PR 2 - 20251

Piles, cues i recursivitat

Universitat Oberta
de Catalunya

Informació rellevant:

- Data límit de lliurament: 05/12/2025.
- Pes en la nota de PRs: 30%.

Contingut

Informació docent

Aquesta pràctica està vinculada a les seccions ***04. Disseny descendent***, ***05. Recursivitat I***, ***06. Recursivitat II***, ***07. TAD en memòria dinàmica*** i ***08. TAD de TAD*** dels recursos d'aprenentatge de l'assignatura.

Prerequisits

Per realitzar aquesta pràctica necessites:

- Tenir instal·lat i saber utilitzar un entorn de desenvolupament integrat (IDE), preferiblement CodeLite, per crear, compilar, executar i depurar aplicacions en C.
- Haver llegit els materials de l'assignatura relacionats amb el disseny descendent, la recursivitat i la gestió de la memòria dinàmica en TAD i estructures TAD de TAD.

Objectius

Amb aquesta pràctica, l'equip docent de l'assignatura busca que:

- Interpretar i seguir el codi d'altres persones.
- Compilar projectes de codi organitzats en carpetes i biblioteques.
- Desenvolupar un projecte de codi a partir de la seva especificació.

Resultats d'aprenentatge

Amb aquesta pràctica has de demostrar que ets capaç de:

- Configurar i compilar projectes modulars en C utilitzant una biblioteca estàtica i una aplicació executable.
- Dissenyar, implementar i utilitzar tipus abstractes de dades (TAD) i estructures dinàmiques en C.
- Dissenyar, implementar i utilitzar funcions recursives en C.
- Desenvolupar una aplicació funcional i eficient que implementi múltiples nivells de disseny descendent en C.
- Utilitzar les eines bàsiques de l'entorn de desenvolupament integrat (IDE) com el terminal i el depurador.

Avís

Es recorda que **està totalment prohibit copiar a les PAC i pràctiques** de l'assignatura. S'entén que pot haver-hi treball o comunicació entre els estudiants durant la realització de l'activitat, però el lliurament d'aquesta ha de ser individual i diferenciat de la resta. Els lliuraments seran analitzats amb **eines de detecció de plagi**.

Així doncs, els lliuraments que continguin alguna part idèntica respecte a lliuraments d'altres estudiants seran considerats còpies, i tots els implicats (sense que sigui rellevant el vincle existent entre ells) suspendran l'activitat lliurada.

En cas de voler referenciar o citar un text, s'ha de fer seguint els estàndards definits en els següents recursos:

- Guia de citació: https://biblioteca.uoc.edu/ca/continguts/Com-citar/index.html.
- Monogràfic sobre plagi: http://biblioteca.uoc.edu/ca/biblioguies/biblioguia/Plagi-academic/.

Criteris de correcció

Cada exercici porta associada una puntuació sobre el total de l'activitat. A més, podeu observar la qualificació màxima que podreu obtenir si executeu el programa. No obstant això, aquesta pot disminuir segons la qualitat del vostre codi.

El codi que desenvolupeu **ha de compilar correctament i no ha de tenir errors de memòria** durant la seva execució per tal que sigui avaluat. En cas de complir aquests requisits, es valorarà la qualitat del codi segons els criteris següents:

- Que **funcioni** tal com es descriu a l'enunciat.
- Que obtingui el **resultat esperat** donades unes condicions i dades d'entrada específiques (proves proporcionades). No cal passar totes les proves, però com a mínim ha de mostrar el resultat a la pantalla.
- Que es respectin els **criteris d'estil** i que el codi estigui **comentat**. Es valorarà especialment l'ús de comentaris en anglès.
- Que les **estructures** utilitzades siguin les correctes.
- Que es **separin adequadament la declaració i la implementació** de les accions i funcions, utilitzant els fitxers corresponents.
- El **grau d'optimització** en temps i recursos utilitzats en la solució lliurada.

- Que es realitzi una **gestió adequada de la memòria**, alliberant-la quan sigui necessari.

