ANÁLISIS Y DISEÑO DE ALGORITMOS I

INFORME
TRABAJO PRÁCTICO ESPECIAL

Alumnos:

- Vargas Juan Mateo
  vargas.mateo00@gmail.com
- Bugarini Matias
  matibugarini@gmail.com

INTRODUCCIÓN:

En este trabajo práctico se nos pidió especificar e implementar en C++ los tipos de datos básicos Lista, Fila y Árbol binario. Luego utilizar estos para simular, de forma muy simplificada, la llegada, espera y atención de los clientes durante su operatoria en un banco. Además se nos pedía plantear una aplicación con las siguientes funcionalidades:

1. Mesa de entrada: ingresa un nuevo cliente.
2. Atender al próximo cliente.
3. Abrir una nueva cola especial con criterio.
4. Cerrar cola especial.
5. Listar operaciones atendidas según rango de montos.

Primero desarrollaremos los TDA utilizados y luego explicaremos cómo resolvimos cada uno de los servicios.

Especificación formal en Nereus

CLASS Lista[elemento]
IMPORTS elemento, natural;
BASIC CONSTRUCTORS inic, agregar_inicio, agregar_final, agregar_arbitrario;
EFFECTIVE
TYPE Lista
OPERATIONS

inic: Lista;	BC
es_vacia: Lista -> boolean;	O
agregar_inicio: Lista * elemento -> Lista;	BC
agregar_final: Lista * elemento -> Lista;	BC
agregar_arbitrario: Lista * elemento * natural -> Lista;	BC
longitud: Lista -> natural;	O
pertenece: Lista * elemento -> boolean;	O
eliminar: Lista * elemento -> Lista;	M
mostrar_lista: Lista -> Lista;	O
vaciar: Lista -> ();	M
consultar_proximo: Lista -> elemento;	O
consultar_arbitriario: Lista * natural -> elemento;	O

END_CLASS;

Para la clase de lista utilizamos constructores básicos (inic, agregar_inicio, agregar_final y agregar_arbitrario) que nos permiten crear la lista y agregar elementos en ella.
Utilizamos funciones observadoras:

es_vacia: Devuelve un boolean indicando si la lista contiene o no elementos.
mostrar_lista: Utilizado para ver qué elementos hay dentro.
pertenece: Permite saber si un elemento ya está en la lista devolviendo un boolean.
longitud: Devuelve un entero indicando la cantidad de elementos que hay.
mostrar_lista: nos muestra los elementos de la lista.
consultar_proximo: nos devuelve el próximo elemento de la lista.

consultar_arbitriario: pasada una posición, avanza hasta la misma y devuelve el elemento.

Por último utilizamos dos funciones modificadoras, vaciar la cual elimina todos lo elementos de la lista y eliminar que elimina un elemento pasado por parámetro si es que este pertenece a la Lista.

inic ∈ O(1)
es_vacia ∈ O(1)
agregar_inicio ∈ O(1)

agregar_final ∈ O(n)
agregar_arbitrario ∈ O(n)
longitud ∈ O(n)
pertenece ∈ O(n)
eliminar ∈ O(n)
mostrar_lista ∈ O(n)
vaciar ∈ O(n)

consultar_proximo ∈ O(1)

consultar_arbitriario ∈ O(n)

CLASS Fila[elemento];
IMPORTS elemento, natural;
BASIC CONSTRUCTORS inicFila, agregarFila
EFFECTIVE
TYPE Fila
OPERATIONS

inicFila: -> Fila;	BC
agregar_fila: Fila * elemento -> Fila;	BC
es_vacia: Fila -> boolean;	O
retirar_proximo: Fila -> Fila;	M
consultar_proximo: Fila -> elemento;	O
cantidad_elementos: Fila -> natural;	O
mostrar_fila: Fila -> Fila;	O
consultar_arbitriario: Fila * natural -> elemento;	O
retirar_arbitriario: Fila * natural -> Fila;	M

END_CLASS;

En el TDA fila usamos funciones como inicFila y agregar_fila las cuales son constructoras básicas y son utilizadas para crear y agregar elementos a la fila.
Usamos también funciones observadoras que en este caso serían:

es_vacia: nos devuelve un boolean indicando si la fila contiene algún elemento o no.

consultar_proximo: utilizado para saber cual es el próximo elemento en la fila.

