Universitat Oberta de Catalunya uoc.edu

Fundamentos de Programación

PEC10 - 20191

Fecha Límite de entrega: 23/12/2019

Estudiante

Apellidos:

Nombre:

Objetivos

- Comprender qué son los tipos abstractos de datos (TAD) y saberlos definir correctamente.
- Implementar operaciones con TAD.
- Profundizar en la modularización del código utilizando acciones y funciones.
- Profundizar en el uso de parámetros de entrada, parámetros de salida y parámetros de entrada / salida.

Formato y fecha de entrega

Hay que entregar la PEC antes del día **23 de diciembre de 2019 a las 23:59**.

Para la entrega deberá entregar un archivo en formato ZIP, que contenga:

- Este documento con la respuesta del ejercicio 1.
- El workspace de Codelite que contenga **el proyecto y todas las carpetas creadas** en el ejercicio 2

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

Enunciado

Siguiendo con la ayuda que proporcionamos a UOCBookings, la compañía nos ha pedido ahora nuestra colaboración para crear un programa que les ayude a gestionar los movimientos de tickets de los gastos que los clientes generan en los diversos establecimientos de un hotel: restaurante, bar, peluquería, tienda, etc, y que se cargan a la habitación.

En esta PEC, trabajaremos con el tipo abstracto de datos (TAD) pila.

Cada ticket tendrá la siguiente información:

- roomNum: entero que representa el número de la habitación a la que se carga el ticket.
- source: establecimiento en el que se ha originado el ticket, y puede tener los valores siguientes: BAR, RESTAURANT, BEAUTY_SALOON, SHOP.
- date: un entero que representa la fecha en la que se ha emitido el ticket y tiene el formato ***aaaammdd***. Por ejemplo, el valor 20191211 representa el día 11 de diciembre de 2019.
- hour: un entero que representa la hora en la que se ha emitido el ticket y tiene el formato ***hhmm***. Por ejemplo, el valor 1345 representa las 13 horas y 45 minutos.
- amount: un real que representa el importe del ticket.

Cada establecimiento usa un TAD pila en el que va apilando los tickets de los clientes.

Para resolver los problemas planteados tenéis a vuestra disposición las siguientes declaraciones y cabeceras de funciones/acciones, que no es necesario diseñar.

```
type

tSource = {BAR, RESTAURANT, BEAUTY_SALOON, SHOP}
tTicket = record
    roomNum: integer;
    source:tSource;
    date: integer;
    hour:integer;
    amount: real;
end record
end type
```

function createStack() : tStack

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

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

function top (s: tStack): tTicket

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

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

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

Nota: La definición de estas acciones / funciones es de acuerdo a lo que tienes ya definido en la XWiki.

También se dispone de la función

function cmpTickets(t1: tTicket, t2: tTicket): **integer**

que compara la fecha y hora de emisión de ambos tickets y devuelve

1 si t1 tiene una fecha/hora de emisión posterior a la del t2

0 si la fecha/hora de emisión de ambos tickets son idénticas

-1 si t1 tiene una fecha/hora de emisión anterior a la del t2

Ejercicio: Diseño en lenguaje algorítmico (50%)

Apartado 1.a: [30%]

Diseñar la acción ***addTickets,*** que dada una pila, *src,* y el número de una habitación, *roomNumber*, devuelve la suma de todos los importes de los tickets cargados a la habitación indicada. Los parámetros deben ser:

De entrada:

- *src* – una pila de tickets,
- *roomNumber* – el número de una habitación

De salida

- *totalAmount* – que es la suma de los importes de todos los tickets de la pila cargados a *roomNumber*.

