CS 2212 Programming Style Guide 2021

Please read and follow these instructions exactly.

1. Learn to use a good IDE with a quality debugger like Xcode for Mac using GNU C/C++ or Eclipse for PCs using GNU C/C++. I have detailed instruction and a video of the installation process of Eclipse in Canvas. Take the time now to figure this out, come to office hours when you get stuck or see one of the recommended TAs in the CLR or the class TA. It is expected that every computing professional can figure out how to install, configure, and debug a software installation and use software. Getting the IDE to work might be harder on your PC because we are being picky about using a standard version of C/C++ (GNU), not a PC specific version (Cross or MinGW). On Mac, your challenge is finding your files because Steve Jobs hides them from you in a central location (which is right in a professional setting but a challenge for a novice programmer). You each have your challenges regarding working with an IDE. Putting off figuring this out adds to your learning curve later in your degree program, I do not recommend it.
2. Each submitted source code or output file should include your **name at the top and the name of the program file in the header**. The function of the program and version history should be described at the top of the source code file that contains main(), which is usually the controller and in the header of each class. This block is more elaborate. For example:

```
/*  ======================================================================
 *  Programming Style Guide                                   styleGuide.cpp
 *  Author: LPage
 *  Version: Created on Sep 7, 2017, updated 8/22/18
 *  Copyright: Best practices for programmers based on industry
 *    and education standards.  This is for educational purposes only.
 *    All rights reserved, use with my express written permission only.
 *  Description: This program … I’ll leave this for you
 *   to complete. Hint: Check the program goals.
 *  Written in C/C++, Ansi-style.
 *  ======================================================================
 */
```

3. Please, **never** submit your code using any editor application like WordPad, Google Docs or Microsoft word. Use a text editor, there is one in your IDE. If you use another editor, make sure you save your code as plain text (important for Mac users), I will refer to this as text. I want your **source code as a .c or .cpp file**. ***Do not include and compiled or linked code with your folder, it will be rejected by the system as an attempt to send a virus.***
4. Programming assignments should be submitted as requested by your instructor by midnight or so on the date of the deadline. Late submissions, you have about a few hours grace period, are subject to losing points. I only take off up to a set amount (less than 20%) for being late, after that I feel being later than two weeks is bad enough that you do not need any further penalty. Try to avoid this, you will feel lost in class.
5. Most programs are submitted via **repl.it or a similar system** or in Canvas.
    a. For the last few assignments I ask you to submit your assignment to Canvas. Canvas will send them to me as zipped files so simply load each file into canvas separately. I will ONLY Accept source code, header, and text files.

1. Source code files end in .cpp, for example main.cpp or pets.cpp
2. Header files (if relevant) end in .h or .hpp, for example pets.h
3. Text files end with .txt, for example output.txt

ii. For at least one program you will submit a video showing your program. For this assignment, you need to submit a link. **Make sure you put the password for a Zoom or Google Drive link in the comments when you upload the link.** If you have a YouTube account, it is possible to have a private playlist to share your video.

iii. There is a deduction for not following instructions.

6. Good style (computing professional standards), clarity, and good use of C/C++ are needed for a grade better than 85% on a programming project.

   a. **The entire line of code must be visible on the IDE screen without needing to scroll right (120 characters for my screen)**. In XCode or Eclipse you can set a ruler line in preferences. If you cannot do this, simply make the bottom and top comment include dashes to 120 characters, so you know where to stop.

   b. Each submitted source code or output file should include your **name at the top and the name of the program file in the header**. Block comments start with /* and end with */. See item 2, above.

