**Universidad Católica Andrés Bello**

**Facultad de Ingeniería**

**Escuela de Ingeniería Informática**

**Cátedra: Algoritmos y Programación II**

**Prof. Gloria Tarrio**

Informe del Proyecto

Destino Final

**Integrantes:**

* Arrieta, Moisés
* De Oliveira, José
* Drachenberg, Christian
* Flores, Jhonaire

**Especificaciones funcionales**

En el presente informe se detallara las distintas estructuras, procesos y funciones que permiten el correcto funcionamiento del programa. A su vez se explicará en brevedad los objetivos y las consideraciones especiales que se tuvieron presente durante el desarrollo del proyecto.

El objetivo de este proyecto consiste en realizar un programa mediante el uso del código C y el uso de listas y árboles que permita llevar a cabo la gestión de los nacimientos y muertes de las personas en el mundo, así como también el destino que les aguarda después de la muerte.

Para lograr este objetivo se requieren de diversas rutinas con funciones variadas que satisfagan los requerimientos presentados a continuación:

* La creación de una rutina que permita administrar y visualizar los datos de las personas que nacen en el mundo, teniendo en cuenta datos como su fecha de nacimiento, nombre completo, país de nacimiento, género y un código que permite su identificación dentro del sistema.
* Crear una rutina que permita gestionar y visualizar una lista de categorías que a su vez contengan las distintas acciones que pueden realizar las personas. Al mismo tiempo la creación de una rutina que administre dichas acciones dentro de cada categoría.
* Implementación de procesos que permitan llevar un registro de las buenas o malas acciones y arrepentimientos realizados por las personas que se encuentran en el mundo y a su vez poder visualizar dicho registro para llevar a cabo un seguimiento del comportamiento de las personas.
* El desarrollo de distintas rutinas que permitan administrar y visualizar a las personas que fallecen, teniendo en cuenta el destino que les espera ya sea en el cielo, en el infierno o en el purgatorio. Así como también conociendo su comportamiento implementar dos procesos que permitan seleccionar si la personas en el cielo se conviertan en ángeles o las que se encuentran en el infierno se conviertan en demonios.
* Diseñar un proceso que permita ingresar a las personas al cielo una vez cumplida su sentencia en el purgatorio.

Debido a la falta de tiempo y de conocimiento de las personas encargadas de desarrollar este proyecto, para el correcto funcionamiento del programa se deben tomar en cuenta las siguientes consideraciones especiales:

* Las validaciones solo se efectuaron entre elementos del mismo tipo de variable “char, int, strings, etc.” Por lo que si se introduce un dato de tipo equivocado no se garantiza el buen funcionamiento del programa.
* Una vez eliminada la última persona de algún árbol de la lista de fechas o si el árbol esta vacio, no consulte los datos de dicha persona, ni intente mostrar el árbol inexistente luego de la eliminación porque ocasionara un “crash”. Sin embargo puede insertar nuevamente personas en el árbol y consultar normalmente (se recuerda que el proceso permite consultar datos de los **mortales**).

Hay que destacar que en la rutina básica de árboles **Borrar**,se hicieron pequeñas modificaciones para que fuera posible eliminar el nodo, aun si este es el único nodo en el árbol, también antes de borrar el nodo del árbol se agregó un pequeño fragmento de código que guarda el nodo antes de ser eliminado para luego, según el índice de comportamiento almacenado en el mismo enviar a la persona al cielo, infierno o purgatorio.

Otras mejoras mejores del proyecto consisten en que al administrar alguna lista, se muestran las listas correspondientes a cada caso previamente para poder observar en detalle el estado actual de la lista.

Para concluir este proyecto represento un reto a la altura de nuestros conocimientos, nos parece que fue una buena idea utilizar una lista de fechas para organizar los nacimientos por cada fecha ingresada, haciendo que cada nodo de la lista contenga un árbol con cada una de los nacimientos de dicha fecha, sin embargo esto nos impide o dificulta dibujar un árbol general con todos los nacimientos contenidos en el sistema.

