University of Glasgow | School of Computing Science

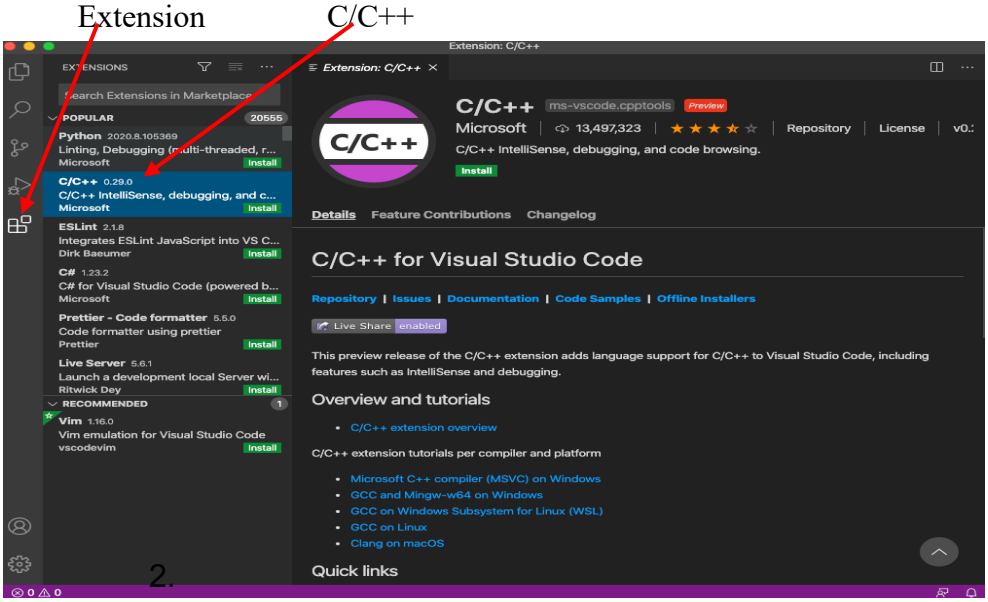
CSC1103 Laboratory/Tutorial 1 : Understanding C

1. Familiarization with C compiler and text editor environment

We will be using Microsoft Visual Studio Code (VS Code) as the text editor environment and using the GNU C Complier (GCC) compiler. If your computing resource has not installed the Microsoft VS Code with C/C++ extension and GCC compiler, please follow the steps below:

a) For Windows Operating System (OS)
   I. Installation of Microsoft VS code
      i. Go to https://code.visualstudio.com/docs/setup/windows to install and set up Microsoft VS code. Follow the installation steps:
         1. Download tDownload the Visual Studio Code installer for Windows.
         2. Once it is downloaded, run the installer (VSCodeUserSetup-{version}.exe). This will only take a minute.
         3. By default, VS Code is installed under C:\users\{username}\AppData\Local\Programs\Microsoft VS Code.

   II. Configuration of Microsoft of VS code for C programming
      i. Install C/C++ extension
         1. Launch the VS code. and click on the extension view icon on the left-hand sidebar. Search for C/C++ extension and click install.



ii. Installation of C/C++ compiler and debugger using GCC complier with Minimalist GNU for Window OS (Mingw-w64) (Mingw) at https://code.visualstudio.com/docs/cpp/config-mingw

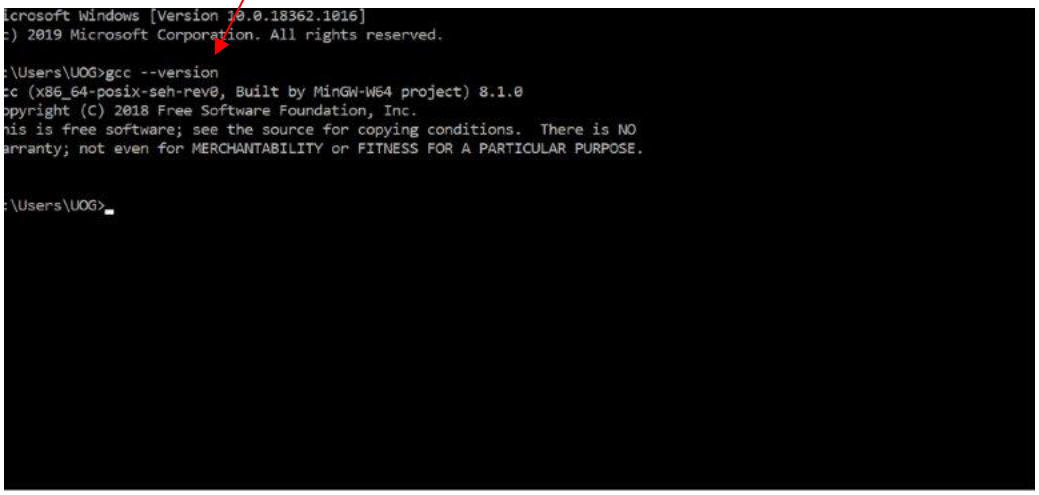
1. Install Mingw-w64 via the MSYS2 website or the direct link ti the installer. Download the Windows Mingw-w64 installer.
2. Run the installer and follow the steps of the installation wizard. For MSYS2, 64-bit windows 8.1 or newer is required.
3. Use the default installation folder and install MinGW. For MSYS2, when completed, ensure **Run MSYS2 now** box is checked and select **Finish**. This will open a MSYS2 terminal window.
4. For MYSY2 terminal, install the MinGW-w64 toolchain by running the following command:

```
pacman -S --needed base-devel mingw-w64-x86_64-toolchain
```

5. Accept the default number of packages in the toolchain group by pressing **Enter.** Enter **Y** when prompted to proceed with installation.
6. Add the path to your Mingw-w64 bin folder to the Windows PATH environment variable by using the following steps:
    a. In the Windows search bar, type 'settings' to open your Windows Settings.
    b. Search for **Edit environment variables for your account**.
    c. In your **User variables**, choose the Path variable and then select **Edit**.
    d. Select **New** and add the Mingw-w64 path to the system path. The exact path depends on which version of Mingw-w64 you have installed and where you installed it. If you used the default settings above to install Mingw-w64, then add this to the path: C:\msys64\mingw64\bin (* if differ, you need to search your folder where this bin folder resides in your computer and cut and paste this path)
    e. Select **OK** to save the updated PATH. You will need to reopen any console windows for the new PATH location to be available.
7. Restart the computer to allow the environment path of GCC compiler environment path to take effect.
8. To check that your Mingw-w64 tools are correctly installed and available, open a new Command Prompt by typing

command prompt in the Windows search bar. A command prompt will appear and type

```
gcc --version
```



b) For Mac Operating System (OS)

I. Installation of Microsoft VS code

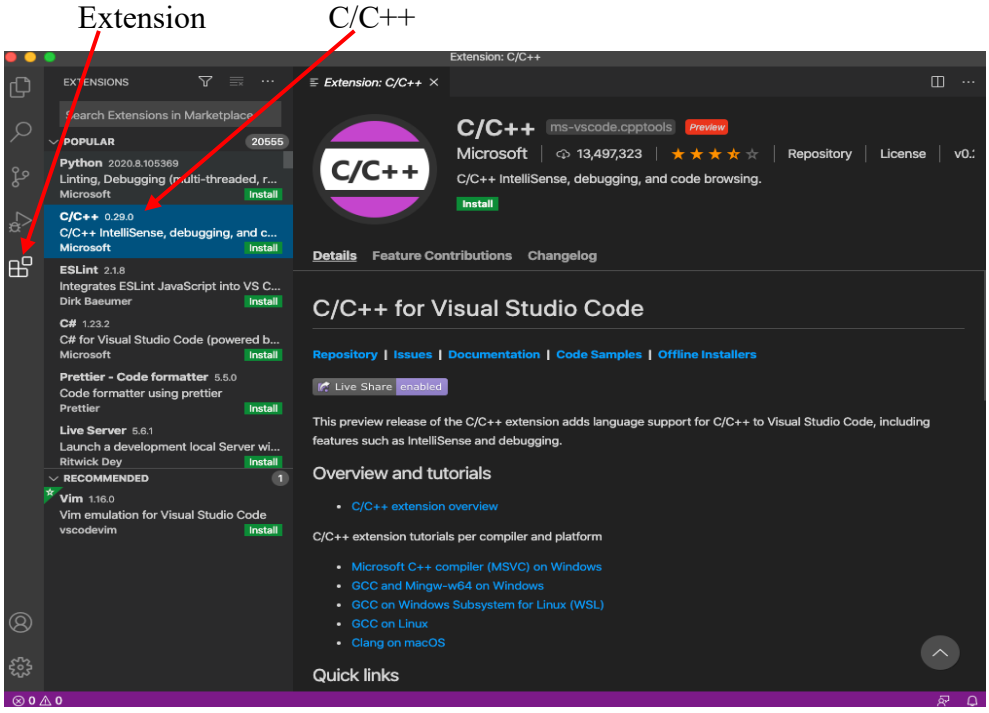
i. Go to https://code.visualstudio.com/docs/setup/mac to install and set up Microsoft VS code. Followed the installation steps in the page namely

1. Download Visual Studio Code for macOS.
2. Open the browser's download list and locate the downloaded archive.
3. Select the 'magnifying glass' icon to open the archive in Finder.
4. Drag Visual Studio Code.app to the Applications folder, making it available in the macOS Launchpad.
5. Optional : Add VS Code to your Dock by launching the VS code in MacOS Launchpad and right-clicking on the icon to bring up the context menu and choosing **Options**, **Keep in Dock**.

II. Configuration of Microsoft of VS code for C programming

i. Install C/C++ extension

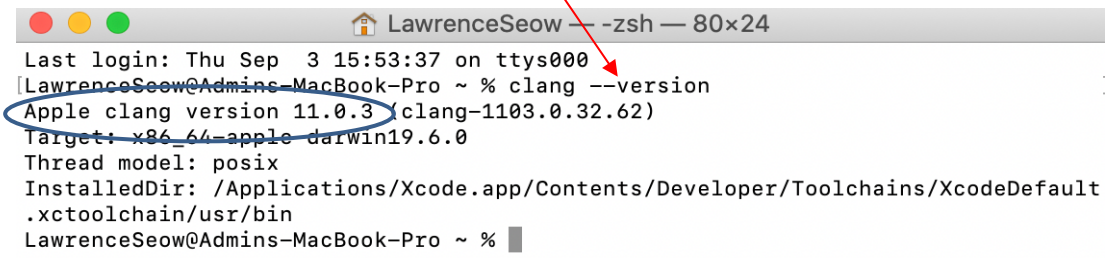
1. Launch the VS code. and click on the extension view icon on the left-hand sidebar. Search for C/C++ extension and click install.



ii. Installation of C/C++ compiler and debugger using GCC complier with Clang for Mac

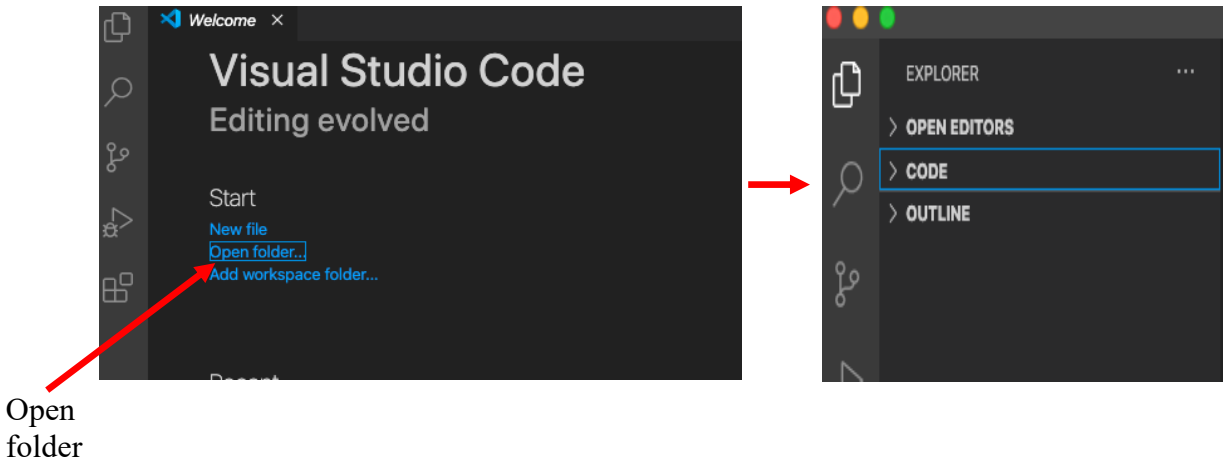
1. Go to apple store apps in your Mac , search for XCode , download and install (need you to login into your apple ID account). Launch XCode in your launchpad, accept the apple agreement and install all components.
2. Open up a MacOS terminal window (open up launch pad and go to other folder) to verify Clang compiler has been installed.
   Enter the following command at the command prompt.

```
% Clang --version
```

