TEC | Tecnológico de Costa Rica

II SEMESTRE

Sistema de administración de colas

Proyecto 0

Instituto Tecnológico de Costa Rica

Campus Tecnológico Local San José

Escuela de Ingeniería en Computación

Bachillerato de Ingeniería en Computación

IC-2001 Estructuras de Datos GR 40

Profesor:

Mauricio Avilés

Integrantes:

David Josué Centeno Araya – 2020103578

Dina Isabel Monge Sandoval -2022117025

Angélica María Díaz Barrios – 2021044256

18 de octubre, 2022

Índice

1. Introducción

Este proyecto nace a partir de una necesidad y de una realidad que ha existido desde hace tiempo atrás, se debe recordar para poder utilizar algún servicio que ofrezca un establecimiento de manera presencial es necesario realizar una fila (ya sea sentados o de pie), en la cual se debe mantener un orden y el primero que esté en esta deberá ser atendido. Es así, como a lo largo de la historia se ha observado que, tanto en instituciones públicas como privadas, las personas tienen qué esperar por un turno para ser atendidas. Por otro lado, también se tiene la situación de que muchas de las personas que necesitan de un servicio no pueden hacer la cola respectiva porque tienen alguna condición especial como, las mujeres en estado de embarazo, una persona con un bebé en brazos, personas con discapacidades físicas y más recientemente pertenecer al grupo del adulto mayor; entonces, estas personas requieren de una atención rápida, por esta razón, siempre en cualquier área de un establecimiento se deben tener en cuenta que existen dos tipos de filas (colas).

Con forme la tecnología avanza, así mismo avanza la sociedad, por esto es posible que se puedan realizar tantas cosas que se hacen de forma presencial como, por ejemplo; realizar un programa que trabaje de manera similar a cualquier establecimiento que contenga diferentes áreas con servicios y en los cuales se necesite realizar una cola para ser atendidos según corresponda. Por esta razón, se ofrece una solución virtual a esta necesidad. En este proyecto se pretende simular lo mencionado anteriormente y con ello facilitar la vida de algunos, gracias a la tecnología.

C++ es el lenguaje de programación en el cual este proyecto será implementado, se tomarán en cuenta la mayoría de las funcionalidades que realizan los funcionarios de una institución y así lograr resolver o minimizar la problemática de las colas/filas de espera, que va más allá de evitar molestias que sufren los clientes, se pretende llevar un orden y evitar la desorganización en la atención de los clientes.

El proyecto consiste en implementar un sistema de administración de colas, el cual va a ser lo suficientemente flexible y configurable como para ser utilizado en diferentes tipos de locales o situaciones; para la realización de este proyecto se debe abordar con sus

respectivas solicitudes las siguientes partes: por un lado, la documentación que desde luego incluye: portada, introducción, presentación y análisis del problema como lo es: la descripción del problema, metodología y análisis crítico, conclusiones, recomendaciones y las referencias de ser necesarias. Por otro lado, la programación, la cual cuenta con varios puntos, tales como: crear áreas y a su vez que estas puedan contener cierta cantidad de ventanillas y en ellas se puedan atender a los diferentes clientes que están en la cola, empezando por los preferenciales y seguidamente por los regulares, por medio de la generación de tiquetes que llevan un orden de entrada y salida. Además, se podrá administrar las diversas áreas y servicios y por último generar estadísticas que se relacionan con los tiquetes y el tiempo.