   c. **Include a block comment for each function including main()** above the function describing what it does, inputs/parameters and outputs/return. Note: main() is a function and needs a block to describe arguments and return. A good example of what to include is in our references such as https://en.cppreference.com/w/cpp/string/basic_string/to_string

```
/*  =======================================================
 *  print()
 *  [Author: if different from the one listed at the top]
 *  Description: This function … I’ll leave this for you
 *   to complete. Hint: Check the program requirements.
 *  Parameters:
 *              mass - the mass in grams, double.
 *              volume – the volume in cubic meters, double.
 *              moles – amount of matter, double
 *  Return Value: a double with the pressure in Kilopascals.
 *  Exceptions: throws type mismatch and division by zero.
 *  Notes: Anything you think the user needs to know.
 *  ======================================================= */
```

   d. **Necessary in-line comments** must be provided, using good naming will minimize the need for comments. If we must hunt around to figure out what you did, it means your code will be hard to maintain. Your TA or instructor looking at your program should be able to understand your code with no other information and without having to scroll back to the variable declarations. If you feel like you have to explain what a variable is, you should try to change the name so it is self-explanatory. E.g. row not i, col or column, not k. Mass not num.

e. Remove notes to yourself. If you have comments as notes to yourself on how things work, or things you need to do (e.g. //TODO comments), make copy for submission, and **remove** those comments from it. Leave only necessary comments to clarify functionality of the program and block comments as noted above.

f. Naming functions and variables in a logical manner makes code easier to understand.

    i. **Use camelCase** for variables and function names – avoid underscores and very long names.

    ii. Put the declaration of variables used throughout your program in a block at the beginning of your program. Scattering them all over your program makes the declarations harder to find. Your assignments are short programs in the professional world. Do not be confused by style guides for specific settings such as the IBM or Google Style Guides. *Later in your career, you will use the style of the organization you work for, it takes no time at all to adjust your style to a company standard.*

    iii. If you have a **variable whose scope is within a loop, or if or switch statement**, put it inside that structure at the top.

    iv. There are exceptions but think about updating a program and making it easy to work with.

g. Use constants declared with **#define** for universal values such as Pi or acceleration due to gravity, maximum number of items in an array, and file names. Use **ALL CAPS for constants**. These go at the top of your program after the #include and function prototypes. This makes it easy to change once and have it work throughout your program.

h. Your code should work well and be efficient. Do not be tricky – it makes code hard to follow. Using coding efficiencies of the language is not being tricky, but if you use non-standard functionality, it will cause portability problems in your code.

    Kernighan's lever

    “Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?”

    — Brian Kernighan, The Elements of Programming Style, 2nd edition, chapter 2

i. **Remove all extra print statements used for debugging** from the code you submit for grading. Do not comment them out and leave them, its clutter.

j. **Remove all system generated comments and empty stubs** from your code. Except if you are handing in an incomplete program or if I explicitly told you to create the stub. Remove all of these: // **TODO** Auto-generated stub

7. **Your Programming Environment**
    a. Please download and use either Xcode (Macintosh) or Eclipse (Windows and Linux).
    b. Eclipse users need to get the the Gnu C++ compiler (g++) not the stripped-down minGW or PC specific Cross compiler. Windows people can use Cygwin to get it.
    c. Everything you turn in must be written entirely in standard C++ 11, C++14 or C++17 (preferred) standard, and I must be able to compile it on my machine without modification. C++20 standard is not fully implemented in most compilers. https://en.cppreference.com/w/cpp/20 but it should be done this year.

       DO NOT include any Windows pathnames. Microsoft-only header files like `conio.h` and `windows.h` or precompiled headers like `bits/stdc++.h`! This is bad coding!

       To make your code portable ONLY use the CPP Standard Library documented at https://en.cppreference.com/w/.