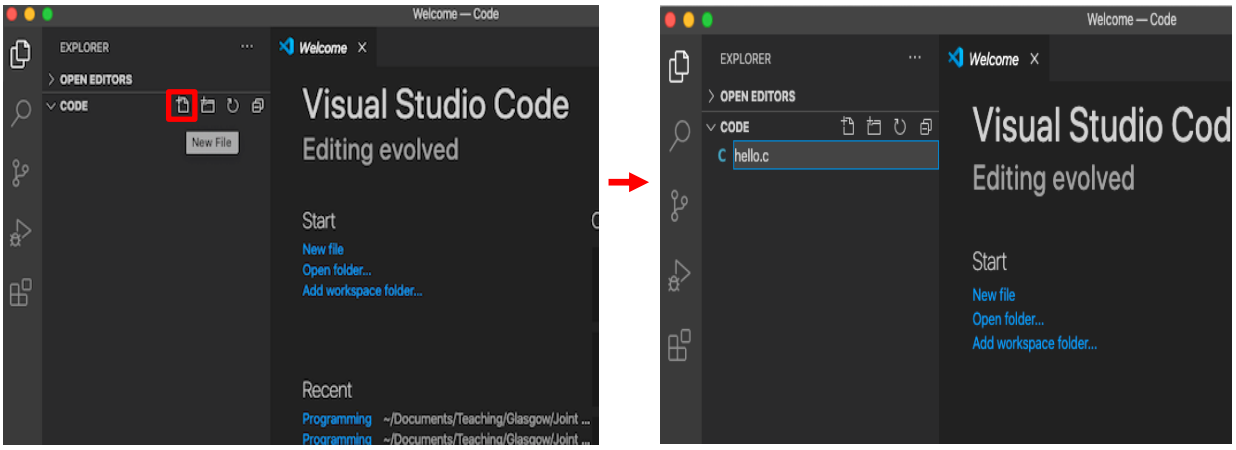


2. *Write a Simple Hello program* and setup one-time configuration

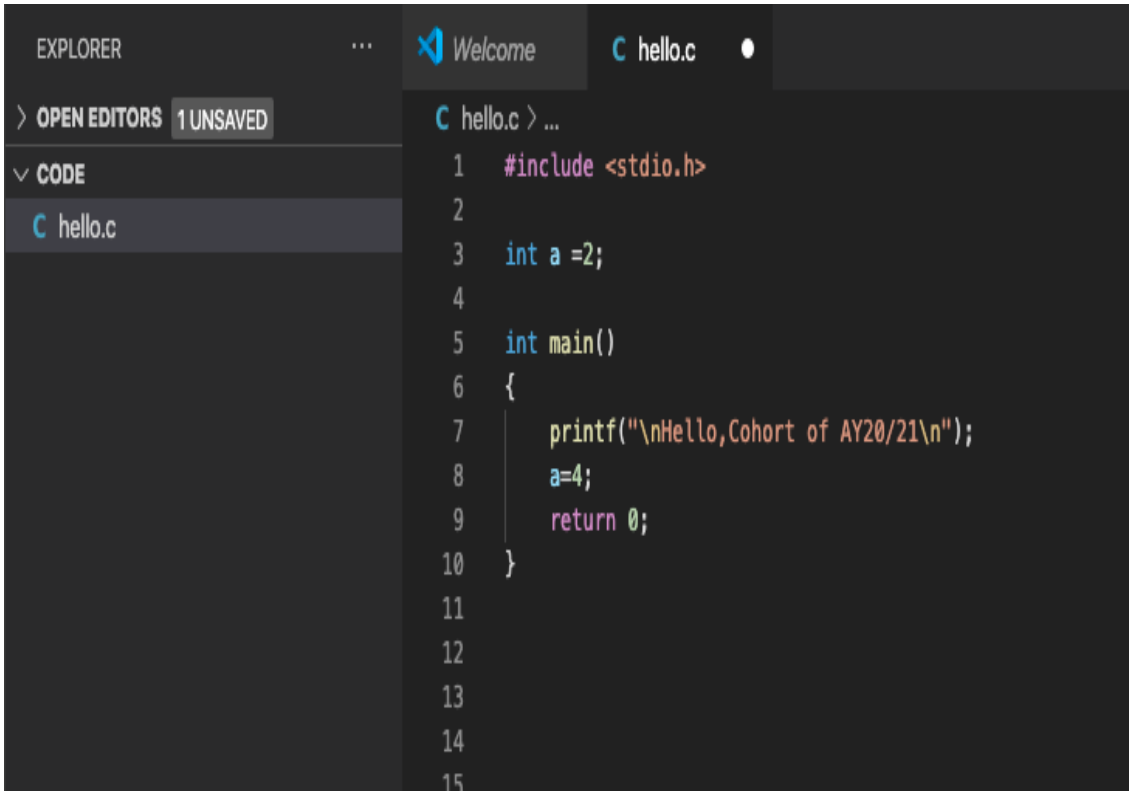
a) Create your preferred folder so that all future programming codes will be stored in this folder. For example, a folder called code under a main folder called programming_methodology. Open the VS code editor and click open folder and pointed to the desired folder(in this case programming_methodology->code)



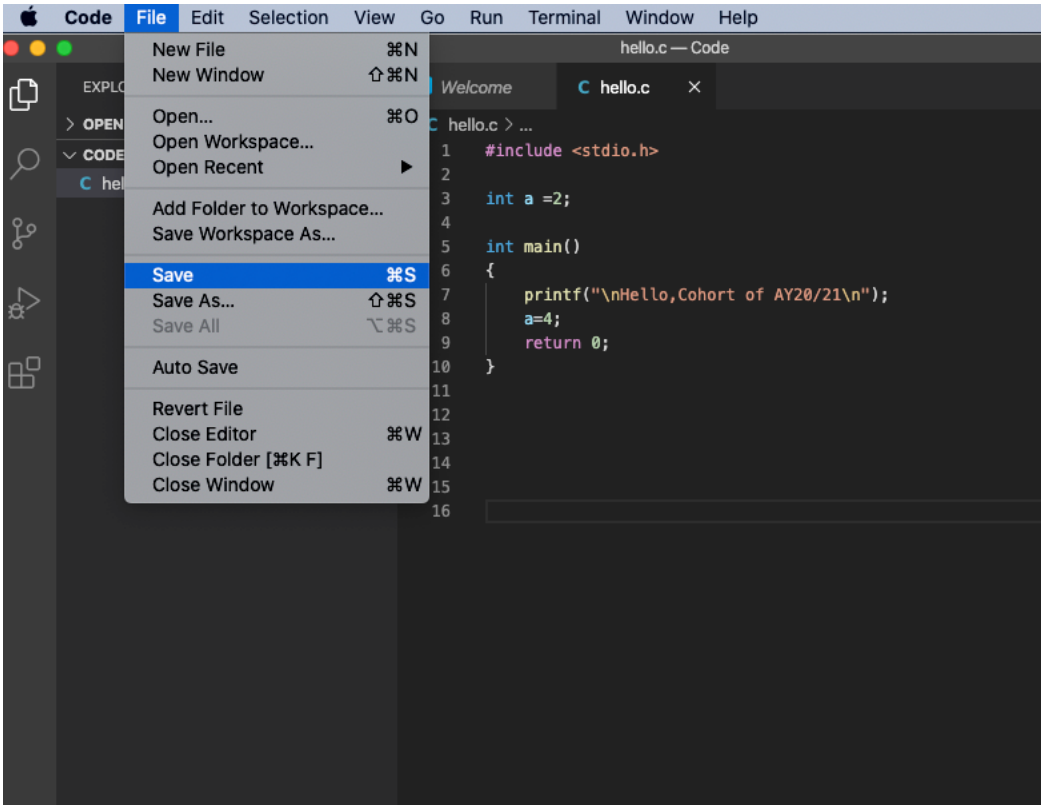
b) In the file explorer title bar, select **New File** and name the file as hello.c.



c) Paste or type in the following C source code

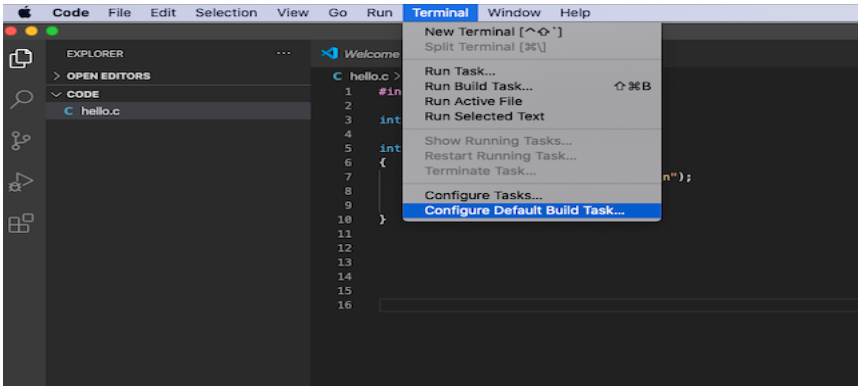


File **Save** the code.

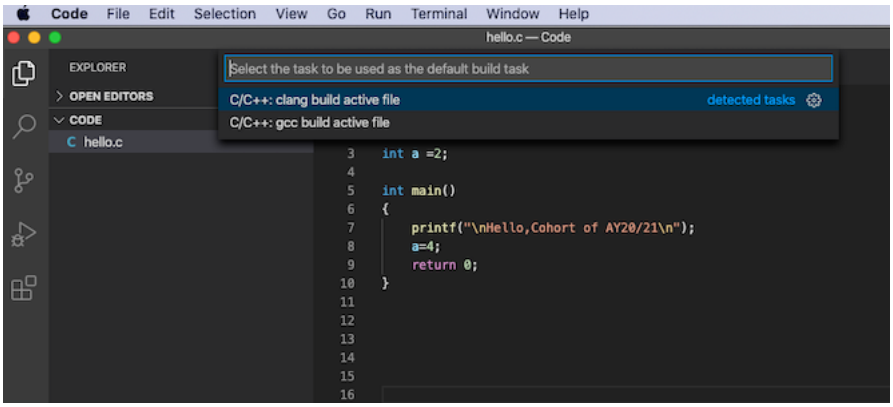


d) Build the compilation environment

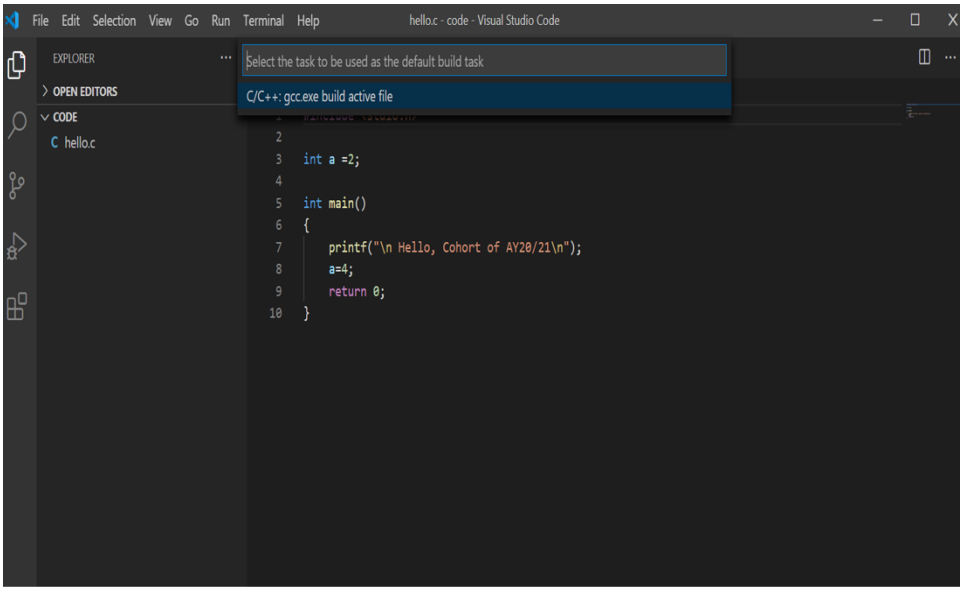
I. Create tasks.json file to tell VS code how to build(compile) the program. This will invoke the Clang C/C++ compiler for MAC OS and GCC for Window OS to create an executable file from source code (one-time effort for all codes under the folder). Go **Terminal->Configure Default Build Task**



A dropdown will appear listing various predefined build tasks for the compilers that VS Code found on your machine. Choose **C/C++ clang build active file** for MAC OS and **C/C++ gcc.exe build active file** for window OS to build the file that is currently displayed (active) in the editor.

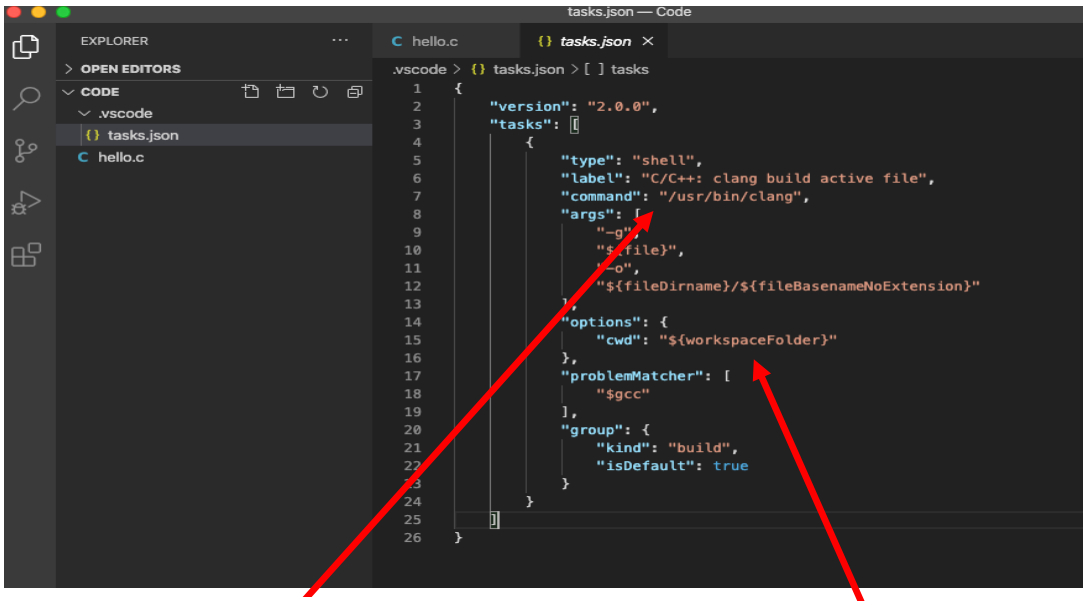


Mac OS



Windows OS

This will create a tasks.json file in the .vscode folder and open it in the editor.

