Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2021

Homework 3 – Display All Ordered Subsequences with a Hybrid Linked List Structure

Due: 31/03/2021, Wednesday, 21:00

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You can NOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism will not be tolerated!

Introduction

In this homework, you are asked to implement a program that stores and displays **all ordered subsequences** of some input integers. Conceptually, it is similar to Homework 2, but this time you will need to store all subsequences in a more complex data structure and the processing rules are different. The ordering mode is only ascending (i.e. no descending option). The data structure used will be a hybrid and 2-dimensional one as we will detail later on. Moreover, there are some other rules for storage (and displaying) order of the subsequences.

More formally, an ordered subsequence S_i of a sequence S consists of a series of numbers in S (not necessarily consecutive), which are numerically in ascending order and do not violate the appearance order in S. For example, if S is `1 2 4 3`, the sequence `2 3` is a valid ordered subsequence of S, since it is in ascending ordered and `2` appears before `3` in S. On the other hand, the sequence `4 3` is not a valid sorted subsequence since it is not in ascending order, although there is no appearance order problem. As another example, the sequence `3 4` is not a valid ordered subsequence of S; although this sequence is in ascending order, the appearance order of `3` and `4` does not match the one in S.

In this homework, you will display all of the ordered subsequences on the screen. This display order must be hierarchically ascending such that; (i) subsequences with smaller number of elements will be displayed first, (ii) subsequences with same number of elements will be displayed in ascending ordered manner with respect to their elements' numeric values by giving priority to the earlier elements. For example, all of the ordered subsequences of `1 2 4 3` must be displayed in the following order (actual display output format is a bit different; see sample runs).

```
1
2
```

```
3
4
1   2
1   3
1   4
2   3
2   4
1   2   3
1   2   4
```

The Program Flow, Inputs and Outputs

The only input is the integer number sequence, which is going to be entered in one line as in the second homework (using `getline` and `istringstream`). The ordering mode is not an input; it is always ascending. You can assume that the numbers are entered as integers separated by blank(s) or tab(s), so no input checks are required for the numbers. However, pressing enter without entering any numbers is possible and you have to consider this case by just finishing the program (see sample runs).

Your program should process the non-negative input numbers one by one. If the current non-negative input number exist in any of the subsequences in the data structure, then this input should be ignored by giving an appropriate message (see sample runs). If it does not exist in any of the subsequences in the data structure, then your program should add all of the relevant ordered subsequences, that this input can take place, to the data structure and display a single line message specifying that the number has been processed. We will exemplify this in the next section and also several example cases are in the sample runs.

The negative input numbers have a special meaning: *deletion*. While processing the numbers, whenever your program reads a negative integer, it should delete every subsequence that contains the magnitude of this negative number (i.e. its absolute value), if any, and display a message that specifies the deletion operation (different messages depending on whether the number exists or not; see sample runs). For example, if the input numbers are `1 2 4 -2 3`, then the subsequences will be as follows.

```
1
3
4
1   3
1   4
```

A particular number can be re-added after deletion if it appears again in the input. See sample runs for examples.

After processing all the input values, you have to display the final content of the data structure line by line; each line starts with the number of elements of a subsequence followed by the elements. See sample runs.

The Data Structure to be Used and the Related Declarations

In this homework, you will need to use two different types of linked lists (both are regular one-way linked lists) somehow connected to each other. One type of linked list is to keep an ordered subsequence, in which each node is an element of it. Of course, there might be several such lists

since you will keep all of the ordered subsequences. The node type of this list, `SubSeqNode`, is given below partially.

```
struct SubSeqNode
{
      int value;
      SubSeqNode * next;

      // constructors come here
};
```

The heads of the subsequence linked lists are stored in another type of linked list (so-called *heads list*), of which the node type name is `SubSeqHeadNode`. In this node type, two different pointers are used: one is for the head of a subsequence list, and another one as the next pointer to form the *heads list*. Moreover, the number of elements in the subsequence is also stored in this `SubSeqHeadNode` node. Partial node `struct` of `SubSeqHeadNode` is given below.

```
struct SubSeqHeadNode
{
      int size;   // number of elements in the subsequence
      SubSeqNode * sHead;   // head of the subsequence list
      SubSeqHeadNode * next;

      // constructors come here
};
```

