In this homework, you should add the exception handling into your program. Besides, we prefer to use the GTK+ library as the GUI library. There is another program to let you practice the GTK+ library. Also, you should continually maintain your test code and modify your class diagram.

1. (50%) Incorrect input may cause run-time error while your program is executing. In previous programs, we don’t consider how to handle the incorrect data in your input file and main program. But now you have to use the exception mechanism in C++ to handle these abnormal inputs. In this section, we define several exceptions as the following list you have to handle in your program:

   I. **UndefinedTagException**
   There are some specified tokens that could change state of process, and we call them “Tags.” You may read an undefined tag which is misspelling, when you read a line from the input file. The program should recognize these undefined tags and throw an exception. For example, if you read a “//geralizations” string, your program should recognize this kind of misspelling exception. Also, in your exception handler, your program should show the line number which exception happens and the misspelling string.

   II. **UnexpectedTagException**
   We use a finite state machine to simulate the parsing process in previous homework. When you stay in one state, there are some limits to restrict your action. For example, being at the state of parsing a class, the next tag you read may be “//class,” “//interface,” “//attributes,” “//operations,” or “//generalizations” tag. There is no possible way to read a “//relationships” tag. So, if your program reads these unexpected tags at one parsing state, it should throw an exception and report the line number and the tag you read.

   III. **ParameterErrorException**
   This type of exception happens in a parsing state. In one parsing state, your program would process some parameters for the specific purpose. For example, being the state of parsing the attributes for a class, each line has two strings. One is the name of the attribute; the other is the type of the attribute. If your program merely read one string in a line, it should throw an exception because there are not enough parameters to represent an attribute. Also, you
should report the line number and the tag you read.

IV. **ConflictException**

In homework 2, we request that each element in a class diagram should be unique. But we don’t tell clearly how to archive this requirement. You may have some complicate logic in your main program in order to check whether the element is unique. Now, the insert function should handle these logics and throw an exception if there’s a conflict happened (one element has already existed). The exception which the insert function throws depends on which kind of element you insert. For example, if you insert a class which has already existed, your program should an exception called ClassConflictException. Also, your program should report a message on the screen.

The first three types of exception are based on the finite state machine of your parser program. You should throw these exceptions in the concrete state class of the state pattern. The last exception occurs when you insert a new element into your diagram. You have to use the try/catch block in the context to handle these exceptions.

2. (30%) You would write a simple program called GTKCounter with GUI mode and use the GTK+ library. In figure 1, there are one label to show the counter value and two buttons to increase or decrease the counter value. Initially, counter value is 0, the Inc button is enabled, and the Dec button is disabled. If we click the Inc button, the counter value becomes 1 and the Dec button is enabled, and so on. The counter is limited to 0…9, i.e., if the Inc button is hit 9 times, counter will display 9 and the Inc button is disabled. Then after we click the Dec button, the counter value becomes 8 and the Inc button is enabled again, and so on.

![Figure 1: Program Layout](image)

3. (10%) Please keep your unit tests running. You have to add any new tests for the new programs written in this homework. Also, you should maintain your unit test programs written before.
4. (10%) Draw class diagrams for the program in UML. To do this they need to have every function encapsulated in a class, if not already so. The class diagrams will be drawn in two perspectives: design class diagram and implementation class diagram. (Note: you do not need to draw the class diagram for unit tests).