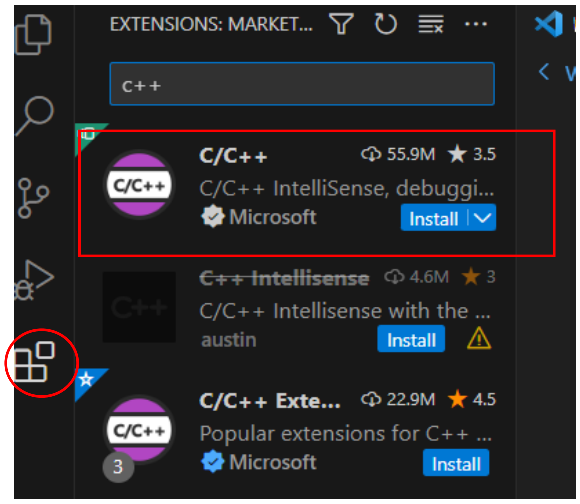
# OOP Lab 1: Set up and Run First C++ Program

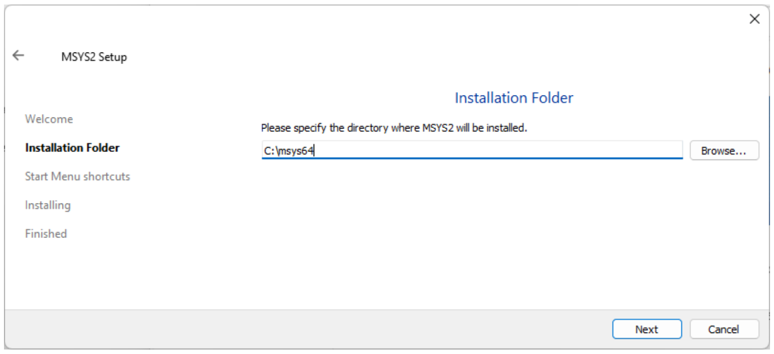
## Install and Setup for Windows

**(Check whether VS Code that already installed and set up for C++ before proceed)**

1. Install Visual Studio Code.
2. Install the C/C++ extension for VS Code. You can install the C/C++ extension by searching for 'C++' in the Extensions view (Ctrl+Shift+X).


EXTENSIONS: MARKET...
C++
C/C++ 55.9M 3.5
C/C++ IntelliSense, debuggi...
Microsoft
Install
~~C++ Intellisense~~ 4.6M 3
C/C++ Intellisense with the ...
austin
Install
C/C++ Exte... 22.9M 4.5
Popular extensions for C++ ...
Microsoft
Install

3. Installing the MinGW-w64 toolchain
   a. download the latest installer from the MSYS2 page or use this https://github.com/msys2/msys2-installer/releases/download/2024-01-13/msys2-x86_64-20240113.exe
   b. Run the installer and follow the steps of the installation wizard.
   c. In the wizard, choose your desired Installation Folder. Record this directory for later. In most cases, the recommended directory is acceptable. The same applies when you get to setting the start menu shortcuts step. When complete, ensure the Run MSYS2 now box is checked and select Finish. This will open a MSYS2 terminal window for you.

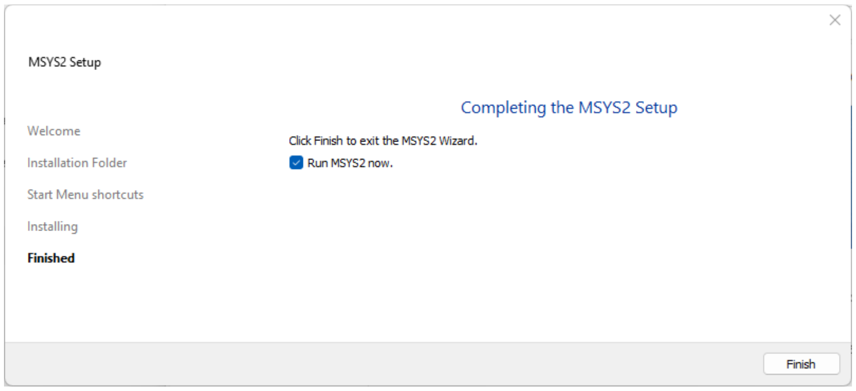


Installation Folder

Please specify the directory where MSYS2 will be installed.

