How to Prepare and Submit Assignments

C/C++ Programming I & II

Assignments vs. Exercises

There are eight course "assignments", each consisting of several "exercises". Each exercise requires either a program, answers to questions, a diagram, or something similar. The requirements for all of these are sent to each student via email at the beginning of the course.

The "Assignment Checker"

Exercises are ONLY accepted via submission to the "assignment checker", which is an easy-to-use automated email application that is available 24/7. It analyzes (but does not grade) each submission and quickly sends back a report to help you correct many issues that would otherwise result in credit loss. Exercises may be corrected and resubmitted without penalty as many times as desired before the assignment deadline. Please follow these simple steps for every submission:

1. Submit only one exercise per email as follows:
   - The subject line must be only as specified in the requirements for that exercise.
   - Individually attach any files indicated in the requirements for that exercise – do not zip.
   - Leave the email body empty.
   - Email to one of the following addresses:

     CheckMyAssignment@MeanOldTeacher.com **(preferred)**
     CheckMyAssignment@KindOldTeacher.com (alternate)
     CheckMyAssignment@att.net (alternate)
2. **Wait for an email response** from the assignment checker. Read it and any attachments carefully before contacting me for help related to any errors or warnings.
3. Make corrections and resubmit as many times as you wish before the assignment deadline.

Please Note:

- The assignment checker always sends a response email, typically within a few minutes of receiving a submission. However, students who wait until the last day to submit their assignments may experience longer delays depending upon the backlog. If you don't receive a response within 15 minutes and have checked your "spam" mailbox, check for errors in the submission address and submit again, possibly using one of the alternate addresses. If there's still no response please send me the details of your submission(s).
- The assignment checker cannot process submissions with subject line or attachment problems and will send you a rejection response if this occurs. Such erroneous submissions must be resubmitted correctly prior to the assignment deadline or no credit will be given.
- After each assignment deadline the instructor will manually grade the newest submission of each exercise, even if it was late and there were earlier on-time submissions. Because late submissions receive significant deductions they should not be used just to correct minor issues.

Exercise Submission Example:

Assume a student whose 9-character UCSD ID is `U99990001` wishes to submit an exercise to the assignment checker. Also assume that the requirements for that exercise state that files named `C1A2E5_Test.cpp` and `C1A2E5_Driver.cpp` are required and that the subject line must be `C1A2E5_`*`ID`*, where *`ID`* represents the student's UCSD ID. The exercise submission email would then have the following parameters:

To: `CheckMyAssignment@MeanOldTeacher.com`
Subject: `C1A2E5_U99990001`
Attachments: `C1A2E5_Test.cpp C1A2E5_Driver.cpp`

Title Blocks

An appropriate "Title Block" must be placed first in EVERY FILE submitted to the assignment checker. Title blocks are commonly used in professional programming environments to document the company, project, developers, development tools, file contents, revisions, and many other things.

DO NOT COPY OR RESTATE MY EXERCISE INSTRUCTIONS IN A TITLE BLOCK OR ANYWHERE ELSE.

"Title Block" for source code files (.c, .cpp, .h, etc.)

C and C++ commenting styles are equally acceptable:

```
//
// Your name and 9-character UCSD student ID (e.g., U12345678)
// Your email address
// Name of this course
// 6-digit course section ID and instructor name
// Date
// Name of this file, such as C1A3E5_main.c, C2A4E7_Test.h, etc.
// Your operating system, such as Win10, macOS 10.14, UNIX, LINUX, etc.
// Your compiler & version, such as Visual C++ 15.0, Xcode 10.0, etc.
//
// Briefly describe the relevant contents of this file, such as names/purposes
// of any functions or macros that are defined.  Do not describe prototypes.
//
```

"Title Block" for non-source code files (.txt, .pdf, etc.)