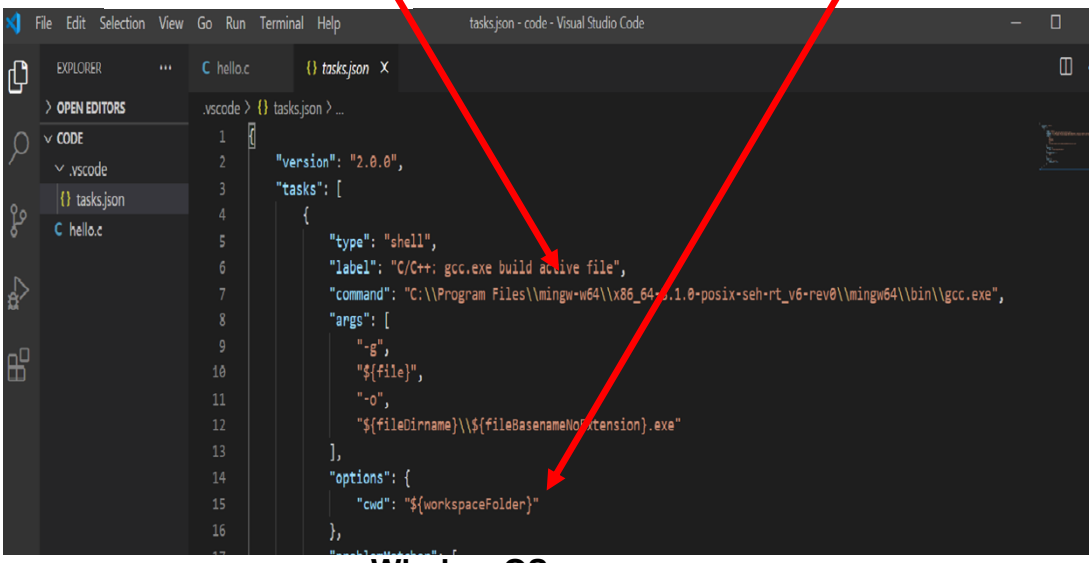


Clang Compiler used

Compile files in working directory

Mac OS

GCC Compiler used



Window OS

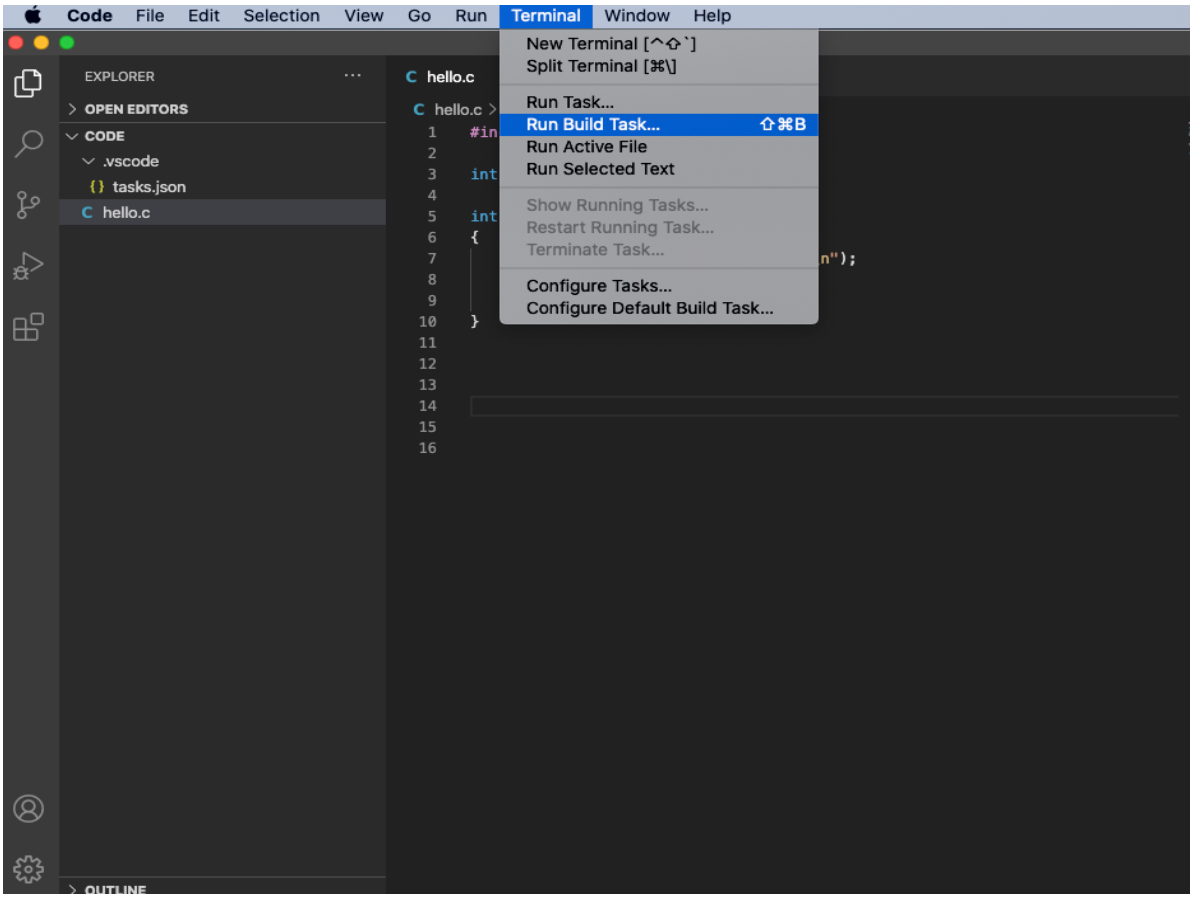
If cant task .jason, go View->Command Palette->C/C++ :Edit Configuration (UI)
c_cpp_properties.json will be formed.
Then go Terminal->Run Build Task after that for method 1 as usual

e) Running the Build (Compile) to run the hello.c program

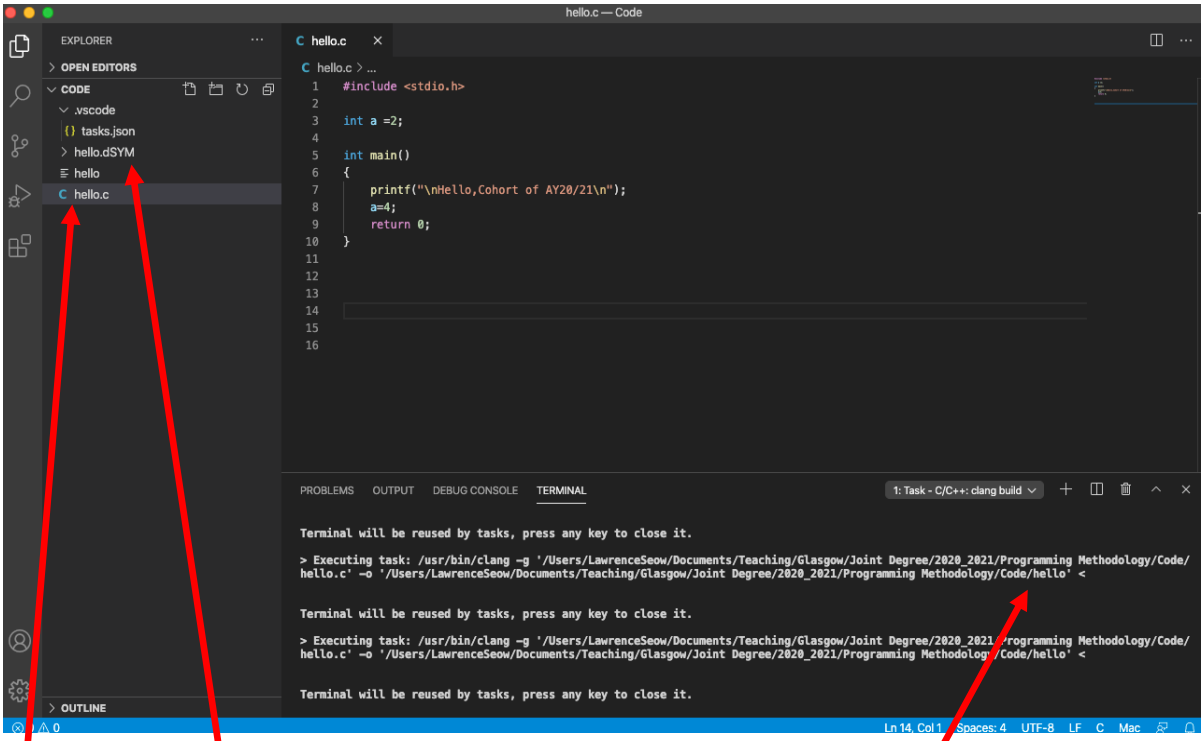
I. **Method 1**

Step 1: Go back to hello.c. Because we want to build hello.c, it is important that this file be the one that is active in the editor for the next step.

Step 2: To run the build task to generate hello.exe that you defined in tasks.json, press **Terminal->Run Build Task**



Step 3 : When the task starts, you should see the Integrated Terminal window appear below the code editor. After the task completes, the terminal shows output from the compiler that indicates whether the build succeeded or failed. For a successful Clang build, the output looks something like this below with two files created namely hello.exe and hello.dSYM(MAC OS Only):

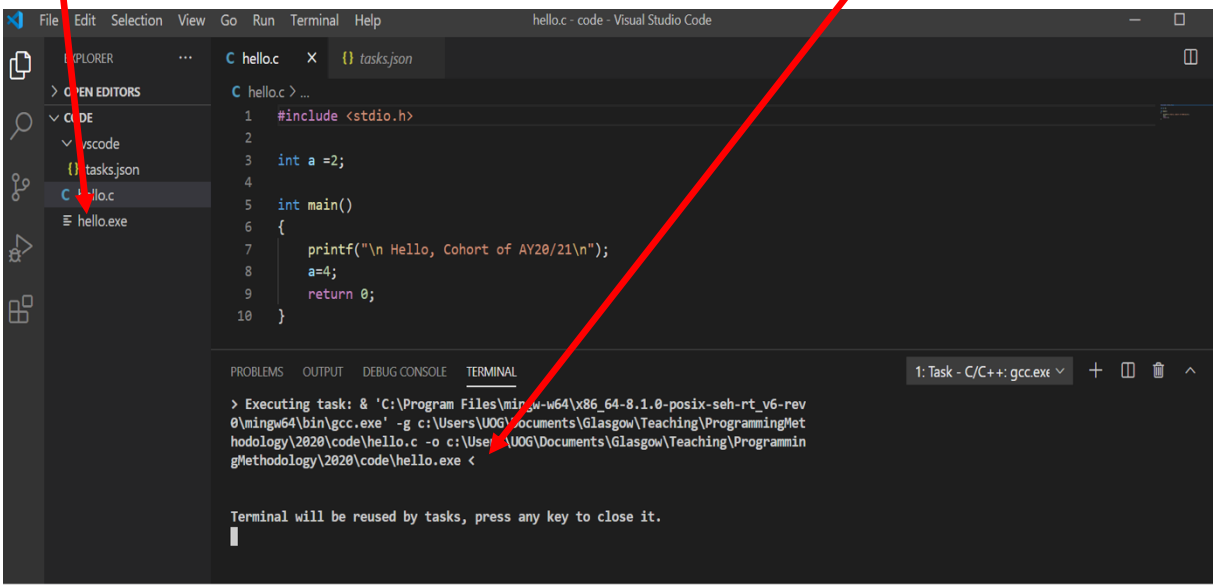


hello.exe

hello.dSYM

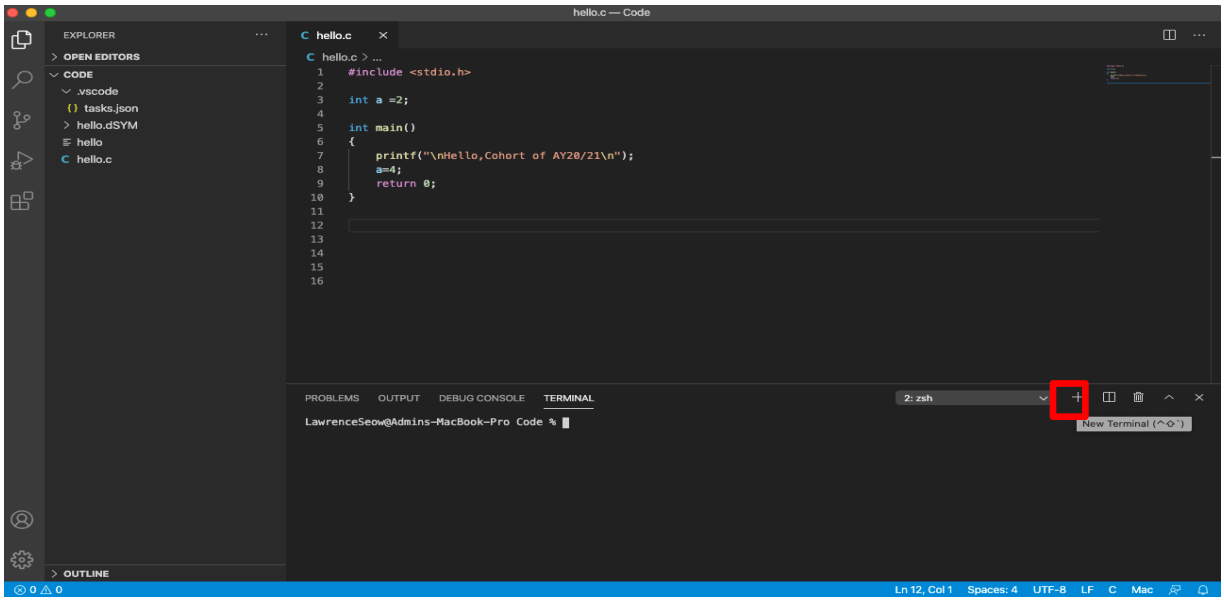
Terminal window show successfully build hello.exe

MAC OS

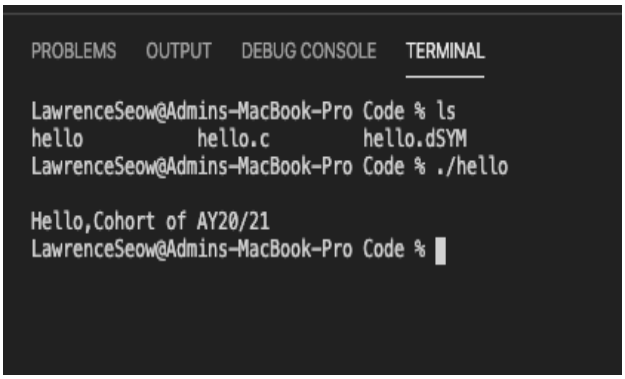


Window OS

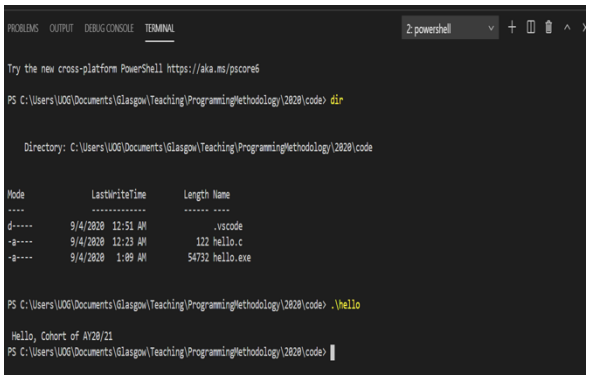
Step 4: Create a new terminal using the **+** button and you'll have a new terminal with the code folder as the working directory.



Step 5 : MAC OS : Run ls and you should now see the executable hello along with the debugging file (hello.dSYM). Type the following at the terminal command prompt ./hello. The hello message should appear.
Window OS : Run dir and you should now see the executable hello . Type the following at the terminal command prompt .\hello. The hello message should appear.



