Universitat Oberta de Catalunya uoc.edu

Fonaments de Programació

PAC10 - 20182

Data límit de lliurament: 27/05/2019

Estudiant

Cognoms: GALEANO SANCHO

Nom: DANIEL

Objectius

- Comprendre què són els tipus abstractes de dades (TAD) i saber definir-los correctament.
- Implementar operacions amb TAD.
- Aprofundir en la modularització del codi utilitzant accions i funcions.
- Aprofundir en l'ús de paràmetres d'entrada, paràmetres de sortida i paràmetres d'entrada/sortida.

Format i data de lliurament

Cal lliurar la PAC abans del dia **27 de maig de 2019 a les 23:59**.

Per al lliurament caldrà que entregueu un fitxer en format ZIP, que contingui:

- Aquest document amb la resposta de l'exercici 1.
- El workspace de Codelite que contingui **el projecte i totes les carpetes creades** a l'exercici 2

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

Enunciat

Seguint amb l'ajuda que proporcionem a UOCAirways, la companyia ens ha demanat ara la nostra col·laboració per crear un programa que els ajudi a gestionar els moviments de maletes que tenen lloc quan un avió aterra, quan un passatger no es presenta a la porta d'embarcament, quan hi ha un enllaç de vols i s'han de recuperar les maletes corresponents a un vol determinat, etc. En aquesta PAC, treballarem amb el tipus abstracte de dades (TAD) pila, conjuntament amb l'entrada i sortida de dades serialitzada.

Per a la gestió dels equipatges, s'han de tenir en compte els itineraris de vol. Per això, cada maleta tindrà vinculada certa informació sobre l'itinerari que realitza el passatger propietari de la maleta:

- airportCodes: un vector d'enters on cada element indica el codi de l'aeroport en el qual el passatger hi té parada. Quan aquest vector tan sols tingui un element, indica que l'aeroport que es mostra és fi de trajecte. Quan en tingui més d'un, vol dir que hi ha una o diverses parades en les quals l'equipatge s'haurà de manipular i traslladar cap a l'avió corresponent per al següent destí.
- countryCodes: un vector d'enters on cada element indica el codi del país en què el passatger hi té parada. Aquest codi ens servirà per a discriminar si la maleta, en la parada que realitza, ha d'anar cap a un vol d'àmbit nacional o internacional, o bé, si es queda a l'aeroport actual. En aquest últim cas, indicarem que la maleta va a *sortida* de l'aeroport.
- carrierCodes: un vector d'enters on cada element indica el codi de la companyia que ha operat el corresponent vol.
- numStopOvers: un enter que indica el nombre de parades que té l'itinerari que ha de seguir la maleta.

Suposem els següents codis d'exemple per aeroports, països i companyies aèries (per tal de simplificar l'enunciat, aquí emprem nombres enters en compte dels codis IATA més usuals):

aeroports: Barcelona-001; Madrid-002; Milà-003; París-004; Lisboa-005
països: Espanya-001; Italia-002; França-003; Portugal-004;
companyies aèries: Iberia-001; British-002; Vueling-003; Air France- 004

Suposem ara, un passatger que vola a Barcelona amb Vueling i finalitza allí el seu recorregut. El contingut dels camps anteriors seria com segueix:

airportCodes: 001

countryCodes: 001
carrierCodes : 003
numStopOvers: 0

En el cas que el passatger vola a Barcelona amb Vueling però tan sols hi fa escala per anar cap a Lisboa amb Iberia, el contingut dels camps sería:

airportCodes: 001, 005
countryCodes: 001, 004;
carrierCodes : 003, 001
numStopOvers: 1

En el cas que el passatger vola a Madrid amb Vueling, fa escala per volar cap a Barcelona amb Iberia i des d'allí agafa un altre avió d'Iberia cap a Milà. El contingut dels camps seria:

airportCodes: 002, 001, 003
countryCodes: 001, 001, 002
carrierCodes : 003, 001, 001
numStopOvers: 2

Per a la gestió de les maletes, partim de les següents definicions de tipus de dades:

```
const
        MAX_STOPOVERS: integer =  10;
        MAX_PASSENGERS: integer = 1000;
end const
type
        tLuggage= record
                idPassenger: integer;
                airportCodes: vector [MAX_STOPOVERS+1] of integer;
                countryCodes: vector [MAX_STOPOVERS+1] of integer;
                carrierCodes: vector [MAX_STOPOVERS+1] of integer;
                numStopOvers: integer;
        end record

        tStack = record
                A: vector [MAX_PASSENGERS] of tLuggage;
                nelem: integer;
        end record
end type

algorithm UOCAirways

{... algorithm to complete …}
```

end algorithm

Quan un avió arriba a un aeroport, associat al vol corresponent, hi tenim una pila d'equipatges que conté la informació dels equipatges dels passatgers que venen en aquest vol. Per tal de poder treballar amb la pila d'equipatges, disposem de les funcionalitats bàsiques, ja implementades, per a manipular els elements d'una pila. Aquestes accions/funcions les podem utilitzar sense necessitat d'implementar de nou.

function createStack() : tStack

action push(**inout** s: tStack, **in** l: tLuggage)

action pop(**inout** s : tStack)

function top (s: tStack): tLuggage

function emptyStack (s: tStack): **tBoolean**

function fullStack (s: tStack): **tBoolean**

function heightStack (s: tStack): **integer**

Nota: La definició d'aquestes accions/funcions és d'acord al que podeu trobar en la xWiki.

