

Lab 1 - Intro to C++ and VS Code

Lab Topics

1. Setup a development environment.
2. Creating a simple C++ program with VS Code.
3. Explore compile time errors.
4. Configure code formatting.
5. Submit code to Canvas.
6. Explore Live Share feature for cooperatively working on code.

Directions for Labs

- Submitting your completed lab file online to CourSys (not Canvas).
- If you would like to attend an in-person lab section, you are welcome to come to any (or more than one) lab section. There is no need to attend the section you are enrolled in. There is no need to attend any lab sections: there is no attendance taken. You can complete the lab on your own time or own computer.
- While completing these labs, you are encouraged to help your classmates and receive as much help as you like. Assignments, however, are individual work.
You may not work on assignments if you are in the CSIL lab during CMPT 130 tutorial times.

1. Setup a Development Environment

Before we can write some C++, we'll need to setup our development environment: the programs we need to write, compile, run and debug!

- **The course website has guides to help you set up an environment.**
There are guides for using the CSIL computers, or using your own computer (Windows, MacOS, or Linux). You can even use a remote connection from a tablet!
- Note that the general SFU computer labs (not CS's CSIL labs in: SRYE 3024, SRYE 4013, SRYE 4024, and in Burnaby's Applied Science Building) may not have VS Code (the software we need for this course).

2. Hello World

This section shows how to use the VS Code IDE (Integrated Development Environment) to create and run a very simple program. The guides on the course website list the steps to creating a folder and making a new `.cpp` file. You should follow those steps as they can be specific to the environment you are using. Here is a broad summary of what you'll need to do:

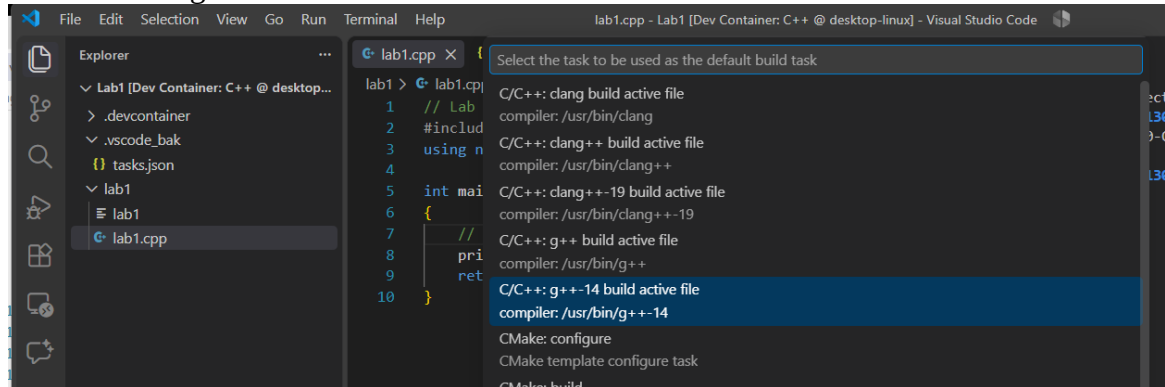
1. Create a folder for all your CMPT 130 work this semester.
 - Name the folder `cmpt130`
 - Do not put a space in the name of your folders or files because it can mess up some tools.
2. Launch VS Code and open your `cmpt130` folder.
3. Create a new folder inside the project for `lab1`:
 - On the left hand side of VS Code, click on the *Explorer* option to see the files in your folder.
 - You can click the *New Folder* button at the top of the explorer view, or you can right-click in an open area of the explorer window.
 - Name your new folder `lab1`
 - I suggest you create a new folder for each activity (lab or assignment, ...) and for different sections of the lecture notes.
4. Inside the `lab1/` folder create a file named `lab1.cpp`.
 - Put the following code into `lab1.cpp`. Type it in! It's a good way to learn a new language (over copy-and-paste).
5. **Save** your file.

```
// Lab 1 Program
#include <print>
using namespace std;

int main()
{
    // Some simple output
    println("Hello lab world!");
    return 0;
}
```

6. Configure the build task with C++ standard c++23

- **Go to Terminal > Configure Default Build Task**
- **Select C/C++ g++-14 build active file**



- Open the file that this created, which is under: `.vscode/tasks.json`
- Under “args”, add the following line:
`"-std=c++23",`

