CS240 — Data Structures

2nd Series of Exercises

Submission Instructions

Exercises can be submitted to the course assistants on **Monday, 30th of October 2023, from 14:00 PM to 15:00 PM** at the TAs' office (**B.208** / **B.210**). Exercises submitted after 15:00 PM on Monday, 30/10/2023, will have a penalty. **Late submissions are accepted in electronic format**, and they must be submitted using the `turnin` program. For more information visit the course website.

Semester:	Spring (2023-24)
University:	University of Crete
Department:	Computer Science
Lecturer:	Panagiota Fatourou
Responsible TA:	Katerina Petraki
Last Modification:	25 / 10 / 2023

Exercise 1 [35 points]

a. A transportation company (e.g. a courier) stores information for the transportation transactions it has completed. Their data management program uses a data structure which saves a record for each transportation transaction. Each record has a unique identifier. Let's assume that the data structure which the company uses is a dynamic simply-linked list, whose elements have the following format:

```
struct courier_order {
  int id;
  int source;
  int destination;
  int dest_continent;
  char date[100];
  struct courier_order *next;
}
```

There are 5 continents in which the company operates; Europe (with ID 1), America (with ID 2), Asia (with ID 3), Africa (with ID 4) and Oceania (with ID 5). The field `dest_continent` inside `struct courier_order` stores the ID of the continent to which a transfer is sent.

Consider that **the list is sorted by the identifiers of the transactions in ascending order**. After many years of operation, the company's Board of Directors decided that they will change the way of managing the company's data so that there is a data structure that will store the information of transportation transactions for each continent. Therefore, there should be 5 lists (as many as the continents), each of which is simply-linked and sorted by the transaction IDs in ascending order.

Following the Board of Directors' decision, as part of the company's data management program, a function should get implemented that will process the original data structure and create five corresponding structures such that: The first will contain records for transactions going to Europe, the second will include records corresponding to transactions going to America, and so on. **Describe a (non-recursive) algorithm that implements this functionality**. The time complexity of this algorithm should be $\mathrm{O}(n)$, where n is the count of elements in the original list containing all the transactions, regardless of continents. [15P]

b. Assume a **doubly-linked list** L.

(i) Present pseudo-code for a non-recursive algorithm that sorts list L. The algorithm should implement `InsertionSort` on the list. Use of auxiliary structures is not permitted for the design of the algorithm. [15P]

(ii) What is the time complexity of your algorithm and why? [5P]

Exercise 2 [35 points]

Suppose a library provides you with access to stacks and queues of characters. The library allows you to define a stack (or a queue) and call the 5 basic functions they have. For example, the definition of a stack (or a queue) is done by writing: `Stack S;` (respectively, `Queue Q;`). The following functions are supported for the stack: (1) `void MakeEpptyStack(Stack S)`, (2) `boolean IsEmptyStack(Stack S)`, (3) `Type Top(Stack S)`, (4) `Type Pop(Stack S)`, (5) `void Push(Stack S, Type x)`. Similarly, the queue supports the following functions: (1) `void MakeEmptyQueue(Queue Q)`, (2) `boolean IsEmptyQueue(Queue Q)`, (3) `Type Front(Queue Q)`, (4) `Type Dequeue(Queue Q)`, (5) `void Enqueue(Queue Q, Type x)`, where `Type` may be any type (`char`, `int`, `float`, `double`, etc.).

a. Show pseudo-code for a non-recursive algorithm, which will read an expression consisting of curly brackets, square brackets and parentheses and will check whether the opening of curly brackets, square brackets and parentheses is compatible with their closure (i.e. left curly brackets, left square brackets and left parentheses, as well as the order in which they are found in the expression, "match" with the curly brackets, right square brackets and right parentheses and the order in which they are presented). For example, the algorithm you plan should print TRUE for the following expression:

```
{{[(((([[{{{()}}}()]{}]()()()))[][[]])]}}
```

and FALSE for the following expression:

```
{{[(((([[{{{()}}}()]{}()()()))[][[]])]}}
```

Only one queue or stack is allowed for the problem's solution. Also, the algorithm must consume (read) the expression only once. Your solution should have the following structure: [8P]

```
while ((ch = getchar()) != '\n') {
  switch(ch) {
    case ch is '{':
      break;
    case ch is '}':
      break;
    ...
  }
}
```

b. Describe a recursive version of the algorithm you presented at question a. [10P]

c. Consider a stack S, each element of which is an integer number. Show pseudo-code for a non-recursive algorithm that will sort S. The algorithm you present should implement `InsertionSort` on the stack. You may use two auxiliary stacks for the design of the algorithm, in addition to S. [10P]

d. Consider a tail Q, each element of which is an integer number and the elements on the queue are sorted in ascending order. Show pseudo-code for an algorithm that deletes all of Q's elements for which their division with 5 results in an even number. Consider you know the number n of elements in Q. The algorithm you are designing must not use any auxiliary structures. [7P]

Notes::

- Algorithms should **only use data structures from the library specified by the exercise**, as well as some auxiliary variables — potentially.
- The use of arrays or other data structures that do not belong to the library is not permitted for storing or processing alphanumeric or character sequences.
- You do not know if the stacks and queues provided by the library are implemented in a static or dynamic manner.

Exercise 3 **[30 points]**

a. In a birthday party, a playlist of songs has been pre-decided. It is stored in a simply-linked list L, in descending order according to each song's duration (the songs with the biggest duration are listened-to first).

(i) Provide pseudo-code for a non-recursive algorithm, that sorts the list in ascending order (according to each song's duration). The designed algorithm should not make use of any auxiliary structures (arrays, lists, queues, stacks). Nevertheless, it can make use of auxiliary variables like pointers etc. Your algorithm should have a time complexity of $\mathrm{O}(n)$, where n is the count of elements in L. **[10P]**

(ii) Provide a recursive version of the algorithm you presented at question a. **[10P]**

b. Consider a doubly-linked, sorted list L. Consider that L has an even number of elements, and specifically this number is $2k$, where k is some integer. Consider that the first k elements of L are sorted in ascending order, while the last k elements of L are sorted in descending order. Sorting is on the unique `id` of each node. Each of the two halves of L (consisting of k elements) can have some elements bigger and some elements smaller than those existing in the other half. You should present pseudo-code for an algorithm that creates a doubly-linked list DL, which is sorted in descending order to the identifier `id`, and include all of L's elements. The list L should remain untouched during the execution of the algorithm. The algorithm should be of $\mathrm{O}(n)$ time complexity, where n is the element count of L. Consider the algorithm knows k (e.g., it takes it as a parameter). **[10P]**