This document covers how to push to the codebase and some details about what to do for 3.0.

Warning! DO NOT PUSH TO MASTER

Requirements

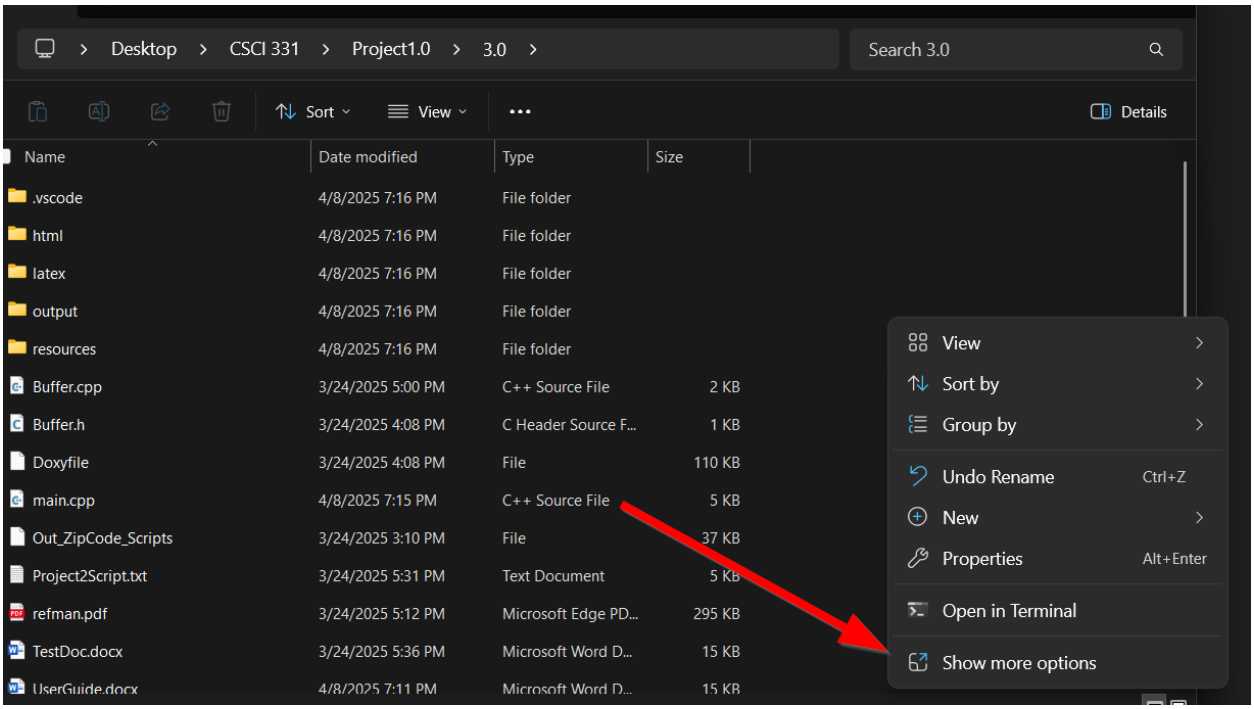
1. Visual studio code
2. Windows
3. Git

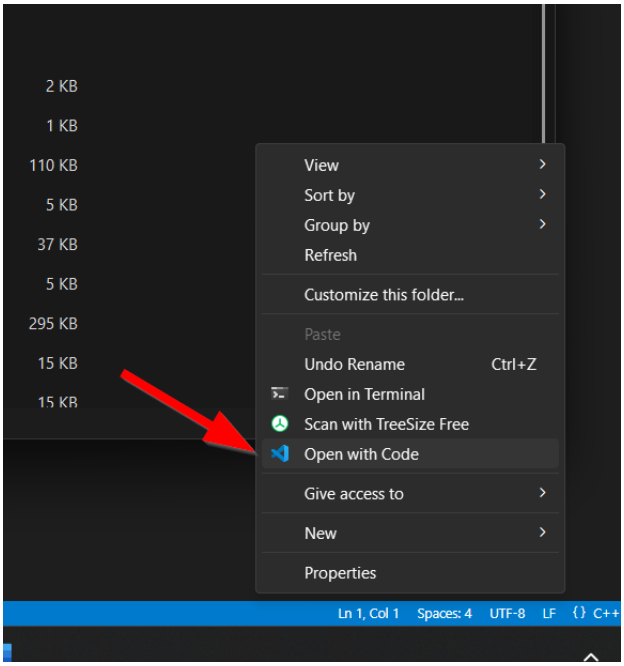
Instructions

1. If you haven't *cloned* already, clone the repository to a folder in your desktop by opening a terminal and running these commands:
   a. `git clone https://github.com/kygm/BufferClassCSCI331.git`
   b. `git remote add origin https://github.com/kygm/BufferClassCSCI331.git`
   These commands clone the repository and add url to the repository to a variable origin
2. Pull
   a. git pull origin master
3. Create a branch
   a. `git branch your_branch_name`
   b. `git checkout your_branch_name`
4. Make code changes
   a. If I say to pull from master in the group chat, run
      i. `git pull origin master`
5. Push your changes
   a. `git add .`
   b. `git commit -m "enter which changes you made to the codebase"`
   c. `git push origin your_branch_name`

