Middle East Technical University | Department of Computer Engineering

CENG 242

Algorithms

Spring 2019-2020

HOMEWORK 4

Kill the Covid-19!

Due date: 16 May 2020, 23:55

1 Objectives

The objective of this assignment is to make yo familiar with the object oriented programming concepts including inheritance, polymorphism and abstract base class.

Keywords: *inheritance, polymorphism, abstraction, dynamic type casting*

2 Problem

In this assignment, you are going to implement a scenario which includes a series of biological events occuring when some types of foreign microorganisms enter into human cells. `MicroOrganism` and `Cell` will be the main classes that play the leading role. Other than those, there will be other classes which play helper role, yet very important in terms of expressing constituents of the main classes. In this homework, the things to greet you are as follows:

- You are going to make each microorganism found a compatible human cell to live in. Here, the compatibility will be measured in terms of their shape and size. Shapes of microorganisms will be one of those 3 basic 2D polygons: triangle, circle or square. Cells can be any type of 2D polygons. Microorganisms and cells which have the same type of shape and size (edge length or radius) will be matched. Inputs will be provided with unique matchings.

Figure 1: He says: "At each region there will be a geometrician. We will distribute free triangle to every one!"

- Microorganisms will be implemented with inheritance. Each microorganism type will be classified according to their shapes and each shape will be stated as a different class: `Circular`, `Triangular` and `Squadratic` derived from abstract `MicroOrganism` base class. Each microorganism type will show a different reaction in the human cell and they will necessitate different type of implementations because of their shape by nature.

- Since the cells initially can not identify the microorganisms, they will refer all just as `MicroOrganism`. This will cost dearly to them since some of those are really harmful viruses such as Covid-19 and kill the cell.

- For you, matching a cell with a microorganism will not be so easy because cells are not classified according to the shape of their boundary. Instead, they will have a cell wall defining a shape for their boundary and this cell wall will consist of many partial staright and circular wall segments. In order to arise shape of the cell wall, you need to combine those partial wall segments.

- In the homework, in order to express the wall segments used in the cell wall, `Wall` class will be used. This is nothing but just the representation of a straight line segment between two points (Points are implemented with `Particle` class). However, for circular segments, you are going to use `CurvyWall` class which is a derivation of the base class `Wall`. CurvyWall objects will be defined by three points instead of two, where the third point is center of the curve segment.

- Lastly, `Tissue` class is the place where you hold the cells. In fact, this class was initially designed for you to implement a plasma treatment to clean the microbic tissue by replacing it with a healthy one in some kind of R-Tree structure. Later, it was decided to remove that part since it makes the homework really tiring. Although the related method declarations were left in the headers for those who wants to struggle with "wheels within a wheel", you are not expected to that part.

In order to go into the details, please read the sections below and examine the class headers.

3 Specifications & Implementation

In the subsections below, each class that you need to implement is given. However, instead of explaining each method of a class one-by-one, only the general structure of the class, its purpose of usage and some illustrative figures are presented here. The methods you are going to implement at each class is given as much detailed as it can be in the individual header files. The header files were removed from here since they make the pdf too long.

Note-1: In the header files you are allowed to change only the `private` and `protected` parts. On the other hand, the `public` parts are already given to you. While implementing, be careful that you may need to add some other methods also, besides your own variables, in `private/protected` parts.

Note-2: Be careful that even the class constructors states nothing specific, in the private parts, initializing the variables defined-by-you with the appropriate values is your responsibility.

3.1 Particle.cpp

This is the most basic class of the assignment. This type of object represents a 2D point on the xy-coordinate sytem.

Note: Use `EPSILON = 0.01` given in the macros to compare the float values.

3.2 Wall.cpp & CurvyWall.cpp

`Wall` class and its derived class `CurvyWall` will be used to give a shape to cells and microorganisms. `Wall` is a base class which represents straight line segments. `CurvyWall` is a derived class of `Wall` which represents circular curve segments. Both `Wall` and `CurvyWall` objects can be constructed in two ways:

- `Wall(float)` is to construct the wall by specifying just a length without defining an exact location for it. This option is presented to you in case that you want to use those kinds of walls to define the shapes for microorganisms (Since Microorganisms do not have an exact location until they penetrate into a cell). Correspondingly, `CurvyWall(float, float)` is to construct a curvy wall by specifying just a length (for the arc segment it represents) and the radius without defining an exact location for it. The right side of Figure 2 shows the examples.
- `Wall(const Particle&, const Particle&)` is to construct the wall whose initial and final points are the particles given in the first and second parameters, respectively. Correspondingly, `CurvyWall(const Particle&, const Particle&, const Particle&)` is to construct curves whose initial, final and center points are the particles given in the first, second and third parameters, respectively. This type of walls are used to define the shape of cells (Cells have a definite location). The leftt side of Figure 2 shows the examples.