2. Presentación y análisis del problema

2.1. Descripción del problema

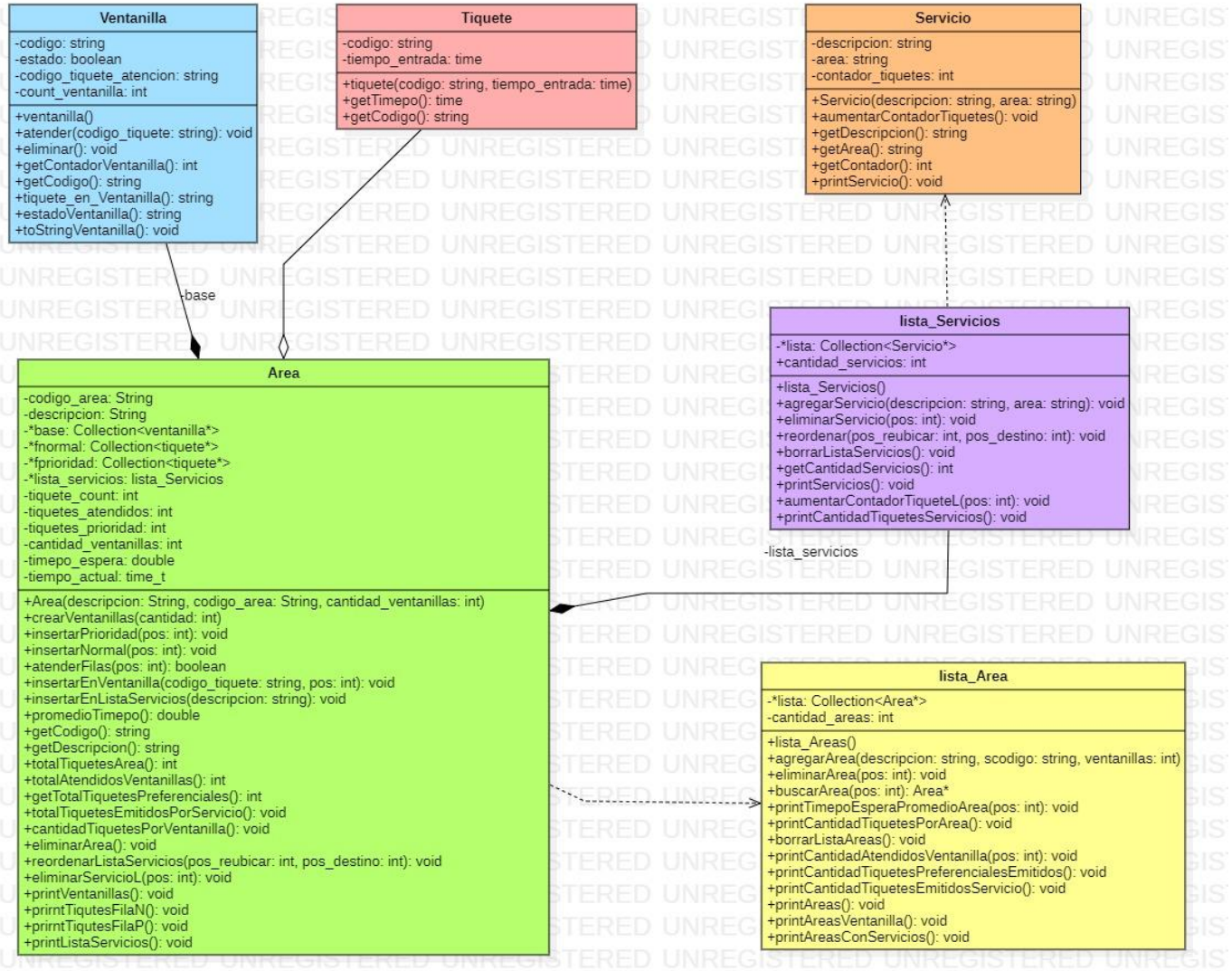
De acuerdo con la situación dada de las molestias que sufren los clientes en diferentes instituciones, especialmente en cuanto al tema de las colas; es que se desea resolver o encontrar una forma/mecanismo estructurado de control de las mismas. Pero a su vez en consecuencia con ellos se deben así mismo resolver una serie de subproblemas, lo que por ende garantice poder disfrutar de una solución virtual (con ella se podría evitar el hecho de exponer a los clientes a largas filas en espacios reducidos o conglomerados y a su vez reducir los tiempos de espera en estos espacios) y desde luego se aproveche de los avances tecnológicos que actualmente ofrecen la posibilidad de acceder a servicios en aplicaciones móviles o por medio del uso de mensajería.

Como parte de los subproblemas que se tienen: se deben contemplar o poder aplicarlo por las diferentes áreas que se posea en la institución o el comercio; es importante que a cada una de ellas se les distinga con un código y una breve descripción. Por otro lado, tenemos cada uno de los servicios ofrecidos en estas áreas, mismo que deben listarse, en este punto la idea es que los tiquetes que se generan estén ligados al área y desde luego al servicio que se solicita. Así mismo, se debe ante todo satisfacer la necesidad de ofrecer una cola preferencial

en ambos casos (por área y servicio). Lo esperado es que en cualquiera de las situaciones el tiquete que se atienda primero corresponda a lo preferencial, y si no hay entonces continuar con los clientes regulares. De tal modo, que se pueda cumplir con uno de los principales requerimientos y es que se le de prioridad a los tiquetes preferenciales en cualquiera de las áreas.