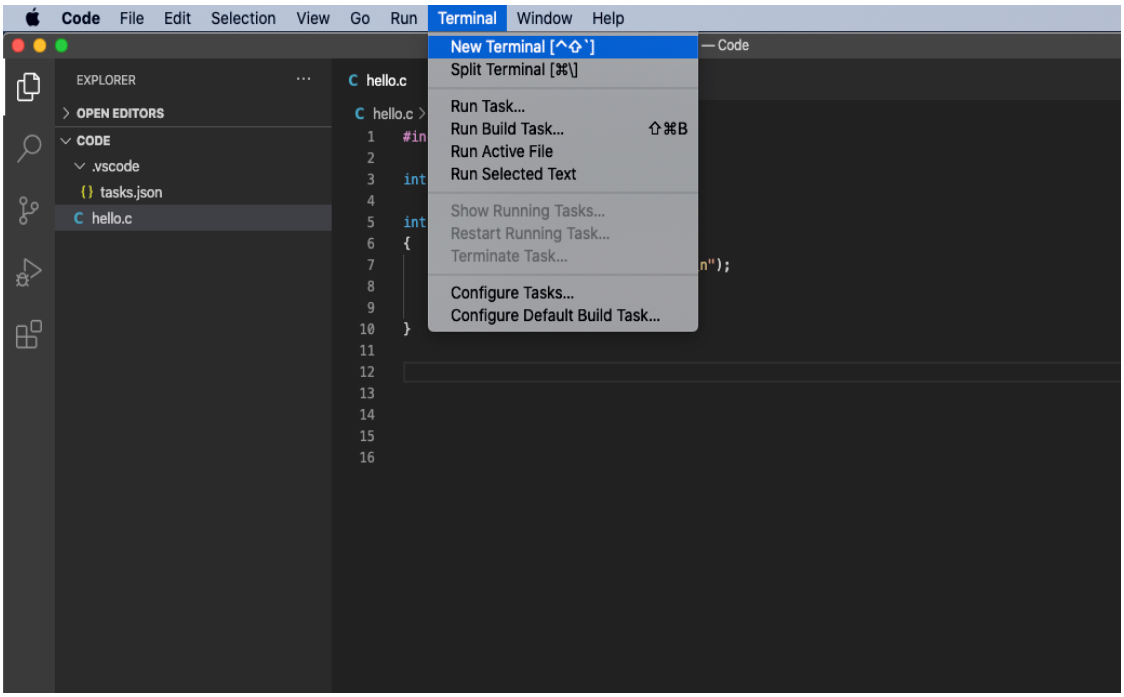
MAC OS



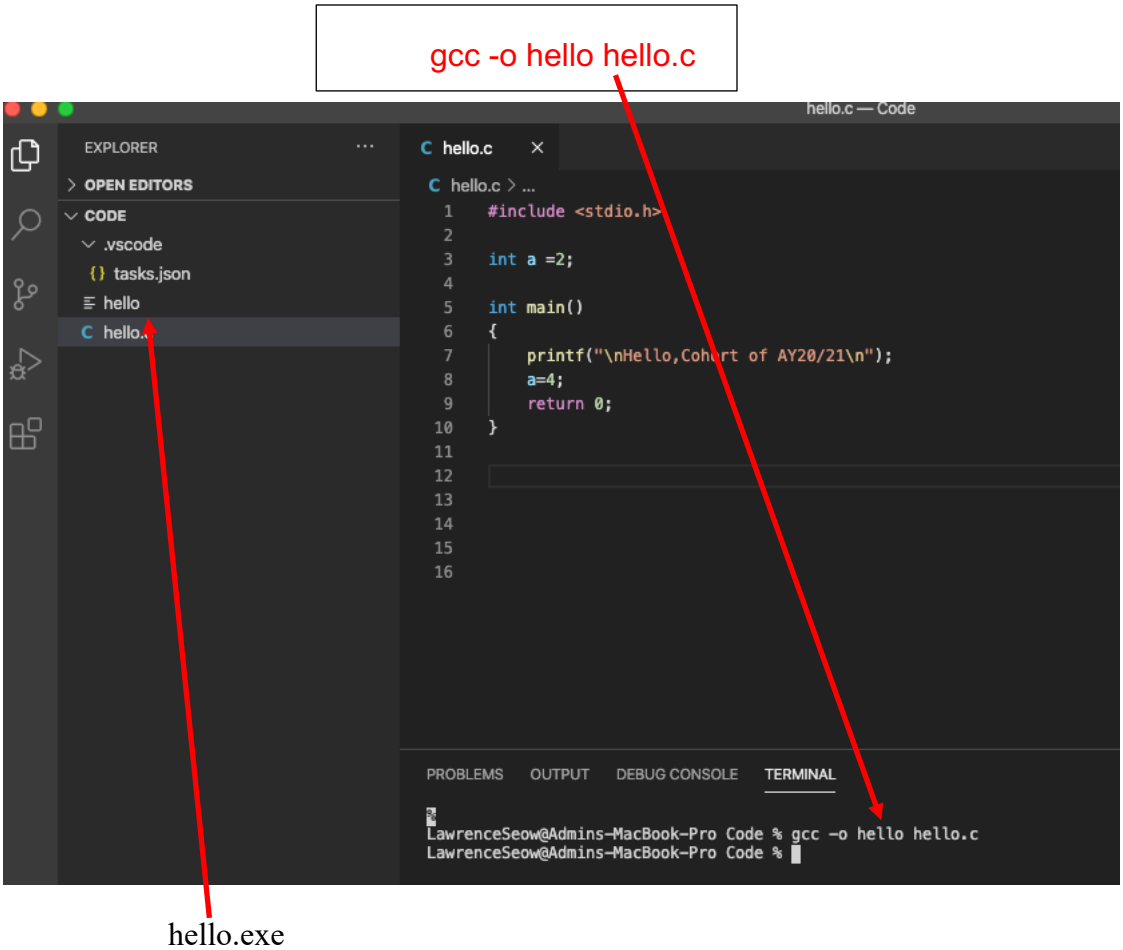
Window OS

II. Method 2

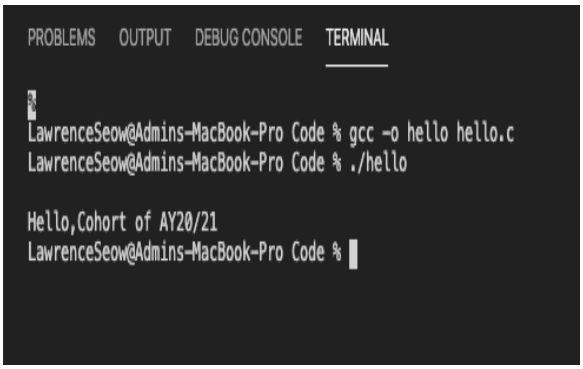
Step 1: Go to drop down menu **Terminal->New Terminal** to generate new terminal if not exist.



Step 2 : Type the following at the Terminal command prompt to compile and generate the hello.exe



Step 3: Run the hello.exe at the terminal command prompt ./hello or .\hello. The hello message should appear.

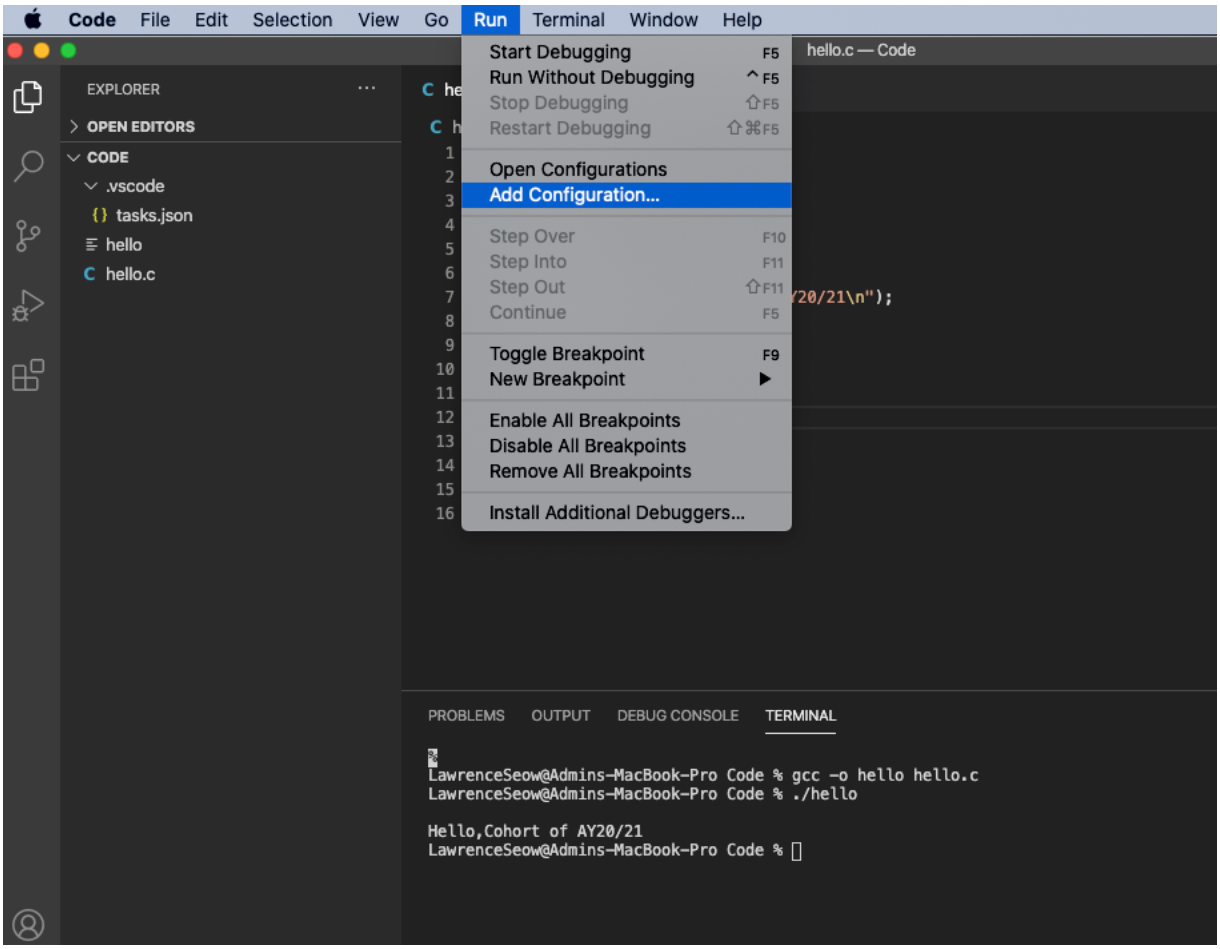


MAC OS

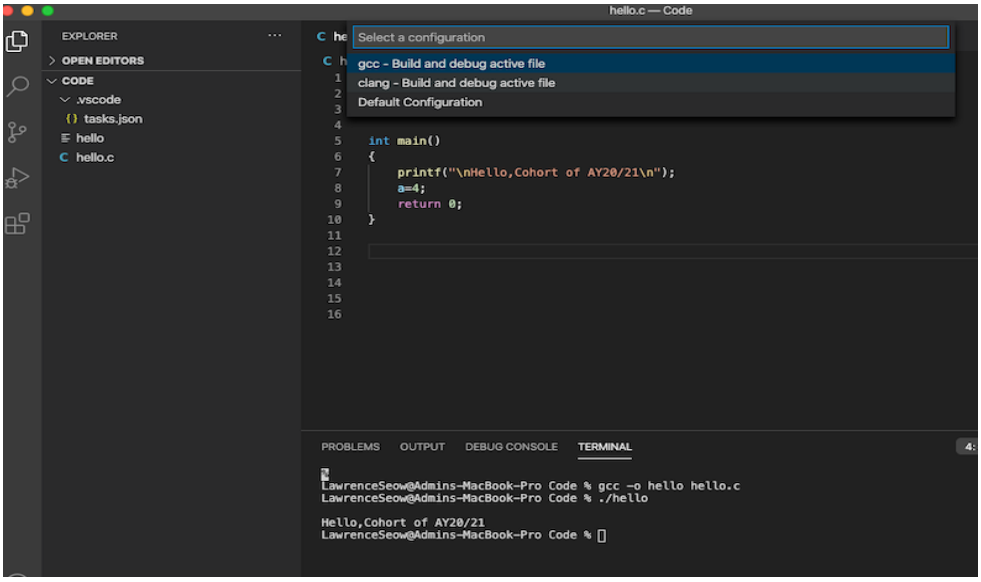
Window OS

f) Create Debug environment. Sometime, it is good to create a debug environment to step tracing to troubleshoot your code. You can create a one-time debug environment for all the codes under the same folder.

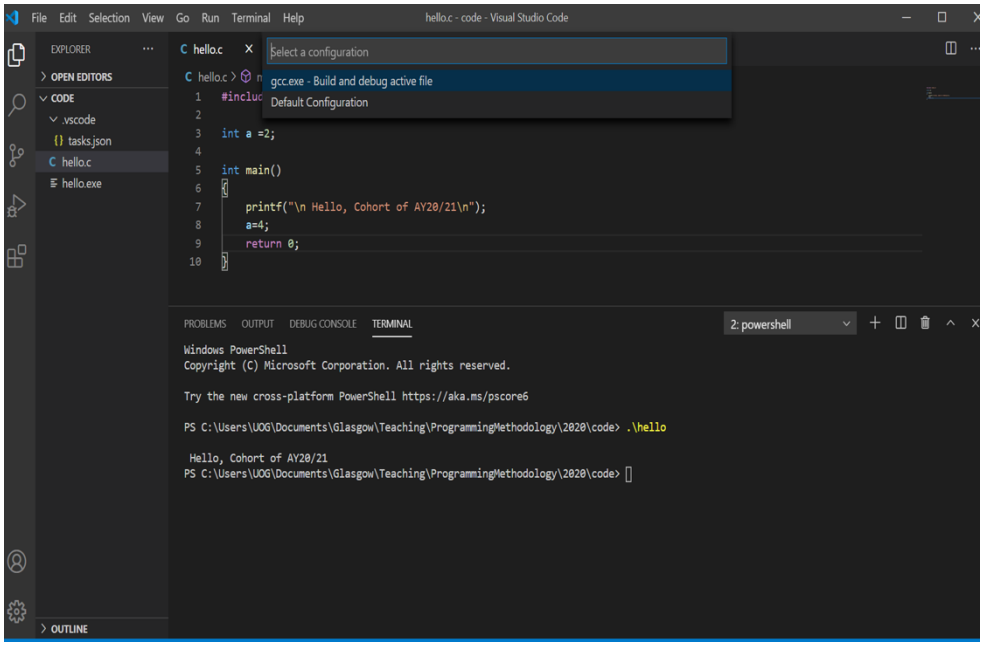
Step 1 : Create the launch.json file to configure VS Code to launch the LLDB debugger when you press **F5** or **Run-> Start Debugging** to debug the program. From the main menu, choose **Run** -> **Add Configuration...** and then choose **C++ (GDB/LLDB)**.



You'll then see a dropdown for predefined debugging configurations. Choose **clang - Build and debug active file** for MAC OS and **gcc.exe - Build and debug active file** for Window OS.