cantidad_elementos: devuelve un natural con la cantidad de elementos que hay en la fila

mostrar_fila: devuelve la fila y nos permite ver qué elementos contiene.

consultar_arbitrario: pasada una posición devuelve el elemento que se encuentre en la misma

Por último utilizamos dos modificadoras, retirar_proximo, esta es utilizada para extraer el siguiente elemento de la fila y, retirar_arbitrario, que se le pasa una posición, avanza hasta la misma y elimina ese nodo de la fila.

inicFila ∈ O(1)
agregar_fila ∈ O(n)
es_vacia ∈ O(1)

retirar_proximo ∈ O(1)
consultar_proximo ∈ O(1)
cantidad_elementos ∈ O(n)
mostrar_fila ∈ O(n)

consultar_arbitrario ∈ O(n)

retirar_arbitrario ∈ O(n)

CLASS Arbol[elemento];
IMPORTS Lista, natural;
BASIC CONSTRUCTORS inicArbol, insertar_ordenado;
EFFECTIVE
TYPE Arbol;
OPERATIONS

inicArbol: -> Arbol;	BC
insertar_ordenado: Arbin * elemento -> Arbol;	BC
es_vacio: Arbol -> boolean;	O

mostrar_arbol: Arbol -> Arbol; O
pertenece_al_arbol: Arbol -> boolean; O
profundidad_arbol: Arbol -> natural; O
cantidad_elementos: Arbol -> natural; O
listar_ordenado: Arbol -> Lista[elemento]; O
listar_frontera: Arbol -> Lista[elemento]; O

END_CLASS;

Para la clase de árbol utilizamos funciones constructores básicas (inicArbol, insertar_ordenado) que nos permiten crear el árbol y agregar elementos en él. Utilizamos funciones observadoras:

es_vacio: Devuelve un boolean indicando si el árbol contiene o no elementos.

mostrar_arbol: Utilizado para ver qué elementos hay dentro del mismo.

pertenece_al_arbol: Permite saber si un elemento o no ya está en el árbol devolviendo un boolean.

profundidad_arbol: Devuelve un entero indicando el número de niveles que hay en el árbol.

cantidad_elementos: Devuelve un entero indicando la cantidad de elementos que hay en el árbol.

listar_ordenado: Devuelve Lista ordenada a partir del árbol.

listar_frontera: Devuelve Lista a partir de nodos frontera del árbol.

inicArbol $\in O(1)$
insertar_ordenado $\in O(n)$
es_vacio $\in O(1)$
mostrar_arbol $\in O(n)$
pertenece_al_arbol $\in O(1)$
profundidad_arbol $\in O(n)$
cantidad_elementos $\in O(n)$
listar_ordenado $\in O(n)$
listar_frontera $\in O(n)$

CLASS CLIENTE;
IMPORTS natural;
BASIC CONSTRUCTORS inicCliente, Agregar;
EFFECTIVE
TYPE CLIENTE;
OPERATIONS
inicCliente: -> CLIENTE; BC
Agregar: CLIENTE * cadena * natural * cadena * cadena * natural * boolean -> CLIENTE; BC
obtener_nombre: CLIENTE -> cadena; O
obtener_edad: CLIENTE -> natural; O
obtener_operacion: CLIENTE -> cadena; O
obtener_destinatario: CLIENTE -> cadena; O
obtener_monto: CLIENTE -> natural; O
obtener_es_cliente: CLIENTE -> bool; O

END_CLASS;

Para la clase cliente sólo utilizamos funciones observadoras que obtienen lo deseado indicado por el nombre de las mismas; todas $\in O(1)$.

EXPLICACIÓN DE RESOLUCIÓN TPE:

Cómo el TPE nos pedía que máximo podía haber 2 colas especiales abiertas decidimos crear un arreglo ya que este es estático y en cada posición del mismo se encontraría una puntero a una fila, un string que contendría el “criterio 1” y un boolean que contendría el “criterio 2”.

```
struct info_arreglo{

    FILA<CLIENTE> * fila;

    string criterio1;

    bool criterio2;

};
```

Además realizamos un menú por el cual el usuario podrá elegir que servicio utilizar, seleccionando 0 para salir del programa.