Paralelo a estos subproblemas lo que se pretende es que el sistema generado también sirva para poder facilitar un control estadístico para las empresas, instituciones o comercios en pro de tener información que les permita evaluar rendimientos tales como: tiempo promedio que los clientes esperan para ser atendidos, la cantidad de tiquetes generados por cada área, de igual modo la cantidad de tiquetes que se atendieron por servicio y ventanilla (esto es muy importante pues se puede obtener información valiosa de cada empleado) y desde luego poder contabilizar los clientes preferenciales. Para lograr estas estadísticas se debe tener cuidado de tener un buen método que registre los tiempos de inicio con los tiempos de finalizado y así sea mas fiable los promedios resultantes; sin olvidar que definitivamente el objeto de principal cuidado gira entorno a los tiquetes.

2.2. Metodología



En el siguiente aparatado se va a dar a conocer cada una de las soluciones implementadas para cada uno de los subproblemas.

2.2.1. Área:

En el caso de las áreas dentro del sistema se decidió implementar una serie de atributos de código y descripción los cuales sirvieran como identificadores de cada una de las áreas que fueran agregadas al sistema. Para las ventanillas de cada una de las áreas se decidió implementarlo por medio de una estructura tipo lista (estructura LinkedList) la cual facilitará el acceso a cada una de las ventanillas presentes en el área y el manejo de los

tiquetes por medio de dos colas (estructura LinkedQueue) separadas las cuales contienen tanto a los tiquetes normales como los preferenciales, además tanto para las estructura de la lista como las colas se decidió implementarlo por medio de punteros y su contenido de la misma forma contiene punteros a un objeto en particular. En el manejo de las estadísticas del área se utilizaron distintos contadores para llevar el registro de los tiquetes atendidos, los tiquetes dispensados en un área o la cantidad de tiquetes preferenciales generados.

Para las operaciones de atención de cada una de las colas se utilizó un método el cual recibiera la posición de la ventanilla que el usuario desea que atienda y en primera instancia se validaba dicha posición con la cantidad de ventanillas presentes en el área, posteriormente se utilizó una estructura de bifurcación que validara en primera instancia que la cola de prioridad no estuviera vacía en cuyo caso se sacaría el primer tiquete presente en la cola de prioridad (operación dequeue de la estructura LinkedQueue), el elemento de retorno que en este caso en particular sería un puntero seria almacenado de manera temporal por otro puntero, con la finalidad de poder registrar el tiempo de atención del tiquete en particular y tomar el dato de del código del tiquete el cual sería utilizado para llamar a un método auxiliar que recibe por parámetros tanto el código del tiquete que se sacó de la fila como la posición de la ventanilla que va a atender. Este método auxiliar (nombre del método insertarEnVentanilla(cod,pos)) se encarga de buscar por medio del índice indicado la posición de la ventanilla y una vez localizado llamar al método clase ventanilla (atender(codigo_tiquete)) para introducir el código del tiquete. Para el manejo de tiempo de atención dentro del sistema se utilizo el atributo de la clase tiquete (tipo de dato time_p de la biblioteca ctime) que almacena el tiempo de entrada de un tiquete en particular y en la clase área se implemento un atributo adicional denominado tiempo actual el cual permite registrar la diferencia de tiempo entre el registro del tiquete y su atención en ventanilla, este diferencia se le va sumando al atributo de la clase área tiempo de espera el cual va registrando todas las diferencias de tiempo de un tiquete. En el caso del tiempo de atención promedio de un área en particular se implementó un método que calculara por medio del atributo tiempo de espera entre la cantidad de tiquetes atendidos en el área. Entre los atributos de la clase área se implementó un atributo del tipo lista servicios el cual permite llevar registro de cada uno servicios asociados a un área en particular.

Tanto para las ventanillas, colas y servicios asociados al área se establecieron métodos para desplegar en la terminal cada uno y poder ver su contenido.

