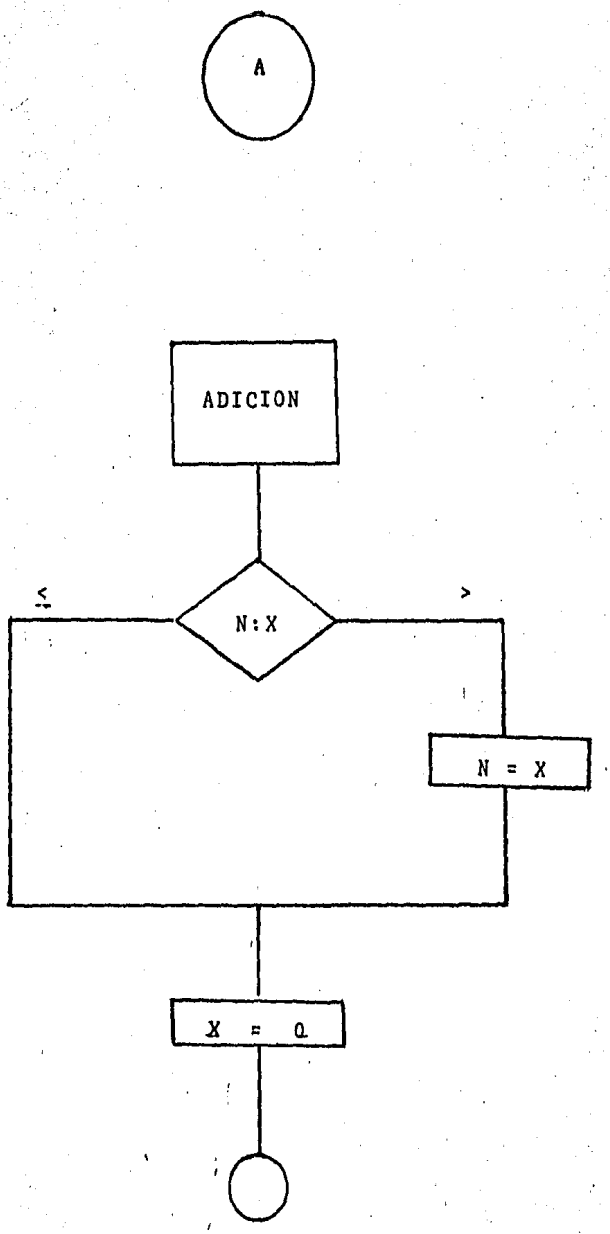
ARGUMENTOS (Pasados en COMMON)

I) Tipo de registro que se encuentra procesando (TR)

I) Tipo de registro anterior (TRA)

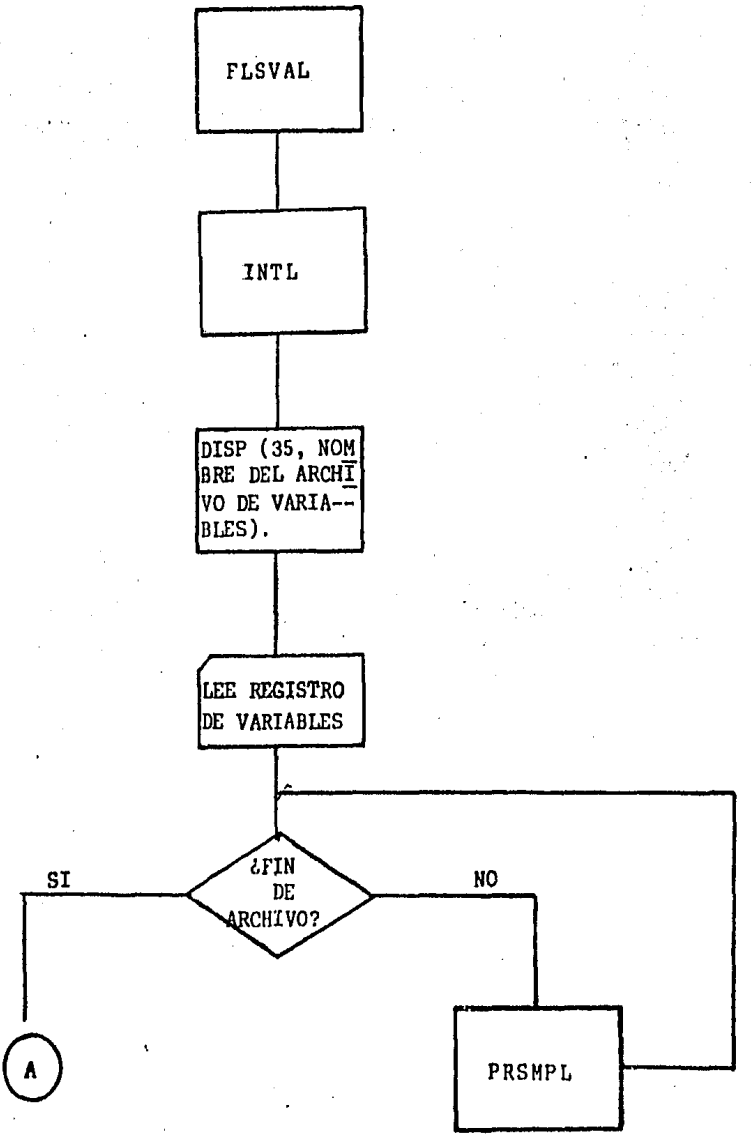
I-O)Máximo número de validaciones que suceden en algún tipo de registro (n).

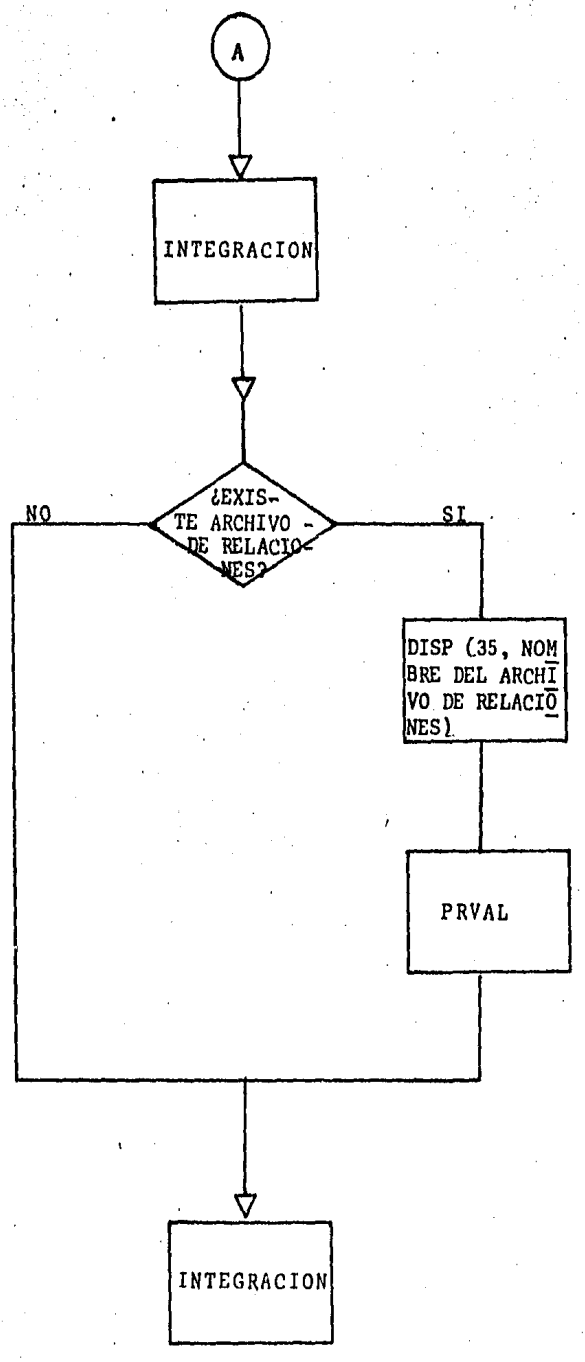



A
ADICION
N:X
≤
>
N = X
X = 0

Esta subrutina genera todas las instrucciones relacionadas con la validación del archivo de datos y deja en una variable a la rutina principal el número máximo de validaciones que ocurren en un tipo de registro para uso posterior en ciertos procesos de integración. Se efectuan aquí dos tipos de adiciones, una tiene lugar cada vez que hay cambio de tipo de registro y en ella se genera el nombre del párrafo donde se efectuarán las validaciones del mismo, la otra se genera para todas las relaciones del archivo y consiste en construir las instrucciones propias para la validación, estas son: una instrucción que identifica el número de relación en el registro que se esta validando y una instrucción IF en donde se hace la prueba de validación y se invoca a un párrafo común en caso que la prueba sea verdadera.

RUTINA PRINCIPAL




A
INTEGRACION
¿EXIS-
TE ARCHIVO -
DE RELACIO-
NES?
NO
SI
DISP (35, NOM
BRE DEL ARCHI
VO DE RELACIO
NES)
PRVAL
INTEGRACION

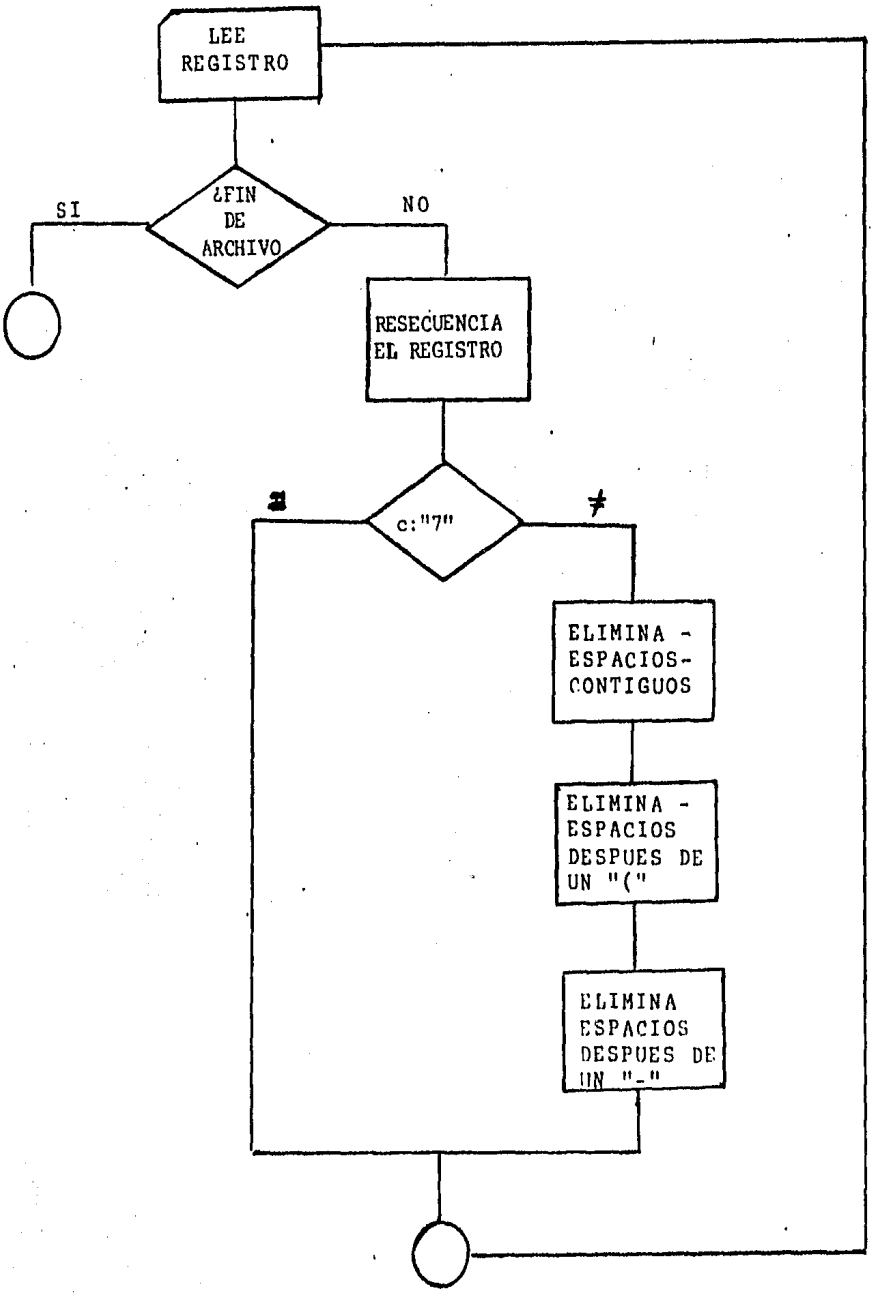
La rutina principal del programa puede dividirse en cuatro partes, la primera efectua operaciones de iniciación de valores a través de dos subrutinas, FLSVAL e INTL, posteriormente abre el archivo que contiene la información de las variables, lo procesa en PRSMPL y continua haciendo una integración que es independiente del proceso de validación, después, si existe un archivo de relaciones se procesa en PRVAL y finalmente se realiza una segunda integración cuyo funcionamiento depende en algunas partes de la existencia del archivo de relaciones. Los datos que se utilizan en los procesos de integración se encuentran en los archivos que proporciona el usuario y los principales son:

Pluralidad de tipos de registro
Existencia de llaves en los registros
Existencia de relaciones a validar.

En base a las combinaciones de estos datos los procesos de integración suprimirán, integrarán y completarán instrucciones del archivo esquema.

EDICION DEL PROGRAMA COBOL

En este diagrama, la variable "c" indica el primer caracter del registro.