SERVICIO 1:

void ingresar_nuevo_cliente(info_arreglo arreglo_colas[]) {...}

Esta función lo que hace es solicitar los datos para cargar un nuevo cliente mediante el constructor de nuestra clase CLIENTE. Una vez cargado chequea si el “criterio 1” y el “criterio 2” coinciden con alguna de las dos filas especiales. En el que caso de que coincidan la agrega a la correspondiente y sino la agrega a la fila por orden de llegada. Luego muestra cómo quedó la fila en la que fue insertado el nuevo cliente. La complejidad del algoritmo $\in O(1)$.

SERVICIO 2:

void atender_cliente(info_arreglo arreglo_colas[], LISTA<CLIENTE> *& atendidos){..}

Chequea la existencias de filas especiales. En el caso de que existan si tienen al menos 1 cliente en ellas las muestra como opción por pantalla para atender un cliente de estas; al igual que con la fila por orden de llegada. En el caso que no existan filas especiales directamente se atiende un cliente de la fila default si es que esta tiene al menos un cliente.

En el caso de que sea exitosa la operación de atender un cliente lo muestra por pantalla y lo agrega a una lista de “atendidos” que utilizaremos en otro inciso más adelante. La complejidad del algoritmo $\in O(1)$.

SERVICIO 3:

bool cola_abierta(info_arreglo arreglo_colas[], int i) {...}

Devuelve si existe o no una cola abierta. La complejidad del algoritmo $\in O(1)$.

void re_ordenar(info_arreglo arreglo_colas[], string criterio1, bool criterio2, int i) {...}

Re-ordena la nueva fila creada en el caso de que existan clientes en la fila por orden de llegada que cumplan con el “criterio 1” y “criterio 2” pasado por parámetro. La complejidad del algoritmo $\in O(n)$.

void nueva_cola_especial(info_arreglo arreglo_colas[], FILA<CLIENTE> * cola_1_criterio, FILA<CLIENTE> * cola_2_criterio) {...}

En el caso de que no se haya alcanzado el límite de filas especiales se solicitan los criterios para la creación de la nueva fila. Una vez creada se re-ordena la fila por

orden de llegada y los clientes que cumplan con los criterios de la nueva fila serán instertados en la misma. La complejidad del algoritmo ∈ O(n).

SERVICIO 4:

void cerrar_cola_especial(info_arreglo arreglo_colas[]) {...}

En el caso que exista alguna cola abierta y esta además no tenga ningún cliente esperando en la fila, se mostrará como opción por pantalla para ser cerrada si el usuario lo desea. La complejidad del algoritmo ∈ O(1).

SERVICIO 5:

void listar_segun_rango(LISTA<CLIENTE> * atendidos, LISTA<CLIENTE> *& listado_rango, int piso, int techo) {...}

Utilizando la lista “atendidos” creada en el servicio 2 se chequea que nodos de la misma cumplen con entrar en el rango dado por (piso y techo) pasado por parámetro. Aquellos nodos que cumplan la condición serán insertados en “listado_rango”. Además al finalizar de cargar “listado_rango” se realizará un promedio de la edad de los clientes que se encuentren en esta lista y se mostrará por pantalla. La complejidad del algoritmo ∈ O(n).

