Sabancı University

Faculty of Engineering and Natural Sciences

CS204 Advanced Programming

Spring 2024

Homework 4 – Operator Overloading for DNA sequences

Due: 22/04/2024, Monday, 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 and homework trading will not be tolerated!

Introduction

In this homework, you are asked to implement a C++ class for DNA sequences with lots of operator overloading in it. The details of these operators are explained below. Our focus in this homework is on the class design, implementation and operator overloading technicalities. The usage of this class in the main program is given to you in the homework pack. You are going to use it without any change.

Class Design for the DNA Sequence

The name of the class for DNA Sequence is `DNAseq`. You will implement the `DNAseq` class as a 1D dynamic array that represents a sequence of nucleotides. The concept of nucleotide will be implemented as an enumerated type with four values, as given below.

```
enum Nucleotide {A, C, G, T};
```

You should add this enumerated type declaration line at the beginning of your `DNAseq.h` header file (after include lines, before the class definition). This **`Nucleotide`** type has been used in main.cpp, and you will need to use it in some operator function implementation as will be described later. If you do not know what enum type is, I have provided some notes under Week 7 notes (also briefly explained in class).

The `DNAseq` class has two private data members: The first one is an integer, named as `length`, which represents the number of nucleotides in the DNA sequence. The second one is the DNA sequence (named as `sequence`), which is going to be implemented as a dynamic `Nucleotide` array; thus, it is a `Nucleotide` pointer.

Basic Class Functions

Your `DNAseq` class implementation must include the following basic class functions:

- **Default constructor**: creates an empty DNA sequence.
- **Parametric constructor**: takes a string as its only parameter, and creates a DNA sequence of which the nucleotides correspond to the elements of the string parameter. Assume that the string parameter does not contain any invalid value, but it might be empty.
- **Deep copy constructor**: takes a `DNAseq` object as const-reference parameter and creates a new `DNAseq` object as a deep copy of the parameter.
- **Destructor**: by definition, destructor deletes all dynamically allocated memory and returns them back to the heap. You should implement this.

Operators to be Overloaded for the DNAseq class

In addition to the basic class functions that are explained above, you will overload several operators for the `DNAseq` class. These operators and the details of overloading are given below. You can overload operators as member or free functions. However, to test your competence in both mechanisms, we require you to have at least three functions implemented as free functions and at least three functions as member functions (the rest is up to you unless there are syntactic obligations). Details are given below.

Below are the details of the operators that you will implement:

`<=` This operator is used to check whether or not its left hand-side (*lhs*) operand is a subsequence of its right hand-side (*rhs*) operand. Thus, it is overloaded such that it takes `DNAseq` objects as both its *lhs* and *rhs* operands. It returns true if the *lhs* object is a subsequence of the *rhs* one. Otherwise, it returns false. In case of equality, true must be returned.

`*` This operator has two operands. It will be overloaded such that *lhs* operand is a `DNAseq` object and *rhs* operand is a positive integer. The operator returns a `DNAseq` object in which *lhs* is repeated by *rhs* times. The operator must not change the *lhs* object itself.

`%` This operator will return the number of occurrences of *lhs* `Nucleotide` in the *rhs* `DNAseq` object.

`!` This is a unary operator, which means it takes only one operand. It will be overloaded such that it returns the complement of its `DNAseq` operand. The complement of a DNA sequence is the one in which each nucleotide is replaced by its complement according to the following mapping: A←→T, G←→C . In other words, every A will be replaced by T, every T will be replaced by A, every G will be replaced by C, and every C will be replaced by G. The operator must not change the operand's value.

`=` This operator will be overloaded such that it will assign the `DNAseq` object on its *rhs* to the `DNAseq` object on its *lhs*. Please follow the rules of the writing assignment operator by checking the samples and the lecture notes (differences in sizes, deallocation of lhs first, self-assignment issues, cascaded assignments, return type, parameter type, etc.). Due to C++ language rules, this operator must be a member function (it cannot be free function).

