

## CSC 172 - Lab 4

Due: 02/27/2022

### Title: DNA Strings with lists

A key element in many bioinformatics problems is the biological sequence. A biological sequence is just a list of characters chosen from some alphabet. Two of the common biological sequences are DNA (composed of the four characters A, C, G, and T) and RNA (composed of the four characters A, C, G, and U).

### Objective:

You will implement some basic functionality for manipulating DNA and RNA sequences.

### Implementation:

You have to implement sequences using *linked lists*, storing one *letter* of the sequence per linked list node. You may implement either a singly linked list or a doubly linked list. You may implement your linked list using the code from the lecture or write it from scratch.

In addition to the linked lists of sequences, you will maintain a sequence array which stores the DNA and RNA sequences. Commands that manipulate sequences will refer directly to entries in the sequence array. The sequence array will store the sequence type (RNA or DNA) and a pointer or other form of access to the linked list that stores the sequence itself.

The type field should use an enumerated type variable, and you should also have an enumeration value to recognize if a given position in the sequence array is unused. All indexing (both for the sequence array and for positions in a sequence) will begin with position *zero*.

The primary design consideration for this project will be the interaction between the list class and its client. If you reuse the code from lecture, it is acceptable to alter or add some list methods. Regardless of whether you reuse the book code or write your own, it is mandatory that the list class remain application independent. No bioinformatics-related code should be part of the list class. All such code should be in some class above the level of the list class.

### Input and Output:

The program will be invoked from the command-line as:

```
java DNAList <arraysize> <commandfile>
```

The name of the program is `DNAList`. Parameter `<arraysize>` is the size of the sequence array, and `<commandfile>` is the input file with the commands to be processed by the program. For the

input provided an example of the correct command for arraysize of 20 is:

```
java DNAList 20 Lab4Test.txt
```

The input for this project will consist of a text file with a series of commands (some with parameters separated by spaces) with at most one command in each line. A blank line may appear anywhere in the command file, and any number of spaces may separate parameters. You need not worry about checking for syntactic errors. That is, only the specified commands will appear in the file, and the specified parameters will always appear. However, you must check for logical errors. These include attempts to access out-of-bounds positions in the sequence array or in a sequence. The output will be written to *standard output*. The program should terminate after reading the EOF mark. The commands your code should implement are as follows:

```
insert pos type sequence
```

Insert sequence to position *pos* in the sequence array. Type will be either DNA or RNA. You must check that sequence contains only appropriate letters for its type, if not the insert operation is in error and no change should be made to the sequence array. If this is the case print "Error occurred while inserting". If there is already a sequence at *pos* and if the sequence is syntactically correct, then the new sequence replaces the old one at that position. It is acceptable that sequence be *null* (contain no characters) in which case a null sequence will be stored at *pos*. Note that a null sequence in a sequence array slot is different from an empty slot.

```
remove pos
```

Remove the sequence at position *pos* in the sequence array. Be sure to set the type field to indicate that this position is now empty. If there is no sequence at *pos*, print "No sequence to remove at specified position".

```
print
```

Print out all sequences in the sequence array. For each sequence indicate its position in the sequence array and its type (RNA or DNA). Don't print anything for slots in the sequence array that are empty. The sequence array should be printed in the following format:

```
[pos]\t[type]\t[array]
```

Use this as an example:

```
3    RNA    UGCUAC
5    DNA    CGAGTTGC
```

```
print pos
```

Print the sequence and type at position *pos* in the sequence array. If there is no sequence in that position, print "No sequence to print at specified position". The sequence should be printed in the following format:

```
[type]\t[array]
```

Use this as an example:

```
DNA    CGAGTTGC
```

```
clip pos start end
```

Replace the sequence at position *pos* with a clipped version of the sequence. The clipped version is the part of the sequence beginning at character *start* and ending with character *end* (inclusive of *start* and *end*). It is an error if *start* has a value less than zero, or if *start* or *end* have invalid values. A *clip* command with such an error should make no alteration to the sequence, but should print the messages: "No sequence to clip at specified position" if there is no sequence at this slot, "Invalid start index" for *start* index less than 0, "Start index is out of bounds" for *start* beyond end of sequence, and "End index is out of bounds" for *end* beyond end of sequence. It is important that these conditions are checked in the order given here. If the value for *end* is less than the value for *start* and no errors occur then the result is a sequence containing no characters.

```
copy pos1 pos2
```

Copy the sequence in position *pos1* to *pos2*. If there is no sequence at *pos1*, print "No sequence to copy at specified position" and do not modify the sequence at *pos2*.

```
transcribe pos1
```

Transcription converts a DNA sequence in *pos1* to an RNA sequence. It is an error to perform the transcribe operation on an RNA sequence. If this is the case print "Cannot transcribe RNA". To transcribe a DNA sequence, change its type field to RNA, complement all the letters in the sequence, convert any occurrences of T to U, and reverse the sequence. Letters A and U are complements of each other, letters A and U are complements of each other, and letters C and G are complements of each other. If the slot is empty, then print "No sequence to transcribe at specified position".

## Submission:

Submit source code from this lab at the appropriate location for Lab 4 on the Blackboard system at [learn.rochester.edu](http://learn.rochester.edu). You should hand in a single zip (compressed archive) **.zip** containing your source code (DNAList, linked list implementation, etc.) and README files, as described below. Do not forget to include your NetId in the zipped filename. Your main (executable) class must be named `DNAList.java`. The README file must include your contact information, your partner's name (if any), a brief explanation of the lab (a one paragraph synopsis. Include information identifying what class and lab number your files represent), and one sentence explaining the contents of any other files you hand in.

## Script Testing:

It is very important that you follow the directions below:

1. Ensure that you have your main executable file named 'DNAList.java'. You should also include all other necessary .java files in your submission, such as linked list implementation.
2. Ensure that your program prints output in the specified format.
3. Create your zip directory (also used for submission) by selecting the necessary files (java files + README) with ctrl+click on windows, cmd+click on mac and compressing those into your zip file. The zip should be named [YourNetID]\_Lab4.zip (replacing the filler with your actual Net ID). There should be no other files or subdirectories in the zip folder - only those required by the lab.

## Grading (10 pts)

See “Grading of coding assignments” posted under Course Materials on Blackboard. Grading will be based on passing test cases.

**Notes:**

All labs are open book. You can get code snippets from the internet if you want (make sure you cite those properly). But that is not the purpose. We want you to think about an algorithm and then implement it together with your partner.

**Credits:**

Programming assignment based on Clifford Shaffer’s (Virginia Tech) assignments in his course of Algorithms and Data Structures.