Middle East Technical University | Department of Computer Engineering

CENG 707

Data Structures and Algorithms

Fall '2014-2015

Programming Assignment I

Due date: 07 December 2014, Sunday, 23:55

1 Introduction

In this assignment, your task is to design and implement a *Chromosome* class which shall be capable of simulating gene aberrations (disruptions of chromosome content) observed in chromosomes. Chromosomes are, as you may recall from high school, macromolecules consisting of *nucleotides* which are the building blocks of our genetic code. There are four types of nucleotides that we may consider as the alphabet of the gene language. Those four nucleotides are Adenine, Thymine, Cythosine and Guanine. An illustration of the chromosome's physical structure is given in the Figure 1.

This structure is called a four-arm structure (Small Left: SL, Small Right: SR, Big Left: BL, Big Right: BR) where arms come together at the *centromere*. Inside each arm, sequences of nucleotides constructs a double helix structure where each nucleotide has a counterpart on the other side of the helix. We, on the other hand, omit the double helix structure and view each arm as a single sequence of nucleotides.

Chromosomal aberrations are disruptions in the normal chromosomal content of a cell and are a major cause of genetic conditions in humans. Most common chromosome mutations are of two forms: single arm mutations and double arm mutations. Three major single arm mutations can be explained briefly as :

1. **Deletion**: A piece of chromosome is torn apart.
2. **Duplication**: A piece of chromosome is duplicated and inserted right after itself.
3. **Inversion**: A piece of chromosome is torn apart and reconnected in reverse order.

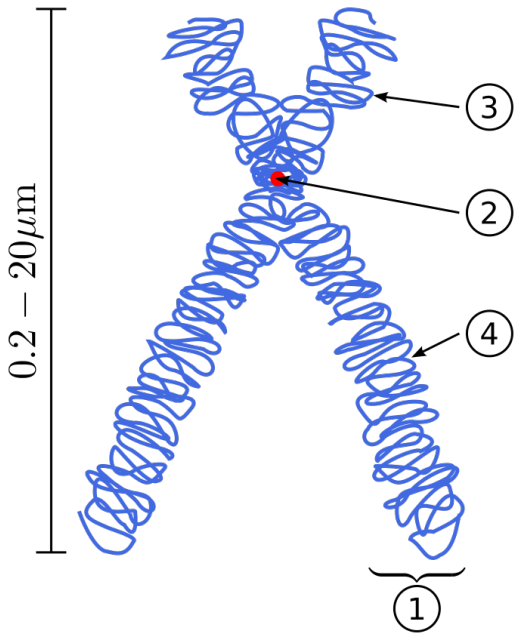

Figure 1: (1) Chromatid (2) Centromere (3) Right small arm (4) Right big arm

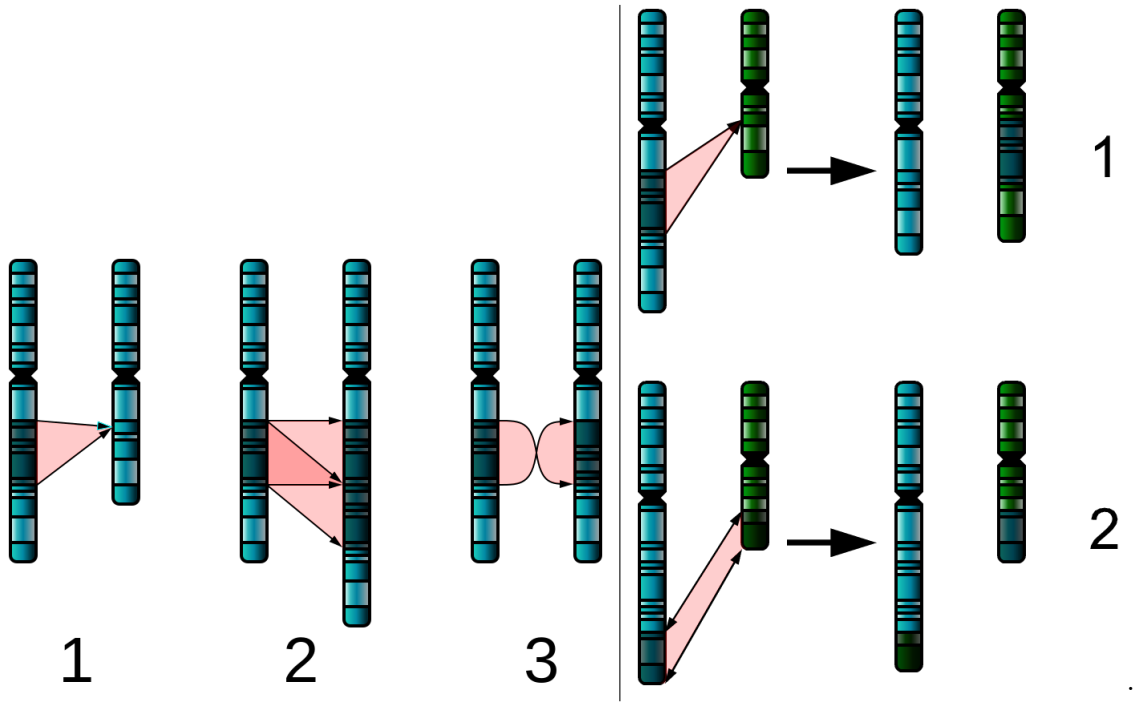

Figure 2: Left:(1) Deletion (2) Duplication (3) Inversion, Right:(1) Insertion (2) Translocation

Major double arm mutations are

1. **Insertion**: A piece of chromosome is torn apart from one arm and inserted into a place of the corresponding arm (SR and SL correspond to each other, BR and BL correspond to each other).