Note that + operator will be overloaded in two different ways as (i) addition of a `DNAseq` object to another `DNAseq` object, and (ii) addition of `Nucleotide` value to a `DNAseq` object.

`+` This operator will be overloaded such that it takes `DNAseq` objects as both its rhs and lhs. It will behave in one of two ways based on whether the <u>first nucleotide</u> of the *rhs* object exists in the *lhs* or not.

(a) If it exists, the operator will return a `DNAseq` object, of which the content is the *lhs* object's sequence with *rhs* object sequence inserted just after the first occurrence of this first nucleotide.

(b) If, on the other hand, it doesn't exist, the operator will work as a concatenator and concatenate (i.e., append) the sequence of the *rhs* after the end of the sequence of the *lhs* in the object to be returned.

In either case, both operands will not be updated.

`+` This operator will be overloaded such that *lhs* operand is a `Nucleotide` and *rhs* operand is a `DNAseq` object. It returns a `DNAseq` object, in which *lhs* is prepended (added to the beginning) to *rhs*. The *rhs* object will not change.

`-` This operator is overloaded such that it has `DNAseq` objects as both its *rhs* and *lhs* operands. The operator returns a `DNAseq` object of which the content is the *lhs* object with *rhs* removed from it, if the DNA sequence of *rhs* exists in the *lhs* object. If not, the returned object will contain only the *lhs* sequence. Note here that, in case of removal, the removal will be only for the first occurrence, in case multiple occurrences exist. The operator should not change both operands' content.

`+=` operator is overloaded such that it uses `DNAseq` objects as both *lhs* and *rhs* operators. It modifies its current *lhs* `DNAseq` object by adding (+) the *rhs* `DNAseq` object to it. Please refer to the rules of addition according to the overloaded + operator above. This function must be implemented in a way to allow cascaded compound assignments.

<< This operator is overloaded such that it puts the DNA sequence of *rhs* `DNAseq` object on the *lhs* output stream. It takes an output stream (of type `ostream`) as its *lhs* parameter and a `DNAseq` object as its *rhs* parameter. We use this operator in main to display the content of a DNA sequence. Please refer to the lecture notes about the details and issues for the implementation.

In all of these operators, we assume that DNA sequences contain valid nucleotides, but they could initially be or result in empty ones.

In order to see the usage and the effects of these operators, please see **sample runs** and the provided **main.cpp**.

In this homework, you are allowed to implement some other helper member functions such as accessors (getters), if needed. However, **you are not allowed to use friend functions and friend classes. Moreover, you are not allowed to implement and use setter functions (mutators). Any violation for these, will result in point deductions.**

A life-saving advice: Any member function that does not change the private data members needs to be **`const` member function** so that it works fine with const-reference parameters. If you do not remember what "`const` member function" is, please refer to CS201 notes. I also explained this in class.

Another important advice: First, please learn the return type and parameter structures of the operators (whether they are reference or value, whether they are const or not) and the tricks of developing them by checking out the lecture notes. Then, start designing and coding the homework. Any wrong design decision or wrong approach taken would cause severe bugs that you cannot resolve easily. We use a unique approach in class design and operator overloading. Thus, please do not use Google/ChatGPT and similar online sources for learning since online resources may mislead you and may cause point deduction due to taking different ways; please always refer to lecture notes.

Provided main cpp file, and the requested class implementation

In this homework, **main.cpp** file is given to you within this homework package and we will test your codes with this main function with different inputs. You are not allowed to make any modifications in the main function (but, of course, you may temporarily change it in your local environment for unit testing of the operators). All class related functions, definitions and declarations (including free operator functions) must be implemented in class header and implementation files named as `DNAseq.h` and `DNAseq.cpp`. Actually, this is your task in this homework.

In **main.cpp**, inputs, outputs and statements are explained with appropriate prompts so that you do not get confused. Also you can assume that there won't be any wrong inputs in test cases; in other words, you do not need to do input checks in the functions that you are going to implement. However, **DNA sequences might be empty** initially or after being modified by the operators. We strongly recommend you to examine the main.cpp file and sample runs before starting your homework.

