



UCSC

Estructuras de Datos

Clínica odontológica al estilo C++

Integrantes : Esteban Cadiz Leyton
Daniel Aravena Contreras
Cristian Vasquez Baeza

Docente/s : Braulio Quiero Hernandez
Hector Ayala Peñailillo

Curso : Estructuras de Datos

Fecha : 25 de junio de 2024

Índice de Contenidos

Introducción.....	3
Propósito.....	3
Objetivos.....	3
Organización del documento.....	4
Definición del problema.....	6
Clinica Odontologica.....	6
Descripción del problema.....	6
Interesados.....	6
Funciones del Producto.....	7
Elaboración.....	10
Análisis.....	10
Diagramas de Nodos.....	11
Estructuras.....	11
Organización del código fuente.....	12
Construcción.....	13
Funcionalidades de la Aplicación.....	13
Requisitos Futuros.....	13
Código Fuente.....	14
Por menores.....	16
Conclusiones.....	17

Introducción

Actualmente, las clínicas odontológicas son cada vez más comunes, y por ende, son cada vez más comunes los problemas para organizar eficientemente las fichas médicas de los pacientes que llegan, al igual que gestionar los turnos y las distintas prioridades. Una clínica que no posea las herramientas necesarias para afrontar esos desafíos, tendrá una existencia muy caótica, llena de problemas y quejas de sus pacientes.

El presente proyecto, está dirigido a eso, a poder solucionar aquellos problemas, o mejor aún, prevenirlos totalmente, así evitando la mala fama que puede generar. Esta aplicación sería un software hecho en C++, usando todos nuestros conocimientos en estructuras y tipos abstractos de datos.

Gracias a esto, las clínicas serán capaces de organizar de manera eficiente todas las fichas de cada paciente, así podrán ver sus datos, recetas, condiciones, y todo lo que sea necesario para los dentistas a la hora de trabajar y evaluar a sus pacientes.

Esto no solo beneficia a los trabajadores, sino también a los pacientes, puesto que lo que ellos buscan, es un gran servicio a su disposición, y mientras más facilidades tenga el negocio, más felices serán los pacientes, y por ende, mejor fama tendrá la clínica, asegurando un futuro próspero y duradero.

Propósito

Explicar de la manera más detallada posible, las distintas funciones que tendrá este programa, quienes son aquellos a los que les interesa una en específico, y el porqué. Planteamos el problema que puede tener una clínica odontológica en sus inicios, y lo eficiente que puede ser la automatización de sus fichas y turnos, en lugar de invertir tiempo y esfuerzo en que se hagan manualmente.

Objetivos

En general, la aplicación es para que el equipo de trabajo pueda gestionar eficazmente las distintas obligaciones de una clínica, podemos particionar esto en los siguientes puntos:

- Diseñar e implementar un sistema donde se facilite al odontólogo y su equipo de trabajo, el acceso a las fichas de los pacientes, para una mayor eficiencia y organización, previniendo problemas a futuro.
- Crear un sistema que pueda priorizar eficazmente las llegadas de los pacientes en relación al motivo de que estén visitando la clínica, atendiendo primero aquellos que vayan por una emergencia.
- Hacer que la interfaz de usuario sea sencilla y entendible para el equipo de trabajo, facilitando su comprensión y evitando que modifiquen algo que no deben.

Organización del documento

En la introducción, se habla sobre que las clínicas odontológicas son cada vez más comunes, y por ende, son cada vez más comunes los problemas para organizar eficientemente las fichas médicas de los pacientes que llegan, al igual que gestionar los turnos y las distintas prioridades. El proyecto, está dirigido a eso, a poder solucionar aquellos problemas, o mejor aún, prevenirlos totalmente, así evitando la mala fama que puede generar.

La sección de Organización del documento, describe la estructura y el contenido que se encontrará en cada sección, facilitando al lector la navegación y comprensión del texto. Gracias a esto, personas que no comprenden mucho del tema, pueden entender el funcionamiento y el porqué de las distintas decisiones.

