Práctica 1

Formato y fecha de entrega

Hay que entregar la Práctica antes del día **9 de Diciembre a las 23:59h**. Para la entrega será necesario que entreguéis un fichero en formato **ZIP** de nombre ***logincampus*_pr1** en minúsculas (donde *logincampus* es el nombre de usuario con el qué hacéis login en Campus). El ZIP tiene que contener:

- Workspace CodeLite entero, con todos los ficheros que se piden.

Para reducir el tamaño de los ficheros y evitar problemas de envío que se pueden dar al incluir ejecutables, hay que eliminar el que genera el compilador. Podéis utilizar la opción "Clean" del workspace o eliminarlos directamente (las subcarpetas Menú y Test son las que contienen todos los binarios que genera el compilador).

Hay que hacer la entrega en el apartado de entregas de EC del aula de teoría.

Los estudiantes que participáis en la prueba piloto PROxA y trabajáis con el entorno Vocareum (Aula Isabel Domènech) lo tenéis que entregar directamente en el laboratorio virtual Vocareum haciendo click en el botón ***submit*** de vuestro workspace de la PA. Se podrá hacer ***submit*** tantas veces como se quiera. Sólo se corregirá la última versión entregada.

Presentación

En esta práctica se presenta el caso en que trabajaremos durante este semestre. Habrá que utilizar el que habéis trabajado en las primeras PECs, también la composición iterativa y la utilización de strings. También tendréis que empezar a poner en juego una competencia básica para un programador: la capacidad de entender un código ya dado y saberlo adaptar a las necesidades de nuestro problema. Con esta finalidad, se os facilita gran parte del código, en el cual hay métodos muy similares a los que se os piden. Se trata de aprender mediante ejemplos, una habilidad muy necesaria cuando se está programando.

Competencias

Transversales

- Capacidad de comunicación en lengua extranjera.

Específicas

- Capacidad de diseñar y construir aplicaciones informáticas mediante técnicas de desarrollo, integración y reutilización.
- Conocimientos básicos sobre el uso y la programación de los ordenadores, sistemas operativos, y programas informáticos con aplicación a la ingeniería.

Objetivos

- Practicar los conceptos estudiados a la asignatura

- Profundizar en el uso de un IDE, en este caso CodeLite
- Analizar un enunciado y extraer los requerimientos tanto de tipos de datos como funcionales (algoritmos)
- Aplicar correctamente la composición alternativa cuando haga falta
- Aplicar correctamente la composición iterativa cuando haga falta
- Utilizar correctamente tipos de datos estructurados
- Utilizar correctamente el tipo de datos Tabla
- Aplicar correctamente el concepto de Modularidad
- Aplicar correctamente los esquemas de búsqueda y recorrido
- Analizar, entender y modificar adecuadamente código existente
- Hacer pruebas de los algoritmos implementados

Recursos

Para realizar esta actividad tenéis a vuestra disposición los siguientes recursos:

- Materiales en formato web de la asignatura
- Laboratorio de C

Criterios de valoración

Cada ejercicio tiene asociada la puntuación porcentual sobre el total de la actividad. Se valorará tanto la corrección de las respuestas como que sean completas.

- Los ejercicios en lenguaje C, tienen que compilar para ser evaluados. En tal caso, se valorará:
  - Que funcionen correctamente
  - Que se respeten los criterios de estilo (Véase la Guía de estilo de programación en C que tenéis a la Wiki)
  - Que el código esté comentado (preferiblemente en inglés)
  - Que las estructuras utilizadas sean las adecuadas

Descripción del proyecto

Este semestre nos han pedido crear una aplicación para gestionar un compañía de reservas hoteleras. Hará falta, pues, gestionar los hoteles, los clientes y las reservas. En las PEC's habéis ido definiendo ciertas variables y programando pequeños algoritmos relacionados con esta aplicación. En el código que os proporcionamos como base para la realización de esta práctica podréis encontrar estas variables ya conocidas de las PEC's y algunos de los algoritmos.

Recordad, por ejemplo, como definisteis variables para guardar los datos de un hotel.

Ahora en esta práctica os pediremos que implementéis la gestión de los clientes. Las acciones que querremos tener programadas serán las siguientes:

- Leer, mediante un menú, los datos correspondientes a un cliente.
- Cargar los datos desde ficheros y también almacenarlos.
- Hacer búsquedas, aplicar filtros y obtener datos estadísticos.

Además, a pesar de que esta primera versión será una aplicación en línea de comandos, queremos que todas estas funcionalidades queden recogidas en una **API** (*Application Programming Interface*), lo que nos permitirá en un futuro poder utilizar esta aplicación en diferentes dispositivos (interfaces gráficas, teléfonos móviles, tabletas, web, ...).

