Assignment Guidelines
CS240, Fall 2020
Instructors: Mark Petrick

1 General Requirements

The solutions you hand in must be your own work. In particular, **you are not supposed to look up the solutions in the literature or on the Internet.** Assignments will involve mostly "written" (non-programming) work but may also have a programming component. Both parts will need to be submitted electronically through MarkUs. Written work is to be submitted in a **pdf** file and, if the assignment includes a programming part, you must submit the program file with the file name specified in the assignment. **No late assignments will be accepted.**

Tools/methods for **creating pdf files** for written solutions are provided on the website at https://www.student.cs.uwaterloo.ca/~cs240/f20/assignments.phtml#pdf for CS240.

MarkUs is a web-interfaced submission and marking system. Instructions for **submitting assignments** to MarkUs and for using MarkUs to view your marked work are on the website at https://www.student.cs.uwaterloo.ca/~cs240/f20/assignments.phtml#submit for CS240. You should always check whether all files you wanted to submit were accepted on MarkUs. Note that you can always re-submit new versions of your files, delete previously submitted files, or submit different parts of the assignment at different times. Programming questions may also require a written description of algorithms, data structure design overview, proofs, analysis, etc. Submit any requested additional information (other than code) together in the pdf file with the written parts.

We recommend that you start early and submit partial solutions, especially for the programming parts, as you finish them, instead of waiting until the last moment.

For algorithm and data structure design questions you should always follow the given runtime. If not mentioned in the assignment, this is by default, worst case runtime. If you cannot think of an algorithm/data structure to match the runtime specifications in the assignment, you can give an answer with an alternative runtime (state the runtime in your explanation/comments) and you may receive partial credit.

2 Non-programming Questions

Ensure that your solutions are complete and mathematically precise, and at the same time, easy to understand and to the point. Your solutions will be judged not only for correctness, but also for quality of your presentation and explanations. Justifications of all claims are implicitly required unless stated otherwise.

For questions that ask you to **design/describe/give an algorithm or a data structure**, you should (unless stated otherwise) design the best algorithms you can come up with. The first criterion for marking is the **correctness**, the second criterion is efficiency. Thus, an algorithm which is slow but correct will receive substantially more marks than an algorithm which is fast but incorrect. Inefficient algorithms will not earn full marks. In your solution, enclose the following:

- describe the main idea first in words,
- present a clearly written pseudocode (at a level of detail mimicking the style of the lectures, the model solutions, or the book),
- give a correctness proof or justification, and
- include an analysis (usually, of the running time).

For questions that ask you to **argue/prove a statement**, you are expected to give a formal proof, and justification for any "non-obvious facts"** that you use in your proofs. In general, whenever you claim a "non-obvious fact"**, you must have a justification for it. This may consist of demonstrating your work, and the series of steps which lead you to the answer.

***A "non-obvious fact" is any fact which has not been stated/proved in class. In the case that you are using facts stated in lectures/modules, be sure to cite where you got them from.*

3 Programming Questions

Programming questions will usually involve designing an algorithm or a data structure, and implementing it. You should include sufficiently many comments in your code that the main ideas of design and correctness are clear; the assignment will specify if you need to give further justification of correctness and/or analysis in a separate (written) file. The rest of this section deals with the instructions for the programming part of the questions.

Your program should be implemented in C++ on the undergrad Linux environment (`ubuntu.student.cs.uwaterloo.ca`). If you are working on a different platform, it is your responsibility to ensure that your program runs properly on the undergrad environment.

Programs will generally be expected to read from the standard input stream (`std::cin`) and write to the standard output stream (`std::cout`). That is, no file names will be given on the command line. Assignments may also specify the names of certain classes that should be implemented, in addition to a main method. Good programming practice generally dictates

that each class definition should appear in a separate header file, to be included in the main file. **The names of all files to be submitted will be specified in the assignment, along with their contents.** Skeleton code will usually be distributed for all these files as well.

We will compile and run your program as follows:

```
g++ -std=c++17 *.cpp -o ProgramName
./ProgramName  <input_file >output_file
```

Since we will compile/run/test your program using an automatic script, it is vital that you follow our instructions to the letter. In particular, your program should read only from `cin` and write only to `cout`, in the format exactly as specified in the assignment. It is a good idea to test your program thoroughly yourself (on your own input) to ensure its correctness. We recommend you use the standard error stream `std::cerr` for debugging output, which will be ignored by our marking scripts. Some additional pre-testing might be available—this will be announced in the course newsgroup (Piazza).

Marking of programming questions will be based on both correctness (as determined by our test runs) and coding style (documentation, design, clarity, etc.).

In the case you passed the public test, and failed some secret tests, if your program works with a **one line change**, then we may accept it for remarking and give some marks after a penalty.

4 Further information

If you have any questions about the formatting of assignments, or about how much information to include, ask the tutor or make a newsgroup (Piazza) post. Also be sure to check the website at https://www.student.cs.uwaterloo.ca/~cs240/f20/info.phtml for CS240 for details on the return policy, remark requests, and academic dishonesty (cheating).

5 How to Get High Marks on Your Assignments

Adapted from a text by Jeff Shallit.

1. Write legibly. If possible, type your solutions, using a word processor or (even better) the LaTeX typesetting utility.

2. If you have left some problems unsolved, say so.

3. Make your answers clear and concise, but don't omit details. Be sure your arguments will convince a skeptical (but intelligent) TA.

4. If a final result is requested, show all your work to get that solution, and put a box around the final answer.

5. Don't try to fool the TAs! The following phrases are tip-offs: "Obviously...", "It can be easily shown that...", "It is clear that..."

6. When doing a proof by induction, state precisely what your induction hypothesis is and on what variable you are doing the induction.

7. If you are unsure whether your solution/proof is correct or not, you need to work out more details and write them down in your solution.

8. Make your solution look more like the text from the text book rather than quick-and-dirty notes from the class.