**Informe Técnico**

**Enunciado del Proyecto:**

Tomando como base el programa “Aleatorio” 1, introducir todas las modificaciones, adaptaciones e incorporaciones que consideren necesarias para cumplir con las siguientes solicitudes:

a. Modificar el registro Estudiante, incorporando, a los campos existentes, el DNI y el Apellido.

b. Modificar el código del programa de modo tal que no se permitan realizar incorporaciones o modificaciones en el archivo de Estudiantes (alumnos.dat) que generen u ocasionen que más de un estudiante posea el mismo DNI (el DNI debe ser único y no puede existir más de un registro con el mismo número de DNI).

c. Todas las consultas de datos, modificaciones y eliminaciones de datos deben hacerse solicitando el DNI del estudiante.

d. Incorporar al programa, un módulo que realice la operación “consultar existencia”. Este módulo debe emplear como método “búsqueda dicotómica” mediante el DNI del estudiante.

e. Agregar un módulo que, solicite al usuario el ingreso de una ciudad, y muestre por pantalla el listado de estudiantes que habitan en la ciudad ingresada. Para este módulo deberán utilizar indefectiblemente la estructura de Lista Doblemente Encadenada con Elemento Ficticio para almacenar en memoria los estudiantes que cumplan la condición de pertenecer a la ciudad ingresada. Como un agregado a este módulo, se dará la posibilidad al usuario de que elimine un estudiante del listado que se está visualizando. Al hacerlo, deberá mostrarse nuevamente el listado por pantalla y actualizar la eliminación en el archivo correspondiente.

f. Incorporar una funcionalidad que calcule el promedio de edad de estudiantes comprendidos entre un rango de edad mínima y máxima (determinadas por el usuario). Para realizar el cálculo de la edad promedio se deberá implementar una función recursiva que reciba como parámetro una Lista Simplemente Encadenada con Elemento Ficticio con los datos necesarios para el cálculo. Finalmente mostrar por pantalla la edad promedio.

g. Modificar el procedimiento “Menu” para poder acceder a las nuevas funcionalidades que hagan falta.

**Lenguaje de programación:** Pascal

**Versión del compilador:** Free Pascal Compiler version 2.6.2-8 for i386 Copyright (c) 1993-2012 by Florian Klaempfl and others .

**Diseño**

**Introducción**

En Base al enunciado anterior la forma de comenzar a pensar en el diseño del programa fue medianamente simple, pues al ya tener parte del código resuelto y los enunciados ser tan precisos, lo principal fue dividir los problemas en tantos módulos como incisos en el enunciado y a partir de ese momento realizar un análisis más exhaustivo sobre cada módulo. En cuanto al diseño general de la solución del problema, optamos por trabajar internamente con la estructura de lista doblemente encadenada con elementos ficticios al inicio y fin de la misma, que si bien es una estructura que no posee acceso indexado a sus componentes, tiene la ventaja (por ser una estructura de datos dinámica) de poder realizar cambios tales como: inserciones, eliminaciones, modificaciones, supresiones, ordenamientos, búsquedas, etc. De una manera mucho más rápida que una estructura de almacenamiento persistente como lo son los archivos (RAM: nanosegundos (10–9seg.); Disco: milisegundos (10–3seg.))

Para terminar de afianzar la idea de la utilización de listas doblemente encadenadas como estructura principal para el desarrollo de la solución del problema, implementamos un programa pascal que consistía en crear nodos de manera “infinita” del tipo compuesto Testudiante (el mismo utilizado en la resolución de este proyecto) para comprobar la cantidad de nodos que podrían ser creados aproximadamente (según la memoria asignada al programa por el S.O) en la ejecución del proyecto terminado, ya que de nada nos serviría la ventaja de realizar cambios en los componentes de forma más rápida, si esto implicaba tener una cantidad reducida de elementos en el sistema.

Al resultar de forma muy satisfactoria dicha prueba (ver foto 1), concluimos por unanimidad que debíamos utilizar esta estructura, contemplando sus ventajas y limitaciones, además del costo que tendríamos que saldar para implementar la solución con dicha estructura, pero comparado con los beneficios resulto un costo mínimo y necesario.

Luego de tener en claro lo dicho anteriormente se procedió con el análisis de los sub-problemas mencionados en el enunciado y los que surgieron por el uso de la estructura de datos seleccionada para desarrollar la solución. A continuación se detalla dicho análisis de cada Sub-Problema a resolver. Cabe destacar que cada Sub-Problema también fue particionado en Sub-Problemas aún más pequeños para facilitar la solución de los mismos y la correcta modularización de la solución algorítmica, haciendo honor a la frase “Divide y Vencerás”.