El poco margen de tiempo, así como la reestructuración del programa para adaptarlo al uso de árboles no nos permitieron crear el programa como nos hubiera gustado o teníamos en mente, sin embargo, dicho programa cumple en su mayoría con lo estipulado, a excepción de algunos errores, problemas o requisitos sin resolver.

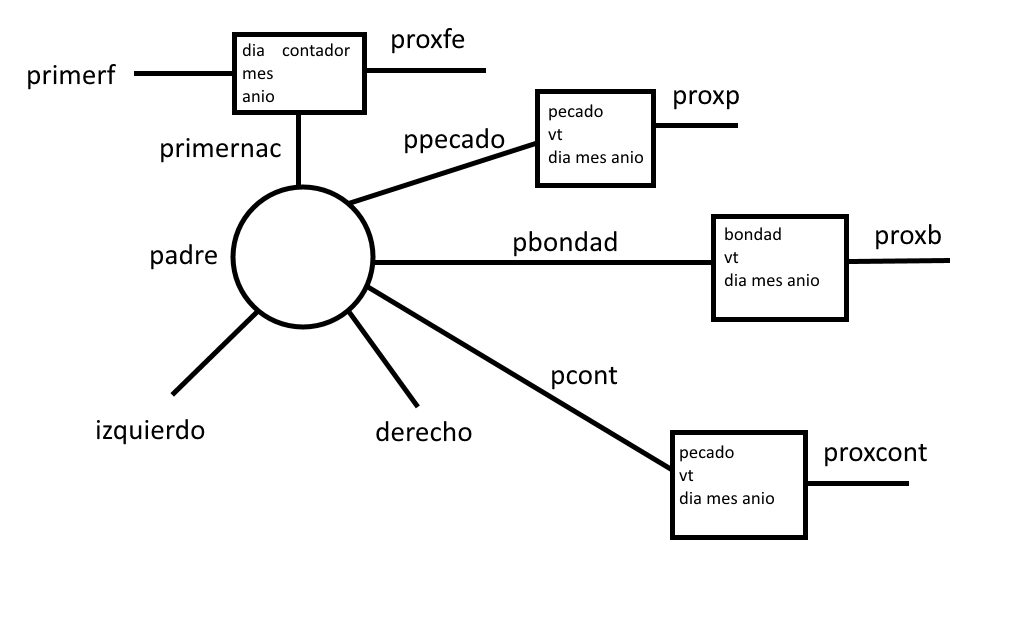
Uno de estos requisitos a los que hacemos referencia ocurre cuando al eliminar un pecado de una persona dada, al agregar dicho pecado a la respectiva lista de arrepentimientos de la persona este pecado queda eliminado sin dejar rastro a pesar de que se requería una eliminación lógica, es decir, que permaneciera allí aunque no se mostrara. Esto se vería solucionado creando una variable booleana en el struct correspondiente a la lista de pecados de la persona, así cuando se “elimine” el pecado y se agregué en arrepentimientos, en lugar de eliminar se da un valor False a la variable booleana y luego se pide en el proceso correspondiente a mostrar los pecados de las personas que solo muestre los nodos que tengan la variable booleana con valor TRUE.

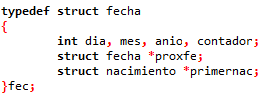
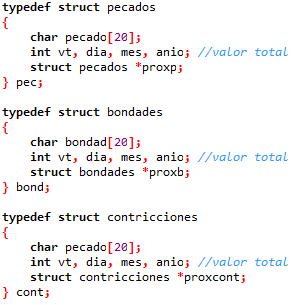
Un problema, por llamarlo así, sería lo explicado previamente en las consideraciones especiales; al eliminar el ultimo nodo de un árbol no se puede consultar los datos de una persona eliminada, ni dibujar el árbol, como es lógico, sin embargo se podrían revisar las rutinas básicas de árboles más a fondo y editarlas de forma q en lugar de hacer “crash” muestre alguna advertencia o no permita el acceso a estas opciones.

