UTN – FACULTAD REGIONAL BUENOS AIRES
Materia: Algoritmos y Estructura de Datos – 2020
CURSO K1031

TRABAJO PRÁCTICO NÚMERO N°1

GRUPO

Cabanas Eliçabe, Manuel Julián	177.562-5	mcabanaselicabe@est.frba.utn.edu.ar
Parente, Franco Daniel	157.907-1	fparente14@est.frba.utn.edu.ar
Quiñones Garnica, Kevin Harold	173.110-5	kquinones@est.frba.utn.edu.ar

FECHA DE PRESENTACIÓN: 01/10/2020 FIRMA PROFESOR _____

FECHA DE DEVOLUCIÓN: CALIFICACIÓN _____

Informe

El trabajo que se nos asignó constaba de un programa que sea capaz de realizar diversas tareas, las cuales serían seleccionadas a través de un menú. Estas tareas son en relación a una base de datos de una aseguradora, la cual comprende un archivo que contiene pólizas de sus clientes (Los clientes pueden tener más de una póliza).

Lo solicitado fue:

Al comenzar la jornada se "levantan" todas las pólizas desde el archivo "Asegurados.BAK". Cada vez que el usuario requiera se procesará un "lote de incidentes". Se desconoce cuántos incidentes puede haber en cada lote, pero puede ser tan grande que haga difícil su manejo en un vector.
El procesamiento de un lote de incidentes implica la actualización de la cantidad de incidentes de aquellas pólizas que sean afectadas. Además, cada vez que se procesa un lote, deben "trasladarse" los registros procesados a un archivo llamado "procesados.BAK", el cual contiene todos los incidentes de todos los lotes procesados en el día.
Al finalizar el día se reescribe el archivo "Asegurados.BAK". con las pólizas activas únicamente y sus cantidades de incidentes actualizadas.
Ud. y su equipo deben confeccionar un programa que permita:

1. *"Levantar" las cuentas del archivo "Asegurados.BAK".*
2. *Cargar una nueva póliza.*
3. *Desactivar una póliza existente.*
4. *Buscar una póliza por Nro. de Póliza o por DNI(Un cliente puede tener más de una póliza).*
5. *Listar todas las pólizas activas ordenadas por incidentes descendentes.*
6. *Procesar un lote de incidentes.*
7. *Finalizar jornada (sobreescribir "Asegurados.BAK".).*
8. *Procedimiento de prueba para crear un incidente en el archivo (para luego poder procesarlo).*

Lo primero que hicimos fue crear dos structs distintos. Uno para las pólizas de los asegurados y otro para los incidentes que se cargarán en el sistema.

Incidente	
CódigoIncidente	9999
FechaHora	AAAAMMDDHH:MM
DNIAsegurado	99.999.999
DNIOtroConductor	99.999.999
Calle	NombreCalle
Altura	99999

Registro de incidente

Póliza Asegurado	
Nro. Póliza	9999
DNI	99.999.999
Nombre	NombreAsegurado
Apellido	ApellidoAsegurado
CuotaAlDia	V o F
PatenteAuto	AAAAMMDDHH:MM
Activa	V o F
Cdad. incidentes	9999

Registro de asegurado

Luego pensamos en crear un menú para que el usuario pueda elegir qué tarea realizar en cada momento. Este menú iría en el main, y cada opción tendría asignada una tarea distinta, para las cuales creamos distintos subprogramas.

Comenzamos por el primer procedimiento, en el cual se permite al usuario cargar una nueva póliza en el sistema. Aquí fue donde tuvimos que crear el archivo “Asegurados.bak” que pide el enunciado. Al cargar la póliza se encuentra activa por defecto y lógicamente la cuota está al día, ya que pertenece a un cliente nuevo.

Luego quisimos hacer un procedimiento que, al ser invocado, muestre el archivo previamente creado. De esta manera se mostrarían todas las pólizas que están cargadas en el sistema, sin importar su estado (activa o inactiva, cuota al día o adeudada).

Para que el usuario pueda desactivar una póliza creamos un subprograma que le pida al usuario el número de póliza que desea desactivar. Una vez ingresado el número de póliza, ésta es desactivada y actualizada en el archivo.