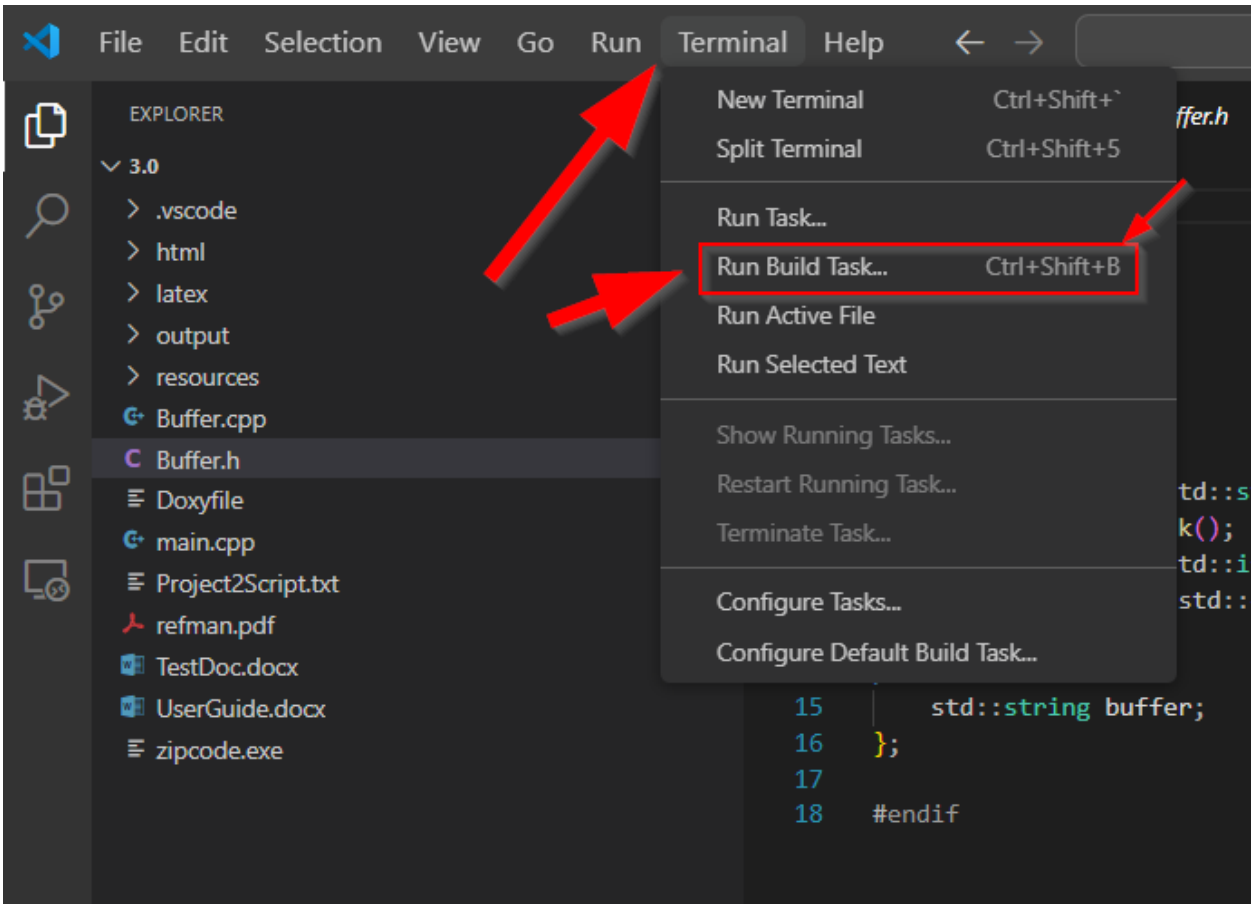
How to build and debug

1. From windows, open visual studio code in the folder where the code is:





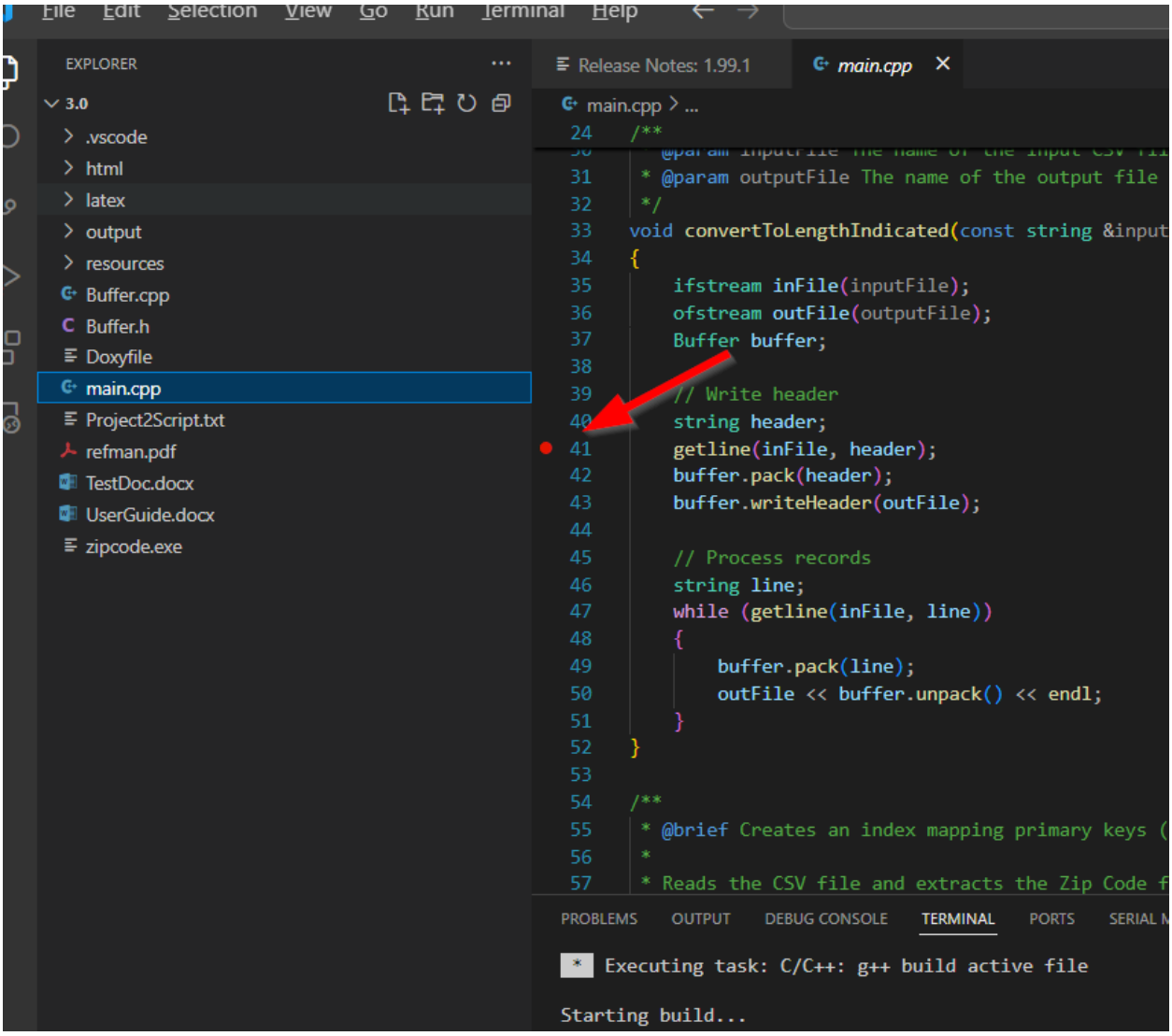