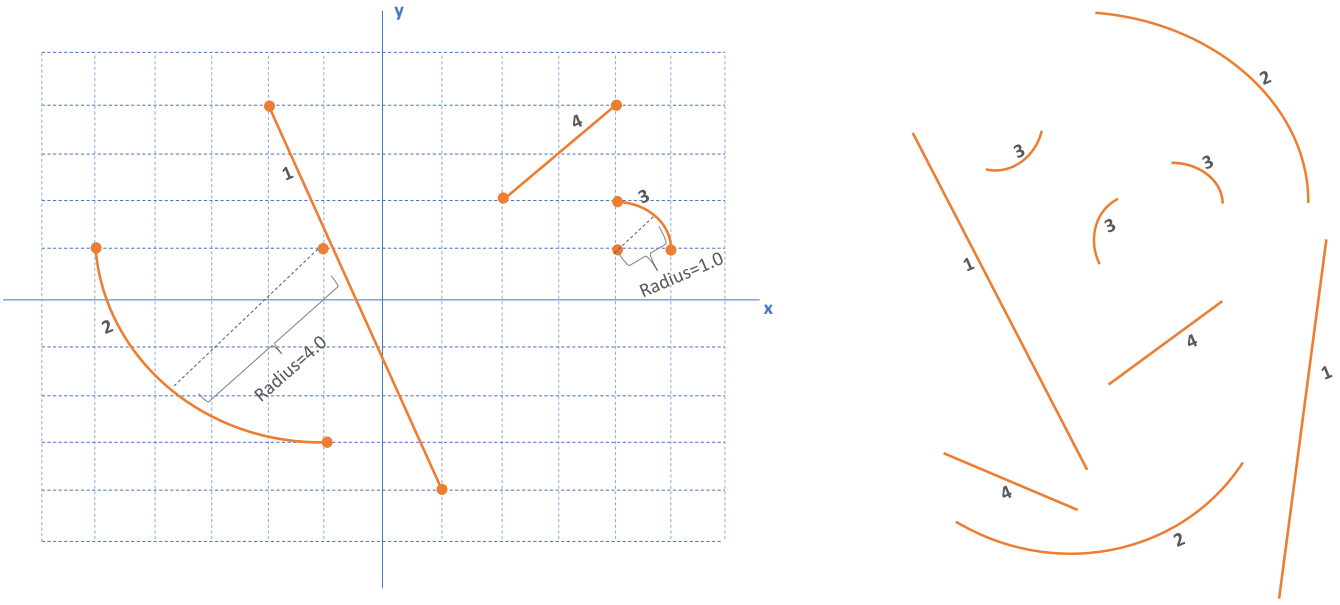


Figure 2: The left side of the Figure shows the walls and curvywalls constructed with a definite location (the second type of construction). This type of walls are used to give shape to Cells. Orange points on them are their initial, final and center Particles. On the other hand, the right side shows the walls and curvywalls constructed with just a length and radius (the first type of construction). This type of walls are used to give shape to Microorganisms until they get position in a certain location (inside a cell). The numbers next to them shows their ids. The ones with the same id on both left and right sides are actually the identical walls. They just differ in terms of their construction.

3.3 Cell.cpp

This is the class to define a cell. In this assignment, cells are the 2D geometric polygons. They can be given in any type of 2D closed pattern in the inputs. Their shapes will be defined as a sequence of straight lines (data type is `Wall`, given in Section 3.2) and curves (data type is `CurvyWall`, given in Section 3.2). Actually, this set of line/curve sequence represents nothing but the cell wall. The walls (lines/curves) constructing the cell wall will be given in counterclockwise order. Even for the most basic type of cells such as triangles, squares and circles, the cell wall can be defined with many partial wall segments instead of just giving 3 or 4 edges, or one curvy wall, respectively. The left side of Figure 3 shows cell examples.

