Exercise 8:

Submit your solutions (source code) to the questions below by email to your instructor and TA(s) by Monday, November 26th (16:30).

Question 1: Introduction to exception (30 points).

In this question, you will learn how to throw an exception to signal an unexcepted behavior in a program and how to catch the exception and handle it.

Consider the function stringToInt below that takes a string object representing an integer as argument and returns this integer. A possible implementation of this function is given below:

```
// stringToInt.cpp
#include < string >
#include < sstream >
#include < iostream >


int stringToInt(const string& input) {
 stringstream instream;
 instream << input;
 int number;
 instream >> number;

 if (instream.fail()) {
   // Error: the input can not be converted
   cerr << "input can not be converted to an int" << endl;
   return -1;
 }

 char left;
 instream >> left;
 if (!instream.fail() {
  // Error: there are some characters left after the int
  cerr << "input can not be converted to an int" << endl;
  return -1;
 }

 // everything went fine: returns the int
 return number;
}

int main(void) {
 string test1 = "11";
 cout << stringToInt(test1) << endl;

 string test2 = "cc11";
 cout << stringToInt(test2) << endl;

 string test3 = "11cc";
 cout << stringToInt(test3) << endl;

 cout << "Tests passed" << endl;
```

```
    return 0;
}
```

In this code, if the argument does not represent a valid integer, an error message is printed out and a dummy number (-1) is returned. This approach presents several problems (e.g. what happens if the argument is "-1"?).

Replace the code in stringToInt such that it throws an exception instead. Use the exception invalid_argument (this exception takes as argument a string that describes the type of error that occurred). invalid_argument is defined in the header <stdexcept>. Modify the main function such that it catches exception of type invalid_argument. To get details about the type of error that occurred, please use the method of the class invalid_argument named what().

Question 2: Exception - catch and throw (30 points).

In this question, you will learn to handle allocated resources when an exception is thrown.

Consider the following code:

```
// exception.cpp
#include <stdexcept>
#include <cstdio<
#include <iostream<

using namespace std;

void doSomeComputation() {
 throw runtime_error("failure during doing some computation");
}

void example() {
 FILE* logfile = fopen("logfile.txt", "w+");
 if (!logfile) {
   throw runtime_error("failed to open file");
 }

 fputs("log1", logfile);

 // call a function that performs some computation and may throw
 // an exception
 doSomeComputation();

 cout << "closing logfile" << endl;
 fclose(logfile);
}

int main(void) {
 cout << "Calling example()" << endl;
 example();
 cout << "After calling example()" << endl;
 return 0;
}
```

Compile the code above and run the generated program. This program will abort because an exception thrown in the function doSomeComputation() is not caught. Modify the code above by using try-catch such that it handles exception correctly. Make sure that allocated resources (e.g. the pointer to FILE in the function example()) are correctly released (Use the try-catch-throw idiom).

Question 3: Exception - RAII (40 points).

In this question, you will see how to use the Resource Acquisition Is Initialization (RAII) paradigm to handle allocated resources when an exception is thrown.

Write a class LogFile encapsulating the pointer to FILE in the code above. The constructor of the class LogFile should take as argument a filename. Using this class rewrite the code in question 2 so that it is exception safe