C:\msys64

   d. In the wizard, choose your desired Installation Folder. Record this directory for later. In most cases, the recommended directory is acceptable. The same applies when you get to setting the start menu shortcuts step. When complete, ensure the **Run MSYS2 now** box is checked and select **Finish**. This will open a MSYS2 terminal window for you.

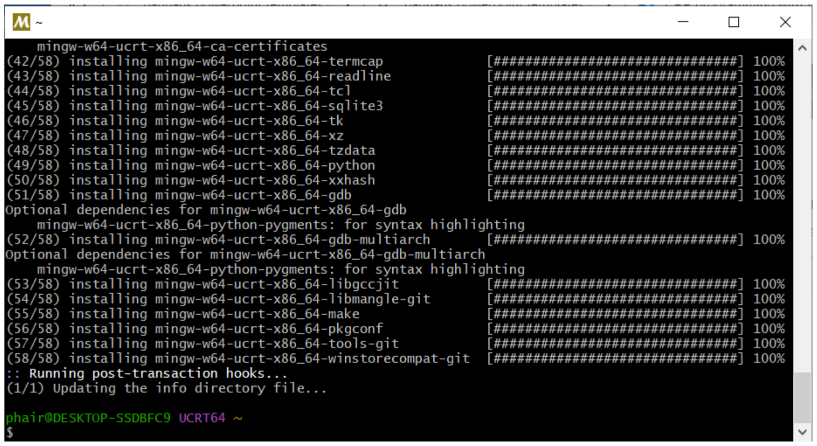


Click Finish to exit the MSYS2 Wizard.

✓ Run MSYS2 now.

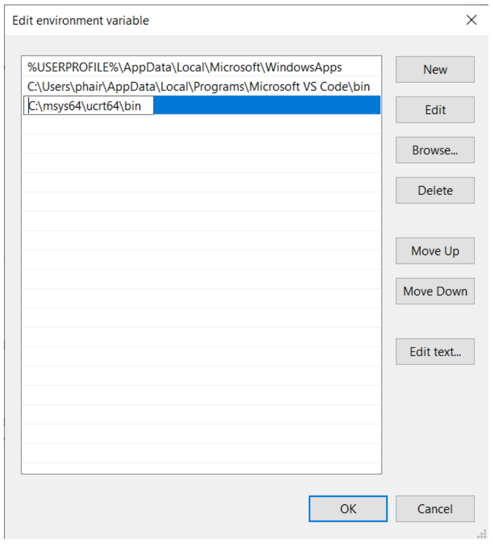
e. In this terminal, install the MinGW-w64 toolchain by running the following command:
pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain

f. Accept the default number of packages in the `toolchain` group by pressing `Enter`.



```
    mingw-w64-ucrt-x86_64-ca-certificates
(42/58) installing mingw-w64-ucrt-x86_64-termcap          [###############################] 100%
(43/58) installing mingw-w64-ucrt-x86_64-readline         [###############################] 100%
(44/58) installing mingw-w64-ucrt-x86_64-tcl              [###############################] 100%
(45/58) installing mingw-w64-ucrt-x86_64-sqlite3          [###############################] 100%
(46/58) installing mingw-w64-ucrt-x86_64-tk               [###############################] 100%
(47/58) installing mingw-w64-ucrt-x86_64-xz               [###############################] 100%
(48/58) installing mingw-w64-ucrt-x86_64-tzdata           [###############################] 100%
(49/58) installing mingw-w64-ucrt-x86_64-python           [###############################] 100%
(50/58) installing mingw-w64-ucrt-x86_64-xxhash           [###############################] 100%
(51/58) installing mingw-w64-ucrt-x86_64-gdb              [###############################] 100%
Optional dependencies for mingw-w64-ucrt-x86_64-gdb
    mingw-w64-ucrt-x86_64-python-pygments: for syntax highlighting
(52/58) installing mingw-w64-ucrt-x86_64-gdb-multiarch    [###############################] 100%
Optional dependencies for mingw-w64-ucrt-x86_64-gdb-multiarch
    mingw-w64-ucrt-x86_64-python-pygments: for syntax highlighting
(53/58) installing mingw-w64-ucrt-x86_64-libgccjit        [###############################] 100%
(54/58) installing mingw-w64-ucrt-x86_64-libmangle-git    [###############################] 100%
(55/58) installing mingw-w64-ucrt-x86_64-make             [###############################] 100%
(56/58) installing mingw-w64-ucrt-x86_64-pkgconf          [###############################] 100%
(57/58) installing mingw-w64-ucrt-x86_64-tools-git        [###############################] 100%
(58/58) installing mingw-w64-ucrt-x86_64-winstorecompat-git [###############################] 100%
:: Running post-transaction hooks...
(1/1) Updating the info directory file...

phair@DESKTOP-SSDBFC9 UCRT64 ~
$
```

