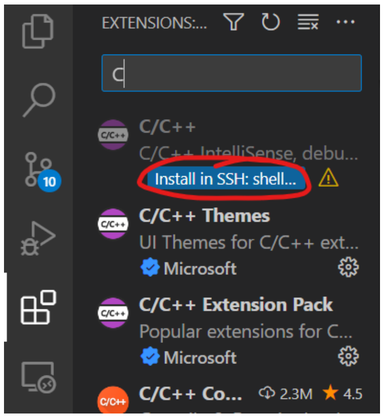
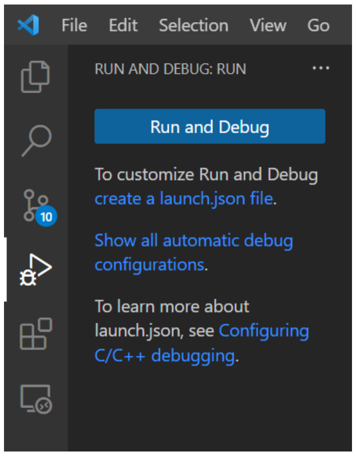
# Setting Up GDB in VSCode

GDB is the GNU Project Debugger. It is a useful command line tool for debugging your C programs when some part of your code is broken and you want to examine your code at runtime (e.g. set breakpoints, print local variables, view backtrace). VSCode has a nice built-in debugger that uses GDB. Here is how to set it up.

1. SSH into shell.seas.gwu.edu in VSCode
2. Install the C/C++ extension on the shell. Note you probably already installed the C/C++ extension on your local machine, this is installing the extension the VSCode server on your shell account.



3. Open the debugging panel. Click "create a launch.json file"

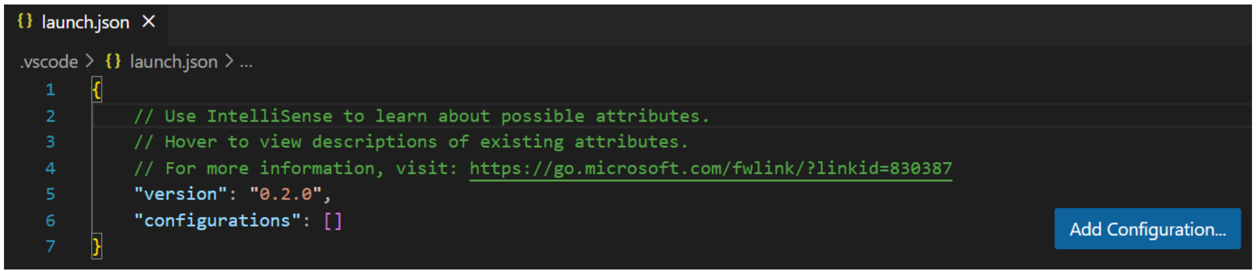


To customize Run and Debug create a launch.json file.

Show all automatic debug configurations.

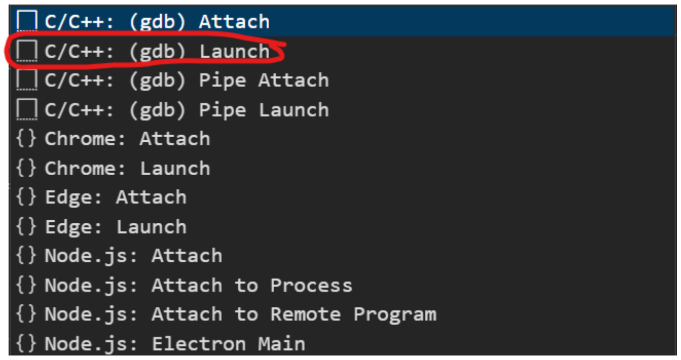
To learn more about launch.json, see Configuring C/C++ debugging.

4. You will see a json file like this that was created in ~/.vscode/launch.json



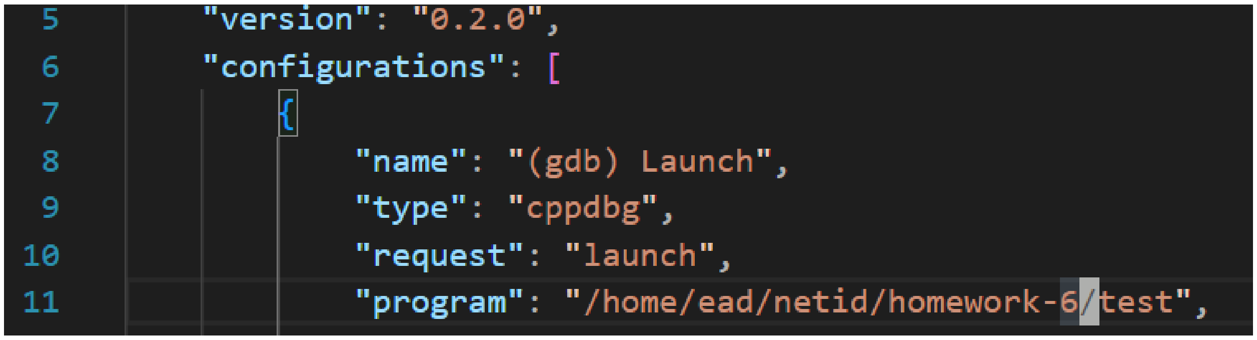
```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": []
}
```