You are requested to design and implement a class for the mentioned hybrid and 2D data structure. Name this class `SubSeqsList`. There is one private data member of this class, which is the head of the heads list, `hHead`. You have to perform all operations via the member functions of this class (including the operations regarding the subsequence lists). Partial class definition is given below. We deliberately did not give the member function prototypes since you are expected to design this class. As hint, we can say that you might need member functions the display the entire structure, to insert one or more subsequence list(s) to the data structure, to delete one or more subsequence list(s) from the data structure, to find whether a particular value exists in the data structure, to delete the entire data structure, etc. You can also add some helper functions to the private part, if you need to do so. You can copy/paste this class definition and structs given above in a header file. However, you have to implement the member functions in another .cpp file (other than the one includes main) and use them in your main program. The main program will be in a separate .cpp file.

```
class SubSeqsList
{
public:
      SubSeqsList(); //default constructor that creates an empty list
      // member functions come here

private:
      SubSeqHeadNode * hHead;
      // any helper functions you see necessary
};
```

Figure 1 depicts the data structure that you will use in this homework.

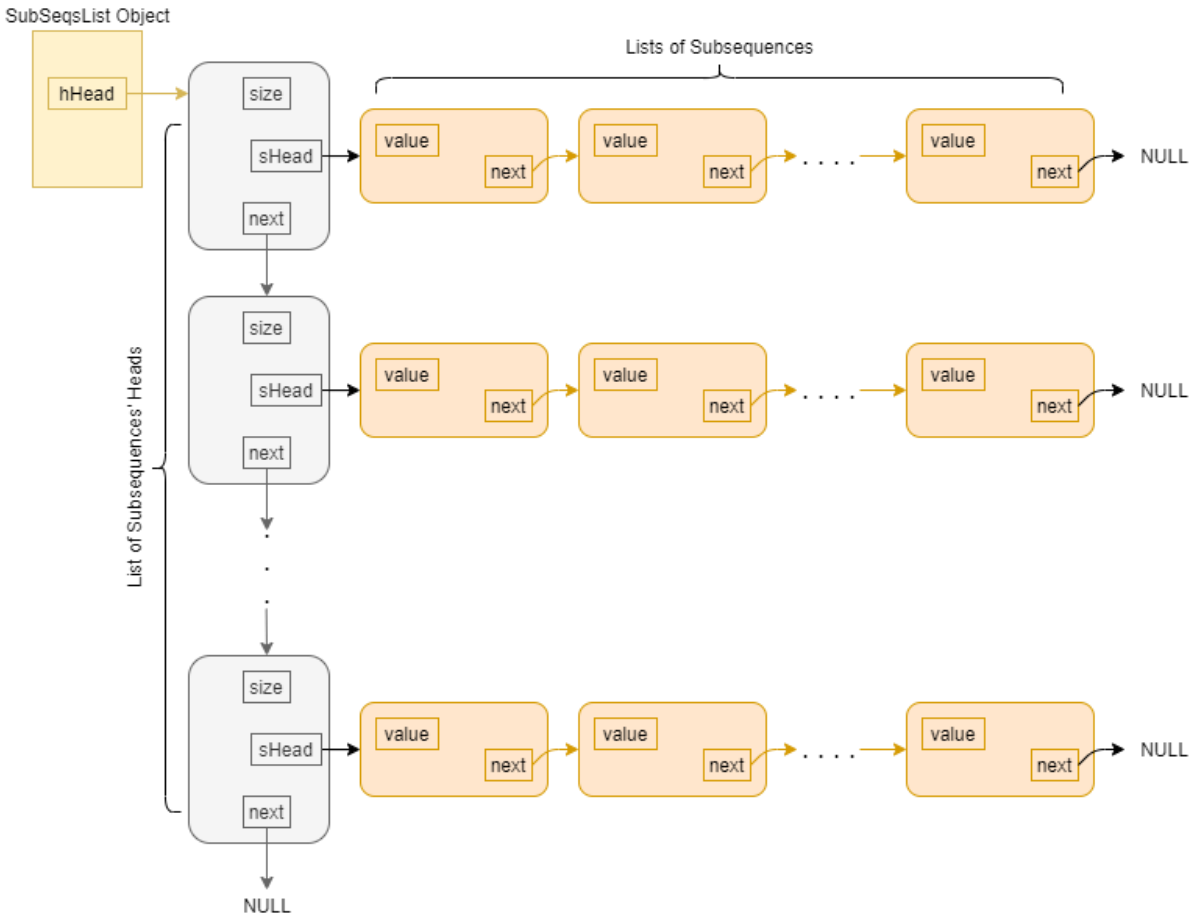

Figure 1. The data structure

<u>No</u> other multiple data containers (built-in array, dynamic array, vector, matrix, 2D array, etc.) can be used. Do not use string data type to avoid these restrictions. Especially do not even think of bypassing the requested negative number processing via such extra storages.

Object Oriented Design Manifest

We believe it is clear that you have to apply proper object oriented design principles in this homework, but our past experience says that most of the students are either unaware or not willing to follow these principles. Thus, we wanted to manifest some important rules about the class design and implementation.

1) According to the rule #1 of object oriented design, each member function **must be multi-purpose and must perform as single task as possible**. For example, reading all the numbers adding/deleting everything to/from the data structure is **<u>not</u>** such a function. We gave some hints about possible member functions above.

