Fonaments de Programació

Pràctica 2 - Segona Pràctica

Presentació

En aquest document es presenta la segona pràctica. Aquesta prova constitueix un 50% de la nota final de pràctiques.

Competències

La principal competència del grau que es treballa en aquesta pràctica és la capacitat de dissenyar i construir aplicacions informàtiques mitjançant tècniques de desenvolupament, integració i reutilització. Per treballar aquesta competència, la pràctica planteja diversos exercicis al voltant de l'ús d'estructures de dades com **taules** i **tuples**, així com les combinacions entre elles.

Objectius

- Identificar i declarar els tipus de dades (taules, tuples, variables i constants) necessàries a partir d'un problema / enunciat.
- Especificar com seria una crida a una acció/funció que té paràmetres estructurats.
- Entendre el funcionament de l'accés als elements d'una taula i/o als camps d'una tupla.
- Dissenyar un algorisme, que pugui contenir esquemes de cerca/recorregut, i que tracti amb taules/tuples.
- Donada una definició de tipus que presenta una combinació de taules i tuples, es demana la traducció a C d'un algorisme que accedeixi a aquestes estructures.

Descripció de la pràctica

La pràctica consta de 5 exercicis, que treballen els diferents objectius que s'han enumerat com a claus en aquesta prova. Cada exercici consta d'un enunciat i un pes que indica el percentatge que representa respecte a la nota global.

Recursos

Serà necessari llegir prèviament el capítols del temari relatius a l'ús de taules i tuples, tant dels apunts de l'assignatura com de les TS que trobareu en les cites del calendari de la vostra aula.

Addicionalment, per resoldre els exercicis de codificació, caldrà que reviseu les taules de traducció i tot el material relacionat amb la codificació en C que trobareu a l'apartat de "recursos" de la vostra aula.

Criteris de valoració

S'avaluarà cada exercici d'acord amb el seu pes, que apareix indicat abans del seu enunciat. Els exercicis que constin de diversos apartats es valoraran com a completament correctes o incorrectes. Les respostes incorrectes no disminueixen la nota. Raoneu i justifiqueu totes les respostes, especialment si així ho indica l'enunciat (si, tot i demanar-ho l'enunciat, no justifiqueu la resposta, es comptabilitzarà com a incorrecte).

Format i data de lliurament

Cal lliurar la solució dels exercicis 1 a 4 en un fitxer anomenat CognomsNom_FP_Pract2 adreçat a la bústia "Lliurament d'activitats" de l'aula de teoria. Cal lliurar la codificació de la pregunta 5 a través del Corrector Automàtic, que trobareu a l'aula de teoria.

Data límit per lliurar la solució: dimecres, 10 de desembre de 2014 (a les 23:59 hores).

Exercici 1: Declaració de tipus estructurats [20%]

Objectius: Identificar i declarar els tipus de dades (taules, tuples, variables i constants) necessàries a partir d'un problema / enunciat.

Materials: *Mòdul 4: Tipus estructurats de dades*

M 4.1: Introducció i motivació. Estructuració de dades

M 4.2: Taules

M 4.3: Tuples

Tasca: Es vol guardar la informació de tots els quioscs d'una població en concret. Per fer-ho cal fer el següent:

a) **[5%]** Declareu el tipus de dades *tNewspaperMagazine* i les constants i tipus enumerats necessaris per emmagatzemar la informació referent a cadascun dels diaris i revistes que pot vendre el quiosc. Concretament es vol guardar el tipus de premsa que ven el quiosc, que pot ser diari (*newspaper*) o revista (*journal*), el seu nom i el seu preu. Per emmagatzemar les cadenes de text utilitzeu el tipus *tString (*podeu suposar que ja està declarat).

b) **[5%]** Declareu el tipus de dades *tCollectable* i les constants i tipus de dades necessaris per emmagatzemar la informació de cadascun dels col·leccionables que pot vendre el quiosc. Per cada col·leccionable es vol guardar el seu nom, el preu, el nombre de parts del col·leccionable, i si es tracta només d'una revista o conté peces. Per emmagatzemar les cadenes de text utilitzeu el tipus *tString* (podeu suposar que ja està declarat).