Full file contents may be:

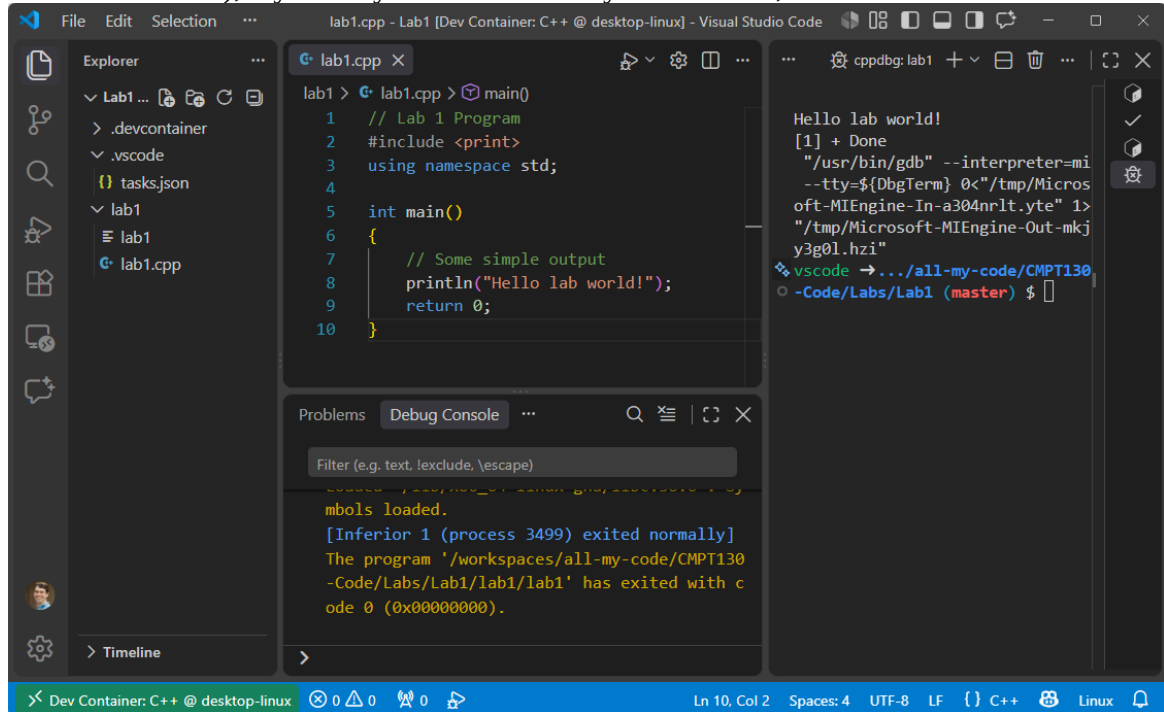
```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "cppbuild",
      "label": "C/C++: g++-14 build active file",
      "command": "/usr/bin/g++-14",
      "args": [
        "-std=c++23",
        "-fdiagnostics-color=always",
        "-g",
        "${file}",
        "-o",
        "${fileDirname}/${fileBasenameNoExtension}"
      ],
      "options": {
        "cwd": "${fileDirname}"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "detail": "compiler: /usr/bin/g++-14"
    }
  ]
}
```

- Close `task.json` and return to your `lab1.cpp` file.

7. Build and Run your program:

On the menu, select **Run > Start Debugging**. Or, press **F5**; This will save you time!

- Make sure there are no errors in the **Terminal** window. If there are, click or CTRL+click on the error and correct the problem; see next section for more.
- A successful compile will look as follows (output shown on **Terminal** tab on the right in the screenshot); layout of your VS Code may be different; that's OK!



The screenshot shows the Visual Studio Code interface with a C++ file named `lab1.cpp` open. The code is as follows:

```
1 // Lab 1 Program
2 #include <print>
3 using namespace std;
4
5 int main()
6 {
7     // Some simple output
8     println("Hello lab world!");
9     return 0;
10 }
```

The **Terminal** window on the right shows the output of the program:

```
Hello lab world!
[1] + Done
"/usr/bin/gdb" --interpreter=mi
--tty=${DbgTerm} 0<"/tmp/Micros
oft-MIEngine-In-a304nr1t.yte" 1>
"/tmp/Microsoft-MIEngine-Out-mk
y3g0l.hzi"
vscode ->.../all-my-code/CMPT130
-Code/Labs/Lab1 (master) $
```

The **Debug Console** window at the bottom shows the following output:

```
Filter (e.g. text, \exclude, \escape)
mbols loaded.
[Inferior 1 (process 3499) exited normally]
The program '/workspaces/all-my-code/CMPT130
-Code/Labs/Lab1/lab1/lab1' has exited with c
ode 0 (0x00000000).
```

- See videos on course website to view the full process.
8. Add some more output statements (`println`) to have your program tell your favourite knock-knock joke. Compile and run your program.
 9. Congratulations! You've got your C++ program running!