2) Another important rule of object oriented programming is to hide the details from the programmer. That means in any free and main function, you cannot reach the head of the data structure (`hHead` and others) directly. You have to make all of the updates, searches, etc. in main program <u>through the member functions</u>. Yes, technically it is possible and really easy to write a member function that returns `hHead` so that you can freely manipulate the data structure in main program, but this is totally against the spirit behind object oriented programming and **do not do this**.

3) In a cpp file, you have to have the member function implementations. You must not have main and other free functions there. Class and struct definitions will be in a header file.

4) Main and other free functions will be in another cpp file; so in total you will submit three files.

Addition and Deletion of Data to the Data Structure

As mentioned above, the subsequences are stored in ascending ordered way in the data structure (first criterion is the size of the subsequence, second criterion is the values in it, prioritizing the numbers closer to head). When a new non-negative input number, which does not exist in the structure, is read, you will need to add one or more subsequences that this number will take part. Definitely, there will be a single node subsequence that contains only the input number. Moreover, depending on the existing subsequences, some new subsequences might be created anew using the existing values appended by the input number. Here, please pay attention that whenever you generate a new subsequence via such an appending, you have to create new `SubSeqNode`s by copying the existing values rather than sharing the same nodes in different subsequences.

Consider the following example. Initially our `SubSeqsList` object's `hHead` is `NULL`, specifying empty structure. This is shown in Figure 2.

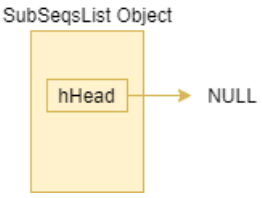

Figure 2. Empty data structure

Suppose the data entered is `1 2 4 3`. When the first number, `1`, is read, only one subsequence with a single element is created as shown in Figure 3.

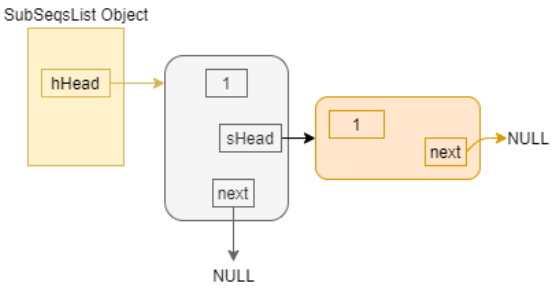

Figure 3. The data structure after reading `1`

The next number is `2`. This will cause to add a subsequence with value `2` only. Moreover, since there is an existing subsequence with value `1`, another subsequence will be added by appending `2` to it, yielding `1 2`; however, be careful that we create two new nodes for `1` and `2` in this new subsequence list. The data structure after processing `2` is shown in Figure 4.