**Diseño**

A continuación se detallan y explican las formas seleccionadas para resolver cada inciso del enunciado.

**Listado de diseño de las soluciones:**

**Inciso A:** Este inciso no requirió de demasiado análisis pues su enunciado es muy claro y solo debió agregarse campos en el registro Testudiante (utilizado en el código del programa original) según las indicaciones para adaptarse a la resolución del resto de los incisos. Nota: se realizaron algunas modificaciones no establecidas en los incisos pero necesarias para el correcto funcionamiento de la solución que decidimos implementar.

Programa Original: Proyecto:

Estudiante = record Testudiante = Record;

Nombre: Cadena40; dni: longint;

Calle: string [20]; nombre: string [25];

Ciudad: string [25]; apellido: string [25];

CodigoPostal: string [5]; calle: string [25];

Edad: integer; ciudad: string [25];

Sw: boolean; codigoPostal: longint;

end; edad: longint; End;

**Inciso B:** Lo que se solicita en este inciso es utilizado en los módulos:

\***LeerRegistros:** Modulo que se encarga de solicitar al usuario los datos necesarios para agregar al sistema un nuevo elemento del tipo Testudiante.

\***ModificarDni:** Sub-Modulo de la acción **Modificar,** encargado de modificar el DNI de un elemento ya existente en el sistema.

En los Módulos mencionados arriba, se valida que el elemento a insertar o modificar sea distinto a alguno ya existente, es decir, que cada DNI cargado en el sistema sea único.

**Inciso C:** Para el cumplimiento de esta consigna el análisis derivo principalmente en que cada vez que el usuario solicite consultar, modificar u eliminar un elemento del sistema se le solicitara el ingreso del DNI correspondiente al elemento deseado, teniendo especiales consideraciones dependiendo de la acción a realizar, tales como:

\***BorrarElemento:** La consideración principal en este modulo fue comprobar la existencia del DNI ingresado para posteriormente realizar la eliminación o informarle al usuario que el elemento a eliminar no existe.

\***Modificar:** En este modulo la consideración tomada fue la de solicitar al usuario el DNI a modificar, si el elemento existe en el sistema y se desea modificar el campo DNI del mismo, al leerlo se comprueba que el nuevo dato no sea igual a uno ya existente, en ese caso, se solicita un reingreso hasta que ingrese un DNI no existente. En caso de querer modificar otro campo, solo se solicita la información correspondiente para la modificación.

\***ConsultarElemento**: La consideración de este modulo fue pura y exclusivamente por su funcionamiento pues solicita al usuario el ingreso de un DNI y lo busca en el sistema, en caso de existir imprime todos sus datos por pantalla, caso contrario imprime “El Estudiante no Existe”.

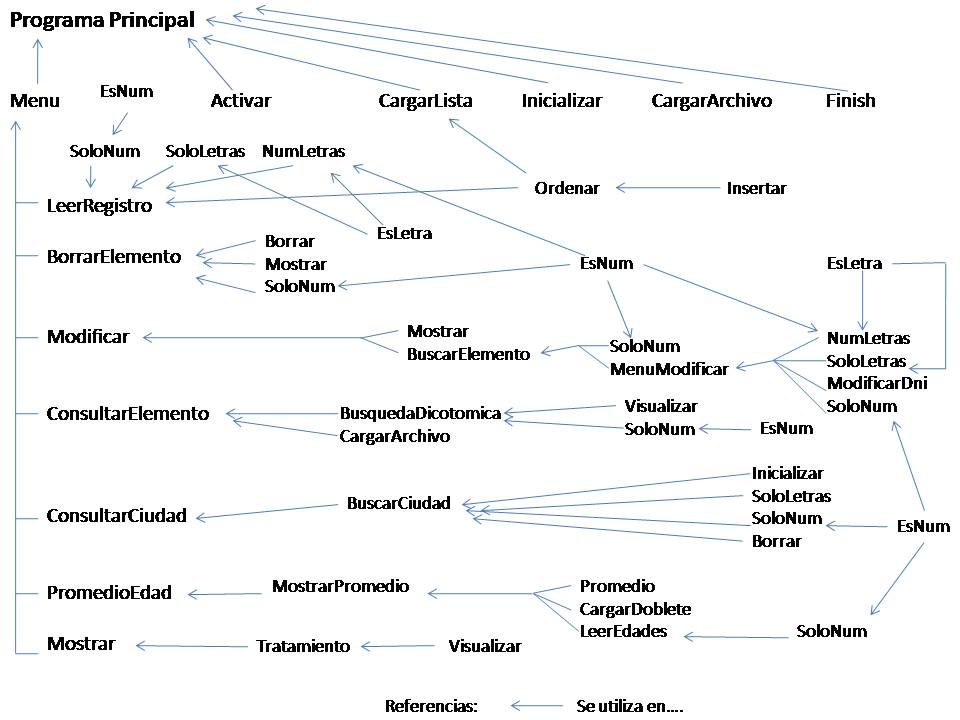
**Inciso D:** En este inciso se solicita incorporar al menú la funcionalidad de consultar existencia, la cual es realizada en nuestra solución por una acción llamada **ConsultarElemento**, la cual solicita al usuario el ingreso de un DNI y mediante una búsqueda dicotómica busca el elemento y si existe imprime sus datos por pantalla, sino imprime “El Estudiante no Existe”.