c) **[5%]** Declareu el tipus de dades *tKiosk* i les constants i tipus de dades necessaris per emmagatzemar la informació relacionada amb un quiosc. Per cada quiosc es requereix l'identificador (DNI) i nom de la persona que el porta, el mes i l'any en que es va obrir, un llistat amb els diaris i revistes que ven (no seran mai més de 150 diferents), i el llistat amb els col·leccionables que ven (no seran mai més de 100 diferents). Per emmagatzemar les cadenes de text utilitzeu el tipus *tString* (podeu suposar que ja està declarat).

d) **[5%]** Declareu el tipus de dades *tAllKiosks* i les constants i tipus de dades necessaris per emmagatzemar un llistat amb tots els quioscs del municipi (no seran mai més de 10 quioscs). A més, es vol guardar el codi postal del municipi

en el que es troben els quioscs. Per emmagatzemar les cadenes de text utilitzeu el tipus *tString* (podeu suposar que ja està declarat) .

.

Exercici 2: Crides a funcions/accions que treballen amb tipus estructurats [10%]

Objectius: Especificar com seria una crida a una acció/funció que té paràmetres estructurats.

Materials: *Mòdul 4: Tipus estructurats de dades*

M 4.1: Introducció i motivació. Estructuració de dades

M 4.2: Taules

M 4.3: Tuples

Tasca: Utilitzant el tipus definits a l'exercici anterior, declareu les accions i funcions que s'indiquen a continuació. No cal dissenyar-les.

a. **[2.5%]** Declareu l'acció/funció *checkNewspaperOrMagazine* que donat un paràmetre de tipus *tNewspaperMagazine* comprovi si es tracta d'un diari o d'una revista.

b. **[2.5%]** Declareu una acció/funció *writeCollectables* que donat un paràmetre de tipus *tAllKiosks* escrigui pel canal de sortida estàndard el nom de tots els col·leccionables que venen els quioscs.

c. **[2.5%]** Declareu l'acció/funció *getLastKioscOpened* que donada una variable de tipus *tAllKiosks* retorni l'últim quiosc obert en el municipi, el nom de la persona que el porta, i el mes i any d'obertura.

d. **[2.5%]** Declareu l'acció/funció *compareCollectablesKioscs* que donats dos quioscs (dos paràmetres de tipus *tKiosk*) retorni cert si els dos quioscs venen els mateixos col·leccionables i fals en cas contrari.

Exercici 3: Accés a tipus de dades estructurats [20%]

Objectius: Entendre el funcionament de l'accés als elements d'una taula i/o als camps d'una tupla.

Materials: *Mòdul 4: Tipus estructurats de dades*

M 4.1: Introducció i motivació. Estructuració de dades

M 4.2: Taules

M 4.3: Tuples

Tasca: Disposem d'una seqüència, en el canal estàndard d'entrada, amb la informació de reserves i cancel·lacions d'un complex hoteler de Barcelona.

L'hotel disposa d'apartaments i d'habitacions diferenciant entre:

- Apartaments, segons la capacitat:
  - per a 2 persones.
  - per a 4 persones.
  - per a més de 4 persones.

- Habitacions, segons la capacitat i depenent de si disposen o no de TV:
  - per a 2 persones amb TV.
  - per a 2 persones sense TV.
  - per a 4 persones amb TV.
  - per a 4 persones sense TV.

Disposem d'una seqüència, en el canal d'entrada estàndard, amb la informació de les reserves i cancel·lacions realitzades durant el mes de novembre per les vacances de Nadal..

<$idPerson_1$ $email_1$ $action_1$ $infoBooking_1$… $idPerson_i$ $email_i$ $action_i$ $infoBooking_i$… $idPerson_n$ $email_n$ $action_n$ $infoBooking_n$ -1>

On:

- $idPerson_i$ és un enter positiu identificador de la persona *i* que fa la reserva.
- $email_i$ és l'adreça de correu electrònic de la persona *i* que fa la reserva.
- $action_i$ és un valor enter que pot tenir el valor 0 o 1 i indica l'acció a realitzar. El valor 0 indica que es fa una reserva, i el valor 1 que es vol cancel·lar una reserva ja realitzada.
- $infoBooking_i$ és informació addicional, i que únicament té dades si s'està fent una reserva (en cas d'una cancel·lació aquesta informació està buida). Aquesta seubseqüència de dades té el següent format:
  <$typeRoom_i$ $numberPeople_i$ $withTV_i$ $maxPrice_i$>
  on:
  - $typeRoom_i$ és un caràcter que indica si en la reserva *i* es vol reservar un apartament ('A') o una habitació ('R').