2. Build the project



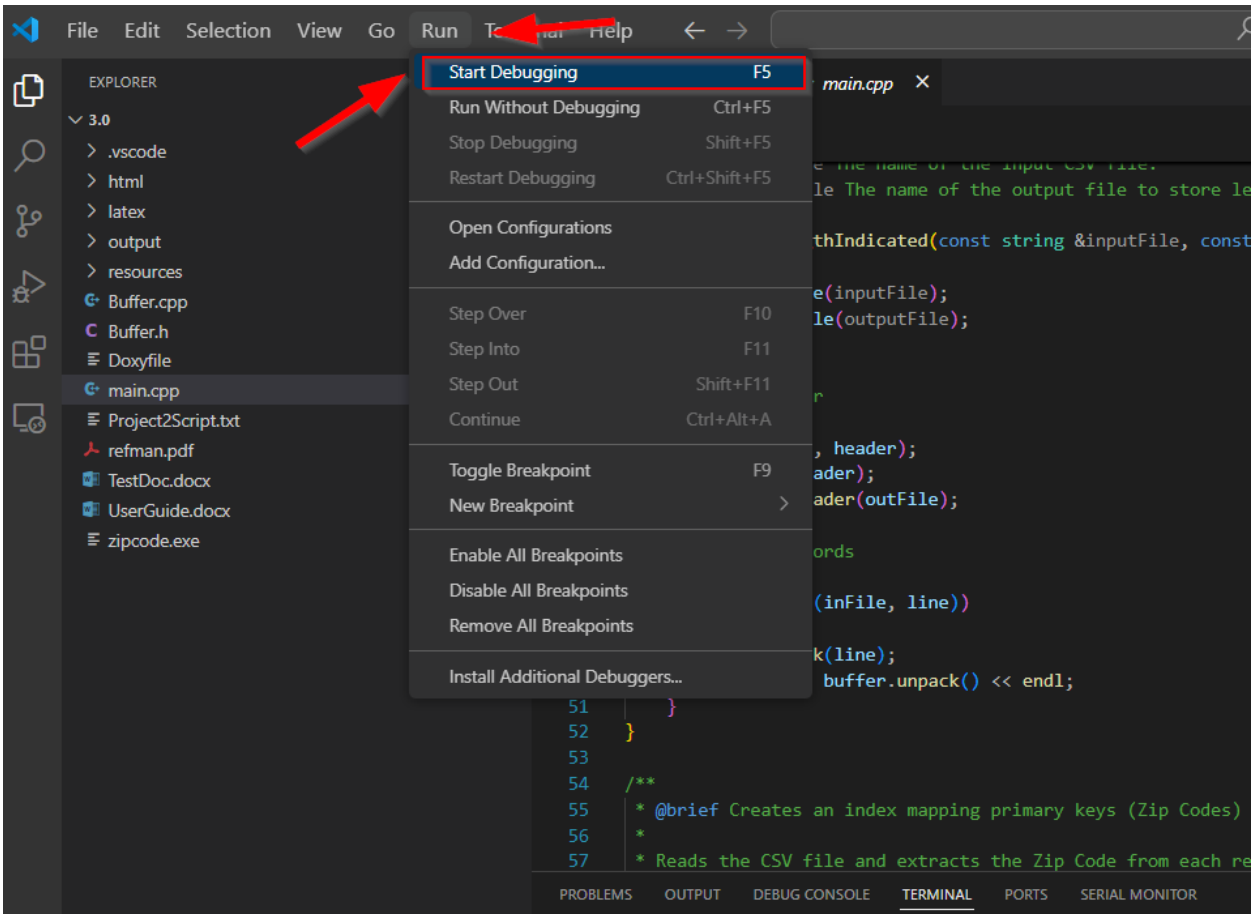
Notice: you can also build using the keyboard shortcut ctrl+shift+b

3. Set breakpoints – breakpoints stop the code wherever you set them and let you see what the values of variables are. **Use them!** They can be set by clicking to the left of

