UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Análisis de algoritmos y estructura de datos

Laboratorio 1 – Veterinaria

Alberto Rodríguez Z.

Profesor:	Pablo Schwarzenberg Riveros
Ayudantes:	Javiera Torres
	Javiera Sáez
	Diego Opazo

Santiago - Chile

1-2018

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

CAPÍTULO 1. INTRODUCCIÓN

Dos de los contenidos dentro del curso Algoritmos y Estructuras de Datos son los arreglos y las listas enlazadas. Dos estructuras de datos que nos ayudan a organizar elementos en un cierto orden para poder trabajar con ellos.

Este laboratorio cuenta con una veterinaria llamada el “Bulto Feliz”, que guarda la información de las mascotas de sus pacientes en registros y estos son almacenados en un archivo de texto. Se tiene que implementar un programa que permita buscar y modificar información de una mascota en específico, como también agregar y eliminar registro de mascotas en el sistema. Además, se ingresa un archivo de texto llamado “Bultos.in” y el programa debe ser capaz de cargar su información. Finalmente se crea un archivo llamada “Bultos.out” donde contiene todos los registros con las mascotas que tienen que ser atendidas en el mes determinado.

Para lograrlo se utilizarán las estructuras anteriormente mencionadas, se trabajará con el lenguaje de programación C, específicamente con ANSI C (estándar de lenguaje publicado por el Instituto Nacional Estadounidense de Estándares). Los algoritmos se trabajan con estas estructuras cada una por separado sin mezclarse.

A continuación, se mencionarán los objetivos de este informe junto a la descripción del problema y cómo se abordó para solucionarlo.

Objetivos

1.1 Generales:

El propósito central de este proyecto es la implementación de arreglos y listas enlazadas y trabajar con ellos en la solución de la veterinaria.

1.2 Específicos:

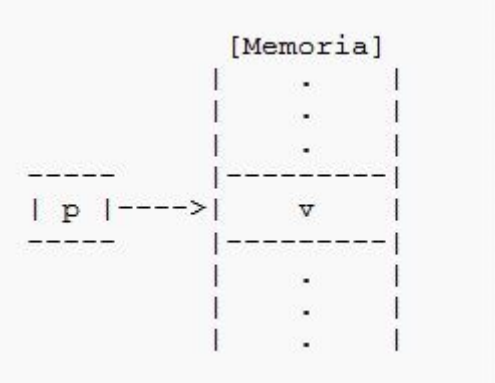
1. Aplicación de conceptos clave de arreglos y listas enlazas aprendidos en clases, para poder trabajarlos con los archivos de la veterinaria
2. Estudiar los algoritmos desde variados aspectos, desde la representación a través de los tipos de datos abstractos hasta los tiempos de ejecución de aquellos algoritmos.
3. Elaborar un programa que respete los estándares ANSI C con el fin de crear un programa eficiente, óptimo y estable.

CAPÍTULO 2. DESCRIPCIÓN DE LA SOLUCIÓN

A continuación, se explicarán las herramientas, técnicas y algoritmos empleados para la elaboración de una solución además del material investigado, el cual sirvió como base para resolver el problema

2.1 Marco teórico:

Punteros:



Para entender esta definición se necesita entender que es una variable estática, la cual significa que a un elemento le asignas una posición fija de memoria de un tamaño previamente definido. Un puntero, en cambio es una variable estática cuyo contenido es una posición de memoria y se puede apuntar a otra posición de memoria y así sucesivamente.

Fig 2.1 Almacenamiento en memoria de una variable y un puntero apuntando a ella

Arreglos:

Se pueden definir como una colección de variables que se referencian por un nombre común, en cada posición del arreglo este contenido un elemento en específico, por lo que el arreglo se puede recorrer mediante un índice, donde la posición más baja corresponde al primer elemento y la posición mas alta al último elemento. Existen dos tipos de arreglos: los estáticos y los dinámicos.

Los arreglos estáticos almacenan el arreglo en una posición fija de materia, por lo que se debe indicar cual es el tamaño de dicho arreglo.