5. Click "Add Configuration". Select "(gdb) Launch". This creates a new GDB process when you start debugging.



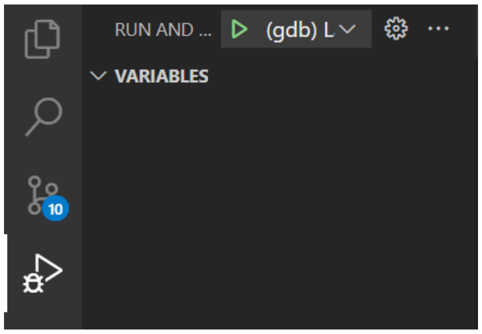
```
C/C++: (gdb) Attach
C/C++: (gdb) Launch
C/C++: (gdb) Pipe Attach
C/C++: (gdb) Pipe Launch
{} Chrome: Attach
{} Chrome: Launch
{} Edge: Attach
{} Edge: Launch
{} Node.js: Attach
{} Node.js: Attach to Process
{} Node.js: Attach to Remote Program
{} Node.js: Electron Main
```

6. Next, modify the configuration file to include the path to your executable. For example, if you run your executable ./test, then you'll put the "/path/to/test" in "program". You can get your path by running "pwd" in the terminal or by right clicking the filename and "Copy Path".



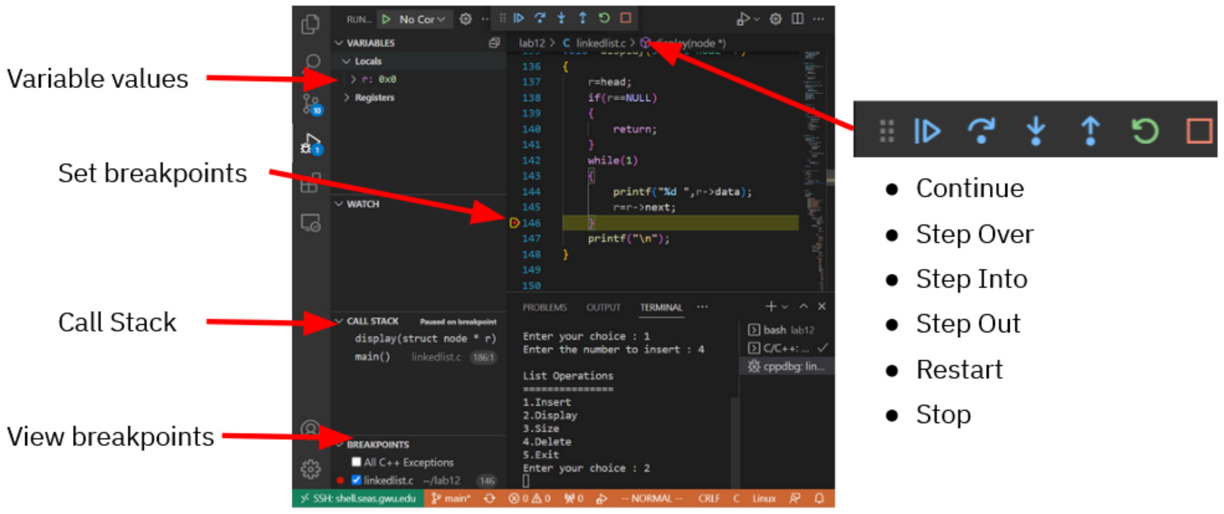
```
    "version": "0.2.0",
    "configurations": [
        {
            "name": "(gdb) Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "/home/ead/netid/homework-6/test",
```

7. Next, go into the debugging panel and Start Debugging.



8. If you get an error about needing an extension for comments in JSON files, then delete the comments in the launch.json file and run again.

9. This is the view you should get when debugging. Select the terminal output corresponding to the debugger (cppdbg).



```
{
    r=head;
    if(r==NULL)
    {
        return;
    }
    while(1)
    {
        printf("%d ",r->data);
        r=r->next;
    }
    printf("\n");
}
```

```
Enter your choice : 1
Enter the number to insert : 4

List Operations
===============
1.Insert
2.Display
3.Size
4.Delete
5.Exit
Enter your choice : 2
```

10. If you want to debug a different program, you will need to update your launch.json file. You can find more about customizing your configuration file here: [https://code.visualstudio.com/docs/editor/debugging#_launch-configurations](https://code.visualstudio.com/docs/editor/debugging#_launch-configurations)

## Tips on Using GDB in VSCode

Debugging can be a frustrating experience, but it is something that all programmers go through. Finding a methodical and efficient debugging practice will save you lots of time and headache. Over time you'll develop an intuition for solving bugs, so don't give up! Here are a few tips you can use:

1. If you find yourself using print statements to get values of local variables, try setting a breakpoint instead. Add a breakpoint by clicking to the left of line number and a red dot appears. When you run the program, you'll be able to click "Continue" to view the state of the program at each breakpoint.
2. When you get a segmentation fault, try to identify the function that is causing it. Seg Faults are caused by an invalid access to memory (e.g.) trying to access the field of a Null struct. Look for lines of code that might make illegal memory accesses, like pointers. Set a breakpoint at the function and run your code line by line using the Step Over button. You can also use Step Into and Step Out when a function is called and you want to see what happens in the function.
3. Finally, verify the contents of your variables is what you expect. For structs and arrays, you can click the drop down to view the contents of the fields.