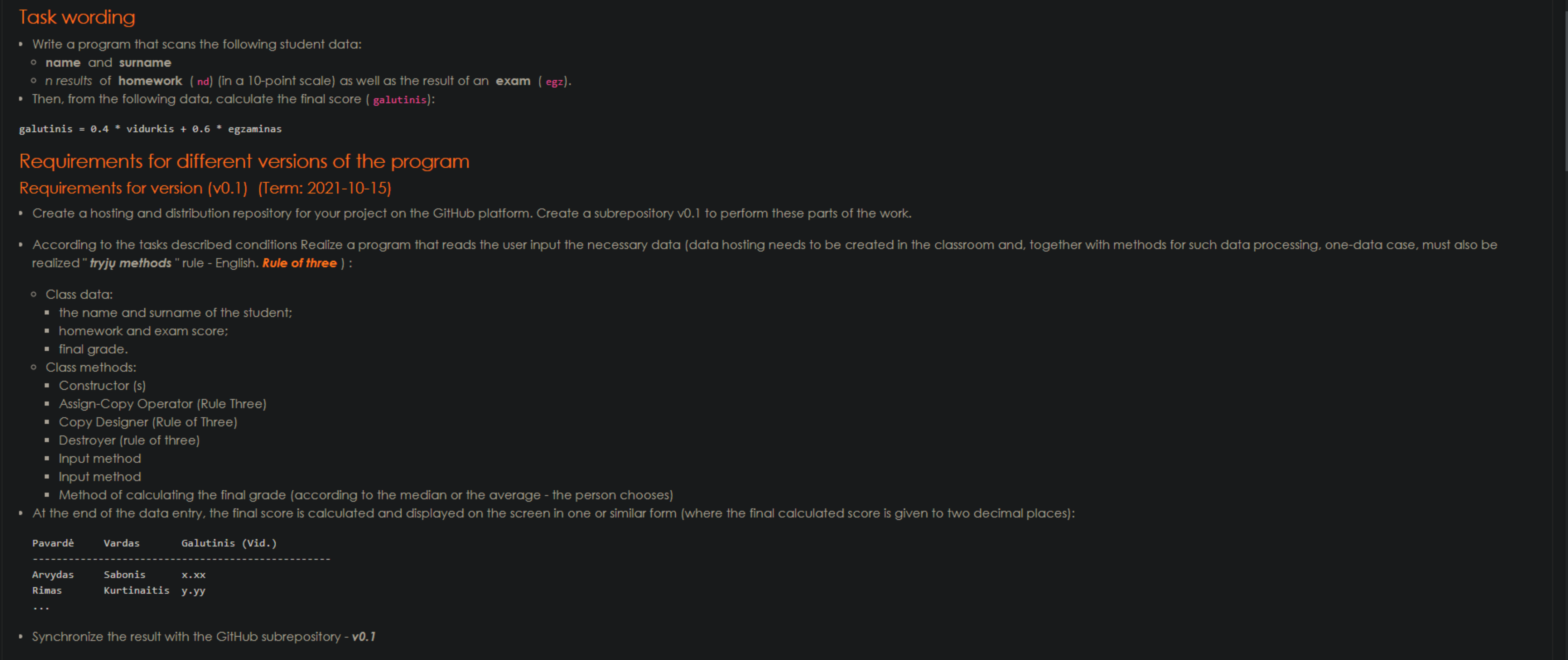


Task wording

- Write a program that scans the following student data:
  - **name** and **surname**
  - *n results* of **homework** (`nd`) (in a 10-point scale) as well as the result of an **exam** (`egz`).
- Then, from the following data, calculate the final score (`galutinis`):

```
galutinis = 0.4 * vidurkis + 0.6 * egzaminas
```

Requirements for different versions of the program

Requirements for version (v0.1) (Term: 2021-10-15)