Nota: En el diseño de la acción hay que usar las acciones y funciones que gestionan el TAD pila

Apartado 1.b: [70%] Diseñar la acción ***mergeTicketStacks*** que recibe como parámetros de entrada dos pilas de tickets que están ordenadas por fecha/hora de generación (con el ticket más antiguo en la base de la pila y el más reciente en la parte superior de la misma) y devuelve como parámetro de salida una tercera pila que contiene la fusión de ambas pilas y en la que se mantiene el orden inicial de los tickets (del más antiguo al más reciente). Si dos tickets tienen exactamente la misma fecha y hora, el de la pila 1 se considera más antiguo. Los parámetros deben ser:

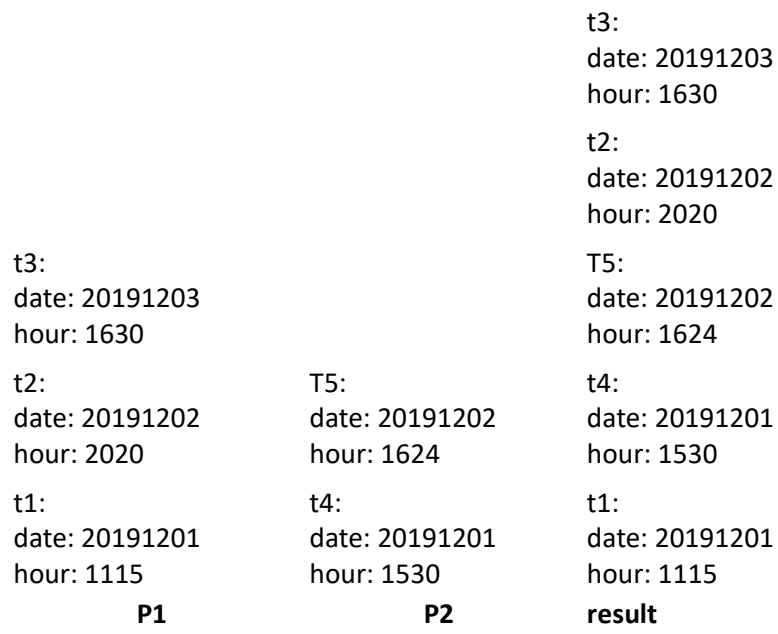
De entrada:

- *src1* – pila de tickets, con los tickets ordenados por fecha/hora de emisión.
- *src2* – pila de tickets, con los tickets ordenados por fecha/hora de emisión

De salida

- *dest* – pila de tickets, con todos los tickets de las pilas src1 y src2 ordenados por fecha/hora de emisión

La imagen siguiente muestra un ejemplo de dos pilas de entrada y el resultado que debe generar la acción.



Ejercicio 2: Programación en C (50%)

En este ejercicio hay que codificar en C las dos acciones de la pregunta 1 en el fichero stack.c, completando el código proporcionado y siguiendo la misma estructura de carpetas y ficheros.

Concretamente se debe

1. añadir a stack.h las cabeceras de las dos acciones diseñadas en la pregunta 1.
2. añadir a stack.c el código de las dos acciones diseñadas en la pregunta 1.

Se proporciona un proyecto inicial UOCBookings en el que ya hay implementadas las funcionalidades de manipulación de pilas así como funcionalidades para leer tickets y código que prueba las acciones. Concretamente el código proporcionado está estructurado en carpetas: carpeta *include* con los ficheros ***data.h, ticket.h, stack.h y test.h*** y la carpeta *src* con los ficheros ***main.c, ticket.c, test.c y stack.c.***

Para cada uno de los apartados del ejercicio 1, hay el correspondiente test en el fichero test.c para verificar el correcto funcionamiento de la codificación que se ha realizado. Cada test tiene un comentario indicando cual es su objetivo.

Criterios de corrección:

Para el ejercicio 1:

- Que se siga la notación algorítmica utilizada en la asignatura. Véase documento Nomenclator en la XWiki de contenido.
- Que se siguen las instrucciones dadas y el algoritmo responda al problema planteado.
- Que se diseñen y se llamen correctamente las acciones y funciones pedidas.

Para el ejercicio 2:

- Que el programa se adecue a las indicaciones dadas.
- Que el programa compile y funcione de acuerdo a lo que se pide.
- Que se respeten los criterios de estilo de programación C. Véase la Guía de estilo de programación en C que tiene en la Wiki de contenido.
- Que se implemente correctamente la modularización del proyecto, dividiendo el código en carpetas y poniendo lo que corresponde en cada carpeta.