3. Read from Keyboard

1. Modify your `lab1.cpp` file to be the following (adds reading from the keyboard). I suggest typing it vs copy-paste so that you experience writing the code.

```
// Lab 1 Program
#include <iostream>
#include <print>
using namespace std;

int main()
{
    // Some simple output to the screen.
    println("Hello lab world!");

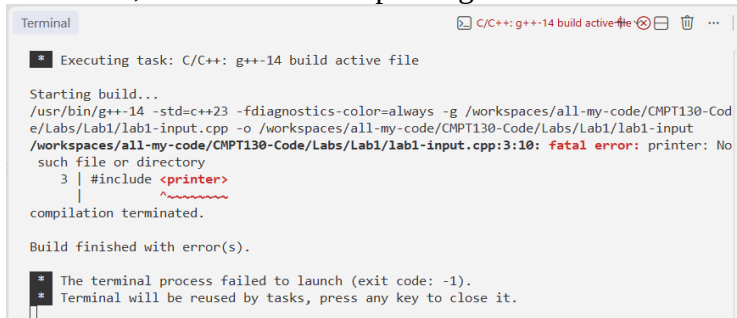
    // Read value from keyboard
    int favNum = 0;
    print("What is your favourite number? ");
    cin >> favNum;
    println("Wow! {} is a great choice!", favNum);

    return 0;
}
```

2. Re-run your program by pressing F5. Did it work?
 - If it worked, move on to the next section. Look at the Terminal tab for the output and keyboard prompts (42 was user input, so it's bold/italics).

```
Hello lab world!
What is your favourite number? 42
Wow! 42 is a great choice!
[1] + Done                                     "/usr/bin/gdb" --interpreter=mi --
tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-u846cgde.13i"
1>"/tmp/Microsoft-MIEngine-Out-t1ns1d8b.tys"
```

- Text shown in purple is generated by the operating system when our program finishes. You may ignore it.
- If it failed, read on. Your output might look like the following:



```
Terminal
C/C++: g++-14 build active file

Executing task: C/C++: g++-14 build active file

Starting build...
/usr/bin/g++-14 -std=c++23 -fdiagnostics-color=always -g /workspaces/all-my-code/CMPT130-Code/Labs/Lab1/lab1-input.cpp -o /workspaces/all-my-code/CMPT130-Code/Labs/Lab1/lab1-input
/workspaces/all-my-code/CMPT130-Code/Labs/Lab1/lab1-input.cpp:3:10: fatal error: printer: No
such file or directory
3 | #include <printer>
  |           ~~~~~~
compilation terminated.

Build finished with error(s).

The terminal process failed to launch (exit code: -1).
Terminal will be reused by tasks, press any key to close it.
```

- Check the output for indications on what went wrong. Correct the problems in your code.
 - Also, make sure you are not trying to “run” one of the .json files: be in a .cpp file before running.
 - If the error talks about “error: ‘println’ was not declared in this scope, did you mean printf?” then double check:
 1. Did you #include <print>
 2. Did you correctly add to the `"-std=c++23"`, tasks.json?
3. Above we ran the program using the built-in run (or debug) feature. You can also use a terminal where you can type in commands to directly run the same program!
- **Compile** your program via the menu:
Terminal → Run Build Task... (Ctrl+Shift+B)
This generates the executable file without running it.
 - **Run** the executable manually in the integrated terminal:
Terminal > New Terminal to launch a new text terminal.
The terminal will start in the root directory of your project; change to the folder of your .cpp and executable file by typing the command:
`cd lab1`

Run your program (has same name as your .cpp file):
`./lab1`

You should see:



```
vscode → /workspaces/cmpt130 $ cd lab1/
vscode → /workspaces/cmpt130/lab1 $ ./lab1
Hello lab world!
What is your favourite number? 42
Wow! 42 is a great choice!
vscode → /workspaces/cmpt130/lab1 $
```

4. When coding/debugging, after you make a change to your code, you’ll need repeat the previous two compile/run steps listed above:
- Compile: press Ctrl + Shift + B
 - Run either using:
debugger: F5
terminal: `./lab1`
 - Note that if you just use F5, it will try and build your project for you and then run it. However, if there are any compile time errors the build will fail, but it may then run a previous version of your application, which is very confusing! So, build first.