Anàlisi dinàmica

En aquesta activitat s'utilitza memòria dinàmica, cosa que requereix que el programador reservi, inicialitzi i alliberi la memòria. Per ajudar a detectar memòria que no s'ha alliberat correctament o errors en les operacions amb punters relacionades, existeixen eines que executen una anàlisi dinàmica del programa. Una eina de codi obert molt utilitzada és Valgrind (https://valgrind.org/). L'ús d'aquesta eina queda fora de l'abast del curs.

Per entendre el significat dels **codis d'error**, podeu utilitzar l'enllaç següent, on també trobareu alguns exemples de codi que us ajudaran a entendre quan es generen aquests errors:

https://bytes.usc.edu/cs104/wiki/valgrind/

DSLab

En aquesta pràctica, s'utilitza l'eina DSLab (https://sd.uoc.edu/dslab/). Aquesta eina també s'utilitza en altres assignatures i té com a objectiu:

- Proporcionar un entorn comú on avaluar els exercicis de codificació.

Us aconsellem fer enviaments periòdics a l'eina dels diferents exercicis de codi, ja que us permetrà detectar possibles errors abans del lliurament final. **Tingueu en compte que és l'eina utilitzada per corregir els vostres codis, i que no es corregirà cap codi en un altre entorn o màquina**. Per tant, si el vostre codi no funciona a l'eina DSLab, es considerarà que no funciona, encara que ho faci al vostre ordinador.

Recordeu que **el lliurament que s'avaluarà serà el darrer que feu abans que acabi el termini de lliurament** a l'eina DSLab. En qualsevol cas, també **heu de lliurar l'informe del codi desenvolupat a l'apartat corresponent de l'aula, tal com indica l'enunciat**.

La informació bàsica que us serà útil en utilitzar DSLab:

- DSLab considera el lliurament correcte únicament si passa tots els tests.
- Es mostra un resum ràpid del nombre de tests passats. Per norma general, no serà necessari passar-los tots per aprovar el lliurament.
- En els detalls es mostra:
    - El detall dels tests passats i dels que han fallat.

- És possible descarregar un fitxer amb el text que el programa mostra en pantalla (sortida estàndard). S'ha inclòs l'ús de **valgrind** en la sortida estàndard, de manera que en aquest apartat podreu veure l'informe sobre la **gestió de la memòria**. És recomanable revisar-lo per assegurar-se que es fa un ús correcte dels punters i de la memòria.

- També hi ha un log d'execució que desa l'evolució de l'execució del programa. Si, a causa d'una codificació incorrecta, el programa falla i no és capaç de mostrar el resultat dels tests, s'ha de revisar aquest log per determinar en quin punt s'ha interromput l'execució.

Nota: Si trobeu algun problema amb l'eina DSLab, informeu-ne els professors perquè puguin corregir la incidència al més aviat possible.

Enunciat

Aquesta pràctica conté 4 exercicis avaluables i un informe obligatori. Has d'entregar la teva solució dels 4 exercicis i l'informe (vegeu l'últim apartat).

