CSC 228 Course Guide

J.N. Clarke and D.L. Horton

Contents

1 Programming language

In this course you must write your assignments in C++. If you have a version of C++ on your computer at home, you may use it to write your programs initially, but keep in mind that you may be required to submit your assignments electronically, and your programs must be able to run on the machines in our course lab.

2 Help with bugs

This section offers some suggestions for debugging. They are rather straightforward ideas that should be familiar to you, but judging from my experience, they bear repeating.

2.1 How to find a bug

When you have a logical error, these are some general steps you can take to fix it:

- Use your powers of deduction to narrow your focus to a small piece of code that is likely the source of your bug. Don't try to do this by tracing the whole program — that takes too long.

  One way to localize the bug is to put a print statement, roughly in the middle of the main function, that prints out the state of affairs. If your program gets to there without crashing and with appropriate values in the variables, the the problem is in the second half. If not, it's in the first half. Keep doing this. Ever heard of divide and conquer, or binary search? `:-)`

  If you narrow the bug down to a function call, then apply the same technique to the body of that function. Eventually, you will have the bug localized to a very small piece of code containing no calls to your own functions.

- At this point, your print statements show you the values of the relevant variables just before and just after that small piece of code. If the before values are sensible and the after values aren't, then you have successfully localized the bug.

- Then get out some paper and **trace the offending code by hand**. Your trace will show you what's happening (and why), which will not be what you wanted to happen (hence the bug!). Now you know what needs to be fixed! If your trace indicates that the code *does* do what you wanted (and yet step 2 tells you that it doesn't), then there is an error in your trace. Do it again, being even more detailed and accurate.

Once you learn to use a debugger, such as gdb or the debugger that comes with your development environment (if you have CodeWarrior or Borland, for example), you'll be able to do some of this more easily. A debugger will let you start and stop your program anywhere you like, and to inspect your variables — without using any print statements.

2.2 Writing your code so that debugging will be easier

Debugging is, of course, much easier if your program is relatively small and simple. This will be the case — even if you're working on a huge program — if you write your code incrementally. The first code you write, run, and debug, will be a small step on the way to the finished product. So it will be easy to debug. Don't just get the easy bugs out though; test this version thoroughly while you're here.

The next step will build a little more on top of an already working program, so the new code (which is the part most likely to have bugs) will be small, and easy to debug. And so on.

This is why writing code incrementally can get you to the final product *faster* than trying to write, debug and test the whole thing at once.

2.3 How to bring a bug to office hours

Before you bring a bug to office hours, you must first take steps such as those described above to find it yourself. Almost all of the time, those steps will show you the bug and you won't even need to talk to us. But if they don't (this might be because you have a misconception that prevents you from doing a correct trace, for example) then you can ask us for help. But we won't be interested in looking at it unless you bring along:

- A completely up-to-date printout of your code. If it has hand-written changes, we'll ask you to go get an up-to-date printout.
- An explanation of exactly what small piece of the program it is that you've narrowed the bug down to. This piece of code should be extremely short — at most about 20 lines.
- Input and output from a test run showing the state of affairs before and after the offending piece of code is executed.
- Your careful and detailed hand trace of the offending piece of code.

3 Assignments

There are several assignments in this course, which you will do alone, followed by a project, which you will do in a group of two or three students. Your project will be done in 3 parts: a preliminary plan written early in the project, a brief oral presentation made during a tutorial, and a final written report (which will include your programs, testing etc.). Your mark will depend on these three items and on an assessment of how well you worked as a team.

Few of you are used to group work, so collaborating on the project will require more careful planning than usual. (See section 5.)

There will be handouts giving specific details of each assignment at the appropriate times. This section gives an overall picture of what we are trying to teach you with the assignments, and how you can be successful with them.

4 What your reports should look like

What really counts in your assignment is that your tutor must be able to understand what you did. Most of the marks, therefore, are determined by your report, not your program. Think of the report as being similar to a laboratory report in physics or chemistry: you wouldn't expect to be marked primarily on circuits or test tubes that you hand in, so here you shouldn't expect your mark to be based primarily on your program.

Because the report is so important, you should not make the mistake of trying to write it at the last minute. You should work on the report from the beginning, developing it in parallel with your program.

You will want to write a complete and working program, but if you are unable to do so, tidy up what you have so that you can demonstrate the parts that do work. For example, you can comment out faulty functions or write "stubs" to replace them. Or pull out working pieces and write a small program to test them. Write a good report, including a clear and honest explanation of the parts that do and don't work, and thorough testing of the parts that do. (All of this will be easy to do if you have debugged and tested your program in small pieces, as you developed it, and hard to do if you have not.) This sort of assignment solution can earn a very good mark.

