Instituto Tecnológico de Costa Rica

Escuela de Computación

Campus Tecnológico San José

Estructura de Datos

Proyecto #0

Sistema de administración de colas

Profesor:

Mauricio Avilés

Estudiantes:

José Manuel Granados Villalobos (2022049007)

Adrián Josué Vega Barrantes (2022104007)

Introducción

El siguiente proyecto consiste en el desarrollo de un software que se encargará de mantener un control de las personas que llegan a un local mediante el uso de tiquetes; estos tiquetes permiten que el usuario se pueda movilizar alrededor del área sin la necesidad de quedarse de pie en una fila real y, de esta forma, solo deben tener presente cuales tiquetes se están atendiendo para que sea atendido en su momento.

Sin embargo, este sistema no es el que se puede encontrar normalmente en un banco u hospital, en el que la persona solo retira su tiquete y espera; este sistema presenta en pantalla un tipo de menú que le brinda la posibilidad al usuario de elegir la opción que desee utilizar en ese momento.

Primeramente, se encuentra el menú principal, en donde vienen opciones solo para los usuarios que serán atendidos por cada área del local, y también opciones para los que administran el local; estos últimos tendrán la posibilidad de añadir o eliminar áreas y servicios que se presenten en cada área respectivamente; además de también conocer estadísticas del sistema, como los tiempos que se tardan en atender a los clientes o saber cuántos tiquetes se han entregado a lo largo de su uso; y los estados en los que se encuentran las colas (filas).

Este sistema tiene el objetivo de volver más eficaz y cómodo el atendimiento que las instituciones o locales brindan a sus clientes e incluso la configuración necesaria por parte de los administradores, convirtiendo lo tradicional en algo más virtualizado y sencillo para los clientes.

Ahora bien, el proyecto tiene su importancia pues se hace uso de ciertas estructuras de datos que facilitan el manejo de estos mismos, siendo útiles para tener listas (ArrayList) o hacer un manejo exitoso de la entrada y la salida de los datos tal cual una cola, brindada por la LinkedQueue, por ejemplo.

Por último, se incluye un software que simula este sistema de administración de colas en un local. Este software imprime en pantalla las diferentes áreas, servicios, ventanillas, tiquetes, operación con las mismas y estadísticas del programa. Cada parte del código del software incluye una documentación de que hace dicha parte. Además, el presente

documento explica detalladamente las soluciones implementadas en el código y cómo se logró desarrollarlas.

Presentación y análisis

1. Descripción del problema:

El problema que se resolvió en este proyecto consiste en el desarrollo de las bases de un sistema de administración de colas. Este sistema debe ser lo suficientemente abstracto para poder ser utilizado en distintos tipos de escenarios. Se establece que el programa debe cumplir con los siguientes requerimientos:

a) Manejar áreas, las cuales cada una tiene su nombre, código, descripción y una cola de espera. Dentro de estas áreas, debe ser posible solicitar tiquetes y ser ubicado en la última posición de la fila, ya sea la fila regular o la preferencial.
b) Cada área debe tener elementos donde se atiendan a los clientes de la cola, estos elementos se llaman ventanillas y cada una tiene un código que las identifica y el último tiquete atendido.
c) El programa también debe tener servicios, los cuáles van a depender del ambiente en el que se esté utilizando el programa. Cuando se seleccione un servicio, se debe crear un tiquete en el área que se corresponde a dicho servicio.
d) Por otro lado, debe ser posible atender un tiquete desde una ventanilla, esto significa quitar el tiquete de mayor prioridad de la cola. En caso de no haber ningún tiquete debe indicarse.
e) Tiene que ser posible ver el estado de todas las colas que existen.
f) Debe poderse crear nuevas áreas y eliminar áreas existentes.
g) También tiene que poderse crear nuevos servicios y eliminar existentes, además de poderse cambiar el orden en el que se encuentra un servicio.
h) El programa tiene que poder mostrar ciertas estadísticas del sistema. Se debe poder mostrar el tiempo promedio de espera por área, el total de tiquetes creados por área, el total de tiquetes de cada ventanilla, el total de tiquetes por cada servicio existente y el total de tiquetes preferenciales creados en todo el sistema.

i) Por último, el menú del programa tiene que poder regresar de un lugar al anterior. Esto quiere decir que si se selecciona la opción de agregar un área, debe ser posible regresar al menú principal después de agregar un área sin que el programa termine.