En los objetivos, se dice que el programa permitirá al equipo de trabajo de una clínica odontológica gestionar de forma eficaz sus labores mediante un sistema que facilitará el acceso rápido y organizado a las fichas de los pacientes, priorizando las llegadas según la urgencia, y cuenta con una interfaz de usuario sencilla y comprensible. Esto mejorará la eficiencia y prevendrá problemas futuros.

En la organización del documento, se dará un breve resumen de cada sección del mismo, para que se pueda dar una idea de cómo decidimos que era mejor ordenar cada parte, y de esta forma, quien lo lea y no esté familiarizado con el tema, no tenga mucho problema para entender lo que se plantea.

La sección de Definición del problema plantea las principales dificultades que enfrenta una clínica odontológica, dando detalladamente todas las necesidades básicas para su correcto funcionamiento. Se explican los métodos de trabajo actuales y cómo la falta de organización puede afectar negativamente la reputación e ingresos de la clínica.

En la sección de Interesados, se identifican los miembros del equipo que estarán involucrados en las diferentes tareas de la clínica, describiendo sus roles y cómo cada uno se beneficia de las distintas funciones del programa propuesto. Esto incluye desde la secretaría hasta los dentistas, pasando por los propios pacientes.

En las funciones, se dan detalles sobre las funciones del programa, separados por dos módulos, uno correspondiente a una Fila, y la otra, a la Tabla hash. Cada módulo contiene dos funciones dentro que permiten su correcto funcionamiento, siendo “Consultar ficha del paciente” y “Manipular fichas”, las que corresponden al Módulo tabla hash, y “Asignar prioridad” junto a “Siguiente turno”, las funciones del Módulo Fila de espera.

La sección de Elaboración describe el proceso de desarrollo del programa, incluyendo la metodología y las herramientas utilizadas. Aquí se detallan las etapas del desarrollo, desde la planificación hasta la implementación, asegurando que el producto final cumpla con los objetivos propuestos.

En el Análisis, se examinan los datos y estructuras de la aplicación, explicando cómo se manejan internamente para asegurar que sea eficiente. Se incluye un análisis detallado de las estructuras de datos y su implementación en el sistema.

Los Diagramas de Nodos presentan de forma visual las funciones del programa, mostrando la relación entre diferentes componentes y cómo interactúan entre sí. Esto facilita la comprensión del sistema tanto para desarrolladores como para otros interesados.

La sección de Estructuras, se detalla las estructuras de datos empleadas en el desarrollo del sistema, explicando su diseño y cómo contribuyen a la eficiencia del programa en la gestión de las fichas y turnos de los pacientes, junto a los motivos del porqué lo decidimos.

La Construcción del sistema se aborda en una sección dedicada, explicando los pasos seguidos para construir el programa, desde la codificación hasta las pruebas, de esta forma, asegurando que todas las funcionalidades trabajen de forma correcta.

En Funcionalidades, se enumeran y describen las diferentes características y capacidades del programa final, explicando cómo cada funcionalidad se integra en el sistema de manera uniforme, para mejorar la eficiencia y organización de la clínica.

En requisitos futuros, analizamos las posibles mejoras y optimizaciones que podríamos incluir al programa en caso de una actualización a futuro, dependiendo de las necesidades que pueda tener la clínica en un futuro, añadiendo lo que se pida.

En código fuente, se muestran y explican fragmentos del código que forma todo el software programado, en el lenguaje C + +, para que haya transparencia sobre lo que se hizo en el mismo. En caso de necesitar otros desarrolladores que modifiquen el programa, estará a su alcance para entender mejor el funcionamiento del mismo.

En la sección de Por menores, mostramos los diferentes problemas que se presentaron a lo largo del desarrollo del programa, los cuales aunque al inicio no se presentaban con frecuencia, empezaron a surgir a medida que avanzábamos con el programa, siendo bastante diversos y en su mayoría, rebuscados de solucionar, pero gracias a los que aprendimos nuevas maneras de implementar diferentes opciones que ofrece el lenguaje de programación C + +.