CÓDIGO FUENTE:

```
#include <iostream>
#include "LISTA.h"
#include "ARBOL.h"
#include "FILA.h"
#include "CLIENTE.h"

#include "servicio_1.h"
#include "servicio_2.h"
#include "servicio_3.h"
#include "servicio_4.h"
#include "servicio_5.h"
```

```
using namespace std;


int main()

{

    info_arreglo arreglo_colas[3];
    FILA<CLIENTE> * cola_principal = new FILA<CLIENTE>;
    FILA<CLIENTE> * cola_1_criterio = new FILA<CLIENTE>;
    FILA<CLIENTE> * cola_2_criterio = new FILA<CLIENTE>;
    LISTA<CLIENTE> * atendidos = new LISTA<CLIENTE>;
    LISTA<CLIENTE> * listado_rango = new LISTA<CLIENTE>;
    arreglo_colas[0].fila = cola_principal;
    arreglo_colas[1].fila = cola_1_criterio;
    arreglo_colas[2].fila = cola_2_criterio;


    int valor;


    cout << "BIENVENIDO AL BANCO V&B.inc" << endl;
    cout << "                           " << endl;
    cout << "ingrese segun lo operacion que desea realizar" << endl;
    cout << "1 ingresar nuevo cliente" << endl;
    cout << "2 atender proximo cliente" << endl;
    cout << "3 abrir nueva cola con orden especial" << endl;
    cout << "4 cerrar cola especial" << endl;
    cout << "5 Listar operaciones atendidas según rango de montos" << endl;
    cout << "0 para salir" << endl;
    cin >> valor;
    system("cls");   //borra todo lo de la consola para que quede mas limpio el menu


    while (valor != 0) {
        if (valor == 1) {
            ingresar_nuevo_cliente(arreglo_colas);
        }

        if (valor == 2) {
            atender_cliente(arreglo_colas, atendidos);
        }

        if (valor == 3) {
```

```
        if ((cola_abierta(arreglo_colas, 1) == false) || (cola_abierta(arreglo_colas, 2)
== false)) {
            nueva_cola_especial(arreglo_colas, cola_1_criterio, cola_2_criterio);
        } else cout << "numero maximo de colas especiales alcanzado." << endl;
    }

    if (valor == 4) {
        cerrar_cola_especial(arreglo_colas);
    }

    if (valor == 5) {
        int piso;
        int techo;
        cout << "ingrese minimo: "; cin >> piso;
        cout << "ingrese maximo: "; cin >> techo;
        if (piso <= techo) {
            if (atendidos->longitud() > 0) {
              listar_segun_rango(atendidos, listado_rango, piso, techo);
              cout << "LISTADO POR RANGO: " << endl;
              listado_rango->mostrar_lista();
            }
        } else {
            cout << "ingrese valores apropiados." << endl;
        }
    }

    if (valor > 5) {
        cout << "ingrese una opcion valida" << endl;
    }
    cout << "ingrese segun lo operacion que desea realizar" << endl;
    cout << "1 ingresar nuevo cliente" << endl;
    cout << "2 atender proximo cliente" << endl;
    cout << "3 abrir nueva cola con orden especial" << endl;
    cout << "4 cerrar cola especial" << endl;
    cout << "5 Listar operaciones atendidas según rango de montos" << endl;
    cout << "0 para salir" << endl;
    cin >> valor;
    system("cls");   //borra todo lo de la consola para que quede mas limpio el
menu

  }


  delete cola_principal;
```

```
    delete cola_1_criterio;
    delete cola_2_criterio;            //LIBERAR MEMORIA
    delete atendidos;
    delete listado_rango;

    return 0;
}
```

SERVICIO 1:

```
#ifndef SERVICIO_1_H_INCLUDED
#define SERVICIO_1_H_INCLUDED

struct info_arreglo{
    FILA<CLIENTE> * fila;
    string criterio1;
    bool criterio2;
};

void ingresar_nuevo_cliente(info_arreglo arreglo_colas[]) {

    string nombre;
    unsigned int edad;
    string operacion;
    string destinatario;
    unsigned int monto;
    bool es_cliente;
    string sn_cliente;;

    cout << "nombre: "; cin >> nombre;
    cout << "edad: "; cin >> edad;
    cout << "operacion (retiro, deposito, transferencia o pago): "; cin >> operacion;
    cout << "destinatario (persona, banco o impuesto): "; cin >> destinatario;
    cout << "monto: "; cin >> monto;
    cout << "es_cliente (si/no): "; cin >> sn_cliente;
    if ((sn_cliente == "si") || (sn_cliente == "SI")) {
        es_cliente = true;
    }
    if ((sn_cliente == "no") || (sn_cliente == "NO")) {
        es_cliente = false;
    }

    CLIENTE cliente = CLIENTE(nombre, edad, operacion, destinatario, monto,
es_cliente);
```

```
   if ((arreglo_colas[1].criterio1 == cliente.obtener_operacion()) &&
(arreglo_colas[1].criterio2 == cliente.obtener_es_cliente())) {
            arreglo_colas[1].fila->agregar_fila(cliente);
            cout << "cola por criterio 1: " << endl;
            arreglo_colas[1].fila->mostrar_fila();
   } else {
       if ((arreglo_colas[2].criterio1 == cliente.obtener_operacion()) &&
(arreglo_colas[2].criterio2 == cliente.obtener_es_cliente())) {
            arreglo_colas[2].fila->agregar_fila(cliente);
            cout << "cola por criterio 2: " << endl;
            arreglo_colas[2].fila->mostrar_fila();
       } else {
       arreglo_colas[0].fila->agregar_fila(cliente);
       arreglo_colas[0].fila->mostrar_fila();
       }
   }
}

#endif // SERVICIO_1_H_INCLUDED
```

