UNIVERSIDADE DA CORUÑA

FACULTY OF COMPUTER SCIENCE

Programming II - Course 2021/22

Practical 2: Instructions

1. The Problem

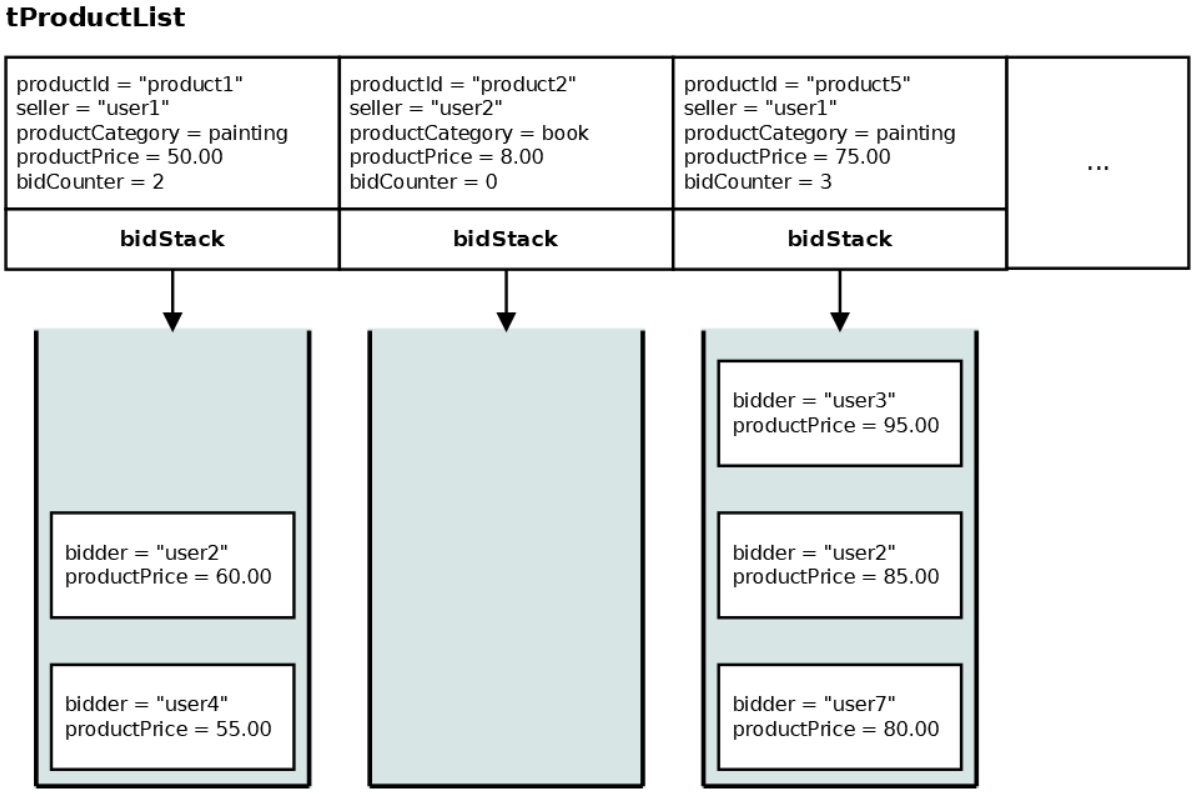
This second practical consists in adding a set of new features to BIDFIC, the system previously developed in the first practical. Specifically, a **bid stack will be associated to every product**. The aim of this work is to understand how several abstract data types (ADTs) work, how they can be implemented and how the interdependencies between them can be managed.

2. Data Structures

To solve the problem, two different abstract data types (ADTs) will be used:

- An Ordered List (ADT ProductList) to store the list of auctioned products.
- A Stack (ADT BidStack) to store the bids made for each product.

These two ADTs will be combined. Thus, every BIDFIC product, that is, every node of the ordered list, will have its own bid stack, as shown in the following figure:



2.1 Header File `types.h`

Some data types will be defined in the header file `types.h`, since they are used by several ADTs.

`NAME_LENGTH_LIMIT`	Maximum length of user and product identifiers (constant)
`tUserId`	User identifier (`string`)
`tProductId`	Product identifier (`string`)
`tProductCategory`	Category of a product (enumerated type: `{book, painting}`)
`tProductPrice`	Price of a product (`float`)
`tBidCounter`	Bid counter (`int`)

2.2 ADT Product List

The system will make use of a **singly-linked dynamic implementation** of the ADT Ordered List to hold the list of products (`product_list.c` and `product_list.h`).

