ATAD | Software Requirements

The installation of all the necessary software is presented, according to your operating system, in the following sections:

1. **Windows/WSL** or native **Linux**.
2. **MacOSX**.

> If you are not able to perform a manual software installation, please see Software_AlternativeMethods.pdf
>
> The *Docker Container* method is particularly advised to owners of Apple silicon machine, e.g., M1 and later chips. It is, howeer, also a good option for Apple Intel machines.

Software Listing

The complete list of software is as follows:

- Windows Subsystem for Linux (Ubuntu LTS 20.04) – **windows-only**
- GNU Compiler Collection (GCC)
- GNU Project Debugger (GDB)
- GNU Make
- Git (already installed in Windows/WSL and Linux)
- Valgrind
- Doxygen
- Visual Studio Code (IDE):
  - (Extension) C/C++
  - (Extension) Doxygen Documentation Generator

1 | Windows/WSL or Linux

Requirements

The overall system requirements are the following:

- An updated installation of Windows 10/11 or a recent Linux distribution (all mainstream Linux distributions have the necessary packages);

- 8GB of RAM;
- 5GB of disk space for Windows/WSL (you'll be installing a base Linux environment), or an additional 500MB for a Linux machine.

Installing

> **Linux machine?**
>
> **Go to step 6 if you're on a native Linux environment**. The commands provided are for *Ubuntu/Debian* systems, but you should easily find the required packages for other distributions and transpose the commands to your package manager.

Weblink: Install Linux on Windows with WSL | Microsoft Docs

Prerequisites:

- Make sure your operating system is updated;
- Make sure *Virtualization* is enabled in your computer's BIOS (google: "<your machine brand> enable virtualization bios", e.g., "thinkpad enable virtualization bios").

Quick procedure:

You should be able to install WSL+Ubuntu 20.04 in a single command, as per the *link* above. Open PowerShell or Windows Command Prompt in administrator mode by right-clicking and selecting "Run as administrator", enter the wsl –install command:

```
PS> wsl --install
```

then restart your machine and go to **step** 4.

Alternative procedure (manual):

1. Open PowerShell as Administrator, run the following command and restart.

```
PS> Enable-WindowsOptionalFeature -Online -FeatureName
 ↪  Microsoft-Windows-Subsystem-Linux
```

2. Open **Microsoft Store** and search for "Ubuntu". Install **Ubuntu LTS 20.04 LTS**.
3. Run *Ubuntu* application and wait for installation.

4. During the installation you'll be asked for a **default UNIX user account**. Enter one (no spaces allowed and I recommend all lowercase letters) and your **password** (do not forget this password!)
5. The CLI will be presented afterwards.

Updating Ubuntu image

6. Update installed packages with:

```
$> sudo apt update && sudo apt upgrade
```

… this may take a few minutes. Drink some☕.

Installing packages

7. Run the following command to install the necessary packages:

```
$> sudo apt install gcc gdb make valgrind doxygen
```

8. Make sure *git* is already installed (if it is, there is no need for reinstall):

```
$> sudo apt install git
$> git --version
```

9. Close the Ubuntu terminal.

Installing Visual Studio Code in Windows

10. Download and install **System installer 64bit** from Download Visual Studio Code - Mac, Linux, Windows.

and Install the WSL extension:

- Name: **WSL**
  ID: ms-vscode-remote.remote-wsl
  Description: Open any folder in the Windows Subsystem for Linux (WSL) and take advantage of Visual Studio Code's full feature set.
  Publisher: Microsoft

Complete and Test the development environment

11. Use Explorer to create a folder in your Windows filesystem to contain your C projects, e.g., `ATAD`.
    - In Explorer open the created folder and perform *Shift* + *Right-Click*. You should see in the context menu an option similar to **Open Linux Shell here**; choose that option and the Ubuntu terminal will open on that folder.
12. Clone the `CProgram_Template` repository.

```
$> git clone https://github.com/estsetubal-atad/CProgram_Template.git
 ↪  FirstProgram
```

13. Open the project with *VS Code*:

```
$> code FirstProgram/
```

    - **Note:** If you're **not using the *docker container*** methodology, when asked if you would like to "reopen folder to develop in a container", **choose "Don't Show Again"**. Otherwise, use that development option.