    - *numberPeople*$_i$ és un enter positiu que indica el nombre de persones que s'allotjaran.
    - *withTV*$_i$ és un booleà que indica si es vol tenir TV o no.
    - *maxPrice*$_i$ és un real amb el preu màxim per nit que està disposada a pagar la persona *i* que fa la reserva.
- **-1** és la marca final de la seqüència d'entrada.

Podem suposar que les dades d'entrada són correctes, i coherents amb les estructures de dades definides i que:

- Sempre hi ha un -1 al final de la seqüencia d'entrada.
- Una persona únicament pot tenir activa una reserva en tot l'hotel.
- Existeix el tipus *tString* (sense necessitat de definir-lo) per guardar cadenes de caràcters, i les funcions relacionades *readString* (per llegir un tipus *tString* del canal d'entrada estàndard), i *writeString* (per escriure un tipus *tString* pel canal de sortida estàndard).

El següent algorisme llegeix del canal d'entrada estàndard una seqüencia amb aquest format i genera una seqüencia de sortida pel canal estàndard de sortida on apareix per cada apartament/habitació **el nombre que queda disponible després de processar les reserves i cancel·lacions, i les persones que es queden sense reserva**, tal i com es mostra a continuació::

<type$_1$ *idAptRoom*$_1$ *free*$_1$ … *type*$_i$ *idAptRoom*$_i$ *free*$_i$ … *type*$_n$ *idAptRoom*$_n$ *free*$_n$ *personWithouth -1>*

En concret, mostra per cada tipus d'apartament/habitació:

- *type*$_i$ és un caràcter que indica si és un apartament o una habitació (el caràcter 'A' per apartament i el caràcter 'R' per habitació).
- *idAptRoom*$_i$ és un enter que identifica el tipus d'apartament o habitació, seguint els següents identificadors:
    - Valor enter 1 per a apartament per 2 persones.
    - Valor enter 2 per a apartament per 4 persones.
    - Valor enter 3 per a apartament per més de 4 persones.
    - Valor enter 1 per a habitació per 2 persones amb TV.
    - Valor enter 2 per a habitació per 2 persones sense TV
    - Valor enter 3 per a habitació per 4 persones amb TV.
    - Valor enter 4 per a habitació per 4 persones sense TV.

- *free*$_i$ és un enter amb el nombre d'habitacions de tipus *i* que queden lliures.

I a continuació es mostra:

- *personWithouth* és un enter que conté el nombre de persones a les que no se'ls ha pogut fer la reserva sol·licitada.
- *-1* per marcar el final de la seqüència.