2. Metodología:

La metodología que se utilizó para resolver el problema fue orientada a objetos. Se dividió el problema en diferentes objetos, esto para tener una mayor claridad a la hora de programar las clases y los métodos que debía tener cada clase. Los objetos que se identificaron son los siguientes: área, servicio, ventanilla y tiquete. La implementación de la descripción del problema propuesta anteriormente es la siguiente:

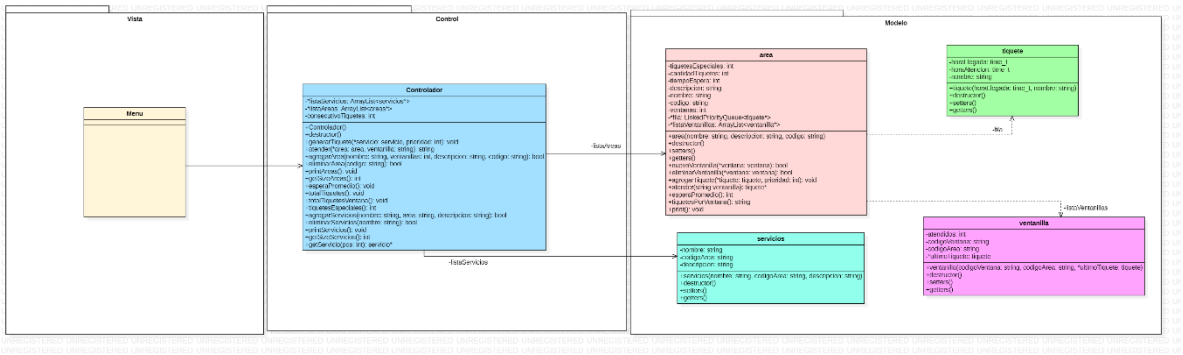
- Área: este objeto o clase tiene asociado su nombre, descripción, código, una lista de ventanas, una cola de tiquetes, la cantidad de tiquetes en el área, la cantidad de tiquetes especiales en el área y el tiempo de espera. Para implementar la lista de ventanas se utilizó el tipo de dato abstracto ArrayList, creada durante las lecciones del profesor Mauricio Áviles. Se creó un arreglo de punteros a objetos tipo ventana, esto nos permitió ahorrarnos el código que define la asignación y comparación, lo cual fue suficiente para decidir entre si se debía utilizar un arreglo de punteros o un arreglo normal. Por otro lado, para implementar la cola de tiquetes se usó el tipo de dato abstracto LinkedPriorityQueue, creada durante las lecciones del profesor Mauricio Áviles. Este tipo de dato abstracto se eligió debido a que permite crear una especie de arreglo de colas, dependiendo de la cantidad de prioridades que haya y los elementos mantienen el orden de llegada. Esta última característica es clave debido a que en la fila del sistema se deben atender los clientes en el orden de llegada. Esta clase tiene constructor y destructor, es capaz de crear y eliminar ventanas, agregar tiquetes a la cola y atender tiquetes, calcular la cantidad de tiquetes por ventana y el promedio de espera y retornar un string para poder imprimir cada área.
- Ventanilla: los atributos de este objeto son su código de ventana, es decir el código de área concatenado con el consecutivo de ventanas, el código del área, el último tiquete que se atendió y la cantidad de tiquetes atendidos. En cuanto a sus métodos, únicamente se establecen los valores de sus atributos y se retornan los mismos, además de su constructor y destructor.

- Servicio: esta clase tiene asociado su nombre, código de área y descripción. En cuanto a sus funcionalidades, únicamente puede establecer sus atributos y retornarlos, además de su constructor y destructor.
- Tiquete: esta clase tiene como atributos el nombre del tiquete, la hora de llegada y la hora de salida. Sus métodos son el constructor, destructor y métodos setters y getters.
- Controlador: en esta clase es donde se concentra de la mayoría de la lógica del programa y se conectan todas las clases. Esta clase tiene una lista de áreas, una lista de servicios y el consecutivo de tiquetes. En el controlador, la lista de áreas y servicios también se implementaron como un arreglo de punteros a objetos tipo área para evitar tener que definir los operadores que omite el lenguaje c++. Su constructor crea ambas listas e inicializa el consecutivo de tiquetes en 0. Tiene métodos para generar tiquetes en su respectiva área, atender un tiquete de un área, agregar y eliminar áreas, agregar y eliminar servicios, imprimir las áreas y los servicios, obtener cuántas áreas y servicios existen, calcula el tiempo de espera promedio de cada área, la cantidad total de tiquetes por área, la cantidad total de tiquetes por ventana y la cantidad total de tiquetes especiales creados.
- Menú: el menú consiste en una serie de funciones que le permiten al usuario navegar a través del sistema y utilizar las diferentes funcionalidades del sistema.