Este programa constituye la última fase del paquete y su objetivo consiste en modificar algunas proposiciones del programa generado para que pueda ser compilado. Las funciones que realiza el programa son las siguientes:

Genera una nueva secuencia para cada instrucción con incremento de 10 en 10

Reduce una cadena de varios espacios a un solo espacio

Borra los espacios a la derecha de un "(" o de un "-".

Los registros que empiezan con el caracter "7" son aquellos que se formaron con las relaciones proporcionadas por el usuario por lo cual no se modifican.

INSTRUCCIONES COMMON

COMMON /BLK/ SERIAL, FILLER(66), IP

Contiene el registro que proviene del archivo esquema o que se grabará en el programa resultante.

MNEMOTECNICO	TIPO	DESCRIPCION
SERIAL	REAL	Contiene el número de secuencia del registro COBOL.
FILLER	BYTE	Arreglo que contiene la instrucción COBOL.
IP	INTEGER	Posición inicial a partir de la cual se inserta un argumento en caso necesario.

NOTA: La primera posición del arreglo SERIAL contiene un valor que puede ser espacio, * o @ y que se utiliza por el programa para tomar una acción especial. Cuando una instrucción va a ser grabada esta variable toma el valor espacio.

COMMON /DATBLK/ FMTS, LENGTH, TFMTS(9), VFMTS(9), FLNAME(20)

Contiene variables proporcionadas por el usuario sobre las características generales de su archivo.

MNEMOTECNICO	TIPO	DESCRIPCION
FMTS	INTEGER	Número de tipos de registro que tiene el archivo de datos.
LENGTH	REAL	Longitud del registro.
TFMTS	INTEGER	Arreglo que indica los tipos de registro que deben existir para cualquier unidad de información.
VFMTS	INTEGER	Arreglo que indica los tipos de registro que contienen relaciones a validar.

FLNAME	BYTE	Arreglo que contiene el nombre del archivo.

COMMON / VARBLK/ RECFMT, FLDNM(16), RCTYPE, STATUS, FMTID, KEYID, FLDIP, FLDLEN, FLDDEC, CNTMRK.

Contiene datos sobre las variables a filtrar.

MNEMOTECNICO	TIPO	DESCRIPCION
RECFMT	INTEGER	Número de tipo de registro en el que se encuentra la variable.
FLDNM	BYTE	Nombre de la variable.
RCTYPE	LOGICAL	Tipo de la variable.
STATUS	LOGICAL	Variable que indica si el dato en cuestión debe tomar un valor diferente a espacios.
FMTID	LOGICAL	Variable que indica si el dato en cuestión es identificador de tipo de registro.
KEYID	LOGICAL	Variable que indica si el dato es cuestión forma parte de la llave del registro.
FLDIP	INTEGER	Posición inicial de la variable.
FLDLEN	INTEGER	Longitud de la variable.
FLDDEC	INTEGER	Número de decimales de la variable si esta es numérica.
CNTMRK	INTEGER	Variable que indica si el dato en cuestión tiene asociados un conJunto de valores permitidos.

COMMON /VALBLK/ MAXVAL/

MNEMOTECNICO	TIPO	DESCRIPCION
MAXVAL	INTEGER	Máximo número de validaciones que se encuentraron para algún tipo de registro.

COMMON /SEQBLK/ PSQ, DDSQ, VSQ(9), ITVSQ, WSQ, VALSEQ(9).

Contiene números de secuencia, de acuerdo al tipo de instrucción que se genera se escoge uno en particular.

MNEMOTECNICO	TIPO
PSQ	REAL
DDSQ	REAL
VSQ	REAL
ITVSQ	REAL
WSQ	REAL
VALSEQ	REAL

COMMON /CRBLK/ CMNREC(80)

MNEMOTECNICO	TIPO	DESCRIPCION
CMNREC	BYTE	Arreglo que simula el registro COBOL, desde este arreglo se grabán las instrucciones generadas para el programa resultante.

COMMON /TABBLK/ VF(9)

MNEMOTECNICO	TIPO	DESCRIPCION
VF	INTEGER	Arreglo en donde cada elemento contiene el número de variables que trae el correspondiente tipo de registro.

Las siguientes instrucciones COMMON contienen solamente un arreglo en su lista que esta destinado a recibir una instrucción COBOL particular o un dato utilizado por el programa resultante.

El arreglo es de tipo BYTE y su dimensión se muestra en el COMMON.

COMMON /IFBLK/ IFREC(62)

Imágen de la instrucción IF.

COMMON /PFMBLK/ PFMREC(30)

Imágen de la instrucción PERFORM.

COMMON /HDRBLK/ FMTHDR(14)

Arreglo que contiene el nombre asociado a un tipo de registro.

COMMON /CHFBLK/ CHKFMT(14)

Arreglo que contiene el nombre de un párrafo donde se efectua el filtrado de un tipo de registro.

COMMON /FLRBLK/ FLRREC(22)

Imágen de la clausula FILLER.

COMMON /FLDBLK/ FLDREC(38)

Arreglo que contiene la descripción de una variable.

COMMON /CNDBLK/ CNDREC(62)

Arreglo que contiene los valores asociados a una variable.

COMMON /MVEBLK/ MVEREC(52)

Imágen de la instrucción MOVE.

CAPITULO VII

CONCLUSIONES Y OBSERVACIONES

El paquete para generar programas de filtrado y validación de archivos tiene ciertas limitaciones que pueden ser subsanadas:

Dado que el programa generador esta escrito en FORTRAN y el programa resultante en COBOL se hizo necesario un programa de edición para suprimir espacios y resecuenciar que permitiera al programa resultante ser compilado correctamente. Además, es común que varios equipos utilizen un solo lenguaje, y en caso de que este sea COBOL el programa resultante no puede obtenerse directamente ahí. Actualmente, estas limitantes se pueden resolver realizando todo el trabajo en el equipo donde se programo y posteriormente trasladando los programas generados al equipo donde se van a procesar. Una mejor solución es la de tener el paquete en un solo lenguaje: COBOL para las instituciones administrativas y FORTRAN, PASCAL o algún otro para las científicas.

Los datos a validar deben ser tipo despliegue, esto es, no se permiten datos empacados. Dado que la forma de representar los datos empacados es particular del COBOL que se este utilizando en un equipo determinado, se puede modificar el paquete para ese equipo de tal forma que pueda aceptar este tipo de datos.

No existe un diálogo con el usuario para que este pueda

encuentran sus semejantes. De forma en que rápidamente se encuentran en base de tablas. La implantación de esta forma se encuentran los datos puede realizarse rápidamente una. subrutina en el programa generador.

Por otro lado las ventajas son:

Dado que es un generador se produce bien código objeto bien como la calidad del compilador que se use y rápidamente este es superior al código de implementación que puede hacerse.

El programa resultante es estructurado lo cual le permite aceptar con facilidad modificaciones o nuevas rutinas a nivel de usuario quien puede añadir el código generado para añadir validaciones particulares.

El programa resultante contiene instrucciones completamente estándares de COBOL, por lo que presenta facilidades para su traslado de un equipo a otro.

El coste del paquete es simple por lo que puede ser utilizado por usuarios que no estén relacionados con Computación.

Considerando las limitaciones y las ventajas que ofrece el paquete, se concluye que el trabajo cumple satisfactoriamente el objetivo para el cual fue realizado, teniendo en cuenta que puede mejorarse y ampliarse.