Es demana que completeu els requadres.

```
const
        {Constants per diferenciar si s'està fent una reserva o una cancel·lació}
         ACTION_BOOKING : enter = 0;
         ACTION_CANCELLATION : enter = 1;

        {Constants per diferenciar si s'està sol·licitant un apartament o una habitació }
         TYPE_APARTMENT : caracter = 'A';
         TYPE_ROOM : caracter = 'R';

         MAX_APARTS_ROOMS : enter = 50;

         NUM_TYPE_APT : enter = 3;
         TYPE_APT_2: enter = 1;
         TYPE_APT_4: enter = 2;
         TYPE_APT_MORE_4: enter = 3;

         NUM_TYPE_ROOM: enter = 4;
         TYPE_ROOM_2_TV: enter = 1;
         TYPE_ROOM_2: enter = 2;
         TYPE_ROOM_4_TV: enter = 3;
         TYPE_ROOM_4: enter = 4;

         ID_NO_PERSON: enter = -1;
         MAX_PERSON: enter = 1000;

         ENDSEQ : enter  =  -1;
fconst

tipus
        tPerson = tupla
                idPerson: enter;
                email: tString;
        ftupla

        tApartmentRoom = tupla
                idPerson: enter;
        ftupla

        tBookApartmentRoom = tupla
                atpRooms: taula[MAX_APARTS_ROOMS] de tApartmentRoom;
                numAptRooms: enter;
        ftupla


        tHotel = tupla
                bookApartments: taula[NUM_TYPE_APT] de tBookApartmentRoom;
```

```
            bookRooms: taula[NUM_TYPE_ROOM] de tBookApartmentRoom;
            personNoBooking: taula[MAX_PERSON] de tPerson;
            numPersonNoBooking : enter;

        ftupla
ftipus


{ Pre: En el canal estàndard d'entrada tenim una seqüencia amb el format < idPerson1 email1 action1 infoBooking1… idPersoni emaili actioni infoBookingi… idPersonn emailn actionn infoBookingn -1> }
algoritme hotelBooking

        var
            hotel : tHotel;
            infoPerson: tPerson;
            action : enter;
            bOkBooking : boolea;
        fvar

        {Inicialització}
        initializeHotel(hotel);

        {Llegim la seqüència d'entrada}
        infoPerson.idPerson := readInteger();

        mentre (infoPerson.idPerson ≠ ENDSEQ) fer
            infoPerson.email := readString();

            {Llegim les dades de la reserva o cancel·lació}
            action := readInteger();

            si action = ACTION_BOOKING llavors
                readAndProcessBooking(infoPerson, hotel, bOkBooking);
                si no bOkBooking llavors
                    insertPersonNoBooking(infoPerson, hotel);
                fsi
            sino

            fsi
        fmentre

        printResults(hotel);
falgoritme
{ Post: En el canal estàndard de sortida hi ha una seqüencia que mostra per cada tipus d'apartament/habitació el nombre que ha quedat lliure, i a continuació el nombre de persones que s'han quedat sense la reserva sol·licitada.}
```

```
accio initializeHotel (sor hotel: tHotel)
        var
                i: enter;
        fvar
        per i := 1 fins NUM_TYPE_APT fer
                initializeBookApartmentRoom(hotel.bookApartments[i]);
        fper

        per i := 1 fins NUM_TYPE_ROOM fer
                initializeBookApartmentRoom(hotel.bookRooms[i]);
        fper
        hotel.numPersonNoBooking := 0;
faccio

accio initializeBookApartmentRoom(sor book: tBookApartmentRoom)

faccio

accio insertPersonNoBooking(ent person: tPerson, entsor hotel: tHotel)
        hotel.numPersonNoBooking := hotel.numPersonNoBooking + 1;
        hotel.personNoBooking[hotel.numPersonNoBooking].idPerson := person.idPerson;
        hotel.personNoBooking[hotel.numPersonNoBooking].email := person.email;
faccio
```

{ Pre: En el canal estàndard d'entrada tenim una seqüencia amb el format < $typeRoom_i$ $numberPeople_i$ $withTV_i$ $maxPrice_i$ > }

```
accio readAndProcessBooking(ent person: tPerson,
                                entsor hotel: tHotel, sor bOkBooking: boolea)
        var
                typeRoom, numberPeople: enter;
                withTV: boolea;
                maxPrice : real;
                infoPerson: tPerson;
                isBooking : boolea;
        fvar

        typeRoom := readCharacter();
        numberPeople := readInteger();
        withTV := readBoolean();
        maxPrice := readReal();

        si typeRoom = TYPE_APARTMENT llavors
```

```
            bookingApartment(person, numberPeople, hotel, bOkBooking);
        sino
            bookingRoom(person, numberPeople, withTV, hotel, bOkBooking);
        fsi
faccio
```