2.2.1. Data types included in the ADT Product List

`tList`	Represents a list of products **ordered alphabetically** by their identifiers (`productId`)
`tItemL`	Data for an element of the list (a product). It contains: • `seller`: type `tUserId` • `productId`: type `tProductId` • `productCategory`: type `tProductCategory` • `productPrice`: type `tProductPrice` • `bidCounter`: type `tBidCounter` • `bidStack`: type `tStack` (stack with the bids for that product)
`tPosL`	Position of an element of the list
`LNULL`	Constant used to represent null positions in the list

2.2.2. Operations included in the ADT Product List

The operations included in this ADT are identical to those of Practical 1 (see instructions of such practical). The only exceptions are the following operations, whose specifications have changed:

- `insertItem (tItemL, tList) → tList, bool`
  Inserts an element in the List **ordered** by the field `productId`. If the element could be inserted, the value `true` is returned; otherwise `false` is returned.
  PostCD: The positions of the elements in the list following that of the inserted one may have varied.

- `deleteAtPosition (tPosL, tList) → tList`
  Deletes the element at the given position from the list.
  PreCD: The indicated position is a valid position in the list, and **the bid stack of the product at that position is empty**.
  PostCD: The positions of the elements in the list following that of the deleted one may have varied.

Moreover, the operation `findItem` does not change its specification, but its implementation must now take into account that the list is ordered.

2.3 ADT Bid Stack

This ADT implements a **static stack** containing the bids received for a product (`bid_stack.c` and `bid_stack.h`). Its maximum size must be of 25 bids.

2.3.1. Data types included in the ADT Bid Stack

`tStack`	Represents a stack of bids
`tItemS`	Data of an element of the stack, that is, a bid. It contains: • `bidder`: type `tUserId` (user that makes the bid) • `productPrice`: type `tProductPrice` (price offered by the bidder)
`tPosS`	Position of an element of the stack
`SNULL`	Constant used to represent null positions in the stack

2.3.2. Operations included in the ADT Bid Stack

A common precondition for all these operations (except `createEmptyStack`) is that the stack must be previously initialised:

- `createEmptyStack (tStack) → tStack`
  Creates an empty stack Crea una pila vacía.
  PostCD: The stack has no elements.
- `push (tItemS, tStack) → tStack, bool`
  Inserts an element at the top of the stack. **If the element could be inserted, the value `true` is returned; otherwise `false` is returned.**
- `pop (tStack) → tStack`
  Removes from the stack the element located at its top.
  PreCD: The stack is not empty.
- `peek (tStack) → tItemS`
  Recovers the content of the element at the top of the without removing it.
  PreCD: The stack is not empty.
- `isEmptyStack (tStack) → bool`
  Determines whether the stack is empty or not.

3. Description of the Task

The task consists of implementing a single main program (`main.c`) that, by making use of **Product List** and **Bid Stack**, processes the requests received from BIDFIC users, which follow this format:

`N productId userId productCategory productPrice`	***[N]ew**:* A new product is added
`D productId`	***[D]elete***: The product is removed
`B productId userId productPrice`	***[B]id***: Bid for a given product
`A productId`	***[A]ward:*** The winning bidder for that product is declared
`W productId userId`	***[W]ithdraw***: Current highest bid for that product is withdrawn
`R`	***[R]emove:*** Deletes all products with no bids
`S`	***[S]tats:*** List of current BIDFIC products and their data

The main program will contain a loop to process, one by one, the requests of the users. In order to simplify the development and testing of the system, the program must not prompt the user to input the data of each request. Instead, the program will take as input a file containing the sequence of requests to be executed (see document `RunScript.pdf`). For each loop iteration, the program will read a new request from the file and then process it. In order to make correction easier, all requests in the input file have been numbered consecutively.

For each line of the input file, the program will do the following:

1. A header with the operation to be performed is shown. This header consists of a first line with 20 asterisks and a second line indicating the operation as shown below:

```
********************
NN␣T:␣product␣PP␣seller/bidder␣UU␣category␣CC␣price␣ZZ
```

where `NN` is the number of the request; `T` is the type of operation (`N`, `D`, `B`, `A`, `W`, `R` or `S`); `PP` is the identifier of the product; `UU` is the identifier of the user (preceded by the word `seller` or `bidder` depending on the operation); `CC` is the category of the product; `ZZ` is the price of the product (with 2 decimal places); and ␣ represents a blank. Only the necessary parameters are printed; i.e., for a ***[S]tats*** request we will only show "`01 S`", while for a ***[N]ew*** request we will show "`01 N: product Product1 seller User2 category book price 15.00`".