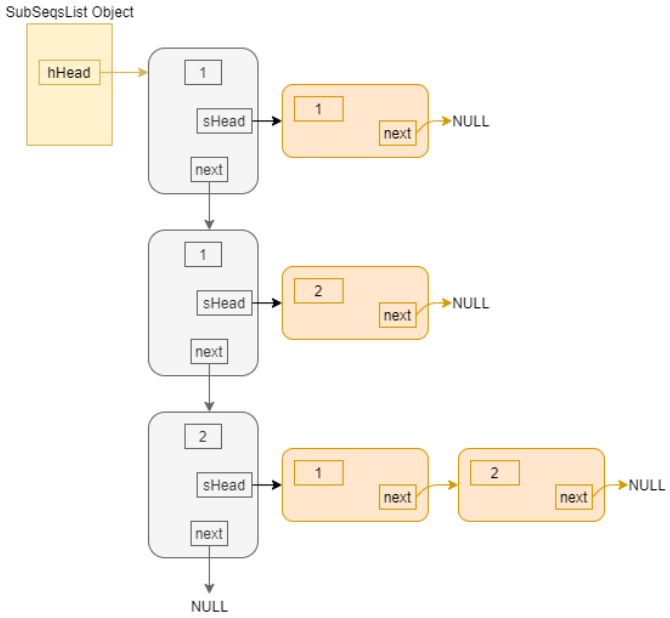

Figure 4. The data structure after reading `1` and `2`

The next number is `4`. This brings a new single element subsequence with `4`. Moreover, since there already exist subsequences `1`, `2` and `1 2`, and appending `4` to them would generate new ordered subsequences, three more subsequences are added to the data structure using newly created nodes: `1 4`, `2 4`, `1 2 4`. This is shown in Figure 5.

