GDB

In this worksheet, we are going to learn about using GDB to debug C++ programs.

The first part of this document contains generic information about using our Visual Studio Code labs environment. Skip to the section entitled 'Write and run a test program' if you are already familiar

Set up your environment

We have set up a lab environment for you to work in on the Coursera platform. It has all the tools you need to develop basic C++ code, namely a text editor, a compiler, a linker and a terminal in which to run your program in.

For this lab, we do not support development on your local machine as setting up a development environment is beyond the scope of this lab. We advise you to use the coursera labs environment.

Working in the Coursera lab environment

The Coursera lab environment has already been configured for you with all the essentials to complete this activity. The lab is integrated with Visual Studio Code, an extremely popular code editor optimised for building and debugging modern web and cloud applications.

Start the lab environment application

It is simple to launch a lab exercise. You only need to click on the button "Start" below the activity title to enter a lab environment. Let's explore this lab activity. Go ahead and click on the "Start" button!

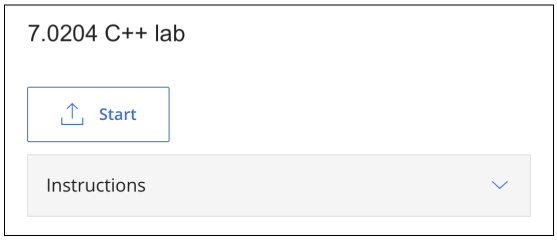

Figure 1: Coursera "Start" button

The Lab environment application

The lab environment application may take some time to load and you will be prompted with the following animation:

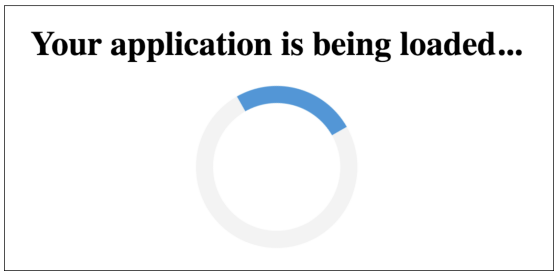

Figure 2: Coursera loading screen

Just wait a few seconds for your lab activity to start. It should not take longer than thirty seconds. If you experience any issues please load the activity again.

Once the application loads, you will see an instance of Visual Studio Code IDE as shown below:

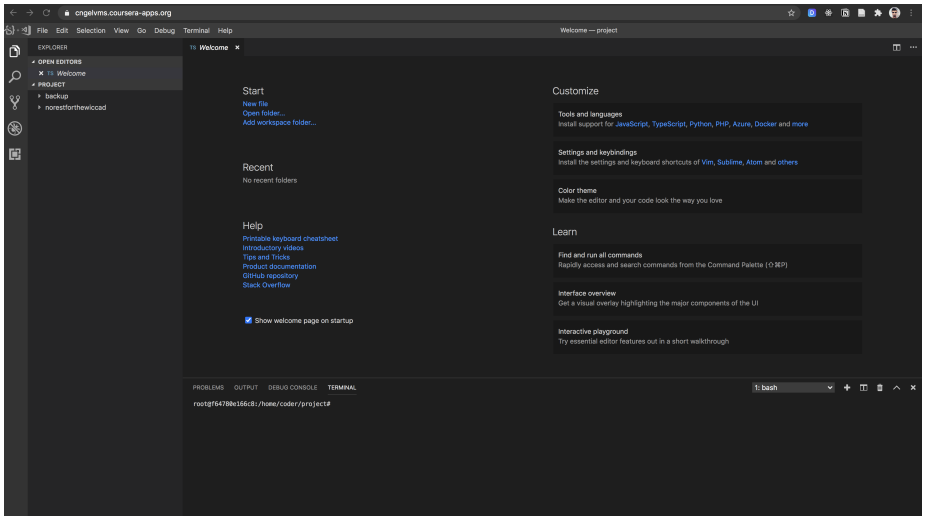

Figure 3: Visual Studio Code editor

The "Welcome" page illustrates most of the features to get you started with the environment. For now you can close the "Welcome" tab by clicking on the X icon next to the tab name as we will explore the main sections together. The "Welcome" page does not always appear so do not worry as it will not make any difference in the way the IDE works.

The Visual Studio User Interface

The Visual Studio User Interface is very easy to learn and it comes with a great selection of features. We will not explore in details all of the functionalities as it is not in the purpose of this course. We will instead look at the main sections required for you to get started with the labs.

VS Code comes with a simple and intuitive layout that maximises the space provided for the editor while leaving ample room to browse and access the full context of your folder or project.

The VS Code layout sections that you will require for this exercises are the Side Bar, the Editor Groups and the Panels.