2. The corresponding request is processed:

- If the operation is ***[N]ew***, that product must be added to the product list, with its corresponding product identifier, the user identifier of its seller, its category and price. Its bid counter will be initialized to 0, and an empty bid stack will be created and associated to it. In addition, a message like this will be displayed:

  ```
  * New: product PP seller UU category CC price ZZ
  ```

  In the event that a product with such product identifier already exists, or the insertion cannot be completed, the following message will be printed:

  ```
  + Error: New not possible
  ```

- If the operation is ***[D]elete***, the system will locate and remove that product from the list, also **deleting all its associated bids**. In addition, a message like this will be displayed:

  ```
  * Delete: product PP seller UU category CC price ZZ bids II
  ```

  In the event that there is no product with the given identifier, the following message must be printed:

  ```
  + Error: Delete not possible
  ```

- If the operation is ***[B]id***, the product is located and, if the new bid value exceeds its current highest bid (if there are no bids, we check the original price instead), then: (i) the new bid is added to its stack; (ii) its bid counter is updated; and (iii) a message like this, with the value of the new bid, will be displayed:

  ```
  * Bid: product PP bidder UU category CC price ZZ bids II
  ```

  If there is no product with such identifier (`productId`), the bidder and the seller are the same person, the bid value is not higher than the current highest bid, or the stack is full, the following message must be printed:

  ```
  + Error: Bid not possible
  ```

- If the operation is ***[A]ward***, the product is located and the following message is shown:

  ```
  * Award: product PP bidder UU category CC price ZZ
  ```

  where `UU` is the winning bidder and `ZZ` is the final price. Since the product has been sold, it must be removed from the product list after emptying its bid stack.

  If there is no product with that identifier (`productId`) or its bid stack is empty, the following message will be printed:

```
+ Error: Award not possible
```

In the latter case, with the stack empty, the product will not be removed from the list.

- If the operation is ***[W]ithdraw***, the product is located, the current highest bid is deleted, the bid counter is decreased, and the following message is displayed:

```
* Withdraw: product PP bidder UU category CC price ZZ bids II
```

where, this time, `ZZ` is the amount **of the bid withdrawn** and `II` is the number of bids for that product **before** proceeding to withdraw such bid.

If there is no product with that identifier (`productId`), its bid stack is empty, or the `userId` is not that of the highest bidder, this message will be printed instead:

```
+ Error: Withdraw not possible
```

- If the operation is ***[S]tats***, the whole list of current products will be displayed as follows:

```
Product PP1 seller UU1 category CC1 price ZZ1 bids II1 top bidder BB1
Product PP2 seller UU2 category CC2 price ZZ2. No bids
Product PP3 seller UU3 category CC3 price ZZ3 bids II3 top bidder BB3
...
Product PPn seller UUn category CCn price ZZn bids TTn top bidder BBn
```

where `ZZi` is the original price. Below this list we will also print a table showing, for each product category, the number of products in that category, the sum of their original prices, and their average price. Such table must follow the following format:

```
Category  Products    Price  Average
Book      %8d %8.2f %8.2f
Painting  %8d %8.2f %8.2f
```

Finally, we will show the product for which its current highest bid represents the largest increase (in percentage terms) over its original price:

```
Top bid: Product PP seller UU category CC price ZZ bidder BB top price TT increase RR%
```

where `RR` is said increment in percentage (with 2 decimal places). If there is no bid, the following message will be displayed:

```
Top bid not possible
```

In the event that the product list is empty, the following message must be printed:

```
+ Error: Stats not possible
```

- If the operation is ***[R]emove***, those products with no bids will be removed from the list. The following message will be displayed:

```
Removing product PP1 seller UU1 category CC1 price ZZ1 bids II1
Removing product PP2 seller UU2 category CC2 price ZZ2 bids II2
Removing product PP3 seller UU3 category CC3 price ZZ3 bids II3
...
Removing product PPn seller UUn category CCn price ZZn bids IIn
```

If there is no product without bids, the following message will be printed:

```
+ Error: Remove not possible
```

4. Reading the Input Files

To facilitate the development of this practical, the following materials are provided: (1) a folder `CLion` that includes a template project (`P2.zip`); and (2) a folder `script` which contains a file (`script.sh`) that allows you to test the system with all the test files supplied at once. A document explaining how to run it is also provided (`RunScript_P2.pdf`). Finally, to avoid problems when running the script, it is recommended **NOT to directly copy-paste the text of this document into the source code**, since the PDF format may include invisible characters that may result in (apparently) valid outputs to be considered incorrect.

5. Important Information

The document `DeliveryGuidelines.pdf`, available on the course website, clearly outlines the delivery guidelines to be followed. For an adequate **follow-up of this practical**, two **mandatory partial deliveries** will be made before the deadlines and with the contents indicated below:

- **Checkpoint #1: Friday April 8th, at 22:00.** Implementation and testing of the ADT Product List and the ADT Bid Stack (submission of files `types.h`, `product_list.c`, `product_list.h`, `bid_stack.c` and `bid_stack.h`).

- **Checkpoint #2: Friday April 22nd, at 22:00.** Implementation and testing of part of the operations of the main program: ***[N]ew***, ***[B]id*** and ***[S]tats*** (delivery of the files `types.h`, `product_list.c`, `product_list.h`, `bid_stack.c`, `bid_stack.h` and `main.c`). To check the correct functioning of these operations, input test files `new.txt` and `bid.txt` will be provided.

An automatic assessment will be made using the script provided to check whether the requirements have been accurately fulfilled or not (see `AssessmentCriteria.pdf`).

Final submission deadline: **Friday, April 29th, 2022 at 22:00.**