In writing your report, remember that your audience is your tutor: an experienced programmer who knows the requirements you are trying to satisfy, but does not know the internal details of your program. Describe what you have done so that it will be easy for the marker to understand whatever parts of the program he or she may need to read.

The report should include:

- Title page and table of contents.
- Introduction. This is simply an overview of the program's purpose, generally one paragraph.
- Program structure. A description of how you broke your program into classes and a brief description of each class. Include a diagram of the inheritance hierarchy. Be sure to use a legend to explain the precise meaning of all notation in your diagram.
- Data structures. Explain your data structures and why you chose them. Include a diagram of each data structures.
- File layouts. The exact format of each input and output file. Consistency is very important in your descriptions; the same format should be used to describe each data file.
- Bugs and missing features. Be honest about problems you encountered and precise about exactly what does and does not work. Explain the reasons why you had trouble and what you did about it. If you know of no bugs or missing features, you must still include this section. Just say "No known bugs or missing features".
- Testing strategy. How did you select your test cases and what does each one actually demonstrate? Your goal here is to convince the reader that your test cases demonstrate not just that your program works on those particular cases, but on *all* possible cases. This is a rather strong claim and needs a strong argument to support it.

  Each test case should be given a number so that the report can be cross-referenced to the test input and output.
- Conclusions. This may be a discussion of your results, or simply a brief statement of what your program accomplished.
- Appendices. You must include the program source code as well as any data input and output files you used. The program files should appear in alphabetical order and each output file should appear immediately following its corresponding input file. Output files should be annotated to help the reader see what are the most important things to look at, and what conclusions you would like them to draw.

In the Program structure, Data structures, and File layouts, if a design decision was left to you, you should discuss the alternative designs you considered, tradeoffs among them, and how you made your choices.

If your program is not working and you cannot finish debugging it before an assignment deadline, make sure that your report is well-written and indicates which part of the program does not work. The appropriate place for this is in the *bugs and missing features* section, not in the introduction or conclusions.

5 How to form a group for the project

For the term project, you must work in teams of two or three. You can choose your own partner or partners, or you can ask your tutor or instructor to assign partners. In either case, you must think carefully about how your team can share its work most effectively.

Important talents in group programming work include:

- designing programs: the high-level job of designing objects and their public interfaces, how the objects will relate to each other (by inheritance, for example), and choosing data structures and algorithms.
- writing code: the low-level work of declaring variables, writing loops, etc., and debugging.
- writing essays and reports, and giving presentations — communicating with people rather than computers.
- testing programs. It is a challenge to do very good testing. You may find you enjoy the process, especially in a group setting where one partner can take on the job of devising fiendish test files to poke holes in the others' code.
- organizing the group: making sure things happen on schedule, and planning a recovery strategy when they don't.

Each team should include members with a range of skills. Report-writing and testing are areas that may be difficult for CSC 228 students, but right from the beginning you must think about how you will accomplish these tasks.

See page 9 for a self assessment questionnaire that you may find helpful. Answer it honestly, and then share your answers with potential partners to help determine whether or not you would form a productive team.

5.1 How to organize your group

You should think about who you might like to work with from the beginning of term. Once your group has formed, think about how you will work together. This is something that will evolve as you get to know each other — and remember, even old friends don't know each other in the way co-workers do.

One possible organization is "democratic," perhaps better described as "anarchic". This will work nicely if all of you take a responsible attitude and make sure that you each do a little more than you really should, so that the boundaries are well covered, everyone is prepared for group meetings, and there isn't one person who is always left to pick up the jobs nobody wants.

Another organization is "hierarchical," or perhaps "autocratic". If one of you is a good organizer and the others tend to put things off, working best to externally-imposed deadlines, then perhaps the organizer should be put in charge of calling meetings, assigning tasks, and so on. The organizer must be careful not to assign too much or too little work to any one person; and the other group members must remember that organizing is significant work, and give the organizer credit for taking care of it.

Different group organizations will work well for different groups. What matters is that you should deal with this issue explicitly rather than hoping it will solve itself. If your group does run into trouble, think of the way you're organized as a possible cause.

5.2 What if your group is in trouble?

Group assignments, properly organized in a cooperative spirit, can reduce the total amount of work for everyone. If things go wrong, however, the group experience can be very unpleasant. One of the reasons for doing group work in this course is so that you will learn how to organize it; naturally, since most of you are new to this approach, there will be some difficulties.