Without regarding the shape of its cell wall, all cells is an object of `Cell` class. Each cell is located on a certain piece of some tissue. Location of a cell is naturally defined by the cell walls which were constructed between certain 2D points (data type is `Particle`, given in Section 3.1). A cell can accept only the microorganisms which have the exact shape of the cell itself. The details about the microorganisms are given in Section 3.4.

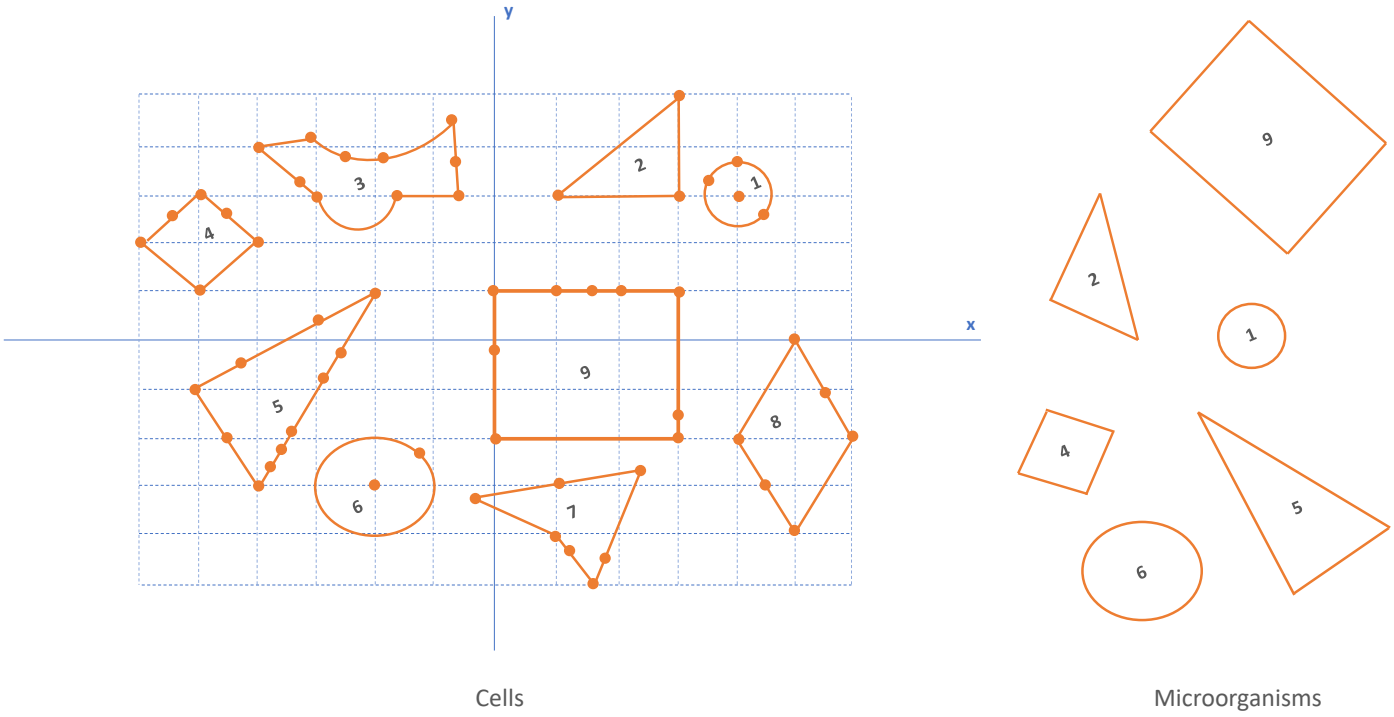
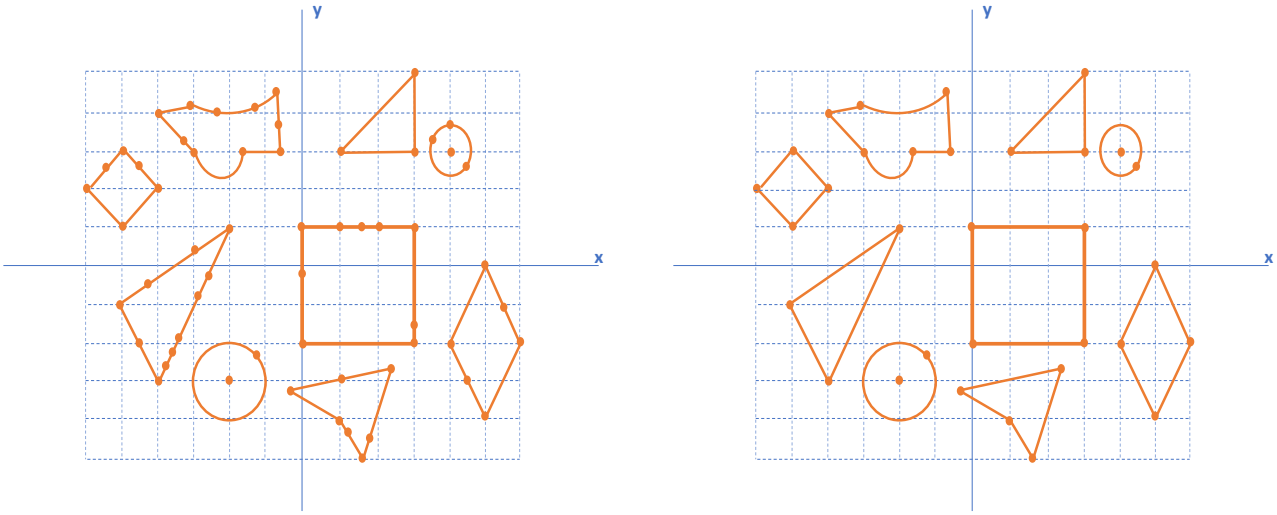