ANEXO A

PROGRAMA GENERADOR

```
INTEGER RECFMT , FLDIP , FLDLEN , FLDDEC , I , IF ,
1        FMTS , SW  , VF(9) , TFMTS(9) , NRV , MAXVAL , VFMTS(9)
1        , COLUMN , WF , V , W , KV , KEYLEN , CNTMRK , X
REAL LENGTH , PSQ , DDSQ , VSQ(9) , ITVSQ , WSQ , Y , SERIAL ,
1     VALSEQ(9)
LOGICAL RCTYPE , STATUS , FMTID , KEYID , WKEYID
BYTE FLDNM(16) , FLRREC(22) , CNDREC(62) , FILLER(66) , OPCODE ,
1     CL(6) , IFREC (62) , LBLREC(22) , CHKFMT(14) , MVEREC(52)
1     , PFMREC(30) , CMNREC(80) , FMTHDR(14) , FLDREC(38) ,
1     FLNAME(20) , PGMFLE(35) , RDFREC(60)
COMMON /VARBLK/ RECFMT , FLDNM , RCTYPE , STATUS , FMTID ,
1                KEYID , FLDIP , FLDLEN , FLDDEC , CNTMRK
COMMON /DATBLK/ FMTS , LENGTH , TFMTS , VFMTS , FLNAME
COMMON /SEQBLK/ PSQ , DDSQ , VSQ , ITVSQ , WSQ , VALSEQ
COMMON /CLBLK/ CL
COMMON /IFBLK/ IFREC
COMMON /LBLBLK/ LBLREC
COMMON /PFMBLK/ PFMREC
COMMON /CRBLK/ CMNREC
COMMON /HDRBLK/ FMTHDR
COMMON /CHFBLK/ CHKFMT
COMMON /LBLK/ SW , NRV
COMMON /WBLK/ COLUMN , WF , V , W , KV , KEYLEN , WKEYID
COMMON /FLRBLK/ FLRREC
COMMON /FLDBLK/ FLDREC
COMMON /BLK/ SERIAL , FILLER , IP
COMMON /TABBLK/ VF
COMMON /CNDBLK/ CNDREC
COMMON /MVEBLK/ MVEREC
COMMON /VALBLK/ MAXVAL
COMMON /RDFBLK/ RDFREC
EQUIVALENCE( FILLER , OPCODE )
DATA X , NRV , MAXVAL / 3 * 0 /
FORMAT( I1 , 16A1 , 4L1 , 2I4 , I2 , I1 )
FORMAT(I4)
FORMAT(F7.0)
FORMAT(I1)
FORMAT(I3)
FORMAT(F4.0)
FORMAT( 1H , 80A1 )
CALL FLSVAL
CALL INTL
CALL DISP( 20 , 'ARCHIVO DE VARIABLES' , PGMFLE )
OPEN( UNIT = 3 , NAME = PGMFLE , TYPE='OLD' )
READ(3,18) RECFMT , FLDNM , RCTYPE , STATUS , FMTID ,
1           KEYID , FLDIP , FLDLEN , FLDDEC , CNTMRK
DO 75 SW = 1 , 2
  CALL PRSMPL
IF ( .NOT. WKEYID .OR. FMTS .EQ. 1 ) GO TO 99
ENCODE( 1 , 39 , FILLER(IP) ) FMTS
CALL CPYFL
GO TO 53
CALL PSSRCS(0)
CALL CPYFL
IF ( WKEYID .OR. FMTS .GT. 1 ) CALL CPYFL
IF ( .NOT. WKEYID .AND. FMTS .EQ. 1 ) CALL PSSRCS(0)
CALL CPYFL
DECODE( 4 , 16 , LENGTH ) Y
FLDIP = Y + 1
IF ( FLDIP .NE. COLUMN ) CALL GENFLR
IF ( KEYLEN .EQ. 0 ) GO TO 32
CALL MOVE( 22 , '01  WKEYD PIC X(    ).' , CMNREC(8) )
WSQ = WSQ +  1
ENCODE( 7 , 66 , CMNREC ) WSQ
```

```
      ENCODE( 4 , 43 , CMNREC(24) ) RETLEN
      WRITE( 4 , 9 ) CMNREC
      CALL CLNREC( CMNREC , 8 , 72 )
      W = Y / 100 + 1
      IF ( Y - 100 * ( W - 1 ) .EQ. 0.0 ) W = W - 1
      ENCODE( 3 , 83 , FILLER(IP) ) W
      CALL CPYFL
      IF ( NRV .GT. 0 ) CALL CPYFL
      IF ( NRV .EQ. 0 ) CALL PSSRCS(0)
      IF ( FMTS .GT. 1 ) CALL CPYFL
      IF ( FMTS .EQ. 1 ) CALL PSSRCS(0)
      CALL MOVE( 8 , 'DFT ( ),' , MVEREC(26) )
      CALL MOVE( 52 , MVEREC , CMNREC(12) )
      VF(WF)=0
      DO 50 I = 1 , FMTS
        ENCODE( 4 , 43 , CMNREC(17) ) VF(I)
        ENCODE( 1 , 39 , CMNREC(42) ) I
        ITVSQ = ITVSQ + 1
        ENCODE( 7 , 66 , CMNREC ) ITVSQ
        WRITE( 4 , 9 ) CMNREC
        IF ( VF(I) .GT. X ) X = VF(I)
        CALL CPYFL
        ENCODE( 4 , 43 , FILLER(IP) ) VF(I)
      CALL CLNREC( MVEREC , 6 , 21 )
      CALL CLNREC( MVEREC , 26 , 52 )
      CALL CLNREC( CMNREC , 12 , 72 )
      CALL CPYRCS(3)
      IF ( OPCODE .NE. '@' ) CALL PSSRCS(0)
      IF ( OPCODE .EQ. '@' ) OPCODE = ' '
      CALL CPYFL
      ENCODE( 4 , 43 , FILLER(IP) ) X
      CALL CPYFL
      ENCODE( 4 , 43 , FILLER(IP) ) X
      CALL CPYFL
      ENCODE( 1 , 39 , FILLER(IP) ) FMTS
      CALL CPYFL
      IF ( NRV .EQ. 0 ) GO TO 29
      ENCODE( 1 , 39 , FILLER(IP) ) FMTS
      CALL CPYFL
      GO TO 70
      CALL PSSRCS(0)
      IF ( .NOT. WKEYID .OR. FMTS .EQ. 1 ) GO TO 12
      CALL CPYFL
      ENCODE( 1 , 39 , FILLER(IP) ) FMTS
      CALL CPYFL
      GO TO 85
      CALL PSSRCS(0)
      CALL CPYFL
      ENCODE( 3 , 83 , FILLER(IP) ) W
      CALL CPYFL
      ENCODE( 3 , 83 , FILLER(IP) ) W
      CALL CPYFL
      I = 100 * W
      ENCODE( 4 , 43 , FILLER(IP) ) I
      CALL CPYFL
      IF ( NRV .GT. 0 ) CALL PRVAL
      IF ( MAXVAL .GT. 0 ) GO TO 89
      CALL PSSRCS(0)
      GO TO 2
      CALL CPYFL
      ENCODE( 3 , 83 , FILLER(IP) ) MAXVAL
      CALL CPYFL
      ENCODE( 3 , 83 , FILLER(IP) ) MAXVAL
      CALL CPYFL
      CALL CPYFL
      CALL MOVE( 29 , FLNAME , FILLER(IP) )
```

```
      CALL CPYFL
      IF ( NRV .GT. 0 ) GO TO 7°
      CALL PSSRCS(0)
      GO TO 72
79    CALL CPYFL
      CALL MOVE( 20 , FUNAME , FILLER(IP) )
      CALL CPYFL
72    IF ( WKEYID ) CALL CPYFL
      IF ( .NOT. WKEYID ) CALL PSSRCS(0)
      IF ( WKEYID .AND. FMTS .GT. 1 ) CALL CPYFL
      IF ( .NOT. WKEYID .OR. FMTS .EQ. 1 ) CALL PSSRCS(0)
      IF ( FMTS .GT. 1 ) CALL CPYFL
      IF ( FMTS .EQ. 1 ) CALL PSSRCS(0)
      CALL CPYFL
      IF ( NRV .GT. 0 ) CALL MOVE ( 6 , 'REPVAL' , FILLER(IP) )
      CALL CPYFL
      IF ( WKEYID .AND. FMTS .GT. 1 ) CALL CPYFL
      IF ( .NOT. WKEYID .OR. FMTS .EQ. 1 ) CALL PSSRCS(0)
      CALL CPYFL
      IF ( NRV .GT. 0 ) CALL CPYFL
      IF ( NRV .EQ. 0 ) CALL PSSRCS(0)
      IF ( NRV .GT. 0 ) CALL MOVE ( 6 , 'REPVAL' , FILLER(IP) )
      CALL CPYFL
      IF ( WKEYID .AND. FMTS .GT. 1 ) CALL CPYFL
      IF ( .NOT. WKEYID .OR. FMTS .EQ. 1 ) CALL PSSRCS(0)
      IF ( WKEYID .AND. FMTS .EQ. 1 ) CALL CPYFL
      IF ( .NOT. WKEYID .OR. FMTS .GT. 1 ) CALL PSSRCS(0)
      IF ( FMTS .EQ. 1 ) GO TO 3
      CALL CPYFL
      IF ( WKEYID ) GO TO 76
      FILLER(IP) = ','
      CALL CPYRCS(1)
      CALL PSSRCS(0)
      GO TO 65
76    CALL CPYFL
      GO TO 65
3     CALL PSSRCS(0)
65    IF ( WKEYID .OR. FMTS .GT. 1 ) GO TO 10
      CALL PSSRCS(0)
      CALL CPYFL
      GO TO 73
10    CALL CPYFL
      CALL PSSRCS(0)
73    CALL CPYFL
      IF ( FMTS .EQ. 1 ) GO TO 98
      DO 15 I = 1 , FMTS
        CALL CPYRCS(2)
        ENCODE( 4 , 43 , FILLER(IP) ) VF (I)
        CALL CPYRCS(2)
15      CONTINUE
      IF ( OPCODE .NE. '@' ) CALL PSSRCS(0)
      IF ( OPCODE .EQ. '@' ) OPCODE = ' '
      CALL PSSRCS(2)
      GO TO 63
'8    CALL PSSRCS(0)
      CALL CPYFL
      ENCODE( 4 , 43 , FILLER(IP) ) X
:3    CALL CPYFL
      IF ( NRV .GT. 0 ) GO TO 33
      FILLER(IP) = ','
      CALL CPYRCS(1)
      CALL PSSRCS(0)
      GO TO 91
 3    CALL CPYFL
      IF ( FMTS .GT. 1 ) CALL MOVE( 2 , 'CL' , FILLER(IP) )
      IF ( FMTS .EQ. 1 ) CALL MOVE( 2 , ' 1' , FILLER(IP) )
```

```
CALL CPYFL
IF ( FMTS .GT. 1 ) CALL MOVE( 11 , 'VAL-FORMATS' , FILLER(IP) )
IF ( FMTS .EQ. 1 ) CALL MOVE( 11 , 'CHECK-VAL-1' , FILLER(IP) )
CALL CPYFL
IF ( WKEYID .AND. FMTS .GT. 1 ) CALL CPYFL
IF ( .NOT. WKEYID .OR. FMTS .EQ. 1 ) CALL PSSRCS(0)
IF ( WKEYID ) CALL CPYFL
IF ( .NOT. WKEYID ) CALL PSSRCS(0)
IF ( WKEYID .OR. FMTS .GT. 1 ) CALL CPYFL
IF ( .NOT. WKEYID .AND. FMTS .EQ. 1 ) CALL PSSRCS(0)
CALL CPYFL
IF ( FMTS .EQ. 1 ) CALL MOVE( 2 , ' 1' , FILLER(IP) )
IF ( FMTS .GT. 1 ) CALL MOVE( 2 , 'CL' , FILLER(IP) )
CALL CPYFL
IF ( FMTS .GT. 1 ) CALL CPYFL
IF ( FMTS .EQ. 1 ) CALL PSSRCS(0)
CALL CPYFL
IF ( NRV .GT. 0 .AND. FMTS .GT. 1 ) GO TO 46
CALL PSSRCS(0)
GO TO 69
CALL CPYRCS(1)
DO 92 I = 1 , FMTS
 IF ( VFMTS(I) .EQ. 1 ) CALL CPYRCS(2)
  IF ( VFMTS(I) .EQ. 0 ) CALL PSSRCS(2)
  CONTINUE
IF ( OPCODE .NE. '@' ) CALL PSSRCS(0)
IF ( OPCODE .EQ. '@' ) OPCODE = ' '
IF ( NRV .GT. 0 ) CALL CPYFL
IF ( NRV .EQ. 0 ) CALL PSSRCS(0)
CALL CPYFL
STOP
END
```

```
      SUBROUTINE CLNREC( STRING , I , J )
      INTEGER I , J , K
      BYTE STRING(J)
      DO 27 K = I , J
        STRING(K) = ' '
      RETURN
      END
      SUBROUTINE DISP( N , X , Y )
      BYTE X(35) , Y(35)
      INTEGER I , N
      FORMAT( 1H , 35A1 )
      FORMAT( 35A1 )
      TYPE 1 , ( X(I) , I = 1 , N )
      ACCEPT 3 , Y
      RETURN
      END
      SUBROUTINE CPYFL
      INTEGER IP , I
      REAL SERIAL
      BYTE OPCODE , FILLER(66)
      COMMON /BLK/ SERIAL , FILLER , IP
      EQUIVALENCE( FILLER , OPCODE )
      FORMAT( 1H , F7.0 , T8 , 66A1 , 8X )
      FORMAT(I4)
      FORMAT( F6.0 , 66A1 , 4X , I2 )
      IF ( OPCODE .NE. ' ' ) GO TO 69
      WRITE(4,22) SERIAL , FILLER
      READ( 2 , 55 ) SERIAL , FILLER , IP
      GO TO 91
      OPCODE = ' '
      IP = IP - 6
      RETURN
      END
      SUBROUTINE CPYRCS(N)
      INTEGER N , K , IP
      REAL SERIAL
      BYTE FILLER(66)
      COMMON /BLK/ SERIAL , FILLER , IP
      FORMAT( F6.0 , 66A1 , 4X , I2 )
      FORMAT( 1H , F7.0 , T8 , 66A1 , 8X )
      DO 16 K = 1 , N
        WRITE( 4 , 22 ) SERIAL , FILLER
        READ( 2 , 55 ) SERIAL , FILLER , IP
        CONTINUE
      IP = IP - 6
      RETURN
      END
```

```
SUBROUTINE FLSVAL
INTEGER WF , V , W , KEYLEN , VF(9)
REAL PSQ , DDSQ , VSQ(9) , ITVSQ , WSQ , VALSEQ(9)
LOGICAL WKEYID
BYTE IFREC(62) , FMTHDR(14) , PFMREC(30) , CL(6) , CHKFMT(14)
1      , CMNREC(80) , MVEREC(52) , FLRREC(22) , FLDREC(38) ,
1      CNDREC(62) , RDFREC(60)
COMMON /IFBLK/ IFREC /PFMBLK/ PFMREC /HDRBLK/ FMTHDR
COMMON /CLBLK/ CL /CHFBLK/ CHKFMT /CRBLK/ CMNREC
COMMON /MVEBLK/ MVEREC /FLRBLK/ FLRREC /FLDBLK/ FLDREC
COMMON /CNDBLK/ CNDREC /TABBLK/ VF /RDFBLK/ RDFREC
COMMON /SEQBLK/ PSQ , DDSQ , VSQ , ITVSQ , WSQ , VALSEQ
COMMON /WBLK/ COLUMN , WF , V , W , KV , KEYLEN , WKEYID
DATA DDSQ / 100000.0 /
DATA WSQ / 200000.0 /
DATA PSQ / 250000.0 /
DATA VSQ / 410000.0 , 420000.0 , 430000.0 , 440000.0 ,
1    450000.0 , 460000.0 , 470000.0 , 480000.0 , 490000.0 /
DATA VALSEQ / 710000.0 , 720000.0 , 730000.0 , 740000.0 ,
1     750000.0 , 760000.0 , 770000.0 , 780000.0 , 790000.0 /
DATA ITVSQ / 900000.0 /
DATA WF / 0 /
DATA KV / 0 /
DATA VF / 9 * 0 /
DATA RDFREC / 60 * ' ' /
DATA WKEYID / .FALSE. /
DATA IFREC / 62 * ' ' /
DATA PFMREC / 30 * ' ' /
DATA CMNREC / 80 * ' ' /
DATA MVEREC / 52 * ' ' /
CALL MOVE( 2 , 'IF' , IFREC )
CALL MOVE( 8 , 'PERFORM ' , PFMREC )
CALL MOVE( 11 , '01  RECORD-' , FMTHDR )
FMTHDR(13) = ','
CALL MOVE( 5 , 'CL = ' , CL )
CALL MOVE( 13 , 'CHECK-FORMAT-' , CHKFMT )
CALL MOVE( 5 , 'MOVE ' , MVEREC )
CALL MOVE( 2 , 'TO' , MVEREC(23) )
CALL MOVE( 16 , '02 FILLER PIC X(' , FLRREC)
CALL MOVE( 2 , ').' , FLRREC(21) )
CALL MOVE( 3 , '02 ' , FLDREC )
CALL MOVE( 19 , ' PIC  (    )       ', FLDREC(20) )
CALL MOVE( 15 ,'88 V~    VALUE ' , CNDREC )
CNDREC(61) = ','
CALL MOVE( 4 , '02 R' , RDFREC )
CALL MOVE( 10 , 'REDEFINES ' , RDFREC(22) )
CALL MOVE( 12 , 'PIC X(    ).' , RDFREC(49) )
RETURN
END
```

```
      SUBROUTINE PSSRCS( N )
      INTEGER I
      REAL SERIAL
      BYTE FILLER(66) , OPCODE
      COMMON /BLK/ SERIAL , FILLER
      EQUIVALENCE ( FILLER , OPCODE )
5     FORMAT( F6.0 , 66A1 )
      IF ( N .EQ. 0 ) GO TO 17
      DO 38 I = 1 , N