2. **Translocation**: Two pieces from corresponding arms are torn apart and inserted into one another's place.

These mutations are depicted in Figure 2.

Some nucleotide sequences are known to tend to mutate more often. Hence, the more a chromosome contains those sequences, the more it is susceptible to mutation. It is important to count these occurrences and detect mutation-prone chromosomes.

2 Specifications

You are expected to implement the class Chromosome by providing the above explained methods and overloading operator `<<`. These methods are declared in the "chromosome.h" file. An example "test.cpp" file is also provided so that you can test your implementations using it. Inside the file "chromosome.h", you will see two enum types:

```
typedef enum Nucleic_Acid {Adenine, Cytosine, Guanine, Thymine, Nullin};
```

The first four is explanatory. The "Nullin" value is used as a termination value of Nucleic_Acid arrays.

```
typedef enum Chromatid_Pos {SL ,SR ,BL, BR};
```

They are used to refer to chromosome arms Small Left, Small Right, Big Left, and Big Right respectively (See method explanations and "test.cpp" outputs).

Explanation of the class methods and overloaded operator `<<` are as follows. Note that indexing inside each arm starts with zero. Intervals defined by `start` and `finish` pairs include the nucleic acids indexed with `start` and `finish`:

```
Chromosome(Nucleic_Acid Seq_L[], Nucleic_Acid Seq_R[], int centromere);
```

Constructor for *Chromosome* class. First argument is an array ending with value *Nullin* and it contains the content of left arms (SL and BL) concatenated. Second argument does the same for right arms (SR and BR). Third argument determines where two Nucleic_Acid sequences merge. In other words, lengths of the SL and SR arms should be equal to the value of *centromere*. Note that you should apply deep copying for construction. Hence, when content of say `Seq_L` changes, content of the object constructed should not change.

```
friend ostream& operator<< (ostream &out, Chromosome &C);
```

Operator `<<` overloading. Form of the output should be as follows:

```
<Content of SL arm>  <Content of BL arm>
                    \/
                    /\
<Content of SR arm>  <Content of BR arm>
```

See example output for more detail.

```
void Deletion(Chromatid_Pos pos, int start, int finish);
```

This method should delete the Nucleic Acids of chromosome arm specified by argument `pos` with indices starting from argument `start` and ending with argument `finish` (both inclusive). See example output for clarification. Note that first Nucleic Acid is indexed with 0.

```
void Duplication(Chromatid_Pos pos, int start, int finish);
```

A copy of subsequence `[start,finish]` of chromosome arm `pos` should be inserted after the Nucleic Acid with index `finish`. See example output for clarification.

```
void Inversion(Chromatid_Pos pos, int start, int finish);
```

Content of the subsequence `[start,finish]` of chromosome arm `pos` should be reverted. See example output for clarification.

```
void Insertion(Chromatid_Pos source_pos, int start, int finish, int insertion_point);
```

Content of the subsequence `[start,finish]` of chromosome arm `source_pos` should be pulled apart and inserted right after the Nucleic Acid `insertion_point` of the corresponding arm. See example output for clarification.

```
void Translocation(Chromatid_Pos pos1, int start1, int finish1, int start2, int finish2);
```

Content of the subsequence `[start1,finish1]` of chromosome arm `pos1` should be swapped with content of the subsequence `[start2,finish2]` of the corresponding arm. Note that method call `Translocation(SL,s1,f1,s2,f2)` is equivalent to the method call `Translocation(SR,s2,f2,s1,f1)`. See example output for clarification.

```
int Sequence_Occurences(Nucleic_Acid Seq[]);
```

This method will return the number of occurrences of Nucleic Acid Sequence defined by argument `Seq`. As in the constructor, `Seq` ends with `Nullin`. Your implementations of this method will not be tested with self repetitive sequences such as `TATAT`, nor the length of the sequences will exceed the length of the smallest arm.

3 Other Instructions

1. You must use **linked list** inside the *Chromosome* class to store *Nucleic_Acid* sequences. Your method implementations, too, must perform operations on those linked lists. Submissions not following this instruction will get **zero** grade.
2. You may assume that special cases will not be tested, while grading your implementations. Arguments such as `start` and `finish` will define a proper subsequence:

   $$0 \leq start \leq finish < (length\ of\ the\ arm).$$

   Similarly constructor's `centromere` argument will be in proper range.
3. You are not allowed to use any container libraries (vector, list, etc.)
4. You are not allowed to use the algorithm library.
5. As you notice, you can gain partial grades as long as your code compiles and you implement the constructor and overload operator `<<`, correctly. If you fail to implement a particular method, try to fill it with a dummy code to be certain that your code compiles.
6. With that being said, do not fear about possible runtime errors. Runtime errors will be caught and handled during grading so that you may get partial credits from methods you correctly implemented.

4 Regulations

1. **Late Submission:** General submission rules for the course is applied.
2. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.
3. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.
4. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. Your program will be evaluated on inek machines.

5 Submission

Submission will be done via COW. Create a tar.gz file named `hw1.tar.gz` that contains all your source code files one of them being "chromosome.cpp". To test your codes, following terminal commands will be executed:

```
$ tar -xf hw1.tar.gz
$ g++ -o main main.cpp
$ ./main
```

There should **not** be a main function inside "chromosome.cpp". You must write your own test driver file or modify and use provided driver file "test.cpp". You may obtain sample "output.txt" by following commands:

```
$ g++ -o test test.cpp
$ ./test > output.txt
```

Please do not submit any driver files.
Note: The submitted archive should not contain any directories!