g. Add the path to your MinGW-w64 `bin` folder to the Windows `PATH` environment variable by using the following steps:
1) In the Windows search bar, type **Settings** to open your Windows Settings.
2) Search for **Edit environment variables for your account**.
3) In your **User variables**, select the `Path` variable and then select **Edit**.
4) Select **New** and add the MinGW-w64 destination folder you recorded during the installation process to the list. If you used the default settings above, then this will be the path: `C:\ msys64\ucrt64\bin`.

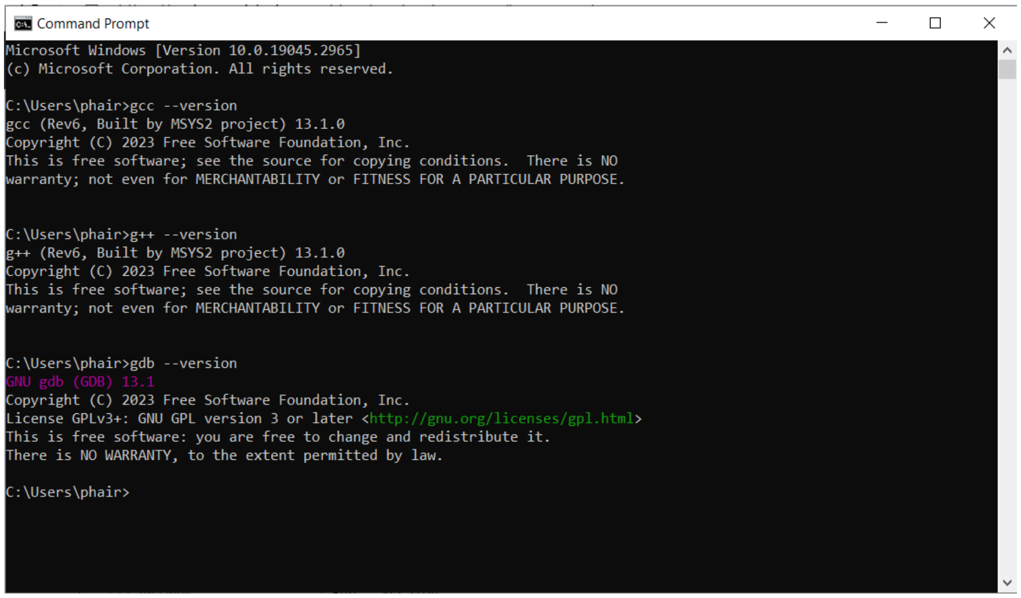


%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\phair\AppData\Local\Programs\Microsoft VS Code\bin
C:\msys64\ucrt64\bin

5) Select **OK** to save the updated PATH. You will need to reopen any console windows for the new PATH location to be available.

4. Check your MinGW installation. To check that your MinGW-w64 tools are correctly installed and available, open a **new** Command Prompt and type:

```
gcc –version
g++ --version
gdb –version
```



