Debugging

CMSC 240 Software Systems Development

Today

- Debugging your code
- Using the debugger
- In-class debugging exercise

RICHMOND


Debugging Your Code
RICHMOND

Software Bug

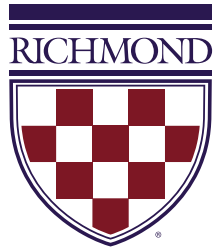
Bug - an error, flaw, failure, or fault in a computer program that produces an incorrect, unintended, or unexpected result

Computer pioneer Grace Hopper

RICHMOND

What is Debugging?

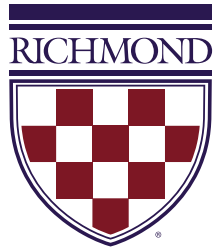
- When you have written a program, **it will have bugs**
  - It will do something, but not what you expected
  - How do you find out what it actually does?
  - How do you correct it?
  - This process is called **debugging**



Stepping Through a Program

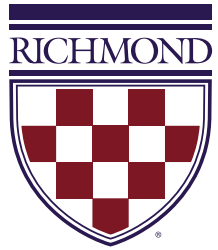
- Carefully follow the program through the specified steps
- Pretend you're the computer executing the program
- Does the output match your expectations?
- Need more information? Add a few debug output statements:

```
cout << "x == " << x << ", y == " << y << endl;
```



Beginnings and Ends

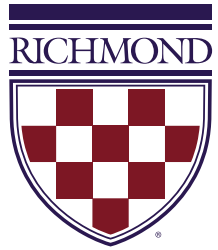
- Pay special attention to beginnings and ends
  - of loops (for/while)
  - of functions
  - of classes (constructor/destructor)
- Did you initialize every variable?
  - To a reasonable value
- Did the function get the right arguments?
  - Did the function return the right value?
- Did you handle the first element correctly?
  - The last element?



Be Guided By Output

"If you can't see the bug, you're looking in the wrong place"

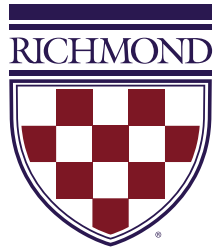
- It's easy to be convinced that you know what the problem is and stubbornly keep looking in the wrong place
  - **Don't just guess**
  - **Be guided by output**
- Work forward through the code from a place you know is right
  - What happens next? Why?
- Work backwards from some bad output
  - How could that possibly happen?



Types of Debugging

Adding `cout <<` statements

- print the values of variables that to see what is going on
- print when you enter and exit functions
- print to find where you are in the code
- print to confirm that a class was properly initialized
- print useful diagnostic information



Adding a Debug Function

```
#define DEBUG_ON true

// Create a debug function to output only if debug is on.
void debug(string message)
{
    if (DEBUG_ON)
    {
        cerr << message << endl;
    }
}
```

Adding a Debug Function

```
// Calls the area function after reducing
// the length and width by frame size.
int framedArea(int length, int width)
{
    debug("Begin framedArea");

    int frameSize = 2;

    // Do not catch exception here.
    int result = area(length - frameSize, width - frameSize);

    debug("Return from framedArea with result == " + to_string(result));

    return result;
}
```

Redirect Debug Output

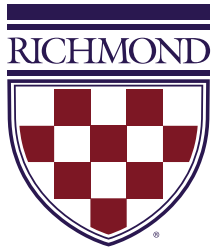
Since we used **`cerr`** in our debug function, another trick is to redirect the **`cerr`** standard error stream (2) to a log file