Métodos principales de la clase Área		
Nombre	**Parámetros**	**Operación**
crearVentanillas(): void	Cantidad:int	Ciclo para generar los punteros tipo ventanilla y agregarlos a la lista base.
insertarPrioridad():void	No recibe parámetros	Ingresa en la cola de prioridad un puntero tipo tiquete y aumenta los contadores de tiquetes generales y los de prioridad.
insertarNormal():void	No recibe parámetros	Ingresa en la cola de normal un puntero tipo tiquete y aumenta los contadores de tiquetes generales.
atenderFilas():bool	Pos:int	Saca un tiquete de la fila (se le valida en primera instancia la condición de la fila de prioridad) y obtiene el código y el tiempo del objeto. Utilización de un método auxiliar para insertar en ventanilla. El contador de tiquetes atendiditos aumenta.
insertarEnVentanilla():void	codigo_tiquete :string , pos int	Por medio de un ciclo ubica la ventanilla que se ingresó por parámetro, una vez ubicado se llama al método de atención de la ventanilla y se ingresa el código del tiquete.

2.2.2. Ventanillas:

En el caso de las ventanillas se creó una clase ventanilla, la cual cuenta con atributos, entre estos un código (código de la ventanilla) que es un string, un estado (atendiendo o no) que es un booleano, un código_tiquete_atencion (que es el código del tiquete atendiendo) que es un string y un contador de ventanilla (indica cantidad de tiquetes atendidos) que es un int. Entre sus métodos cuenta: con un constructor, un método atender (el cual primero llama al método eliminar, para sacar el tiquete anterior de la ventanilla, seguidamente el estado pasa a true, ya que está atendiendo; la cuenta de clientes en la ventanilla aumenta, y por último, imprime un mensaje del cliente que se está ateniendo) que recibe como parámetros el código del tiquete que es un string, el método eliminar (el cual primero hace que su estado pase a false, ya que deja de estar atendiendo, y después, deja el atributo código_tiquete_atencion como un string vacío) que no recibe parámetros, el método getContadorVentanilla (retorna el atributo count_ventanilla, que contiene la cantidad de tiquetes atendidos) que no recibe parámetros, el método getCodigo (retorna el atributo código, que contiene el código de la ventanilla) que no recibe parámetros, el método tiquete_en_Ventanilla (retorna el atributo código_tiquete_atencion, que contiene el código del tiquete atendiendo) que no recibe parámetros, el método estadoVentanilla (que si el atributo estado, es true, retorna "En atención" y de lo contrario retorna "Desocupada") que no recibe parámetros y por último, el método toStringVentanilla (que imprime el código, el estado de la ventanilla y el codigo_tiquete_atencion).

2.2.3. Tiquetes:

En el caso de los tiquetes se creó una clase tiquete, la cual cuenta con atributos, entre estos un código (código del tiquete) que es un string y un tiempo_entrada (el cual contiene el tiempo de entrada del tiquete) que es un tipo de dato time_t, de la biblioteca ctime. Entre sus métodos cuenta: con un constructor, un método getTimepo (el cual retorna el atributo tiempo_entrada, el cual contiene el tiempo de entrada) que no recibe parámetros, el método getCodigo (el cual retorna el atributo código, que contiene el código del tiquete) que no recibe parámetros, el método toStringTiquete (que imprime el código del tiquete), y, por último, una función que permite imprimir el contenido del tiquete, esto se implementó dado a que

llamar al método print en una cola lo que se imprimiría, sería un puntero, pero no el contenido; esas líneas de código permiten desplegar el contenido del objeto.

2.2.4. Lista Servicios:

Esta clase permite almacenar cada uno de los punteros a objetos servicio, un único atributo consiste en una lista (LinkedList). Entre sus métodos están el agregar un servicio a la lista el cual realiza un append al servicio que sea agregado recibiendo por parámetros su descripción y área a la cual pertenece. Para eliminar un servicio en particular o para borrar todos los existentes se implementó dos métodos los cuales eliminan tanto el objeto en memoria dinámica como el puntero que se encuentra en la lista. Dado a que el sistema requiere llevar una estadística de tiquetes por servicio se elaboró un métodos el cual recibe la posición del servicio que se quiere aumentar y por medio de los métodos propios de la estructura LinkedList se busca dicho servicio (goToPos()) , una vez encontrado el servicio se llama al método de aumentar tiquete de la clase servicio para modificar el atributo de ese servicio. Para el despliegue de todos los servicios de una determinada área se creó un método que recorra todos los punteros de la lista (ciclo de repetición for) y obtenga el nombre del servicio con el índice en que se encuentra dentro de la lista, de este mismo método se implementó uno con la variación de que incluye el numero de tiquetes que han sido dispensados por área esto para reflejar las estadísticas de cada una de las listas de servicio.