Los arreglos dinámicos se almacenan los arreglos asignando memoria que puede ser modificada en el transcurso de la ejecución, para declarar el arreglo se utilizan punteros, aunque para trabajar en ellos se utiliza la indexación de los arreglos dinámicos.

Estructuras:

Datos complejos que están conformados por dos o más datos de distintos tipos, estos datos compuestos en C se denominan por medio de la palabra "struct" y "typedef".

```
//Declaración de la estructura cd:
struct cd
{
    char titulo[30];
    char artista[25];
    int nro_canciones;
    float precio;
    char fecha_compra[10];
};
```

Fig 2.1.1 Ejemplo de estructura en lenguaje C.

Listas Enlazadas:

Una de las estructuras de datos fundamentales, consiste en una secuencia de nodos, en los que se guardan datos, se implementan mediante punteros, donde cada puntero apunta al siguiente nodo y el ltimo nodo tiene un puntero que apunta a nulo.

Se pide memoria para cada nodo de la lista, y no necesariamente estos nodos estarán en posiciones continuas de la memoria. Por lo que se debe indicar la posición de memoria a la que apunte cada puntero, así como para agregar o eliminar nodos a la lista.

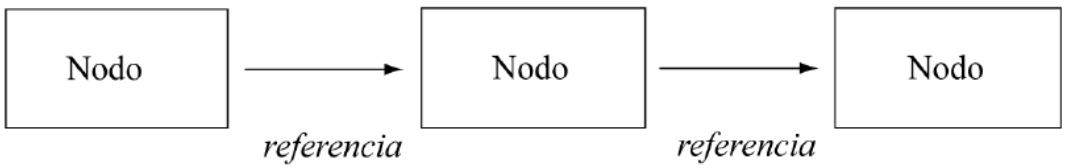


Fig 2.1.2 Representación simple de lista enlazada

2.2 Herramientas y Técnicas:

Para la solución con arreglos, se crea una estructura donde se contiene todos los datos de un registro, esta estructura se trabaja como un arreglo, ósea se guardará en cada posición los datos de un registro determinado, como se utiliza el TDA de arreglos y todas sus operaciones correspondientes, se define el tamaño del arreglo y los de datos contenidos en cada estructura de cada posición.

```
/*ESTRUCTURA DEL PROGRAMA */

typedef struct Ficha{
    char nombre[31],apellidop[31],apellidom[31],nomMascota[21],especie[21],edad[4],telefono[21],atenciones[4],vacunas[4],fecha[11];
    int mes;
}fichas;

fichas *fich;
```

Fig 2.2.1 Estructura del programa con arreglos.

Para la solución con listas enlazadas, también se crea una estructura donde contiene todos los datos de un registro, esta estructura se trabaja como un nodo de la lista enlazada, cada estructura tiene un puntero a la siguiente estructura (nodo). Como se utiliza el TDA de listas y todas sus operaciones correspondientes, se define el nodo inicial de la lista como el nodo final, con ello se puede trabajar cada nodo de la lista.

```
/*ESTRUCTURA DEL PROGRAMA*/

typedef struct Registro{
    char nombre[31],apellidop[31],apellidom[31],nomMascota[21],especie[21],edad[4],telefono[21],atenciones[4],vacunas[4],fecha[11];
    int mes;
    struct Registro *siguiente;
}Ficha;
```

Fig 2.2.2 Estructura del programa con listas enlazadas.

Otra técnica importante fue como trabajar con el tipo de dato string dinámico. En el lenguaje C trabajar con string dinámicos tiene sus ciertas funciones que operan sobre los strings, algunas de las funciones utilizadas en el programa fueron:

-char *strcpy(char *cad1,char *cad2): copia el contenido de cad2 en cad, devolviendo cad1

-char *strtok(char *cad, “dato”): guarda en otro string los carácteres de un string hasta el dato indicado, osea si el string es strtok(“Hola Mundo”, “ “) guardará hasta el espacio

-char *strcat(char *cad1,char *cad2): añade el string cad2 en cad1 y devuelve cad1 con cad2 unido a el.