SERVICIO 2:

```
#ifndef SERVICIO_2_H_INCLUDED
#define SERVICIO_2_H_INCLUDED
#include "servicio_3.h"

void atender_cliente(info_arreglo arreglo_colas[], LISTA<CLIENTE> *& atendidos) {

   int valor = 0;

   if ((cola_abierta(arreglo_colas, 1) == true) || (cola_abierta(arreglo_colas, 2) ==
true)) {
       if (arreglo_colas[0].fila->cantidad_elementos() > 0) {
           cout << "que cola desea atender?" << endl;
           cout << "0 para fila default" << endl;
       }

   }

   if (cola_abierta(arreglo_colas, 1) == true)
       if (arreglo_colas[1].fila->cantidad_elementos() > 0)
           cout << "1 para el siguiente criterio: " << arreglo_colas[1].criterio1 << ", " <<
arreglo_colas[1].criterio2 << endl;
```

```
    if (cola_abierta(arreglo_colas, 2) == true)
        if (arreglo_colas[2].fila->cantidad_elementos() > 0)
            cout << "2 para el siguiente criterio: " << arreglo_colas[2].criterio1 << ", " <<
arreglo_colas[2].criterio2 << endl;

    if ((cola_abierta(arreglo_colas, 1) == true) || (cola_abierta(arreglo_colas, 2) ==
true)) {
         if ((arreglo_colas[1].fila->cantidad_elementos() > 0) ||
(arreglo_colas[2].fila->cantidad_elementos() > 0) ||
(arreglo_colas[0].fila->cantidad_elementos() > 0)) {
            cin >> valor;
            CLIENTE datos = arreglo_colas[valor].fila->consultar_proximo();
            cout << "nombre: " << datos.obtener_nombre() << endl;
            cout << "edad: " << datos.obtener_edad() << endl;
            cout << "operacion realizada: " << datos.obtener_operacion() << endl;
            cout << "destinatario: " << datos.obtener_destinatario() << endl;
            cout << "monto: " << datos.obtener_monto() << endl;
            arreglo_colas[valor].fila->retirar_proximo();
            atendidos->agregar_final(datos);
         }
          else {
            cout << "no hay clientes por atender" << endl;
          }
        } else {
            if (arreglo_colas[0].fila->cantidad_elementos() > 0) {
                CLIENTE datos = arreglo_colas[0].fila->consultar_proximo();
                cout << "nombre: " << datos.obtener_nombre() << endl;
                cout << "edad: " << datos.obtener_edad() << endl;
                cout << "operacion realizada: " << datos.obtener_operacion() << endl;
                cout << "destinatario: " << datos.obtener_destinatario() << endl;
                cout << "monto: " << datos.obtener_monto() << endl;
                arreglo_colas[valor].fila->retirar_proximo();
                atendidos->agregar_final(datos);
            } else {cout << "no hay clientes por atender" << endl;}

        }

}

#endif // SERVICIO_2_H_INCLUDED
```