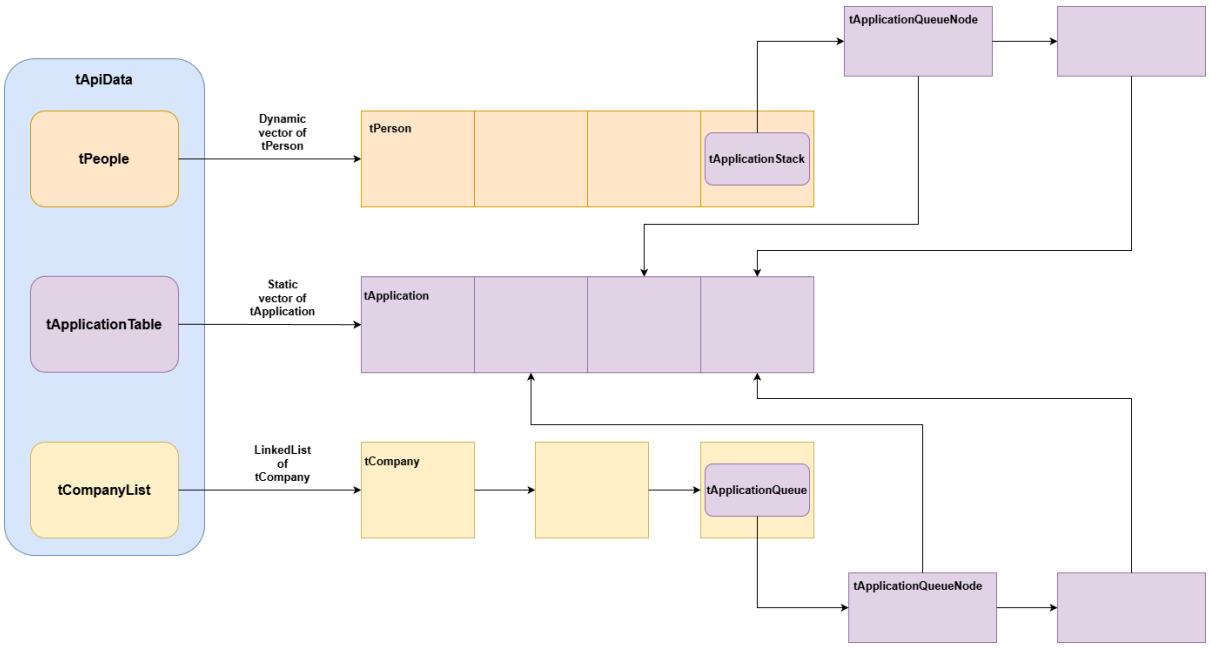
Com que les activitats estan encadenades (és a dir, per fer-ne una s'ha d'haver comprès l'anterior), **és molt recomanable fer la tasca i els exercicis en l'ordre en què apareixen en aquest enunciat**.

El punt de partida d'aquesta pràctica és la solució oficial de la PR1. El codi base de l'enunciat inclou tots els fitxers necessaris de les activitats anteriors, amb els quals veureu que tots els tests de la PR1 passen correctament.

En aquesta pràctica utilitzarem l'estructura de dades plantejada a la PR1 amb algunes modificacions i noves estructures amb les seves funcions relacionades per poder gestionar les sol·licituds dels usuaris per apuntar-se a una oferta de feina d'una empresa.

A més, cada usuari podrà consultar les seves pròpies sol·licituds de manera eficient, de manera que **dins de l'estructura `tPerson` s'emmagatzemarà una pila de sol·licituds** d'aquell mateix usuari. És a dir, s'inseriran en l'ordre invers d'arribada, essent la darrera en arribar la primera que apareixerà a la pila (LIFO, *last in, first out*).

Així mateix, cada empresa ha de poder consultar totes les sol·licituds de manera eficient, de manera que **cadascuna de les ofertes de feina (`tJob`) tindran una cua de sol·licituds**. Això significa que les sol·licituds s'han d'emmagatzemar per ordre d'arribada, essent la primera a arribar la primera a sortir (FIFO, *first in, first out*).



Tanmateix, tal com s'il·lustra en el diagrama anterior, **tant a la pila com a la cua** no s'emmagatzemaran les sol·licituds completes, sinó **únicament referències** (punters) a aquestes sol·licituds. **Les sol·licituds estaran guardades realment en l'estructura `tApiData`**, mentre que la pila i la cua es limitaran a mantenir referències a aquestes mateixes sol·licituds ja emmagatzemades.

Exercici 1 (25%) - Manipulació de piles

Una vegada entès el disseny de les estructures amb què es treballarà en aquesta pràctica, heu de fer una primera compilació per assegurar-vos de tenir l'entorn ben configurat per poder començar amb el desenvolupament dels exercicis.

Consell: Es recomana llegir i estudiar el material d'estudi vinculat a aquesta pràctica abans de començar a codificar.

En aquest exercici s'han de codificar les funcions que permeten gestionar una pila (`tApplicationStack`) que emmagatzema referències (punters) a sol·licituds (`tApplication*`) prèviament creades.

Nota: Trobareu totes les funcions que heu de codificar per a aquest exercici als fitxers `application.c` i `application.h`.