        READ( 2 , 55 ) SERIAL , FILLER
8       CONTINUE
      RETURN
7     DO 80 I = 1 , 2
        I = 0
        READ( 2 , 55 ) SERIAL , FILLER
        IF ( OPCODE .EQ. '@' ) I = 2
0       CONTINUE
      OPCODE = ' '
      RETURN
      END
      SUBROUTINE MOVE( N , SOURCE , OBJECT )
      INTEGER I , N
      BYTE SOURCE(N) , OBJECT(N)
      DO 28 I = 1 , N
8       OBJECT(I) = SOURCE(I)
      RETURN
      END
      SUBROUTINE GENFLR
      INTEGER FLDIP , FLRFLD , COLUMN
      REAL PSQ , DDSQ
      BYTE FLRREC(22) , CMNREC(80) , FILLER(34)
      COMMON /VARBLK/ FILLER , FLDIP
      COMMON /CRBLK/ CMNREC
      COMMON /FLRBLK/ FLRREC
      COMMON /SEQBLK/ PSQ , DDSQ
      COMMON /WRLK/ COLUMN
      FORMAT( 1H , 80A1 )
3     FORMAT(I4)
)     FORMAT(F7.0)
      FLRFLD = FLDIP - COLUMN
      ENCODE( 4 , 43 , FLRREC(17) ) FLRFLD
      CALL MOVE( 22 , FLRREC , CMNREC(12) )
      DDSQ = DDSQ + 1
      ENCODE( 7 , 66 , CMNREC ) DDSQ
      WRITE( 4, 9 ) CMNREC
      CALL CLNREC( CMNREC , 12 , 72 )
      RETURN
      END
```

```
      SUBROUTINE INTL
      INTEGER FMTS , TFMTS(9) , IP , VFMTS(9) , NRV , DUMMY
      REAL LENGTH , BLKL , SERIAL
      BYTE FLNAME(20) , CORE(10) , FILLER(66) , PGMFLE(35)
      COMMON /BLK/ SERIAL , FILLER , IP
      COMMON /DATBLK/ FMTS , LENGTH , TFMTS , VFMTS , FLNAME
      COMMON /LBLK/ DUMMY , NRV
6     FORMAT( 30A1 , 2A4 , 19I1 )
5     FORMAT( F6.0 , 66A1 , 4X , I2 )
9     FORMAT( I1 )
      CALL DISP( 17 , 'DATOS DEL ARCHIVO' , PGMFLE )
      OPEN( UNIT = 1 , NAME = PGMFLE , TYPE = 'OLD' )
      OPEN( UNIT = 2 , NAME = 'SCHEMA.DAT' , TYPE = 'OLD' )
      READ( 1 , 76 ) FLNAME , CORE , LENGTH , BLKL , FMTS , TFMTS
     1               , VFMTS
      DO 86 I = 1 , FMTS
        IF ( VFMTS(I) .EQ. 1 ) NRV = NRV + 1
6       CONTINUE
      READ( 2 , 55 ) SERIAL , FILLER , IP
      CALL CPYFL
      CALL MOVE( 10 , CORE , FILLER(IP) )
      CALL CPYFL
      IF ( NRV .GT. 0 ) CALL CPYFL
      IF ( NRV .EQ. 0 ) CALL PSSRCS(0)
      CALL CPYFL
      IF ( NRV .GT. 0 ) CALL CPYFL
      IF ( NRV .EQ. 0 ) CALL PSSRCS(0)
      CALL CPYFL
      CALL MOVE( 4 , LENGTH , FILLER(IP) )
      CALL CPYFL
      CALL MOVE( 4 , BLKL , FILLER(IP) )
      CALL CPYFL
      RETURN
      END
      SUBROUTINE CPYID
      INTEGER FILLER
      REAL PSQ
      BYTE MVEREC(52) , CMNREC(80) , FLDNM(16)
      COMMON /MVEBLK/ MVEREC
      COMMON /CRBLK/ CMNREC
      COMMON /SEQBLK/ PSQ
      COMMON /VARBLK/ FILLER , FLDNM
5     FORMAT(F7.0)
      FORMAT( 1H , 80A1 )
      CALL MOVE( 16 , FLDNM , MVEREC(6) )
      CALL MOVE( 3 , 'CL,' , MVEREC(26) )
      PSQ = PSQ + 1
      ENCODE( 7 , 66 , CMNREC ) PSQ
      CALL MOVE( 28 , MVEREC , CMNREC(12) )
      WRITE( 4 , 9 ) CMNREC
      CALL CLNREC( CMNREC , 12 , 72 )
      CALL CLNREC( MVEREC , 6 , 21 )
      CALL CLNREC( MVEREC , 26 , 52 )
      RETURN
      END
```

```
      SUBROUTINE CHFMT
      INTEGER RECFMT , FLDIP , DUMMY , WFLDIP ,
     1        WF , COLUMN , V , VF(9)
      REAL PSQ , DDSQ , VSQ(9) , Y , FILLER(8) , LENGTH
      BYTE FMTHDR(14) , CMNREC(80)
      COMMON /DATBLK/ DUMMY , LENGTH
      COMMON /SEQBLK/ PSQ , DDSQ , VSQ
      COMMON /CRBLK/ CMNREC
      COMMON /HDRBLK/ FMTHDR
      COMMON /WBLK/ COLUMN , WF , V , W
      COMMON /TABBLK/ VF
      COMMON /VARBLK/ RECFMT , FILLER , FLDIP
      FORMAT(I1)
      FORMAT(F7.0)
      FORMAT(F4.0)
      FORMAT( 1H , 80A1 )
      IF ( WF .EQ. 0 ) GO TO 33
      VF (WF) = V
      WFLDIP = FLDIP
      DECODE( 4 , 16 , LENGTH ) Y
      FLDIP = Y + 1
      IF ( FLDIP .NE. COLUMN ) CALL GENFLR
      FLDIP = WFLDIP
      ENCODE( 1 , 39 , FMTHDR(12) ) RECFMT
      DDSQ = DDSQ + 1
      ENCODE( 7 , 66 , CMNREC ) DDSQ
      CALL MOVE( 13 , FMTHDR , CMNREC(8) )
      WRITE( 4 , 9 ) CMNREC
      COLUMN = 1
      V = 0
      W = 0
      CALL MOVE( 12 , 'CHECK-LABEL-' , CMNREC(8) )
      ENCODE( 1 , 39 , CMNREC(20) ) RECFMT
      CMNREC(21) = '.'
      WF = RECFMT
      VSQ(RECFMT) = VSQ(RECFMT) + 1
      ENCODE( 7 , 66 , CMNREC ) VSQ(RECFMT)
      WRITE( 4 , 9 ) CMNREC
      CALL CLNREC( CMNREC , 8 , 11 )
      RETURN
      END
      SUBROUTINE NUMPIC
      INTEGER RECFMT , FLDIP , FLDLEN , FLDDEC , N
      LOGICAL RCTYPE , STATUS , FMTID , KEYID
      BYTE FLDNM(16) ,  FLDREC(38)
      COMMON /NBLK/ N
      COMMON /FLDBLK/ FLDREC
      COMMON /VARBLK/ RECFMT , FLDNM , RCTYPE , STATUS , FMTID ,
     1                KEYID , FLDIP , FLDLEN , FLDDEC
      FORMAT(I4)
      FORMAT(I2)
      N = 38
      CALL MOVE( 3 , 'V9(' , FLDREC(32) )
      CALL MOVE( 2 , '),' , FLDREC(37) )
      ENCODE( 4 , 43 , FLDREC(27) ) FLDLEN - FLDDEC
      ENCODE( 2 , 80 , FLDREC(35) ) FLDDEC
      RETURN
      END
```

```
SUBROUTINE DEFSVL
INTEGER RECFMT , FLDIP , FLDLEN , W  , FF
REAL DUMMY(8) , SEQS(111) , ITVSQ , FILLER
BYTE MVEREC(52) , CMNREC(80)
COMMON /SEQBLK/ SEQS , ITVSQ
COMMON /WBLK/ FILLER , W
COMMON /CRBLK/ CMNREC
COMMON /MVEBLK/ MVEREC
COMMON /VARBLK/ RECFMT , DUMMY , FLDIP , FLDLEN
FORMAT(F7.0)
FORMAT(I4)
FORMAT(I1)
FORMAT( 1H , 80A1 )
ENCODE( 4 , 43 , MVEREC(6) ) FLDIP
CALL MOVE( 12 , "IF-  (    )." , MVEREC(26) )
ENCODE( 1 , 39 , MVEREC(29) ) RECFMT
ENCODE( 4 , 43 , MVEREC(32) ) W
CALL MOVE( 52 , MVEREC , CMNREC(12) )
ITVSQ = ITVSQ + 1
ENCODE( 7 , 66 , CMNREC ) ITVSQ
WRITE( 4 , 9 ) CMNREC
CMNREC(37) = 'F'
FF = FLDIP + FLDLEN - 1
ENCODE( 4 , 43 , CMNREC(17) ) FF
ITVSQ = ITVSQ + 1
ENCODE( 7 , 66 , CMNREC ) ITVSQ
WRITE( 4 , 9 ) CMNREC
CALL CLNREC( CMNREC , 12 , 72 )
CALL CLNREC( MVEREC , 6 , 21 )
CALL CLNREC( MVEREC , 26 , 52 )
RETURN
END
SUBROUTINE RDFTST
INTEGER RECFMT
REAL RFIL(2) , VSQ(9)
BYTE CMNREC(80) , FLDNM(16)
COMMON /CRBLK/ CMNREC
COMMON /SEQBLK/ RFIL , VSQ
COMMON /VARBLK/ RECFMT , FLDNM
FORMAT(F7.0)
FORMAT( 1H , 80A1 )
CALL MOVE( 4 , 'AND ' , CMNREC(15) )
CMNREC(19) = 'R'
CALL MOVE( 16 , FLDNM , CMNREC(20) )
CALL MOVE( 16 , 'NOT EQUAL SPACES' , CMNREC(37) )
VSQ(RECFMT) = VSQ(RECFMT) + 1
ENCODE( 7 , 66 , CMNREC ) VSQ(RECFMT)
WRITE( 4 , 9 ) CMNREC
CALL CLNREC( CMNREC , 12 , 72 )
RETURN
END
```

```
      SUBROUTINE EDTFLD
      INTEGER RECFMT , N , F(9) , CNTMRK
      REAL FILLER(2) , VSQ(9)
      LOGICAL RCTYPE
      BYTE MVEREC(52) , CMNREC(80) , FLDNM(16)
      COMMON /SEQBLK/ FILLER , VSQ
      COMMON /CRBLK/ CMNREC
      COMMON /MVEBLK/ MVEREC
      COMMON /VARBLK/ RECFMT , FLDNM , RCTYPE , F , CNTMRK
39    FORMAT(I1)
66    FORMAT(F7.0)
9     FORMAT( 1H , 80A1 )
      IF ( .NOT. RCTYPE .AND. CNTMRK .EQ. 0 ) N = 1
      IF ( RCTYPE ) N = 2
      IF ( CNTMRK .NE. 0 ) N = 3
      ENCODE( 1 , 39 , MVEREC(6) ) N
      MVEREC(31) = 'N'
      CALL MOVE( 31 , MVEREC , CMNREC(12) )
      VSQ(RECFMT) = VSQ(RECFMT) + 1
      ENCODE( 7 , 66 , CMNREC ) VSQ(RECFMT)
      WRITE( 4 , 9 ) CMNREC
      CALL CLNREC( CMNREC , 12 , 72 )
      CALL MOVE( 22 , 'PERFORM TYPE-OF-ERROR.' , CMNREC(12) )
      VSQ(RECFMT) = VSQ(RECFMT) + 1
      ENCODE( 7 , 66 , CMNREC ) VSQ(RECFMT)
      WRITE( 4 , 9 ) CMNREC
      CALL CLNREC( CMNREC , 12 , 72 )
      CALL CLNREC( MVEREC , 6 , 21 )
      CALL CLNREC( MVEREC , 26 , 52 )
      RETURN
      END
      SUBROUTINE GRFREC
      INTEGER INTFIL , FLDLEN , ARFIL(9)
      REAL RFIL , DDSQ
      BYTE RDFREC(60) , FLDNM(16) , CMNREC(80)
      COMMON /CRBLK/ CMNREC
      COMMON /RDFBLK/ RDFREC
      COMMON /SEQBLK/ RFIL , DDSQ
      COMMON /VARBLK/ INTFIL , FLDNM , ARFIL , FLDLEN
43    FORMAT(I4)
66    FORMAT(F7.0)
9     FORMAT( 1H , 80A1 )
      CALL MOVE( 16 , FLDNM , RDFREC(5) )
      CALL MOVE( 16 , FLDNM , RDFREC(32) )
      ENCODE( 4 , 43 , RDFREC(55) ) FLDLEN
      CALL MOVE( 60 , RDFREC , CMNREC(12) )
      DDSQ = DDSQ + 1
      ENCODE( 7 , 66 , CMNREC ) DDSQ
      WRITE( 4 , 9 ) CMNREC
      CALL CLNREC( CMNREC , 12 , 72 )
      RETURN
      END
```

```
      SUBROUTINE RPTFLD
      INTEGER RECFMT , FILLER(7) , CNTMRK
      REAL PSQ , DDSQ , VSQ(9)
      LOGICAL RCTYPE , STATUS , FMTID , KEYID
      BYTE MVEREC(52) , IFREC(62) , CMNREC(80) , FLDNM(16)
      COMMON /SEQBLK/ PSQ , DDSQ , VSQ
      COMMON /IFBLK/ IFREC
      COMMON /CRBLK/ CMNREC
      COMMON /MVEBLK/ MVEREC
      COMMON /VARBLK/ RECFMT , FLDNM , RCTYPE , STATUS , FILLER ,
     1                CNTMRK
    6 FORMAT(F7.0)
      FORMAT( 1H , 80A1 )
      CALL MOVE( 16 , FLDNM , IFREC(5) )
      IF ( CNTMRK .GT. 0 .AND. RCTYPE ) IFREC(4) = 'R'
      IF ( CNTMRK .GT. 0 ) RCTYPE = .FALSE.
      IF ( .NOT. RCTYPE ) CALL MOVE(13,'EQUAL SPACES ',IFREC(22))
      IF ( RCTYPE ) CALL MOVE( 12 , 'NOT NUMERIC ' , IFREC(22) )
      CALL MOVE( 61 , IFREC , CMNREC(12) )
      VSQ(RECFMT) = VSQ(RECFMT) + 1
      ENCODE( 7 , 66 , CMNREC ) VSQ(RECFMT)
      WRITE( 4 , 9 ) CMNREC
      CALL CLNREC( IFREC , 4 , 61 )
      CALL CLNREC( CMNREC , 12 , 72 )
      IF ( RCTYPE .AND. .NOT. STATUS ) CALL RDFTST
      IF ( CNTMRK .EQ. 0 ) GO TO 19
      CALL MOVE( 13 , 'NEXT SENTENCE' , CMNREC(12) )
      VSQ(RECFMT) = VSQ(RECFMT) + 1
      ENCODE( 7 , 66 , CMNREC ) VSQ(RECFMT)
      WRITE( 4 , 9 ) CMNREC
      CALL CLNREC( CMNREC , 12 , 72 )
      CALL MOVE( 4 , 'ELSE' , CMNREC(12) )
      VSQ(RECFMT) = VSQ(RECFMT) + 1
      ENCODE( 7 , 66 , CMNREC ) VSQ(RECFMT)
      WRITE( 4 , 9 ) CMNREC
      CALL CLNREC( CMNREC , 12 , 72 )
      RETURN
    9 CALL EDTFLD
      RETURN
      END
```

```
      SUBROUTINE CNDVAL
      INTEGER I , N , X , W , Y , WF , COLUMN , V , K , J , L
      INTEGER RECFMT , FLDIP , FLDLEN , FLDDEC , WMRK , CNTMRK
      REAL PSQ , DDSQ , VSQ(9)
      LOGICAL RCTYPE , STATUS , FMTID , KEYID
      BYTE FLDNM(16) , IFREC(61) , STRVAL(40) , CNDCNT(6)
      BYTE CNDREC(62) , CMNREC(80) , MVEREC(52)
      COMMON /VARBLK/ RECFMT , FLDNM , RCTYPE , STATUS , FMTID ,
     1                KEYID , FLDIP , FLDLEN , FLDDEC , CNTMRK
      COMMON /CNDBLK/ CNDREC
      COMMON /CRBLK/ CMNREC
      COMMON /IFBLK/ IFREC
      COMMON /SEQBLK/ PSQ , DDSQ , VSQ
      COMMON /MVEBLK/ MVEREC
      COMMON /WBLK/ COLUMN , WF , V , W
      DATA CNDCNT / 'T' , 'H' , 'R' , 'U' , ' ' , ' ' /
83    FORMAT(I3)
66    FORMAT(F7.0)
24    FORMAT( 40A1 , I1 )
9     FORMAT( 1H , 80A1 )
      Y = 0
      IF ( .NOT. STATUS ) CALL RPTFLD
      DO 98 I = 1 , 2
        I = 0
        WMRK = CNTMRK
        READ( 3 , 24 ) STRVAL , CNTMRK
        CALL MOVE( 40 , STRVAL , CNDREC(16) )
        X = 100 * WF + W
        ENCODE( 3 , 83 , CNDREC(6) ) X
        W = W + 1