Figure 3: The left side of the Figure shows sample cells whereas the right side shows sample microorganisms. Cells can be formed up with many Wall and CurvyWall segments. The numbers inside them are their ids and the corresponding objects with the same ids on the right side are the microorganisms identical to those cells. Note cells can be any type of 2D polygons whereas microorganisms can be either triangle (Triangular), or circle (Circular), or square (Squadratic).

In the `Cell` class, there is a method named as `StrengthenCellWall()` which is supposed to combine the partial wall segments connectable to each other (The details of which types of walls can be connected with which types are given in `+operator()` of `Wall` class in the header file). You can consider `StrengthenCellWall())` method as a "simplifier" for the cell shape. It reduces the partial straight line and curve segments on the cell wall and extracts out the plain form of the cell. The effect of `StrengthenCellWall()` is shown in Figure 4.



Initial cells

Cells after StrengthenCellWall() method called for each.

Figure 4: On the left of the figure, the initial forms of the cells are given. As it is seen, they consist of many partial Wall and CurvyWall segments. On the other hand, the bare forms of the cells obtained after applying `StrenthenCellWall()` on each is given on the right. Note that, after strengthening, circle cells include only one curvywall whose initial and final particles are the same. As a footnote: After "strengthen" operation, it is not important that from which wall the cell-wall-chain starts. The only important thing is that the walls are ordered in counterclockwise direction. Moreover, the fact that from which particle a curvywall starts and ends at on a circle is not important.

3.4 MicroOrganism.cpp, Circular.cpp, Squadratic.cpp and Triangular.cpp

As you know, there are different types of microorganisms and they enter into our body in various ways. In this homework, you will deal with 3 types of microorganisms where each one has a different 2D geometric shape: the **"Circular**, **"Squadratic"** and **"Triangular"** ones. All these three types of microorganisms intrinsically want to enter into a cell belonging to an other live being and each type results in a different situation for the cell. The right side of Figure 3 shows microorganism examples. Here are the properties of each microorganism type:

1. **Circular microorganisms** are represented as a 2D circle. They can go into only the cells which have a circle cell wall of the same size. They are well-behaved organisms such that they do not give harm to the cell that they are in. On the contrary, they feed the cell and make its size double.

2. **Squadratic microorganisms** are represented as a 2D square. They can go into only the cells which have a square cell wall whose edge length is equal to the edge length of the microorganism. This type of microorganisms may get into meiosis cell division inside the cell. In that case, there occurs 4 little squadratic type of microorganisms while the parent organism dies, illustrated in Figure 5.