Exercici 1: Disseny en llenguatge algorísmic (50%)

Apartat 1.a: [25%]

Dissenyeu l'**acció splitLuggageOnArrival** que rep com a paràmetres quatre piles d'equipatge. La primera pila representa l'equipatge que porta un avió que acaba d'aterrar, i els altres tres paràmetres representen tres piles buides. El que fa aquesta acció és retornar en els tres darrers paràmetres, les maletes que hi ha en la primera pila de manera que l'equipatge quedi separat d'acord al seu itinerari en internacional, nacional o cap a *sortida* de l'aeroport. Així, la segona pila contindrà l'equipatge d'aquells passatgers que es queden en aquest aeroport, la tercera pila l'equipatge d'aquells que continuen en vol nacional, i la quarta pila l'equipatge d'aquells que continuen en vol internacional.

```
type
   tDestination = {EXIT, NATIONAL, INTERNATIONAL}
end type

action splitLuggageOnArrival (inout s: tStack, out exit: tStack, out national: tStack, out
international: tStack)
   var
      isInternational: boolean;
      l: tLuggage;
      numStops: integer;
      destination: tDestination;
```

```
    end var

    isInternational := false;

    while (not emptyStack(s)) do
        l := top(s);
        numStops := l.numStopOvers;
        if (numStops = 0) then
            destination := EXIT;
        else
            if (l.countryCodes[1] ≠ l.countryCodes[2]) then
                destination := INTERNATIONAL;
            else
                destination := NATIONAL;
            end if
        end if
        push(destination, l);
        pop(s);
    end while

end action
```

Apartat 1.b: [25%] Dissenyeu la funció **deletePassengerLuggage** que rep com a paràmetre una pila d'equipatges corresponent a un vol concret i un enter que representa l'identificador d'un passatger. En aquells casos en què un passatger no arriba a embarcar, s'ha de buscar el seu equipatge i treure'l, i per aquesta tasca farem servir aquesta funció. Aquesta funció busca l'equipatge del passatger, i:

- retorna TRUE si el troba en la pila d'equipatge i FALSE en cas contrari.
- en cas de que es trobi l'equipatge del passatger, actualitza la pila d'equipatges de manera que s'elimini l'equipatge del passatger. S'ha de respectar l'ordre original dels equipatges.
- en cas de que no es trobi l'equipatge, la pila d'equipatges ha de quedar inalterada.

```
action deletePassengerLuggage (inout s: tStack, in idPassenger: integer, inout found:
boolean)
    var
        aux: tStack;
        l: tLuggage;
    end var

    found := false;
    aux := createStack();

    while (not emptyStack(s) or not found) do
        l := top(s);

        if l.idPassenger ≠ idPassenger then
            push(aux, l);
        else
            found := true;
```

```
        end if

        pop(s);
    end while

    while not emptyStack(aux) do
        l := top(aux);
        push(s, l);
        pop(aux);
    end while
end action
```

Apartat 1.c: [20%] Dissenyeu l'acció **extractFlightLuggage** que rep tres paràmetres. Els dos primers són de tipus pila i el tercer és un enter que representa el codi de l'aeroport de destinació d'un vol. Aquesta acció parteix del primer paràmetre de tipus pila per omplir el segon paràmetre de tipus pila amb aquells equipatges que es corresponguin amb el vol identificat amb el codi passat com a tercer paràmetre. La pila amb l'equipatge del vol desitjat no cal que mantingui el mateix ordre en que era a la pila inicial.

```
action extractFlightLuggage (inout s: tStack, out filteredStack: tStack, in airportCode:
integer)
    var
        aux: tStack;
        l: tLuggage;
    end var

    aux := createStack();

    while not emptyStack(s) do
        l := top(s);
        if l.airportCodes[1] = airportCode then
            push(filteredStack, l);
        else
            push(aux, l);
        end if
        pop(s);
    end while

    while not emptyStack(aux) do
        l := top(aux);
        push(s, l);
        pop(aux);
    end while

end action
```

Apartat 1.d: [30%] Una companyia aèria vol considerar la possibilitat d'una gestió específica d'aquells equipatges que, en el seu itinerari, han de realitzar connexions a vols internacionals. Per temes de control de seguretat, aquest equipatge ha de passar uns controls més exhaustius. Es tracta de dissenyar una acció **getWithInternationalConnection** que rep dos paràmetres de tipus pila:

- una pila d'entrada amb els equipatges d'un vol que arriba a l'aeroport,
- una pila de sortida, amb els equipatges d'aquest vol que, al llarg del seu itinerari, tenen alguna connexió internacional.