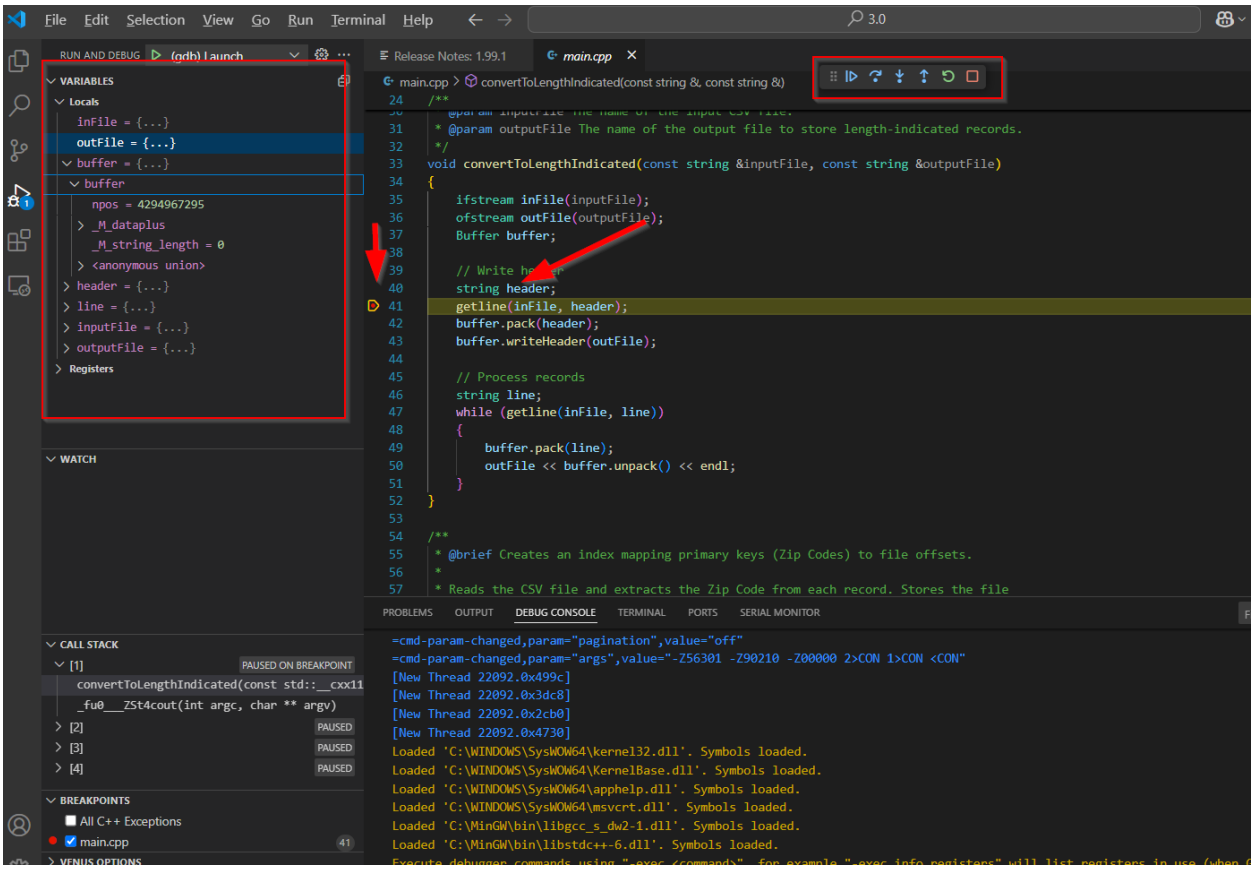
the line number



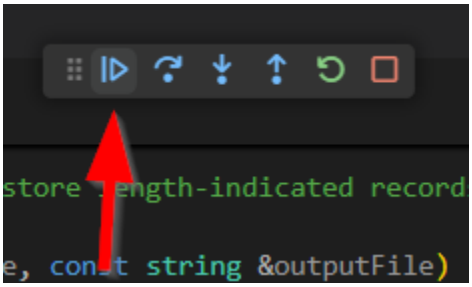
4. Start debugging:



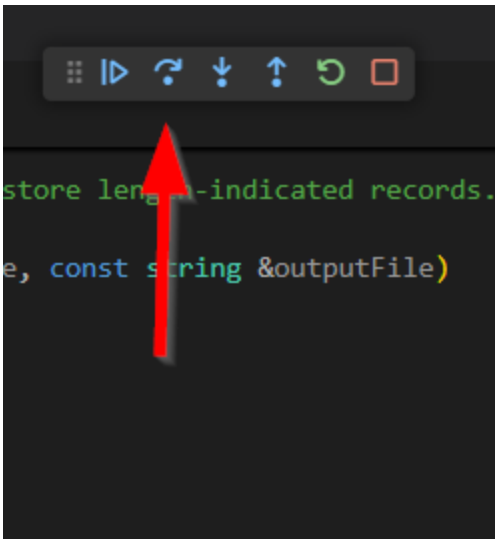
5. Breakpoints will be hit if you set any. To continue, hit the forward arrow:



This arrow to move past the breakpoint:

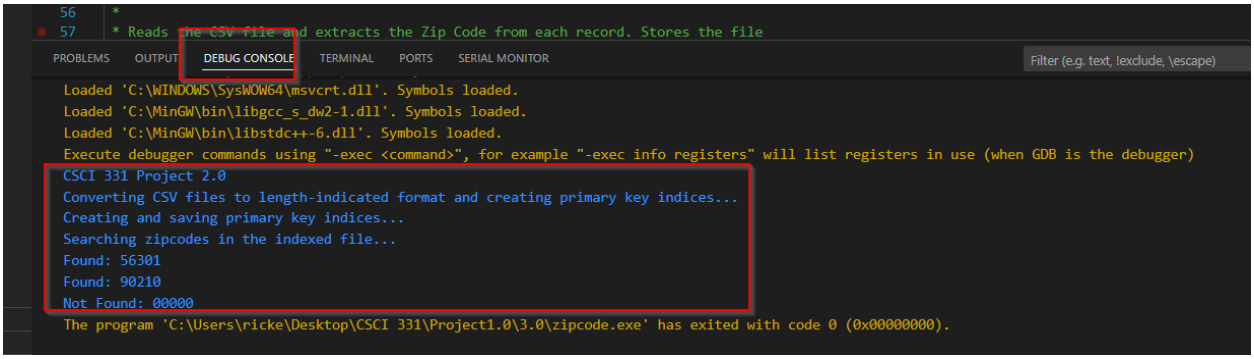


This one to go to the next line:

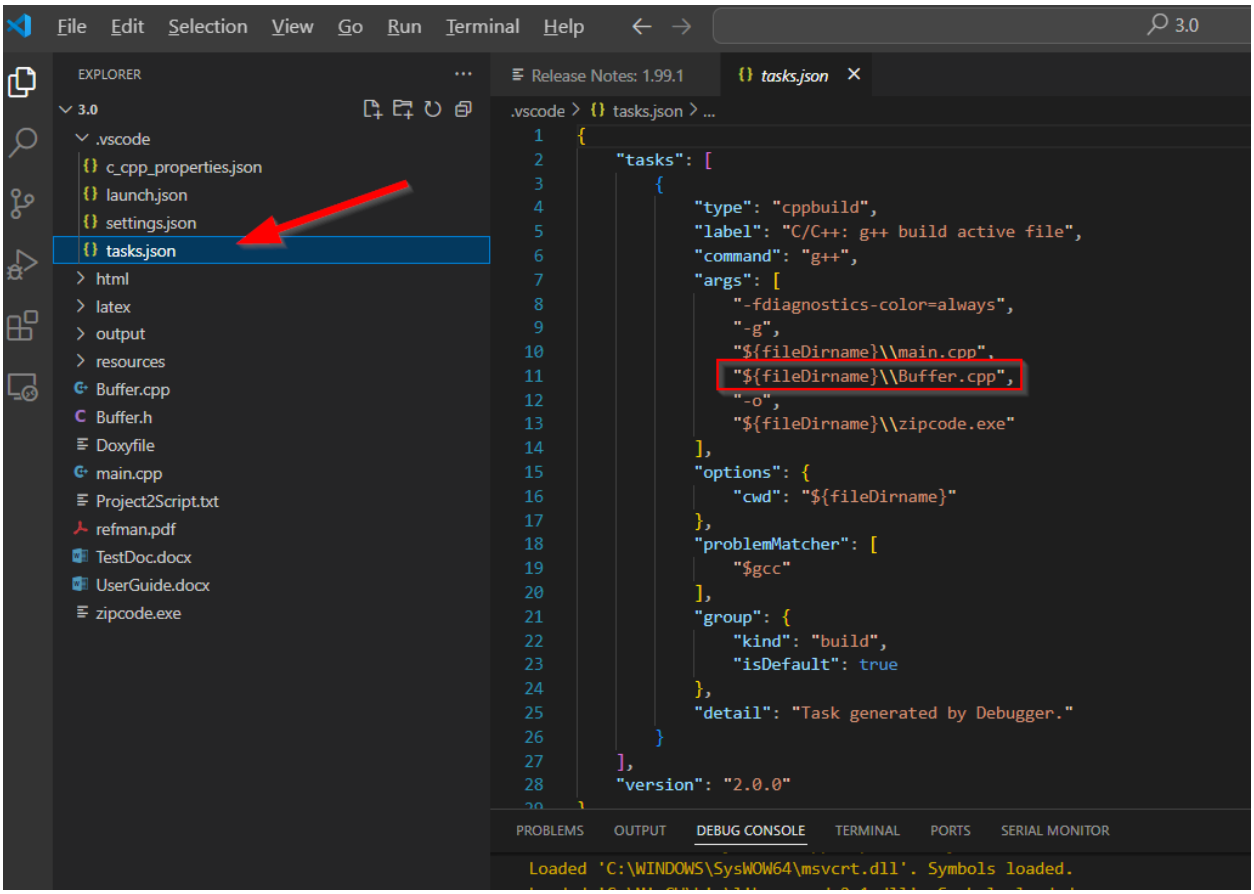


this one to restart and the square to stop:

6. Your output will be in the debug console

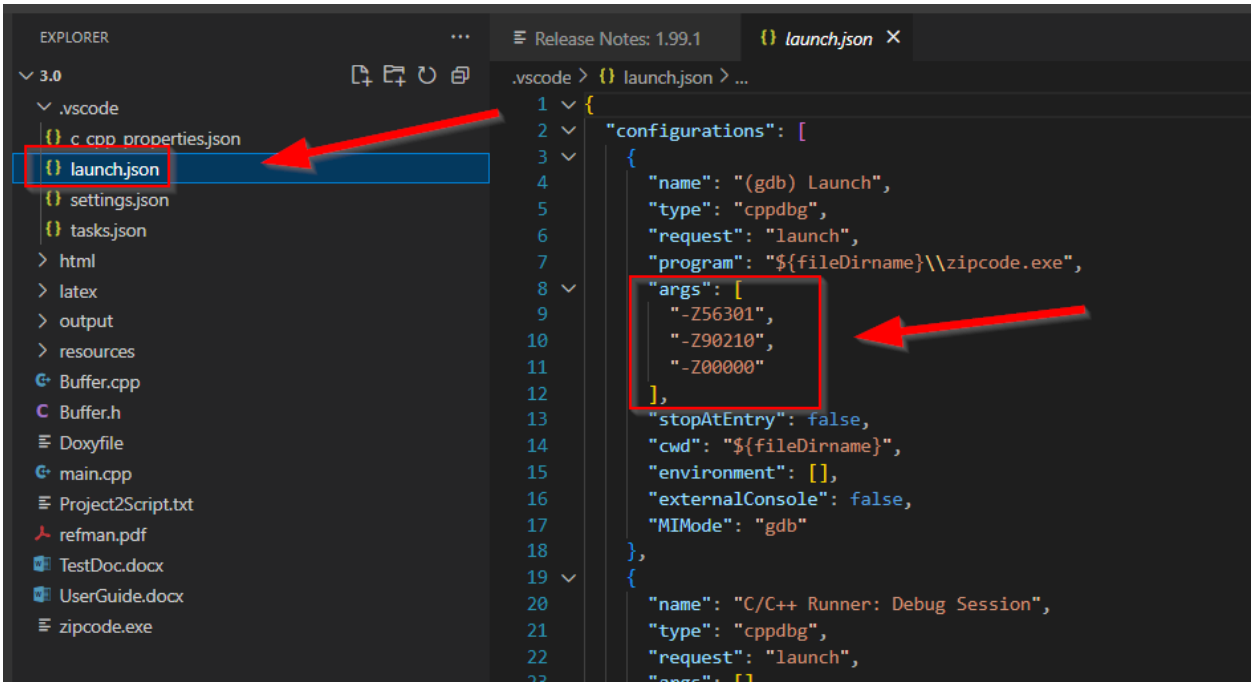


7. If you need to add new files to compile, modify tasks.json in .vscode



So if you need to compile MyNewFile.cpp, duplicate the line in the rectangle and rename Buffer.cpp to MyNewFile.cpp