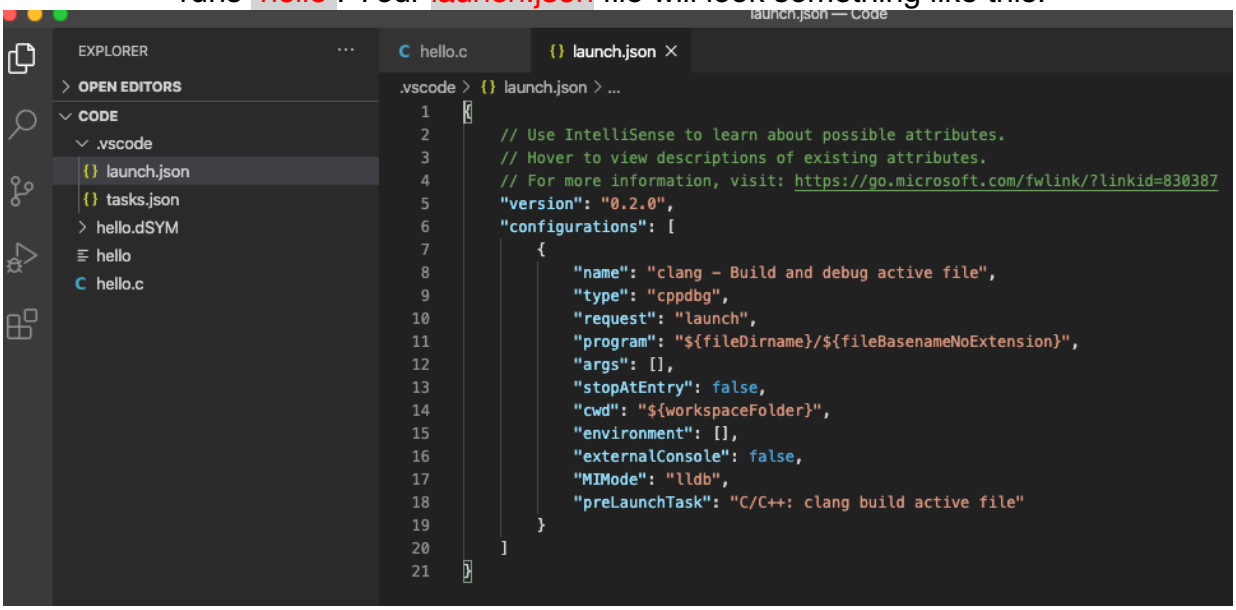


MAC OS

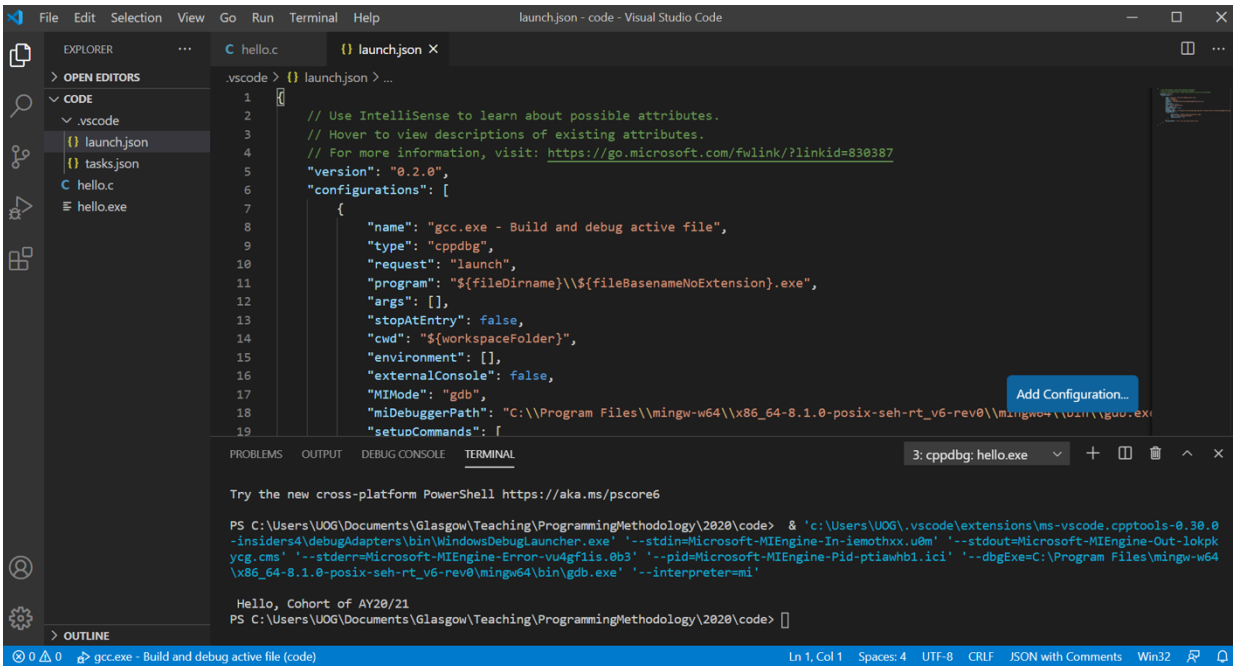


Window OS

VS Code creates a launch.json file, opens it in the editor, and builds and runs 'hello'. Your launch.json file will look something like this:

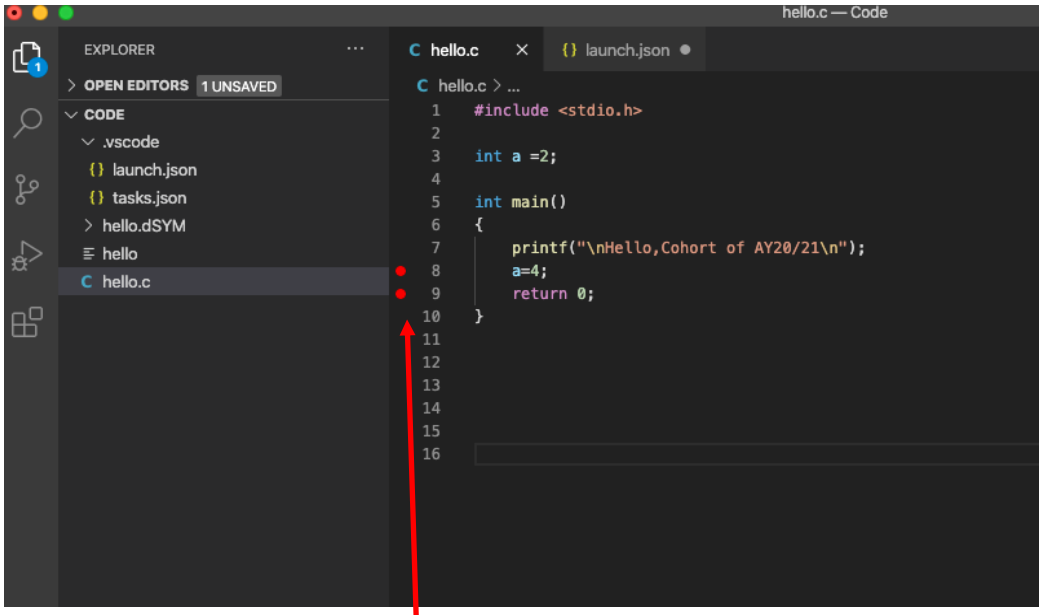


Mac OS



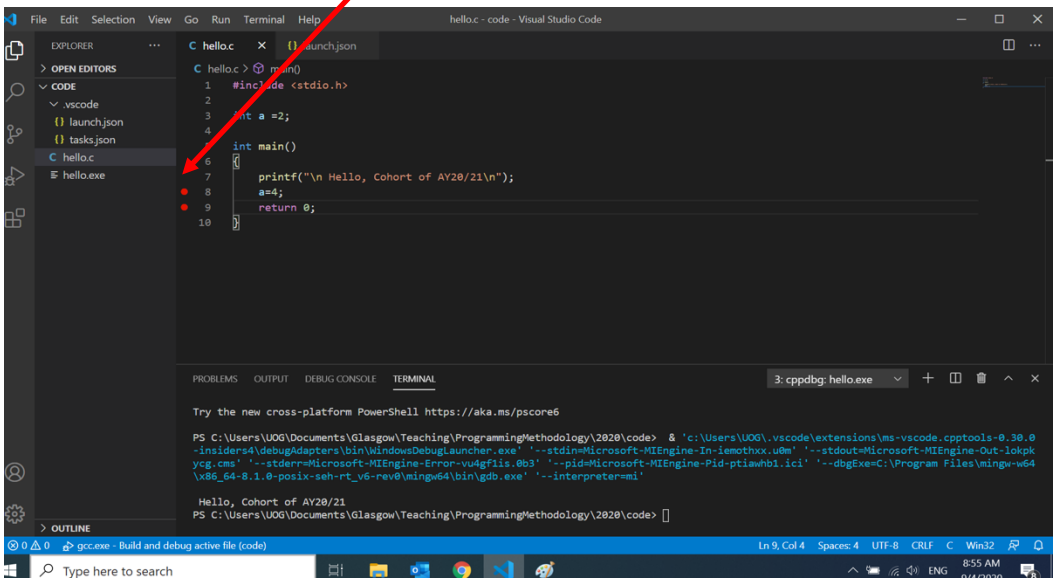
Window OS

Step 2: Go back to hello.c so that it is the active file in the editor. This is important because VS Code uses the active file to determine what you want to debug. Create two toggle points at line 8 and line 9 to allow the see the usefulness of debug mode. Go to line 8 and line 9 and choose **Run->Toggle BreakPoint**



Breakpoint

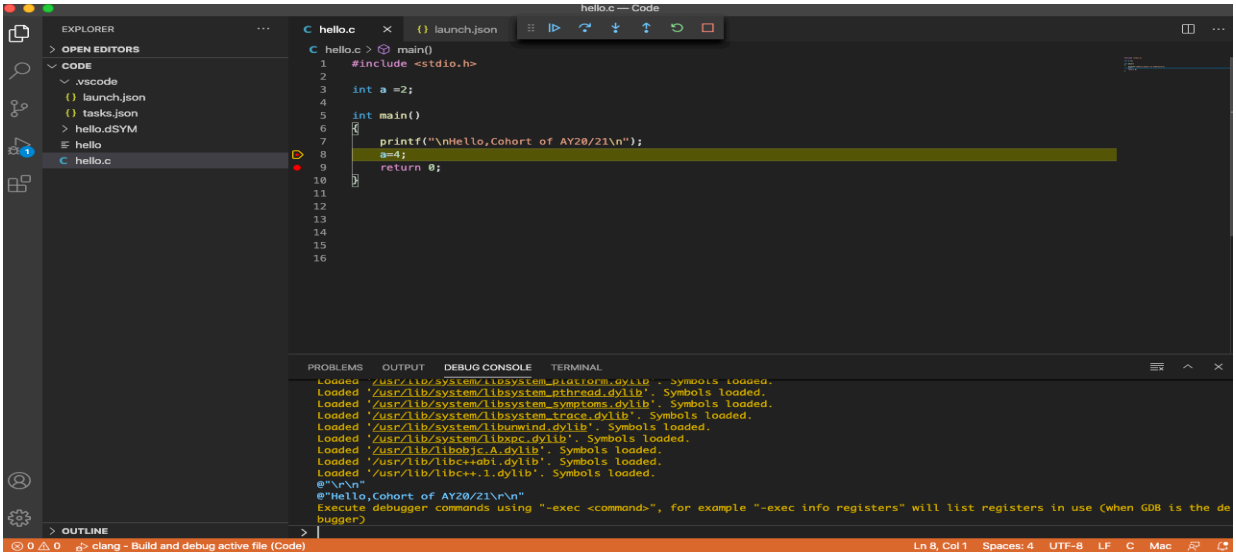
MAC OS



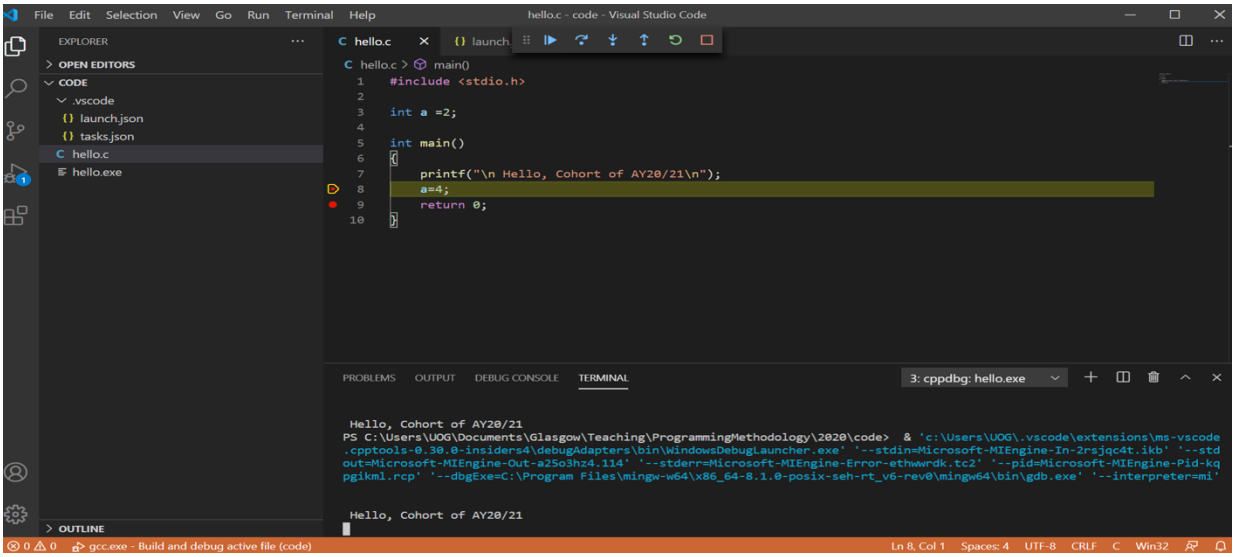
Window OS

Step 3: Press **F5** or from the main menu choose **Run -> Start Debugging**. Before you start stepping through the source code, let's take a moment to notice several changes in the user interface:

- The Integrated Terminal appears at the bottom of the source code editor. In the **DEBUG CONSOLE Output** tab, you see output that indicates the debugger is up and running.
- The program stops at the breakpoint where a=4.