Right from the start of your group work, you must realize that you are responsible to others as well as to yourself. You may very successfully have done all your assignments at the last minute in the past — but to do so in a group is not fair to your colleagues, who at the very least will be seriously worried about whether you will complete your share of the task. It is not good enough to assure them of your competence; you must try to accommodate others' expectations.

For these and other reasons, you may feel that your group is in trouble. These troubles can only be resolved by discussion: first, among yourselves, and then, if that does not succeed, by discussion with your tutor or with the course instructor. Be pro-active: initiate discussion *as soon as you realize there might be a problem,* so that there will be time for us to help you resolve it.

5.3 Warning

Working in a group is harder than most CSC228 students anticipate. If you take seriously the tasks of choosing team members, coordinating your work, and negotiating when you inevitably have disagreements, you will do well. But if you don't, problems are almost guaranteed to arise, even if you've know each other since grade 3. In fact, you wouldn't be the first student to lose a longtime friend over a CSC228 course project.

So do take the group work issue seriously, and if you have problems, see me sooner rather than later.

6 How your assignments are marked

6.1 The basic marking scheme

Each assignment and the project has its own particular requirements, but marking will follow the general principles described here. Some aspects may change, because every class and every marker is individual, and because the experience of marking an assignment tells a marker things that can't be known beforehand; this outline explains what aspects of your work are most important.

Your tutor will read your report first, referring to your program and data files when directed to by your report. The marker will be interested in the length of your report, its organization, as well as spelling and grammar. Use a spell-checker to correct mistakes and have a friend who is not in CSC228 read over your report to make sure they can understand it. The marker will look closely at your *conclusions* section, as well as your *bugs and missing features* section. If these sections are missing, you will lose marks. (If you have no bugs or missing features to report, say so.) The marker will also look closely at your *testing strategy.* His or her impression of the correctness of your program will be based heavily on this; after all, it is your responsibility to convince us that your program works. Your computer output will be read as a reference: to check a point you make in the report, or to learn something extra about your work.

It is all right to copy pieces of code from other references, for example a library text, **but you must acknowledge the source!** Not to do so is an academic offense. Of course, any such pieces of code must be small relative to the whole assignment.

Your mark will be based on four components:

- Quality of report.
  Is the report complete and *concise*, or does it just babble on? Is the student honest about problems? Is the report well written? See section 4 below for more details.
- Achievement.
  Were the requirements of the assignment completely or just partially met?
- Quality of code.
  Is the code well designed? Is it modular and general? Are the comments thorough? Are there unexplained hacks? Would the code be easy to modify, extend, or reuse in a different context?
- Testing.
  Are the test cases well chosen or is the reader merely inundated with output? Are boundary cases well

covered? Is the testing scheme clearly explained in the report, with every test case justified, as well as the overall scheme? Is the output organized, easy to read and cross-referenced to the report?

6.2 Assigning marks to members of your group

Assignments done in groups will be marked by the same scheme as individual work, except that some allowance is made for the size of the teams and for the distribution of work among the team members. Groups of two will be treated more generously in assessing "achievement" than groups of three, for example.

As a first step, group assignments will be marked as a whole, resulting in a global mark for the group. Next, the global mark will be modified appropriately for individual members of the group. So that we can evaluate the work done by individuals in your group, you must tell us who did which parts of the work on the assignment. To avoid influence by peer pressure, each member of the group must submit her or his own summary of how the work was divided, including the work done by the other members as well as by the author of the summary. If you really did everything together, you can write the summary together too, but you still have to submit one copy per person. The summaries should be substantially less than a page long.

Here's how we'll use your "who did what" statements: If the report, say, is particularly good compared with the rest of the work, the person primarily concerned with writing the report will gain a little in their mark, and similarly for coding and testing.

6.3 Assessing your teamwork

Part of your project mark explicitly depends on the quality of your team's collaboration. This is separate from the correction described in the previous section for the quality of work done by each individual. For example, if you wrote the report and did it well, your project mark might be raised to account for your good individual effort; but the group's overall mark for teamwork would not be affected by the quality of the report.

It is difficult to judge teamwork, and certainly cannot be done with high precision. The mark for teamwork will probably be a letter grade, and for most groups will be the same for all group members.