5. Suggested terminal commands:

- `ls` List files / directories in current directory
- `cd xyz` Change to directory xyz
- `cd ..` Change to the parent directory
- `pwd` Display current path
- `Ctrl + c` Cancel current program
- Up key Repeats previous command

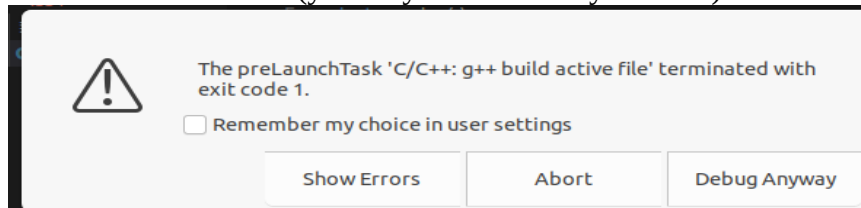
6. Understanding question

In your lab's .cpp file, type the answer to all of the Understanding Questions as a comment. It may be useful to copy-and-paste the question in as well for future reference.

- When you make changes to your code, what two steps do you need to do in order to see what your program does?

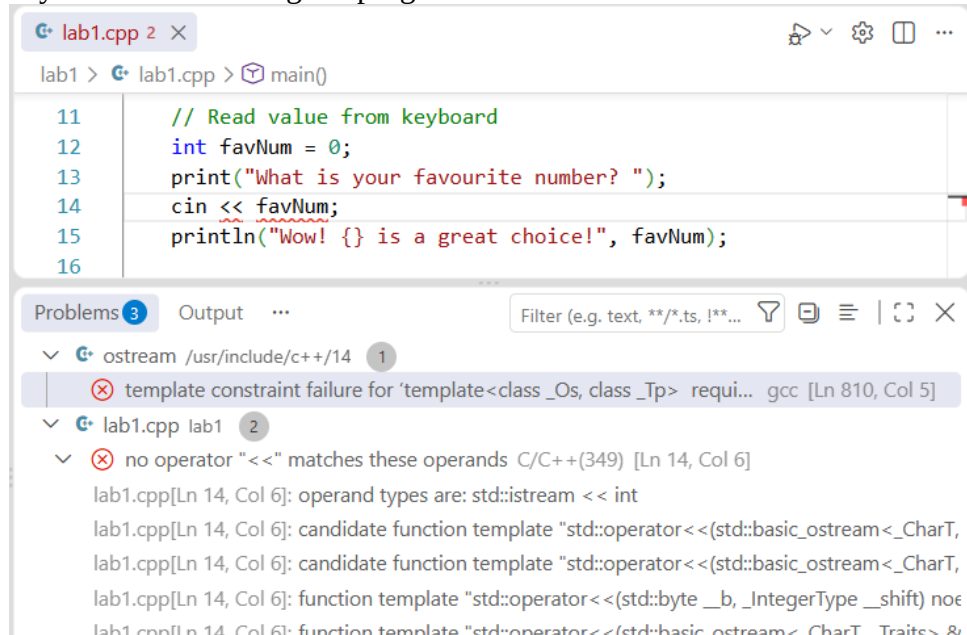
4. Exploring Compile Time Errors

1. Add a syntax error to your code (for example, remove a ';' from the end of a statement).
 - Notice the editor shows an underline on the statement(s) in the file. If you mouse over the red-underline, it will show a description of the error.
2. Try to run the program (F5).
 - You may see an error message box pop up stating the build activity failed. Click "**Show Errors**" (you may set it to always do this).



- Click on the error message to jump straight to the line of code which has the problem.
 - On some systems, VS Code may just run an old version of the application's executable! This is very confusing, so I recommend manually running a build after each change (CTRL+Shift+B), before running the debugger.
3. Correct the error. Recompile and rerun your program and the error should disappear.