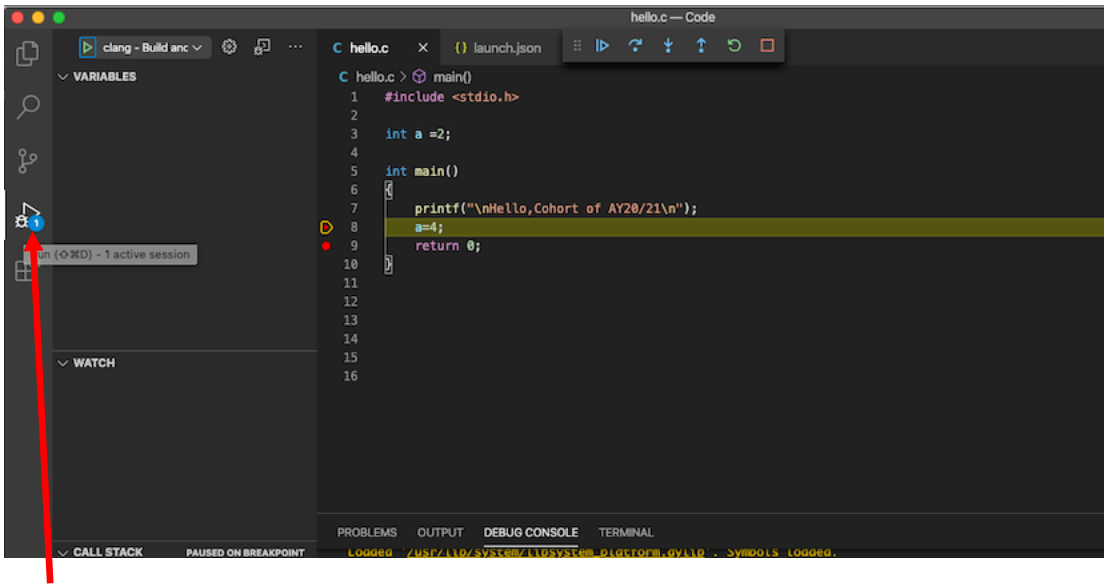


MAC OS



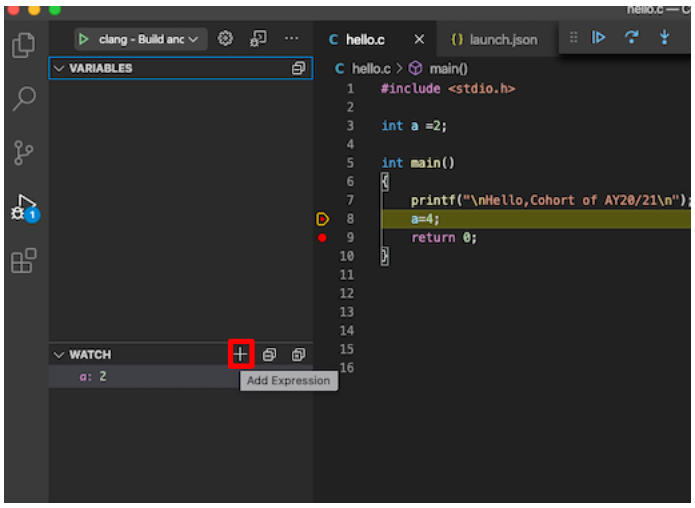
Window OS

Step 4: Double click on the **Run** view



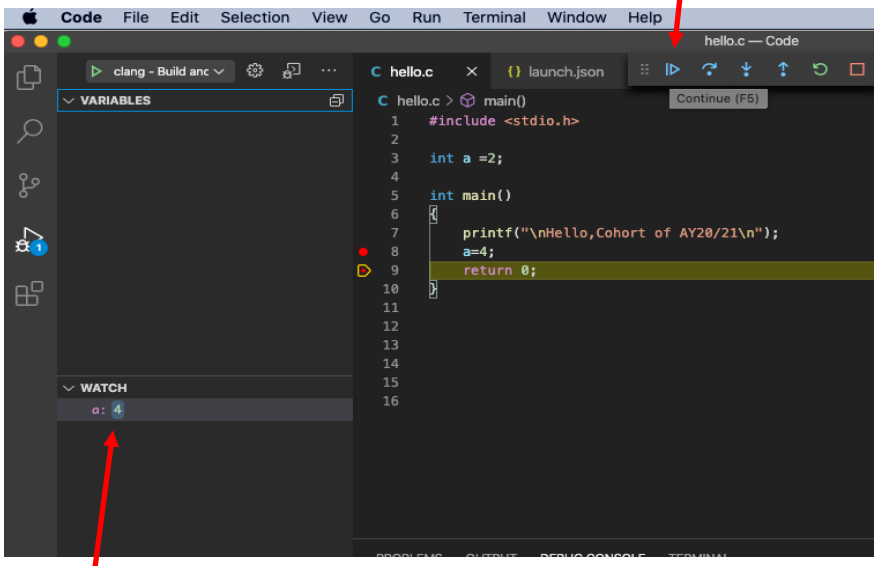
Run view

At the **WATCH** view , **Add Expression** and type **a** for variable a to observe how the variable a change value from 2 to 4



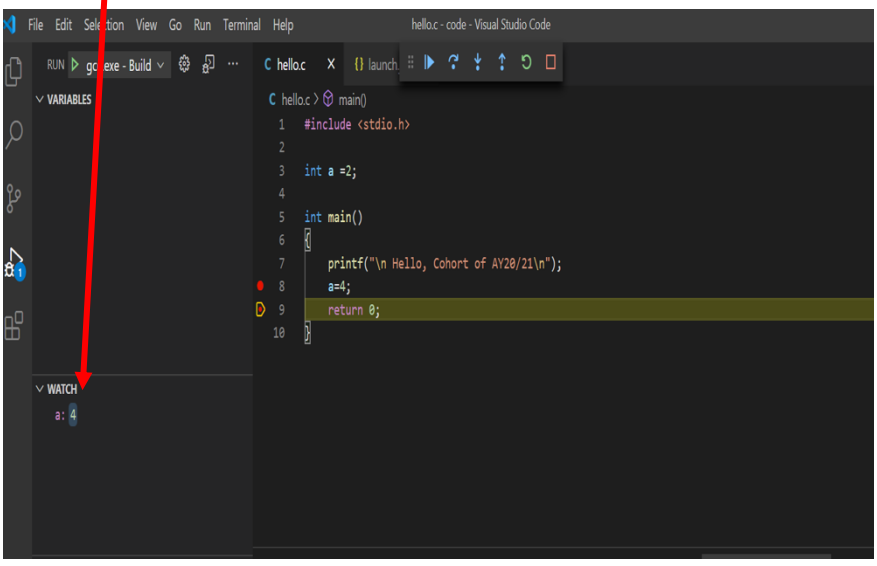
Step 5: Click on the **continue** button or F5 to step through to the next toggle point and observe the variable a change from 2 to 4. Press **F5** to complete the program run.

Continue button



variable a changes from 2 to 4

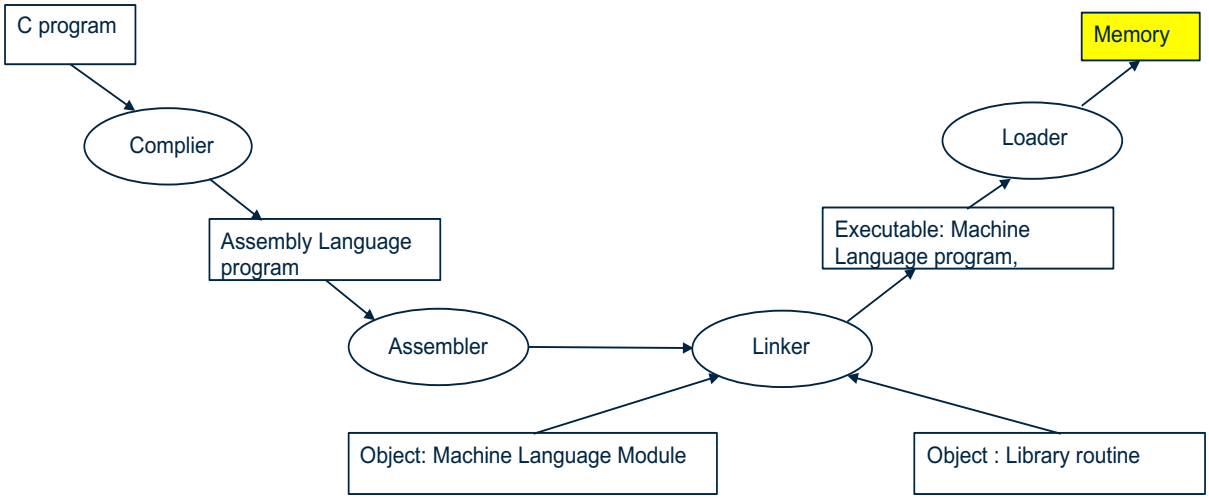
MAC OS



Window OS

3. *Understanding the Edit-Compile-Link-Execute (ECLE) Process of a C Program*

The full cycle of ECLE is depicted as followed



In general, the ECLE process of a C program are mainly
a) Preprocessing
b) Compilation
c) Assembly
d) Linking

Pre-processing

In this stage, pre-processor directives that start with a # character are interpreted by the *pre-processor*. Before interpreting, the preprocessor perform initial processing such as joining continued lines (lines ending with a \) and stripping comments.

To see the effect of the preprocessing stage, type the following at the terminal

```
gcc -E hello.c
```

using the same hello.c program above, the preprocessor will produce the contents of the stdio.h header file joined with the contents of the hello.c file, stripped free from its leading comment as shown below

stdio.h code

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    2: powershell

cale_t _Locale,...);
  __attribute__ ((__dllimport__)) int __attribute__((__cdecl__)) _snwscanf_l(const wchar_t *_Src,size_t _MaxCount,const wc
har_t *_Format,_locale_t _Locale,...);
  __attribute__ ((__dllimport__)) int __attribute__((__cdecl__)) _wscanf_l(const wchar_t *_Format,_locale_t _Locale,...);









  __attribute__ ((__dllimport__)) size_t __attribute__((__cdecl__)) _fread_nolock_s(void *_DstBuf,size_t _DstSize,size_t _ElementSize,size
_t _Count,FILE *_File);
# 1398 "C:/Program Files/mingw-w64/x86_64-8.1.0-posix-seh-rt_v6-rev0/mingw64/x86_64-w64-mingw32/include/stdio.h" 2 3

# 1 "C:/Program Files/mingw-w64/x86_64-8.1.0-posix-seh-rt_v6-rev0/mingw64/x86_64-w64-mingw32/include/_mingw_print_pop.h" 1 3
# 1400 "C:/Program Files/mingw-w64/x86_64-8.1.0-posix-seh-rt_v6-rev0/mingw64/x86_64-w64-mingw32/include/stdio.h" 2 3
# 2 "hello.c" 2


# 3 "hello.c"
int a =2;

int main()
{
    printf("\n Hello, Cohort of AY20/21\n");
    a=4;
    return 0;
}
PS C:\Users\UOG\Documents\Glasgow\Teaching\ProgrammingMethodology\2020\code>
```

Window OS

stdio.h code

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    1: zsh

# 407 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
# 1 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include/secure/_stdio.h" 1 3 4
# 31 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include/secure/_stdio.h" 3 4
# 1 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include/secure/_common.h" 1 3 4
# 32 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include/secure/_stdio.h" 2 3 4
# 42 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include/secure/_stdio.h" 3 4
extern int __sprintf_chk (char * restrict, int, size_t,
      const char * restrict, ...);
# 52 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include/secure/_stdio.h" 3 4
extern int __snprintf_chk (char * restrict, size_t, int, size_t,
      const char * restrict, ...);









extern int __vsprintf_chk (char * restrict, int, size_t,
      const char * restrict, va_list);









extern int __vsnprintf_chk (char * restrict, size_t, int, size_t,
       const char * restrict, va_list);
# 408 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/usr/include/stdio.h" 2 3 4
# 2 "hello.c" 2

int a =2;

int main()
{
    printf("\nHello,Cohort of AY20/21\n");
    a=4;
    return 0;
}
LawrenceSeow@Admins-MacBook-Pro Code %
```

MAC OS

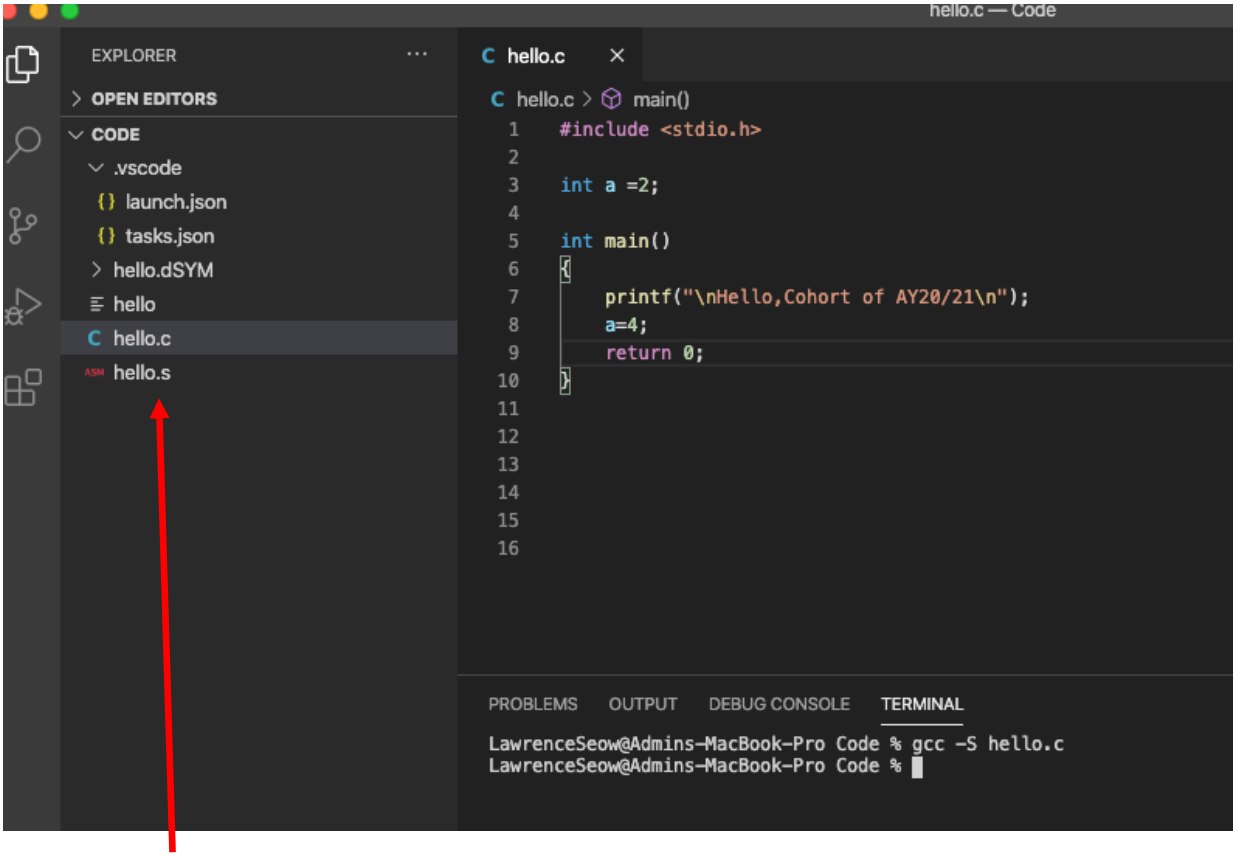
Compilation

In this stage, the preprocessed code is translated to *assembly instructions* specific to the target processor architecture. Some compilers also supports the use of an integrated assembler, in which the compilation stage generates *machine code* directly, avoiding the overhead of generating the intermediate assembly instructions and invoking the assembler.