Most groups will get a B or C. (Groups who fail to report on how the work was shared will get a C.) You get an A only if you meet and overcome difficulties in the group structure, or if you show noteworthy originality in your group organization. A straightforward approach such as dividing the project "vertically", with each member writing, testing, and reporting on a part of the software, is not particularly effective (and can be a sign of inability or unwillingness to really have a group plan). A more integrated approach, with work divided up according to tasks — coding for one part of the software done by one member, testing by another, and reporting by another — is better. But don't make such a split rigid. Instead, assign *primary* responsibilities and share the work of each task. This way, everyone learns something about all aspects of the project, which is important since any aspect could be tested on the final exam.

You will receive a D or worse if you encounter but do not overcome difficulties in organization.

Usually, all group members will get the same mark for teamwork, because you are jointly responsible for this aspect of the project. Only if you have had to consult the instructor for help with the group dynamics might your grade for teamwork be differentiated.

7 How to get an assignment remarked

7.1 First talk to your TA

If you disagree with or are confused about your mark on an assignment or test, then you should speak directly to your TA. If it is a very simple problem (for example, an error in adding up your mark) or something on which you can quickly come to agreement, your TA can deal with it on the spot. If, however, it is not such a simple matter, you must submit your request in writing to your TA. Fill out a Remark Request form (available on the course web site) and submit both it and your complete assignment (everything you handed in, and the marking sheet the TA gave back to you, all in the original envelope) to your TA. He or she will it to you in a subsequent tutorial.

The deadline for requesting a remark by your TA is one week after the assignment was returned in class.

7.2 Then talk to me if necessary

Only if you have taken the above steps and are not satisfied should you contact me. In that case, you must fill out a second remark request form on which you explain why you are not satisfied with the response from your TA. Bring it plus the whole assignment and the first remark request form to my office hours.

7.3 About marking standards

I try hard to give the markers detailed marking schemes that will maximize the chance of everyone marking to the same standard. And the TAs try hard to make that happen. However, some discrepancies are still inevitable.

Do not ask for a remark of your assignment because you think your tutor was tougher on your class than some other tutor(s). Discrepancies, if any, are compensated for by me at the end of term, when I decide on any cross-tutorial mark adjustments that may be appropriate. Generally, these adjustments change marks only by increasing them. Reductions are possible as well, but I try very hard to avoid them.

University of Toronto
csc228— File Structures and Data Management

Self Assessment

Instructions:

- Fill out the Your Skills and Your Work Habits sections, and then discuss them with the other members of your team. Teams members should have complementary skill sets, and compatible work habits. Think about this seriously.
- Then fill out the Your Team section. This involves swapping phone numbers etc., picking a goofy name for your team, and most importantly, **choosing a regular time for team meetings**.

Your Skills

1. software design and analysis:
   very good   good   okay   weak   poor
2. programming:
   very good   good   okay   weak   poor
3. debugging:
   very good   good   okay   weak   poor
4. designing and carrying out a testing strategy:
   very good   good   okay   weak   poor
5. writing:
   very good   good   okay   weak   poor
6. public speaking:
   very good   good   okay   weak   poor
7. time management:
   very good   good   okay   weak   poor
8. working with people:
   very good   good   okay   weak   poor
9. organizing people:
   very good   good   okay   weak   poor

Your Work Habits

1. I expect to receive a grade of ______________ in this course.
2. When working on assignments I usually start them:
   just before they are due   in the week before they are due   as soon as I get them
3. I stay up all night the night before my assignments are due:
   for all assignments   for most assignments   occasionally   never
4. I rarely turn an assignment in late.
   strongly agree   agree   neutral   disagree   strongly disagree

5. I am a night person and avoid any meeting before 11:00 a.m. if I can.
   strongly agree   agree   neutral   disagree   strongly disagree
6. I am on the St. George Campus
   (a) daily, including most weekends
   (b) daily, but not on weekends
   (c) at least four times a week
   (d) a couple of times a week
   (e) only for this class
7. I read my email
   (a) several times a day, including most evenings and weekends
   (b) several times a day, except evenings and weekends
   (c) about once a day
   (d) a few times a week
   (e) once a week
8. Other things that take up my time besides this course include: (explain each one briefly)

   work ______________________________

   family obligations ______________________________

   other (explain) ______________________________

Your team

Use your skills and habits assessments to settle on your team, and then select regular meeting times. Meeting twice a week is a *minimum*. Then record the relevant information about your one or two partners.

Team Name: ______________________________

Regular Meetings: ______________________________

Partner 1: ______________________________

email: ______________________________

Telephone Number: ______________________________

Best times to call: ______________________________

Partner 2: ______________________________

email: ______________________________

Telephone Number: ______________________________

Best times to call: ______________________________