Redirect standard output (cout)

```
$ ./helloworld > output.txt
```

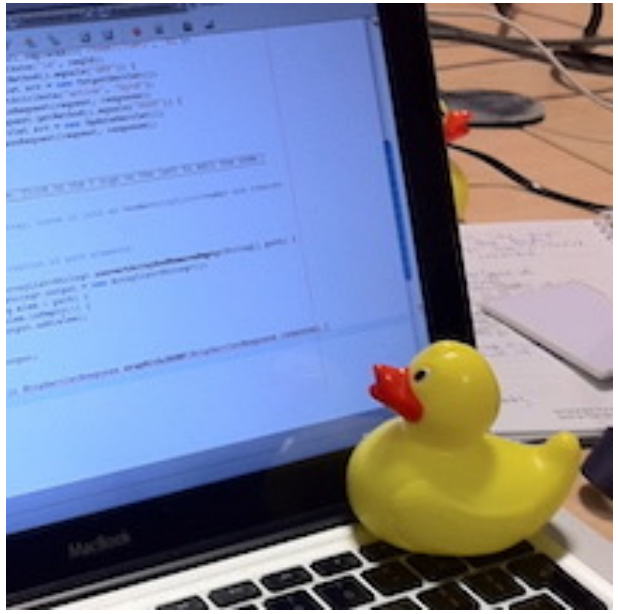
Redirect standard error output (cerr)

```
$ ./helloworld 2> debug_output.txt
```



Rubber Duck Debugging

https://en.wikipedia.org/wiki/Rubber_duck_debugging

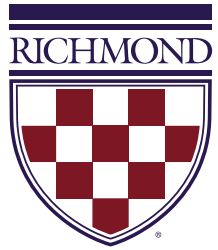


- Talking through code with someone else
- If someone else is not available use a rubber duck
- Just by talking through your code, step-by-step, will help you realize what is wrong with the code