To generate the assembly language program , type at the terminal prompt

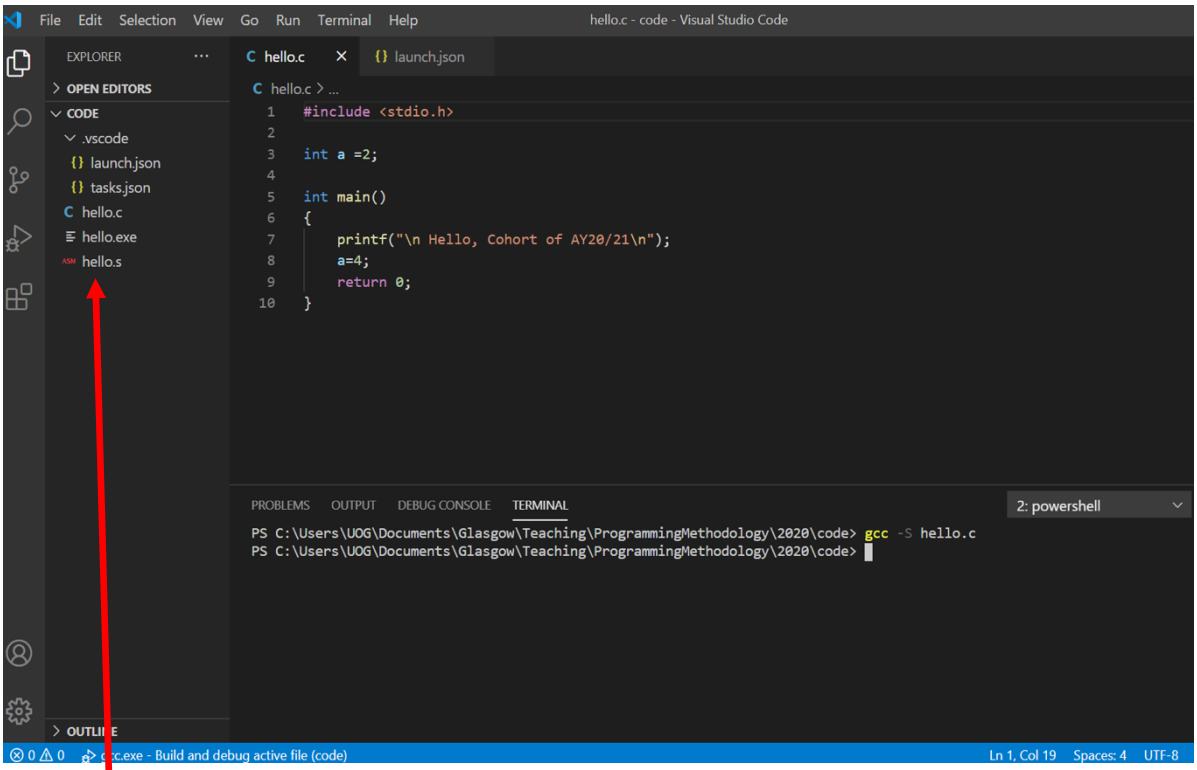
```
gcc -S hello.c
```

This will create a file named hello.s, containing the generated assembly instructions.



hello.s file created

MAC OS



hello.s file created

Window OS

Click on the hello.s file to see the assembly code

hello.c hello.s

hello.s

```
1       .section    __TEXT,__text,regular,pure_instructions
2       .build_version macos, 10, 15    sdk_version 10, 15, 6
3       .globl  _main                   ## -- Begin function main
4       .p2align    4, 0x90
5   _main:                                  ## @main
6       .cfi_startproc
7   ## %bb.0:
8       pushq   %rbp
9       .cfi_def_cfa_offset 16
10      .cfi_offset %rbp, -16
11      movq    %rsp, %rbp
12      .cfi_def_cfa_register %rbp
13      subq    $16, %rsp
14      movl    $0, -4(%rbp)
15      leaq    L_.str(%rip), %rdi
16      movb    $0, %al
17      callq   _printf
18      xorl    %ecx, %ecx
19      movl    $4, _a(%rip)
20      movl    %eax, -8(%rbp)          ## 4-byte Spill
21      movl    %ecx, %eax
22      addq    $16, %rsp
23      popq    %rbp
24      retq
25      .cfi_endproc
26                                      ## -- End function
27      .section    __DATA,__data
28      .globl  _a                      ## @a
29      .p2align    2
30  _a:
31      .long   2                       ## 0x2
32
33      .section    __TEXT,__cstring,cstring_literals
34  L_.str:                             ## @.str
35      .asciz  "\nHello,Cohort of AY20/21\n"
36
37
38  .subsections_via_symbols
39
```

MAC OS

hello.c hello.s

hello.s

```
1       .file   "hello.c"
2       .text
3       .globl  a
4       .data
5       .align 4
6   a:
7       .long   2
8       .def    __main; .scl    2;  .type   32; .endef
9       .section .rdata,"dr"
10  .LC0:
11      .ascii "\12 Hello, Cohort of AY20/21\0"
12      .text
13      .globl  main
14      .def    main;   .scl    2;  .type   32; .endef
15      .seh_proc   main
16  main:
17      pushq   %rbp
18      .seh_pushreg    %rbp
19      movq    %rsp, %rbp
20      .seh_setframe   %rbp, 0
21      subq    $32, %rsp
22      .seh_stackalloc 32
23      .seh_endprologue
24      call    __main
25      leaq    .LC0(%rip), %rcx
26      call    puts
27      movl    $4, a(%rip)
28      movl    $0, %eax
29      addq    $32, %rsp
30      popq    %rbp
31      ret
32      .seh_endproc
33      .ident  "GCC: (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0"
34      .def    puts;   .scl    2;  .type   32; .endef
35
```

Window OS

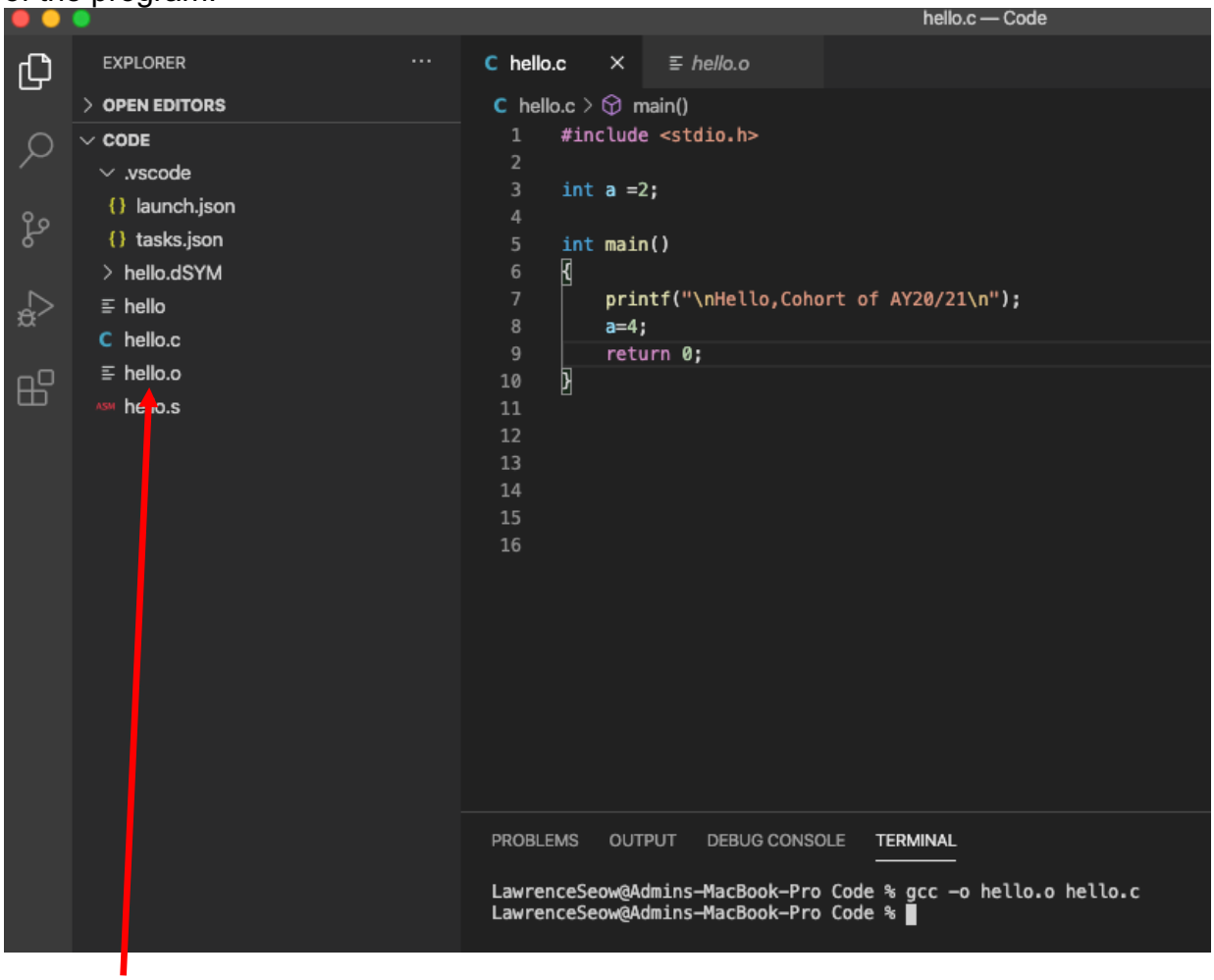
Assembly

During the assembly stage, an assembler is used to translate the assembly instructions to machine code, or *object code*. The output consists of actual instructions to be run by the target processor.

To generate machine/object code at the terminal prompt

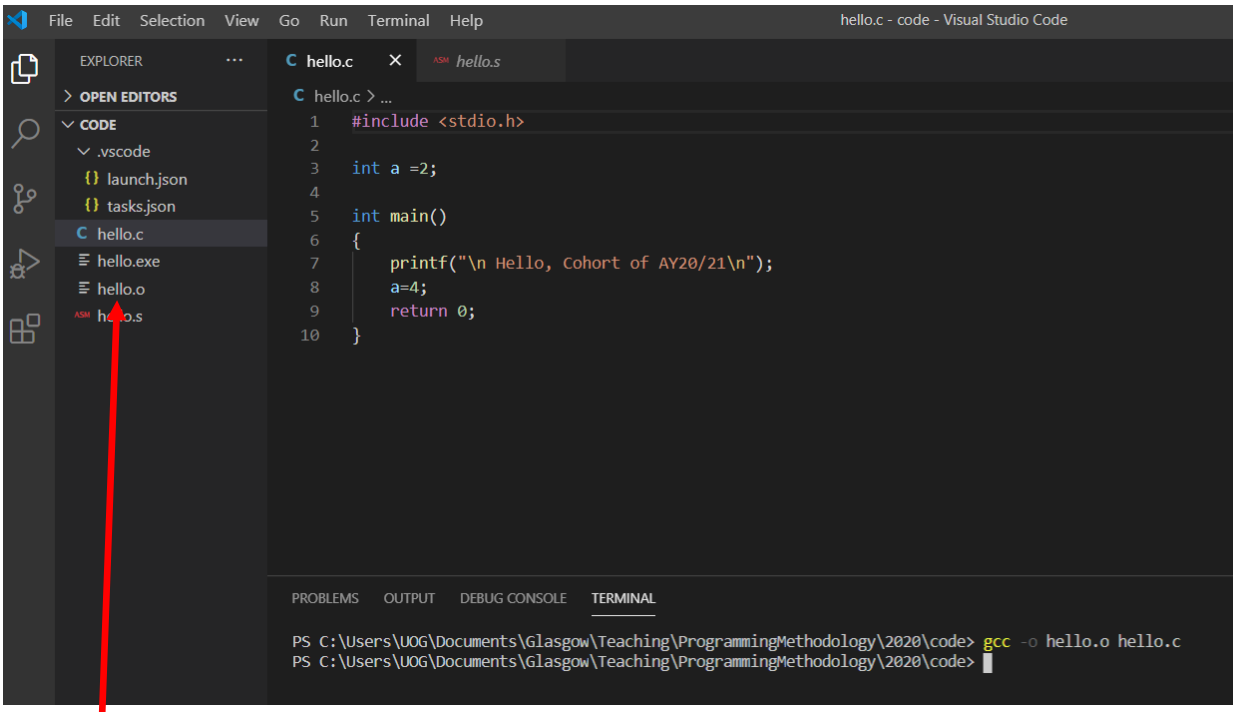
```
gcc -o hello.o hello.c
```

Running the above command will create a file named hello.o, containing the object code of the program.



hello.o file created

MAC OS



hello.o file created

Window OS

The contents of this file is in a binary format and can be viewed for MAC OS at terminal prompt with

```
hexdump hello.o
```