2.2.5. Lista Área:

Esta lista manipula la mayoría de los componentes de las clases implementadas en el sistema, en el caso de esta clase se decidieron implementar tres métodos básicos para agregar un área a la lista, eliminar un área de la lista y un borrado general de la lista. Esta estructura al igual que la lista servicios fue implementada por medio de una LinkedList con la variación de que esta clase almacena punteros a objetos Área, para facilitar el acceso a los elementos de la lista se diseñó un método de búsqueda que recibe la posición del área que se desea y por medio de la función GoToPos de la estructura LinkedList se recorre la lista hasta encontrarlo y devuelve como resultado el puntero que apunta al área que fue indicada en el parámetro. Esta clase presenta varios métodos para imprimir resultados en la terminal los cuales presentan una estructura de repetición muy similar con pequeñas variaciones en el

contenido que despliegan en cada una de las funciones de impresión pantalla (imprimir cada área, imprimir área con ventanillas, imprimir área con servicios).

2.2.6. Solución del problema:

Para la administración de cada uno de los componentes requeridos para dar solución al problema planteado, se decidió incorporar distintas clases las cuales manejaran los datos y las acciones necesarias. Se partió de la idea de una clase tiquete, servicio y ventanilla las cuales iban a formar parte de las clases base para el desarrollo de una clase general que las iba a contener, este clase general se le asigno el nombre de área y como se mencionó en la parte de clase Área presenta distintas estructuras que le permite administrar en el caso de los tiquetes por medio de colas, las ventanillas por medio de listas y el caso de los servicios que se implementó de manera distinta debido a que se concluyó que la manera más factible de manipular los servicios seria tener una clase aparte de lista servicios la cual permitiera acceder y modificar su contenido pero por medio de métodos esta clase para evitar problemas en manejar todo la información desde la clase área, una vez implementado tanto área como las clases bases, se implementó una clase de listas área la cual almacenaría todas las áreas del sistema y facilitaría su acceso en las llamadas que se hicieran en el main.

2.3. Análisis crítico

2.3.1. Implementaciones:

Parte de lo que se logró implementar en el presente trabajo se encuentran cada una de las clases necesarias para poder administrar los contenidos de las colas en las cuales, tanto sus datos como el su estado se pudieron diseñar de manera adecuado permitiendo el registro del tiempo de espera de cada uno en tiquetes en la cola. En el caso de las ventanillas al ser una estructura de lista su acceso por medio de índices permitió que a la hora de que el usuario quisiera que una determinada ventanilla atendiera el método de búsqueda fuera más sencillo al delegar dicha función al método ya establecido en el LinkedList. En el apartado de servicios se logró implementar una estructura que gestionara todas las acciones necesarias sin tener que recurrir a manipularlo todo con una sola clase Área, esto permitió encapsular muchas de las acciones que eran propias de un servicio. Otras de las implementaciones que

cambiaron la comunicación o relación que existía entre los servicios y las áreas fue el sí implementar una clase lista servicios que tuviera todos los servicios de todas las áreas o manipular una lista individual para cada una de las áreas presentes en el sistema. Esta última alternativa facilito el manejo de muchos de los aspectos a la hora de generar un tiquete de un servicio en particular y el registro de cantidad de tiquetes dispensados para cada uno de los servicios.

El tiempo de atención represento un problema al principio de la implementación de los tiquetes dado a que no se sabía el cómo registrar cada uno de los tiempos de los tiquetes que iban siendo atendidos, no obstante, con el agregado de un atributo adicional a la clase Área permitió ir llevando el registro de las diferencias entre el tiempo de entrada de un tiquete y el de su atención.