Some Important Programming Rules

Please do not use any non-ASCII characters (Turkish or other) in your code (not even as comments). And also do not use non-ASCII characters in your file and folder names. We really mean it; otherwise, you may encounter some errors.

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.

You are **not** allowed to use codes found somewhere online. These restrictions include code repositories, sites like geeksforgeeks, codes generated by GenAI tools, etc. Moreover, you are not allowed to use any statement, command, concept, topic that has not been covered in CS201 and CS204 (until now). Trying to find help online would generally cause such a problem. Thus, you always have to find help within the course material.

You cannot use break and continue statements. Global variables cannot be used either.

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 to 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 lab1.cpp */

…

/* End: code taken from lab1.cpp */
```

Since CodeRunner configuration in this homework does not allow to use extra non-standard C++ library/function/class files (other than the class files that you will implement), if you want to use such functions/classes covered in the classes (such as strutils), you will need to copy the necessary declarations and implementations to your files to be submitted by following the citation rules mentioned above.

Submission Rules (PLEASE READ)

It'd be a good idea to write your name and last name in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). For example, if your full name is "Satılmış Özbugsızkodyazaroğlu", then you must type it as follows:

// Satilmis Ozbugsizkodyazaroglu

We use CodeRunner in SUCourse+ for submission. No other way of submission is possible. Since the functionality part of the grading process will be automatic, you have to strictly follow these guidelines; otherwise we cannot grade your homework.

The advantage of CodeRunner is that you will be able to test your code against sample test cases. However, the output should be exact, but the textual differences between the correct output and yours can be highlighted (by pressing "show differences" button) on the interface.

You will submit two files: `DNAseq.cpp`, and `DNAseq.h` files. **`DNAseq.cpp` file's content will be copied and pasted into the "Answer" area** as in other assignments. **However, you will upload `DNAseq.h` file as attachment** to your submission in the relevant assignment submission page on SUCourse+.

The file name that you will upload must definitely be **`DNAseq.h`**; otherwise it does not work. Moreover, you will not upload the provided main.cpp file; we have already put it there.

Even any tiny change in the output format will result in your grade being zero (0) for that particular test case, so please test your programs yourself, and against the sample runs that are available at the relevant assignment submission page on SUCourse+ (CodeRunner).

In the CodeRunner, there are some visible and invisible (hidden) test cases. You can test your code via CodeRunner against the sample runs (by pressing the "Precheck" button). You can precheck as much as you can; there is no penalty for multiple prechecks. This Precheck is a bit different than "Check" that you have used in previous assignments. After pressing "Precheck" you will be able to see the performance of your code **only** for the visible test cases; you will see a green check mark (for success) or a red cross (for failure) at the beginning of each visible test case. There will **not** be any feedback for the hidden test cases and there will **not** be any cumulative feedback at the end. Moreover, there will **not** be "show differences" button during Precheck.

You will be able to get feedback about whether or not your code passed all test cases, including the hidden ones, only after you finish your attempt and complete the submission process (while previewing the submission). After the submission, "show differences" button will also be there for you to highlight the textual differences between your output and the expected one.

If you see a problem after you finish your attempt and complete the submission process, you will be able to re-attempt and make another submission. There is no penalty for re-attempts. However, you have to finish attempt and complete the submission process every time; there is no automatic submission on the deadline.

This process ensures that you will know whether or not your code has successfully passed all the test cases (visible and hidden) before finalizing your ultimate submission. However, we keep our rights to add more test cases in the grading process after the submission. **Thus, please make sure that you have read this documentation carefully and covered/tested all possible cases, even some other cases you may not have seen on CodeRunner or the sample runs.** Due to these reasons, **your final grade may conflict with what you have seen on CodeRunner**. We will also **manually** check your code against some criteria, comments, indentations and so on; hence, please do not object to your grade based on the **CodeRunner** results, but rather, consider every detail on this documentation and in general homework rules.

We will consider your last submission in grading with no exceptions. Thus, please make sure that the last submission is your final solution version. Also, we still do not suggest that you develop your solution on CodeRunner, but rather on your IDE on your computer.

Last, even if you cannot completely finish your homework, you can still submit.

Please see the syllabus for general homework grading issues.

Plagiarism

Plagiarism is checked by automated tools, and we are very capable of detecting such cases. Be careful with that. Exchange of abstract ideas are totally okay but once you start sharing the code with each other, it is very probable to get caught by plagiarism. So, do NOT share any part of your code to your friends by any means or you might be charged as well, although you have done your homework by yourself.

Homework is to be done personally and you have to submit your own work. **Cooperation will NOT be counted as an excuse.**

Our experience shows that code taken from online sources or generated by AI tools also show resemblance; thus if you try to get such help, you also may be charged with plagiarism with a person that you do not know.

In case of plagiarism, the rules written in the Syllabus apply.

Sample Runs

Sample runs are given below, but these are not comprehensive, therefore you must consider **all possible cases** to get a full mark. User inputs are shown in **bold**.

We configured CodeRunner to test these sample runs for you (as visible test cases). However, there also are some hidden test cases that would affect your grade. We will not disclose the hidden test cases before the grading has been completed.

We do **not** recommend you to copy and paste the prompts and messages from this document since some hidden control characters and non-standard characters might cause problems in CodeRunner.

Sample Run 1

