Programación 4 - 2022
Laboratorio 0

Consideraciones generales:

- La entrega podrá realizarse hasta el **lunes 28 de marzo de 2022 a las 15 hrs**.
- El código fuente y el archivo Makefile deberán ser entregados mediante el EVA del curso [1] dentro de un archivo con nombre `<número de grupo>_lab0.zip (o tar.gz)`. Dentro del mismo archivo comprimido, se deberán entregar 2 archivos de nombres: `resp_lab0.txt` e `integrantes.txt`. En el primero, se incluirán las respuestas a las preguntas planteadas y las aclaraciones que se consideren necesarias. En el segundo se incluirán los nombres y correos electrónicos institucionales (`@fing.edu.uy`) de cada uno de los miembros del grupo.
- El archivo `Makefile` entregado debe ser independiente de cualquier entorno de desarrollo integrado (IDE, por sus siglas en inglés).
- Las entregas que no cumplan estos requerimientos no serán consideradas. El hecho de no realizar una entrega implica la insuficiencia del laboratorio completo.

Objetivo

En el Laboratorio 0 se espera que el estudiante practique la implementación de conceptos básicos de orientación a objetos utilizando el lenguaje C++ [2], así como el uso/repaso básico del entorno de programación en Linux [3,4]. También se espera que el estudiante consulte el material disponible en el EVA del curso [1] y complemente las consultas recurriendo a Internet con espíritu crítico y corroborando, en la medida de lo posible, que las fuentes consultadas sean confiables.

Problema

Cada verano, estudiantes de la Facultad de Ingeniería vacacionan de forma masiva en las playas de la costa de Rocha. Sin embargo, en los últimos años los precios de los alquileres han subido demasiado y cada vez se hace más difícil disfrutar de unas merecidas vacaciones. Es por esto, que un grupo de egresados de la FIng, administradores de un hostel, tuvo la brillante idea de ofrecer habitaciones a estudiantes de FIng (coloquialmente llamados Fingers) a precios preferenciales.

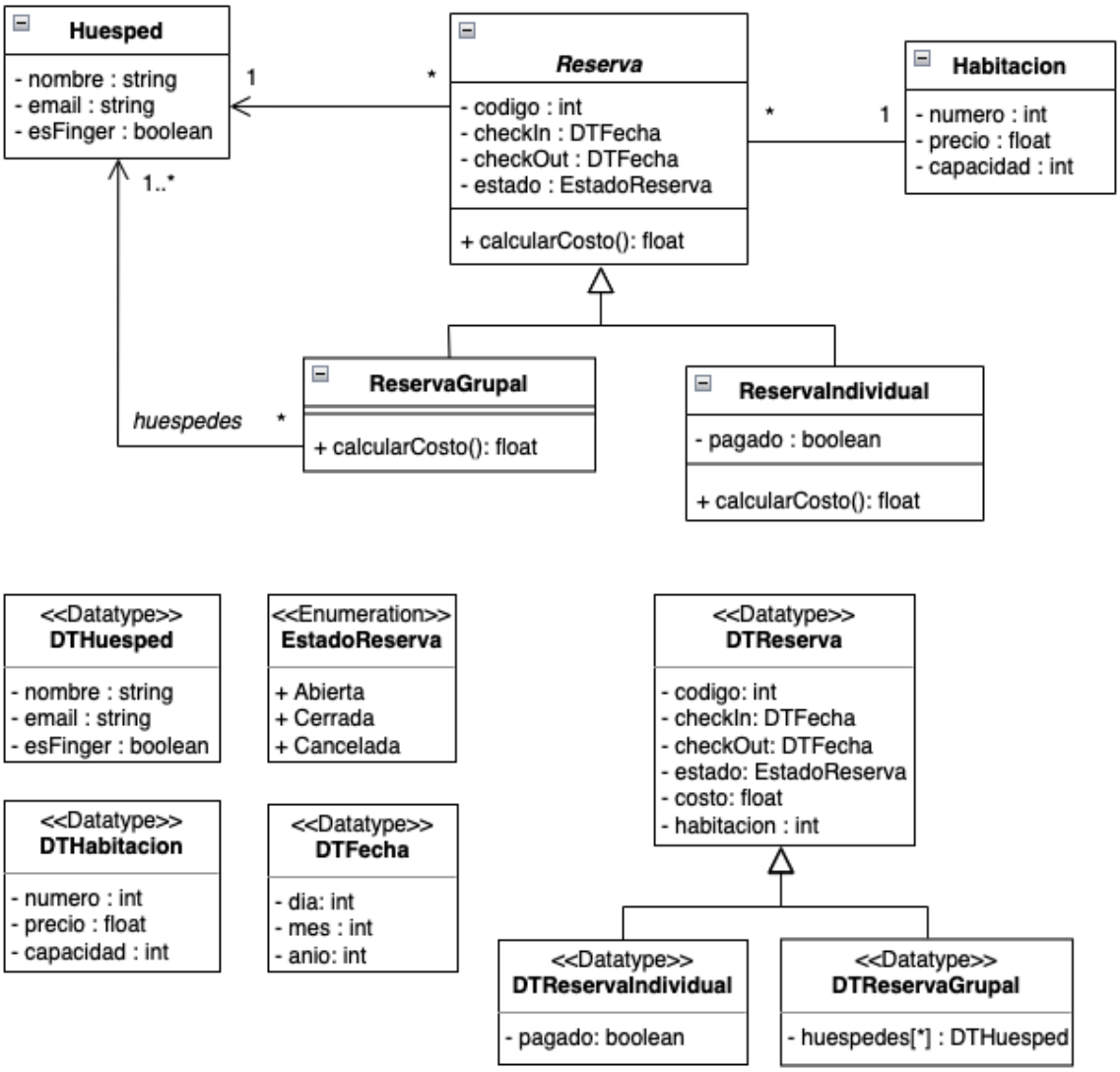
Ante el planteo de estos egresados, un grupo de estudiantes (que cursan P4) se ofreció a diseñar una aplicación que permita administrar las reservas dentro del hostel. Como “LosFingersVeraneanMuyBarato” no era un muy buen nombre para una aplicación, decidieron llamarla “FingVMB” y le han encomendado el desarrollo de la misma.