2.3.2. Aspectos que se podrían mejorar:

El uso recurrente de punteros en cada una de las estructuras utilizadas en las clases genera una necesidad de estar pidiendo memoria dinámica, esto implica la necesidad de estar devolviendo toda la memoria que está siendo solicitada. En algunas de estas estructuras se podría sustituir el uso de punteros por el uso solamente de objetos en memoria estática. Esto también conduce a la estructura elegida para contener tanto a las áreas como a los servicios el cual es una LinkedList que recurre igualmente al uso de memoria dinámica con nodos, el usar una estructura como un Arraylist pudo haber sido una elección más conveniente para almacenar estos objetos dado a que si bien los dos utilizan memoria dinámica el ArrayList lo utiliza en menor medida.

Otro de los puntos son los métodos para llamar a imprimir un determinado conjunto de datos, esto debido a que el método área y lista áreas poseen métodos muy similares, y podría haber algunos que no sean necesarios e incluso se puedan simplificar para evitar tantas llamadas a métodos entre clases que se podrían modificar de modo que solo se requiera una sola llamada para la obtención de ciertos datos.

3. Conclusiones

1. Utilizar listas con punteros es más conveniente que utilizar listas con arreglos, ya que, aunque existan fugas de memoria al borrar punteros el programa siempre funcionará, pero si se utilizan listas con arreglos el programa no funciona hasta que se libere correctamente la memoria.
2. Utilizar memoria dinámica es mejor que utilizar memoria estática, ya que existen casos en los cuales no se sabe con exactitud cuánta memoria van a requerir los datos al ejecutarse. Porque si se usa memoria estática se debe conocer el tamaño que utilizarán las estructuras que se van a ejecutar, debido a que no se puede establecer ni modificar en tiempo de ejecución y esto no lo hace conveniente.
3. Es provechoso hacer clases para separar ciertas funcionalidades en el programa, porque esto permite trabajar el concepto de modularidad, el cual trata de dividir el problema en partes más pequeñas haciendo que este sea más fácil de comprender y de manejar más rápido los errores.
4. Es conveniente utilizar las mismas estructuras que se crearon en la clase, ya que esto permite emplearlas de manera correcta dentro del programa (porque ya se tiene un conocimiento de ellas) y además son estructuras que están muy completas y son fáciles de comprender y manipular.
5. Utilizar switch para la realización de un menú es una forma de simplificar un poco el código y al mismo tiempo se ve más ordenado trabajar con este, que trabajar solo con el condicional if que implica un poco más de labor.
6. Al momento de cerrar un programa existen diversas formas (se puede utilizar exit, abort o return) por eso se debe hacer de la forma correcta, de manera tal que, siempre se debe cerciorar que se haya devuelto la memoria tanto estática como la dinámica solicitada, esto con el fin de seguir las buenas prácticas de programación y también de evitar las fugas de memoria.
7. La estructura cola (Queue) permite trabajar de manera rápida programas en los cuales se necesita llevar un control de elementos que se deben atender según su orden de llegada, ya que sus métodos al estar implementados de esta forma (FIFO) hacen posible y de manera más eficiente esta labor.

8. En algunas ocasiones manejar los posibles errores que se pueden presentar al recibir entradas dadas por el usuario se convierte en algo tedioso, ya que esto implica generar más código y pensar en todas las posibles entradas inválidas que se pueden recibir, pero si se utilizan algunas funciones internas de C++ que hagan ciertas validaciones con solo invocarlas se puede ahorrar tiempo y trabajo.
9. Manejar el tiempo en C++ es muy fácil, ya que existen librerías con funciones especializadas para manejar tanto el tiempo como la fecha actual del sistema.
10. Trabajar con constructores de copia y con los operadores de asignación en una clase se vuelve complicado al momento de desear crear otras clases con constructores que inicialicen sus atributos porque ese constructor y ese operador se perderán, ya que se C++ no hace el constructor de copia ni la asignación de manera automática y se debe tener mucho cuidado a la hora de crearlos.