3. **Triangular microorganisms** are represented as a 2D triangle. They can go into only the cells which have a triangle cell wall whose edges have a one-to-one correspondence with the ones of the

microorganism in terms of their lengths. During the construction, edges of a Triangular may not be given in a certain order (clockwise or counterclockwise).
Actually, Triangular microorganisms are Covid-19 viruses and they kill the cell that they are in. Moreover, this Covid-19 virus has a RNA which can mutate. In the homework, the RNA structure of Covid19 is represented by `NucleoBase` class which is given in Section 3.5. Mutation of a RNA occurs via an other Triangular microorganism's getting involved such that two microorganisms interchange their corresponding Adenin-Urasil and Guanin-Sitozin nucleobases in non-matching parts of their RNAs. The details are in Section3.5.

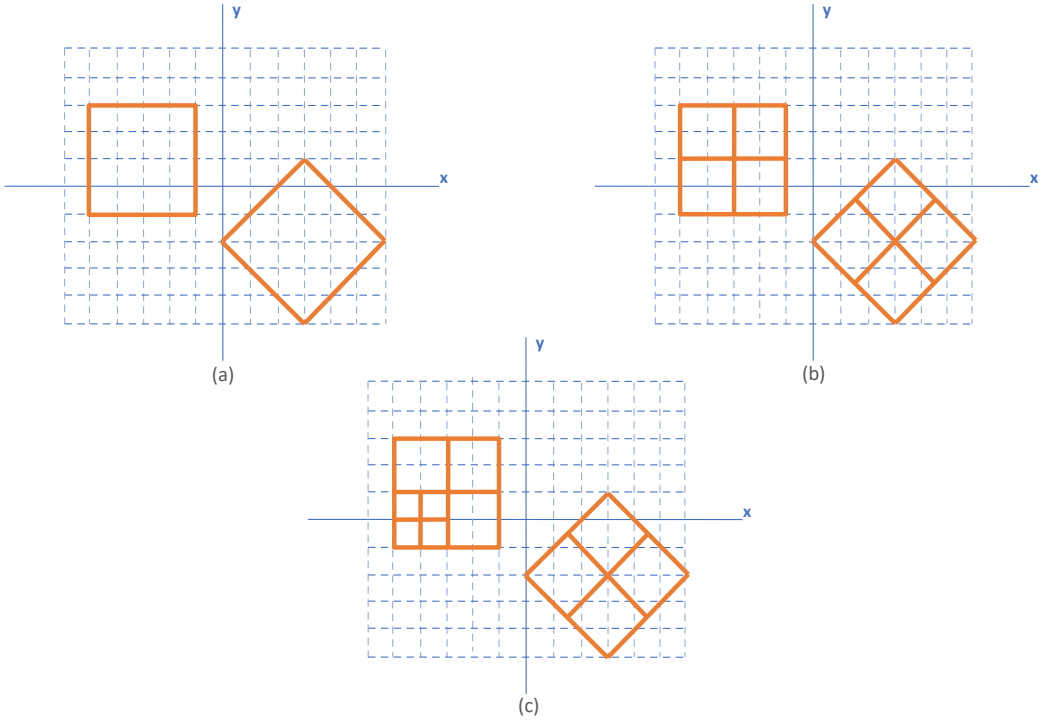


Figure 5: In the Figure, meiosis division is demonstrated on two squadratic microorganisms. The subfigure (a) shows the initial forms of squadratics. Later, these two squadratics are divided into 4 child squadratics through meiosis division as shown in subfigure (b). Please note the positions of the children. Later, the lower left child of the left squadratic is divided more and 4 more little baby squadatics appears, which is shown in subfigure (c).

Since the brain can not identify the type of a microorganism when it enters into the human body, it just collects all the distinct types under a common general title which is simply `MicroOrganism`. That is, you are going to implement a base class named `MicroOrganism` and 3 derived classes from it; `Circular`, `Squadratic` and `Triangular`.

3.5 NucleoBase.cpp

This class implements a linked-list structure and used only for RNA operations inside `Triangular.cpp`. Each of A(denin), U(rasil), G(uanin) and S(itozin) nucleobases in a RNA sequence corresponds to a `NucleoBase` object in the linked list structure. Each NucleoBase object is required to hold a char with a value of either one of A, U, G or S. For a mutation operation, two RNAs are compared both from beginning to end and also in the reverse direction. The points where the RNAs start to differ from each

other are marked in both sides of the sequences. When the differing middle parts are detected, those are started compare and each Adenin-Urasil and Guanin-Sitozin nucleobases at the corresponding positions are replaced with each other. The other nucleobase correspondences in that middle part are deleted. An example scenario is given on strings in Figure 6.