Veureu que moltes funcions d'aquest exercici retornen el tipus de dada `tApiError`, una enumeració que podeu trobar al fitxer `error.h`. A continuació, us mostrem una taula dels tipus d'errors que s'espera que retornin aquestes funcions per a aquest exercici:

`E_SUCCESS`	Operació executada correctament.
`E_NOT_IMPLEMENTED`	La funcionalitat encara no està implementada.
`E_APP_NOT_FOUND`	No existeix cap sol·licitud a l'estructura.
`E_MEMORY_ERROR`	Es produeix un error durant la reserva de memòria.

A l'enunciat veureu que totes les funcions que no han estat implementades i tenen com a retorn un valor de l'enumeració `tApiError` retornen el valor `E_NOT_IMPLEMENTED`. Heu d'eliminar o comentar aquesta línia de codi i retornar el valor correcte de l'enumeració segons les necessitats de la funció.

Consell: Utilitza l'eina de cerca en fitxers de CodeLite cercant el codi de cada funció per localitzar-la ràpidament al *workspace*.

Per tant, en aquest exercici se sol·licita:

- **PR2_1a**. Codifica la funció **`applicationStack_createStack`** perquè es puguin inicialitzar tots els camps de l'estructura rebuda per referència **`stack`** de tipus **`tApplicationStack`**. La funció ha de retornar el literal **`E_SUCCESS`**.

0.2 pts. tests proporcionats

- **PR2_1b**. Codifica la funció **`applicationStack_emptyStack`** perquè retorni **`true`** si la pila rebuda al paràmetre **`stack`** està buida i **`false`** en cas contrari.

Nota: El test associat a aquesta funció no es podrà superar fins que no s'hagin superat els tests de la funció **`applicationStack_push`**.

0.2 pts. tests proporcionats

- **PR2_1c**. Codifica la funció **`applicationStack_push`** per apilar una sol·licitud a la pila de sol·licituds (LIFO). La funció rep els paràmetres **`stack`** (**`tApplicationStack*`**) i **`application`** (**`tApplication*`**), que representen una pila de sol·licituds i un punter a la sol·licitud que es vol apilar respectivament. Si durant la reserva dinàmica de memòria ocorre un error, la funció ha de retornar **`E_MEMORY_ERROR`**. Si l'operació es realitza correctament, ha de retornar **`E_SUCCESS`**. Els nodes de la pila han de quedar correctament enllaçats.

0.75 pts. tests proporcionats

- **PR2_1d**. Codifica la funció **`applicationStack_pop`** per desapilar una sol·licitud de la pila de sol·licituds. La funció rep el paràmetre **`stack`** (**`tApplicationStack*`**), que representa la pila que conté els punters a les sol·licituds. En cas que la pila estigui buida, la funció ha de retornar **`E_APP_NOT_FOUND`**. En cas contrari, ha de retornar **`E_SUCCESS`** i els nodes de la pila han de quedar correctament enllaçats.

0.75 pts. tests proporcionats

- **PR2_1e**. Codifica la funció **`applicationStack_top`** perquè retorni un punter a la sol·licitud que es troba al cim de la pila. La funció rep el paràmetre **`stack`** (**`tApplicationStack`**), que representa la pila de sol·licituds de la qual s'ha d'obtenir el seu cim. En cas que la pila estigui buida, la funció ha de retornar **`NULL`**.

0.3 pts. tests proporcionats

- **PR2_1f**. Codifica la funció **`applicationStack_free`** perquè desapili totes les sol·licituds de la pila de sol·licituds rebuda al paràmetre **`stack`** (**`tApplicationStack*`**). D'aquesta manera, la pila de sol·licituds ha de quedar buida. La funció ha de retornar **`E_SUCCESS`** un cop finalitzi l'operació.

0.3 pts. tests proporcionats

Requisit mínim per avaluar aquest exercici: El programa ha de compilar i no ha de provocar errors de memòria durant la seva execució.

Nota: L'estudiant pot rebre una penalització de fins a **1,25 punts** de la nota obtinguda en aquest exercici en funció de la qualitat del codi proporcionat.

Exercici 2 (25%) - Manipulació de cues

En aquest exercici s'han de codificar les funcions que permeten gestionar una cua (`tApplicationQueue`) que emmagatzema referències (punters) a sol·licituds (`tApplication*`) prèviament creades.