RICHMOND

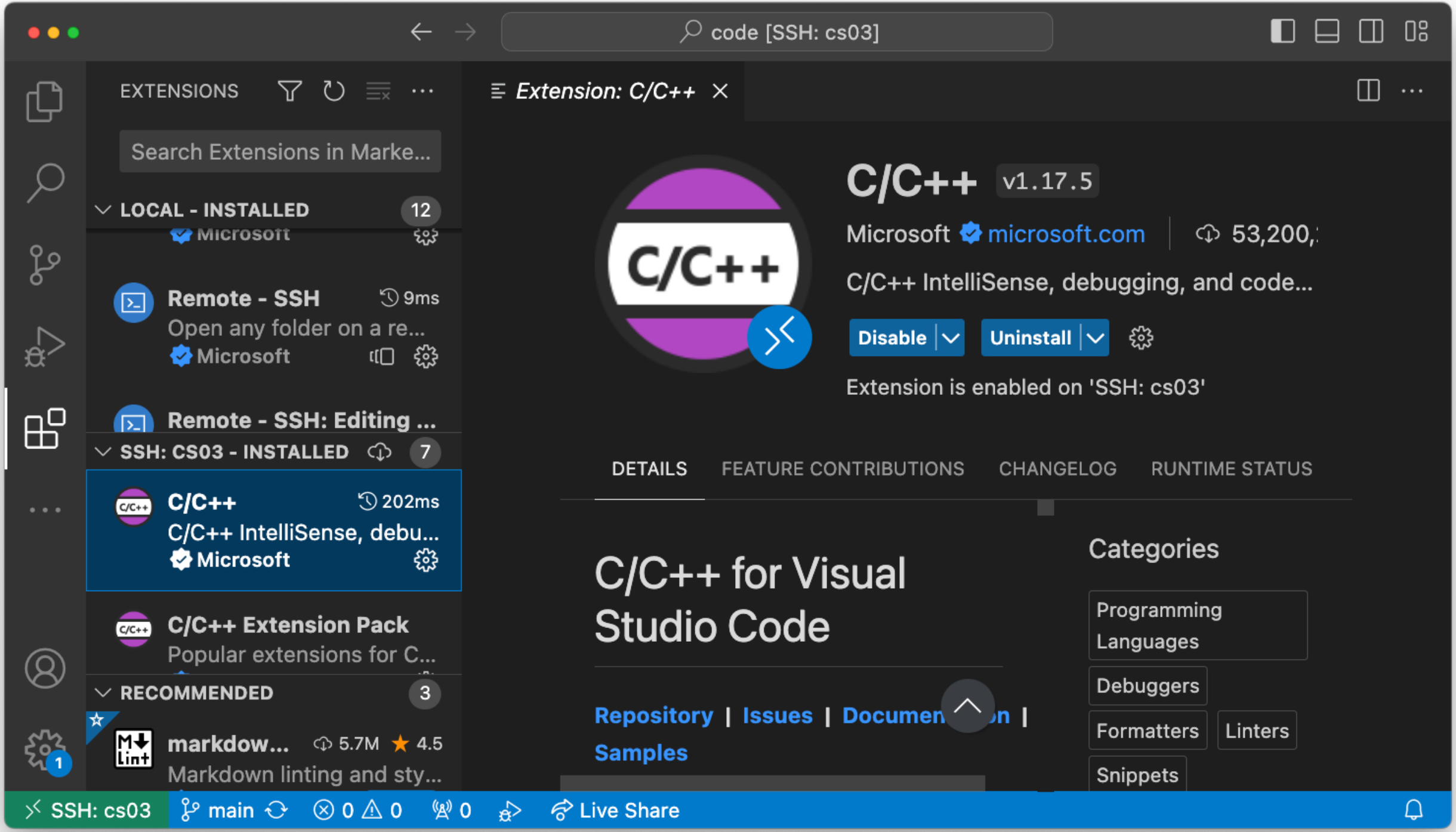
Debugger

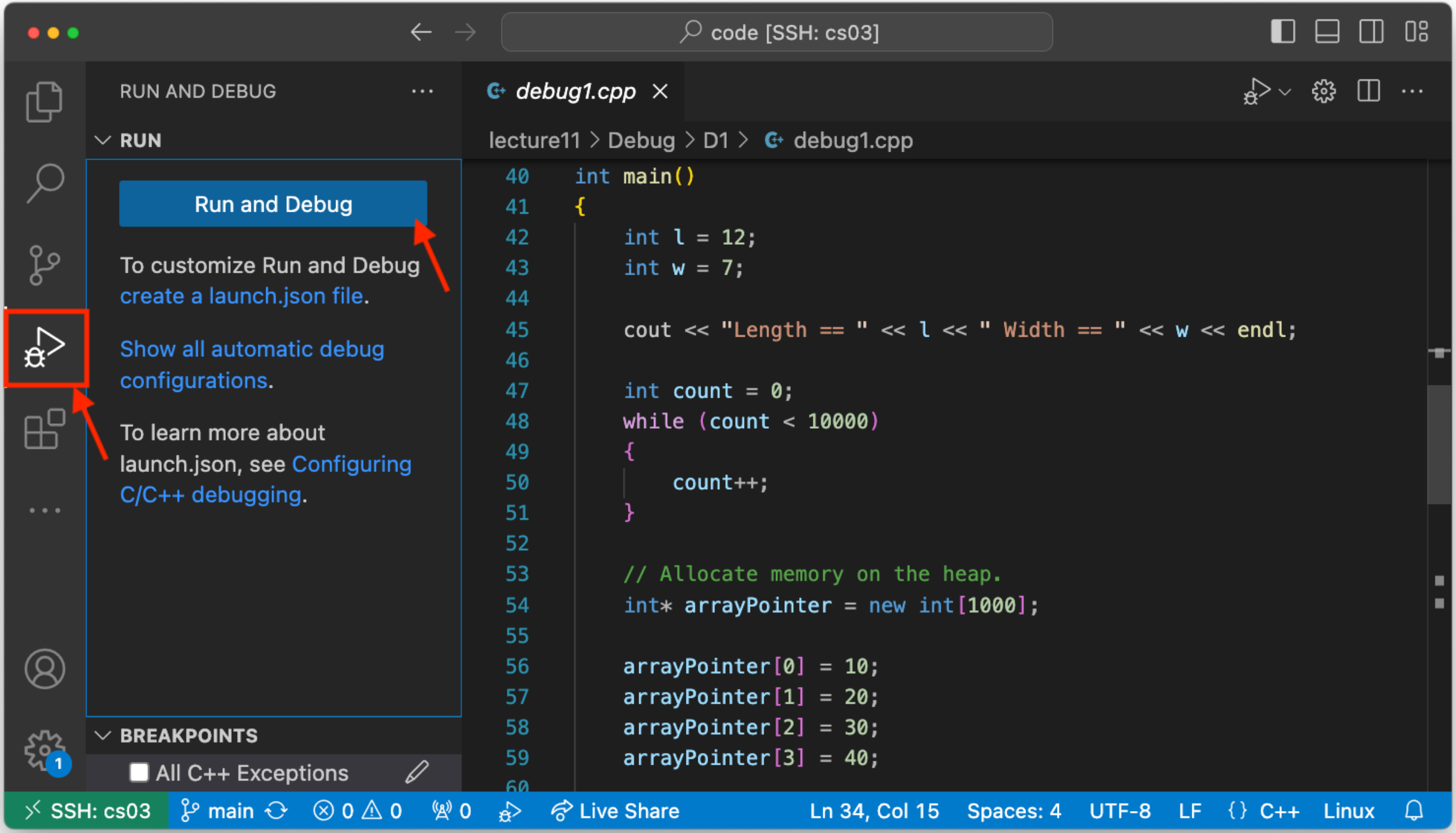
A **debugger** is a developer tool that attaches to your running program and allows you to inspect your code