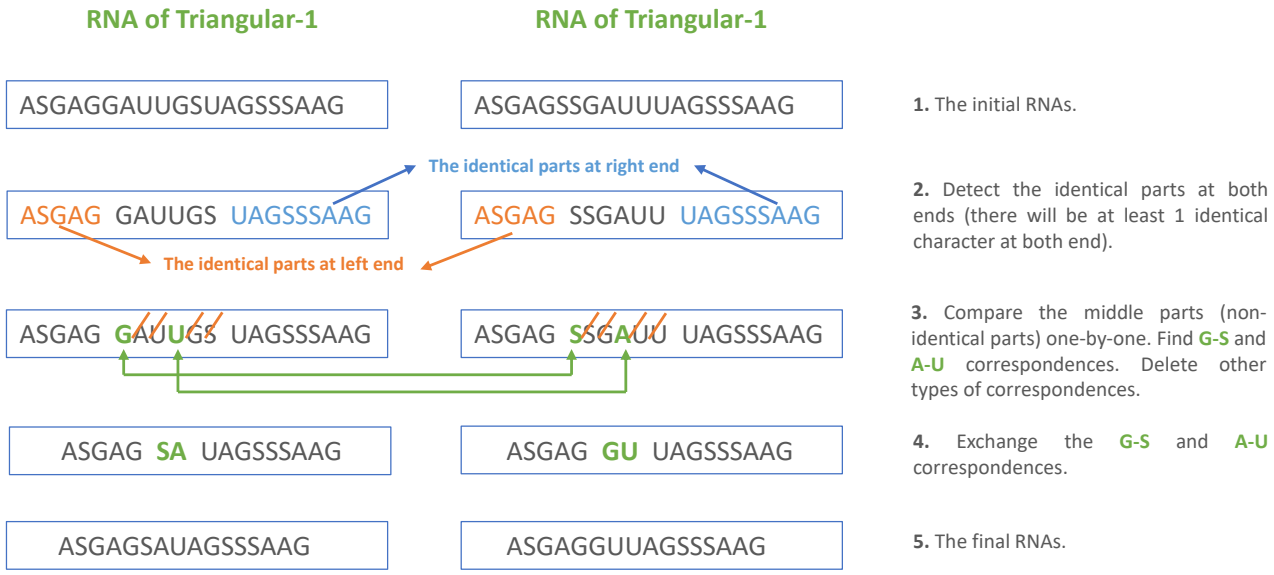


Figure 6: An example of RNA mutation occurring when two Triangular microorganisms come together.

Note: RNA can be of any length, yet two RNAs getting involved in mutation will be of the same length.

3.6 Tissue.cpp

This class holds the cells given. You should be careful while deleting the `Cell` objects killed by Trangular microorganisms since you should remove them from `Tissue` object also. Note that `CreatePartitions()` and `TreatByPlasma()` methods are related with implementing an R-Tree structure for plasma treatment which was left to "your call". You can leave this 2 methods empty (See explanation in the last item of Section 2).

3.7 Exception.h

This header file includes the exceptions defined by the brain. There is nothing you need to implement in this file. However, you are going to use the exceptions defined in this file inside other methods. There are 3 exceptions defined in this file: `ApplePearException`, `DeadCellException` and `NotBornChildException`. You are going to use them as follows: When trying to add different types of `Wall` objects, you should throw `ApplePearException`. For the cases that you are trying to reach a dead cell which was killed by some microorganism, you should throw `DeadCellException`. Lastly, while trying to reach a non-existing child cell of a parent cell (the cell may have not been divided upto that phase), then you are going to throw `NotBornChildException`.

3.8 Main.h & Input Files

In the assignment, cells and microorganisms may also be supplied from txt fles (other than the ones defined in main()). Therefore, there will be given 2 command line arguments: The first one is the file name including cells and the second one is the file name including microorganisms. In Main.h, you are provided two implemented methods: ReadMicroOrganisms() and ReadCells() which are used to read the file of microorganisms and file of cells, respectively. You should not modify this header file. Since file reading part is already implemented for you not to exert additional effort on reading, you do not need to know the details about content of the input files. Nonetheless, in case that you need or wonder, the format of the files are given below:

Cells.txt File

It may have a different name other than "Cells.txt".

```
N M                 // N: number of particles, M: number of cells

first particle      // format: id x_coordinate y_coordinate
second particle
.
.
.
Nth particle

id W                // first cell, W: number of walls
first wall          // if straight wall, format: s p1 p2
second wall         //  ->  's' for straight
.                   //  ->   p1 and p2 are ids of the initial and final particles
.
Wth wall

id W                // second cell
first wall          // if curvy wall, format: c p1 p2 p3
second wall         //  ->  'c' for curvy
.                   //  ->   p1, p2 and p3 are ids of the initial, final
                    //       and center particles
.
Wth wall

.........           // Another cell block
.........           // Another cell block
.........           //          .
.........           //          .
.........           //          .
.........           // Mth cell
```

Microorganisms.txt File

It may have a different name other than "Microorganisms.txt".

```
N                       // N: number of microorganisms

id                      // id of the microorganism
microorganism data      // format: <type> <radius/edge length>
```

```
id                          // if "Circular d"
microorganism data          //   -> Circular whose radius is d

id                          // if "Squadratic d"
microorganism data          //   -> Squadratic whose edge length is d

Another microorganism       // if "Triangular d1 d2 d3
Another microorganism       //     xxxxxxxxxx...x"
.                           //   -> Triangular whose edge lengths are d1, d2 and d3
.                           //      and whose RNA is equal to xxxxxxxxxxxxxx...x
.                           //      where each refers to a nucleobase.
.                           //      RNA can be of any length.
.
Another microorganism       // Nth microorganism
```

4 Grading

- Wall.cpp & CurvyWall.cpp
  - `virtual bool IsContinuousLinear(const Wall&) const;` ..... 8 points
  - `virtual const Wall& operator+(const Wall&) const;` ..... 10 points
- Cell.cpp
  - `void StrengthenCellWall();` ..... 15 points
- Circular.cpp
  - `bool DoesFitInto(const Cell&) const;` ..... 6 points
  - `void React();` ..... 8 points
  - `bool DoesContain(const Particle&) const;` ..... 4 points
- Squadratic.cpp
  - `bool DoesFitInto(const Cell&) const;` ..... 6 points
  - `void React();` .....15 points
- Triangular.cpp
  - `bool DoesFitInto(const Cell&) const;` ..... 6 points
  - `void React();` ..... 7 points
  - `void Mutate(Triangular&);` ..... 15 points
- Memory leak ..... 15 points penalty from total grade

Note that class contructors/destructors/copy constructors and other methods which are not given above will not be graded However, they are already necessary to be well-implemented for the other methods to work correctly! Moreover, to achieve a full grade, valgrind runs should give 0 errors.

5 Regulations

- **Memory-leak:** The class destructors must free all of the used heap memory. Any heap block, which is not freed at the end of the program will result in grade deduction. Please check your codes using `valgrind --leak-check=full` for memory-leaks.
- **Allowed Libraries:** You may include the libraries supplied in header files. Use of any other library (especially the external libraries found on the internet) is forbidden.
- **Programming Language:** You must code your program in C++. Your submission will be compiled with `g++` on department lab machines. You are expected to make sure your code compiles successfully with `g++` using the flags `-ansi -pedantic`.
- **Late Submission:** You have a total of 10 days for late submission. You can spend this credit for any of the assignments or distribute it for all. For each assignment, you can use at most 3 days-late.
- **Cheating:** In case of cheating, the university regulations will be applied.
- **Newsgroup:** You must follow the Ceng242 newsgroup on cow.ceng.metu.edu.tr for discussions and possible updates on a daily basis.

6 Submission

Submission will be done via Ceng Class. Do **NOT** write a `main` function in any one of the files. You are going to submit the following files by directly compressing as **HW4.zip**:

- Wall.h, Wall.cpp
- CurvyWall.h, CurvyWall.cpp
- Cell.h, Cell.cpp
- MicroOrganism.h, MicroOrganism.cpp
- Circular.h, Circular.cpp
- Squadratic.h, Squadratic.cpp
- Triangular.h, Triangular.cpp
- NucleoBase.h, NucleoBase.cpp
- Tissue.h, Tissue.cpp

You are not going to submit a `Main.cpp` file. I will use my own `Main.cpp` during grading. There is a single `Main.cpp` among the homework documents. You may use it while developing and debugging you codes, if you like.

Note: The submitted zip file should not contain any directories. The following command sequence is expected to run your program on a Linux system.

```
$ unzip HW4.zip
$ make clean
$ make all
$ make run
$ -optional- make valgrind
```