Nota: Trobareu totes les funcions que heu de codificar per a aquest exercici als fitxers `application.c` i `application.h`.

Seguint el mateix criteri que en l'exercici anterior, al llarg d'aquest exercici trobareu funcions que han de retornar un valor de tipus `tApiError`, que indicarà si s'ha produït algun error o si l'acció s'ha executat correctament.

A continuació es detallen els errors que s'espera que retornin les funcions que retornen un valor de tipus `tApiError` en aquest exercici.

`E_SUCCESS`	Operació executada correctament.
`E_NOT_IMPLEMENTED`	La funcionalitat encara no està implementada.
`E_APP_NOT_FOUND`	No existeix cap sol·licitud a l'estructura.
`E_MEMORY_ERROR`	Es produeix un error durant la reserva de memòria.

Seguidament, es detalla el comportament de les funcions a desenvolupar:

- **PR2_2a**. Codifica la funció `applicationQueue_createQueue` perquè es puguin inicialitzar tots els camps de l'estructura rebuda per referència `queue` de tipus `tApplicationQueue`. La funció ha de retornar el literal `E_SUCCESS`.

  0.2 pts. tests proporcionats

- **PR2_2b**. Codifica la funció `applicationQueue_emptyQueue` perquè retorni `true` si la cua rebuda en el paràmetre `queue` està buida i `false` en cas contrari.

Nota: El test associat a aquesta funció no es podrà superar fins que no s'hagin superat els tests de la funció **`applicationQueue_enqueue`**.

0.2 pts. tests proporcionats

- **PR2_2c**. Codifica la funció **`applicationQueue_enqueue`** per encolar una sol·licitud a la cua de sol·licituds (FIFO). La funció rep els paràmetres **`queue`** (**`tApplicationQueue*`**) i **`application`** (**`tApplication*`**), que representen una cua de sol·licituds i un punter a la sol·licitud que es vol encolar respectivament. Si durant la reserva dinàmica de memòria ocorre un error, la funció ha de retornar **`E_MEMORY_ERROR`**. Si l'operació es realitza correctament, ha de retornar **`E_SUCCESS`**. Els nodes de la cua han de quedar correctament enllaçats.

0.75 pts. tests proporcionats

- **PR2_2d**. Codifica la funció **`applicationQueue_dequeue`** per desencolar una sol·licitud de la cua de sol·licituds. La funció rep el paràmetre **`queue`** (**`tApplicationQueue*`**), que representa la cua que conté els punters a les sol·licituds. En cas que la cua estigui buida, la funció ha de retornar **`E_APP_NOT_FOUND`**. En cas contrari, ha de retornar **`E_SUCCESS`** i els nodes de la cua han de quedar correctament enllaçats.

0.75 pts. tests proporcionats

- **PR2_2e**. Codifica la funció **`applicationQueue_head`** perquè retorni un punter a la sol·licitud que es troba a la primera posició de la cua. La funció rep el paràmetre **`queue`** (**`tApplicationQueue`**), que representa la cua de sol·licituds de la qual s'ha d'obtenir la seva primera posició. En cas que la cua estigui buida, la funció ha de retornar **`NULL`**.

0.3 pts. tests proporcionats

- **PR2_2f**. Codifica la funció **`applicationQueue_free`** perquè desencoli totes les sol·licituds de la cua de sol·licituds rebuda en el paràmetre **`queue`** (**`tApplicationQueue*`**). D'aquesta manera, la cua de sol·licituds ha de quedar buida. La funció ha de retornar **`E_SUCCESS`** un cop finalitzi l'operació.

0.3 pts. tests proporcionats

Requisit mínim per avaluar aquest exercici: El programa ha de compilar i no ha de provocar errors de memòria durant la seva execució.

Nota: L'estudiant pot rebre una penalització de fins a **1,25 punts** de la nota obtinguda en aquest exercici en funció de la qualitat del codi proporcionat.