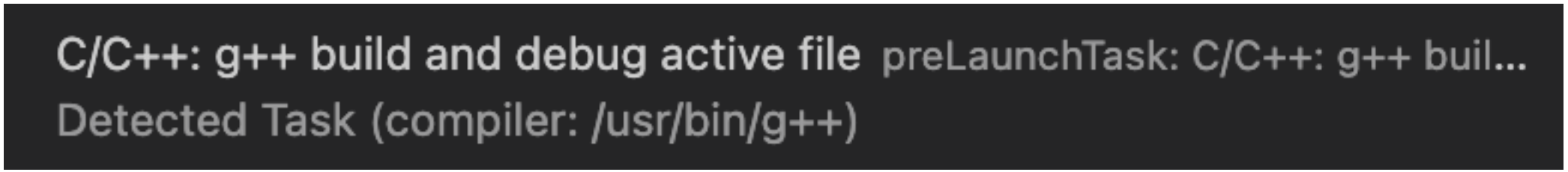
Setting a Breakpoint


```
10  int main()
11  {
12      int x = 22;     // Set a breakpoint here.
13      int y = 10;
14
15      cout << "x == " << x << ", y == " << y << endl;
16
17      int sum = add(x, y);
18
19      cout << "sum == " << sum << endl;
20
21      return 0;
22  }
```