4. Sometimes error messages are hard to figure out.
 - Change the `cin` line in your program to use `<<` (stream insertion operator instead of the correct `>>` (stream extraction operator).
 - Notice that an error is displayed in the editor.
 - Try to build and debug the program and see the errors:



Notice that multiple errors are reported. Look at the first error: it means that `cin` does not have a `<<` operator that will work with an `int` (totally cryptic!).

- If you click on this error, it will take you to the line of code in `main()` which is using the incorrect stream operator.
 - If you click on some other errors (which have *even* less useful descriptions) you may be taken to other files. Here are a couple rules to figuring out build errors:
 1. Always look at the first error (only). Track that one down, then rebuild (i.e., recompile/run). One problem in your code can cause many build errors.
 2. The problem is almost always going to be in code *you* write, not in code provided with the compiler. Therefore, if you follow an error and it's showing you code you didn't write, look elsewhere for the error.
 3. The actual problem in your code may be a line or two above where the compile tells you it thinks the problem is. This is especially tricky if you have a problem with your curly-braces like an extra `{` or missing a `}`.
5. Correct all compile time errors and re-run your code to prove it is working.
 6. **Understanding Question:**
If your code does not compile and the error message shows the problem in some code provided by the compiler, what should you do?

5. Code Formatting

“There is always time to correctly format your code. Always.”

- Dr. Brian!

- Correct code formatting means having the start of each line of code correctly aligned using tabs (or spaces, depending on your IDE), plus (later we’ll learn) where to put the { and }’s.
- **Always format your code correctly as you write your code.**
- Students sometimes ask me about a problem in their code when their code is also incorrectly formatted. Often this incorrect formatting can make it hard to find the bug! So, always format your code correctly as you write your code.
- If you ask for help on incorrectly formatted code, I’ll say, “I can’t read your code; it’s incorrectly formatted.” However, **I am very happy to help you reformat your code!!**
- Hint: Always format your code correctly as you write your code.
- As you code, VS Code will generally give you pretty good indentation! Each time you hit enter, it will usually start the next line at a good place.
You can also have VS Code reform all the code in your file!
 1. Press CTRL + SHIFT + P
 2. Type in “Format Document”
 3. Press ENTER and you’ll see VS Code’s suggestion on how to format your document. It’s usually pretty good, but may not be exactly what you want.
- Nothing need be done for this part of the lab, other than to have a look at how your code is formatted.
- In summary:

**Always format your code correctly
as you write your code.**

6. Exploring `print`

Get used to exploring and self-discovery with these labs. Here's an easy start:

1. Experiment by adding additional `print/println` statements in your code. Have it print out the name of your favorite colour.
2. Add statements to your program to, in addition to the above behaviour, print out the following text, including the two lines of ten `*`'s:

```
*****
```

A quote by Sir John A. Macdonald:

Let us be French, let us be English, but most importantly let us be Canadian!

```
*****
```

Note to Python programmers: C++ does not have an operator to repeat a character; you will have to type the all the `*`'s you want directly.

7. Optional: Extra Challenge

Try these tasks for an extra challenge. You don't need to submit any of this code for marks (but it's OK if you do submit it).

- ◆ Using `print/println` statements, draw a rectangle on the screen out of `*`'s:

```
*****
```

```
*       *
```

```
*****
```

- ◆ Write some code which prints out a Haiku or a (clean) limerick. Format it so that it looks good on screen.
- ◆ Write some code which outputs the following text on the screen:
He'll say, "WOW! She\he did great!"

8. Submitting

Assignments and labs are submitted using CourSys where you can also see your marks. Get in the habit of submitting carefully! You can only get marks if you submit your work correctly.

Depending on how you are writing your C++ code (locally on your computer, remote SSH, VDI, ...), the details on how to upload your code may differ. **See the online guides for details.**

1. In a web browser, go to **CourSys**: <https://coursys.sfu.ca/>
2. Browse into **CMPT 130**, and select the **Lab 1** activity; it will likely be highlighted.
3. **Upload your lab1.cpp file.** If you resubmit, only the last version submitted is marked.
4. Finally, through CourSys, open your submitted file in your browser and see if it's the correct contents! This is a great final step to do for all assignments and labs to ensure you submitted the correct file.