En cuanto a los problemas en el proceso de la implementación de la solución del problema descrita anteriormente, se tuvieron los siguientes:

a) Al inicio del proceso, se había planteado crear el sistema sin un controlador. La idea era que el main funcionara como controlador y como main a la vez. Sin embargo, se tuvieron numerosos problemas en la implementación de la lógica que correspondería al controlador y la dificultad del proyecto estaba comenzando a aumentar cada vez más, no solo a la hora de crear las funciones sino que también se estaba perdiendo mucha legibilidad del código. Debido a esto, se decidió crear una nueva clase llamada controlador, donde se implementó la lógica necesaria y fue mucho más sencillo y el código quedó más legible.

Diagrama UML:



3. Análisis crítico:

Analizando objetivamente la implementación de la solución, en cuanto a la clase área, se logró implementar la clase área con sus respectivos arreglos de punteros y cola de prioridad enlazada y los métodos descritos en la metodología. Para la clase ventanilla, servicio y tiquete, también se logró implementar todos sus atributos y métodos getters y setters. Respecto al controlador, se logró implementar la lista de punteros a áreas y la lista de punteros a servicios. Se logró que el controlador generara y atendiera tiquetes, imprimiera las áreas y los servicios, retornara la cantidad de áreas y servicios, calculara el promedio de espera de cada área, la cantidad de tiquetes por área, ventanilla y especiales. Por último, se logró que el menú pasara las pruebas de robustez hechas por los autores.

En cuanto a las cosas que se podrían mejorar del proyecto se tienen los siguientes aspectos:

a) Se podría mejorar mucho la legibilidad del menú. Este aspecto dificultó bastante la implementación de la solución en el main, ya que muchas veces no q uedaba claro el orden de ejecución de las funciones, por lo que se perdió mucho tiempo.
b) Faltó implementar el borrado correcto de algunos punteros. Esto porque a la hora de poner delete puntero, el programa se caía. Esto lo solucionamos quitando los delete de los punteros, pero está quedando memoria dinámica perdida.

Conclusiones

1. Los inputs del usuario son difíciles de manejar, pues el hecho de que exista la posibilidad de que el usuario ingrese datos que el software no es capaz de procesar y que, el mismo tenga que saber cómo actuar y no caerse es realmente un desafío. En partes como en el menú, donde se tuvo que hacer varias pruebas de funciones del string, pues este no podría ser de más de un carácter y, la letra que haya ingresado debía ser únicamente una de las que se presentaban. Esto implicó abarcar todas las posibles opciones que se pueden ingresar.
2. Se debe tener cuidado y orden con las subfunciones, sobre todo cuando se necesitan los datos que estas retornan en otras partes del código. En el proyecto se eligió este método para la clase principal (main), en donde la función main llamaba a la función menú y a partir de esta, se realizaban el resto de los llamados en donde, dependiendo del número que retornaban (subfunciones que retornaban un int) era la siguiente instrucción. Por lo tanto, es muy sencillo perderse y provocar que el sistema ya no funcione de forma óptima si tenemos un error al retornar un dato erróneo.
3. Conocer a fondo el funcionamiento de las estructuras de datos, ya sean proyectos creados por uno mismo o bibliotecas importadas, facilita el desarrollo de proyectos en base a estas. Pues, a primera vista nos topamos con un desconocimiento de cómo implementar dichas estructuras de datos, como la de colas, pero una vez se analizan con mayor profundidad, la implementación y utilización de estas fue sencilla.
4. Cuando se utiliza la memoria dinámica del lenguaje c++, se requiere el uso de punteros. En caso de no utilizar punteros, se estaría perdiendo la variable, dato o elemento que hayamos creado o guardado en memoria dinámica.
5. Cuando se utilizan punteros a memoria dinámica, es necesario borrar manualmente a lo que apuntan dichos punteros. Si no se borra manualmente esta memoria, el programa va a tener fuga de memoria y esto puede generar un problema.
6. Cuando se establece el constructor de una clase, el lenguaje c++ omite la construcción automática de los operadores de asignación y comparación de dicha clase. Cuando se crea una clase en el lenguaje c++, este crea automáticamente el constructor y los operadores de asignación y comparación. Sin embargo, si se crea el constructor de una clase, también

se tiene que definir las operaciones de asignación y comparación, de lo contrario el programa va a generar problemas cuando se intente asignar o comparar dicha clase.