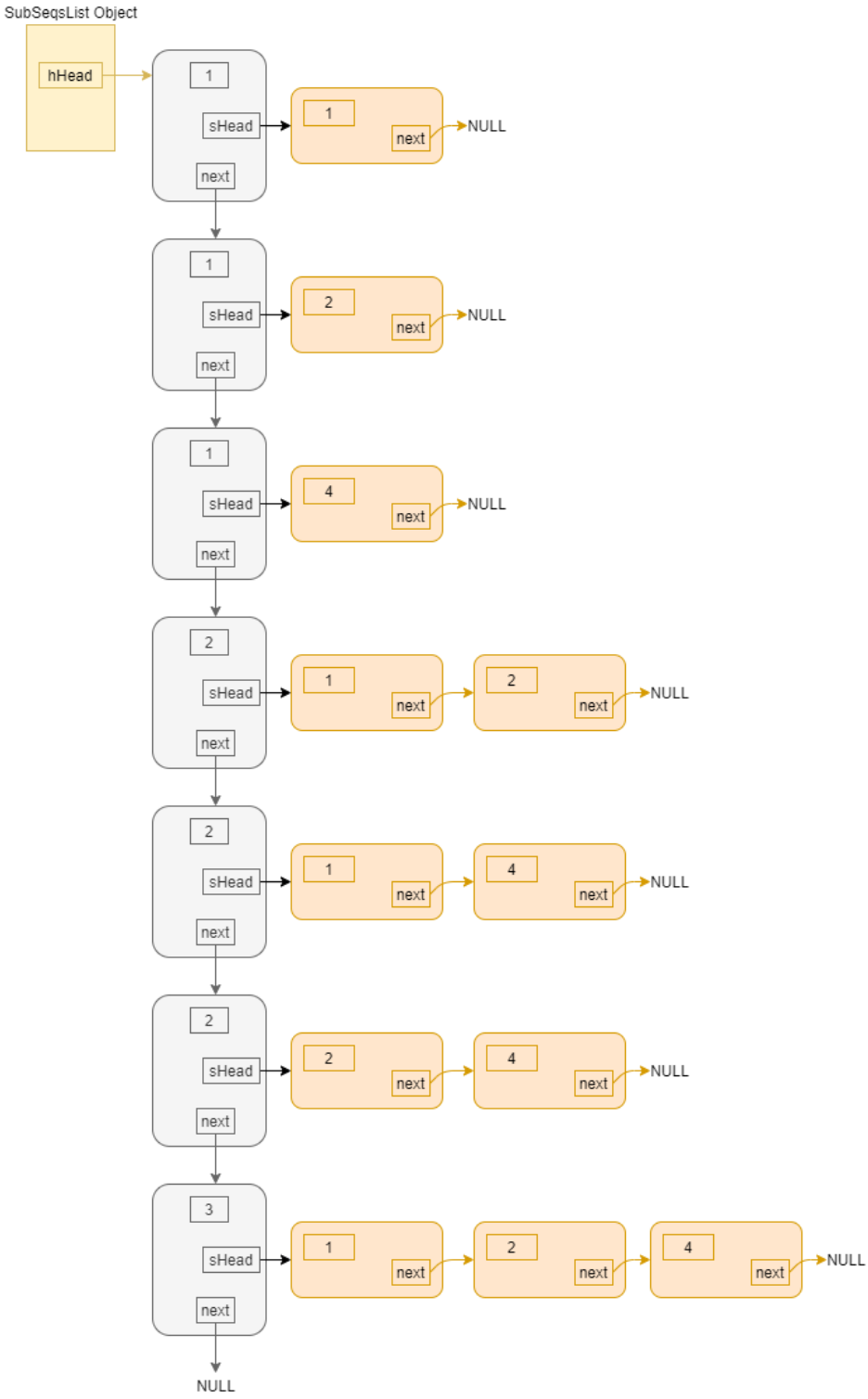

Figure 4. The data structure after reading `1`, `2` and `4`

The final input is `3`. This would cause a single element subsequence as well as some multi element subsequences. Appending `3` to the existing subsequences `1`, `2`, and `1 2` would be used to generate three new ordered subsequences: `1 3`, `2 3`, and `1 2 3`. This is depicted in Figure 5.