En la conclusión se analizará cómo el código, tras un proceso complejo de desarrollo y depuración, se logra compilar correctamente y funcionar con éxito a través de la terminal. Se destacó el cumplimiento de los objetivos iniciales, al igual que se mencionará la importancia de la investigación adicional para superar los desafíos encontrados, lo que ha contribuido a una mayor experiencia en el uso del lenguaje de programación C + +.

Definición del problema

Clinica Odontologica

La principal función de una clínica odontológica es proveer servicios y atenciones de salud dental a quienes asisten. Hay varias necesidades básicas que se deben cumplir para su correcto funcionamiento. Ya que está centrada en la atención de pacientes, la organización y atención de estos toma prioridad. Al tener generalmente, una gran cantidad de estos, es de suma importancia que no haya confusiones ni errores entre personas. Dentro del proyecto se va a abordar este punto.

Descripción del problema

Al asistir a una clínica odontológica, como paciente lo que se espera es poder recibir atención específica. Cada uno llega con problemas, necesidades e historial particular, los cuales deben de ser conocidos por los dentistas para proveer el tratamiento necesario. También las consultas pueden variar en urgencia, es más prioritario atender a alguien que llegue con una herida abierta que a alguien que pueda tener una hora de control agendada que se pueda desplazar. Todos estos factores mencionados se complejizan rápidamente cuando la cantidad de pacientes comienza a aumentar. Es necesario disponer de una manera de organizar los horarios de atención de una clínica, desde el momento que entra una persona hasta que se retira.

Se va a programar una aplicación donde esta problemática se solucione. Para trabajar esto, se va a preparar un sistema donde se organice la atención de las personas mediante van llegando. Una vez estén asignados, por prioridad de urgencia y llegada, se preparará una ficha en un sistema que sea de fácil acceso, tanto para el área de secretaría como los dentistas, para que el tratamiento sea de la mayor calidad posible. Seguido de esto, se dispondrá de un sistema que permita agendar la siguiente cita, tomando en cuenta los horarios disponibles de los pacientes y los de atención de los dentistas.

Interesados

Nombre	Descripción	Interés
Secretaría	Personas que agendan horas de pacientes y también los horarios de los dentistas. Necesita una manera de ordenar a los pacientes que van llegando para atenderse. Tiene que	Tener turno definido de cada paciente para atenderlo. Tener ficha dental de cada paciente

	separarse entre Urgencias y horas ya agendadas basándose en prioridades.	para poder identificar cada caso y asignar dentista.
Pacientes	Personas que necesitan pedir una hora dental, ya sea de urgencia o no.	Solicitud de hora dental. Asignación de turno al llegar.
Dentistas	Personas que revisan a los pacientes y redactan su ficha dental con las necesidades de estos. Necesitan revisar al paciente para atender que problema dental específico a este.	Tomar registro de lo que le sucede al paciente particularmente.

Funciones del Producto

Módulo 1: Tabla Hash:

Consultar ficha paciente	
Personal Involucrado	Dentista: Consultar la ficha de un paciente particular rápidamente para saber qué atención brindar. Secretaría: Consultar ficha de un paciente para saber que doctor asignar y que horarios poder ofrecer.
Precondiciones	Tiene ya haber asistido el paciente y la ficha de este haber sido ingresada.
Entrada	Ingreso nombre de paciente
Flujo Básico	<ul style="list-style-type: none"> - Se ingresa nombre de paciente - Nombre es pasado mediante una función hash para generar clave - Se busca lista que coincida con la clave generada - Se recorre lista revisando nombres y retorna primera ficha donde sean iguales
Extensiones	Caso de excepción: una colisión <ul style="list-style-type: none"> - Si dos nombres tienen la misma clave, se continúa recorriendo la lista asignada hasta encontrar nombres coincidentes
Salida	Se devuelven datos de la ficha a quien consulta.

