

Pràctica 1

Format i data de lliurament

Cal lliurar la Pràctica abans del dia **13 de Maig a les 23:59**. Per al lliurament caldrà que entregueu un fitxer en format **ZIP** de nom **logincampus_pr1** en minúscules (on *logincampus* és el nom d'usuari amb el què feu login al Campus). El ZIP ha de contenir:

- Workspace CodeLite sencer, amb tots els fitxers que es demanen.

Per reduir la mida dels fitxers i evitar problemes d'enviament que es poden donar en incloure executables, cal eliminar el que genera el compilador. Podeu utilitzar l'opció "Clean" del workspace o eliminar-los directament (les subcarpetes Menu i Test són les que contenen tots els binaris que genera el compilador).

Cal fer el lliurament a l'apartat de lliuraments d'AC de l'aula de teoria.

Presentació

En aquesta pràctica es presenta el cas en què treballarem durant aquest semestre. Caldrà utilitzar el que heu treballat a les primeres PACs, també la composició iterativa i la utilització de *strings*. També haureu de començar a posar en joc una competència bàsica per a un programador: la capacitat d'entendre un codi ja donat i saber-lo adaptar a les necessitats del nostre problema. Amb aquesta finalitat, se us facilita gran part del codi, en el qual hi ha mètodes molt similars als que se us demanen. Es tracta d'aprendre mitjançant exemples, una habilitat molt necessària quan s'està programant.

Competències

Transversals

- Capacitat de comunicació en llengua estrangera.

Específiques

- Capacitat de dissenyar i construir aplicacions informàtiques mitjançant tècniques de desenvolupament, integració i reutilització.
- Coneixements bàsics sobre l'ús i la programació dels ordinadors, sistemes operatius, i programes informàtics amb aplicació a l'enginyeria.

Objectius

- Practicar els conceptes estudiats a l'assignatura
- Aprofundir en l'ús d'un IDE, en aquest cas CodeLite
- Analitzar un enunciat i extreure'n els requeriments tant de tipus de dades com funcionals (algorismes)
- Aplicar correctament la composició alternativa quan calgui
- Aplicar correctament la composició iterativa quan calgui
- Utilitzar correctament tipus de dades estructurats
- Utilitzar correctament el tipus de dades Taula
- Aplicar correctament el concepte de Modularitat
- Aplicar correctament els esquemes de cerca i recorregut
- Analitzar, entendre i modificar adequadament codi existent
- Fer proves dels algorismes implementats

Recursos

Per realitzar aquesta activitat teniu a la vostra disposició els següents recursos:

- Materials en format web de l'assignatura
- Laboratori de C

Criteris de valoració

Cada exercici té associada la puntuació percentual sobre el total de l'activitat. Es valorarà tant la correctesa de les respostes com la seva completesa.

- Els exercicis en llenguatge C, han de compilar per ser avaluats. En tal cas, es valorarà:
 - Que funcionin correctament
 - Que es respectin els criteris d'estil (Vegeu la Guia d'estil de programació en C que teniu a la Wiki)
 - Que el codi estigui comentat (preferiblement en anglès)
 - Que les estructures utilitzades siguin les adequades

Descripció del projecte

Aquest semestre ens han demanat crear una aplicació per a gestionar un companyia de transport aeri. Caldrà doncs gestionar els avions, els passatgers i els vols. A les PACs heu anat definint certes variables i programant petits algorismes relacionats amb aquesta aplicació. En el codi que us proporcionem com a base per a la realització d'aquesta pràctica podreu trobar aquestes variables ja conegudes de les PACs i alguns dels algorismes.

Recordeu, per exemple, com vau definir variables per emmagatzemar dades d'un avió.

Ara en aquesta pràctica us demanarem que implementeu la gestió dels passatgers. Les accions que voldrem tenir programades seran les següents:

- Llegir, mitjançant un menú, les dades corresponents a un passatger.
- Carregar les dades des de fitxers i també emmagatzemar-les.
- Fer cerques, aplicar filtres i obtenir dades estadístiques.

A més a més, tot i que aquesta primera versió serà una aplicació en línia de comandes, volem que totes aquestes funcionalitats quedin recollides en una **API** (*Application Programming Interface*), el que ens permetrà en un futur poder utilitzar aquesta aplicació en diferents dispositius (interfícies gràfiques, telèfons mòbils, tauletes, web, ...).