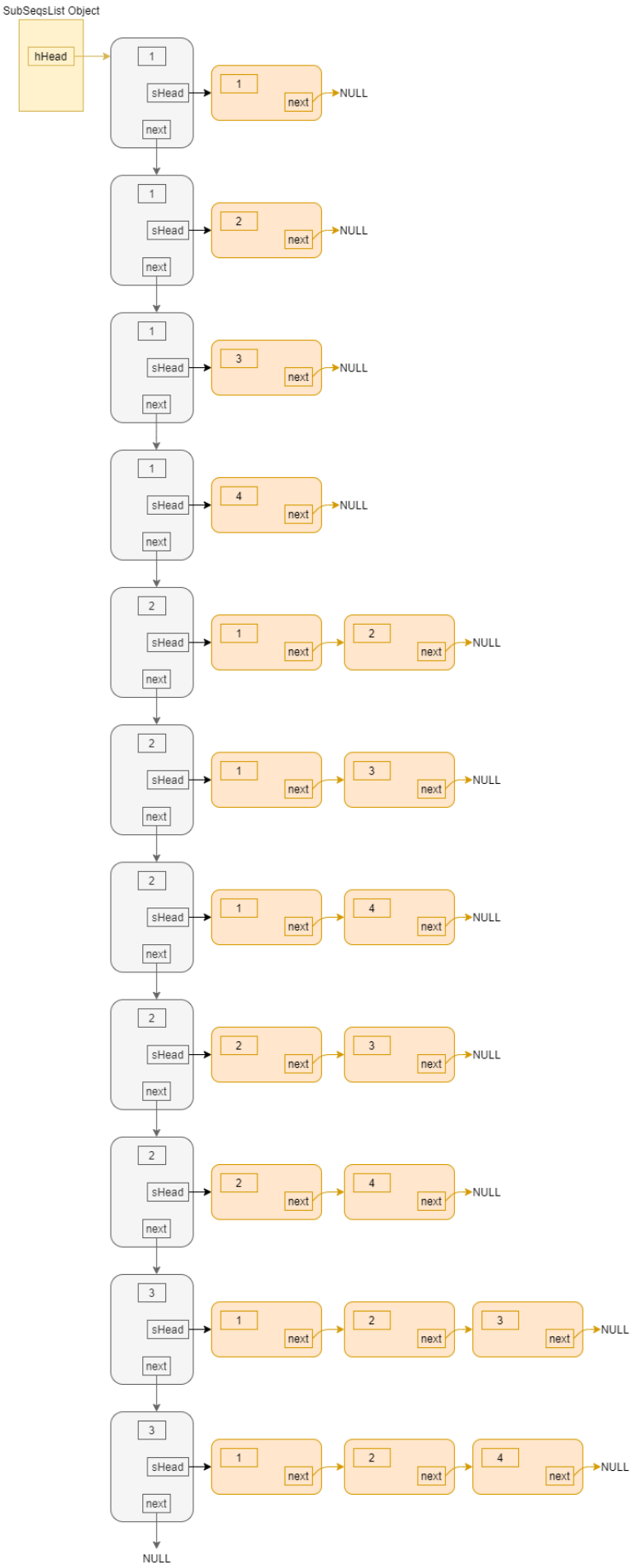

Figure 5. The data structure after reading `1`, `2`, `4` and `3`

Another operation is deletion of subsequences that contains a particular value. Here, please be advised that you have to delete the entire subsequence that contains the deleted value, not a single node. For example, if the input is `1 2 4 3 -2`, then the final data structure should contain the subsequences of Figure 5 with all lists that contain the value `2` removed. This is shown in Figure 6.

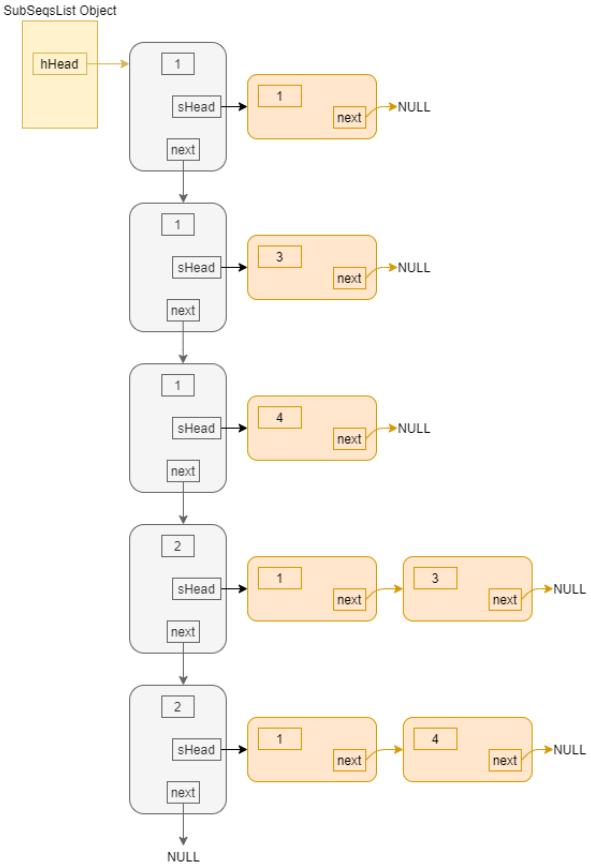

Figure 6. The data structure after removing `2`

The addition and deletion operations might cause to modify `hHead` of `SubSeqsList` object. Moreover, depending on the input values, all of the lists could be deleted yielding an empty structure (e.g. if the input is `2 1 3 -2 -1 4 -3 -4`). Be aware of these cases while developing your algorithm.

