Estructuras de datos:

Proyecto 1

Estudiante:

Ariana Jiménez Paniagua

Carnet (2023201036)

Carrera: Ingeniería en Computación

Estructuras de datos (Grupo 40)

Profesor: Mauricio Avilés

2024

Introducción

Este trabajo se hace con el objetivo de desarrollar un sistema de administración de colas que sea empleable en distintos contextos, utilizando el lenguaje de programación C++, también haciendo uso de distintas estructuras de datos y la programación orientada a objetos.

El programa a desarrollar debe ser flexible para así ser utilizable en diferentes aspectos, el sistema administra una lista de tipos de usuario, una lista de servicios disponibles y una lista de áreas. Es decir, el programa debe manejar distintitos tipos de usuario y cada uno de estos debe tener una prioridad. También se deben almacenar servicios, estos están asociados a un área y tienen prioridad. Por otro lado están las áreas que tienen una descripción y un código además de ventanillas, cada una con un código y estás deben manejar que tiquete se está atendiendo actualmente. Por último están los tiquetes que también tienen un código que se crea a partir de un servicio, los tiquetes tienen prioridad y almacenan la hora en la que fueron solicitados.

El programa en general debe manejar las siguientes funcionalidades principales:

- Un usuario solicita un tiquete.
- Una ventanilla atiende un tiquete.
- Sección para administrar el sistema (agregar, borrar, modificar datos).
- Consultar las estadísticas del sistema.

También se va a tener un menú principal con una interfaz de consola para así poder probar las distintas funcionalidades del sistema.

Además en este documento podrá encontrar la presentación y análisis del problema que consta de la presentación, metodología y análisis crítico de la implementación. También están las conclusiones y recomendaciones y por último las referencias.

Presentación y análisis del problema

1. Presentación del problema:

Se debe analizar y leer bien el problema, para así entender y comprender que es lo que se está pidiendo realizar, las distintas funcionalidades que se quieren tener y el contexto del problema.

Es necesario identificar las distintas clases para manejar adecuadamente todas las funcionalidades y datos que pide el sistema, y además lograr un código más ordenado y legible.

El programa debe abarcar las funcionalidades principales, como un método que permita al usuario solicitar un tiquete y la vez solicitar los datos necesarios para crear este, otro para que en determinado momento la ventanilla atienda x tiquete, aquí hay que tener en cuenta la hora a la que fue atendido.

Se debe mostrar distintos datos en pantalla, esto involucra imprimir el contenido de las listas y colas, para esto es necesario un adecuado manejo de las estructuras en las que se van a almacenar los distintos datos.

Deben implementarse métodos para poder administrar el sistema, esto implica agregar, eliminar y modificar elementos de las colas y listas. Hay que tener cuidado al realizar modificaciones porque pueden crear inconsistencias en el sistema cuando estás modificaciones también abarcan otros datos. Ya que los diferentes elementos están relacionados entre sí como los servicios y los tipos de usuario con los tiquetes, que a la vez están relacionados con las áreas, así como estás tienen sus servicios y ventanillas relacionados. Debido a esta complejidad que se da por estas distintas relaciones es importante dar un mensaje de advertencia cuando se vayan a borrar datos que impliquen eliminar otros datos, además de manejar distintas excepciones que se puedan crear al hacer esto.

Por otra parte en la sección de administración se debe tener una funcionalidad para limpiar las colas y estadísticas, esto es como una reinicialización del sistema sin alterar tipos de usuario, servicios ni áreas. Pero si deben eliminarse todos los contenidos de las colas de prioridad, y los tiquetes atendidos en ventanillas, junto con los datos que llevan el control de las estadísticas.

El sistema también tiene que manejar distintas estadísticas como el tiempo promedio de espera por cada área, para esto es necesario llevar el control de cuando se creo un tiquete y cuando se atendió, también las estadísticas de cuántos tiquetes se dispensaron por área, cuántos tiquetes atendió cada ventanilla, y cuántos fueron solicitados para cada servicio o tipo de usuario.

Se debe liberar la memoria adecuadamente al terminar de usar cada cola y lista o se elimina un elemento que ya no se vaya a utilizar, ya que todo esto se va a manejar en memoria dinámica para facilitar su uso, que involucra modificar, insertar y borrar elementos, además de recorrer e imprimir.

Debe hacerse un programa principal que maneje distintos menús en consola, para su adecuado funcionamiento hay que manejar las excepciones que se puedan crear cuando el usuario ingresa datos y que el programa no se caiga, también es importante que el menú sea fácil de utilizar para el usuario y la información sea desplegada de la forma más clara posible.

2. Metodología

Primero se leyó el problema planteado varias veces para poder identificar las clases, relaciones entre ellas, atributos, métodos y las estructuras necesarias para resolver el problema planteado. Primero se hizo un pequeño y simple diagrama de clases para organizar cómo iban a quedar los atributos métodos y relaciones, después la idea fue comenzar la implementación con la clase más sencilla, en este caso la clase TipoUsuario para manejar todo lo que tiene que ver con los tipos de usuario, su descripción, prioridad, el conteo de tiquetes emitidos para el tipo de usuario y distintos métodos para obtener y modificar estos datos.