**Inciso E:** Para la resolución de este inciso se emplea la acción llamada **ConsultarCiudad** la cual solicita al usuario el ingreso de una ciudad la cual será consultada en el sistema e imprimirá por pantalla a todos los estudiantes que residan en la ciudad ingresada. Luego se le da la opción de eliminar algún estudiante de esa ciudad solicitando su DNI, en caso de existir y pertenecer a esa ciudad procede con la eliminación. En caso de no existir el DNI ingresado o no pertenecer a la ciudad dada se imprime por pantalla “El estudiante no pertenece a la ciudad ingresada”.

**Inciso F:** Para este punto se implemento una acción llamada **PromedioEdad** q recibe del usuario los rangos de edad y calcula el promedio de todos los estudiantes dentro de ese rango.

**Inciso G:** Se modifico el **Menú** principal del programa original, para agregar las funcionalidades solicitadas en el enunciado.

A continuación se ilustra cómo interactúan los módulos con el programa principal según nuestro análisis y diseño de la solución, para su posterior implementación algorítmica.



A partir de este esquema se detallan los Sub-Módulos pertenecientes a cada módulo principal.

**CargarArchivo:** Acción que se encarga de cargar un archivo de tipo TEstudiante llamado “alumnos.dat” con los datos almacenados en una lista doblemente encadenada con ficticio (al principio y final). Este Sub-Modulo se utiliza en la ConsultarElemento.

**Activar:** Acción encargada de verificar la existencia del archivo “alumnos.dat”. En caso de que no exista, crea el mismo.

**CargarLista:** Acción cuya tarea es la de, dado un archivo del tipo TEstudiante, insertar de forma ordenada los datos del mismo en una lista doblemente encadenada con ficticio al principio y final).

**Inicializar:** Esta acción tiene como tarea cargar los elementos ficticios al inicio y al fin de una lista doblemente encadenada dada como parámetro de entrada. Es un Sub-Modulo de: BuscarCiudad. y el programa principal.

**Ordenar:** Decidimos pagar el costo de mantener siempre los datos ordenados y este procedimiento se encarga de insertar datos del tipo Testudiante en una lista doblemente encadenada de manera ordenada, de menor a mayor según el campo DNI. Es un Sub-Modulo de CargarLista, LeerRegistro y ModificarDni.

**SoloNum:** Este es uno de los procedimientos más utilizado en el programa pues es necesario en muchos módulos principales. Se encarga de solicitar al usuario el ingreso de un numero y guardarlo, comprobando que el ingreso sea valido (números positivos). Es Sub-Módulo de: BusquedaDicotomica, Modificar, Menu Modificar, BuscarDni, LeerRegistro, LeerEdades, BorrarElemento, BuscarCiudad

**SoloLetras:** Se encarga de solicitar al usuario el ingreso de una cadena y guardarlo, comprobando que el ingreso sea valido (letras y espacio). Es Sub-Módulo de: LeerRegistro, MenuModificar, BuscarCiudad.

**Borrar:** Procedimiento que recibe un DNI, lo busca en la lista doblemente encadenada que contiene todos los datos del sistema y si existe lo elimina. En caso de no tener elementos almacenados, imprime” No hay Elementos”. Y si el elemento no se encuentra imprime” El elemento no existe”. Es Sub-Módulo de: BorrarElemento, BuscarCiudad.

