

How to Prepare and Submit Assignments

C/C++ Programming I & II

Assignments vs. Exercises

Each weekly homework "assignment" consists of several "exercises" that require a program, answers to questions, a diagram, or something similar. All assignments are customized for and sent to each student via email or a download link at the beginning of the course.

The "Assignment Checker"

Exercises are ONLY accepted for credit 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 understand and 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, but NO credit will be given for late submissions unless the instructor has pre-approved an extension. Please follow these simple steps for every submission:

1. Submit only one exercise per email as follows:

- The subject line must be exactly as specified in the exercise requirements.
- Specified files must be individually attached (not zipped).
- The email body must be empty.
- Email to one of the following addresses:

CheckMyAssignment@MeanOldTeacher.com	(preferred)
CheckMyAssignment@KindOldTeacher.com	(alternate)
CheckMyAssignment@att.net	(alternate)
CheckMyAssignment@gmail.com	(alternate)
CheckMyAssignment@NiceOldTeacher.com	(alternate)

2. **Always wait for an email response** from the assignment checker in case you accidentally made a typo in the email address, subject line, or attachment(s). Carefully read the response and any attached or linked files before contacting me for help.

3. Make corrections and resubmit as many times as you wish before the assignment deadline.

Please Note:

- Although the assignment checker always sends a response email, typically within a few minutes of receiving a submission, students who submit their work just before the deadline may experience longer delays depending upon the backlog. If you do not 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 is still no response, please send me the details of your submission(s).
- The assignment checker cannot process submissions that have 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 that was submitted before that deadline. Work submitted after the deadline will receive NO credit unless the instructor has pre-approved an extension.

...See the next page for an exercise submission example...

Assignment Checker Exercise Submission Example

The assignment checker produces one or more files for you for every properly submitted exercise, and your subject line determines whether they will be provided as email attachments or via a download link. Although attachments are simpler, some email systems are very picky about them and either delay or block emails containing them. Select which works best for you as follows:

Assume a student whose 9-character "Student ID" is U98765432 is enrolled in a course whose 6-character "Section ID" is 123456 and an exercise requires files named C1A2E5_Test.cpp and C1A2E5_Driver.cpp. The email submission to the assignment checker would then have the following parameters, with the email body left empty:

To: CheckMyAssignment@MeanOldTeacher.com

Subject (files returned as attachments): C1A2E5_123456_U98765432

Subject (files returned via download link): C1A2E5_123456_U98765432 -cloud

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 RESTATE OR PARAPHRASE MY EXERCISE INSTRUCTIONS IN A TITLE BLOCK OR ANYWHERE ELSE.

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

C and C++ style block commenting syntaxes are legal in both languages, but C++ style is preferred:

```
//  
// 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 12, UNIX, LINUX, etc.  
// Your compiler & version, such as Visual C++ 17.0, Xcode 20.3, etc.  
//  
// Briefly describe the relevant contents of this file, such as names and purposes  
// of any functions or macros that are defined. Do not describe prototypes.  
//
```



For readability, leave an empty comment line before your contents description.

"Non-Code" Title Block for quiz answers, drawings, tables, etc. (.txt, .pdf, etc.)

Note: There are no comment delimiters and no mention of compilers or operating systems in non-code title blocks 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, etc., since these are also considered magic numbers. To associate meaningful names with literals, use #define directives in C, **const**-qualified variables in C++, or if appropriate, enumerated types in either language. Use the following general guidelines along with common sense or instructor help 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. the number represents nothing beyond 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 do not 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 literal value in your code unless there is 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 (ask) the user for anything not specifically called out in the exercise requirements. For example, asking the user if he/she would like to rerun 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. Rerun 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.