Insertar/modificar/borrar ficha paciente	
Personal Involucrado	<p>Dentista: Se encarga de enviar los datos de la ficha del paciente al sistema de secretaria para que esta tenga un acceso total.</p> <p>Secretaría: Se encarga de insertar/modificar/borrar la ficha de un paciente en caso de que el dentista se lo pida</p>
Precondiciones	<p>Una vez finalizada la atención dental, el dentista le envía los resultados de la ficha del paciente al sistema de la Secretaría para que esta reciba acceso a modificaciones</p> <p>Si el dentista pide modificar la ficha (en caso de que haya un nuevo caso que en la atención que no se haya visto) la secretaria modificará la ficha.</p> <p>Si el dentista pide borrar la ficha (porque el problema dental ya está solucionado) la secretaria borrará la ficha.</p>
Entrada	Datos de la ficha dental
Flujo Básico	<p>Los datos de la ficha son asignados a la secretaria, se define mediante 3 funciones lo que la secretaria decida qué hacer con la ficha, una vez realizada una función, la ficha va a la salida con los datos requeridos.</p>
Extensiones	<p>El sistema no logra distinguir si recibe dos fichas que tengan exactamente los mismos datos.</p> <p>(se quiera modificar/borrar una ficha y se termina modificando/borrando la ficha equivocada)</p>
Salidas	Se devuelven datos de la ficha (ya sea insertada, modificada o borrada).

Módulo 2: Fila:

Asignar prioridad al llegar a la clínica	
Personal Involucrado	Secretaría: define si el caso del paciente se trataría de una urgencia, en caso de que lo fuera, le asigna prioridad de ingreso mediante una hora de urgencia, en caso de que no lo fuera, le asigna una hora como un paciente normal. Paciente: pide una hora en la secretaría, tiene que especificar si su consulta se trata de urgencia o no.
Precondiciones	Que llegue un paciente con una urgencia, como puede ser; perder el diente por un accidente, dolor intenso, absceso dental, etc. Al ocurrir esto, se le dará prioridad sobre los demás pacientes.
Entrada	Caso de urgencia o normal del paciente.
Flujo Básico	-La hora ingresada como caso de urgencia va directamente a ser revisada por un dentista. - la hora ingresada como caso normal se guarda en una lista de espera.
Extensiones	Si llega una urgencia peor a otra, se le asignará como más prioritaria, sin embargo, si es menos "importante", se le asignará un lugar más bajo. Esto queda a juicio de la secretaría o el mismo dentista.
Salidas	Se devuelven fecha y horas asignadas al paciente, junto con la emergencia por la cual fue puesto como prioridad.

Consultar proximo turno	
Personal Involucrado	Secretaría: Consulta el próximo turno para que el siguiente paciente pueda atenderse. Paciente: Recibe atención cuando le toca su turno.
Precondiciones	La secretaria consulta al siguiente turno que sigue en espera, para que los pacientes puedan atenderse de forma ordenada.
Entrada	Turno del paciente siguiente.
Flujo Básico	Agarra el primer turno de las urgencias, si no hay urgencias busca en la fila agendada.
Extensiones	En un colapso de pacientes que se están atendiendo al mismo tiempo, la función tendrá que detenerse hasta que un paciente termine su atención.
Salidas	Se devuelve el próximo turno.

Elaboración

Para solucionar el problema se va a desarrollar una aplicación usando el lenguaje de programación C + + y estructuras de datos propias. Principalmente se va a trabajar con la terminal en esta versión. Dentro de esta sección se abordarán principalmente puntos relacionados con la estructura y funcionalidad del código específico a la aplicación y el problema.

Análisis

Para poder solucionar el problema, se va a trabajar mediante estructuras de datos propias. Se van a utilizar dos filas separadas, una de urgencia y otra de atención normal, para organizar a los pacientes mientras van llegando. La urgencia de la situación se va a dejar a criterio de quien utilice la aplicación. Para el problema de las fichas se va a utilizar una tabla hash para poder mantener un rápido acceso de datos entre los interesados, ya sea para consulta o modificación.