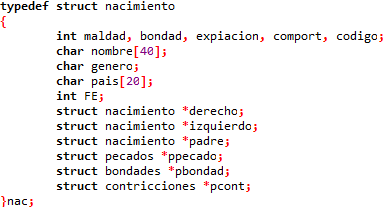
Como dijimos anteriormente por falta de tiempo no implementamos muchas de las cosas que nos hubiera gustado, así que como recomendación proponemos lo siguiente:

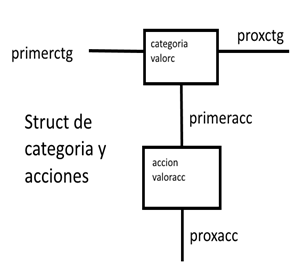
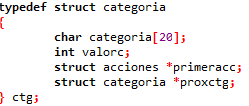
* Implementación de música por medio de librerías, acorde a cada una de las secciones del programa.
* Implementación de validaciones sin importar el tipo de variable.
* Mejoras a nivel visual de la interfaz gráfica en la que se muestra el programa.
* Desarrollar por medio de manejo de archivos una rutina que permita cargar distintos datos al programa por medio de dichos archivos.
* Presentar cada uno de los procesos y funciones relacionados a la administración de árboles dentro de una librería creada por nosotros mismos, que permita que el código fuera mucho más organizado y fácil de visualizar para el usuario.
* La fusión de los structs denominados pecados, bondades y contriciones en un struct general del mismo tipo, permitiendo así disminuir la cantidad de procesos y funciones utilizadas.
* La conversión de algunas funciones a procesos que funcionen con parámetros por referencia.
* La creación de una rutina que permita, en caso de que todas las personas murieran a la vez (apocalipsis, extinción, etc), que se vayan ingresando las personas a las respectivas listas de cielo, infierno o purgatorio según su comportamiento mientras se van “podando” los nodos de cada árbol de la lista de fechas mediante una versión modificada de la rutina básica de árboles **Podar**.

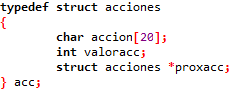
**Especificaciones de programación:**

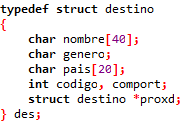
Diseño general de las estructuras de fecha y nacimiento así como también las estructuras de pecado, bondad y contrición:

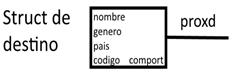


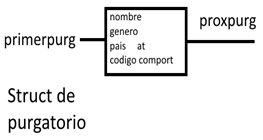
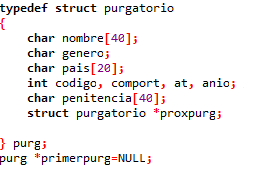


Diseño de los structs de categoría y acciones, destino y purgatorio:









A continuación se detallarán las rutinas desarrolladas en nuestro programa:

Dentro del main() encontramos algunos comentarios sobre el programa y una llamada al proceso void menu(), este contiene la interfaz gráfica de nuestro programa así como también las múltiples llamadas a las distintas rutinas que permitieron desarrollar el objetivo. Estas llamadas hacen referencia a las siguientes rutinas (**omitiendo las rutinas básicas de árboles dadas en clase**):

* **nac \*BuscarDevolver(Arbol a, int dat):** Esta función se encarga de buscar en el árbol recibido como parámetro, el nodo correspondiente al dato recibido y devuelve dicho nodo a la función. Para ello recorre el árbol buscando el nodo y una vez encontrado lo devuelve como una variable de tipo “nac” a la función.
* **fec \*comprobarFecha(fec \*p):** Esta función tiene por objetivo comprobar s la fecha ingresada se encuentra en la lista recibida, y en caso contrario llama a otra rutina que la creará y llamar a la rutina encargada de crear los nacimientos. Se recibe como parámetro una lista de tipo fec y devolverá una lista de tipo fecha con las modificaciones realizadas en las rutinas convocadas.