{ Post: S'ha llegit la informació de reserva del canal d'entrada estàndard i s'ha processat.}

```
accio bookingApartment(ent person: tPerson, ent numberPeople: enter,
                       entsor hotel: tHotel, sor bOkBooking: boolea)
        var
            index : enter;
        fvar

        si numberPeople > 4 llavors
            index := TYPE_APT_MORE_4;
        sino si numberPeople ≤ 4 i numberPeople > 2 llavors
            index := TYPE_APT_4;
          sino
            index := TYPE_APT_2;
          fsi
        fsi
        addBookingApartmentRoom(person, hotel.bookApartments[index], bOkBooking);

faccio

accio bookingRoom(ent person: tPerson, ent numberPeople: enter, ent withTV: boolea,
                  entsor hotel: tHotel, sor bOkBooking: boolea)
        var
            index : enter;
        fvar

        si numberPeople = 4 llavors
            si withTV llavors
                index := TYPE_ROOM_4_TV;
            sino
                index := TYPE_ROOM_4;
            fsi
        sino
            si withTV llavors
                index := TYPE_ROOM_2_TV;
            sino
                index := TYPE_ROOM_2;
            fsi

        fsi
        addBookingApartmentRoom(person, hotel.bookRooms[index], bOkBooking);
```

```
faccio

accio addBookingApartmentRoom(ent person: tPerson,
                              entsor book: tBookApartmentRoom, sor bOk: boolea)

        bOk := (book.numAptRooms < MAX_APARTS_ROOMS);
        si bOk llavors
                book.numAptRooms := book.numAptRooms + 1;
                book.aptRooms[book.numAptRooms].idPerson :=person.idPerson;
        fsi

faccio

accio processCancellation(ent person: tPerson, entsor hotel: tHotel)
        var
                found: boolea;
                idPerson: enter;
                i : enter;
        fvar

        i := 1;
        found := fals;
        idPerson := person.idPerson;

        {Buscamos la reserva en los apartamentos}
        mentre no found i i ≤ NUM_TYPE_APT fer
                searchBookingAndCancell(idPerson, hotel.bookApartments[i], found);
                i := i + 1;
        fmentre

        {Si no correspondía con un apartamento, buscamos la reserva en las habitaciones}
        i := 1;
        mentre no found i i ≤ NUM_TYPE_ROOM fer
                searchBookingAndCancell (idPerson, hotel.bookRooms[i], found);
                i := i + 1;
        fmentre
faccio
```

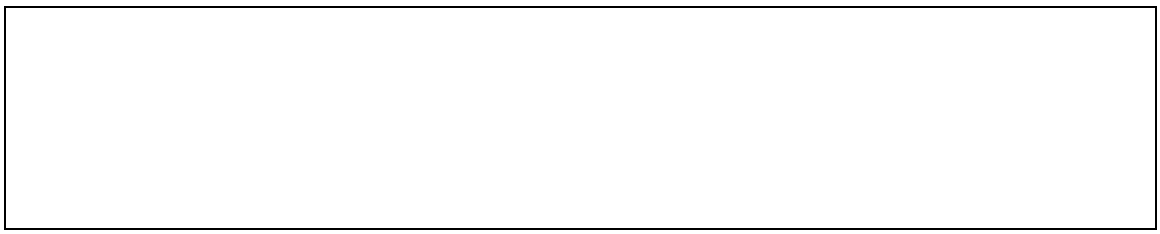
{ *Pre: idPerson = IDPERSON, book = BOOK i found = FOUND,* on IDPERSON és l'identificador de persona que realitza la cancel·lació, BOOK és el tipus en el que tenim que buscar l'identificador de persona *}*

```
accio searchBookingAndCancell (ent idPerson: enter, entsor book: tBookApartmentRoom, sor
found: boolea)
```

faccio

{ *Pre: Si existeix IDPERSON a BOOK, FOUND té valor* cert*, i s'ha realitzat la cancel·lació: s'ha eliminat la persona de la taula de reserves, i s'han mogut les reserves de la taula per no deixar espais dins la taula. En cas contrari FOUND té valor* fals *}*

accio printResults(**ent** hotel: tHotel)



faccio

Exercici 4: Disseny/modificació d'algorismes [20%]

Objectius: Dissenyar un algorisme, que pugui contenir esquemes de cerca/recorregut, i que tracti amb taules/tuples.

Materials: *Mòdul 4: Tipus estructurats de dades*

M 4.1: Introducció i motivació. Estructuració de dades

M 4.2: Taules

M 4.3: Tuples

Tasca: Partint de l'algorisme de l'exercici 3, feu les modificacions adequades a fi de que l'algorisme tingui en compte també el preu màxim que està disposada a pagar la persona que fa la reserva.