Estructuración del código

Junto con el enunciado se os facilita un proyecto Codelite que será el esqueleto de la solución (los usuarios de Vocareum encontraréis la estructura de carpetas y el código en el propio entorno).

A continuación se dan algunas indicaciones del código que os hemos dado:

main.c: Contiene el inicio del programa. Está preparado para funcionar con dos modos diferentes:

- **Menú**: Muestra un menú que permite al usuario gestionar los datos.
- **Test**: se ejecutan un conjunto de pruebas sobre el código para asegurar que funciona. Inicialmente muchas de estas pruebas fallarán pero, una vez realizados los ejercicios, todas tendrían que pasar correctamente.

Si se ejecuta normalmente, la aplicación muestra el menú. Para que funcione en modo test, hay que pasar el parámetro "-t" a la aplicación. Tal como se ha configurado el proyecto de Codelite, si lo ejecutáis en modo Test (seleccionando "Test" en el desplegable "workspace build configuration") se ejecutarán los tests, y si lo hacéis en modo Menu, veréis el menú.

Los usuarios de Vocareum veréis como el código, cuando se ejecuta, pide un valor que puede ser T (de Test) o M (de Menú).

data.h: Se definen los tipos de datos que se utilizan a la aplicación. A pesar de que se podrían haber separado en los diferentes ficheros de cabecera, se han agrupado todos para facilitar la lectura del código.

customer.h / customer.c: contienen todo el código que gestiona los clientes.

hotel.h / hotel.c: contienen el código que gestiona los hoteles.

menu.h / menu.c: contienen todo el código para gestionar el menú de opciones que aparece cuando se ejecuta en modo Menu.

api.h / api.c: contienen los métodos (acciones y funciones) públicos de la aplicación, el que sería el API de nuestra aplicación. Estos métodos son los que se llaman desde el menú de opciones y los que utilizaría cualquier otra aplicación que quisiera utilizar el código.

test.h / test.c: contienen todas las pruebas que se pasan al código cuando se ejecuta en modo Test.

La mayoría de los ejercicios referencian acciones y funciones ya existentes al código proporcionado que son muy similares a las que se os piden. Analizadlas y utilizadlas como base.

Enunciados

[10%] Ejercicio 1: Definición de datos

Tal como hemos comentado en la presentación de la práctica, una de las estructuras de información que usaremos es la del cliente. Utilizaremos un tipo ***tCustomer*** que está parcialmente definido en el fichero **data.h**:

- id: Identificador del cliente (valor entero)
- name: Es el nombre de pila del cliente (cadena de como máximo 15 caracteres alfanuméricos y sin espacios)

Y en este ejercicio lo tenemos que completar con los siguientes campos:

- surname: es una cadena de, como máximo, 25 caracteres (sin espacios) que indica el primer apellido del cliente
- docNumber: es el número de documento de identidad, que será una cadena de caracteres alfanuméricos de, como mucho, 20 posiciones.
- birthDate: fecha de nacimiento del cliente. Tendréis que apoyaros en una estructura auxiliar tDate que incluya los valores de año, mes y día de nacimiento (los tres, enteros)
- birthCity: es una cadena de, como máximo, 25 caracteres (sin espacios) que indica la ciudad de nacimiento del cliente
- status: estado civil del cliente. Será uno de los valores siguientes: SINGLE, MARRIED, SEPARATED, DIVORCED, WIDOWED.
- profile: es el perfil del cliente, según la categorización que ha hecho la compañía sobre el historial de reservas pasadas. Podrá ser uno de los valores siguientes: FAMILIAR, CONSERVATIVE, ADVENTUROUS, LONELY, CURIOUS, INNOVATIVE, DARED.
- bookingsCounter: es un contador (entero positivo) de reservas del cliente
- points: es un contador (entero positivo) de todos los puntos acumulados por el cliente, obtenidos a partir de las diferentes reservas que ha ido realizando a lo largo del tiempo.
- satisfaction: es un valor real que representa un porcentaje. Es el valor promedio de las valoraciones que ha hecho el cliente de sus reservas.

Nota: Como estamos definiendo la estructura de datos, que es la base de la práctica, tenéis que empezar obligatoriamente por este ejercicio. Es importante que respetáis la orden indicado de los campos y su nomenclatura. Una vez hecho esto, descomentad todas las líneas de código comentadas en bloques marcados con "Uncomment to test" antes de continuar con el resto de ejercicios. Encontraréis las líneas a descomentar en los ficheros **test.c** y **customer.c**.