Luego se realizó la clase Servicio que maneja todo lo relacionado con los servicios, su descripción, prioridad, área a la que están relacionados y el conteo de tiquetes solicitados, además de métodos para manipular estos datos según se necesite, a grandes rasgos getters y setters. Después se hizo una implementación sencilla de la clase Area pero surgió la necesidad de una clase Ventanilla que se encarga de inicializar ventanillas con su respectivo nombre (código de área + un consecutivo), tiquete actual siendo atendido y un conteo de la cantidad de tiquetes atendidos en la ventanilla, también tiene métodos getters y setters además de otros para agregar un tiquete actual o borrarlo. Desde la clase Area se utiliza la clase Ventanilla para almacenar las correspondientes a esa área, desde allí también se inicializan las mismas, además la clase Area cuenta con una descripción, un código, una cantidad de ventanillas, un contador de tiquetes dispensados por el área, una lista de ventanillas, servicios y tiquetes, además de la cola de tiquetes, se utiliza una lista paralela a la cola para imprimir los tiquetes que se encuentran almacenados en el área.

Para poder mostrar los datos de las distintas colas de tiquetes se decidió manejar en paralelo una lista, para poder recorrerla y mostrarla, está a la vez se va modificando cada vez que su respectiva cola se modifica.

Por último está clase Tiquete, tiene un código dado por el código de área más un consecutivo que empieza en 100, también tiene un tipo de usuario, servicio y un área asociados, cada tiquete se inicializa con la hora en que fue creado, y cuenta con una prioridad final que se obtiene al sumar la prioridad del tipo de usuario más la prioridad del servicio asociados al tiquete. Además tiene métodos getters para obtener los datos y hacer distintas operaciones con estos en el menú principal. La clase Tiquete además almacena la hora a la que fue atendido el tiquete, y calcula el tiempo de espera que tardó en ser atendido restando la hora de atención con la hora de creación.

También es importante mencionar que se implementó un método print en todas las clases mencionadas anteriormente para facilitar imprimir determinados datos de cada clase en pantalla.

Se tuvo problemas a la hora de utilizar las estructuras para almacenar objetos tipo Area, Tiquete, Servicio, Ventanilla, y TipoUsuario, así que se decidió utilizar todo en memoria dinámica para simplificar la implementación y no tener que utilizar el constructor de copia y el operador de asignación. Esto facilitó la manipulación de los distintos datos pero es importante considerar el cuidado que hay que tener con la memoria, es necesario liberarla adecuadamente cuando se terminan de utilizar los objetos.

Para liberar la memoria lo que se hizo fue recorrer cada cola y lista para así ir borrando elemento por elemento al que apuntaba cada puntero de la lista o cola, esto se hace sobre todo cuando el programa termina de utilizarse es decir al finalizar el main. También se implementó un destructor en la clase Area para liberar la memoria que se pide al hacer las listas de ventanillas, servicios, tiquetes y su respectiva cola. También se libera la memoria cuando se borran escogen las opciones de eliminar los distintos elementos. Sin embargo, a veces identificar donde más liberar memoria sobre todo en el programa principal ya que se manejan muchos datos.

Conforme se fue avanzando con la implementación del menú principal, fue necesario agregar más métodos y atributos a las distintas clases, para llevar el conteo de las estadísticas en TipoUsuario, Servicio, Ventanilla y Area, luego en la clase tiquete también fue necesario agregar atributos y métodos para llevar la cuenta del tiempo de espera entre ellos uno para convertir una hora en formato int a string.

Lo relacionado al tiempo de espera y mostrar esta estadística fue lo último en implementarse, se usó la biblioteca ctime y time.h para obtener el tiempo actual en horas, minutos y segundos, y se retorna un int con el tiempo total en segundos,

cuando se va a imprimir la hora de creación o atención del tiquete se usa una función para convertir el tiempo de un entero a String y que se muestre en el formato 00:00:00. Para esto se utilizaron las bibliotecas iomanip y sstream.

3. Análisis Crítico de la implementación

Luego de terminada la implementación se puede decir que se logró prácticamente cumplir con todo lo que se pedía en el proyecto.

Se logró dividir el menú principal por funcionalidades y que de esta manera quedara más organizado y claro cada una de las tareas que realiza el programa, aún así la extensión terminó siendo bastante grande.

Se podría mejorar el manejo de memoria, debido a la cantidad de lugares en los que usan estructuras en memoria dinámica es difícil identificar donde devolver esta memoria para evitar fugas, entonces sería mejor organizar esto para evitar futuros problemas.

Se podría hacer que al mostrar las estadísticas del tiempo promedio de espera de cada área, estas se vieran en un mejor formato, y no se diera el tiempo en segundos si no en minutos o incluso horas dependiendo la cantidad de tiempo.