        IF ( WMRK .EQ. 1 ) GO TO 55
        K = 15
        DO 96 J = 1 , 2
          J = 0
          K = K + 1
          IF ( CNDREC(K) .EQ. '-' ) J = 2
96        CONTINUE
        CNDREC(K) = ' '
        L = K - 1
        DO 88 J = 55 , L , -1
88        CNDREC( J + 5 ) = CNDREC(J)
        J = K + 1
        N = K + 5
        L = 0
        DO 53 K = J , N
          L = L + 1
53        CNDREC(K) = CNDCNT(L)
55      CALL MOVE( 61 , CNDREC , CMNREC(12) )
        DDSQ = DDSQ + 1
        ENCODE( 7 , 66 , CMNREC ) DDSQ
        WRITE( 4 , 9 ) CMNREC
        CALL CLNREC( CMNREC , 12 , 72 )
        IF ( Y .EQ. 1 ) GO TO 16
        Y = 1
        CALL MOVE( 6 , 'NOT V-' , IFREC(5) )
        CALL MOVE( 61 , IFREC , CMNREC(12) )
        GO TO 3
16      CALL MOVE( 10 , 'AND NOT V-' , CMNREC(12) )
3       ENCODE( 3 , 83 , CMNREC(22) ) X
        VSQ(RECFMT) = VSQ(RECFMT) + 1
        ENCODE( 7 , 66 , CMNREC ) VSQ(RECFMT)
        WRITE( 4 , 9 ) CMNREC
        IF ( CNTMRK .EQ. 0 ) I = 2
```

```
78        CONTINUE
          CNTMRK = 1
          CALL EDTFLD
          RETURN
          END



          SUBROUTINE KEYREC
          INTEGER RECFMT , FLDIP , FLDLEN , FILLER(9) , KV , KEYLEN
          REAL PSQ , DDSQ , VSQ(9) , ITVSQ , WSQ , RFLR(2)
          LOGICAL WKEYID
          BYTE CMNREC(80) , FLDREC(38) , FLDNM(16) , MVEREC(52)
          COMMON /VARBLK/ RECFMT , FLDNM , FILLER , FLDLEN
          COMMON /CRBLK/ CMNREC
          COMMON /FLDBLK/ FLDREC
          COMMON /WBLK/ RFLR , KV , KEYLEN , WKEYID
          COMMON /SEQBLK/ PSQ , DDSQ , VSQ , ITVSQ , WSQ
          COMMON /MVEBLK/ MVEREC
80        FORMAT(I2)
66        FORMAT(F7.0)
9         FORMAT( 1H , 80A1 )
          KV = KV + 1
          IF ( KV .NE. 1 ) GO TO 30
          WKEYID = .TRUE.
          CALL MOVE( 9 , '01  WKEY.' , CMNREC(8) )
          WSQ = WSQ + 1
          ENCODE( 7 , 66 , CMNREC ) WSQ
          WRITE( 4 , 9 ) CMNREC
          CALL CLNREC( FLDREC , 4 , 20 )
          CALL CLNREC( CMNREC , 8 , 9 )
30        CALL MOVE( 4 , 'KEY-' , FLDREC(4) )
          ENCODE( 2 , 80 , FLDREC(8) ) KV
          ENCODE( 7 , 66 , CMNREC ) WSQ
          CALL MOVE( 38 , FLDREC , CMNREC(12) )
          WSQ = WSQ + 1
          ENCODE( 7 , 66 , CMNREC ) WSQ
          WRITE( 4 , 9 ) CMNREC
          CALL MOVE( 16 , FLDNM , MVEREC(6) )
          CALL MOVE( 4 , 'KEY-' , MVEREC(26) )
          ENCODE( 2 , 80 , MVEREC(30) ) KV
          MVEREC(32) = ','
          CALL MOVE( 52 , MVEREC , CMNREC(12) )
          PSQ = PSQ + 1
          ENCODE( 7 , 66 , CMNREC ) PSQ
          WRITE( 4 , 9 ) CMNREC
          CALL CLNREC( MVEREC , 6 , 21 )
          CALL CLNREC( MVEREC , 26 , 52 )
          KEYLEN = KEYLEN + FLDLEN
          CALL CLNREC( CMNREC , 12 , 72 )
          RETURN
          END
```

```
SUBROUTINE PRSMPL
INTEGER RECFMT , FLDIP , FLDLEN , FLDDEC , FMTS , V , CNTMRK
1         , WF , COLUMN , N , SW , VF(9) , TFMTS(9) , J , NRV
1         , VFMTS(9) , INTFIL(3)
REAL PSQ , DDSQ , VSQ(9) , FILLER , ITVSQ
LOGICAL RCTYPE , STATUS , FMTID , KEYID , WKEYID , WRT
BYTE FLDREC(38) , CMNREC(80) , MVEREC(52) , FMTHDR(14) ,
1     FLRREC(22) , FLDNM(16) , CNDREC(62) , RDFREC(60)
COMMON /VARBLK/ RECFMT , FLDNM , RCTYPE , STATUS , FMTID ,
1                KEYID , FLDIP , FLDLEN , FLDDEC , CNTMRK
COMMON /DATBLK/ FMTS , FILLER , TFMTS , VFMTS
COMMON /LBLK/ SW , NRV
COMMON /NBLK/ N
COMMON /WBLK/ COLUMN , WF , V , INTFIL , WKEYID
COMMON /SEQBLK/ PSQ , DDSQ , VSQ , ITVSQ
COMMON /CRBLK/ CMNREC
COMMON /MVEBLK/ MVEREC
COMMON /HDRBLK/ FMTHDR
COMMON /TABBLK/ VF
COMMON /CNDBLK/ CNDREC
COMMON /FLRBLK/ FLRREC
COMMON /FLDBLK/ FLDREC
COMMON /RDFBLK/ RDFREC
FORMAT(F7.0)
FORMAT(I4)
FORMAT( I1 , 16A1 , 4L1 , 2I4 , I2 , I1 )
FORMAT(I1)
FORMAT( 1H , 80A1 )
WRT = RCTYPE
IF ( RECFMT .GT. WF ) CALL CHFMT
V = V + 1
CALL TBLSVL
IF ( FMTS .GT. 1 .AND. FMTID ) CALL CPYID
IF ( FLDIP .NE. COLUMN ) CALL GENFLR
N = 32
CALL MOVE( 16 , FLDNM , FLDREC(4) )
FLDREC(25) = '9'
ENCODE( 4 , 43 , FLDREC(27) ) FLDLEN
FLDREC(32) = '.'
IF ( .NOT. RCTYPE ) FLDREC(25) = 'X'
IF ( RCTYPE .AND. FLDDEC .GT. 0 ) CALL NUMPIC
CALL MOVE( N , FLDREC , CMNREC(12) )
DDSQ = DDSQ + 1
ENCODE( 7 , 66 , CMNREC ) DDSQ
WRITE( 4 , 9 ) CMNREC
CALL CLNREC( FLDREC , 4 , 19 )
CALL CLNREC( CMNREC , 12 , 72 )
CALL MOVE( 18 , 'COMPUTE V = V + 1.' , CMNREC(12) )
VSQ(RECFMT) = VSQ(RECFMT) + 1
ENCODE( 7 , 66 , CMNREC ) VSQ(RECFMT)
WRITE( 4 , 9 ) CMNREC
CALL CLNREC( CMNREC , 12 , 72 )
IF ( RCTYPE .AND. CNTMRK .EQ. 0 ) CALL RPTFLD
IF ( .NOT. RCTYPE .AND. STATUS .AND. CNTMRK .EQ. 0 ) CALL RPTFLD
IF ( KEYID ) CALL KEYREC
COLUMN = FLDIP + FLDLEN
IF ( CNTMRK .NE. 0 ) CALL CNDVAL
IF ( WRT .AND. .NOT. STATUS ) CALL GRFREC
READ(3,18,END=54) RECFMT , FLDNM , RCTYPE , STATUS , FMTID ,
```

```
      1          KEYID , FLDIF , FLDLEN , FLDREC , CNTMRK
      SW = 0
      RETURN
64    SW = 2
      MVEREC(6) = '1'
      IF ( .NOT. WKEYID .OR. FMTS .EQ. 1 ) GO TO 64
      CALL MOVE( 18 , 'FORMAT-STATUS ( ).' , MVEREC(26) )
      CALL MOVE( 52 , MVEREC , CMNREC(12) )
      DO 19 J = 1 , FMTS
        IF( TFMTS(J) .EQ. 0 ) GO TO 19
        ITVSQ = ITVSQ + 1
        ENCODE( 1 , 39 , CMNREC(52) ) J
        ENCODE( 7 , 66 , CMNREC ) ITVSQ
        WRITE( 4 , 9 ) CMNREC
19      CONTINUE
64    IF ( NRV .EQ. 0 ) GO TO 83
      CALL CLNREC( MVEREC , 26 , 52 )
      CALL MOVE( 7 , 'VF ( ).' , MVEREC(26) )
      CALL MOVE( 52 , MVEREC , CMNREC(12) )
      DO 26 J = 1 , FMTS
        IF ( VFMTS(J) .EQ. 0 ) GO TO 26
        ITVSQ = ITVSQ + 1
        ENCODE( 1 , 39 , CMNREC(41) ) J
        ENCODE( 7 , 66 , CMNREC ) ITVSQ
        WRITE( 4 , 9 ) CMNREC
26      CONTINUE
83    CALL CLNREC( MVEREC , 6 , 21 )
      CALL CLNREC( MVEREC , 26 , 52 )
      CALL CLNREC( CMNREC , 12 , 72 )
      RETURN
      END
```

```
      SUBROUTINE PRVAL
      INTEGER I , N , X , CV , FMT , WCV , WFMT
      REAL VALSEQ(9) , FILLER(13)
      BYTE CMNREC(80) , HDRREC(80) , STRING(50) , PGMFLE(35)
      COMMON /CRBLK/ CMNREC
      COMMON /SEQBLK/ FILLER , VALSEQ
      COMMON /VALBLK/ X
      DATA HDRREC / 80 * ' ' /
91    FORMAT( I1 , 50A1 , I1 )
39    FORMAT(I1)
66    FORMAT(F7.0)
9     FORMAT( 1H , 80A1 )
      CALL DISP( 31 , 'ARCHIVO DE RELACIONES A VALIDAR' , PGMFLE )
      OPEN( UNIT = 6 , NAME = PGMFLE , TYPE = 'OLD' )
      CALL MOVE( 9 , 'IF NOT ( ' , CMNREC(12) )
      CALL MOVE( 10 , 'CHECK-VAL-' , HDRREC(8) )
      HDRREC(19) = ','
      N = 0
      WCV = 0
      READ( 6 , 91 ) FMT , STRING , CV
      ENCODE( 1 , 39 , HDRREC(18) ) FMT
      VALSEQ (FMT) = VALSEQ (FMT) + 1
      ENCODE( 7 , 66 , HDRREC ) VALSEQ(FMT)
      WRITE( 4 , 9 ) HDRREC
      WFMT = FMT
      CALL MOVE( 50 , STRING , CMNREC(21) )
      READ( 6 , 91 , END = 84 ) FMT , STRING , CV
      DO 58 I = 1 , 2
        I = 0
        IF ( FMT .EQ. WFMT ) GO TO 83
        CMNREC(72) = ')'
        VALSEQ(WFMT) = VALSEQ(WFMT) + 1
        ENCODE( 7 , 66 , CMNREC ) VALSEQ(WFMT)
        WRITE( 4 , 9 ) CMNREC
        CALL CLNREC( CMNREC , 12 , 72 )
        CALL MOVE( 17 , 'COMPUTE I = I + 1' , CMNREC(14) )
        VALSEQ(WFMT) = VALSEQ(WFMT) + 1
        ENCODE( 7 , 66 , CMNREC ) VALSEQ(WFMT)
        WRITE( 4 , 9 ) CMNREC
        CALL CLNREC( CMNREC , 14 , 72 )
        CALL MOVE( 24 , 'MOVE N TO REC-ST-VAL (I)' , CMNREC(14) )
        VALSEQ(WFMT) = VALSEQ(WFMT) + 1
        ENCODE( 7 , 66 , CMNREC ) VALSEQ(WFMT)
        WRITE( 4 , 9 ) CMNREC
        CALL CLNREC( CMNREC , 14 , 72 )
        CALL MOVE( 16 , 'MOVE 1 TO SWVAL.' , CMNREC(14) )
        VALSEQ(WFMT) = VALSEQ(WFMT) + 1
        ENCODE( 7 , 66 , CMNREC ) VALSEQ(WFMT)
        WRITE( 4 , 9 ) CMNREC
        CALL CLNREC( CMNREC , 14 , 72 )
        N = N + 1
        CALL MOVE( 9 , 'IF NOT ( ' , CMNREC(12) )
        IF ( N .GT. X ) X = N
        N = 0
        WFMT = FMT
        ENCODE( 1 , 39 , HDRREC(18) ) FMT
        VALSEQ (FMT) = VALSEQ (FMT) + 1
        ENCODE( 7 , 66 , HDRREC ) VALSEQ(FMT)
        WRITE( 4 , 9 ) HDRREC
```

```
          GO TO 63
83        IF ( CV .NE. 0 ) GO TO 81
          CMNREC(72) = ')'
          VALSEQ(FMT) = VALSEQ(FMT) + 1
          ENCODE( 7 , 66 , CMNREC ) VALSEQ(FMT)
          WRITE( 4 , 9 ) CMNREC
          CALL CLNREC( CMNREC , 12 , 72 )
          CALL MOVE( 17 , 'COMPUTE I = I + 1' , CMNREC(14) )
          VALSEQ(FMT) = VALSEQ(FMT) + 1
          ENCODE( 7 , 66 , CMNREC ) VALSEQ(FMT)
          WRITE( 4 , 9 ) CMNREC
          CALL CLNREC( CMNREC , 14 , 72 )
          CALL MOVE( 24 , 'MOVE N TO REC-ST-VAL (I)' , CMNREC(14) )
          VALSEQ(FMT) = VALSEQ(FMT) + 1
          ENCODE( 7 , 66 , CMNREC ) VALSEQ(FMT)
          WRITE( 4 , 9 ) CMNREC
          CALL CLNREC( CMNREC , 14 , 72 )
          CALL MOVE( 16 , 'MOVE 1 TO SWVAL.' , CMNREC(14) )
          VALSEQ(FMT) = VALSEQ(FMT) + 1
          ENCODE( 7 , 66 , CMNREC ) VALSEQ(FMT)
          WRITE( 4 , 9 ) CMNREC
          CALL CLNREC( CMNREC , 14 , 72 )
          N = N + 1
          CALL MOVE( 18 , 'COMPUTE N = N + 1.' , CMNREC(12) )
          VALSEQ(FMT) = VALSEQ(FMT) + 1
          ENCODE( 7 , 66 , CMNREC ) VALSEQ(FMT)
          WRITE( 4 , 9 ) CMNREC
          CALL CLNREC( CMNREC , 12 , 72 )
          CALL MOVE( 9 , 'IF NOT ( ' , CMNREC(12) )
          GO TO 63
81        VALSEQ(FMT) = VALSEQ(FMT) + 1
          ENCODE( 7 , 66 , CMNREC ) VALSEQ(FMT)
          WRITE( 4 , 9 ) CMNREC
          CALL CLNREC( CMNREC , 12 , 72 )
63        CALL MOVE( 50 , STRING , CMNREC(21) )
          WCV = CV
          READ( 6 , 91 , END = 84 ) FMT , STRING , CV
58        CONTINUE
84      CMNREC(72) = ')'
        VALSEQ(FMT) = VALSEQ(FMT) + 1
        ENCODE( 7 , 66 , CMNREC ) VALSEQ(FMT)
        WRITE( 4 , 9 ) CMNREC
        CALL CLNREC( CMNREC , 12 , 72 )
        CALL MOVE( 17 , 'COMPUTE I = I + 1' , CMNREC(14) )
        VALSEQ(FMT) = VALSEQ(FMT) + 1
        ENCODE( 7 , 66 , CMNREC ) VALSEQ(FMT)
        WRITE( 4 , 9 ) CMNREC
        CALL CLNREC( CMNREC , 14 , 72 )
        CALL MOVE( 24 , 'MOVE N TO REC-ST-VAL (I)' , CMNREC(14) )
        VALSEQ(FMT) = VALSEQ(FMT) + 1
        ENCODE( 7 , 66 , CMNREC ) VALSEQ(FMT)
        WRITE( 4 , 9 ) CMNREC
        CALL CLNREC( CMNREC , 14 , 72 )
        CALL MOVE( 16 , 'MOVE 1 TO SWVAL.' , CMNREC(14) )
        VALSEQ(FMT) = VALSEQ(FMT) + 1
        ENCODE( 7 , 66 , CMNREC ) VALSEQ(FMT)
        WRITE( 4 , 9 ) CMNREC
        CALL CLNREC( CMNREC , 14 , 72 )
        N = N + 1
        IF ( N .GT. X ) X = N
        RETURN
        END
```