Se pide ingresar una fecha, si esta se encuentra en la lista de fechas, simplemente se llamara a la rutina encargada de crear nacimientos, si la fecha no se encuentra simplemente se llamara a la rutina encargada de crear la fecha y acto seguido se llamara a la encargada de crear nacimientos.

* **fec \*crearFecha(fec \*p, int d, int m , int a):** Es una función que se encarga de crear una fecha en la lista de fechas obtenida como parámetro. Recibe como parámetros una lista de fechas de tipo fec donde se creara la nueva fecha, y también recibe tres números enteros correspondientes a las variables dia, mes, anio respectivamente.

Se procede con los parámetros obtenidos a crear un nuevo nodo, al cual se le añadirán los distintos datos, y luego se retornara dicho nodo como el valor de la función.

* **fec \*crearNacimientos(fec \*q):** Esta función tiene por objetivo crear un nacimiento en el árbol correspondiente a cada nodo de la lista de fechas, éste recibe como parámetros únicamente dicha lista de fechas.

En esta función se pide ingresar el código que identificara a la persona dentro del sistema, así como otros datos sobre la persona, luego se hace un llamado a la rutina básica de árboles **Insertar** pasando como parámetros el primer nacimiento de la lista de fechas y el código, y se llama también a el proceso CalcularIndices que recibe la dirección del primer nacimiento y el código. Por último se retorna la lista de fecha actualizada.