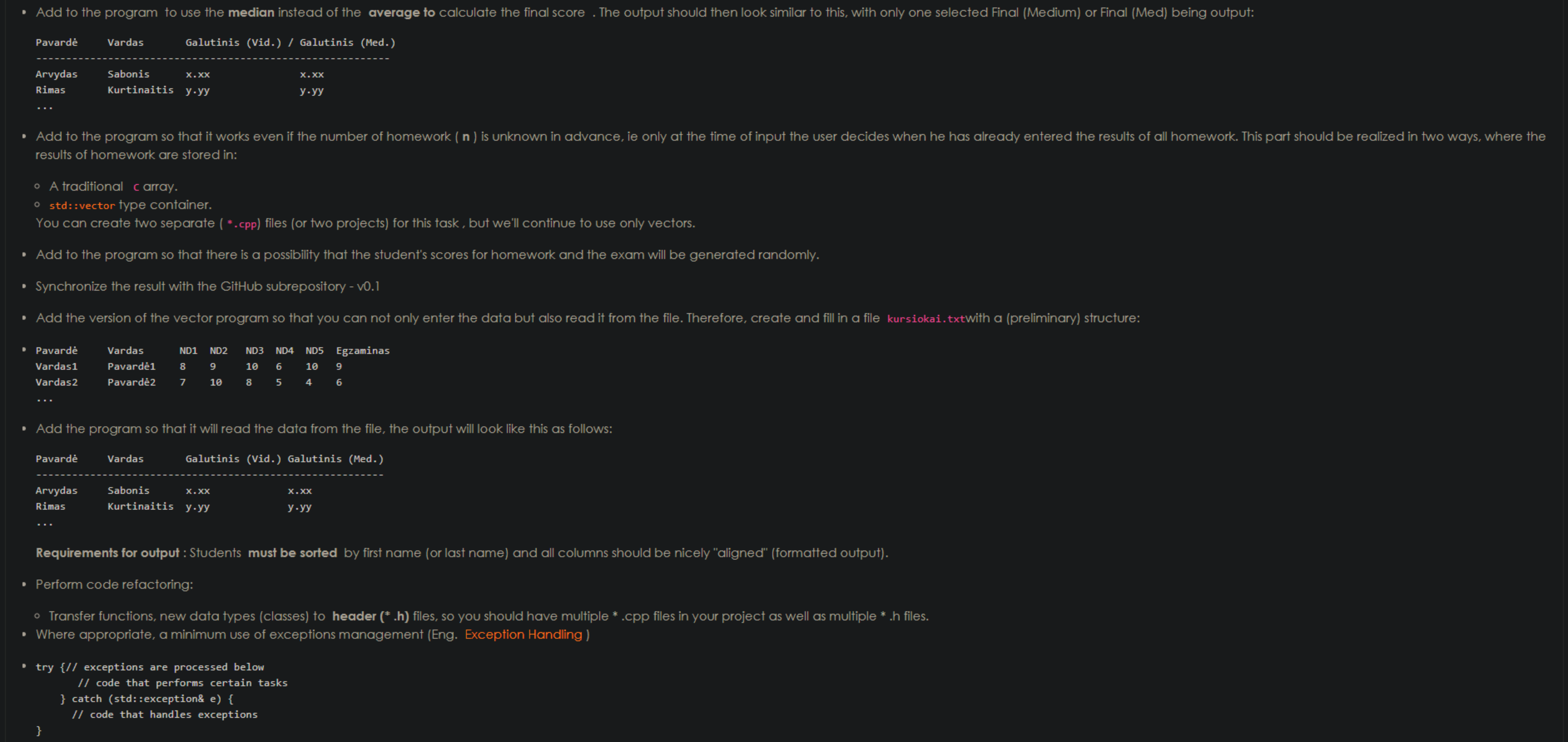
- Create a hosting and distribution repository for your project on the GitHub platform. Create a subrepository v0.1 to perform these parts of the work.