**MenuModificar:** Este procedimiento es simple pues muestra un menú en la función modificar y permite elegir que campos modificar de un estudiante, Es un Sub-Módulo de Modificar.

**LeerEdades:** Esta acción se encarga de solicitar al usuario el ingreso de un rango de edades para su posterior uso en PromedioEdad, validando que el rango sea de números positivos y la segunda edad mayor o igual a la primera. Este Sub-Modulo se utiliza en la MostrarPromedio

**Promedio:** Función que recibe una lista simplemente encadenada cargada con los datos de estudiantes en un rango de edad “Min..Max”, la cantidad de elementos en dicha lista y devuelve el promedio de edad entre los estudiantes de esa lista (Calculado mediante una recursividad en cola). Es un Sub-Módulo de MostrarPromedio.

**CargarDoblete:** Acción que recibe un rango de edad “Min, Max” y carga en una lista simplemente encadenada todos los estudiantes pertenecientes a dicho rango. Es un Sub-Módulo de MostrarPromedio.

**BuscarDni:** Esta acción se encarga de solicitarle al Usuario un DNI y verificar si ese DNI ya es existente o esta disponible. Este Sub-Modulo se utiliza en la LeerRegistro, ModifiarDni.

**Insertar:** Esta Acción se encarga de crear un doblete agregarlo en una posición determinada anteriormente. Este Sub-Modulo se utiliza en la Ordenar.

**BuscarElemento:** Esta Acción se encarga de pedirle al usuario que ingrese el DNI a modificar en cual efectuaremos una búsqueda para ver si ese DNI existe y poderlo modificar en caso contrario el DNI ingresado no existe. Este Sub-Modulo se utiliza en la Modificar.

**BuscarCiudad:** Esta Accion se encarga de pedirle al usuario que ingrese la Ciudad, si la encuentra muestra todos los estudiantes con esa ciudad, en cual podra borrarlo o volver al menu. Este Sub-Modulo se utiliza en la ConsultarCiudad.

**EsLetra:** Esta Funcion se encarga de verificar si el Caracter que ingreso es una letra y devolvera verdadero, en caso contrario falso. Este Sub-Modulo se utiliza en la SoloLetras.

**MostrarPromedio:** Esta Accion se encarga de Recibir las edades minima y maxima, y carga un doblete con el rango de esas edades y asi poder sacar el promedio. Este Sub-Modulo se utiliza en la PromedioEdad.

**Visualizar:** Esta Accion se encarga de recibir el Estudiante y mostrar todos sus campos Nombre, Apellido, DNI, Calle, Ciudad, Codigo Postal, Edad. Este Sub-Modulo se utiliza en la Mostrar, BusquedaDicotomica.

**BusquedaDicotomica:** Esta Accion se encarga de recibir el Archivo y el usuario debera ingresar el DNI del estudiante a buscar y a traves de la dicotomica buscaremos al estudiante y si no lo encuentra, el estudiante que ingreso no existe. Este Sub-Modulo se utiliza en la ConsultarElemento.

**Mostrar:** Esta Accion se encarga de Recorrer toda la lista , si hay elemento mostrara cada uno de los Estudiantes con sus respectivos campos. Este Sub-modulo se utiliza en la BorrarElemento, BuscarCiudad, Modificar.

**EsNum:** Esta función recibe un carácter y devuelve true si es un carácter numérico.

Este sub-módulo es utilizado en SoloNum y NumLetras.

**NumLetras:** Esta acción recibe carga una cadena que no contenga caracteres raros.

Este sub-módulo es utilizado en LeerRegistro y MenuModificar.

**Observacion:** Los módulos **tratamiento, barras, inicio y finish** fueron hechos simplemente con fines de mejorar la interacción grafica del programa, pero no influyen en su correcto funcionamiento, por lo tanto no consideramos relevante la especificación de los mismos, además de ser meramente exclusivos del lenguaje de programación utilizado. Pero vale aclarar que:

\*tratamiento es sub-módulo de Mostrar.

\*barras es sub-módulo de inicio y fin.

\*inicio es sub-módulo de CargarLista.

\*fin es un módulo del programa principal.

NOTA: Los módulos no especificados en el listado anterior fueron especificados en el listado de **diseño de las soluciones** de cada inciso planteado en el enunciado del problema.