Estructuració del codi

Juntament amb l'enunciat se us facilita un projecte Codelite que serà l'esquelet de la solució. A continuació es donen algunes indicacions del codi que us hem donat:

main.c: Conté l'inici del programa. Està preparat per funcionar amb dos modes diferents:

- **Menu:** Mostra un menú que permet a l'usuari gestionar les dades.
- **Test:** s'executen un conjunt de proves sobre el codi per assegurar que funciona. Inicialment moltes d'aquestes proves fallaran però, un cop realitzats tots els exercicis, totes haurien de passar correctament.

Si s'executa normalment, l'aplicació mostra el menú. Per tal que funcioni en mode test, cal passar el paràmetre "-t" a l'aplicació. Tal com s'ha configurat el projecte de Codelite, si l'executeu en mode Test (seleccionant "Test" al desplegable "workspace build configuration") s'executaran els tests, i si ho feu en mode Menu, veureu el menú.

data.h: S'hi defineixen els tipus de dades que s'utilitzen a l'aplicació. Tot i que es podrien haver separat en els diferents fitxers de capçalera, s'han agrupat tots per facilitar la lectura del codi.

passenger.h/passenger.c: contenen tot el codi que gestiona els passatgers.

plane.h/plane.c: contenen el codi que gestiona els avions.

menu.h/menu.c: contenen tot el codi per gestionar el menú d'opcions que apareix quan s'executa en mode Menu.

api.h/api.c: contenen els mètodes (accions i funcions) públics de l'aplicació, el que seria l'API de la nostra aplicació. Aquests mètodes són els que es criden des del menú d'opcions i els que utilitzaria qualsevol altra aplicació que volgués utilitzar el codi.

test.h/test.c: contenen totes les proves que es passen al codi quan s'executa en mode Test.

La majoria dels exercicis referencien accions i funcions ja existents al codi proporcionat que són molt similars a les que se us demanen. Analitzeu-les i utilitzeu-les com a base.

Enunciats

[10%] Exercici 1: Definició de dades

Tal com hem comentat en la presentació de la pràctica, una de les estructures d'informació que farem servir és la del passatger. Utilitzarem un tipus **tPassenger** que està parcialment definit en el fitxer **data.h**:

- id: Identificador del passatger (valor enter)
- nom: És el nom de pila del passatger (cadena de com a màxim 15 caràcters alfanumèrics i sense espais)

I en aquest exercici l'hem de completar amb els següents camps:

- surname: és una cadena de, com a màxim, 25 caràcters (sense espais) que indica el primer cognom del passatger
- docType: és el tipus de document identificatiu (tipus enumerat tDocumentType) que podrà ser un dels següents valors: NIF, NIE, PASSPORT, DRIVING_LICENSE i un valor genèric que servirà per designar altres documents (OTHER_DOCUMENTS).
- docNumber: és el número de document, que serà una cadena de caràcters alfanumèrics de, com a molt, 20 posicions.
- birthDate: data de naixement del passatger. Haureu de recolzar-vos en una estructura auxiliar tDate que inclogui els valors d'any, mes i dia de naixement (tots tres, enters)
- countryISOCODE: és el país de naixement del passatger, una cadena de dos caràcters que representen un país segons codificació ISO
- customerCardHolder: és un booleà que indica si aquest passatge posseeix la tarja de fidelització de la companyia aèria.
- cardNumber: és el número de targeta de fidelització del passatger (enter positiu). Aquest camp només tindrà un valor diferent de zero en cas que customerCardHolder tingui valor CERT.
- fidelityPoints: és el nombre de punts de la tarja de fidelització (enter positiu). Aquest camp només tindrà un valor diferent de zero en cas que customerCardHolder tingui valor CERT.

[20%] Exercici 2: Entrada interactiva de dades

Quan treballem amb aplicacions que utilitzen la línia de comandes per comunicar-se amb l'usuari, sovint es fa necessari utilitzar menús d'opcions. El programa mostra la llista d'opcions identificades per un número a l'usuari, de forma que l'usuari pugui triar l'opció que li interessa introduint per teclat aquest identificador.