- According to the tasks described conditions Realize a program that reads the user input the necessary data (data hosting needs to be created in the classroom and, together with methods for such data processing, one-data case, must also be realized " ***tryjų methods*** " rule - English. ***Rule of three***) :

  - Class data:
    - the name and surname of the student;
    - homework and exam score;
    - final grade.
  - Class methods:
    - Constructor (s)
    - Assign-Copy Operator (Rule Three)
    - Copy Designer (Rule of Three)
    - Destroyer (rule of three)
    - Input method
    - Input method
    - Method of calculating the final grade (according to the median or the average - the person chooses)
- At the end of the data entry, the final score is calculated and displayed on the screen in one or similar form (where the final calculated score is given to two decimal places):

```
Pavardė     Vardas       Galutinis (Vid.)
--------------------------------------------------
Arvydas     Sabonis      x.xx
Rimas       Kurtinaitis  y.yy
...
```

- Synchronize the result with the GitHub subrepository - ***v0.1***



- Add to the program to use the **median** instead of the **average to** calculate the final score . The output should then look similar to this, with only one selected Final (Medium) or Final (Med) being output:

```
Pavardė     Vardas       Galutinis (Vid.) / Galutinis (Med.)
-----------------------------------------------------------
Arvydas     Sabonis      x.xx              x.xx
Rimas       Kurtinaitis  y.yy              y.yy
...
```

- Add to the program so that it works even if the number of homework (**n**) is unknown in advance, ie only at the time of input the user decides when he has already entered the results of all homework. This part should be realized in two ways, where the results of homework are stored in:
  - A traditional `C` array.
  - `std::vector` type container.

  You can create two separate (`*.cpp`) files (or two projects) for this task , but we'll continue to use only vectors.

- Add to the program so that there is a possibility that the student's scores for homework and the exam will be generated randomly.

- Synchronize the result with the GitHub subrepository - v0.1

- Add the version of the vector program so that you can not only enter the data but also read it from the file. Therefore, create and fill in a file `kursiokai.txt`with a (preliminary) structure:

-
  ```
  Pavardė     Vardas      ND1  ND2   ND3  ND4  ND5  Egzaminas
  Vardas1     Pavardė1    8    9     10   6    10   9
  Vardas2     Pavardė2    7    10    8    5    4    6
  ...
  ```

- Add the program so that it will read the data from the file, the output will look like this as follows:

```
Pavardė     Vardas       Galutinis (Vid.) Galutinis (Med.)
---------------------------------------------------------
Arvydas     Sabonis      x.xx             x.xx
Rimas       Kurtinaitis  y.yy             y.yy
...
```

  Requirements for output : Students **must be sorted** by first name (or last name) and all columns should be nicely "aligned" (formatted output).

- Perform code refactoring:
  - Transfer functions, new data types (classes) to **header (* .h)** files, so you should have multiple * .cpp files in your project as well as multiple * .h files.
- Where appropriate, a minimum use of exceptions management (Eng. Exception Handling)

-
  ```
  try {// exceptions are processed below
         // code that performs certain tasks
      } catch (std::exception& e) {
        // code that handles exceptions
  }
  ```

- Synchronize the resulting result with GitHub subrepository v0.1 and master, release the working version "release".

Who needs all this? And e.g. what happens if the file you are trying to open does not exist; or you are trying to get an element of an array that does not exist

Requirements for version (v0.2) (Term: October 25, 2021)

- Release the new subrepository ***v0.2*** to perform the next task.
- Upgrade (if needed for `v0.1` the implementation available for the last assignment) and generate **five** random student list files consisting of: 1,000, 10,000, 100,000, 1,000,000, 10,000,000 entries. You can generate "Templates" for Names, such as. `Vardas1 Pavarde1`, `Vardas2 Pavarde2` etc.
- Sort (divide) students into two categories:
  - Students with a **final score <5.0** could be called “poor”, “downtrodden” and so on.
  - Students with a **final score> = 5.0** could be called "hard", "heads" and so on.
- Export sorted students to two new files.
- **Perform a program speed analysis** , ie measure the speed of the program by distinguishing how long each of the following steps took:
  - file creation;
  - retrieving data from files;
  - grading students into two groups / categories;
  - output graded students to two new files.

Repeat the test of the program and describe it using the five data files generated with different record sizes.

- Synchronize the result with the GitHub subrepository ***v0.2*** .