Per això, fa falta que respongueu els següents apartats:

i) **[10%]** Dissenyeu una funció *priceApartmentRoomBooking* que retorni el preu en euros de cada tipus d'apartament/habitació, donat un caràcter amb el tipus ('A' per apartaments, i 'R' per habitacions), i un paràmetre de tipus enter amb el tipus d'apartament o habitació, seguint la següent relació:

a. Valor enter 1 per a apartament per 2 persones.
b. Valor enter 2 per a apartament per 4 persones.
c. Valor enter 3 per a apartament per més de 4 persones.
d. Valor enter 1 per a habitació per 2 persones amb TV.
e. Valor enter 2 per a habitació per 2 persones sense TV
f. Valor enter 3 per a habitació per 4 persones amb TV.
g. Valor enter 4 a per habitació per 4 persones sense TV.

Els preu per nit per apartament/habitació per el mes de desembre (tenint en compte les persones que si poden allotjar) són:

a. Apartament per a 2 persones: 100€/nit.
b. Apartament per a 4 persones: 140€/nit.
c. Apartament per a més de 4 persones: 200€/nit.
d. Habitació per a 2 persones amb TV: 80€/nit.
e. Habitació per a 2 persones sense TV: 60€/nit.
f. Habitació per a 4 persones amb TV: 160€/nit.
g. Habitació per a 4 persones sense TV: 120€/nit.

ii) **[10%]** Adapteu l'algorisme de l'exercici 3 per que quan es processa una reserva tingui en compte el preu màxim que està disposada a pagar la persona interessada, i no faci la reserva en cas que l'apartament o habitació sobrepassi aquest pressupost màxim.

Exercici 5: Codificació en C [30%]

Objectius: Donada una definició de tipus que presenta una combinació de taules i tuples, es demana la traducció a C d'un algorisme que accedeixi a aquestes estructures.

Materials: *Mòdul 4: Tipus estructurats de dades*

M 4.1: Introducció i motivació. Estructuració de dades

M 4.2: Taules

M 4.3: Tuples

Tasca: El següent algorisme és utilitzat pel magatzem general d'una cadena de botigues per mantenir la gestió dels estocs dels seus productes. Per fer aquest manteniment utilitza dues seqüències d'entrada, que es llegeixen del canal d'entrada estàndard i que representen respectivament l'estoc del magatzem a l'inici del dia i les peticions que rep de les botigues.

La primera seqüencia té el següent format d'entrada:

$<Prod_1 \quad Units_1 \quad Min_1 \quad ... \quad Prod_i \quad Units_i \quad Min_i \quad ... \quad Prod_N \quad Units_N \quad Min_N \quad -1>$

On

- $Prod_i$ és un enter positiu que identifica de manera única l'i-èssim producte del magatzem.
- $Units_i$ és un enter que indica la quantitat d'unitats del i-èssim producte que hi ha al magatzem a l'inici del dia.
- Min_i és un enter que indica el nivell mínim d'estoc desitjable al magatzem pel i-èssim producte.
- *-1* és l'element que marca el final de la seqüencia.

Els productes, a la seqüència d'entrada no segueixen cap ordre pre-establert, i cada producte tant sols apareix una única vegada a la seqüència d'entrada.

La segona seqüencia es correspon amb les comandes del dia fetes per les diferents botigues de la cadena. El format és el següent:

$<Store_1 \quad NumProd_1 \quad Prod_{11} \quad Units_{11} \ldots \quad Prod_{1i} \quad Units_{1i} \quad ... \quad Prod_{1NumProd1} \quad Units_{1NumProd1}$
$Store_2 \quad NumProd_2 \quad Prod_{21} \quad Units_{21} \ldots Prod_{2i} \quad Units_{2i} \quad ... \quad Prod_{2NumProd2} \quad Units_{2NumProd2}$
....
$Store_m \quad NumProd_m \quad Prod_{m1} \quad Units_{m1} \quad Prod_{m2} \quad Units_{m2} \quad ... \; Prod_{mNumProdm} \quad Units_{mNumProdm}$
$-1>$