    d. The ~~Visual Studio~~ IDE puts inappropriate things into your code and does not give good error comments. Students who use it inevitably have more trouble than others. ~~CodeBlocks~~ does not supply the minimum level of support and diagnostics that you need. Do not waste your time on these IDEs.
8. **After the midterm** you will be asked to write your own **test plans** and submit it and your test results in a file. Here is some guidance.
    a. There are **notes on testing** in L5 Program Architecture Basics. Think of white box (cases to make every line of code and path through the program execute), black box (based on every requirement) and security cases (look for edge cases and security issues) for your test plan.
    b. For example if your program that starts by reading data from a file into a collection, has a menu that includes print data, add data (from keyboard entry), and quit, this would be some test cases you might include, the specifics of the requirements and your code will help you figure out what else is needed for your program:
        i. Black Box Test case: Run program with data file missing and quit. Expected outcome, error message to console (program does not crash) and continue to menu. Empty file created when you quit, normal program ending message to console.
        ii. Black Box Test Case: Run program with empty file and quit. Expected outcome, message to console that file is empty (program does not crash) and continue to menu, normal program ending message to console. File saved is still empty.
        iii. Black Box Test Case: Run program add one payment and quit. Expected outcome, data of new item is in file, normal program ending message to console.
        iv. Black Box Test Case: Run program and quit. Expected outcome: File data is unchanged, normal program ending message to console.
        v. Black Box Test Case: Run program, add two items, print data and quit. Expected outcome, all three items print to console, normal program ending message to console. File now contains all three items.
        vi. **White Box Test Cases:** *There will be several specific data entries or menu sequences so that all the menu items and paths in your program are used.*
        vii. Security Test Case: Enter invalid menu item. Expected outcome: Error message and return to menu (program should not crash).

viii. Security Test Case – edge case: Invalid data, maxvalue +1.
ix. Security Test Case – edge case: Invalid data, maxvalue.
x. Security Test Case – edge case: Invalid data, maxvalue - 1.
xi. **Security Test Cases –** there should be several test cases with a mix of invalid data and menu choices to see what slips by.

c. For each test, copy and paste the console output to a text file. I usually label each one with a, easy to see separator:

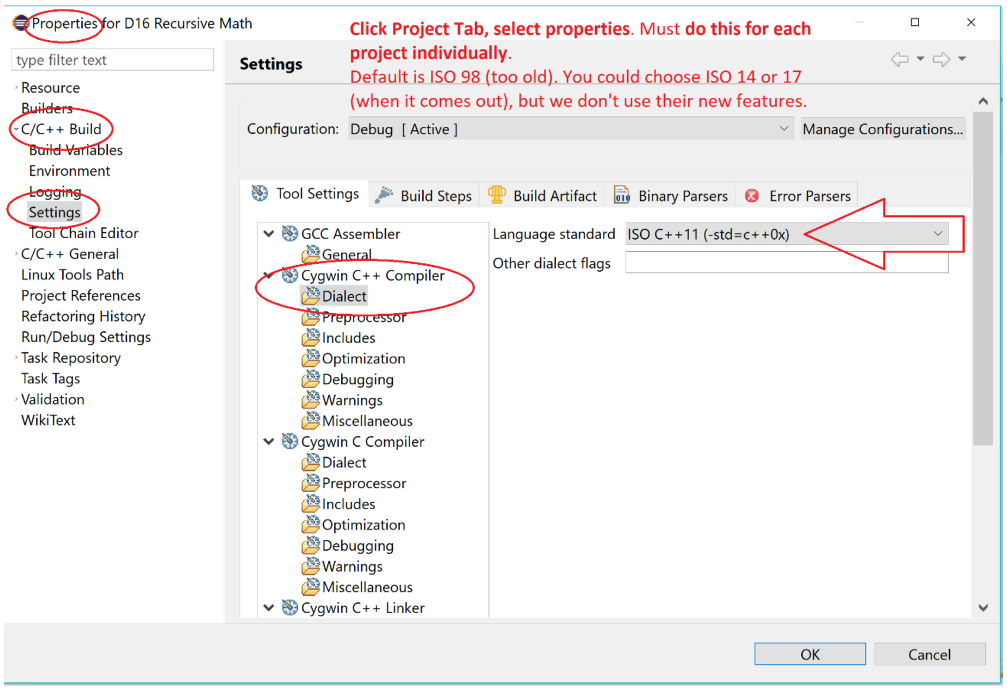
```
+++++++  Run program with data file missing and quit. +++++
Console output goes here for this test case.

+++++++++++  Run program with empty file and quit. ++++++
Console output goes here for this test case.
```