SERVICIO 3:

```
#ifndef SERVICIO_3_H_INCLUDED
#define SERVICIO_3_H_INCLUDED

bool cola_abierta(info_arreglo arreglo_colas[], int i) {

   bool resultado = false;

   if ((arreglo_colas[i].criterio1 == "retiro") || (arreglo_colas[i].criterio1 == "deposito") ||
(arreglo_colas[i].criterio1 == "transferencia") || (arreglo_colas[i].criterio1 == "pago")) {
      resultado = true;
   }

   return resultado;

}


void re_ordenar(info_arreglo arreglo_colas[], string criterio1, bool criterio2, int i) {

      int posicion = 1;
      int longitud = arreglo_colas[0].fila->cantidad_elementos();

      while (posicion <= longitud) {
         CLIENTE datos = arreglo_colas[0].fila->consultar_arbitriario(posicion);
         if ((datos.obtener_operacion() == criterio1) && (datos.obtener_es_cliente()
== criterio2)) {
            arreglo_colas[i].fila->agregar_fila(datos);
            arreglo_colas[0].fila->retirar_arbitriario(posicion);
            posicion = 0;
            longitud = arreglo_colas[0].fila->cantidad_elementos();
         }
         posicion ++;
   }
}


void nueva_cola_especial(info_arreglo arreglo_colas[], FILA<CLIENTE> *
cola_1_criterio, FILA<CLIENTE> * cola_2_criterio) {

   string criterio1, sn_cliente;
   bool criterio2;
   cout << "ingrese el criterio que desea para la nueva cola" << endl;
   cout << "criterios disponibles..." << endl;
   cout << "para criterio 1 (retiro, deposito, transferencia, pago)" << endl;
```

```
    cout << "criterio 1: "; cin >> criterio1;
    cout << "para criterio 2 (si, no) cliente del banco" << endl;
    cout << "criterio 2: "; cin >> sn_cliente;
    if ((sn_cliente == "si") || (sn_cliente == "SI")) {
        criterio2 = true;
    }
    if ((sn_cliente == "no") || (sn_cliente == "NO")) {
        criterio2 = false;
    }

    if (cola_abierta(arreglo_colas, 1) == false) {
        arreglo_colas[1].fila = cola_1_criterio;
        arreglo_colas[1].criterio1 = criterio1;
        arreglo_colas[1].criterio2 = criterio2;
        re_ordenar(arreglo_colas, criterio1, criterio2, 1);
        }
        else {
            arreglo_colas[2].fila = cola_2_criterio;
            arreglo_colas[2].criterio1 = criterio1;
            arreglo_colas[2].criterio2 = criterio2;
            re_ordenar(arreglo_colas, criterio1, criterio2, 2);
            }

}

#endif // SERVICIO_3_H_INCLUDED
```

SERVICIO 4:

```
#ifndef SERVICIO_4_H_INCLUDED
#define SERVICIO_4_H_INCLUDED
#include "servicio_3.h"

void cerrar_cola_especial(info_arreglo arreglo_colas[]) {

    if (cola_abierta(arreglo_colas, 1) == true) {
        if (arreglo_colas[1].fila->es_vacia() == true) {
            cout << "1 para cerrar la fila con el siguiente criterio: " <<
arreglo_colas[1].criterio1 << ", " << arreglo_colas[1].criterio2 << endl;
        }
    }

    if (cola_abierta(arreglo_colas, 2) == true) {
```

```
        if (arreglo_colas[2].fila->es_vacia() == true) {
            cout << "2 para cerrar la fila con el siguiente criterio: " <<
arreglo_colas[2].criterio1 << ", " << arreglo_colas[2].criterio2 << endl;
        }
    }

    if ((cola_abierta(arreglo_colas, 1) == true) || (cola_abierta(arreglo_colas, 2) ==
true)) {
        if ((arreglo_colas[1].fila->es_vacia() == true) || (arreglo_colas[2].fila->es_vacia()
== true)) {
            int valor;
            cin >> valor;
            arreglo_colas[valor].fila = NULL;
            arreglo_colas[valor].criterio1 = "CERRADA";
            cout << "se cerro con exito la cola especial " << valor << "." << endl;
    }
    }
}
#endif // SERVICIO_4_H_INCLUDED
```

SERVICIO 5:

```
#ifndef SERVICIO_5_H_INCLUDED
#define SERVICIO_5_H_INCLUDED


void listar_segun_rango(LISTA<CLIENTE> * atendidos, LISTA<CLIENTE> *&
listado_rango, int piso, int techo) {

    int posicion = 1;
    int longitud = atendidos->longitud();
    int suma_edades = 0;

    while (posicion <= longitud) {
        CLIENTE datos = atendidos->consultar_arbitriario(posicion);
        if ((datos.obtener_monto() >= piso) && (datos.obtener_monto() <= techo))
            {
                listado_rango->agregar_inicio(datos);
                suma_edades = suma_edades + datos.obtener_edad();
            }
        posicion ++;
    }
```

```
    cout << "                                                    " << endl;
    cout << "edad promedio de la lista: " << (suma_edades /
listado_rango->longitud()) << endl;
    cout << "                                                    " << endl;

}


#endif // SERVICIO_5_H_INCLUDED
```

CONCLUSIÓN:

Este trabajo nos ayudó para poder entender los temas aprendidos en la materia con el uso de contenidos teóricos que son realmente importantes, además del uso de Code Blocks la cual es una herramienta para poder aprender el lenguaje de programación C++. En el cual utilizamos las clases y encabezados para un manejo más eficiente y ahorrativo de espacio y datos.