on

- $Store_i$ és un enter positiu que indica el codi de la botiga que realitza la comanda. Com a molt, la cadena té 50 botigues i els seus codis van del 1 al 50.
- $NumProd_i$ és un enter que indica el nombre de productes que inclou en la seva comanda la botiga i-èssima.
- $Prod_{ij}$ és un enter que identifica el producte j-èssim de la comanda de la botiga i-èssima. Tots els codis de producte seran correctes, i existiran en la primera seqüencia d'entrada.
- $Units_{ij}$ és un enter que indica la quantitat d'unitats sol·licitades del producte j-èssim en la comanda i-èssima.
- *-1* és el valor que indica el final de la seqüencia.

El següent algorisme llegeix les dues seqüències d'entrada i genera una seqüència de sortida amb la llista de productes i unitats que cal comprar per mantenir el magatzem amb els estocs. Per això, per cada producte que després de processar les comandes té el seu estoc per sota del mínim s'ha de calcular quantes unitats s'ha de sol·licitar per cobrir l'estoc mínim + 50.

L'algorisme també calcula quin és el producte més venut i les unitats totals venudes, i quines botigues l'han sol·licitat i amb quina quantitat. El format de sortida ha de ser:

<$Prod_1$ $Left_1$ $Quantity_1$ … $Prod_x$ $Left_x$ $Quantity_x$ -1 MaxProdCode MaxProdUnits $Store_1$ $Quantity_1$ …$Store_t$ $Quantity_t$ -1>

On

- $Prod_i$ és un enter que indica el codi del producte i-èssim
- $Left_i$ és un enter que indica el nombre d'unitats del i-èssim producte que queden al magatzem al final del dia
- $Quantity_i$ és un enter que indica el nombre d'unitats que s'han de comprar del producte i-èssim
- *MaxProdCode* és el codi del producte que més s'han demanat.
- *MaxProdUnits* és el nombre d'unitats que les botigues han demanat del producte més sol·licitat.
- $Store_i$ és el codi de la botiga i-èssima
- $Quantity_i$ és la quantitat d'unitats de *MaxProdCode* que ha sol·licitat la botiga.