- **Container testing was performed as follows:** Measure the performance of the enhanced sales depending on the use of one of the three containers:

  - std::vector
  - std::list
  - std::about

  That is, if you have created an object **Students** (or otherwise) and so far used `std::vector<Studentai>`, you have to examine whether the change and how to change the speed of the program, if instead of `std::vector<Studentai>` use `std::list<Studentai>` and `std::deque<Studentai>`.

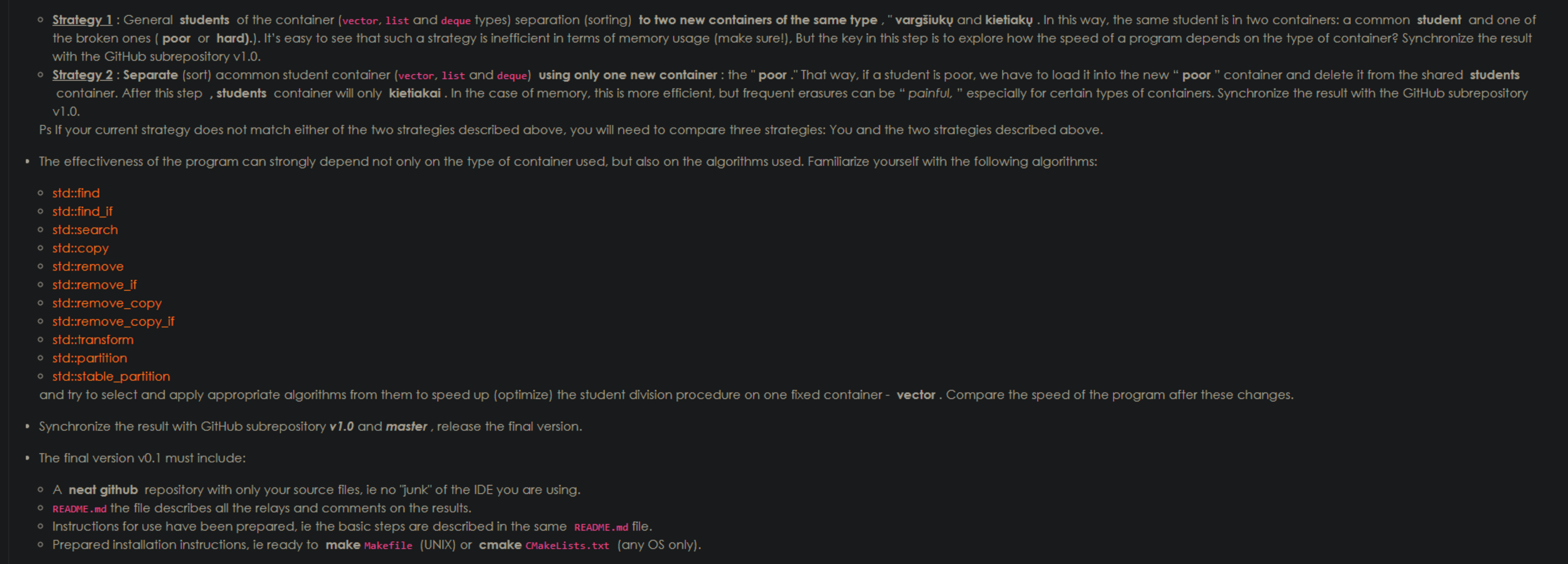
This is important because both the creation of the files and the output of the sorted results to the files do not depend on the container used, so only the following steps in the program should be measured in this task:

- retrieving data from files;
- sorting students into two groups / categories;

- Synchronize the result with GitHub subrepository ***v0.2*** and ***master*** , release the working version ***releas*** .