i un tercer paràmetre de tipus enter amb el codi de la companyia aèria. El que es vol considerar són aquells passatgers que tenen la connexió amb aquesta companyia. És a dir, tant el vol d'arribada com el següent són amb la companyia aèria indicada pel codi del tercer paràmetre.

```
action  getWithInternationalConnection (inout s: tStack, out getWithLuggage: tStack, in
carrierCode: integer)
   var
      aux: tStack;
      l: tLuggage;
      isSelected: boolean;
      i: integer;
      numStops: integer;
   end var

   aux := createStack();

   while not emptyStack(s) do
      l := top(s);
      numStops := l.numStopOvers;
      isSelected := false;

      if numStops = 0 then
         push(aux, l);
      else
         for i := 2 to l.numStopOvers+1 do
            if l.countryCodes[i] ≠ l.countryCodes[i-1] and
               l.carrierCodes[i-1] = carrierCode and
               l.carrierCodes[i] = carrierCode then
                  isSelected:= true;
            end if
         end for
         if isSelected then
            push(getWithLuggage, l);
         else
            push(aux, l);
         end if
      end if
      pop(s);
   end while
```

```
    while not emptyStack(aux) do
        l := top(aux);
        push(s, l);
        pop(aux);
    end while
end action
```

Exercici 2: Programació en C (50%)

En aquest exercici cal codificar en C l'exercici 1 seguint amb l'estructura de carpetes i fitxers. Es proporciona un projecte inicial UOCAirways en el que ja hi ha implementades les funcionalitats de manipulació de piles així com funcionalitats per

- tractar amb els equipatges
- llegir un arxiu d'equipatges i carregar-lo en una pila
- escriure una pila en un arxiu d'equipatges
- comparar dos arxius d'equipatges per verificar si son iguals

Aquestes funcionalitats aportades ens serviran per a poder contrastar els resultats del nostre algoritme amb els resultats esperats.

1. Descomprimiu el fitxer *UOCAirways.zip* que inclou els fitxers de projecte. El projecte està estructurat en carpetes: carpeta *include* on hi ha els fitxers ***data.h, luggage.h i stack.h*** i carpeta *src* on hi ha els fitxers ***main.c, stack.c*** i ***luggage.c.***

Per a cada un dels apartats de l'exercici 1, tenim el corresponent test per a verificar el correcte funcionament de la implementació que hem fet. Tots aquests test obeeixen a la mateix alògica que teniu explicada en comentaris en el ***main.c***

El primer test el teniu tot implementat per tal de que us pugui servir d'exemple. Senzillament, carreguem una pila d'equipatges, en creem un altra amb l'ordre invertit al resultat de la càrrega i escrivim el resultat en l'arxiu corresponent. Fixeu-vos en la implementació de la funció *invertStack* dintre de l'arxiu ***stack.c*** i la corresponent definició dintre de ***stack.h***. Dintre de ***stack.c*** es on teniu ja implementades totes les funcionalitats per manipular el TAD pila.

   a. Compilar i executar el projecte
   b. Verifiqueu que la sortida del programa es "*... Test 1 passed OK*"

A continuació, en base a aquest exemple de test, anirem implementant la resta d'accions de l'exercici 1 i realitzant els corresponents Test que teniu ja plantejats i comentats en el fitxer **main.c**

2. Dins del fitxer ***luggage.h*** defineix la capçalera de l'acció de l'exercici 1.a

3. Dins del fitxer ***luggage.c*** implementeu l'acció de l'exercici 1.a
4. Completar la part que resta del Test2. Compilar i executar el projecte, verifiqueu que la sortida del programa es "*... Test 2 passed OK*"

Cal notar que en el cas de l'exercici 1.b, donat que és generen tres piles, haurem de generar tres arxius de sortida i, per tant, realitzar les comprovacions per a cadascun dells. Haurem d'obtenir, per tant, tres vegades el missatge "*... Test 2 passed OK*"

Realitzar el mateix per a cada una de les funcionalitats dels exercicis 1.b, 1.c i 1.d. Totes elles les declararem en el arxiu ***luggage.h*** i les implementarem en el arxiu ***luggage.c***.

Criteris de correcció:

A l'exercici 1:

- Que es segueixi la notació algorísmica utilitzada a l'assignatura. Vegeu document *Nomenclator* a la xWiki de contingut.
- Que es segueixen les instruccions donades i l'algorisme respongui al problema plantejat.
- Que es dissenyin i es cridin correctament les accions i funcions demanades.

A l'exercici 2:

- Que el programa s'adeqüi a les indicacions donades.
- Que el programa compili i funcioni d'acord amb el que es demana.
- Que es respectin els criteris d'estil de programació C. Vegeu la G*uia d'estil de programació en C* que teniu a la Wiki de contingut.
- Que s'implementi correctament la modularització del projecte, dividint el codi en carpetes i posant el que correspon a cada carpeta.