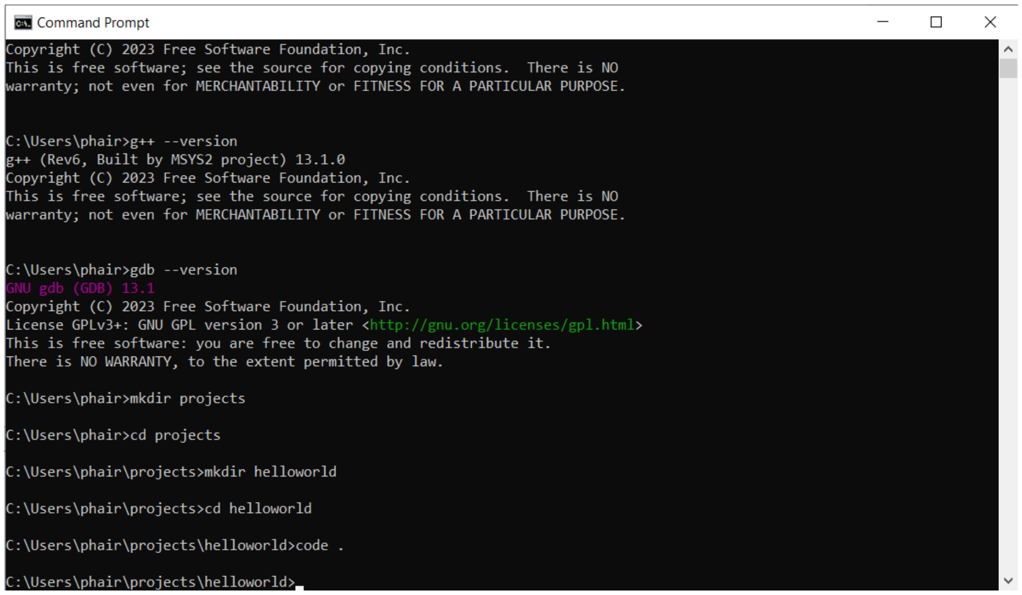
```
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\phair>gcc --version
gcc (Rev6, Built by MSYS2 project) 13.1.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


C:\Users\phair>g++ --version
g++ (Rev6, Built by MSYS2 project) 13.1.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


C:\Users\phair>gdb --version
GNU gdb (GDB) 13.1
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

C:\Users\phair>
```

******* wait for TA ***** - Check point 1**

5. Create a C++ file.
   a. On Windows, launch a Windows command prompt (Enter **Windows command prompt** in the Windows search bar).
   b. Run the following commands. They are creating an empty folder called `projects` where you can place all your VS Code projects. The next commands create and navigate you to a subfolder called `helloworld`. From there, you are opening `helloworld` directly in VS Code using the `code` command.

```
mkdir projects

cd projects

mkdir helloworld

cd helloworld
```

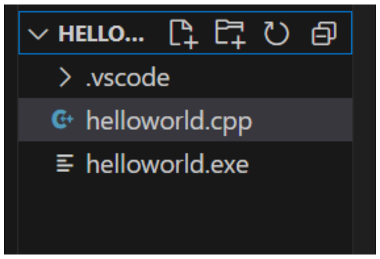
```
code .
```



```
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


C:\Users\phair>g++ --version
g++ (Rev6, Built by MSYS2 project) 13.1.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


C:\Users\phair>gdb --version
GNU gdb (GDB) 13.1
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

C:\Users\phair>mkdir projects

C:\Users\phair>cd projects

C:\Users\phair\projects>mkdir helloworld

C:\Users\phair\projects>cd helloworld

C:\Users\phair\projects\helloworld>code .

C:\Users\phair\projects\helloworld>
```

c. The "code ." command opens VS Code in the current working folder, which becomes your "workspace". Accept the Workspace Trust dialog by selecting **Yes, I trust the authors** since this is a folder you created.

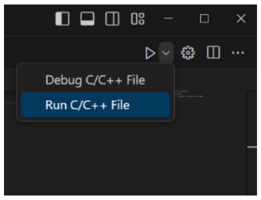
d. Now create a new file called `helloworld.cpp` with the **New File** button in the File Explorer or **File** > **New File** command.



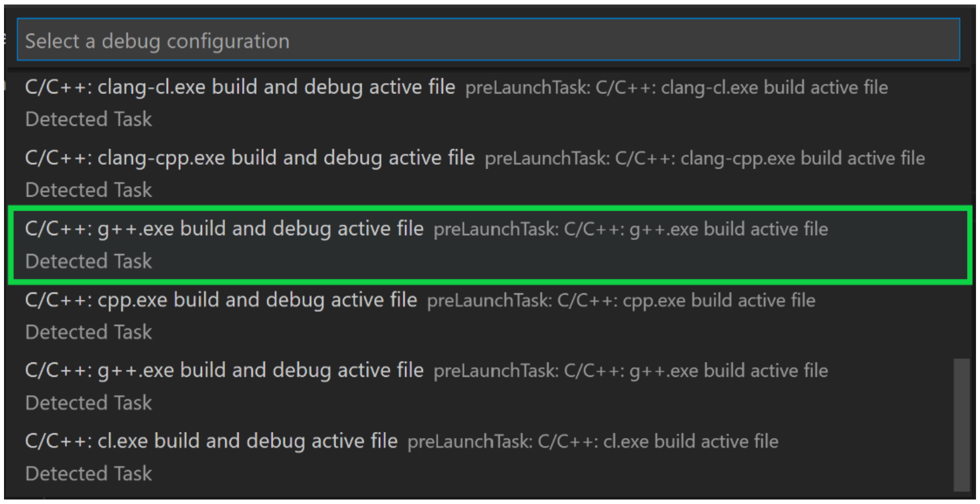
e. Paste in the following code:

```
#include <iostream>
int main()
{
   std::cout << "Hello World" << std::endl;
   std::cout << "This is my first program in C++" << std::endl;
   std::cout << "Good bye!" << std::endl;
}
```