```
Enter the first DNAseq sequence:
TA
Enter the second DNAseq sequence:
TTTT
dna1 is: TA
dna2 is: TTTT
dna1 is not a subsequence of dna2.
dna1's nucleotide counts:
A: *
C:
G:
T: *
dna2's nucleotide counts:
A:
C:
G:
T: ****
dna1: TA
dna2: TTTT
dna3 (dna1 + dna2): TTTTTA
dna4 = dna1 - dna2: TA
dna1's complement: AT
dna1 = dna1 * 3: TATATA
dna4 % A: 1
dna5 = A + (T + (G + dna2)): ATGTTTT
dna6 = dna6 + dna4 + dna2*2 - dna2 + (C + dna4) + (dna6 - dna6*4): TTTTTACTA
dna2: TTTT
dna4: TA
dna6 = T + dna1 + !dna1: TTAATATATTATA
After dna6 = dna1 = dna2 = dna4
dna1: TA
dna2: TA
dna4: TA
dna6: TA
```

```
dna6 = dna6: TA
After dna7 += dna4 += dna2
dna2: TA
dna4: TTAA
dna7: TTAA
(dna8 - dna4 + dna4) - dna8 + dna8: TTAA
```

Sample Run 2

```
Enter the first DNAseq sequence:
ACGT
Enter the second DNAseq sequence:
TGCA
dna1 is: ACGT
dna2 is: TGCA
dna1 is not a subsequence of dna2.
dna1's nucleotide counts:
A: *
C: *
G: *
T: *
dna2's nucleotide counts:
A: *
C: *
G: *
T: *
dna1: ACGT
dna2: TGCA
dna3 (dna1 + dna2): ACGTTGCA
dna4 = dna1 - dna2: ACGT
dna1's complement: TGCA
dna1 = dna1 * 3: ACGTACGTACGT
dna4 % A: 1
dna5 = A + (T + (G + dna2)): ATGTGCA
dna6 = dna6 + dna4 + dna2*2 - dna2 + (C + dna4) + (dna6 - dna6*4): ACCACGTGTTGCA
dna2: TGCA
dna4: ACGT
dna6 = T + dna1 + !dna1: TTGCATGCATGCAACGTACGTACGT
After dna6 = dna1 = dna2 = dna4
dna1: ACGT
dna2: ACGT
dna4: ACGT
dna6: ACGT
dna6 = dna6: ACGT
After dna7 += dna4 += dna2
dna2: ACGT
dna4: AACGTCGT
dna7: AACGTCGT
(dna8 - dna4 + dna4) - dna8 + dna8: AACGTCGT
```

Sample Run 3