Moreover, you must deallocate all dynamic memory before your program terminates. You have to perform this by implementing and calling a member function of the class.

Some Important Rules

In order to get a full credit, your programs must be efficient and well presented, presence of any redundant computation or bad indentation, or missing, irrelevant comments are going to decrease your grades. You also have to use understandable identifier names, informative introduction and prompts. Modularity is also important; you have to use functions wherever needed and appropriate. Since using classes is mandated in this homework, a proper object-oriented design and implementation will also be considered in grading.

Since you will use dynamic memory allocation in this homework, it is very crucial to properly manage the allocated area and return the deleted parts to the heap whenever appropriate. Inefficient use of memory may reduce your grade.

When we grade your homework we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we may run your programs in *Release* mode and **we may test your programs with very large test cases**. Of course, your program should work in *Debug* mode as well.

Please do not use any non-ASCII characters (Turkish or other) in your code.

You are allowed to use sample codes shared with the class by the instructor and TAs. However, you cannot start with an existing .cpp or .h file directly and update it; you have start with an empty file. Only the necessary parts of the shared code files can be used and these parts must be clearly marked in your homework by putting comments like the following. Even if you take a piece of code and update it slightly, you have to put a similar marking (by adding "`and updated`" to the comments below.

```
/* Begin: code taken from ptrfunc.cpp */

…

/* End: code taken from ptrfunc.cpp */
```

Sample Runs

Sample runs are given below, but these are not comprehensive, therefore you have to consider **all possible cases** to get full mark.

Inputs are shown in **bold** and *italic*.

Sample run 1:

```
Please enter the numbers in a line: 5 12 18 99

Subsequence(s) containing 5 has/have been added
Subsequence(s) containing 12 has/have been added
Subsequence(s) containing 18 has/have been added
Subsequence(s) containing 99 has/have been added

FINAL CONTENT
1 | 5
1 | 12
1 | 18
1 | 99
2 | 5 --> 12
2 | 5 --> 18
2 | 5 --> 99
2 | 12 --> 18
2 | 12 --> 99
2 | 18 --> 99
3 | 5 --> 12 --> 18
```

```
3 | 5 --> 12 --> 99
3 | 5 --> 18 --> 99
3 | 12 --> 18 --> 99
4 | 5 --> 12 --> 18 --> 99
```

Sample run 2:

```
Please enter the numbers in a line: 79 37 19 11 7 5

Subsequence(s) containing 79 has/have been added
Subsequence(s) containing 37 has/have been added
Subsequence(s) containing 19 has/have been added
Subsequence(s) containing 11 has/have been added
Subsequence(s) containing 7 has/have been added
Subsequence(s) containing 5 has/have been added

FINAL CONTENT
1 | 5
1 | 7
1 | 11
1 | 19
1 | 37
1 | 79
```

Sample Run 3:

```
Please enter the numbers in a line:  5 -5 -5 -11 11 5 19 -11 1

Subsequence(s) containing 5 has/have been added
All subsequence(s) containing 5 has/have been deleted
There is no subsequence that contains 5 to be deleted
There is no subsequence that contains 11 to be deleted
Subsequence(s) containing 11 has/have been added
Subsequence(s) containing 5 has/have been added
Subsequence(s) containing 19 has/have been added
All subsequence(s) containing 11 has/have been deleted
Subsequence(s) containing 1 has/have been added

FINAL CONTENT
1 | 1
1 | 5
1 | 19
2 | 5 --> 19
```

Sample Run 4:

```
Please enter the numbers in a line: -5 5 11 19 17 -11 -5 7 -19 -7 -17

There is no subsequence that contains 5 to be deleted
Subsequence(s) containing 5 has/have been added
Subsequence(s) containing 11 has/have been added
Subsequence(s) containing 19 has/have been added
Subsequence(s) containing 17 has/have been added
All subsequence(s) containing 11 has/have been deleted
All subsequence(s) containing 5 has/have been deleted
Subsequence(s) containing 7 has/have been added
All subsequence(s) containing 19 has/have been deleted
All subsequence(s) containing 7 has/have been deleted
All subsequence(s) containing 17 has/have been deleted
```

```
FINAL CONTENT
List is empty!
```