6. Run helloworld.cpp.
   a. Make sure you have `helloworld.cpp` open so it is the active file in your editor.
   b. Press the play button in the top right corner of the editor.
   c. Choose **C/C++: g++.exe build and debug active file** from the list of detected compilers on your system.

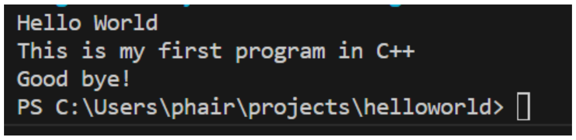


d. You are only prompted to choose a compiler the first time you run `helloworld.cpp`. This compiler becomes "default" compiler set in your `tasks.json` file.



```
Select a debug configuration
C/C++: clang-cl.exe build and debug active file   preLaunchTask: C/C++: clang-cl.exe build active file
Detected Task
C/C++: clang-cpp.exe build and debug active file   preLaunchTask: C/C++: clang-cpp.exe build active file
Detected Task
C/C++: g++.exe build and debug active file   preLaunchTask: C/C++: g++.exe build active file
Detected Task
C/C++: cpp.exe build and debug active file   preLaunchTask: C/C++: cpp.exe build active file
Detected Task
C/C++: g++.exe build and debug active file   preLaunchTask: C/C++: g++.exe build active file
Detected Task
C/C++: cl.exe build and debug active file   preLaunchTask: C/C++: cl.exe build active file
Detected Task
```

e. After the build succeeds, you should see "Hello World" appear in the integrated **Terminal**.

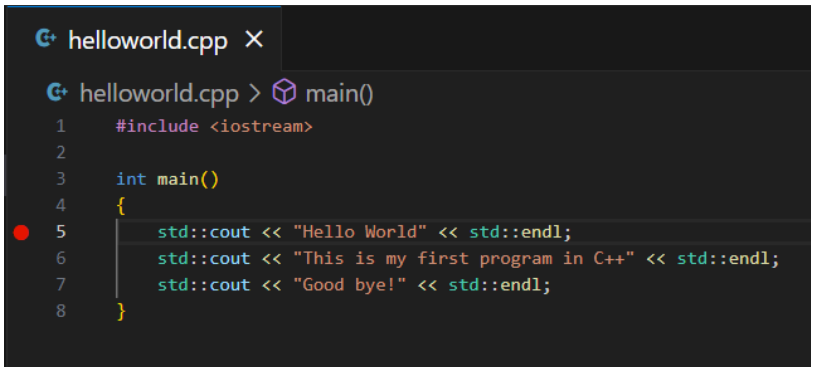


```
Hello World
This is my first program in C++
Good bye!
PS C:\Users\phair\projects\helloworld>
```

******* wait for TA ***** - Check point 2**

## Debug helloworld.cpp

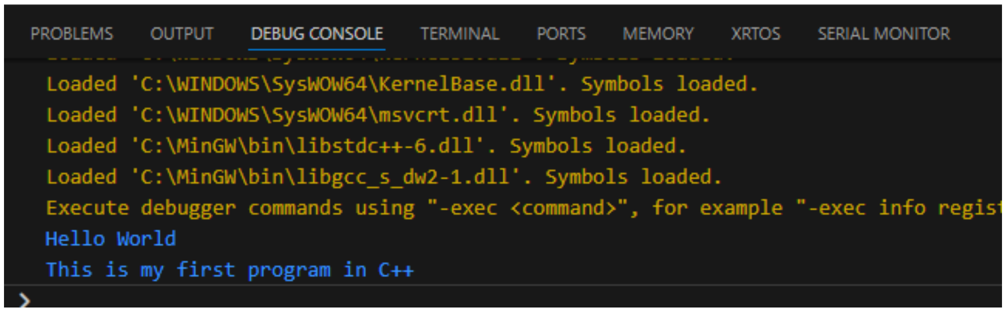
1. Go back to `helloworld.cpp` so that it is the active file.
2. Set a breakpoint by clicking on the editor margin or using F9 on the current line.



```
#include <iostream>

int main()
{
    std::cout << "Hello World" << std::endl;
    std::cout << "This is my first program in C++" << std::endl;
    std::cout << "Good bye!" << std::endl;
}
```