Diagramas de Nodos

Los siguientes diagramas fueron realizados mediante draw.io

Diagrama de módulo de Fila

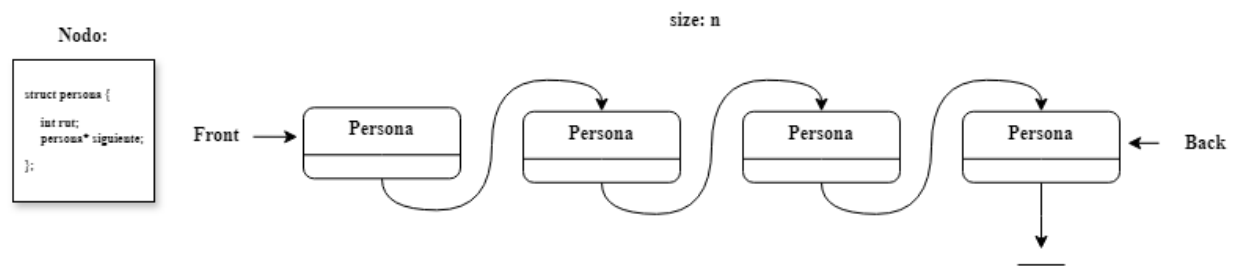


Imagen 1: Diagrama de Fila

Diagrama de módulo de Tabla Hash

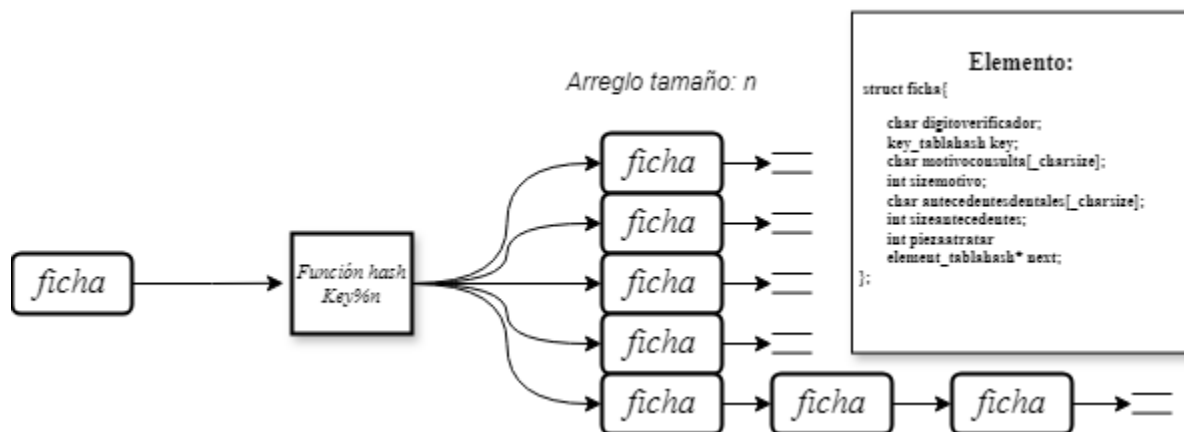


Imagen 2: Diagrama de Tabla Hash.

Estructuras

Fila:

Para las personas usaremos nodos; en el nodo se define a la persona mediante un string que asigne su nombre, y luego se asigna un puntero que haga que la persona apunte a siguiente (es una referencia a otra persona).

Para organizar a las personas usaremos filas; las filas se crearán mediante el nodo "persona", en este caso, cada nodo apunta a next (la siguiente persona que será atendida

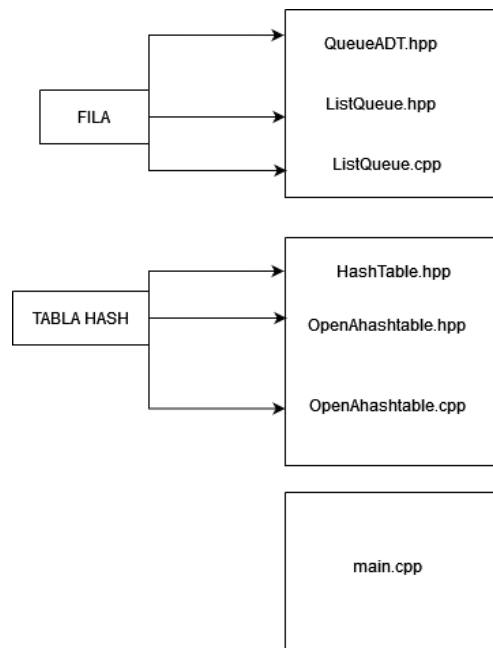
primero). Para esta aplicación se crearán dos filas, una de urgencia y otra de atención normal. La fila de urgencia tendrá que estar vacía para que avance la normal.