Requirements for version (v1.0) (Term: 2021-11-05)

- Release the new subrepository ***v1.0*** to perform the next task.
- **Optimize the implementation of student sorting (division) into two categories ("poor" and "hard") (examine memory requirements):** ie for all three container types (`vector`, `list` and `deque`) measure the speed of the program depending on the strategy of dividing students into two categories:



- **Strategy 1** : General **students** of the container (`vector`, `list` and `deque` types) separation (sorting) **to two new containers of the same type** , " **vargšiukų** and **kietiakų** . In this way, the same student is in two containers: a common **student** and one of the broken ones (**poor** or **hard).**). It's easy to see that such a strategy is inefficient in terms of memory usage (make sure!), But the key in this step is to explore how the speed of a program depends on the type of container? Synchronize the result with the GitHub subrepository v1.0.
- **Strategy 2** : **Separate** (sort) acommon student container (`vector`, `list` and `deque`) **using only one new container** : the " **poor** ." That way, if a student is poor, we have to load it into the new " **poor** " container and delete it from the shared **students** container. After this step **, students** container will only **kietiakai** . In the case of memory, this is more efficient, but frequent erasures can be " *painful,* " especially for certain types of containers. Synchronize the result with the GitHub subrepository v1.0.

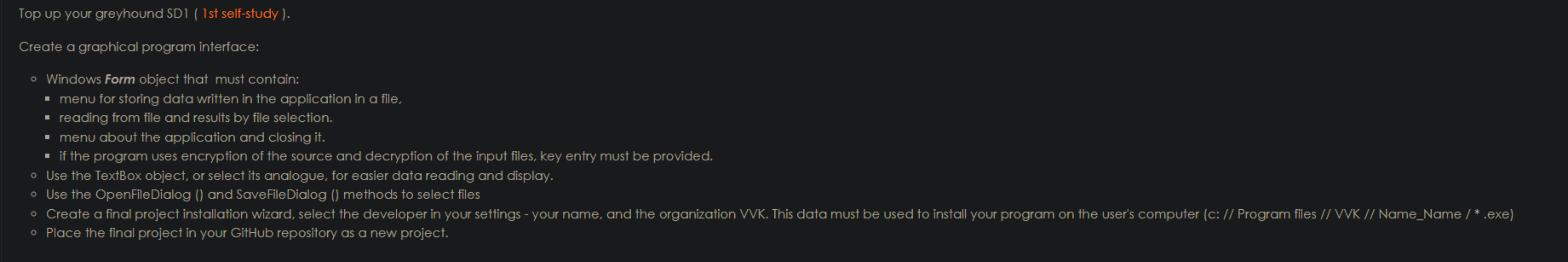
Ps If your current strategy does not match either of the two strategies described above, you will need to compare three strategies: You and the two strategies described above.

- The effectiveness of the program can strongly depend not only on the type of container used, but also on the algorithms used. Familiarize yourself with the following algorithms:
  - std::find
  - std::find_if
  - std::search
  - std::copy
  - std::remove
  - std::remove_if
  - std::remove_copy
  - std::remove_copy_if
  - std::transform
  - std::partition
  - std::stable_partition

  and try to select and apply appropriate algorithms from them to speed up (optimize) the student division procedure on one fixed container - **vector** . Compare the speed of the program after these changes.

- Synchronize the result with GitHub subrepository ***v1.0*** and ***master*** , release the final version.

- The final version v0.1 must include:
  - A **neat github** repository with only your source files, ie no "junk" of the IDE you are using.
  - `README.md` the file describes all the relays and comments on the results.
  - Instructions for use have been prepared, ie the basic steps are described in the same `README.md` file.
  - Prepared installation instructions, ie ready to **make** `Makefile` (UNIX) or **cmake** `CMakeLists.txt` (any OS only).



Top up your greyhound SD1 (1st self-study).

Create a graphical program interface:

- Windows ***Form*** object that must contain:
  - menu for storing data written in the application in a file,
  - reading from file and results by file selection.
  - menu about the application and closing it.
  - if the program uses encryption of the source and decryption of the input files, key entry must be provided.
- Use the TextBox object, or select its analogue, for easier data reading and display.
- Use the OpenFileDialog () and SaveFileDialog () methods to select files
- Create a final project installation wizard, select the developer in your settings - your name, and the organization VVK. This data must be used to install your program on the user's computer (c: // Program files // VVK // Name_Name / * .exe)
- Place the final project in your GitHub repository as a new project.