7. Definir los operadores de asignación y comparación para una clase es complicado, debido a que se tiene que tomar en cuenta todos los posibles casos que esto implica. De lo contrario el programa va a tener muchos errores.
8. El uso de punteros a clases permite asignarlas y compararlas sin necesidad de crear los operadores de asignación y comparación. Esto facilita y reduce el código que a desarrollar.
9. El diseñar una solución antes de crear el código reduce el tiempo, esfuerzo y la dificultad de la programación. Crear un diseño o una idea de como solucionar el problema que se plantea antes de implementar la solución ahorra muchos recursos, esto porque se reduce la cantidad de rutas erróneas y se tiene una mayor organización.
10. La reutilización del código ahorra muchos recursos para el programador. Cuando se está desarrollando código, es una buena práctica generalizar el mismo para poderlo utilizar en un futuro, ya que no haría falta volver a programar todo desde el inicio, sino que simplemente se reutiliza lo que ya se había desarrollado previamente.
11. Tener una buena comunicación con el resto de los compañeros es esencial cuando se trabaja en equipo. En base a nuestro conocimiento, no existen plataformas para trabajar sobre un mismo proyecto en c++, por lo que el único método que encontramos eficiente es que, cada uno vaya trabando y mande este proyecto como “actualización”. Ahora bien, la comunicación hace parte importante cuando encontramos clases o métodos nuevos, para así saber qué hacen, si hay que borrarlos o si de verdad sirve de algo y demás.

Recomendaciones

1. Se recomienda que se reduzca al máximo la interacción entre el programa y el usuario, esto para disminuir la cantidad de posibles errores que se generen, de manera que si simplemente no es lo que el sistema espera, se lance el error.
2. Se debe buscar en todo el sistema los momentos en el que el usuario debe ingresar algún dato y crear las restricciones dependiendo de cada momento.
3. Con respecto a trabajar con retornos, una sencilla forma de trabajar con eso es tener comentarios en el código que indiquen qué se está haciendo y que debería de retornar para saber cuál es la próxima instrucción en la función a la que se retorna.

4. CodeBlocks presenta una pequeña función que permite plegar todas las funciones, esto sería de gran ayuda para no perder en el código y que algunas funciones que no se necesita ver en un momento dado, dejen de ocupar campo visual.
5. Se recomienda investigar a profundidad todo código que se vaya a utilizar que no haya sido creado por uno mismo o repasar el código que se esté reutilizando para tener claras las funcionalidades de este a la hora de utilizarlo y no perder tiempo.
6. Cuando se necesite comparar o asignar clases, se recomienda el uso de punteros, ya que esto ahorra muchos problemas, como casos específicos, que se tendrían que tomar en cuenta en el código, ya que de lo contrario se generarían errores.
7. Cuando se utilicen punteros a memoria dinámica, estos se tienen que eliminar, por lo que se recomienda tener cuidado con qué es lo que apunta dicho puntero. En caso de que el puntero apunte a un arreglo de punteros, no basta con eliminar el puntero hacia el arreglo, hace falta ir eliminando cada elemento del arreglo.
8. Se recomienda realizar un diagrama que brinde una buena idea de cómo se podría solucionar el problema y cómo programar dicha solución, de esta manera se pueden evitar errores antes de haber perdido el tiempo programando algo que no va a servir.
9. También se recomienda programar código reutilizable para poder utilizarlo en futuros proyectos y no perder tiempo haciendo todo desde el inicio.
10. Para tener una buena comunicación cuando se trabaja de manera grupal, sería bueno que señalen el trabajo que acaban de hacer o correcciones al momento de enviarlo
11. Se puede mejorar la comunicación, mediante una pequeña explicación en el código diciendo para qué sirve lo que hicieron.

Referencias