Exercici 3 (30%) - Integració a l'API

Tal com es va fer a la pràctica anterior, en aquest exercici s'integraran totes les noves estructures i funcionalitats a l'API per no exposar els seus tipus de dades interns. No obstant això, és necessari adaptar funcions que manipulen altres estructures, ja que les persones i les ofertes de feina han d'emmagatzemar les piles i les cues de sol·licituds respectivament. D'aquesta manera, es podrà implementar el disseny plantejat a través del diagrama que hi ha a l'inici d'aquest enunciat.

Consell: És altament recomanable reutilitzar funcions i constants al llarg d'aquest exercici perquè el codi segueixi els principis del disseny descendent.

A continuació, es presenten els canvis que s'han fet per a aquesta pràctica.

- **PR2_3a**. Fixeu-vos en els següents camps (no s'ha de fer cap acció):
  - El camp **`applications`** de tipus **`tApplicationTable`** que es troba a l'estructura **`tApiData`** al fitxer **`api.h`**.
  - El camp **`applicationsStack`** de tipus **`tApplicationStack`** que es troba a l'estructura **`tPerson`** al fitxer **`person.h`**.
  - El camp **`applicationsQueue`** de tipus **`tApplicationQueue`** que es troba a l'estructura **`tJob`** al fitxer **`job.h`**.

0 pts. tests proporcionats

Les següents funcions a modificar es troben al fitxer **`api.c`**:

- **PR2_3b**. Modifica les funcions **`api_initData`** i **`api_freeData`** perquè s'inicialitzi i es destrueixi (s'alliberi la memòria) el camp applications de tipus **`tApplicationTable`** respectivament, que has descomentat al primer apartat.

0.3 pts. tests proporcionats

Les següents funcions a modificar es troben al fitxer **`person.c`**

- **PR2_3c**. Modifica les funcions **`person_parse`** i **`person_cpy`** perquè puguin inicialitzar l'estructura **`applicationsStack`** de tipus **`tApplicationStack`** que has descomentat al primer apartat. Per simplificar l'exercici, la funció que copia una persona (**`person_cpy`**) només ha d'inicialitzar la pila de sol·licituds, és a dir, no n'ha de copiar el contingut. A continuació, modifica la funció **`person_free`** perquè la mateixa estructura es pugui destruir correctament.

0.45 pts. tests proporcionats

Les següents funcions a modificar es troben al fitxer **`job.c`**:

- **PR2_3d**. Modifica les funcions job_parse i job_cpy perquè puguin inicialitzar l'estructura **`applicationsQueue`** de tipus **`tApplicationQueue`** que has descomentat al primer apartat. Igual que a l'apartat anterior, per simplificar l'exercici, la funció que copia una oferta de feina (**`job_cpy`**) només ha d'inicialitzar la cua de sol·licituds, és a dir, no n'ha de copiar el contingut. A continuació, modifica la funció **`job_free`** perquè la mateixa estructura es pugui destruir correctament.

0.45 pts. tests proporcionats

Finalment, les següents funcions a modificar es troben de nou al fitxer **`api.c`**:

- **PR2_3e**. Implementa la funció **`api_addApplication`** perquè, donada una estructura **`tApiData`** i una sol·licitud en format CSV (**`tCSVEntry`**), afegeixi la sol·licitud al camp applications de tipus **`tApplicationTable`**. Una vegada afegida, ha d'obtenir la referència de la sol·licitud ja afegida i emmagatzemar la seva referència a la pila de sol·licituds de la persona a qui correspon (**`tApplicationStack`**) i, posteriorment, a la cua de sol·licituds de l'oferta de feina a què correspon (**`tApplicationQueue`**). Aquesta implementació ha de seguir el disseny que mostra el diagrama de l'inici de l'enunciat.

  Així mateix, la funció ha de fer les següents validacions en l'ordre següent:

  - Si el tipus del CSV no és "**`APPLICATION`**", llavors la funció ha de retornar el valor **`E_INVALID_ENTRY_TYPE`**.
  - Si el nombre de camps del CSV no és **`5`**, llavors la funció ha de retornar el valor **`E_INVALID_ENTRY_FORMAT`**.
  - Si el document de la persona que està formalitzant la sol·licitud no existeix al programa, llavors la funció ha de retornar el valor **`E_PERSON_NOT_FOUND`** i, a més, no ha d'inserir cap dada a cap estructura.
  - Si l'identificador de l'oferta de feina de la sol·licitud no existeix al programa, llavors la funció ha de retornar el valor **`E_JOB_NOT_FOUND`** i, a més, no ha d'inserir cap dada a cap estructura.
  - En cas de finalitzar correctament les operacions, la funció ha de retornar **`E_SUCCESS`**.