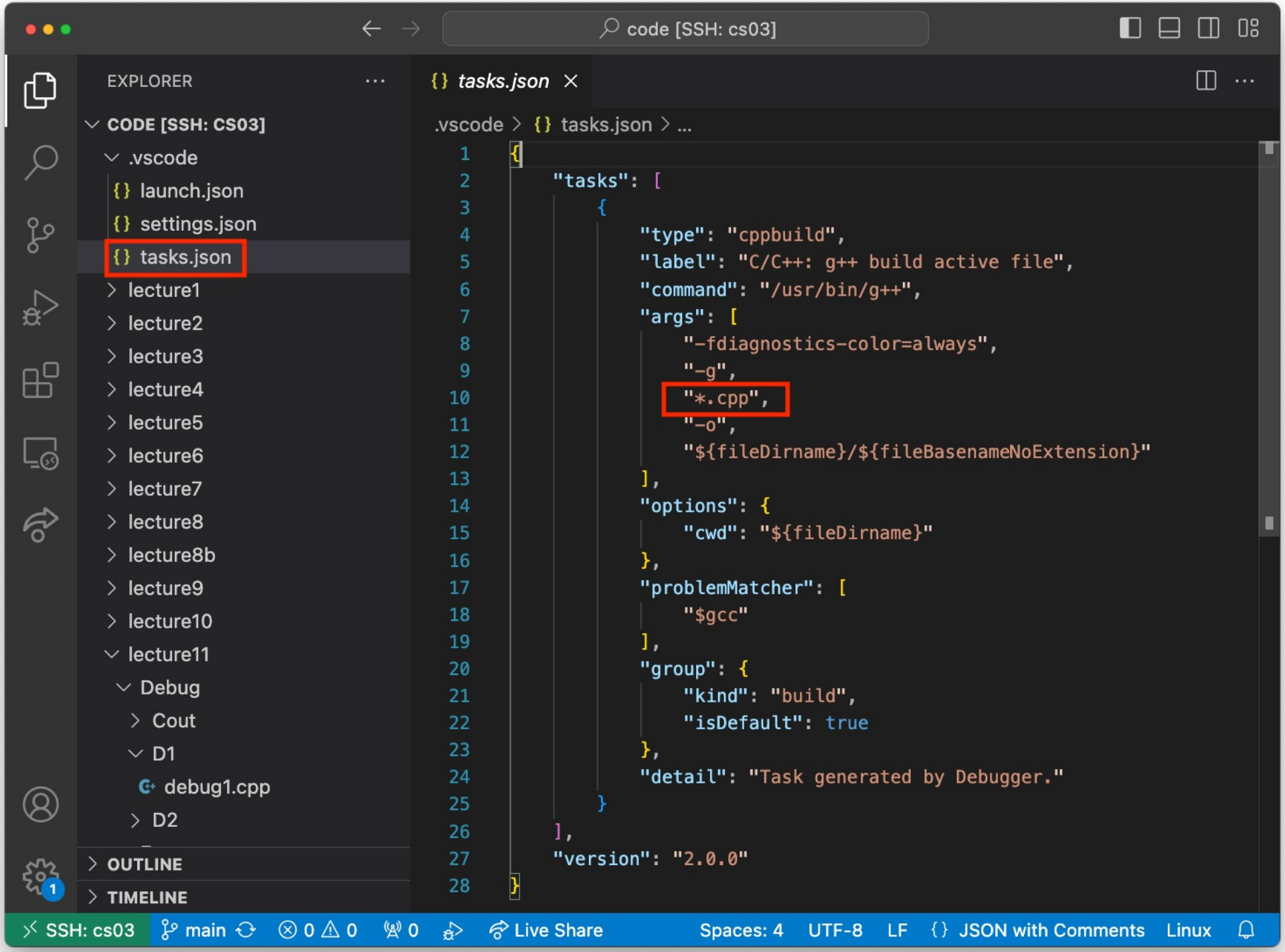

code [SSH: cs03]
EXTENSIONS
Search Extensions in Marke...
LOCAL - INSTALLED 12
Remote - SSH 9ms
Open any folder on a re...
Microsoft
Remote - SSH: Editing ...
SSH: CS03 - INSTALLED 7
C/C++ 202ms
C/C++ IntelliSense, debu...
Microsoft
C/C++ Extension Pack
Popular extensions for C...
RECOMMENDED 3
markdow... 5.7M 4.5
Markdown linting and sty...
Extension: C/C++
C/C++ v1.17.5
Microsoft microsoft.com 53,200,
C/C++ IntelliSense, debugging, and code...
Disable
Uninstall
Extension is enabled on 'SSH: cs03'
DETAILS
FEATURE CONTRIBUTIONS
CHANGELOG
RUNTIME STATUS
C/C++ for Visual Studio Code
Repository | Issues | Documen n |
Samples
Categories
Programming Languages
Debuggers
Formatters
Linters
Snippets
SSH: cs03
main
0 0
0
Live Share


code [SSH: cs03]
RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug create a launch.json file.
Show all automatic debug configurations.
To learn more about launch.json, see Configuring C/C++ debugging.
BREAKPOINTS
All C++ Exceptions
debug1.cpp
lecture11 > Debug > D1 > debug1.cpp
40 int main()
41 {
42 int l = 12;
43 int w = 7;
44
45 cout << "Length == " << l << " Width == " << w << endl;
46
47 int count = 0;
48 while (count < 10000)
49 {
50 count++;
51 }
52
53 // Allocate memory on the heap.
54 int* arrayPointer = new int[1000];
55
56 arrayPointer[0] = 10;
57 arrayPointer[1] = 20;
58 arrayPointer[2] = 30;
59 arrayPointer[3] = 40;
60
SSH: cs03
main
0 0
0
Live Share
Ln 34, Col 15
Spaces: 4
UTF-8
LF
C++
Linux