How to Create and Run a Programming Project in Eclipse:

1. **In Eclipse (for Xcode – look online for help)**, to create a new C Project or a C++ Project.
    a. File – New- C Project or C++ Project
    b. Fill in your name for this project in the dialog box, you need a new project for each assignment you do.
    c. Create a naming system so you can find your code easily. The executable file is named after the project.
2. Create a new C source file or C++ Source File:
    a. File – New – C Source File (NOT Header File or Source Folder)

    Or

    File – New – C++ Source File (NOT Header File or Source Folder)
    b. In the dialog box:
        i. Make sure the Source File is in your project
        ii. Give the C source code file the name trySomething.c for C++ use trySomething.cpp
        iii. Pick the Template, 'Default C source template' or 'Default C++ source template' from the menu
3. Write your program, checking for errors as you go. You can use an example from the notes to try at first. Make small changes. Compile and debug your code at the end of each idea (item in the requirements or 10 lines of code. The more you test and debug, the less time it takes to correct an error. Do you want 50 candidates or 5 for the source of a mistake?
4. You should add no more than a few lines before you compile and check that the program is doing what you want. You can add print statements to check the contents of variable (Xcode shows you automatically). Save often!.
5. There are exercises at the end of each chapter in your book for extra practice, this is how you get really good at programming.
6. Make sure you are using the dialect C or C++ ISO Standard 11, or higher (17 is preferred).
    a. **Right click** on your project name in the Eclipse Project Explorer, then click on **Properties** at the bottom of the pop-up menu.
    b. In the dialog box:


Properties for D16 Recursive Math

Click Project Tab, select properties. Must do this for each project individually.
Default is ISO 98 (too old). You could choose ISO 14 or 17 (when it comes out), but we don't use their new features.

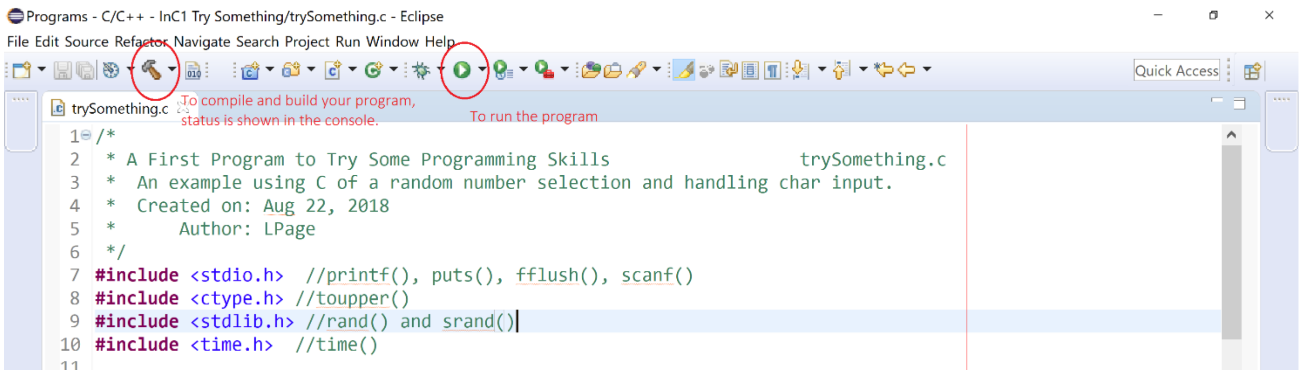
Settings

Configuration: Debug [Active]

Language standard: ISO C++11 (-std=c++0x)
Other dialect flags:

7. On your own device, compile (Build) and run the program trySomething.c or trySomething.cpp on your system.

For Eclipse Neon:


Programs - C/C++ - InC1 Try Something/trySomething.c - Eclipse

To compile and build your program, status is shown in the console.