[10%] Ejercicio 2: Entrada interactiva de datos

Cuando trabajamos con aplicaciones que utilizan la línea de pedidos para comunicarse con el usuario, a menudo se hace necesario utilizar menús de opciones. El programa

muestra la lista de opciones identificadas por un número al usuario, de forma que el usuario pueda elegir la opción que le interesa introduciendo por teclado este identificador.

Completad la implementación de la acción ***readCustomer***, que encontraréis en el archivo **menu.c**, que permite dar de alta de manera interactiva un nuevo cliente. Esta acción es llamada desde la opción interactiva de añadir un nuevo cliente “2) ADD CUSTOMER”. Para acceder hay que elegir primero “3) MANAGE CUSTOMERS” del menú principal de la aplicación. Podéis usar estas opciones para comprobar que la entrada interactiva de datos funciona correctamente.

Los datos que hay que pedir por cada cliente son: id, name, surname, docNumber, birthDate (3 enteros), birthCity, civilState y profile. Los campos bookingsCounter, points y satisfaction no se pedirán por teclado sino que se inicializarán a cero.

Habrá que comprobar si los valores introducidos están dentro de los límites especificados al ejercicio 1 y, si es así, hará falta que el parámetro de salida de la acción retVal contenga el valor OK y, si no, el valor ERROR. Tomad como modelo la lectura de datos de hoteles que se hace a la acción **readHotel** del mismo archivo menu.c.

Notad que este ejercicio sólo se puede probar interactivamente dado que se trata precisamente de implementar la entrada / salida por teclado y pantalla. Por este motivo, no se proporcionan tests asociados a este ejercicio.

[15%] Ejercicio 3: Comparación de la estructura cliente

Un problema que nos encontramos con los tipos estructurados es que muchos de los operadores que tenemos definidos con los tipos básicos de datos, como por ejemplo los de comparación (==, !=, <, >, ...) o el de asignación (=), no funcionan para los nuevos tipos que nos creamos.

Por este motivo a menudo se hace necesario definir métodos que nos den estas funcionalidades. Por ejemplo, ya hemos visto que para asignar una cadena de caracteres no lo hacemos con el operador de asignación normal (=), sino que tenemos que recurrir a la función **strcpy**. El mismo pasa en las comparaciones, donde en vez de utilizar los operadores normales (==, !=, <, >, ...) utilizamos **strcmp**.

Se pide:

a) [10%] Completad, en el fichero **customer.c**, la función ***strcmpUpper*** que compara dos strings (s1 y s2) pasados por parámetro pasándolos, previamente, a mayúsculas. La función no tiene que alterar los parámetros de entrada. Para hacerlo, habrá que declarar variables locales de tipos string, copiar el contenido de los parámetros de entrada, transformar a mayúsculas y, finalmente, comparar los strings resultantes. La función tendrá que devolver:

-1 si s1<s2 0 si s1=s2 1 si s1>s2

Nota: Apoyaos en el método **strcmp** de la librería **string.h** y también en la función **convertToUpper** que encontraréis **a customer.c.**

b) [5%] Completad, en el fichero **customer.c**, la función ***customer_cmp*** que dados dos clientes c1 y c2, nos devuelve:

-1 si c1<c2 0 si c1=c2 1 si c1>c2

El orden de los clientes vendrá dado por el valor de sus campos con la prioridad:

1. Nombre (ascendente)
2. Apellido (ascendente)
3. Número de documento (ascendente)
4. Fecha de nacimiento (ascendente)
5. Ciudad de nacimiento (ascendente)
6. Estado civil (ascendente)
7. Perfil (ascendente)
8. Contador de reservas (ascendente)
9. Puntos acumulados (ascendente)
10. Satisfacción (ascendente)

Esto quiere decir, que si el nombre de d1 es “Gerard” y el de d2 “Manel", consideraremos que d1<d2. En caso de que sean iguales, se comprobará el apellido para desempatar.

Si también son iguales, habrá que comprobar el número de documento, y así sucesivamente... Si todos los datos son iguales, indicará que d1=d2.

Nota: Se pide que la comparación de clientes ignore las diferencias de mayúsculas y minúsculas que pueda haber entre cualquier campo de tipo string. Por lo tanto, apoyaos en la función **strcmpUpper** del apartado anterior cuando convenga. Disponéis también la función de comparación de fechas **date_cmp** en customer.c.

[15%] Ejercicio 4: Filtros basados en expresiones

Por medio de expresiones podemos construir condiciones que nos permitan seleccionar determinados elementos dentro de un conjunto más grande. En el caso de los clientes, este conjunto podría llegar a ser muy grande y a menudo será necesario recurrir a determinados filtrajes para focalizar la atención en un colectivo más pequeño.