De cada huésped interesa conocer su email, que lo identifica, su nombre y si efectivamente es Finger o no. Para cada habitación del hostel se conoce un número que la identifica, su precio por noche y su capacidad. Por otro lado, los huéspedes realizan reservas sobre una habitación. Para cada reserva el sistema almacena su código que la identifica, la fecha de check in (entrada) y la fecha de check out

(salida). Adicionalmente, cada reserva tiene un estado que puede ser: “Abierta”, “Cerrada” o “Cancelada”. Toda reserva comienza siendo “Abierta” y puede pasar a “Cerrada” cuando el usuario se hospeda en la habitación reservada, o “Cancelada” si el huésped la cancela antes. Las reservas pueden ser individuales o grupales. Para cada reserva individual interesa además saber si fue abonada o no.

Las reservas tienen además un costo. En una reserva individual el costo es el precio de la habitación por la cantidad de días. En una reserva grupal, si hay por lo menos 2 Fingers, el costo total de la reserva tiene un 30% de descuento.

En base a la descripción anterior se ha diseñado la estructura de clases que muestra la siguiente figura.



Ejercicio 1

Se pide

1. Implementar en C++ todas las `clases` (incluyendo sus atributos, pseudo-atributos, constructores y destructores, y los getters y setters necesarios para las operaciones que se indican más adelante), `enumerados` y `datatypes` que aparecen en el diagrama. Para las fechas en caso de recibir dd > 31 o dd < 1 o mm > 12 o mm < 1 o aaaa < 1900, se debe lanzar la excepción `std::invalid_argument`. No se deben hacer más que estos controles en la fecha (ej. la fecha 31/2/2021 es válida para esta realidad).

2. Implementar en C++ las siguientes operaciones:

a) **`void agregarHuesped(string nombre, string email, boolean esFinger)`**

Registra un nuevo huésped en el sistema. Si ya existe un huésped registrado con el mismo email, se lanza una excepción de tipo `std::invalid_argument`.

b) **`void agregarHabitacion(int numero, float precio, int capacidad)`**

Registra una nueva habitación en el hostel. Si ya existe una habitación registrada con el mismo número, se lanza una excepción de tipo `std::invalid_argument`.

c) **`DtHuesped** obtenerHuespedes(int& cantHuespedes)`**

Devuelve un arreglo con información sobre los huéspedes registrados en el sistema. El parámetro `cantHuespedes` es un parámetro de salida donde se devuelve la cantidad de huéspedes devueltas por la operación (corresponde a la cantidad de instancias de `DTHuesped` retornadas).

d) **`DtHabitacion** obtenerHabitaciones(int& cantHabitaciones)`**

Devuelve un arreglo con información sobre las habitaciones registradas en el sistema. El parámetro `cantHabitaciones` es un parámetro de salida donde se devuelve la cantidad de habitaciones devueltas por la operación (corresponde a la cantidad de instancias de `DtHabitacion` retornadas).

e) **`DtReserva** obtenerReservas(DTFecha fecha, int& cantReservas)`**

Devuelve un arreglo con información de las reservas para la fecha indicada. El parámetro `cantReservas` es un parámetro de salida donde se devuelve la cantidad de reservas devueltas por la operación (corresponde a la cantidad de instancias de `DTReserva` retornadas).
Entre los datos específicos de cada reserva individual se encuentra si la misma fue pagada o no, mientras que para cada reserva grupal se indican los datos de cada huésped.

f) **`void registrarReserva(string email, DtReserva* reserva)`**

Registra una reserva individual o grupal para el huésped identificado por `email`. El parámetro de entrada `reserva` contiene la información completa de la reserva. Entre los datos comunes a ambos tipos de reserva se encuentra el número de habitación, la fecha de checkIn y checkOut. Además, si `reserva` es una instancia de `DtReservaIndividual` contiene si la reserva fue pagada o no, mientras que si es un instancia de `DtReservaGrupal`, se indica la lista de huéspedes. La reserva se da de alta con el estado "Abierta" y un código generado por el sistema. Si no existe una habitación registrada en el sistema con el número indicado en el campo habitación de `reserva`, o

si no existe un huésped registrado con el email `email`, se lanza una excepción de tipo `std::invalid_argument`.