To run the program

```
/*
 * A First Program to Try Some Programming Skills                trySomething.c
 *  An example using C of a random number selection and handling char input.
 *  Created on: Aug 22, 2018
 *      Author: LPage
 */
#include <stdio.h>  //printf(), puts(), fflush(), scanf()
#include <ctype.h> //toupper()
#include <stdlib.h> //rand() and srand()
#include <time.h>  //time()
```

File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
trySomething.c

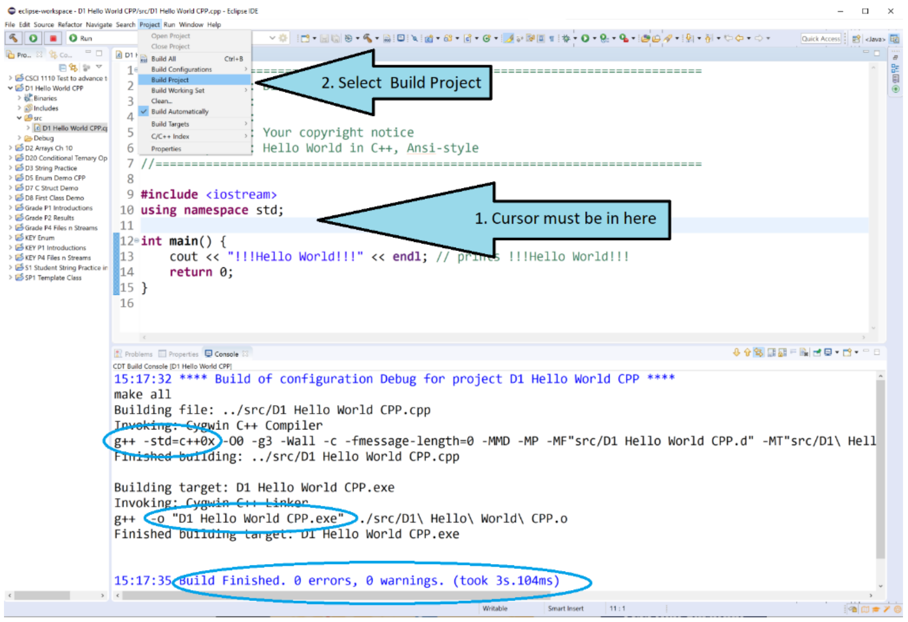
For **versions newer than Eclipse Neon**, you must **configure the build and launch** before you can build or run your program.

First you must Build the program (compile), this creates the .exe file in the Debug folder. To do this is to **put your cursor in the program you want to build**, **select Project** from the top menu and **select Build Project** (if it is greyed out, your cursor in not in a program). After the first time, it should build your program correctly using the hammer icon or shortcuts until you change the configuration.

The **console output during the build** will show you the dialect that was used to build your executable and any errors. **Make sure the correct compiler (Cygwin C++) and**

dialect (standard of C++) is being used, look at the console output. The dialect code is after the -std=

e.g. -std=c++0x would be C++ standard 11. If your program used an old standard, you would have syntax errors on any new features of C++. See the instructions on choosing the dialect in item 6 above.


eclipse-workspace - D1 Hello World CPP/src/D1 Hello World CPP.cpp - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Open Project
Close Project
Build All Ctrl+B
Build Configurations
Build Project
Build Working Set
Clean...
Build Automatically
Build Targets
C/C++ Index
Properties
2. Select Build Project
1. Cursor must be in here
Problems Properties Console
CDT Build Console [D1 Hello World CPP]
Writable Smart Insert 11 : 1