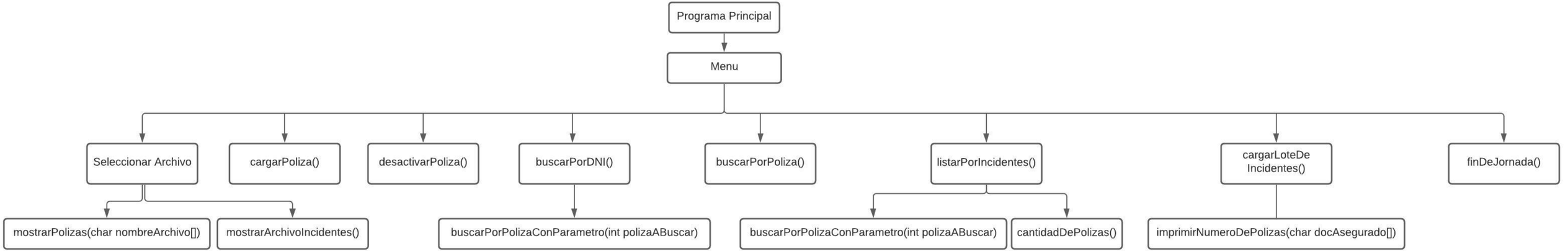
Se nos pide que el usuario pueda buscar por número de póliza o por DNI. Aquí tuvimos problemas con el subprograma para buscar por DNI, dado que al existir la posibilidad de que cada DNI esté asociado a más de una póliza, al encontrar la primer póliza, nuestro subprograma terminaba de inmediato. Lo que tuvimos que hacer fue crear un subprograma que busque por póliza (similar al que usamos luego también para buscar por número de póliza) para luego poder utilizarlo en el subprograma que buscaría por número de DNI. Entonces el usuario podría ingresar el número de DNI que desee y ver todas las pólizas que estén asociadas a él. Finalmente, para buscar por número de póliza, el procedimiento fue más sencillo. El usuario ingresa el número de póliza que necesite, y al ser únicos los números de póliza, al ser encontrado en el archivo, el subprograma imprime la póliza asociada a ese número (si es que está activa) e inmediatamente finaliza.

Cuando pensamos el cómo listar las pólizas por cantidad de incidentes de forma descendente, lo primero que pensamos fue almacenar todas las cantidades de incidentes de los asegurados, para luego ordenarlos con el algoritmo de burbujeo y posterior a ello, compararlos con las pólizas existentes (utilizando un subprograma que nos indica la cantidad de pólizas) e ir imprimiéndolas por pantalla reutilizando los subprogramas hechos para buscar por póliza. La única complicación fue la de que sucedía a la hora de encontrar dos pólizas con la misma cantidad de incidentes, pero se pudo solucionar.

A la hora de procesar un lote de incidentes tuvimos varias complicaciones. No logramos que nos incremente la cantidad de incidentes de una póliza, luego de cargarle un incidente, por lo que decidimos crear un “sub-subprograma” que imprima las pólizas asociadas al DNI ingresado por el usuario, y que luego permita seleccionar la póliza deseada. Finalmente el subprograma para cargar los incidentes incremente la cantidad de incidentes de dicha póliza y se guarda en el archivo de los asegurados.

Para finalizar lo jornada lo que necesitábamos era que el programa verifique las pólizas activas y las guarde en el archivo, pero nosotros no queríamos perder la información de las pólizas que fueron desactivadas. Para lograr esto pensamos en crear un archivo auxiliar, donde se volcarían todas las pólizas activas, y en el archivo original permanecerían todas las pólizas (activas e inactivas). De esta

manera creímos que facilitaríamos las tareas del usuario en caso de desactivar erróneamente una póliza.
Con respecto a la manera de trabajar en equipo, nos organizamos de la siguiente manera. Lo primero que hicimos fue hacer una reunión (Discord) para entender qué era lo que se nos pedía, armamos una estructura conceptual de lo que sería el programa y luego cualquiera de nosotros que estuviese disponible comenzaría a codearlo. Y así íbamos relevándonos a medida que nos liberábamos. A veces, cuando un subprograma presentaba alguna complicación en particular, hacíamos una reunión e intentábamos solucionarlo en conjunto. Si hubiese que determinar quién se encargó en su mayoría de cada programa podríamos establecer lo siguiente:
Manuel Cabanas particularmente se encargó de los subprogramas “Mostrar archivos de incidentes”, “Cantidad de pólizas”, “Imprimir número de polizas”, “Fin de jornada” y el de “SeleccionarArchivo”.
Kevin Quiñones se ocupó de programar los de “Cargar póliza”, “Mostrar pólizas”, “Desactivar pólizas”, “Buscar póliza con parámetro” y “Listar por incidentes”.
Franco Parente llevó a cabo los de “Buscar por DNI”, “Buscar por póliza” y luego el presente desarrollo del TP así como también una revisión del formato del programa para facilitar su legibilidad.
El de “cargar lote de incidentes”, al ser el que más conflictos nos presentó, lo hicimos juntos, al igual que los diagramas.


Programa Principal
Menu
Seleccionar Archivo
cargarPoliza()
desactivarPoliza()
buscarPorDNI()
buscarPorPoliza()
listarPorIncidentes()
cargarLoteDe Incidentes()
finDeJornada()
mostrarPolizas(char nombreArchivo[])
mostrarArchivoIncidentes()
buscarPorPolizaConParametro(int polizaABuscar)
buscarPorPolizaConParametro(int polizaABuscar)
cantidadDePolizas()
imprimirNumeroDePolizas(char docAsegurado[])