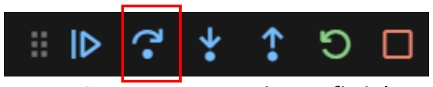
3. From the drop-down next to the play button, select **Debug C/C++ File**.
4. Choose **C/C++: g++ build and debug active file** from the list of detected compilers on your system (you'll only be asked to choose a compiler the first time you run or debug `helloworld.cpp`).
5. The play button has two modes: **Run C/C++ File** and **Debug C/C++ File**. It will default to the last-used mode. If you see the debug icon in the play button, you can just select the play button to debug, instead of using the drop-down.
6. The Integrated Terminal appears at the bottom of the source code editor. In the **Debug Output** tab, you see output that indicates the debugger is up and running.
7. The editor highlights the line where you set a breakpoint before starting the debugger:
8. The **Run and Debug** view on the left shows debugging information. You'll see an example later in the tutorial.
9. At the top of the code editor, a debugging control panel appears. You can move this around the screen by grabbing the dots on the left side.

10. Click or press the **Step over** icon in the debugging control panel. This will advance program execution.



```
Loaded 'C:\WINDOWS\SysWOW64\KernelBase.dll'. Symbols loaded.
Loaded 'C:\WINDOWS\SysWOW64\msvcrt.dll'. Symbols loaded.
Loaded 'C:\MinGW\bin\libstdc++-6.dll'. Symbols loaded.
Loaded 'C:\MinGW\bin\libgcc_s_dw2-1.dll'. Symbols loaded.
Execute debugger commands using "-exec <command>", for example "-exec info regist
Hello World
This is my first program in C++
```

11. Press **Step over** again to advance to the next statement in this program.



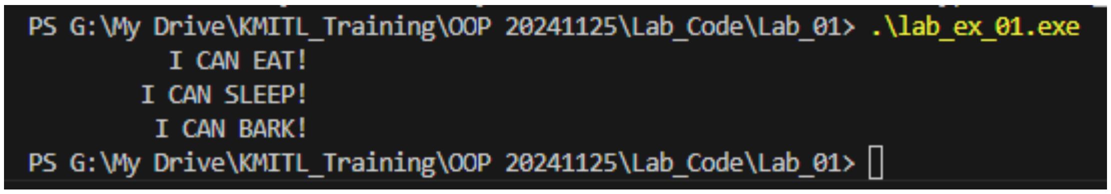
12. Press **Step over** again to finish print all 3 lines.

******* wait for TA ***** - Check point 3**

## Lab Exercise

**Inheritance and Output Formatting with Manipulators**

1. Write a C++ program that uses inheritance to create a class hierarchy for animals and demonstrates proper use of manipulators for formatted output.
2. Task Description:
    - Create a base class called Animal with the following member functions:
        - eat() that prints "I can eat!"
        - sleep() that prints "I can sleep!"
    - Create a derived class called Dog that inherits from Animal and adds an additional member function:
        - bark() that prints "I can bark!"
    - In the main function:
        - Create an instance of the Dog class and call the eat, sleep, and bark methods in that order.
3. **Output Formatting Requirements**:
    - Use the manipulators from the attached table to format your output.
        - Set a width of 20 characters for each output using setw.
        - Align the output to the right using right.
        - Use uppercase for all outputs by applying the uppercase manipulator.
    - Example Output (formatted):



```
PS G:\My Drive\KMITL_Training\OOP 20241125\Lab_Code\Lab_01> .\lab_ex_01.exe
          I CAN EAT!
        I CAN SLEEP!
         I CAN BARK!
PS G:\My Drive\KMITL_Training\OOP 20241125\Lab_Code\Lab_01> 
```

****** **TA Checking**:  ********