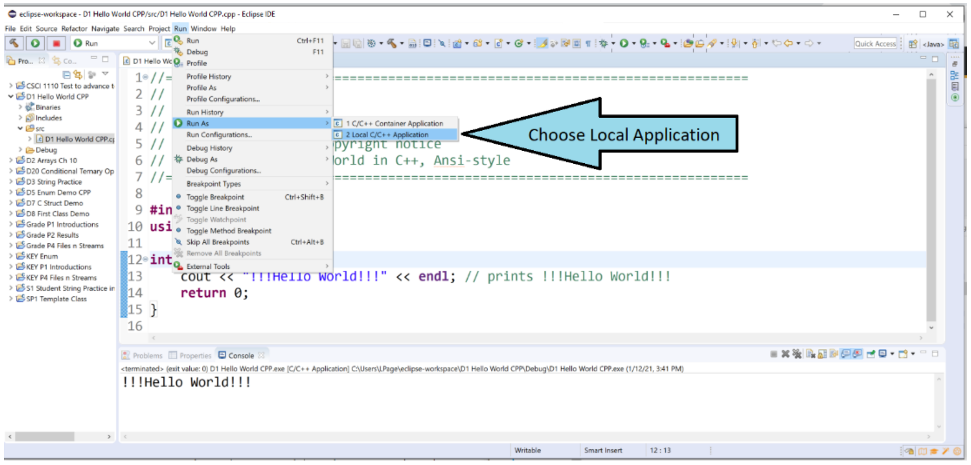
```
//============================================================================
// Your copyright notice
// Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
using namespace std;

int main() {
    cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
    return 0;
}
```

```
15:17:32 **** Build of configuration Debug for project D1 Hello World CPP ****
make all
Building file: ../src/D1 Hello World CPP.cpp
Invoking: Cygwin C++ Compiler
g++ -std=c++0x -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/D1 Hello World CPP.d" -MT"src/D1\ Hell
Finished building: ../src/D1 Hello World CPP.cpp

Building target: D1 Hello World CPP.exe
Invoking: Cygwin C++ Linker
g++ -o "D1 Hello World CPP.exe" ./src/D1\ Hello\ World\ CPP.o
Finished building target: D1 Hello World CPP.exe


15:17:35 Build Finished. 0 errors, 0 warnings. (took 3s.104ms)
```

Resolve all errors and make sure your build finished. Note the name of the executable file, this should be program you see next to the run icon on the left. If not, it did not build successfully or is not building your program, go back and do the step above.

After you have built your program successfully, you can run it (launch the executable). To do this, you should be able to just click any of the run icons (green play symbol ▶).

If that does not work, you need to select how to run your application. With the cursor in your program, select **Run** from the top menu, and **Run As**, from the sub-menu choose **Local C/C++ Application**.


eclipse-workspace - D1 Hello World CPP/src/D1 Hello World CPP.cpp - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Run Ctrl+F11
Debug F11
Profile
Profile History
Profile As
Profile Configurations...
Run History
Run As
Run Configurations...
Debug History
Debug As
Debug Configurations...
Breakpoint Types
Toggle Breakpoint Ctrl+Shift+B
Toggle Line Breakpoint
Toggle Watchpoint
Toggle Method Breakpoint
Skip All Breakpoints Ctrl+Alt+B
Remove All Breakpoints
External Tools
1 C/C++ Container Application
2 Local C/C++ Application
Choose Local Application
Problems Properties Console
Writable Smart Insert 12 : 13

```
<terminated> (exit value: 0) D1 Hello World CPP.exe [C/C++ Application] C:\Users\LPage\eclipse-workspace\D1 Hello World CPP\Debug\D1 Hello World CPP.exe (1/12/21, 3:41 PM)
!!!Hello World!!!
```

8. Try it out! Does it work? If not, look at the errors and see if you can figure out what's wrong. Stuck for more than an hour? See the class TA or my recommended CLR tutors (the tutors I recommend are Computer Science majors who have done well in the fundamental courses and know the content I am teaching), or see me at office hours.

Part 2 – Use in class or textbook examples to understand the language.

1) Run the program several times, figure out what it does line by line.
    a) You should look things up online, so you are sure you know what each function does and what the output is.
    b) Add print statements to test your ideas.
    c) Make sure YOU know; you will need to combine these ideas in your programming projects.
    d) You should be able to explain everything in your program in detail.