* **ctg \*CrearCategoria(ctg \*p):** Esta función se encarga de crear una nueva categoría en la lista de categorías recibida como parámetro. Para ello se procede a crear un nuevo nodo, ingresándole los datos correspondientes y luego retorna dicho nodo a la función.
* **void mostrarCategoria(ctg \*p):** Este proceso tiene por objetivo mostrar la lista de categorías que recibe como parámetro por pantalla. Para ello recorre la lista e imprime por pantalla la información guardada en cada nodo de la lista.
* **ctg \*crearAccion(ctg \*q):** Esta función consiste en crear una acción dentro de la lista de acciones correspondiente a cada nodo de la lista de categorías, la cual recibe como parámetro. Para ello recorre la lista de categorías hasta encontrar donde se quiere crear la acción y luego se procede a crear un nuevo nodo en el apuntador a la primera acción de dicha categoría.
* **void mostrarAccionpC(ctg \*q):** Este proceso busca mostrar las acciones de un nodo especifico de la lista de categorías recibida por parámetros. Para ello se pide al usuario ingresar el nombre de la categoria y se recorre la lista hasta encontrar el nodo con el mismo nombre y luego se procede a recorres la lista de acciones del nodo e imprimir por pantalla los datos correspondientes.
* **Void mostrarAcciones(ctg \*p):** Este procedimiento se encarga de mostrar las acciones contenidas en toda la lista de categorías recibida por parámetros, para ello recorre la lista de categorías mientras que a su vez recorre las listas de acciones de cada nodo e imprime por pantalla dichas acciones.
* **ctg \*modificarCTG(ctg \*p):** Esta función se encarga de modificar los valores correspondientes a cada nodo de la lista de categorías que recibe como parámetro. Para ello se pide al usuario ingresar el nombre de la categoría a la cual se quiera cambiar el valor y se recorre la lista dada hasta encontrar el nodo con dicho valor y sobrescribir el mismo.
* **ctg \*modficarAccion(ctg \*p):** Esta función busca modificar los valores correspondientes a cada nodo de la lista de acciones contenida en cada nodo de la lista de categorías que recibe por parámetro, para ello se pide al usuario ingresar el nombre de la acción y se procede a recorrer la lista de categorías y a su vez la listas de acciones contenidas en cada nodo de la lista de categoría hasta encontrar el nodo que corresponda al nombre y sobrescribe el valor.
* **ctg \*EliminarCategoria(ctg \*p):** Esta función se encarga de eliminar una categoría de la lista de categorías recibida por parámetro siempre y cuando no contenga acciones en la lista de acciones correspondiente a cada nodo de la misma, para ello se pide al usuario ingresar el nombre de la categoría a eliminar y se recorre dicha lista, una vez encontrado el nodo a eliminar mediante el uso de la sentencia free y otros auxiliares se elimina dicho nodo y se retorna la lista actualizada a la función.
* **ctg \*EliminarAccion(ctg \*p):** La función se encarga de eliminar una acción contenida en la lista de acciones de algún nodo de la lista de categorías recibida como parámetro, para ello se pide al usuario ingresar el nombre de la acción y se recorre la lista de categorías , recorriendo a su vez cada lista de acciones de cada nodo y una vez encontrada la acción correspondiente se elimina mediante el uso de la sentencia free y otros auxiliares.
* **void FaltasCometidas(fec \*\*q, ctg \*p):** Este proceso se encarga de agregar un pecado en la lista de pecados de cada persona, siempre y cuando el pecado se encuentre en la lista de categorías, para ello se pide al usuario el nombre del pecado cometido, asi como el código de la persona que lo cometió, acto seguido se busca dicho pecado en las listas de acciones de la lista de categoría que se obtiene como parámetro y una vez encontrado se procede recorrer la lista de fechas, recorriendo a su vez cada árbol hasta encontrar a la persona y luego agrega dicho pecado a la lista de pecados de la persona.
* **void BuenasAcciones(fec \*\*q, ctg \*p):** Este proceso se encarga de agregar una bondad o buena accion en la lista de bondades de cada persona, siempre y cuando la bondad se encuentre en la lista de categorías, para ello se pide al usuario el nombre de la bondad, asi como el código de la persona, acto seguido se busca dicha bondad en las listas de acciones de la lista de categoría que se obtiene como parámetro y una vez encontrado se procede recorrer la lista de fechas, recorriendo a su vez cada árbol hasta encontrar a la persona y por ultimo luego agrega bondad a la lista de bondades de la persona.
* **void Arrepentimientos(fec \*\*p):** Este proceso busca elimina un pecado de la lista de pecados de la persona y agregarlo a una lista de expiación, para ello se pide al usuario el código de la persona y el nombre del pecado , luego se recorre la lista de fechas recibida como parámetro, y simultáneamente los árboles de cada nodo hasta encontrar a la persona, luego se busca el pecado de la lista de pecados se copia en la lista de contriciones o arrepentimientos y luego se vuelve a armar la lista de pecados sin tomar en cuenta el pecado expiado.
* **void CalcularIndices(fec \*\*p, int código):** Este proceso se encarga de calcular los índices de bondad, maldad, expiación y comportamiento de cada persona en el árbol de la vida. Para ello recorre la lista de fecha obtenida por parámetro y a su vez cada árbol contenido en cada nodo de fecha en busca del código igualmente obtenido por parámetro y una vez encontrado se recalculan los índices mencionados anteriormente.
* **void Perdon(purg \*\*p, des \*\*q):** Este proceso se encarga de mover al cielo al cielo a las personas que cumplieron su condena en la lista de purgatorio para ello se pide al usuario ingresar el año actual, luego se calcula mientras se recorre la lista de purgatorio cuales usuarios han cumplido su condena, si la cumplieron son movidos a la lista de cielo sino se van agregando a una lista temporal que sustituirá al purgatorio.
* **void MostrarComportamiento(fec \*p):** Este proceso se encarga de mostrar por pantalla los pecados u buenas acciones realizadas por una persona para ello se pide al usuario ingresar el código de la persona a consultar y se recorre en la lista de fechas obtenida como parámetro con sus respectivos árboles y una vez encontrada la persona, dependiendo de la selección del usuario se mostrara sus buenas acciones, malas acciones y arrepentimientos.
* **void MostrarDatos(fec \*p):** Este proceso permite visualizar en pantalla la información general de una persona. Para ello se pide al usuario el código de la persona, se recorre la lista de fechas obtenida por parámetros con sus respectivos árboles y una vez encontrado el código se muestran los datos en el nodo.
* **void mostrarFechas(fec \*p):** Este proceso permite visualizar en pantalla lista de fechas. Para ello se recorre la lista de fechas obtenida por parámetros mientras se escriben los datos en cada nodo.
* **void mostrararbol(fec \*p):** Este proceso tiene como objetivo dibujar el árbol perteneciente a los nacimientos de cada fecha en la pantalla de manera horizontal. Este proceso recibe como parámetro una lista de fechas. Este proceso muestra cada nodo del árbol completamente balanceado, siendo el dato de un nodo el código con el que se identifica a una persona en el sistema.