Sería interesante implementar un método para cargar datos, y se carguen cierto número de áreas, ventanillas, servicios, tipos de usuario, y tiquetes ya establecidos en el sistema. De hecho se implementó un método parecido, este es llamado al inicio del main para cargar una serie de datos ya establecidos, sin embargo se encuentra comentado, pero es posible probarlo. Ligado a esto podría haber un método para que se guarden y agreguen los nuevos datos y las distintas modificaciones de estos que fueron realizada. Esto podría hacerse cuando se cierra el programa o cada cierta cantidad de tiempo para respaldar los datos en caso de algún inconveniente que se presente.

Se podrían utilizar otras estructuras de datos para almacenar o manejar los distintos objetos Area, Ventanilla, Servicio, Tiquete y TipoUsuario y tal vez así facilitar el manejo de los elementos o la eficiencia de las distintas operaciones que se realizan sobre estos. Por ejemplo al mostrar la cola de tiquetes, estos no siempre salen en orden de prioridad.

A veces con las funcionalidades de borrar elementos se dan fallos a la hora de mostrar las estadísticas o eliminar los elementos relacionados, esto puede deberse a algún error en la manipulación de ciertos datos, sobre todo ocurre al borrar alguna área ya que es la que se relaciona con más clases y objetos.

Conclusiones

Es importante tener bien claro que es lo que se está pidiendo en el problema, y también tener claro el contexto de este, informarse bien sobre cada aspecto y preguntar de ser necesario, porque de esta manera se podrá iniciar mejor el proyecto, con una idea clara y concisa.

Plantear un diagrama de clases ayuda a organizar mejor las funcionalidades del proyecto, permite subdividir el problema en subproblemas y así facilitar la implementación.

Entender y utilizar las estructuras de datos y los métodos de estos no se tan complicado debido a la experiencia previa que se ha tenido con estas, ya que han sido utilizadas e implementadas en clase y en otras tareas. Ver las similitudes de estas implementaciones pasadas sirve como guía para hacer uso de estas en otros contextos.

Al realizar una interfaz de consola, es importante dividir bien las funcionalidades y responsabilidades de cada parte del código para evitar confusiones o errores, el código se vuelve más legible, claro y fácil de mantener.

La biblioteca ctime proporciona tipos y funciones para trabajar con el tiempo y las fechas en C++. En particular, se usa la función time() para obtener la hora actual en segundos. También permite utilizar la estructura tm para representar la hora local dividida en componentes como horas, minutos y segundos.

Para convertir la hora de int a String se pueden utilizar las bibliotecas iostream, sstream y iomanip que permiten manipular y formatear la hora en segundos a un formato de cadena de hora legible.

Utilizar las estructuras de datos para imprimir y almacenar objetos tipo área, servicio, ventanilla, entre otros, puede ser complejo ya que hay que tener en cuenta cosas como el constructor de copia y el operador de asignación.

Hay que tener cuidado con las referencias circulares, cuando al mismo tiempo una clase utiliza otra y viceversa puede haber errores a la hora de compilar.

A la hora de utilizar memoria dinámica hay que tener en cuenta el manejo adecuado de la memoria, esto implica devolverla cuando sea necesario, para evitar posibles fugas o perdida datos.

Recomendaciones

Cuando se va a implementar un proyecto nuevo, se debe tener muy claro que es lo que se quiere hacer, para esto hay que considerar preguntar, investigar y averiguar todo lo que haga falta.

Si se quiere tener una mejor y más clara idea de cómo abordar un problema enfocado en la programación orientada a objetos, puede considerarse realizar un diagrama de clases, y esto facilitará la implementación a futuro.

Siempre es bueno recordar y apoyarse en materiales, proyectos y conocimientos previos que se tengan, porque esto puede servir de apoyo para implementar cosas parecidas a otras que ya se hayan hecho antes.

Si se va a realizar una interfaz en consola es recomendable pensar en cómo dividir las distintas funcionalidades y responsabilidades, para evitarse problemas y esto mejorará y facilitará la implementación en muchos aspectos.

Para manipular y realizar operaciones con el tiempo en c++, es recomendable utilizar la biblioteca ctime, ya que es bastante flexible, no es muy complicada de usar y es fácil de entender.

Si se quiere imprimir y utilizar distintas estructuras de datos con instancias de clases es mejor usar todo en memoria dinámica, haciendo que las estructuras contengan punteros a objetos en lugar de los objetos, esto simplifica la implementación.

Si se van a utilizar distintas clases que tienen referencias a otras y viceversa es mejor no hacer el #include de la clase y en lugar de eso poner class Area; lo que ayuda para que el compilador sepa que hay una definición de la clase pero que la compila después.

Es muy importante tener en cuenta el adecuado manejo de memoria para lograr un mejor código e implementación en términos de eficiencia, evitar fugas de memoria y así evitar que eventualmente el programa falle.

Referencias

Tutorialspoint. (2020). *C++ date and time*. https://www.tutorialspoint.com/cplusplus/cpp_date_time.htm

TylerMSFT. (2023). *Funciones. Microsoft Learn: Build skills that open doors in your career.* https://learn.microsoft.com/es-es/cpp/standard-library/iomanip-functions?view=msvc-170

Eferion. (2017). *Diferencia string Y StringStream.* Stack Overflow en español. https://es.stackoverflow.com/questions/60358/diferencia-string-y-stringstream