Tabla hash:

Para crear las fichas y modificar los datos de esta, usaremos una tabla hash; en la tabla hash se ingresara el nombre del paciente, su rut y su dígito verificador, luego, se ingresará una “llave” que se igualará al rut, y luego se crea un puntero que haga que la ficha apunte a siguiente.

Este puntero se va a utilizar para las colisiones, las que se van a solucionar un encadenamiento. Después de pasar por la función Hash, los datos ingresados se guardaran mediante un arreglo en la posición correspondiente.

Organización del código fuente



Se usarán dos archivos hpp y un archivo cpp para las estructuras, junto a un main.cpp que compilara todos los archivos juntos que será el programa en sí.

Imagen 3: Modularización del código fuente.

Construcción

En esta sección, se detallarán todas las cosas que influyen en la programación del software, un resumen de las funcionalidades del mismo, los requisitos y mejoras que se podrían implementar a futuro, nuestro código fuente, y los principales problemas que surgieron a lo largo del desarrollo, para una futura evaluación.

Funcionalidades de la Aplicación

Módulo de fila:

Nos percatamos de que este sería el método más eficiente y sencillo para implementar una fila de espera, puesto que se comportaría exactamente igual, es decir, el primero en llegar, es también el primero en salir. De esta forma, los pacientes podrán estar tranquilamente sentados, esperando su turno, sin miedo a que alguien se los quite en un despiste, puesto a que funcionará exactamente como una fila cualquiera.

Módulo de tabla hash:


Para el problema de las fichas se decidió resolver usando una tabla hash. Al guardar datos en lugares específicos basados en una llave, esto permite a los interesados acceder con rapidez a los datos. Específicamente se va a usar con fichas dentales, ya que al guardar una gran cantidad de datos, es necesario tener una manera de recuperar los de cada paciente de forma eficaz.

Requisitos Futuros

La principal falta que se produjo es la falta de interfaz gráfica. Esto habría permitido que la utilización de esta herramienta hubiera sido mucho más agradable para un usuario promedio que no tiene experiencia en trabajar con terminales. Por falta de tiempo no se pudo implementar, pero el funcionamiento de la aplicación está al nivel de poder realizar sus funciones.

Código Fuente

A continuación están capturas del código que se encargan de las funciones explicadas:



```
1 void consultaFilas(ListQueue *urgencia, ListQueue *normal) // imprime al paciente que ahora le toca
2 {
3     if (urgencia->empty() == true)
4     { // si no hay urgencia
5         if (normal->empty() == true)
6         { // si no hay nadie en ambas, es decir vacío
7             std::cout << "\n"
8                 << "No hay personas en espera" << "\n";
9         }
10        else
11        { // Caso donde no hay urgencia
12            std::cout << "\n"
13                << "Ahora es el turno de:\n"
14                << (*normal).front() << "\n";
15            (*normal).dequeue();
16        }
17    }
18    else
19    { // Caso donde hay urgencia
20        std::cout << "\n"
21            << "Ahora es el turno de urgencia:\n"
22            << (*urgencia).front() << "\n";
23        (*urgencia).dequeue();
24    }
```

Imagen 4: Función de consultar el próximo turno junto con los métodos de fila utilizados.

El primer código a mostrar es el encargado en la entrada main.cpp de consultar quien sigue en la fila de espera. Para esto usa métodos definidos en la fila. Primero usa el uno llamado “empty()” para asegurarse si hay personas en estas. Revisa cola de urgencia y cola normal. Luego si hay urgencia la hace pasar, pero si está vacía, se le da preferencia a la normal.