ANEXO B

PROGRAMA EDITOR

```
-EBCBLPG4,0A1
^US UC -1UY -1UX < 0A-13"E 0UX 0UY' 0A-0C"E 0UX C ' 0X; C >
1800 1801 1802 1803 1804 Y
< < 0UX 0 (0A-55)"L 10X ' 1%1
    (01-56)"E 4804 1%3
          (03-58)"E 4803 1%2
                (02-58)"E 4802 1%1
                      (01-58)"E 4801 1%0 ' ' ' '
  7D 0031// 0131// 0231// 0331// 0431// -10
    0U"E
      U 3F < (0A-33); F > C I !  0L 11C
    -< (0A-33)+ C-> 0UD
      32MS 1U6 -1UY
      < 01-33"E 0UY' 0Y;
       0A-32"G 0UA
        0A-32"F
            0A"G
              D R 1%DA
              10B'
        C >
    0L <-45MS 0Y> 2R
          0A-32"G
                2C < 0A-32"G 0U4 ' 0Y; D 1%D > '
        -0A-32"F 2C ' >
      0L < 4015 0Y; < 0A-32"G 0UY ' JY; D 1%D > >
      0L S!  -2D U0<I- >L 3R D I  '
      L .-Z; > P .-Z; >
EY
```

ANEXO C

PROGRAMA ESQUEMA UTILIZADO POR EL PAQUETE

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. VAL-FILE.
000300 ENVIRONMENT DIVISION.
000400 CONFIGURATION SECTION.
000500     SOURCE-COMPUTER. HP3000.
000600     OBJECT-COMPUTER. HP3000.
000700     SPECIAL-NAMES. TOP IS CANAL-1.                                  36
000800 INPUT-OUTPUT SECTION.
000900 FILE-CONTROL.
001000*    SELECT INFILE ASSIGN TO
001100     SELECT LINEX ASSIGN TO LINEX.
001200*    SELECT REPVAL ASSIGN TO REPVAL.
001300@DATA DIVISION.
001400 FILE SECTION.
001500 FD  LINEX.
001600 01  RLINE PIC X(132).
001700*FD  REPVAL.                                                        28
001800 01  REPREC PIC X(132).                                            27
001900@FD  INFILE
002000*    RECORD CONTAINS      CHARACTERS
002100*    BLOCK CONTAINS       RECORDS.
110000 WORKING-STORAGE SECTION.                                          26
110100 77  AP PIC 9.
110200*77  FORMATS VALUE   PIC 9.
110300@77  KLI PIC 9.
110400 77  N VALUE 0 PIC 9.
110500 77  SL VALUE 0 PIC 9.
110600 77  SV PIC 9.
110700*77  SWDUP PIC 9.
110800@77  WCH VALUE 0 PIC 9.                                             28
110900 77  LI VALUE 0 PIC 9(2).
111000*77  INTLENGTH VALUE     PIC 9(3).
111100 77  H VALUE 0 PIC 9(4).
111200 77  I PIC 9(4).
111300 77  J PIC 9(4).
111400 77  K PIC 9(4).
111500 77  Q VALUE 0 PIC 9(4).
111600 77  V VALUE 0 PIC 9(4).
111700 77  W PIC 9(4).
111800 77  RL VALUE 0 PIC 9(9).
111900 77  RCF VALUE 0 PIC 9(9).
112000 77  REF VALUE 0 PIC 9(9).
112100 77  CHR PIC X.
112200*77  SWVAL VALUE 0 PIC 9.
112300 77  LIV VALUE 0 PIC 9(2).
112400 77  HV VALUE 0 PIC 9(4).
112500 77  RV VALUE 0 PIC 9(9).
112600 77  RCV VALUE 0 PIC 9(9).
112700 77  REV VALUE 0 PIC 9(9).
112800@01  CL PIC X.
112900     88 CLV VALUES ARE "1" THRU "9".                                35
113000 01  WF PIC X.
113100@01  TAB-FORMAT-1.
113200*    02 REC-FORMAT-1 OCCURS      TIMES.
113300        03 IP-1 PIC 9(4).
113400        03 FP-1 PIC 9(4).                                           35
113500 01  TAB-FORMAT-2.
113600*    02 REC-FORMAT-2 OCCURS      TIMES.
113700        03 IP-2 PIC 9(4).
113800        03 FP-2 PIC 9(4).                                           35
113900@01  TAB-FORMAT-3.
114000*    02 REC-FORMAT-3 OCCURS      TIMES.
```

```
14100        03 IP-3 PIC 9(4).
14200        03 FP-3 PIC 9(4).
14300 01   TAB-FORMAT-4.
14400*     02 REC-FORMAT-4 OCCURS       TIMES.                          35
14500        03 IP-4 PIC 9(4).
14600        03 FP-4 PIC 9(4).
14700 01   TAB-FORMAT-5.
14800*     02 REC-FORMAT-5 OCCURS       TIMES.                          35
14900        03 IP-5 PIC 9(4).
15000        03 FP-5 PIC 9(4).
15100 01   TAB-FORMAT-6.
15200*     02 REC-FORMAT-6 OCCURS       TIMES.                          35
15300        03 IP-6 PIC 9(4).
15400        03 FP-6 PIC 9(4).
15500 01   TAB-FORMAT-7.
15600*     02 REC-FORMAT-7 OCCURS       TIMES.                          35
15700        03 IP-7 PIC 9(4).
15800        03 FP-7 PIC 9(4).
15900 01   TAB-FORMAT-8.
16000*     02 REC-FORMAT-8 OCCURS       TIMES.                          35
16100        03 IP-8 PIC 9(4).
16200        03 FP-8 PIC 9(4).
16300 01   TAB-FORMAT-9.
16400*     02 REC-FORMAT-9 OCCURS       TIMES.                          35
16500        03 IP-9 PIC 9(4).
16600        03 FP-9 PIC 9(4).
16700@01   GRAL-TAB.
16800*     02 GRAL-TAB-REC OCCURS       TIMES.                          35
16900        03 IP PIC 9(4).
17000        03 FP PIC 9(4).
17100 01   TAB-STATUX VALUE ZEROS.
17200*     02 STATUX OCCURS       TIMES PIC 9(2).                       29
17300 01   DIMENSIONS-FORMATS-TABLE VALUE ZEROS.
17400*     02 DFT OCCURS    TIMES PIC 9(4).                              26
17500*     02 VF OCCURS    TIMES PIC 9.                                  25
17600@01   KEY-TAB.
17700*     02 REC-KEY-TAB OCCURS    TIMES.                              34
17800        03 FORMAT-STATUX PIC 9.
17900        03 FSB PIC 9.
18000@01   INFILE-TAB.
18100*     02 INFILE-REC OCCURS      TIMES PIC X(100).                   33
18200 01   SEG-TAB.
18300*     02 SEG-REC OCCURS      TIMES PIC X(100).                     30
18400 01   STR REDEFINES SEG-TAB.
18500*     02 WR OCCURS       TIMES PIC X.                               25
18600 01   TOTREC.
18700        02 FILLER VALUE SPACES PIC X(30).
18800        02 FILLER VALUE 'REGISTROS LEIDOS' PIC X(17).
18900        02 FILLER VALUE SPACES PIC X(12).
19000        02 FILLER VALUE 'CORRECTOS' PIC X(10).
19100        02 FILLER VALUE SPACES PIC X(12).
19200        02 FILLER VALUE 'ERRONEOS' PIC X(9).
19300 01   TRNV.
19400        02 FILLER VALUE SPACES PIC X(35).
19500        02 ERL PIC ZZZ,ZZZ,ZZ9.
19600        02 FILLER VALUE SPACES PIC X(11).
19700        02 ERCF PIC ZZZ,ZZZ,ZZ9.
19800        02 FILLER VALUE SPACES PIC X(10).
19900        02 EREF PIC ZZZ,ZZZ,ZZ9.
20000 01   ST-REC.
20100        02 FILLER VALUE SPACES PIC X(9).
20200        02 FILLER VALUE '>>>' PIC X(5).
20300        02 SUB-ST-REC.
20400          03 FILLER PIC X(101).
20500          03 TS OCCURS 5 TIMES PIC ZZB.
20600        02 FILLER VALUE SPACES PIC X(2).
```