Completeu la implementació de l'acció **readPassenger**, que trobareu a l'arxiu **menu.c**, que permet donar d'alta de manera interactiva un nou passatger. Aquesta acció és cridada des de l'opció interactiva d'afegir un nou passatger "2) ADD PASSENGER". Per accedir-hi cal triar primer "3) MANAGE PASSENGERS" del menú principal de l'aplicació. Podeu fer servir aquestes opcions per comprovar que l'entrada interactiva de dades funciona correctament.

Les dades que cal demanar per cada passatger són: id, name, surname, docType, docNumber, birthDate (tres enters), countryISOCode i customerCardHolder.

A més, en cas que customerCardHolder sigui CERT, caldrà llegir el número de tarja (cardNumber) i el nombre de punts obtinguts fins ara (fidelityPoints). En cas contrari, ambdós valors esmentats s'hauran d'inicialitzar a zero.

Caldrà comprovar si els valors introduïts estan dins dels límits especificats a l'exercici 1 i, si és així, caldrà que el paràmetre de sortida de l'acció retVal contingui el valor OK i, si no, el valor ERROR. Preneu com a model la lectura de dades d'avions que es fa a l'acció **readPlane** del mateix arxiu menu.c.

[20%] Exercici 3: Serialització i deserialització de l'estructura passatger

Els tipus estructurats de dades són una bona eina per a representar objectes complexos com ara passatgers i avions. No obstant, sovint interessa obtenir una representació més textual d'aquests objectes, per exemple, per poder mostrar-la a l'usuari quan convingui, o llegir-la o escriure-la en un fitxer. L'acció de convertir en text unes dades d'una estructura l'anomenem serialitzar. L'acció contrària, convertir una cadena de text en una estructura de dades, l'anomenem deserialitzar.

Les dades d'un passatger ens arribaran (o les escriurem) separades per un únic espai i en el mateix ordre que apareixen a l'enunciat de l'exercici 1. Tots els reals tindran 2 decimals.

- a) Completeu al fitxer **passenger.c** l'acció **getPassengerStr** que permet passar una estructura de tipus **tPassenger** a una cadena de caràcters. Observeu com s'ha codificat l'acció **getPlaneStr** al fitxer **plane.c**
- b) Completeu al fitxer **passenger.c** l'acció **getPassengerObject** que permet passar d'una cadena de caràcters a una estructura **tPassenger**. Observeu com s'ha codificat l'acció **getPlaneObject** al fitxer **plane.c**

[20%] Exercici 4: Operadors de l'estructura passatger

Un problema que ens trobem amb els tipus estructurats és que molts dels operadors que tenim definits amb els tipus bàsics de dades, com ara els de comparació (`==`, `!=`, `<`, `>`, ...) o el d'assignació (`=`), no funcionen per als nous tipus que ens creem.

Per aquest motiu sovint es fa necessari definir mètodes que ens donin aquestes funcionalitats. Per exemple, ja hem vist que per assignar una cadena de caràcters no ho fem amb l'operador d'assignació normal (`=`), sinó que hem de recórrer a la funció **strcpy**. El mateix passa en les comparacions, on en comptes d'utilitzar els operadors normals (`==`, `!=`, `<`, `>`, ...) utilitzem **strcmp**.

Es demana:

- a) Completeu al fitxer **passenger.c** l'acció **passenger_cpy** que permet copiar totes les dades d'una estructura `tPassenger` a una altra. Observeu com s'ha codificat l'acció `plane_cpy` al fitxer `plane.c`
- b) Completeu al fitxer **passenger.c** la funció **passenger_cmp** que donats dos passatgers `d1` i `d2`, ens retorna:

-1 si `d1 < d2` 0 si `d1 == d2` 1 si `d1 > d2`

L'ordre dels passatgers vindrà donat pel valor dels seus camps en l'ordre de prioritats següent:

1. Nom (ascendent)
2. Cognom (ascendent)
3. Tipus de document (ascendent)
4. Número de document (ascendent)
5. Data de naixement (ascendent)
6. País (ascendent)
7. Si és posseïdor de la tarja de fidelització (ascendent)
8. Número de targeta (ascendent)
9. Punts de la targeta (ascendent)

Això vol dir, que si el nom de `d1` és "Albert" i el de `d2` "Pau", considerarem que `d1 < d2`. En cas que els noms siguin iguals, caldrà comprovar el cognom per desempatar. Si els cognoms també són iguals, caldrà comprovar el tipus de document, i així successivament... Si totes les dades són iguals, indicarà que `d1 == d2`.