There are no comment delimiters and no mention of compilers or operating systems since these are only relevant in source code files:

```
Your name and 9-character UCSD student ID (e.g., U12345678)
Your email address
Name of this course
6-digit course section ID and instructor name
Date
Name of this file, such as C1A3E0_Quiz.txt, C2A5E3_StateDiagram.pdf, etc.
Indicate the major contents of this file, such as quiz answers, state diagrams, etc.
```

Magic Numbers

A "magic number" is any numeric literal, character literal, or string literal used directly in code or comment. Although magic numbers are acceptable in a few specific cases their misuse makes programs cryptic and difficult to maintain. It is usually more appropriate to associate meaningful names with literals and use those names in the code instead of the literals themselves. However, never choose names that convey actual values, such as ZERO, TEN, LETTER_A, STRING_HELLO, etc., since these are also considered magic numbers. To associate meaningful names with literals use #define directives in C, **const**-qualified variables in C++, and enumerated types in either language. Use the following general guidelines along with common sense when in doubt:

Typical acceptable "magic number" usage:

1. 0 or 1 for array index or loop start/end values
2. values in initializer lists
3. it represents nothing other than the value itself

Typical unacceptable "magic number" usage:

1. its purpose would not be immediately obvious to a programmer unfamiliar with the code
2. its value might need to be changed in a future code revision
3. in comments or print statements

Hard Tabs

Never use "hard" tab characters when writing your code. A "hard" tab is a special character that many text editors produce by default whenever the "Tab" keyboard key is pressed. This character is interpreted differently by different implementations and can produce inconsistent display and print results. Before any code is written your editor should be configured to insert spaces instead of tabs to avoid these problems. See the appropriate version of the course document titled "Using the Compiler's IDE..." if you don't know how to configure this.

Representing the Values of Characters

To specify the value the system uses to represent a particular character, such as the value used to represent the letter **A** or the digit **0**, never actually write that value in your code unless there's no other option. Instead, use a character literal. For example, in the ASCII character set the value of the letter **A** is **65** and the value of the digit **0** is **48**. Thus, to represent the values of these characters in your code you should use **'A'** and **'0'** instead of **65** and **48**, respectively.

Validating User Input

Never attempt to validate the correctness of user input unless an exercise explicitly requires it. While this is a must in "real life" applications the code required is often complex and involves implementation-dependent considerations beyond the scope of what is expected or wanted in this course.

Unwanted User Prompts

Never prompt a user for anything not specifically called out in the exercise requirements. Asking the user if he/she would like to re-run a program is fine for many "real life" programs but in this course will cause the testing feature of the assignment checker to generate a "Program Hung" failure message. Please re-run programs manually rather than with a program loop unless directed otherwise.

External (Global) Variables

An external variable (also called a global variable) is any variable declared outside a function. Except for **const**-qualified external variables in C++, which are a recommended substitute for object-like macros, external variables make code cryptic, error prone, difficult to maintain, and not thread-safe. They are never allowed in any exercise in this course unless explicitly stated otherwise. This avoidance is also a good policy in "real life" programs.

How to Debug a Program

Debugging a program is a step-by-step logical process that requires a full understanding of:

1. the program requirements
2. how the algorithm you've used meets those requirements
3. the specific purpose of each line of code and the details of how it accomplishes that purpose

Attempting to debug a program without this understanding is a pointless exercise in frustration and futility.

Regardless of the tools used for debugging, the overall procedure is the same and consists of systematically stepping through code and examining the values of relevant variables and other expressions to determine if they are as expected. When an errant value is found the cause is then determined and fixed. For programs that process input from the user or other source the input being used when the error occurred should also be used during debugging.

The easiest-to-use debugging tools are those built into the IDEs themselves. They permit programmers to step through source code one statement at a time to examine program flow and check values. They also allow "breakpoints" to be set, which permit the program to run normally until a breakpoint is reached, at which time it pauses for further examination. Debuggers offer many additional features but these are sufficient in most cases.

...continued on the next page

The general procedure for using any IDE debugger is described below. The specific details for three popular IDEs are provided in the appropriate version of the "Using the Compiler's IDE..." documents on the course website.

1. Step to the first statement in the code and check the values of any relevant variables. Are they as expected? If not, why not?
2. Continue to step through the code, making sure that the desired statements are being executed and relevant values are as they should be.
3. If there is a specific point in the code you are interested in inspecting and don't want to waste time stepping to it, set a breakpoint at that point and run to that breakpoint.