```
Enter the first DNAseq sequence:
AAG
Enter the second DNAseq sequence:
AACAAGT
dna1 is: AAG
dna2 is: AACAAGT
dna1 is a subsequence of dna2.
dna1's nucleotide counts:
A: **
C:
G: *
T:
dna2's nucleotide counts:
A: ****
C: *
G: *
T: *
dna1: AAG
dna2: AACAAGT
dna3 (dna1 + dna2): AAACAAGTAG
dna4 = dna1 - dna2: AAG
dna1's complement: TTC
dna1 = dna1 * 3: AAGAAGAAG
dna4 % A: 2
dna5 = A + (T + (G + dna2)): ATGAACAAGT
dna6 = dna6 + dna4 + dna2*2 - dna2 + (C + dna4) + (dna6 - dna6*4): AAACCAAGAAGTAG
dna2: AACAAGT
dna4: AAG
dna6 = T + dna1 + !dna1: TTTCTTCTTCAAGAAGAAG
After dna6 = dna1 = dna2 = dna4
dna1: AAG
dna2: AAG
dna4: AAG
dna6: AAG
dna6 = dna6: AAG
After dna7 += dna4 += dna2
dna2: AAG
dna4: AAAGAG
dna7: AAAGAG
(dna8 - dna4 + dna4) - dna8 + dna8: AAAGAG
```

Sample Run 4

```
Enter the first DNAseq sequence:
TC
Enter the second DNAseq sequence:
TATC
dna1 is: TC
dna2 is: TATC
dna1 is a subsequence of dna2.
dna1's nucleotide counts:
A:
C: *
G:
```

```
T: *
dna2's nucleotide counts:
A: *
C: *
G:
T: **
dna1: TC
dna2: TATC
dna3 (dna1 + dna2): TTATCC
dna4 = dna1 - dna2: TC
dna1's complement: AG
dna1 = dna1 * 3: TCTCTC
dna4 % A: 0
dna5 = A + (T + (G + dna2)): ATGTATC
dna6 = dna6 + dna4 + dna2*2 - dna2 + (C + dna4) + (dna6 - dna6*4): TTATCCTCC
dna2: TATC
dna4: TC
dna6 = T + dna1 + !dna1: TTCTCTCAGAGAG
After dna6 = dna1 = dna2 = dna4
dna1: TC
dna2: TC
dna4: TC
dna6: TC
dna6 = dna6: TC
After dna7 += dna4 += dna2
dna2: TC
dna4: TTCC
dna7: TTCC
(dna8 - dna4 + dna4) - dna8 + dna8: TTCC
```

Sample Run 5

```
Enter the first DNAseq sequence:
TCAAG
Enter the second DNAseq sequence:
AAATTTGG
dna1 is: TCAAG
dna2 is: AAATTTGG
dna1 is not a subsequence of dna2.
dna1's nucleotide counts:
A: **
C: *
G: *
T: *
dna2's nucleotide counts:
A: ***
C:
G: **
T: ***
dna1: TCAAG
dna2: AAATTTGG
dna3 (dna1 + dna2): TCAAAATTTGGAG
dna4 = dna1 - dna2: TCAAG
dna1's complement: AGTTC
dna1 = dna1 * 3: TCAAGTCAAGTCAAG
dna4 % A: 2
dna5 = A + (T + (G + dna2)): ATGAAATTTGG
```

```
dna6 = dna6 + dna4 + dna2*2 - dna2 + (C + dna4) + (dna6 - dna6*4): TCCTCAAGAAAATTTGGAG
dna2: AAATTTGG
dna4: TCAAG
dna6 = T + dna1 + !dna1: TTCAAGTTCAGTTCAGTTCAGTCAAGTCAAG
After dna6 = dna1 = dna2 = dna4
dna1: TCAAG
dna2: TCAAG
dna4: TCAAG
dna6: TCAAG
dna6 = dna6: TCAAG
After dna7 += dna4 += dna2
dna2: TCAAG
dna4: TTCAAGCAAG
dna7: TTCAAGCAAG
(dna8 - dna4 + dna4) - dna8 + dna8: TTCAAGCAAG
```

Sample Run 6