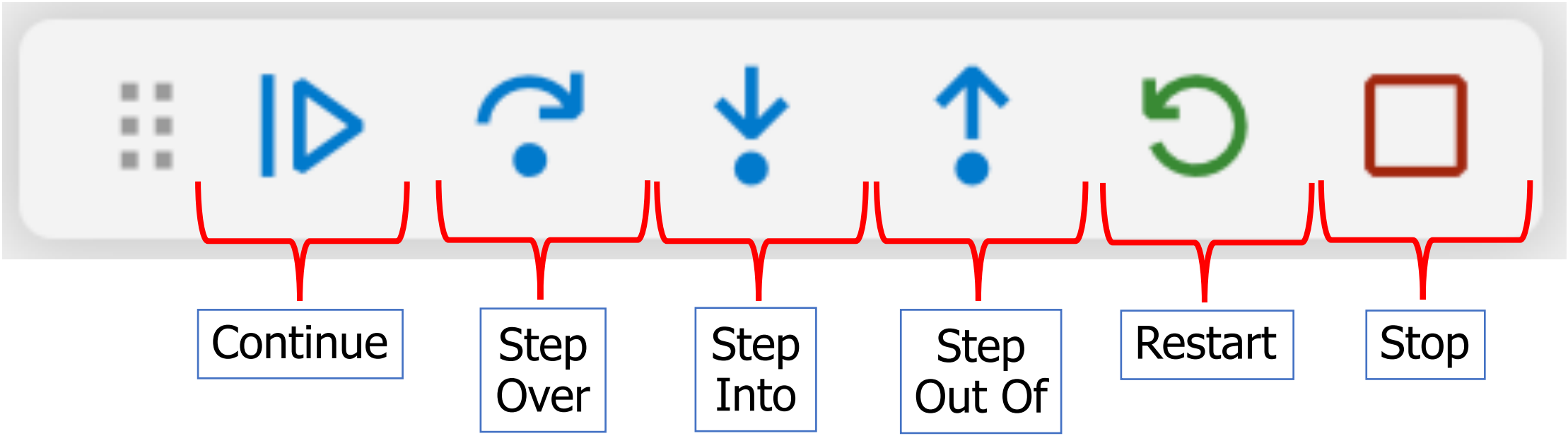
Select:



RICHMOND


code [SSH: cs03]
EXPLORER
CODE [SSH: CS03]
.vscode
launch.json
settings.json
tasks.json
lecture1
lecture2
lecture3
lecture4
lecture5
lecture6
lecture7
lecture8
lecture8b
lecture9
lecture10
lecture11
Debug
Cout
D1
debug1.cpp
D2
OUTLINE
TIMELINE
tasks.json
.vscode > tasks.json > ...
{
"tasks": [
{
"type": "cppbuild",
"label": "C/C++: g++ build active file",
"command": "/usr/bin/g++",
"args": [
"-fdiagnostics-color=always",
"-g",
"*.cpp",
"-o",
"${fileDirname}/${fileBasenameNoExtension}"
],
"options": {
"cwd": "${fileDirname}"
},
"problemMatcher": [
"$gcc"
],
"group": {
"kind": "build",
"isDefault": true
},
"detail": "Task generated by Debugger."
}
],
"version": "2.0.0"
}
SSH: cs03
main
0 0
0
Live Share
Spaces: 4
UTF-8
LF
JSON with Comments
Linux

Debugger Controls



Debugger Demo