8. If you need to change the command line arguments (used in debug), modify them in launch.json:



9. Always push your changes! If you don't push your changes, we wont see them!

The following needs to be done:

1. Generate a *blocked* **sequence set** file from the data file you created in Group Project 2.0
    - Your blocked sequence set generation program's command line options should include:
        - the name of the blocked sequence set data file
        - all other information necessary for the header file
    - All blocks are the same size. (See the **Header Record Architecture** section below for the default size
    - Each block will contain a set of complete records (some blocks may have different counts of records) and a metadata architecture as shown in the **Block Architecture** section below
    - Unused or deleted blocks are *avail* list blocks (See Folk 6.2.2 & 10.1 – 10.3)
2. Process sequentially a blocked **sequence set** file using buffer classes. {functionality from Group Projects 1 & 2}

3. Use both a *block* buffer class and a *record* buffer class to read and unpack Zip Code Records from a sequence set **block** into a sorted container of record objects.
    - The *block* buffer unpacks a *record* from a block into a record buffer.
    - The *record* buffer unpacks *fields* from the record buffer into a record object.
4. *Modify* your data file *header record* buffer class to read and write the blocked sequence set data file header record
5. *Repeat* Group Project 1.0 with this new blocked sequence set file.
6. *Create* and use two blocked sequence set dump method that visibly aggregates Zip Codes into blocks including the respective predecessor & successor R(elative)B(lock)N(umber) links.
   One dump method will list the blocks sequentially by their physical ordering; the other dump method will list the blocks sequentially by their logical ordering. (after initial creation, both dumps will generate identical output, but use of a non-appending avail block will make them different)

```
List Head:  RBN
Avail Head: RBN
RBN  key_a key_b … key_i RBN
RBN  *available*     RBN
RBN  key_a key_b … key_j RBN
:
RBN  key_a key_b … key_k RBN
```

   This dump format makes it rather easy to check the results of insertions and deletions for appropriate changes — you could even use the **diff** program.
   It helps to use the smallest possible non-trivial sub-set of the data initially, so as to generate a dump which fits on a single page/window.

7. Create a simple index file which contains ordered pairs of keys (highest key in each block) & block numbers. (See Folk Figure 10.3)
8. Create a readable dump of the simple index
9. Generate (in RAM), write (as a file), and read (back into RAM), a simple primary key index [Folk Section 10.3] that can be used to display the Zip Code data for all Zip Codes listed on the command line.
   This index will store the ordered pairs: {<*highest* ***key*** *in block*>, <RBN>}

- Your blocked sequence set search program's command line options should include the name of the blocked sequence set data file
- Use a command line flag (e.g. **-Z56301**) to indicate each Zip Code record to search for.
- If the Zip Code record is not in the file, display a message to that effect.
  - Note that to determine that a record is not in the file, the indexed **block** must be read, unpacked, and searched
- Test Run Demonstration: for the blocked sequence set Zip Code data and simple index file pair
  1. Create and run a search test program - include searches (on the command line) for several valid Zip Codes and at least one invalid Zip Code.
     - the program will load the simple primary key index file into an sorted container object in RAM
     - the program will **never** load the blocked sequence set Zip Code data file into RAM
  2. Create and run a record addition and deletion test program
     - **record addition**: use the command line to indicate a file of records to add
       1. When a block is split, log the event.
       2. Optionally, also run the two dumps.
       3. If the index has to be modified, log the event.
       4. Optionally, run a dump of the index
     - **record deletion**: use the command line to indicate a file of keys for records to delete
       1. When two blocks are merged, or participants of a redistribution, log the event.
       2. Optionally, also run the two dumps.
       3. If the index has to be modified, log the event.
       4. Optionally, run a dump of the index

10. All program variables and values that can vary should be initialized either by command line parameters (or their defaults) or meta-data in the the data file or index (e.g. header record info.)
11. Document (*extensively*) your C++ source code with comments and Doxygen tags.
12. Create a Doxygen PDF of your class and application program code.
13. Create a user guide showing how to use your program (including how to use the command line options, and how the output should appear)