```
Enter the first DNAseq sequence:
TGTGA
Enter the second DNAseq sequence:
TGTGA
dna1 is: TGTGA
dna2 is: TGTGA
dna1 is a subsequence of dna2.
dna1's nucleotide counts:
A: *
C:
G: **
T: **
dna2's nucleotide counts:
A: *
C:
G: **
T: **
dna1: TGTGA
dna2: TGTGA
dna3 (dna1 + dna2): TTGTGAGTGA
dna4 = dna1 - dna2:
dna1's complement: ACACT
dna1 = dna1 * 3: TGTGATGTGATGTGA
dna4 % A: 0
dna5 = A + (T + (G + dna2)): ATGTGTGA
dna6 = dna6 + dna4 + dna2*2 - dna2 + (C + dna4) + (dna6 - dna6*4): TGTGAC
dna2: TGTGA
dna4:
dna6 = T + dna1 + !dna1: TTGTGAACACTACACTACACTTGTGATGTGA
After dna6 = dna1 = dna2 = dna4
dna1:
dna2:
dna4:
dna6:
dna6 = dna6:
After dna7 += dna4 += dna2
dna2:
dna4:
dna7:
(dna8 - dna4 + dna4) - dna8 + dna8:
```

Sample Run 7

```
Enter the first DNAseq sequence:
A
Enter the second DNAseq sequence:
CGCGTTA
dna1 is: A
dna2 is: CGCGTTA
dna1 is a subsequence of dna2.
dna1's nucleotide counts:
A: *
C:
G:
T:
dna2's nucleotide counts:
A: *
C: **
G: **
T: **
dna1: A
dna2: CGCGTTA
dna3 (dna1 + dna2): ACGCGTTA
dna4 = dna1 - dna2: A
dna1's complement: T
dna1 = dna1 * 3: AAA
dna4 % A: 1
dna5 = A + (T + (G + dna2)): ATGCGCGTTA
dna6 = dna6 + dna4 + dna2*2 - dna2 + (C + dna4) + (dna6 - dna6*4): ACCAGCGTTA
dna2: CGCGTTA
dna4: A
dna6 = T + dna1 + !dna1: TTTTAAA
After dna6 = dna1 = dna2 = dna4
dna1: A
dna2: A
dna4: A
dna6: A
dna6 = dna6: A
After dna7 += dna4 += dna2
dna2: A
dna4: AA
dna7: AA
(dna8 - dna4 + dna4) - dna8 + dna8: AA
```

Sample Run 8

```
Enter the first DNAseq sequence:
GGG
Enter the second DNAseq sequence:
GGGGGGGGGG
dna1 is: GGG
dna2 is: GGGGGGGGGG
dna1 is a subsequence of dna2.
dna1's nucleotide counts:
A:
C:
G: ***
T:
dna2's nucleotide counts:
A:
```

```
C:
G: **********
T:
dna1: GGG
dna2: GGGGGGGGGG
dna3 (dna1 + dna2): GGGGGGGGGGGGG
dna4 = dna1 - dna2: GGG
dna1's complement: CCC
dna1 = dna1 * 3: GGGGGGGGG
dna4 % A: 0
dna5 = A + (T + (G + dna2)): ATGGGGGGGGGGG
dna6 = dna6 + dna4 + dna2*2 - dna2 + (C + dna4) + (dna6 - dna6*4): GGGGGGGGGGGGGCGGG
dna2: GGGGGGGGGG
dna4: GGG
dna6 = T + dna1 + !dna1: TGGGGGGGGGCCCCCCCCC
After dna6 = dna1 = dna2 = dna4
dna1: GGG
dna2: GGG
dna4: GGG
dna6: GGG
dna6 = dna6: GGG
After dna7 += dna4 += dna2
dna2: GGG
dna4: GGGGGG
dna7: GGGGGG
(dna8 - dna4 + dna4) - dna8 + dna8: GGGGGG
```