```
120700 01  TAB-HC.
120800     02 FILLER VALUE SPACES PIC X(14).
120900     02 HC OCCURS 100 TIMES PIC X.
121000     02 FILLER VALUE SPACES PIC X(16).
121100 01  TAB-LC.
121200     02 FILLER VALUE SPACES PIC X(14).
121300     02 LC OCCURS 100 TIMES PIC X.
121400     02 FILLER VALUE SPACES PIC X(18).
121500*01  TAB-ST-VAL VALUE ZEROS.                                          33
121600*    02 REC-ST-VAL OCCURS     TIMES PIC 9(2).                         28
121700*    02 DIMVAL VALUE     PIC 9(3).
121800@01  GRAL-REC.
121900     02 FILLER VALUE SPACES PIC X(3).
122000     02 ERLV PIC Z(9)BB.
122100     02 REFFLD PIC X(101).
122200     02 FILLER VALUE SPACES PIC X(17).
122300 01  HDR-1.
122400     02 FILLER VALUE SPACES PIC X(3).
122500     02 FILLER VALUE 'NUMERO DE' PIC X(11).
122600     02 FILLER VALUE 'DATOS DEL REGISTRO' PIC X(101).
122700     02 FILLER VALUE 'S T A T U S' PIC X(17).
122800 01  HDR-2.
122900     02 FILLER VALUE SPACES PIC X(4).
123000     02 FILLER VALUE 'REGISTRO' PIC X(8).
123100     02 FILLER VALUE SPACES PIC X(120).
123200 01  TIT.
123300     02 FILLER VALUE 'REPORTE DEL FILTRADO DEL ARCHIVO' PIC X(33).
123400*    02 FILLER VALUE '                    ' PIC X(20).              29
123500     02 FILLER VALUE SPACES PIC X(67).
123600     02 FILLER VALUE 'HOJA' PIC X(5).
123700     02 EH PIC ZZZ9.
123800*01  TIT-VAL.
123900     02 FILLER VALUE 'REPORTE DE LA VALIDACION DEL' PIC X(29).
124000     02 FILLER VALUE 'ARCHIVO' PIC X(8).
124100*    02 FILLER VALUE '                    ' PIC X(20).              29
124200     02 FILLER VALUE SPACES PIC X(66).
124300     02 FILLER VALUE 'HOJA' PIC X(5).
124400     02 EHV PIC ZZZ9.
124500 01  VAL-ST-REC.
124600     02 FILLER VALUE SPACES PIC X(14).
124700     02 ETV OCCURS 34 TIMES PIC Z9B.
124800     02 FILLER VALUE SPACES PIC X(16).
124900 01  VAL-REC.
125000     02 FILLER VALUE SPACES PIC X(28).
125100     02 FILLER VALUE 'REGISTROS VALIDADOS' PIC X(20).
125200     02 FILLER VALUE SPACES PIC X(13).
125300     02 FILLER VALUE 'CORRECTOS' PIC X(10).
125400     02 FILLER VALUE SPACES PIC X(13).
125500     02 FILLER VALUE 'ERRONEOS' PIC X(9).
125600 01  VRNV.
125700     02 FILLER VALUE SPACES PIC X(36).
125800     02 ERV PIC ZZZ,ZZZ,ZZ9.
125900     02 FILLER VALUE SPACES PIC X(11).
126000     02 ERCV PIC ZZZ,ZZZ,ZZ9.
126100     02 FILLER VALUE SPACES PIC X(10).
126200     02 EREV PIC ZZZ,ZZZ,ZZ9.
126300@01  DUP-MSG.
126400     02 FILLER VALUE SPACES PIC X(46).
126500     02 FILLER VALUE 'R E G I S T R O   D U P L I C A D O'
126600                    PIC X(35).
126700     02 FILLER VALUE SPACES PIC X(51).
126800@01  FMTS-REC.
126900     02 FILLER VALUE SPACES PIC X(45).
127000     02 FILLER VALUE 'REGISTROS FALTANTES' PIC X(20).
127100     02 TAB-ERF.
127200       03 ERF OCCURS 9 TIMES PIC 9B.
```

```
127300      02 FILLER VALUE SPACES PIC X(49).
127400@01   INV-FMT-REC.
127500      02 FILLER VALUE SPACES PIC X(49).
127600      02 FILLER VALUE 'F O R M A T O   E R R O N E O' PIC X(29).
127700      02 FILLER VALUE SPACES PIC X(54).
210000@PROCEDURE DIVISION.
210100 SKELETON.                                                        43
210200*     OPEN INPUT INFILE OUTPUT LINEX
210300      MOVE 1 TO K.
210400      MOVE 9 TO W.
210500      PERFORM FILL-VALUES VARYING I FROM 1 BY 1 UNTIL I = 11.
210600      MOVE 0 TO HC(100) LC(100) SV.
210700      PERFORM LOAD-TABLES.
210800      PERFORM READ-INFILE.
210900*     MOVE WKEY TO WKEYB.
211000@     PERFORM PROCESS-INFILE UNTIL SL = 1.
211100      COMPUTE RCF = RL - REF.
211200      MOVE RL TO ERL.
211300      MOVE RCF TO ERCF.
211400      MOVE REF TO EREF.
211500      WRITE RLINE FROM TOTREC AFTER 2.
211600      WRITE RLINE FROM TRNV AFTER 2.
211700*     COMPUTE RCV = RV - REV.
211800      MOVE RV TO ERV.
211900      MOVE RCV TO ERCV.
212000      MOVE REV TO EREV.
212100      WRITE REPREC FROM VALREC AFTER 2.
212200      WRITE REPREC FROM VRNV AFTER 2.                            31
212300@     CLOSE INFILE LINEX
212400      STOP RUN.
212500 READ-INFILE.
250000      READ INFILE INTO INFILE-TAB AT END MOVE 1 TO SL.
250100 PROCESS-INFILE.
250200      COMPUTE RL = RL + 1.
250300      MOVE 0 TO N V.
250400      MOVE SPACES TO SEG-TAB.
250500*     IF WKEY > WKEYB
250600        MOVE ' ' TO WF
250700        MOVE WKEY TO WKEYB
250800        PERFORM CHECK-FMT-TAB VARYING I FROM 1 BY 1
250900                UNTIL I > FORMATS
251000        IF N = 1
251100          MOVE 0 TO N
251200          PERFORM REPORT-FMTS.
251300      IF CLV
251400        IF CL = WF
251500          MOVE 1 TO SWDUP
251600          PERFORM DUP-REC
251700        ELSE
251800            MOVE CL TO WF.
251900@     IF WKEY = WBKEY
252000        MOVE 1 TO SWDUP
252100        PERFORM DUP-REC VARYING I FROM 1 BY 1
252200                UNTIL I > INTLENGTH
252300        WRITE RLINE FROM DUP-MSG AFTER 1
252400      ELSE
252500          MOVE WKEY TO WKEYB.
252600@     IF NOT CLV
252700        MOVE 0 TO SV
252800        MOVE 1 TO SWDUP
252900        MOVE RL TO ERLV
253000        MOVE 2 TO AF
253100        MOVE 4 TO KLI
253200        PERFORM REC-DUP-REC VARYING I FROM 1 BY 1
253300                UNTIL I > INTLENGTH                              29
253400*       PERFORM INV-REC
```

```
53500     ELSE
53600          MOVE 1 TO FSB (CL).
53700@    IF SWDUP = 0
53800       PERFORM CHECK-RECORD
53900     ELSE
54000          MOVE 0 TO SWDUP.
54100@    PERFORM CHECK-RECORD.
54200@    PERFORM READ-INFILE.
54300 CHECK-RECORD.
54400*    IF CL = '1'
54500       MOVE TAB-FORMAT-1 TO GRAL-TAB
54600       PERFORM LFPST VARYING I FROM 1 BY 1 UNTIL I >                60
54700       PERFORM CHECK-LABEL-1.
54800     IF CL = '2'
54900       MOVE TAB-FORMAT-2 TO GRAL-TAB
55000       PERFORM LFPST VARYING I FROM 1 BY 1 UNTIL I >                60
55100       PERFORM CHECK-LABEL-2.
55200     IF CL = '3'
55300       MOVE TAB-FORMAT-3 TO GRAL-TAB
55400       PERFORM LFPST VARYING I FROM 1 BY 1 UNTIL I >                60
55500       PERFORM CHECK-LABEL-3.
55600     IF CL = '4'
55700       MOVE TAB-FORMAT-4 TO GRAL-TAB
55800       PERFORM LFPST VARYING I FROM 1 BY 1 UNTIL I >                60
55900       PERFORM CHECK-LABEL-4.
56000     IF CL = '5'
56100       MOVE TAB-FORMAT-5 TO GRAL-TAB
56200       PERFORM LFPST VARYING I FROM 1 BY 1 UNTIL I >                60
56300       PERFORM CHECK-LABEL-5.
56400     IF CL = '6'
56500       MOVE TAB-FORMAT-6 TO GRAL-TAB
56600       PERFORM LFPST VARYING I FROM 1 BY 1 UNTIL I >                60
56700       PERFORM CHECK-LABEL-6.
56800     IF CL = '7'
56900       MOVE TAB-FORMAT-7 TO GRAL-TAB
57000       PERFORM LFPST VARYING I FROM 1 BY 1 UNTIL I >                60
57100       PERFORM CHECK-LABEL-7.
57200     IF CL = '8'
57300       MOVE TAB-FORMAT-8 TO GRAL-TAB
57400       PERFORM LFPST VARYING I FROM 1 BY 1 UNTIL I >                60
57500       PERFORM CHECK-LABEL-8.
57600     IF CL = '9'
57700       MOVE TAB-FORMAT-9 TO GRAL-TAB
57800       PERFORM LFPST VARYING I FROM 1 BY 1 UNTIL I >                60
57900       PERFORM CHECK-LABEL-9.
58000@    MOVE TAB-FORMAT-1 TO GRAL-TAB.
58100*    PERFORM LFPST VARYING I FROM 1 BY 1 UNTIL I >                58
58200     PERFORM CHECK-LABEL-1.
58300     IF SV > 0
58400       MOVE 0 TO SV K Q
58500       MOVE 1 TO W
58600       MOVE 2 TO AP
58700       MOVE 4 TO KLI
58800       MOVE RL TO ERLV
58900       COMPUTE REF = REF + 1
59000       PERFORM REPORT-STATUX VARYING I FROM 1 BY 1
59100*           UNTIL I > INTLENGTH                                      40
59200     ELSE
59300*         IF VF (  ) = 1                                             24
59400           MOVE 0 TO I
59500           MOVE 1 TO N
59600           COMPUTE RV = RV + 1
59700*          PERFORM                                                  27
59800           IF SWVAL = 1
59900             MOVE 0 TO SWVAL
160000            COMPUTE REV = REV + 1
```

```
                    PERFORM REPORT-VAL.
0000@CHECK-FMT-TAB.
      IF FORMAT-STATUX (I) = 1
        IF FSB (I) = 0
          MOVE 1 TO N
          MOVE I TO ERF (I)
        ELSE
            MOVE 0 TO FSB (I).
00700 REPORT-FMTS.
      IF LI < 4
        PERFORM TITLES.
      WRITE RLINE FROM FMTS-PLC AFTER 2.
      COMPUTE LI = LI + 4.
01200@DUP-REC.
      MOVE RL TO ERLV.
      MOVE 2 TO AP.
      MOVE 4 TO KLI.
      PERFORM REP-DUP-REC VARYING I FROM 1 BY 1 UNTIL
              I > INTLENGTH.
      IF LI < 4
        PERFORM TITLES.
      WRITE RLINE FROM DUP-MSG AFTER 2.
      COMPUTE LI = LI + 4.
02200@REP-DUP-REC.
      IF LI < 4
        PERFORM TITLES.
      MOVE INFILE-REC (I) TO REPFLD.
      WRITE RLINE FROM GRAL-REC AFTER AP.
      COMPUTE LI = LI + KLI.
      MOVE 0 TO ERLV.
      MOVE 1 TO AP.
      MOVE 2 TO KLI.
03100@REPORT-STATUX.
      MOVE INFILE-REC (I) TO REPFLD.
      IF LI < 4
        PERFORM TITLES.
      WRITE RLINE FROM GRAL-REC AFTER AP.
      COMPUTE LI = LI + KLI.
      MOVE 0 TO ERLV.
      MOVE SEG-REC (I) TO SUB-ST-REC.
      COMPUTE K = K + 100.
      PERFORM LOAD-STATUX VARYING J FROM W BY 1 UNTIL IP (J) > K
04100*              OR J > DFT ( ).
      IF Q > 0
        PERFORM CHECK-ST.
      MOVE J TO W.
      MOVE 1 TO AP.
      MOVE 2 TO KLI.
04700 LOAD-STATUX.
      IF STATUX (J) > 0
        COMPUTE Q = Q + 1.
        MOVE STATUX (J) TO TS (Q)
        MOVE 0 TO STATUX (J)
        IF Q = 5
          PERFORM CHECK-ST.
05400 CHECK-ST.
      IF LI < 4
        PERFORM TITLES.
      WRITE RLINE FROM ST-REC AFTER 1.
      COMPUTE LI = LI + 2.
      MOVE SPACES TO SUB-ST-REC.
      MOVE 0 TO Q.
06100*INV-REC.
      IF LI < 4
        PERFORM TITLES.
      WRITE RLINE FROM INV-FMT-REC AFTER 1.
```