4. Recomendaciones

1. Se recomienda trabajar con punteros en todo el programa y al mismo tiempo cerciorarse de que al momento de borrarlos se haga correctamente con el operador delete (como en el caso de listas de punteros que apuntan a otras listas con punteros se deben borrar primero estos últimos y después los primeros).
2. Se recomienda trabajar con memoria dinámica la mayoría del programa y al mismo tiempo devolver correctamente la memoria solicitada con el fin de evitar fugas de memoria.
3. Se recomienda identificar primero las posibles partes que contiene el programa y los requerimientos de cada una de estas, así se podrá establecer si es necesario o no la creación de una clase para cada parte identificada y una vez realizado esto el programa será más fácil de implementar.
4. Se recomienda primero identificar los posibles casos en los que se pueden aplicar ciertas estructuras según las características solicitadas en el problema y también determinar cuál estructura es más eficiente que otra (en el caso de que existan estructuras que tengan la misma funcionalidad).
5. Se recomienda hacer los menús utilizando switch, ya que de esta forma se le facilitará mostrar los distintos casos que solicita un programa y también utilizar la instrucción

do-while para que los menús no se cierren al realizar solo una operación, sino que se mantenga hasta que el usuario decida terminarlo.

6. Se recomienda utilizar la función exit declarada en <stdlib.h> y no usar abort, ya que exit permite que tenga lugar el procesamiento de finalización en tiempo de ejecución, es decir, llama a los destructores de los objetos estáticos globales inicializados y abort finaliza inmediatamente omitiendo lo anterior.
7. Cuando se trabajan con colas se recomienda utilizar la clase LinkedQueue en lugar de usar ArrayQueue, ya que con esta estructura (LinkedQueue) la cola crece dinámicamente al utilizar punteros y no es necesario indicarle un tamaño específico como en el caso del ArrayQueue, lo único malo de utilizar colas enlazadas es que se puede dar overhead por el manejo de punteros.
8. Se recomienda que al momento en el que se vaya a crear una interfaz sin importar el tipo que sea (gráfica o modo texto) primero se debe plantear todas las posibles entradas inválidas que un usuario pueda generar al momento de la interacción con el programa, una vez planteado esto se puede hacer una pequeña investigación de funciones que existen en ciertas librerías utilizadas por C++; por ejemplo: la función stoi( ) permite convertir un string a número entero, empty ( ) permite saber si una cadena está vacía o no, entre otras.
9. Se recomienda utilizar la libería time.h y ctime, porque estas contienen funcionalidades de fácil manejo y entendimiento a la hora de realizar ciertas operaciones con el tiempo, no requiere de mucha implementación para obtener la hora, solo con invocar time_t lo puede lograr fácilmente.
10. Se recomienda no utilizar constructores de copia y operadores de asignación, ya que esto requiere de más trabajo. La mejor manera de hacerlo es crear las clases con los constructores que inicialicen los atributos y a la hora crear un nuevo objeto se puede utilizar lo siguiente: variable -> función (new nombre del constructor (parámetros)); y de esta forma el objeto se crea sin ningún problema. Para que el ejemplo anterior quede más claro se puede imaginar que ya existe una clase llamada Persona y su constructor tiene los atributos: nombre y edad ya inicializados. Si se desea crear un objeto persona y que este sea agregado a una lista (ArrayList o LinkedList) se realiza lo siguiente: persona-> append (new Persona(María, 24)); y eso sería todo.

5. Referencias

Ferreira, C. R. (29 de abril de 2020). *8 Ways to Measure Execution Time in C/C++*. Level Up Coding. Medium. https://levelup.gitconnected.com/8-ways-to-measure-execution-time-in-c-c-48634458d0f9

Delft Stack. (15 de mayo de 2021). *Compruebe si la entrada es un número entero en C++*. https://www.delftstack.com/es/howto/cpp/check-if-input-is-integer-cpp/

Delft Stack. (16 de octubre de 2021). *Cómo convertir la cadena a Int en C++*. https://www.delftstack.com/es/howto/cpp/how-to-convert-string-to-int-in-cpp/

Delft Stack. (25 de febrero de 2021). *Comprueba si la cadena está vacía en C++*. https://www.delftstack.com/es/howto/cpp/cpp-check-if-string-is-empty/

Microsoft Learn. (26 de septiembre de 2022). *Finalización del programa C++*. https://learn.microsoft.com/es-es/cpp/cpp/program-termination?view=msvc-170

Programiz. (s. f.). *C++ time( ).* https://www.programiz.com/cpp-programming/library-function/ctime/time