Sample Run 5:

```
Please enter the numbers in a line: 11 18 19 -11 -18 -19 1 2 3 -3 -2 -1 0

Subsequence(s) containing 11 has/have been added
Subsequence(s) containing 18 has/have been added
Subsequence(s) containing 19 has/have been added
All subsequence(s) containing 11 has/have been deleted
All subsequence(s) containing 18 has/have been deleted
All subsequence(s) containing 19 has/have been deleted
Subsequence(s) containing 1 has/have been added
Subsequence(s) containing 2 has/have been added
Subsequence(s) containing 3 has/have been added
All subsequence(s) containing 3 has/have been deleted
All subsequence(s) containing 2 has/have been deleted
All subsequence(s) containing 1 has/have been deleted
Subsequence(s) containing 0 has/have been added

FINAL CONTENT
1 | 0
```

Sample Run 6 (empty input):

```
Please enter the numbers in a line:
```

Enter pressed without any numbers (with or without whitespaces)

```
FINAL CONTENT
List is empty!
```

Sample Run 7:

```
Please enter the numbers in a line: 21 12 9 -12 9 12 9 12

Subsequence(s) containing 21 has/have been added
Subsequence(s) containing 12 has/have been added
Subsequence(s) containing 9 has/have been added
All subsequence(s) containing 12 has/have been deleted
9 is already in the list!
Subsequence(s) containing 12 has/have been added
9 is already in the list!
12 is already in the list!

FINAL CONTENT
1 | 9
1 | 12
1 | 21
2 | 9 --> 12
```

What and where to submit (PLEASE READ, IMPORTANT)

You should prepare (or at least test) your program using MS Visual Studio 2012 C++. We will use the standard C++ compiler and libraries of the abovementioned platform while testing your homework. It'd be a good idea to write your name and last name in the program (as a comment line of course).

Submissions guidelines are below. Some parts of the grading process might be automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.

Name your cpp file that contains your main program using the following convention:

"SUCourseUserName_YourLastname_YourName_HWnumber.cpp"

Your SUCourse user name is your SUNet user name which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

Cago_Ozbugsizkodyazaroglu_Caglayan_hw3.cpp

In some homework assignments, you may need to have more than one .cpp or .h files to submit. In this case, use the same filename format but add informative phrases after the hw number (e.g. Cago_Ozbugsizkodyazaroglu_Caglayan_hw3_myclass.cpp or Cago_Ozbugsizkodyazaroglu_Caglayan_hw3_myclass.h). However, do not add any other character or phrase to the file names. Sometimes, you may want to use some user defined libraries (such as strutils of Tapestry); in such cases, you have to provide the necessary .cpp and .h files of them as well. If you use standard C++ libraries, you do not need to provide extra files for them.

These source files are the ones that you are going to submit as your homework. However, even if you have a single file to submit, you have to compress it using ZIP format. To do so, first create a folder that follows the abovementioned naming convention ("SUCourseUserName_YourLastname_YourName_HWnumber"). Then, copy your source file(s) there. And finally compress this folder using WINZIP or WINRAR programs (or another mechanism). Please use "zip" compression. "rar" or another compression mechanism is NOT allowed. Our homework processing system works only with zip files. Therefore, make sure that the resulting compressed file has a zip extension. Check that your compressed file opens up correctly and it contains all of the files that belong to the latest version of your homework.

You will receive zero if your compressed zip file does not expand or it does not contain the correct files. The naming convention of the zip file is the same. The name of the zip file should be as follows:

SUCourseUserName_YourLastname_YourName_HWnumber.zip

For example, zubzipler_Zipleroglu_Zubeyir_hw3.zip is a valid name, but

Hw3_hoz_HasanOz.zip, HasanOzHoz.zip

are **NOT** valid names.

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!

Albert Levi, Vedat Peran