-char * strcmp(char *cad1,char *cad2): compara dos strings, de ser iguales devolverá un 0

2.3 Algoritmos y estructuras de datos:

Solución con Arreglos:

Algoritmos:

- *void leerTexto(int numero):*

Función que lee el archivo ingresado por el usuario en este caso “Bultos.in”. La función cuando lee una línea guarda cada información de ella en la posición del arreglo, hasta crear un arreglo con un determinado tamaño.

Tiempo de ejecución: $T(n) = 2(n + c) + c$
Orden de complejidad: $O(n)$.

- *void agregar(int numero):*

Función que agrega un registro al archivo “Bultos.in” para este después ser leído en la función leerTexto, este registro se construye con la una unión de strings mediante la función strcat() creando un string con toda la información.
Tiempo de ejecución: $T(n) = c$
Orden de complejidad: $O(1)$.

- *void eliminar(int tamano):*

Función que elimina un registro indicado por él usuario. Para eliminarla vuelve a crear el archivo “Bultos.in” línea por línea con la excepción de no agregar el registro indicado en el archivo.
Tiempo de ejecución: $T(n) = 2(n+c) +c$
Orden de complejidad: $O(n)$.

- *void modificar(int tamano):*

Función que modifica algún dato de una estructura del arreglo, para ello el usuario debe indicar que registro desea modificar, después indicar que dato de esa posición del arreglo desea modificar. La función devolverá el arreglo con el dato modificado (no se crea un nuevo archivo).
Tiempo de ejecución: $T(n) = n+c$
Orden de complejidad: $O(n)$.

- *void buscar(int tamano):*

Función que buscar si el string ingresado por el usuario se encuentra en el arreglo. El usuario solamente puede buscar las características: nombre del cliente, nombre de la mascota y fecha del próximo control. Si el string se encuentra en algún registro, la función devolverá el registro en donde se encuentra (en caso de ser mas devolverá la esa cantidad), si no se encuentra dirá que el registro no se encuentra. Para comparar los string se usa la función strcmp.

Tiempo de ejecución: T(n) = n+c
Orden de complejidad: O(n).

- *void ImprimirArchivo(int tamaño):*

Función que crea el archivo "Bultos.out" y escribe cada registro según el mes que tenga en la fecha de control, el archivo de salida contiene los 12 meses del año, y guarda cada registro según su mes.

Tiempo de ejecución: T(n) = 12(n+c) + c
Orden de complejidad: O(n).

Solución con listas:

Algoritmos:

- *void leerTexto( )*

Función que lee el archivo ingresado por el usuario en este caso "Bultos.in". La función para guardar la información de una línea crea un nodo, y así va formando la lista enlazadas con los nodos, el tamaño de la lista dependerá de cuantas líneas tenga el archivo.

Tiempo de ejecución: T(n) = (n+c) + (nlog(n)) + c
Orden de complejidad: O(nlog(n)).

- *void agregar(Ficha *inicio,int tamaño, Ficha *ultimo)*

Función que agrega un nodo al final de la lista, y este nodo se convierte en el último nodo, dentro del nodo están los datos del nuevo registro ingresados por el usuario.

Tiempo de ejecución T(n) = (c)
Orden de complejidad: O(1).

- *void eliminar(Ficha *inicio,int tamaño,Ficha *ultimo)*

Función que elimina algún nodo de la lista, este nodo es indicado por el usuario. El puntero del nodo anterior al eliminado ahora apuntará al siguiente del nodo eliminado y el nodo eliminado será liberado de la memoria. Si el nodo eliminado es el primero de la lista, ahora el nodo inicial será el segundo de la lista.

Tiempo de ejecución: $T(n) = (n+c) + c$
Orden de complejidad: $O(n)$.

- *void modificar(Ficha *inicio)*

Función que modifica algún dato del nodo seleccionado por el usuario, retorna la lista enlazada con el dato del nodo modificado.

Tiempo de ejecución: $T(n) = 2(n+c) + c$
Orden de complejidad: $O(n)$.