Notas:

- A los efectos de este laboratorio, considere que el sistema maneja un conjunto acotado de huéspedes, habitaciones y reservas por habitación, donde la cota se define por las constantes `MAX_HUESPEDES`, `MAX_HABITACIONES` y `MAX_RESERVAS`, respectivamente.
- Puede implementar operaciones auxiliares en las clases dadas en el diagrama si considera que le facilitan para la resolución de las operaciones pedidas.
- Se puede utilizar el tipo `std::string` para implementar los atributos de tipo string.
- En este laboratorio no se pueden utilizar estructuras de datos de la biblioteca STL, tales como vector, set, map, etc.
- Se sugiere crear una clase auxiliar llamada `Sistema` que implemente las operaciones pedidas y almacene el conjunto de huéspedes y habitaciones.

3. Sobrecargar el operador de inserción de flujo (ej. <<) en un objeto de tipo `std::ostream`. Este operador debe "imprimir" todos los datos de las distintos datatypes de `DtReserva` (`DtReservaIndividual`, `DtReservaGrupal`), siguiendo un formato similar al ejemplo:

TipoReserva: Individual/Grupal

FechaCheckIn: *dd/mm/aaaa*

FechaCheckOut: *dd/mm/aaaa*

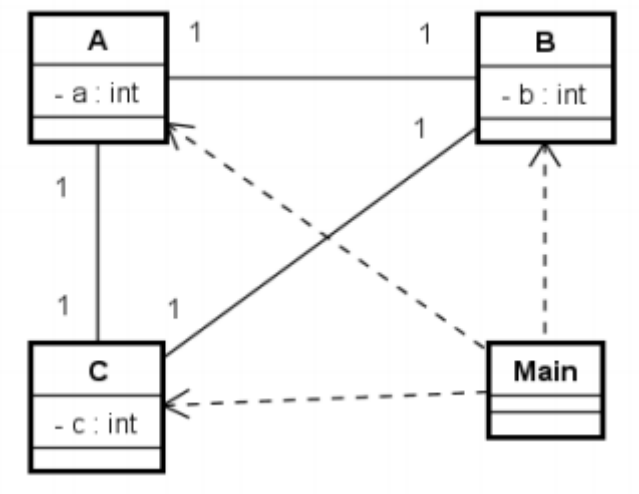
Habitación: *N*
Costo: *$ x,x*
/* Solo para reservas individuales */
Pagado: *Si/No*
/* Solo para reservas grupales*/
Huéspedes: *nombre1 - email1 - es Finger,*
nombre2 - email2,
...

4. Implementar un menú sencillo e interactivo con el usuario para probar las funcionalidades requeridas en los puntos anteriores. Al ejecutar el programa debe primero pedir el ingreso de un número especificando la acción a realizar, y luego pedir los datos necesarios para cada operación. Se pide la siguiente estructura para el menú.

1. Agregar Huesped
2. Agregar Habitación
3. Obtener Huéspedes
4. Obtener Habitaciones
5. Registrar Reserva
6. Obtener Reservas
7. Salir

Ejercicio 2

En este ejercicio se busca que el estudiante se familiarice con la problemática de las dependencias circulares en C++. Para ello, considere la estructura dada por el siguiente diagrama.



Se pide

1. Implementar en C++ y compilar la estructura dada en el diagrama. Incluya en cada una de las clases A, B y C, una operación llamada `printInt()` que muestre en salida estándar el entero que corresponda. Implemente un método `main` que defina un objeto de cada una de las clases e invoque al `printInt()` de cada uno de ellos.

2. Responder las siguientes preguntas:
    a. ¿Cuáles son las dependencias circulares que fueron necesarias solucionar para que el programa compile?
    b. ¿Qué es una *forward declaration*?

Referencias

[1] EVA Programación 4. URL: https://eva.fing.edu.uy/course/view.php?id=413

[2] C++. URL: https://www.cplusplus.com/

[3] Unidad de Recursos Informáticos, Configuraciones.
URL: https://www.fing.edu.uy/sysadmin/configuraciones

[4] Unidad de Recursos Informáticos, Gestión Estudiantil.
URL: https://www.fing.edu.uy/sysadmin/gestion-estudiantil