If the computer don't have hexdump.exe especially Windows computer, downloaded hexdump software at https://www.di-mgt.com.au/hexdump-for-windows.html
or from marketplace in VS code, search for hexdump for VSCode if any
and the machine code is

```
0000000 cf fa ed fe 07 00 00 01 03 00 00 00 02 00 00 00
0000010 10 00 00 00 58 05 00 00 85 00 20 00 00 00 00 00
0000020 19 00 00 00 48 00 00 00 5f 5f 50 41 47 45 5a 45
0000030 52 4f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000040 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
0000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000060 00 00 00 00 00 00 00 00 19 00 00 00 d8 01 00 00
0000070 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000080 00 00 00 00 01 00 00 00 00 10 00 00 00 00 00 00
0000090 00 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00
00000a0 05 00 00 00 05 00 00 00 05 00 00 00 00 00 00 00
00000b0 5f 5f 74 65 78 74 00 00 00 00 00 00 00 00 00 00
00000c0 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
00000d0 40 0f 00 00 01 00 00 00 34 00 00 00 00 00 00 00
00000e0 40 0f 00 00 04 00 00 00 00 00 00 00 00 00 00 00
00000f0 00 04 00 80 00 00 00 00 00 00 00 00 00 00 00 00
0000100 5f 5f 73 74 75 62 73 00 00 00 00 00 00 00 00 00
0000110 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000120 74 0f 00 00 01 00 00 00 06 00 00 00 00 00 00 00
0000130 74 0f 00 00 01 00 00 00 00 00 00 00 00 00 00 00
0000140 08 04 00 80 00 00 00 00 06 00 00 00 00 00 00 00
0000150 5f 5f 73 74 75 62 5f 68 65 6c 70 65 72 00 00 00
0000160 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000170 7c 0f 00 00 01 00 00 00 1a 00 00 00 00 00 00 00
0000180 7c 0f 00 00 02 00 00 00 00 00 00 00 00 00 00 00
0000190 00 04 00 80 00 00 00 00 00 00 00 00 00 00 00 00
00001a0 5f 5f 63 73 74 72 69 6e 67 00 00 00 00 00 00 00
00001b0 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
00001c0 96 0f 00 00 01 00 00 00 1a 00 00 00 00 00 00 00
00001d0 96 0f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001e0 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001f0 5f 5f 75 6e 77 69 6e 64 5f 69 6e 66 6f 00 00 00
0000200 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000210 b0 0f 00 00 01 00 00 00 48 00 00 00 00 00 00 00
0000220 b0 0f 00 00 02 00 00 00 00 00 00 00 00 00 00 00
0000230 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000240 19 00 00 00 98 00 00 00 5f 5f 44 41 54 41 5f 43
0000250 4f 4e 53 54 00 00 00 00 00 10 00 00 01 00 00 00
0000260 00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00
0000270 00 10 00 00 00 00 00 00 03 00 00 00 03 00 00 00
0000280 01 00 00 00 10 00 00 00 5f 5f 67 6f 74 00 00 00
```

```
0000290 00 00 00 00 00 00 00 00 5f 5f 44 41 54 41 5f 43
00002a0 4f 4e 53 54 00 00 00 00 00 10 00 00 01 00 00 00
00002b0 08 00 00 00 00 00 00 00 00 10 00 00 03 00 00 00
00002c0 00 00 00 00 00 00 00 00 06 00 00 00 01 00 00 00
00002d0 00 00 00 00 00 00 00 00 19 00 00 00 e8 00 00 00
00002e0 5f 5f 44 41 54 41 00 00 00 00 00 00 00 00 00 00
00002f0 00 20 00 00 01 00 00 00 00 10 00 00 00 00 00 00
0000300 00 20 00 00 00 00 00 00 00 10 00 00 00 00 00 00
0000310 03 00 00 00 03 00 00 00 02 00 00 00 00 00 00 00
0000320 5f 5f 6c 61 5f 73 79 6d 62 6f 6c 5f 70 74 72 00
0000330 5f 5f 44 41 54 41 00 00 00 00 00 00 00 00 00 00
0000340 00 20 00 00 01 00 00 00 08 00 00 00 00 00 00 00
0000350 00 20 00 00 03 00 00 00 00 00 00 00 00 00 00 00
0000360 07 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00
0000370 5f 5f 64 61 74 61 00 00 00 00 00 00 00 00 00 00
0000380 5f 5f 44 41 54 41 00 00 00 00 00 00 00 00 00 00
0000390 08 20 00 00 01 00 00 00 0c 00 00 00 00 00 00 00
00003a0 08 20 00 00 03 00 00 00 00 00 00 00 00 00 00 00
00003b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00003c0 19 00 00 00 48 00 00 00 5f 5f 4c 49 4e 4b 45 44
00003d0 49 54 00 00 00 00 00 00 00 30 00 00 01 00 00 00
00003e0 00 10 00 00 00 00 00 00 00 30 00 00 00 00 00 00
00003f0 24 01 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0000400 00 00 00 00 00 00 00 00 22 00 00 80 30 00 00 00
0000410 00 30 00 00 08 00 00 00 08 30 00 00 18 00 00 00
0000420 00 00 00 00 00 00 00 00 20 30 00 00 10 00 00 00
0000430 30 30 00 00 38 00 00 00 02 00 00 00 18 00 00 00
0000440 70 30 00 00 06 00 00 00 dc 30 00 00 48 00 00 00
0000450 0b 00 00 00 50 00 00 00 00 00 00 00 01 00 00 00
0000460 01 00 00 00 03 00 00 00 04 00 00 00 02 00 00 00
0000470 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000480 00 00 00 00 00 00 00 00 d0 30 00 00 03 00 00 00
0000490 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00004a0 0e 00 00 00 20 00 00 00 0c 00 00 00 2f 75 73 72
00004b0 2f 6c 69 62 2f 64 79 6c 64 00 00 00 00 00 00 00
00004c0 1b 00 00 00 18 00 00 00 4c c9 e4 37 6b ee 36 99
00004d0 bf 59 06 53 8d 00 28 72 32 00 00 00 20 00 00 00
00004e0 01 00 00 00 00 0f 0a 00 06 0f 0a 00 01 00 00 00
00004f0 03 00 00 00 00 06 2c 02 2a 00 00 00 10 00 00 00
0000500 00 00 00 00 00 00 00 00 28 00 00 80 18 00 00 00
0000510 40 0f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
0000520 0c 00 00 00 38 00 00 00 18 00 00 00 02 00 00 00
0000530 01 64 01 05 00 00 01 00 2f 75 73 72 2f 6c 69 62
0000540 2f 6c 69 62 53 79 73 74 65 6d 2e 42 2e 64 79 6c
0000550 69 62 00 00 00 00 00 00 26 00 00 00 10 00 00 00
0000560 68 30 00 00 08 00 00 00 29 00 00 00 10 00 00 00
0000570 70 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000580 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
0000f40 55 48 89 e5 48 83 ec 10 c7 45 fc 00 00 00 00 48
0000f50 8d 3d 40 00 00 00 b0 00 e8 17 00 00 00 31 c9 c7
0000f60 05 a7 10 00 00 04 00 00 00 89 45 f8 89 c8 48 83
0000f70 c4 10 5d c3 ff 25 86 10 00 00 00 00 4c 8d 1d 85
0000f80 10 00 00 41 53 ff 25 75 00 00 00 90 68 00 00 00
0000f90 00 e9 e6 ff ff ff 0a 48 65 6c 6c 6f 2c 43 6f 68
0000fa0 6f 72 74 20 6f 66 20 41 59 32 30 2f 32 31 0a 00
0000fb0 01 00 00 00 1c 00 00 00 00 00 00 00 1c 00 00 00
0000fc0 00 00 00 00 1c 00 00 00 02 00 00 00 40 0f 00 00
0000fd0 34 00 00 00 34 00 00 00 75 0f 00 00 00 00 00 00
0000fe0 34 00 00 00 03 00 00 00 0c 00 01 00 10 00 01 00
0000ff0 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00
0001000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
0002000 8c 0f 00 00 01 00 00 00 00 00 00 00 00 00 00 00
0002010 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0002020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
0003000 11 23 00 51 00 00 00 00 11 40 64 79 6c 64 5f 73
0003010 74 75 62 5f 62 69 6e 64 65 72 00 51 72 00 90 00
0003020 73 00 11 40 5f 70 72 69 6e 74 66 00 90 00 00 00
0003030 00 01 5f 00 05 00 03 5f 6d 68 5f 65 78 65 63 75
0003040 74 65 5f 68 65 61 64 65 72 00 24 6d 61 69 6e 00
0003050 28 61 00 2d 02 00 00 00 03 00 c0 1e 00 03 00 90
0003060 40 00 00 00 00 00 00 00 c0 1e 00 00 00 00 00 00
0003070 38 00 00 00 0e 08 00 00 08 20 00 00 01 00 00 00
0003080 02 00 00 00 0f 01 10 00 00 00 00 00 01 00 00 00
0003090 16 00 00 00 0f 08 00 00 10 20 00 00 01 00 00 00
00030a0 19 00 00 00 0f 01 00 00 40 0f 00 00 01 00 00 00
00030b0 1f 00 00 00 01 00 00 01 00 00 00 00 00 00 00 00
00030c0 27 00 00 00 01 00 00 01 00 00 00 00 00 00 00 00
00030d0 04 00 00 00 05 00 00 00 04 00 00 00 20 00 5f 5f
00030e0 6d 68 5f 65 78 65 63 75 74 65 5f 68 65 61 64 65
```

```
00030f0 72 00 5f 61 00 5f 6d 61 69 6e 00 5f 70 72 69 6e
0003100 74 66 00 64 79 6c 64 5f 73 74 75 62 5f 62 69 6e
0003110 64 65 72 00 5f 5f 64 79 6c 64 5f 70 72 69 76 61
0003120 74 65 00 00
0003124
```

Linking

The object code generated in the assembly stage is composed of machine instructions that the processor understands but some pieces of the program are out of order or missing. To produce an executable program, the existing pieces have to be rearranged and the missing ones filled in. This process is called linking.

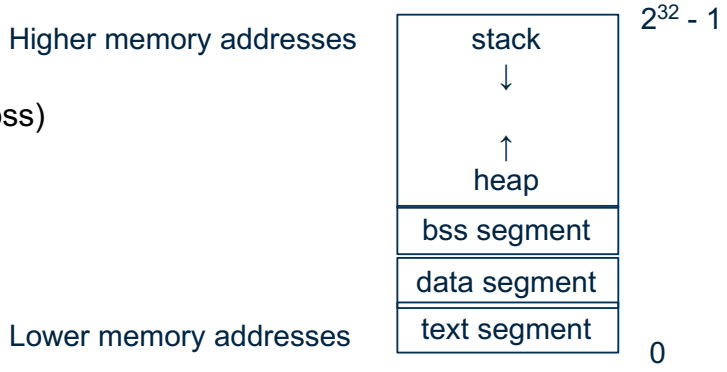
The linker will arrange the pieces of object code so that functions in some pieces can successfully call functions in other pieces. It will also add pieces containing the instructions for library functions used by the program. The result of this stage is the final executable program. Type at the terminal command prompt

```
gcc -o hello hello.c
```

4. *Understanding the memory structure of C program*

When you run any C-program, its executable image is loaded into RAM of computer in an organized manner which is called process address space or Memory layout of C program. In general, the memory layout is organized in following fashion:

1. Text or Code Segment
2. Initialized Data Segments
3. Uninitialized Data Segments (bss)
4. Stack Segment
5. Heap Segment



To have a brief understanding of such structure, type the following code and save it as memory-layout.c

a) Only code segment with no data

Create the following code and saved as memory-layout.c

```
/* memory-layout.c */

#include<stdio.h>

int main()

{

 return 0;

}
```

Compile and the check the memory as follows at the terminal prompt.

```
gcc -o memory-layout memory-layout.c
```

Window OS → `size .\memory-layout.exe`

MAC OS → `size ./memory-layout`

Observe the memory structure

```
LawrenceSeow@Admins-MacBook-Pro Lab_1 % gcc -o memory-layout memory-layout.c
LawrenceSeow@Admins-MacBook-Pro Lab_1 % size ./memory-layout
__TEXT   __DATA   __OBJC   others   dec      hex
4096     0        0        4294971392       4294975488       100002000
```

MAC OS

```
PS C:\Users\UOG\Documents\Glasgow\Teaching\ProgrammingMethodology\2020\code> gcc -o memory-layout memory-layout.c
PS C:\Users\UOG\Documents\Glasgow\Teaching\ProgrammingMethodology\2020\code> size .\memory-layout.exe
   text    data     bss     dec     hex filename
   9708    2200    2432   14340    3804 .\memory-layout.exe
```

Window OS

b) Code segment and uninitialized data

Create the following code and saved as memory2-layout.c

```
/* memory2-layout.c */

#include<stdio.h>

int x;

int main()

{

 return 0;

}
```

Compile and the check the memory as follows at the terminal prompt.

```
gcc -o memory2-layout memory2-layout.c
```

Window OS → `size .\memory2-layout.exe`

MAC OS → `size ./memory2-layout`

Observe the memory structure

```
LawrenceSeow@Admins-MacBook-Pro Lab_1 % gcc -o memory2-layout memory2-layout.c
LawrenceSeow@Admins-MacBook-Pro Lab_1 % size ./memory2-layout
__TEXT   __DATA   __OBJC  others  dec     hex
4096     4096     0       4294971392      4294979584      100003000
```

MAC OS

```
PS C:\Users\UOG\Documents\Glasgow\Teaching\ProgrammingMethodology\2020\code> gcc -o memory2-layout memory2-layout.c
PS C:\Users\UOG\Documents\Glasgow\Teaching\ProgrammingMethodology\2020\code> size .\memory2-layout.exe
   text    data     bss     dec     hex filename
   9708    2200    2440   14348    380c .\memory2-layout.exe
```

Window OS

c) Code segment and uninitialized data

Create the following code and saved as memory3-layout.c

```
/* memory3-layout.c */
#include<stdio.h>
int x;
int y[100000];
int main()
{
        return 0;
}
```

Compile and the check the memory as follows at the terminal prompt.

```
gcc -o memory3-layout memory3-layout.c
```

Window OS → `size .\memory3-layout.exe`

MAC OS → `size ./memory3-layout`

Observe the memory structure

Find the conclusion on the memory structure of C program for the 3 cases

5. *Write a simple computational C program*

a) Open the VS code editor and type the following code and save as cylinder.c

```
/*************************************************************/
/*        cylinder.c                                         */
/*        compute the volume and surface area of a cylinder */
/************************************************************ */
#include <stdio.h>
#define PI 3.141592654

void main (void);
void main (void)
{
   float radius, height, volume, surface_area;
   /* print heading */
   printf("\n Cylinder.c");
   printf("\n computes volume and surface area of a cylinder.");

   /* read in radius and height */
   printf("\n\n Enter raduius of cylinder: ");
   scanf("%f",&radius);
   printf(" Enter height of cylinder: ");
   scanf("%f", &height);

   /* compute volume and surface area */
   volume=PI*radius*radius*height;
   surface_area=2*PI*radius*(radius+height);

   /*print results */
   printf("\n volume of cylinder is %10.4f", volume);
   printf("\n surface area of cylinder is %10.4f \n\n", surface_area);
}
```

b) Compile, run and test the cylinder.c program

6. *Construct linear interpolation program, the basic of machine learning algorithm*

Linear interpolation is one of fundamental techniques of machine learning algorithm in classification. As shown in the diagram, the equation of a straight line is given as

$$y = mx + c$$

where m is the slope and c is the y intercept. Given that the coordinates of the two points P_1 and P_2 are (x_1, y_1) and (x_2, y_2) respectively, write a C program to compute the slope and y intercept of a straight line using the coordinates of the given two points on the straight line. The C program should also able to compute the y coordinates of a point on the straight line, given the x coordinate of the point.