Para consultar al paciente de la fila correspondiente, usa el método “front()” para obtener al que llegó primero y luego usa “dequeue()” para sacarlo de la cola y dar paso al siguiente. La función toma a las filas por referencia por estabilidad.



```
1 void ingresofilaespera(ListQueue *urgencia, ListQueue *normal) // funcion que añade a alguien a la fila de espera
2 {
3     elemento_fila rut;
4     int ingresofila;
5
6     std::cout << "Ingrese rut paciente" << "\n";
7
8     std::cin >> rut;
9     fflush(stdin); // limpia buffer
10
11     std::cout << "Ingrese a que fila lo desea enviar" << "\n";
12
13     std::cout << "1. Atencion Normal" << "\n";
14     std::cout << "2. Atencion de Urgencia" << "\n";
15
16     std::cin >> ingresofila;
17     fflush(stdin); // limpia buffer
18     if (ingresofila == 1)
19     {
20         (*normal).enqueue(rut);
21     }
22     else
23         (*urgencia).enqueue(rut);
24 }
```

Imagen 5: Función de ingresar paciente a la fila de espera.

Para ingresar a un paciente a la fila de espera primero se toma por referencia ambas y se pide el ingreso del paciente mediante su rut. Luego con un simple condicional se elige entre una de las dos, el criterio quedando a disposición del usuario, y se usa el método “enqueue(“rut”)” para que ingrese al final de la fila.



```
1 void consultaficha(HashEncadenado *fichasguardadas) // funcion para consultar fichas
2 {
3
4     element_tablahash fichaaux; // ficha auxiliar
5     key_tablahash rutk;
6     std::cout << "Ingrese rut sin digito verificador ni puntos " << "\n";
7     std::cin >> rutk;
8     fflush(stdin); // limpia buffer
9     fichaaux = (*fichasguardadas).find(rutk);
10    if (fichaaux.key != 0)
11    { // caso donde hay ficha
12        printficha(fichaaux);
13    }
14    else
15    { // Caso donde no encuentra ficha es decir find devuelve key=0
16        std::cout << "No se encontro ficha" << "\n";
17    }
18 }
```

Imagen 6: Función de consulta de fichas dentales.

Para la función de consulta se trabaja por referencia por estabilidad principalmente. Se utiliza una ficha auxiliar para que guarde los valores de la ficha entregada por el método de la tabla hash “find(“llave”)”. Este recibe la llave, obtenida del input en este caso, para acceder a la tabla hash y encontrar la ficha específica. Una vez esta operación termina, se imprime en la terminal la ficha. En el caso donde no se encuentra la ficha específica, el “find()” está configurado para entregar una ficha con rut 0, lo que no es posible, por lo que se usa ese parámetro para revisar.

```
1 void nuevaficha(HashEncadenado *fichasguardadas) // funcion para ingresar nueva ficha a guardar
2 {
3     element_tablahash fichaaux;
4     for (int i = 0; i <= _charsize; i++)
5     {
6         fichaaux.antecedentesdentales[i] = ' ';
7         fichaaux.motivoconsulta[i] = ' ';
8     } // for para inicializar arreglos de caracteres a utilizar
9
10    std::string stringaux, stringaux2; // se va a usar dos string aux para usar el getline
11    // se volvio requerimiento usar fflush para limpiar el buffer de entrada al trabajar con strings
12    std::cout << "Ingrese Rut sin digito verificador" << "\n";
13    std::cin >> fichaaux.key;
14    fflush(stdin);
15
16    std::cout << "Ingrese digito verificador" << "\n";
17    std::cin >> fichaaux.digitoverificador;
18    fflush(stdin);
19
20    std::cout << "Ingrese motivo de consulta:" << "\n";
21    getline(std::cin, stringaux); // get line es una funcion de libreria <string>, obtiene toda la linea en vez de una palabra
22    fflush(stdin);
23    fichaaux.sizemotivo = stringaux.length();
24    stringaux.copy(fichaaux.motivoconsulta, fichaaux.sizemotivo, 0); // metodo copy de string transfiere toda la string a un arreglo char
25
26    std::cout << "Ingrese antecedentes dentales" << "\n";
27    getline(std::cin, stringaux2);
28    fflush(stdin);
29    fichaaux.sizeantecedentes = stringaux2.length();
30    stringaux2.copy(fichaaux.antecedentesdentales, fichaaux.sizeantecedentes, 0); // metodo copy de string transfiere toda la string a un arreglo char
31
32    std::cout << "Ingrese pieza a tratar" << "\n";
33    std::cin >> fichaaux.piezaatratar;
34    fflush(stdin);
35
36    (*fichasguardadas).insert(fichaaux);
37 }
```