Por ejemplo, desde el departamento de marketing, se nos puede pedir filtrar hoteles y clientes para poder orientar, de manera más efectiva, varias campañas y promociones, así como obtener listados de clientes que cumplen determinados niveles de satisfacción con finalidades comerciales.

Con este objetivo, se pide:

a) [10%] Completad la función ***customerTable_selectCustomers*** en el fichero **customer.c** que, dada una tabla de clientes y una ciudad, seleccione aquellos clientes susceptibles de escoger hoteles de tipo urbano en esta ciudad. Desde el departamento de marketing de UOCBookings se ha determinado que los clientes objetivo de este tipo de hoteles son, mayoritariamente, los que tienen una edad comprendida entre 25 y 35 años (ambos incluidos), que están en estado SINGLE y que tienen un perfil de tipo LONELY, CURIOUS o INNOVATIVE. Se excluirán los clientes que hayan nacido en la ciudad pasada por parámetro.

Nota: Necesitaréis utilizar la función auxiliar **calculateAge** de customer.c que, a partir de una fecha de nacimiento, calcula la edad respeto una fecha "actual" (que estableceremos en el día 31 de diciembre de 2019).

b) [5%] Completad la acción ***customerTable_selectSatisfiedCustomers*** del fichero **customer.c** que permite seleccionar aquellos clientes con un índice de satisfacción superior al 85%. Sólo se tendrán en cuenta aquellos clientes que hayan hecho un número significativo de reservas (estableceremos este umbral en un mínimo de 10 reservas).

[20%] Ejercicio 5: Combinación de filtros

Hacer acciones y funciones que aplican determinados filtros puede ser una herramienta útil en sí misma, pero también como “piezas” de determinadas búsquedas y filtros más complejos. En este ejercicio os pediremos que combinéis filtros existentes en otros ejercicios y apartados, tanto de la práctica como de PEC’s anteriores.

Con este objetivo, se pide:

a) [10%] Implementar en **hotel.c** la acción ***customerRecommendation***

que dado un cliente y una tabla de hoteles devuelva la lista de hoteles recomendados en una ciudad dada y con una determinada puntuación mínima. Para hacerlo, invocad a la función *hotelTable_select* de la PEC8 con un precio objetivo, distancia deseada y puntuación mínima (que, por lo tanto, también serán parámetros de customerRecommendation). Evitad recomendar hoteles en la misma ciudad de nacimiento del cliente.

b) [10%] Implementar en menu.c la acción ***allCustomersRecommendation***

que, a partir de una estructura tAppData, una ciudad, precio, distancia y puntos, muestre por pantalla, por cada cliente, la lista de hoteles recomendados para él. Para hacerlo, llamad iterativamente a la función customerRecommendation del apartado anterior con cada uno de los clientes devueltos por la función selectCustomers del ejercicio 4a.

Nota: Para mostrar un cliente por pantalla podéis usar la acción *printCustomer*, y, para mostrar un hotel, lo podéis hacer con *printHotel*.

[20%] Ejercicio 6: Cálculos estadísticos

Tenemos todos los clientes guardados en una tabla de tipo tCustomerTable y la aplicación ya nos permite gestionarlos (añadir, eliminar, etc). En este ejercicio nos interesará obtener algunos datos estadísticos relativos a ellos.

Se pide implementar al fichero **customer.c**, utilizando las acciones que necesitáis de este mismo fichero:

a) [10%] La función ***customerTable_getAvgPointsPerBooking***

   Que, dada una tabla de clientes calcule el número de puntos promedio que un cliente consigue en cada reserva. Para hacerlo, acumulad el número de reservas totales y el número de puntos totales antes de hacer el cálculo.

b) [10%] La acción ***customerTable_getMaxSatisfactionPerAgeInterval***

   Que, dada una tabla de clientes, calcule la satisfacción máxima para los siguientes intervalos de edad: menos de 30 años, entre 30 y 44, entre 45 y 59 y 60 años o más. Devolved la información por medio de parámetros de salida. Si, para una determinada franja de edad no hay datos que permitan hacer el cálculo, devolved un valor cero.

[10%] Ejercicio 7: Operaciones con tablas

Se pide que implementéis la acción **customerTable_del** del fichero **customer.c**, que dada una tabla de clientes y un cliente, borre de la tabla el cliente pasado por parámetro.

Tened presente que el borrado del cliente no tiene que dejar ninguna posición inválida en la tabla. Por lo tanto, hará falta que ocupéis el espacio liberado por el cliente desplazando una posición atrás todos los clientes que había después del cliente borrado.

Nota: Para facilitaros la solución, utilizad la función **customerTable_find** de customer.c para localizar el índice del cliente a borrar.