Nota: Per tal de fer aquest exercici podeu utilitzar el mètode **strcmp** de la llibreria **string.h**. Observeu com s'ha codificat l'acció `plane_cmp` al fitxer **plane.c**.

[10%] Exercici 5: Persistència de dades

Per tal que les dades de passatgers i avions amb les que hem estat treballant no es perdin quan finalitza l'execució del programa (que siguin persistents) hi ha diversos mecanismes. Una estratègia força utilitzada consisteix a obtenir-ne una representació textual (la serialització que ja hem vist a l'exercici 3) i escriure-les en un fitxer de text, una estructura a cada línia. El pas invers, la càrrega de les dades des d'un fitxer de text en una execució posterior del programa, requereix de la deserialització (tornar a obtenir l'estructura de dades original a partir de cada línia del fitxer).

a) Implementar a `passenger.c` l'acció **`passengerTable_save`**

que donada una taula de passatgers i un nom de fitxer, guardi tots els passatgers de la taula en format textual en el fitxer. Cal utilitzar en el seu codi l'acció **`getPassengerStr`** per serialitzar cada estructura passatger de la taula i escriure cada cadena de caràcters en una línia nova del fitxer. El paràmetre de sortida `retVal` contindrà un valor **`OK`** si ha pogut guardar les dades o un valor **`ERR_CANNOT_WRITE`** en cas que, per algun motiu, no es pugui crear el fitxer de sortida. Podeu utilitzar com a exemple la implementació de l'acció **`planesTable_save`** del fitxer `plane.c`.

b) Implementar a `passenger.c` l'acció **`passengerTable_load`**

que donada una taula de passatgers i un nom de fitxer, llegeixi totes les dades de passatgers en format text del fitxer i les afegixi a la taula. Cal que feu servir la funció **`getPassengerObject`** per desserialitzar cadascuna de les línies del fitxer i convertir-les en estructures de tipus passatgers que es puguin afegir a la taula. El paràmetre de sortida `retVal` contindrà un valor **`OK`** si ha pogut guardar les dades o un valor **`ERR_CANNOT_READ`** en cas que, per algun motiu, no es pugui obrir el fitxer d'entrada. Podeu utilitzar com a exemple la implementació de l'acció **`planesTable_load`** del fitxer `plane.c`.

[10%] Exercici 6: Filtre de passatgers

Tenim tots els passatgers guardats en una taula de tipus `tPassengerTable` i l'aplicació ja ens permet gestionar-los (afegir-ne, eliminar-ne, etc). En aquest exercici ens interessarà trobar els passatgers que compleixin unes certes condicions. Donat que no coneixem el nombre de passatgers que compliran les condicions, utilitzarem una taula de passatgers `tPassengerTable` per retornar els resultats de les cerques.

Es demana implementar al fitxer **passenger.c**, utilitzant les accions que necessiteu d'aquest mateix fitxer:

a) L'acció ***passengerTable_filterByCountry***

Que, donada una taula de passatgers i un codi de país ISO, ens retorni tots els passatgers d'aquest país.

b) L'acció ***passengerTable_filterByBirthDateInterval***

Que, donada una data inicial i una data final, retorni tots els passatgers amb data de naixement dins aquests dos límits.

[10%] Exercici 7: Càlculs estadístics de passatgers

En aquest exercici ens interessarà obtenir informació estadística de la taula de passatgers.

Es demana que implementeu al fitxer **passenger.c** les funcions:

- a) La funció ***passengerTable_getNumberOfCardHolders***

Que, donada una taula de passatgers, ens retorni el nombre de passatgers que tinguin targeta de fidelització de la companyia.

- b) La funció ***passengerTable_getMaxFidelityPoints***

Que, donada una taula de passatgers, ens retorni el nombre màxim de punts que té un passatger de la companyia. Tingueu present que només cal contemplar el nombre de punts d'aquells passatgers que siguin posseïdors de la tarja de fidelització.