1.4 pts.tests proporcionats

- **PR2_3f**. Implementa la funció **`api_applicationsCount`** perquè retorni el nombre total de sol·licituds que hi ha emmagatzemades al programa.

0.15 pts. tests proporcionats

- **PR2_3g**. Modifica la funció **`api_addEntryData`** perquè, donada una sol·licitud en format CSV a través del paràmetre **`entry`** (**`tCSVEntry`**), la pugui afegir a l'aplicació seguint els criteris especificats a la funció **`api_addApplication`**. Per poder afegir la sol·licitud, el tipus del CSV ha de ser "**`APPLICATION`**". En cas contrari, la funció ha de retornar el valor **`E_INVALID_ENTRY_TYPE`**.

0.25 pts. tests proporcionats

Requisit mínim per avaluar aquest exercici: El programa ha de compilar i no ha de provocar errors de memòria durant la seva execució.

Nota: L'estudiant pot rebre una penalització de fins a **1,5 punts** de la nota obtinguda en aquest exercici en funció de la qualitat del codi proporcionat.

Exercici 4 (20%) - Funcions recursives

Finalment, en aquest exercici es demana que s'apliquin funcions recursives per a la consulta de dades sobre les sol·licituds de les persones i de les ofertes de feina.

A més, per poder oferir aquestes dades de manera desacoblada de les estructures internes de l'API, s'han de retornar en format CSV. Per això, és altament recomanable utilitzar les funcions següents definides i implementades als fitxers **`csv.c`** i **`csv.h`**:

`csv_init / csv_free`	Inicialitza / allibera una estructura de tipus `tCSVData`.
`csv_addStrEntry`	Afegeix a una estructura de tipus **`tCSVData`** una nova entrada (**`tCSVEntry`**) a partir d'una cadena en format CSV.
`sprintf`	Mètode similar a **`printf`**, però que en lloc de mostrar la informació formatada per pantalla, la guarda en una cadena.

A continuació, es detalla el comportament de les funcions a desenvolupar:

- **PR2_4a**. Codifica la funció **`api_reversePersonApplicationStack`** perquè retorni, al paràmetre **`applications`** (de tipus **`tCSVData*`**), totes les sol·licituds associades a la persona identificada pel document **`document`**, ordenades en ordre invers al que apareixen a la seva pila de sol·licituds.

  La funció ha de localitzar la persona dins l'estructura **`tApiData`**, inicialitzar correctament l'estructura **`applications`** i generar la sortida amb el següent format CSV:

```
"document;name;surname;phone;email;address;cp;birthday"
"applicationId;personDocument;jobId;status;
appliedAt;observations"
```

Per exemple:

```
"1;47051307Z;1;0;06/11/2025;20:51;Text"
```

Nota: Podeu assumir que una cadena en format CSV no superarà mai els **2048** caràcters (2 KB) de longitud. Els següents mètodes poden ser útils per realitzar aquest exercici.

El tipus de registre ha de ser "**APPLICATION**". A més, si la persona no existeix en les dades de l'aplicació, la funció ha de retornar l'error **E_PERSON_NOT_FOUND**; en cas contrari, **E_SUCCESS**.

Per obtenir les sol·licituds en ordre invers heu d'utilitzar obligatòriament la funció auxiliar proporcionada **api_reversePersonApplicationStack_rec**, codificant-la perquè implementi una solució recursiva per recórrer la pila sense modificar-ne l'estat final.

Pista: Desapila totes les sol·licituds fins que la pila quedi buida i, posteriorment, apila-les perquè quedin en el mateix ordre.

1 pt. tests proporcionats

- **PR2_4b**. Codifica la funció **api_reverseJobApplicationQueue** perquè retorni, al paràmetre **applications** (de tipus **tCSVData***), totes les sol·licituds associades a l'oferta de feina identificada per l'identificador **id**, ordenades en ordre invers al que apareixen a la seva cua de sol·licituds.