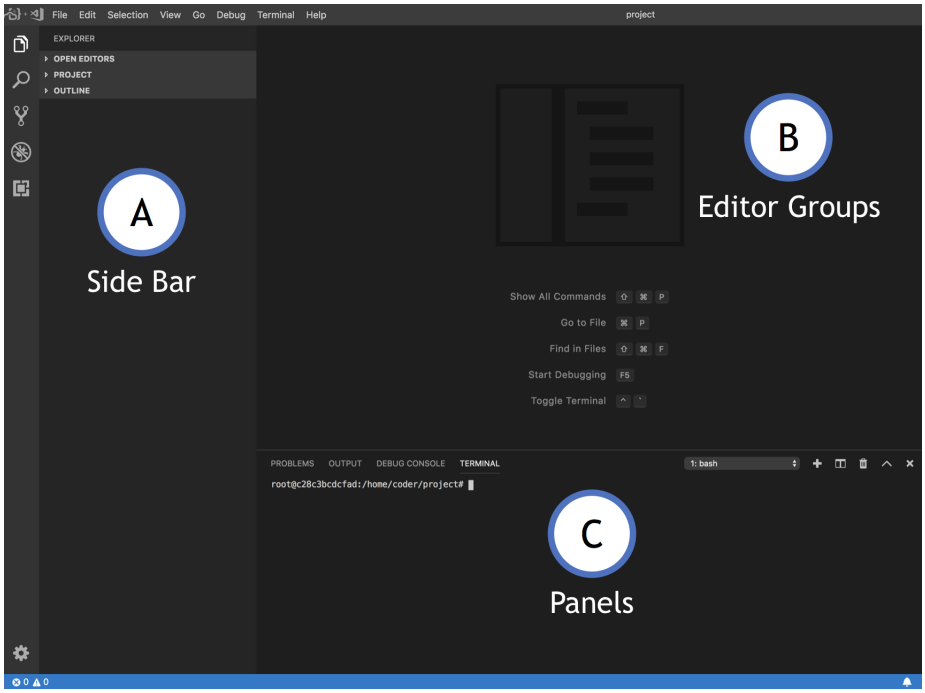

Figure 4: Visual Studio Code layout sections

- **Side Bar (A)** - Contains different views like the Explorer to assist you while working on your project. Mostly used to navigate your folders.
- **Editor Groups (B)** - The main area to edit your files. You can open as many editors as you like side by side vertically and horizontally.
- **Panels (C)** - You can display different panels below the editor region for output or debug information, errors and warnings, or an integrated terminal.

Do not worry if your IDE configuration looks slightly different from the above picture. You can always drag the tabs around to adjust your layout configuration.

Note about the integrated terminal

The integrated terminal panel is normally hidden by default in Visual Studio Code and you need to manually enable it. Click the "View" tab on the top navigation bar and then click the "Terminal" tab to enable the panel window like shown below:

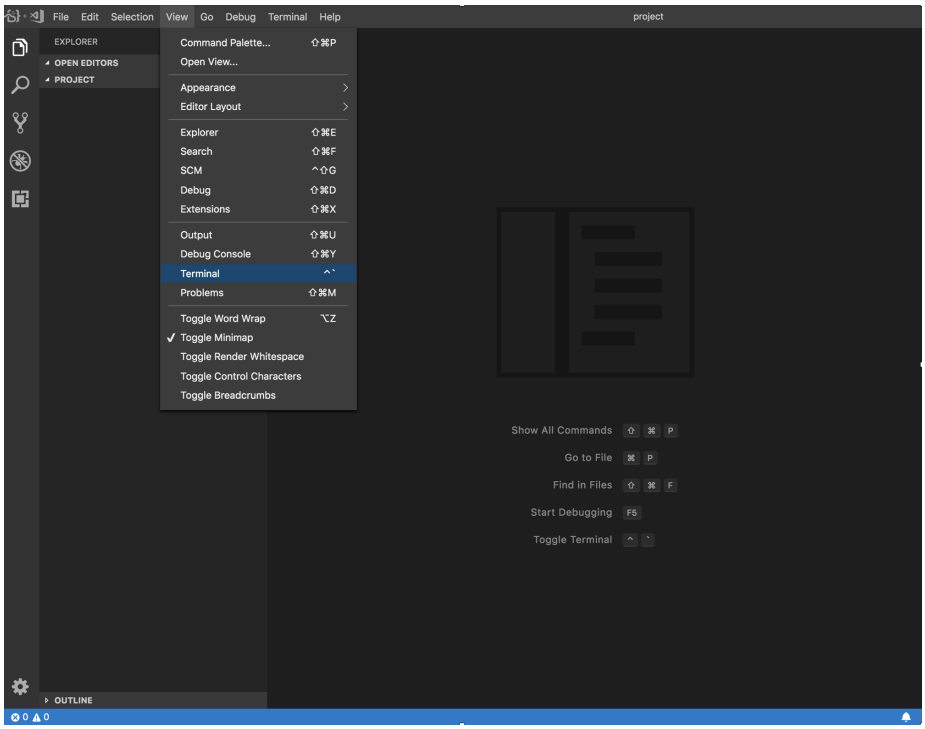

Figure 5: Visual Studio Code Terminal

Final touches

You are now set to work in the Coursera lab environment. Just few other things before you begin this exercise:

- There is a folder called "backup" inside the environment, ignore it for now.
- You can create, edit, and delete files and folders in the Side Bar section.
- The exercise path is /home/coder/project/, open the Terminal and type **cd /home/coder/project/** if you get lost.

Write and run a test program

We are now going to create a program and check we are able to fire up all the tools.

Working in the Coursera labs environment, create a file called main.cpp. Put the following code in there:

```
#include <iostream>

void looper()
{
    printf("In main");
    for(int i=0; i<5;++i)
    {
    printf("Value of i %i \n", i);
    }
}

int main()
{
    printf("In main, calling looper \n");
    looper();
    printf("Back in main again \n");
}
```

Open the Terminal in Visual Studio Code and run the following command:

```
g++ -g main.cpp -o debugme


-g: add debug flags
-o: specify an output
```

Then run the program in normal mode by running this command:

```
./debugme
```

You should see the following output:

```
In main, calling looper
In mainValue of i 0
Value of i 1
Value of i 2
Value of i 3
Value of i 4
Back in main again
```

Start the debugger

Now we'll fire up the debugger:

```
gdb debugme
```

You should see the GDB prompt:

```
(gdb)
```

Try the list command. You should see something like this:

```
(gdb) list
1
2   #include <iostream>
3
4   void looper()
5   {
6       printf("In main");
7       for(int i=0; i<5;++i)
8       {
9           printf("Value of i %i \n", i);
10      }

...
```

Set a breakpoint at line 9, or wherever your printf value of ... statement is:

```
(gdb) break 9
```

Now run the program:

```
(gdb) run
```

You should see something like this:

```
Starting program: /home/matthew/src/bsc-cs/sdd-teaching-materials/worksheets/debugme
In main, calling looper

Breakpoint 1, looper () at main.cpp:9
9       printf("Value of i %i \n", i);
```

Step through the lines

Step past the breakpoint with the next command:

```
(gdb) next
```

Keep running the next command. What do you think is going on here?

Investigate the state of the program

Type run at the gdb prompt and run the program from the start. Make sure you have a breakpoint at the printf command as before.

Once the program stops at the breakpoint, print out the value of i:

```
Breakpoint 1, looper () at main.cpp:9
9       printf("Value of i %i \n", i);
(gdb) print i
```

You should see something like:

```
$1 = 0
```

Are there other variables?

```
(gdb) info locals
```

Where are we in the stack?

```
(gdb) info stack
```

```
#0  looper () at main.cpp:9
#1  0x000055555555520b in main () at main.cpp:16
```

Are there any function arguments?

```
(gdb) info args
```

Try creating a new function that has parameters, and calling it from main. See what info args prints out now.

Conditional breakpoints

Now we are going to use a conditional breakpoint. Re-run your program, using Ctrl-D to exit from GDB, then running the gdb command again on your program. Set a conditional breakpoint:

```
(gdb) break 9 if i > 0
```

(change the 9 to the line on which your printf statement occurs)

Run:

```
(gdb) run
```

Verify that the program only stopped once i reached a certain value.

Watch points

The last technique we will look at is watch points. Change your code to this, so there is a new global variable 'x'.

```
#include <iostream>

int x = 0;

void looper()
{
    printf("In main");
    for(int i=0; i<5;++i)
    {
        printf("Value of i %i \n", i);
        if (i > x) x++;
    }
}

int main()
{
    printf("In main, calling looper \n");
    x = 10;
    looper();
    x++;
    printf("Back in main again \n");
}
```

Recompile and run with gdb:

```
g++ -g main.cpp  -o debugme

gdb debugme
```

Set a watch on x:

```
(gdb) watch x
```

Run:

```
(gdb) run
```

You should see something like this:

```
Hardware watchpoint 1: x

Old value = 0
New value = 10
main () at main.cpp:19
19      looper();
```

To keep running until x changes again:

```
(gdb) continue
```

The debugger should pause the program when x changes again.

Conclusion

Ok that's it. We have now seen several basic GDB operations. If you would like to explore more, you could try running some debug operations on the terriblefall project seen in the unit testing topic.