VS Code Extensions The previous step will open VS Code with a remote connection to your WSL installation.

14. Install the following extensions from VS Code Marketplace (use the extension's **ID** to search for it):
    - Name: **C/C++**
      ID: ms-vscode.cpptools
      Description: C/C++ IntelliSense, debugging, and code browsing.
      Publisher: Microsoft
    - Name: **Doxygen Documentation Generator**
      ID: cschlosser.doxdocgen
      Description: Let me generate Doxygen documentation from your source code for you.
      Publisher: Christoph Schlosser

Compiling (`make` and `gcc`)

15. Open the *integrated terminal*: Menu **Terminal > New Terminal**.

Invoke the makefile:

```
$> make
```

and run the program:

```
$> ./prog
```

Debugging

16. Edit the source file `main.c` and put the following code inside the *main* function:

```
int main() {

    char str[30] = "Debugging in VS Code";

    int i = 0; //Line 9. Place breakpoint here.
    while(str[i] != '\0') {
        printf("%c\n", str[i]);

        i++;
    }
    printf("Done!");

    return EXIT_SUCCESS;
}
```

- Place a *breakpoint* in the line that contains the `while` instruction. You should get a red dot at that position.
- In the **Run and Debug** tab (leftmost side, with a "lady beetle") you should now see a green play icon ▶ at the top beside "gdb - Debug project". Click on it and the debug will start.
  - This will call `make debug` automatically and run `gdb` over the `prog` executable.

- In the **Variables** panel you can see the current values of `str` (all the positions of the array) and `i`. The variable `i` is not yet initialized, because this instruction at line `9` hasn't been executed yet!
- Add the expression "`str[i]`" to the **Watch** list, before continuing;
- Now use the **Step Over (F10)** command to proceed line by line, watching the values change as the program executes.

---

2 | MacOSX

WARNING: Please consider using **Docker container** method described in Software_AlternativeMethods.pdf, as it will be easier and you'll have the complete toolchain, guaranteed.

If you're still aiming to manually install the software on your Mac computer, we'll be using **CLang** and **LLDB** which are themselves *forks* of GCC and GDB, but supported natively by Apple. Consequently, follow the next instructions.

Install CLang

`Clang` may already be installed on your Mac. To verify that it is, open a macOS Terminal window and enter the following command:

```
$> clang --version
```

If `Clang` isn't installed, enter the following command to install the command line developer tools:

```
$> xcode-select --install
```

> `git` is already included with XCode.

Turn off *security prompt* for debugging purposes:

```
$> sudo DevToolsSecurity --enable
```

Install Doxygen

```
$> brew install doxygen
```

This should install fine, since it is supported across all major MacOSX versions.

Install Valgrind

WARNING: *Valgrind* isn't supported for *Apple silicon*, i.e., M1 and later chips.

This is not a guaranteed installation. What has worked is the following:

```
$> brew tap LouisBrunner/valgrind
```

Followed by:

```
$> brew install --HEAD LouisBrunner/valgrind/valgrind
```

These steps are from the following *git repository*: https://github.com/LouisBrunner/valgrind-macos/

Install VS Code and extensions

Install Visual Studio Code section and C/C++ and Doxygen extensions (step 14).

Additionally, install the **CodeLLDB** extension:

https://marketplace.visualstudio.com/items?itemName=vadimcn.vscode-lldb

Test the environment in MacOSX

In your MacOS terminal, clone the following repository and perform steps 15 and 16:

```
$> git clone https://github.com/estsetubal-atad/CProgram_Template_MacOS.git
 ↪  FirstProgram
```

> This is a different repository that contains separate `makefile` and `launch.json` files for MacOS environments and the previous tools.

Also, when *debugging* you should see a green play icon ▶ at the top beside "lldb - Debug project".

Troubleshooting

WSL

Most of the times the problems arise from not enabling *Virtualization* on your BIOS settings.

References:

- https://docs.microsoft.com/en-us/windows/wsl/troubleshooting

Cannot manually install?

Check Software_AlternativeMethods.pdf for alternative methods.

Author and support

Bruno Silva (bruno.silva@estsetubal.ips.pt)

You should ask your PL teacher for any help regarding these contents and procedures.