Se utiliza un auxiliar de tipo fecha y dos auxiliares de tipo nacimiento además de tres variables enteras correspondientes a **dia, mes** y **anio**. Se pide que se ingrese la fecha a consultar y una vez ingresada se recorre la lista de fechas en busca de la fecha que coincida con los datos ingresados y luego se llama a los procesos básicos de árboles **InOrden**, **PostOrden** y **PreOrden** y se escribe por pantalla los datos del árbol en dichos órdenes, luego se llama el proceso también básico de dibujar que imprime por pantalla el árbol completamente balanceado.

* **void muerte(fec \*p):** Este proceso se encarga de llamar a la rutina básica de árboles **Borrar**, para ello se ingresa el código de la persona que se quiere eliminar y se busca en la lista de fechas obtenidas por parámetros con sus respectivos árboles hasta encontrar a la persona y pasarla a dicho procedimiento.
* **des \*insertarDestino(des \*p, nac \*auxn):** Esta función tiene como objetivo crear un nuevo nodo en alguna de las listas de destino (primerciel, primerinf).
* **purg \*insertarPurgatorio(purg \*p, nac \*auxn):** Esta función tiene como objetivo ingresar a las personas que ya no se encuentran en el árbol de la vida, en el purgatorio para ello se crea un nuevo nodo en la lista de purgatorio y se pide al usuario el año de muerte y la penitencia a cumplir.
* **void mostrarDestino(des \*p):** Este proceso tiene como objetivo mostrar la lista de destino dependiendo de los parámetros recibidos (primerciel o primerinf), en el caso de nuestro programa muestra la lista de cielo o infierno, para ello se recorre dicha lista obtenida y al mismo tiempo se muestra la información contenida en cada nodo.
* **void mostrarPurgatorio(purg \*p):** Este proceso tiene como objetivo mostrar la lista de purgatorio obtenida por parámetro para ello se recorre la lista y se muestra por pantalla la información contenida en cada nodo.
* **void InsertarDemonio(nac \*\*d, des \*\*auxd):** Este proceso se encarga de extraer a las personas que cumplan con cierto requisito en la lista de infierno, y asignarlas al árbol de Demonios mostrando en el mismo el código de cada persona; tanto la lista como el árbol son obtenidos como parámetros. Para realizar esto primero se pide una clave al usuario para lograr acceder a este proceso, luego se recorre la lista del infierno en busca de los nodos (personas) que cumplan el requisito de comportamiento, establecido en el mismo proceso por el usuario (San Pedro). Luego se insertan dichos nodos mediante la rutina básica de árboles **Insertar**  en el árbol de demonios; aquellos que no cumplan el requisito se irán guardando en una lista auxiliar que actualizara la lista de infierno sin los nodos faltantes.
* **void InsertarAngel(nac \*\*a, des \*\*auxd):**Este proceso se encarga de extraer a las personas que cumplan con cierto requisito en la lista de cielo, y las asigna en el árbol de Ángeles mostrando en éste el comportamiento de cada persona en lugar de código; tanto la lista del cielo como el árbol son recibidos como parámetros. Para lograr lo anterior primero se recorre la lista del cielo en busca de las personas que cumplan con el requisito asignado por el usuario (San Pedro), los usuarios que cumplan el requisito son insertados en el árbol de Ángeles por medio de la rutina básica de árboles **Insertar**, mientras que aquellos que no cumplen el requisito se irán guardando en una lista auxiliar que actualizara a la lista de cielo recibida.