- *void buscar(Ficha *inicio)*

Función que busca el string ingresado por el usuario en la lista enlazada. Solo puede buscar en los nodos: el nombre del cliente, nombre de la mascota y la fecha del próximo control. Esta función retorna lo mismo que la función buscar en arreglos.

Tiempo de ejecución: $T(n) = 4(n+c) + c$
Orden de complejidad: $O(n)$.

- *void ImprimirArchivo(Ficha *inicio)*

Función que crea el archivo “Bultos.out” donde ingresa los datos de cada nodo según el mes que le corresponda.

Tiempo de ejecución: $T(n) = 12(n+c) + c$
Orden de complejidad: $O(n)$.

2.4 ANÁLISIS DE LOS RESULTADOS

Al trabajar en el lenguaje de programación C, una de sus características es el manejo de memoria, la cual para propósitos de este laboratorio nos permite realizar un programa que aparte de resolver el problema, sea óptimo en cuanto a tiempos de ejecución, para eso se analizó a través de fórmulas que permiten calcular cuánto se demora cada ciclo. Estas fórmulas nos dan un parámetro para poder determinar la eficiencia de nuestro programa.

Una de las funciones más importantes del programa es la de leer texto, que donde más se demora el programa en funcionar, al tener que leer cada línea y guardar cada dato en las estructuras (esto es igual para arreglos y listas). Al ingresar el archivo prueba para este laboratorio, el cual contiene 2 millones de registros, se calculó que el tiempo de demora en cada programa es de: 1 hora la de arreglos y 1 hora y media la de listas. Por lo cual para un archivo con gran cantidad de registro el programa no es eficiente.

Después de leer el texto las otras funciones se demoran un tiempo razonable en realizar las funciones no se demorar más de 5 minutos.

Retomando los objetivos, podemos ver un logro satisfactorio, implementando correctamente un programa que aborde lo que se pide el enunciado, ya que con las pruebas realizadas el programa es capaz de agregar, eliminar, modificar y buscar registros en los dos algoritmos.

Dentro de los objetivos específicos podemos ver cómo logramos estudiar los conceptos clave y aplicar aquellos de manera que se construyó un programa que solucione el problema correctamente. Este programa también cumple nuestro objetivo de respetar los estándares ANSI C compilando en entornos de Windows como en Linux, siendo un programa óptimo y estable para recibir al usuario y permitirle el uso de las herramientas creadas.

CAPÍTULO 3. CONCLUSIÓN

Como se mencionó en los análisis, el proyecto tuvo un grado de logro bastante alto, ya que se logró estudiar en profundidad y con tiempo el TDA de arreglos y de listas enlazadas, para así poder plantear los algoritmos necesarios para su operación y de manera exitosa crear un programa que resuelve todos los problemas planteados en el enunciado tal como se pidieron.

Al comparar los dos algoritmos, el algoritmo de listas enlazadas es más fácil trabajar en él, es decir no es necesario crear de nuevo el archivo, como en el de arreglos que se crea un archivo cada vez que se elimina o agrega un registro. Aunque en eficiencia se puede ver que el algoritmo de arreglos es mas eficiente ya que demora menos en leer los archivos.

Gracias al presente laboratorio fue posible aprender la implementación de una estructura en arreglos y listas, como trabajar en sus TDA y sus operaciones respectivas. Con respecto al tiempo de ejecución podemos analizar que a pesar de no ser muy eficiente el programa es capaz realizar todas las instrucciones que se le piden si riesgo de errores o fallos en el mismo.

A futuro se espera poder realizar un manual de usuario más completo, ya que si bien se aborda todo lo especificado, siempre podría detallarse mejor las instrucciones de compilado y uso del programa junto a sus limitaciones.

REFERENCIAS

Empieza A Programar (2013). *Videos para aprender a programar en C*. (Chile) Recuperado de: https://www.youtube.com/watch?v=9idgIGmQvAQ&list=PLw8RQJQ8K1ySN6bVHYEpDoh-CKVkL_uOF

Jacqueline Köhler Casasempere (2017). *Apuntes de la asignatura Análisis de Algoritmos y Estructuras de Datos*. Chile, Departamento de ingeniería informática (DINF), Usach.