```
506500      COMPUTE LI = LI + 2.
506600@TYPE-OF-ERROR.
506700      MOVE 1 TO SV.
506800      MOVE N TO STATUX (V).
506900      IF WCH = 0
507000        MOVE '*' TO CHB
507100        MOVE 1 TO WCH
507200      ELSE
507300           MOVE '@' TO CHB
507400           MOVE 0 TO WCH.
507500      PERFORM FILL-MARKS VARYING Q FROM IP (V) BY 1 UNTIL
507600            Q > FP (V).
507700 FILL-MARKS.
507800      MOVE CHB TO WR (Q).
507900 FILL-VALUES.
508000      COMPUTE SV = I - 1.
508100      PERFORM FILL-TABLES VARYING J FROM K BY 1 UNTIL J > W.
508200      COMPUTE K = W + 1.
508300      COMPUTE W = W + 10.
508400 FILL-TABLES.
508500      MOVE SV TO HC (J).
508600      COMPUTE N = N + 1.
508700      MOVE N TO LC (J).
508800 TITLES.
508900      COMPUTE H = H + 1.
509000      MOVE H TO EH.
509100      WRITE RLINE FROM TIT AFTER CANAL-1.
509200      WRITE RLINE FROM HDR-1 AFTER 2.
509300      WRITE RLINE FROM HDR-2 AFTER 1.
509400      WRITE RLINE FROM TAB-HC AFTER 2.
509500      WRITE RLINE FROM TAB-LC AFTER 1.
509600      MOVE SPACES TO RLINE.
509700      WRITE RLINE AFTER 1.
509800      MOVE 4 TO LI.
509900 LFPST.
510000      COMPUTE N = N + 1.
510100      PERFORM UNDL-FIELDS VARYING J FROM IP (I) BY 1 UNTIL
510200            J > FP (I).
510300 UNDL-FIELDS.
510400      MOVE N TO WR (J).
600000*VAL-FORMATS.
600100      IF CL = '1'
600200        PERFORM CHECK-VAL-1.
600300      IF CL = '2'
600400        PERFORM CHECK-VAL-2.
600500      IF CL = '3'
600600        PERFORM CHECK-VAL-3.
600700      IF CL = '4'
600800        PERFORM CHECK-VAL-4.
600900      IF CL = '5'
601000        PERFORM CHECK-VAL-5.
601100      IF CL = '6'
601200        PERFORM CHECK-VAL-6.
601300      IF CL = '7'
601400        PERFORM CHECK-VAL-7.
601500      IF CL = '8'
601600        PERFORM CHECK-VAL-8.
601700      IF CL = '9'
601800        PERFORM CHECK-VAL-9.
800000@REPORT-VAL.
800100      MOVE RL TO ERLV.
800200      MOVE 2 TO AP.
800300      MOVE 4 TO KLI.
800400      PERFORM REP-RECORD VARYING I FROM 1 BY 1 UNTIL I > INTLENGTH.
800500      MOVE 2 TO AP.
800600      MOVE 4 TO KLI.
```

```
.....     PERFORM REP-ST-REC VARYING I FROM 1 BY 1 UNTIL I > DIMVAL.
800700     IF J > 0
800800       MOVE 0 TO J
800900       PERFORM REP-VAL-REC.
801000
801100 REP-RECORD.
801200     IF LIV < 4
801300       PERFORM TITLES-VAL.
801400     MOVE INFILE-REC (I) TO REPFLD.
801500     WRITE REPREC FROM GRAL-REC AFTER AP.
801600     COMPUTE LIV = LIV + KLI.
801700     MOVE 0 TO ERLV.
801800     MOVE 1 TO AP.
801900     MOVE 2 TO KLI.
802000 REP-ST-REC.
802100     IF REC-ST-VAL (I) = 1
802200       COMPUTE J = J + 1
802300       MOVE I TO ETV (J)
802400       IF J = 34
802500         MOVE 0 TO J
802600         PERFORM REP-VAL-REC.
802700 TITLES-VAL.
802800     COMPUTE HV = HV + 1.
802900     MOVE HV TO EHV.
803000     WRITE RLINE FROM TIT-VAL AFTER CANAL-1.
803100     WRITE RLINE FROM HDR-1 AFTER 2.
803200     WRITE RLINE FROM HDR-2 AFTER 1.
803300     WRITE RLINE FROM TAB-HC AFTER 2.
803400     WRITE RLINE FROM TAB-LC AFTER 1.
803500     MOVE SPACES TO RLINE.
803600     WRITE RLINE AFTER 1.
803700     MOVE 4 TO LIV.
803800 REP-VAL-REC.
803900     IF LIV < 4
804000       PERFORM TITLES-VAL.
804100     WRITE REPREC FROM VAL-ST-REC AFTER AP.
804200     COMPUTE LIV = LIV + KLI.
804300     MOVE 1 TO AP.
804400     MOVE 2 TO KLI.
900000@LOAD-TABLES.
900100*
```

ANEXO D

EJEMPLOS DE DATOS PROPORCIONADOS POR EL USUARIO Y RESULTADOS OBTENIDOS

ARCHIVO DE VARIABLES

```
1FLDNM-1          TTFF00030001000
1FLDNM-2          TTFF00040004000
1FLDNM-3          TTFF00130002002
01-50                                    0
1FLDNM-4          TTFF00150001000
1FLDNM-5          TTFF00160003000
1FLDNM-6          FTFF00190002000
1FLDNM-7          TFFF00210002000
1FLDNM-8          FTFF00250002000
1FLDNM-9          TTFF00270002000
1FLDNM-10         TFFF00290002000
1FLDNM-11         TFFF00400002001
1 3 5                                    2
10-20                                    0
1FLDNM-12         TFFF00440001002
1-7                                      0
1FLDNM-13         TFFF00530003000
1FLDNM-14         TTFF00670002000
1FLDNM-15         FTFF00690001000
1FLDNM-16         FTFF00700001000
1FLDNM-17         FTFF00710001000
1FLDNM-18         FTFF00720001000
1FLDNM-19         TTFF00730002000
1FLDNM-20         TTFF00750002002
1-12                                     0
1FLDNM-21         TTFF00770002002
1-31                                     0
1FLDNM-22         FFFF00790002000
1FLDNM-23         FFFF00810028000
1FLDNM-24         TTFF01150002000
1FLDNM-25         TTFF01170003000
1FLDNM-26         TTFF01210002000
1FLDNM-27         TTFF01230006000
1FLDNM-28         TTFF01410008020
1FLDNM-29         TTFF01740001001
0 1                                      0
1FLDNM-30         FFFF02000001001
'S' 'N'                                  0
1FLDNM-31         FTFF02020001001
'N' 'E'                                  0
1FLDNM-32         FTFF02070002000
```

CARACTERISTICAS DEL ARCHIVO A FILTRAR Y VALIDAR

```
DATOS                INFILE     02200001110000000000000000
```

REPORTE DEL FILTRADO DEL ARCHIVO DATOS

```
                                                                                                   HOJA
   NUMERO DE  DATOS DEL REGISTRO                                                                   STATUS
   REGISTRO

           0000000001111111111222222222233333333334444444444555555555566666666667777777777888888888899999999990
           1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890

        1  00001320020611 5  #078001 P340618000002510991000000350004010310300002SOMLS00724OOLUIS J SORIANO MARTI
     >>>    12222        334***6677  889900              11  2          333                44567899001122333333333333333333333  2
           NEZ       83062300000##30770482080199999900000000000000000000300830716000000051000180000000000000000
     >>>   33333333       44555 66777777                  88888888                     9                           *  2  3
           IN0100 E00000069111 6

        2  000013190077115000078013 A901000000006991001000000350004010200000009RASRS4032800ROSA E RAMIREZ CANCH
           EZ        83062302040004807575307169999990000000000000000131940753071600000005100430000000000000000
           1A0300 A830501821101
     >>>    0   22                                                                                                  3

        4  00/0#3100086S1 000078013 A91079000001680100100000035000401020000009ROMES701120OJ EDMUNDO ROMAN MACI
     >>>     *@000       **05556677  889900              11  2          333                445678990011223333333333333333333  2  2  3
           AS        830623020400048075848307169999990000000000000000014325283071600000005100046000000000000000
     >>>   33333333       44555 66777777                  88888888                     9                           *  3
           1A0300 A830501821101
     >>>    0    22                                                                                                 3

        5  000991100013115000  ##13  92060000000522000000000000000000000000028ADC27091000CARLOTA BARRERA DIAC
     >>>    12222        334555**00  **9900              00  *          333                445678990011223333333333333333333  1  2  1  3
           IO     51830623000002000000009306019999990000000000000000000000000830716000000051000000000000000000000
     >>>   33333333       44555 66777777                  88888888                     9                           0  3
           000300 A00000000000 0
     >>>    *    22                                                                                                 3

        6  00099110012811500007801 3 A  ##0000000292300000000000###000000000000  MUSE5102140OMURATA SALAZAR JOSE
     >>>    12222        3345556677  8800**              00  *          000                **56789900112233333333333333333333  2  2  3  3  2
     >>>                                                                                                            2
           EDMUNDS18306230000020000000083060199999900000000000000000000000008307160000000510000000000000000000000
     >>>   33333333       44555 66777777                  88888888                     9                           0  3
           001000 A000000000000
     >>>    *    22                                                                                                 3

        7  0000130000031150000 78013 E301000000005703001000000350004010300100012        5310270OFILIBERTO VALDEZ MAR
     >>>    12222        3345556677  889900              11  2          333                440*0*9900112233333333333333333333  1  1
           QUE       8306230206120480750183071699999900000000000000000026505583071600000005100004000000000000000000
     >>>   33333333       44555 66777777                  88888888                     9                           0  3
           1A0500 E830101740301
     >>>    *    22                                                                                                 3

        8  00001300000511500007801 3 E3104120000002030010000000350004010300100008BABV##13##00VICTOR A BRAVO BURGO
     >>>    12222        3345556677  889900              11  2          333                4456780**002233333333333333333333  2  3
           S         830623020500048075038307169999990000000071758000098036030716000000051000590000000000000000
     >>>   33333333       44555 66777777                  88888888                     9                           *  3
           0A0500 E830101680301
     >>>    0    22                                                                                                 3

        9  000013100092115000078013 E32060900000140200100000003500040313300000008RUOR4507200ORAUL RUIZ ...
```

REPORTE DEL FILTRADO DEL ARCHIVO DATOS


```
NUMERO DE  DATOS DEL REGISTRO                                                                                      S T A T U S
REGISTRO

            0000000001111111111222222222233333333334444444444555555555566666666667777777777888888888899999999990
            1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
      >>>     12222     3345556677  889900            **  2         333             4456789900112233333333333333333333  3
                     830623#####004807590830716999999000000000378457200554160830716000000051000590000000000000000
      >>>   33333333        @@*** 66777777               88888288                       9                       @  2  2  3
            0A0400 E821101670401
      >>>    *    22                                                                                               3

        10  00001300002511150000780l3 P2316000000013109910900003500040103001000020IG8480102005ALVADOP URRIETA GAR
            CIA       830623000000048 75238:6101999990#0000000000000000000000008307160002000510001500000000000000
      >>>   33333333      44555 66000000               ********                         @                       *  2     3
            100100  000000740901
      >>>    @     **                                                                                              3

        11  0009911000811150000780l3 S03040000001292300000000000000000000000000002     #####00HERNANDEZ MACIEL TIN
      >>>     12222     3345556677  889900            @@  *         333             440*0*00**002333333333333333333333  3 - 3
      >>>
            DTB0 0518306230000020000000008306019999990000000000000000000000000083071600000005100000000000000000000    1  2  3  3
      >>>   33333333      44555 66777777               88888888                         9                           3
            001000 S04000000000000
      >>>    @    22                                                                                               3

        12  0000131000981150000780l3 S0304000000368010010300007500040102000000009VEML59031300MA LOURDES VERANO ME
            DIN       830623020400048075968307169999990000000000000000001319407830716000000051000480000000000000000
      >>>   33333333      44555 66777777               88888888                         9                           3
            1A0300 S830501821101
      >>>    @    22                                                                                               3

        13  00001310010511150000780l3 S030400000037601001000000035000401020000000098EGC611031000MA CARMEN VELAZQUEZ
            GRO       830623020400048076038307169999990000000000000000001219407830716000000051000430000000000000000
      >>>   33333333      44555 66777777               88888668                         9                        *  3
            1A0300 S830501821101
      >>>    @    22                                                                                               3

        14  00001320021311150000780l3 S030700000072250100100000035000401020000000009RODE47102200ELODIA RGUEZ DELGADO
                      830623020400048077118307169999990000000000000000001432528307160000000510006200000000000000000
      >>>   33333333      44555 66777777               88888888                         9                        *  3
            1A0300 S830201830201
      >>>    @    22                                                                                               3

        15  00001310011311150000780l3 S030700000046201001000000035000401020000000098AJR62032300ROCIO BARRAGAN SAN J
            UAN       830623020400048076118307169999990000000000000000001432528307160000000510004800000000000000000
      >>>   33333333      44555 66777777               88888888                         9                        *
            1A0300 S830501821101
      >>>    @    22                                                                                               3

        16  00001310012011150000780l3 S0307000000472010010000000350004010200000009RUVS61110300SILVIA RUIZ VARGAS
                      830623020400048076188307169999990000000000000000001432528307160000000510004800000000000000000
      >>>   33333333      44555 66777777               88888680                         9                        *  3
            1A0300 S830501821101
```

REPORTE DEL FILTRADO DEL ARCHIVO DATOS

NUMERO DE REGISTRO | DATOS DEL REGISTRO

```
          00000000011111111112222222223333333333444444444555555555566666666677777777788888888899999999990
          12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
   >>>  0    22
 17  0000131001271-I5000078013 $03070000004790100100000035000401020000009BATJ56080500JOSEFA BADILLO TREJO
          830625020400048076258307169999990000000000000000011432528307100000005190048000000000000000
          1A0300 $83050182110I
   >>>  0    22

                    REGISTROS LEIDOS          CORRECTOS          ERRONEOS
                          17                     1                  16
```

ANEXO E

BIBLIOGRAFIA

BIBLIOGRAFIA

COMPUTER ORIENTED BUSINESS SYSTEMS
Wayne S. Boutell

MACROPROCESSORS AND TECHNIQUES FOR PORTABLE SOFTWARE
P. J. Brown

AN INTRODUCTION TO DATA STRUCTURES WITH APPLICATIONS
Tremblay, Sorenson

FUNDAMENTALS OF SYSTEMS ANALYSIS
Fitzgerald J, Fitzgerald A, Stallings W. D.