  La funció ha de localitzar l'oferta de feina recorrent la llista d'empreses dins l'estructura **tApiData**, inicialitzar correctament l'estructura **applications** i generar la sortida amb el següent format CSV:

  ```
  "document;name;surname;phone;email;address;cp;birthday"
  "applicationId;personDocument;jobId;status;
  appliedAt;observations"
  ```

  Per exemple:

  ```
  "1;47051307Z;1;0;06/11/2025;20:51;Text"
  ```

El tipus de registre ha de ser **`"APPLICATION"`**. A més, si l'oferta de feina no existeix en les dades de l'aplicació, la funció ha de retornar l'error **`E_JOB_NOT_FOUND`**; en cas contrari, **`E_SUCCESS`**.

Per obtenir les sol·licituds en ordre invers heu d'utilitzar obligatòriament la funció auxiliar proporcionada **`api_reverseJobApplicationQueue_rec`**, codificant-la perquè implementi una solució recursiva que recorri la cua sense modificar-ne l'estat final.

Pista: Pensa en l'estructura d'anell per poder implementar aquesta funcionalitat.

1 pt. tests proporcionats

Requisit mínim per avaluar aquest exercici: El programa ha de compilar i no ha de provocar errors de memòria durant la seva execució. A més, les funcions auxiliars han d'implementar-se recursivament.

Nota: L'estudiant pot rebre una penalització de fins a **1 punt** de la nota obtinguda en aquest exercici en funció de la qualitat del codi proporcionat.

Informe obligatori

Una vegada finalitzats tots els exercicis avaluables, se sol·licita un **informe en format PDF de màxim 2 pàgines** sobre el codi desenvolupat. Aquest informe ha d'incorporar dos apartats:

- **Dificultats i problemes**: s'han d'enumerar les dificultats i els problemes amb què t'hagis trobat durant el desenvolupament dels exercicis d'aquesta pràctica.
- **Lliçons apreses**: s'han d'esmentar tots aquells conceptes adquirits durant la realització dels exercicis avaluables.

És a dir, **s'espera que l'informe sigui un document d'autoreflexió** i no un resum del que fan les funcions codificades, ja que aquesta informació és la que descriu aquest enunciat.

Requisit mínim per avaluar la pràctica: S'ha de lliurar aquest informe a l'apartat d'entregues de l'AC tal com s'indica en l'últim apartat.

Nota: Aquest informe **no s'utilitzarà per a la qualificació d'aquesta pràctica**, però sí que servirà per constatar el grau de compliment dels objectius d'aprenentatge a títol individual.

Format i data de lliurament

Has de lliurar els següents fitxers seguint les instruccions:

- Els **fitxers amb el codi font dels exercicis 1, 2, 3 i 4** requerits a l'eina **DSLab**:
  - `application.c/application.h`
  - `person.c/person.h`
  - `job.c/job.h`
  - `api.c/api.h`
  - `README.txt` amb el següent format:

> **Format**:
> Correu electrònic UOC
> Cognoms, Nom
> Sistema operatiu utilitzat
>
> **Exemple**:
> estudiant@uoc.edu
> Cognom1 Cognom2, Nom
> Windows 11

Important: Una vegada hàgiu construït el vostre projecte amb el botó *Build*, l'**heu d'executar amb el botó *Submit***. En cas contrari, el vostre codi no s'avaluarà.

- L'**informe obligatori en format PDF** a l'apartat d'**entregues de l'AC de l'aula de teoria**.

No seguir el **format de lliurament**, tant en el **tipus i nom dels fitxers** com en el contingut requerit, implicarà una **penalització important** o la qualificació amb una **D en l'activitat**.

L'últim dia per lliurar aquesta PR és el **05/12/2025** abans de les 23:59. Qualsevol PR lliurada més tard serà considerada com a no presentada. Només s'avaluarà **l'últim enviament dins del període establert**.

Nota: L'incompliment del format de lliurament especificat anteriorment pot suposar el suspens de la pràctica.