Per exemple, si la seqüència d'entrada és

<1 200 20 2 50 10 3 400 100 -1

1 2 2 10 3 100

2 1 3 100

3 3 1 40 2 35 3 200 -1>

La sortida serà

<1 160 0 2 5 55 3 0 150 -1

3 400 1 100 2 100 3 200 -1>

```
const
      MAXPRODUCTS: enter = 100;
      MAXSTORES: enter=50;
      MARGIN: enter  = 50;
      ENDSEQ: enter = -1;

fconst

tipus
tProduct= tupla
                  productCode: enter;
                  availableUnits: enter;
                  minStock: enter;
                  unitsRequested:  enter;
             ftupla

tStock =  tupla
                  products: taula [MAXPRODUCTS] de tProduct;
                  numProd: enter;
            ftupla

tStoreOrder = tupla
                  order: taula [MAXPRODUCTS] de tProduct;
                  numProducts: enter;
                  ftupla


      tChain = tupla
                  stores: taula [MAXSTORES] de tStoreOrder;
                  numStores: enter;
                  ftupla
ftipus

algoritme wareHouseManagement
    var
        stock: tStock;
        chain: tChain;
        currentCode: enter;
        product : tProduct;
    fvar

    stock.numProd:= 0;
    currentCode:= readInteger();

   {llegim la seqüència d'estocs}
    mentre (currentCode ≠ ENDSEQ )  fer
        product.productCode:= currentCode;
        readProduct(product);
        updateWareHouse(stock, product);
        currentCode:= readInteger();
    fmentre
```

```
    {inicialitzem chain}
    initializeChain(chain);

    currentCode:= readInteger();

    mentre (currentCode ≠ ENDSEQ )  fer
        processOrder(stock, chain, currentCode);
        currentCode:= readInteger();
    fmentre

    computeResults(stock, chain);
falgoritme


accio readProduct( entsor product: tProduct)
        product.availableUnits:= readInteger();
        product.minStock:= readInteger();
faccio


accio initializeChain(entsor chain: tChain)
var
      i:enter;
fvar

      per i:=1 fins  MAXSTORES  fer
            chain.stores[i].numProducts:= 0;
      fper
      chain.numStores:= 0;

faccio


accio updateWareHouse ( entsor stock: tStock, ent product: tProduct)
    var
        i: enter;
    fvar

    i:=stock.numProd + 1;
    stock.products[i].productCode:=product.productCode;
    stock.products[i].availableUnits:=product.availableUnits;
    stock.products[i].minStock:=product.minStock;
    stock.products[i].unitsRequested:=0;
    stock.numProd:=stock.numProd +1;
faccio

accio processOrder ( entsor stock: tStock, entsor chain: tChain; ent storeCode: enter)
    var
        numProducts: enter;
        j: enter;
        product: tProduct;
    fvar


    numProducts:=readInteger();
```

```
    per j:=1 fins numProducts fer
        product.productCode:=readInteger();
        product.unitsRequested:=readInteger();

        addProductToStore(chain, storeCode, product);

        {actualizar l'estoc}
        addProductToStock(stock, product);
    fper
faccio


accio addProductToStore ( enrtsor chain: tChain,
                                    ent storeCode: enter, ent product: tProduct)
var
    i:  enter;
    found:  boolea;
fvar

    i:= 1;
    found:=FALSO;
    mentre  i<=chain.stores[storeCode].numProducts  i found = FALS fer
        found := chain.stores[storeCode].order[i].productCode = product.productCode;
        si  found=FALSE  llavors
            i:=i+1;
        fsi
    fmentre

    chain.stores[storeCode].order[i].availableUnits:= chain.stores[storeCode].order[i].availableUnits
                                                        - product.unitsRequested;
    chain.stores[storeCode].order[i].unitsRequested:=
chain.stores[storeCode].order[i].unitsRequested + product.unitsRequested;

    si found=FALSE  llevors
        chain.stores[storeCode].numProducts := chain.stores[storeCode].numProducts  +1;
        chain.stores[storeCode].order[i].productCode:= product.productCode;
    fsi
faccio


accio addProductToStock( entsor stock: tStock, ent product: tProduct)
    var
        i: enter;
        found:boolea;
    fvar

    i:=1;
    found:= FALSE;

    {busquem el producte, que sabem que ha d'estar a la taula de productes}
   mentre i<=stock.numProd i no found  llavors
        si stock.products[i].productCode ≠ product.productCode  fer
            i := i +1;
        sino
            found := TRUE;
        fsi
```

```
   fmentre

    stock.products[i].availableUnits:=
                stock.products[i].availableUnits – product.unitsRequested;
    stock.products[i].unitsRequested:=
                 stock.products[i].unitsRequested + product.unitsRequested;
faccio


accio computeResults(ent stock:tStock, ent chain:tChain)

var
       i: enter;
       j: enter;
       maxProductUnits:  enter;
       maxProductCode: enter;
    fvar

   maxProductUnits:= 0;
   maxProductCode:= 0;

    per i:= 1 fins stock.numProd fer
       writeInteger(stock.products[i].productCode );
       writeInteger(stock.products[i].availableUnits);
       si (stock.products[i].availableUnits  ≥ stock.products[i].minStock  ) llavors
          writeInteger(0);
       sino
          writeInteger(stock.products[i].minStock  - stock.products[i].availableUnits + MARGIN)
       fsi

       si  maxProductUnits < stock.products[i].unitsRequested  llavors
          maxProductUnits:=  stock.products[i].unitsRequested;
          maxProductCode:=   stock.products[i].productCode;
        fsi
    fper

    writeInteger(ENDSEQ);
    writeInteger(maxProductCode);
    writeInteger(maxProductUnits);

    per i:= 1 fins MAXSTORES fer
       per j:=1 fins chain.stores[i].numProducts fer
          si chain.stores[i].order[j].productCode = maxProductCode  llavors
             writeInteger(i);
             writeInteger(chain.stores[i].order[j].unitsRequested);
          fsi
       fper
    fper

    writeInteger(ENDSEQ);

faccio
```