Imagen 7: Función de insertar ficha nueva.

Esta función se encarga de insertar una nueva ficha en la tabla. Para esto se crea una ficha auxiliar y se consigue cada parámetro por ingreso del usuario. Destacan ambos parámetros donde se guardan arreglos de caracteres, fue necesario crear una string auxiliar para cada uno para usar el método “getline()” que obtiene la frase completa en vez de solo una palabra como lo hace el input normal. Se utilizó el tipo de dato char en vez de string directamente por compatibilidad, ya que este último es una clase hecha específicamente para C++ que requiere de el operador “new”, que opera el constructor, en vez de la función “malloc()” que solo asigna la memoria y es el utilizado en esta asignatura.

Por menores

En el desarrollo de esta aplicación se enfrentaron distintos problemas a los que se les tuvo que buscar soluciones. Al trabajar con fila al inicio se intentó usar el nombre de los pacientes como identificador en vez del rut mediante el uso de strings. Esto causó que el

programa se volviera inestable por tratar de usar la función “malloc” con la string, la que al estar definida como una clase específica de C + + tiene que usar constructor, que se hace mediante el operador “new”.

Al trabajar con tabla hash se enfrentó algo parecido, pero en este era imposible reemplazar la string por un int, por lo que se guardó en una variable auxiliar de tipo string que después se traspasó a un arreglo de char específico. Otro pormenor fue que el programa colapsaba con cualquier entrada que se hacía. Se logró sobrellevar usando la función “fflush(stdin)”, la que limpiaba el buffer de entrada.

El último error grande a mencionar es la necesidad que hubo de pasar por referencia todos los valores en funciones, aunque no se editaran. Esto permitió al programa tener mucha mayor estabilidad. Se cree que fue por el hecho de que es más ligero mover en función una dirección de memoria que una estructura con distintos valores y un gran peso.

Conclusiones

El código se compila de forma correcta y funciona exitosamente a través de la terminal, esto fue una tarea muy compleja, pero gracias a eso, al solucionar los detalles que provocaban fallas dentro del programa, se nos terminó haciendo mucho más fácil continuar con el resto del código, como con el caso del RUT.

Los tres objetivos planteados al inicio del documento, fueron exitosamente cumplidos, logramos ser capaces de facilitar a los miembros del equipo de la clínica, facilidad para manipular las fichas de los pacientes, como agregar, eliminar y modificarlas, también siendo posible consultar en cualquier momento las fichas guardadas de cada paciente, mediante su RUT.

De igual forma, se cumplió el objetivo de implementar una fila que representa el orden de llegada de cada paciente, y en caso de presentar una urgencia, se puede agregar con mayor prioridad a esa persona, atendiendo antes que a los demás. En caso de no haber urgencia, la fila funciona normalmente, es decir, en orden de llegada.

Finalmente, la interfaz es simple de entender, y tiene todas las indicaciones necesarias para guiar de forma eficaz al equipo de trabajo en torno al programa diseñado, siendo posible entender cada una de las secciones del mismo de forma clara.

La parte más interesante de este proyecto, fue tener que investigar más de lo que se nos fue enseñado de C + +, ya que habían errores que no entendíamos y tuvimos que recurrir al manual y algunos foros, y si no encontrábamos solución en esos lugares, arreglarnoslas para solucionar cada problema como mejor se nos ocurriera, teniendo que aplicar nuestro ingenio, y gracias a ello, nos consideramos bastante más experimentados dentro de este lenguaje de programación.