Section 3 - Environment Setup - Windows

▼ 3.1 Windows Setup Introduction

1. **VS Code as the IDE**
2. Bunch of compilers: MinGW, MSVC, Clang, LLVM, GCC (GCC is wrapped in MinGW on Windows)

▼ 3.2 Install and Setup VS Code on Windows

- **Download VS Code**
    - **User installation**: Installs for a specific user
    - **System installation**: Installs for all users
- Install the **C/C++ extension** by Microsoft for IntelliSense, debugging, and code browsing.

▼ 3.3 Microsoft Visual Studio Install (MSVC) on Windows

- **MSVC** works on Windows.
- Download **Visual Studio Community 2022**.
- During installation, check **Desktop Development with C++**.
- We won't use Visual Studio to create projects in this course, only the **compiler** that comes with it.
- Open **Developer PowerShell for VS 2022** or **Developer Command Prompt for VS 2022**.
- Type `cl.exe` to check the compiler version, indicating a successful installation.

▼ 3.4 VS Code Setup with MSVC

- Open **Developer PowerShell for VS 2022** or **Developer Command Prompt for VS 2022**.
- Type `code .` to open VS Code.
- In VS Code, open a **new terminal** and type `cl.exe`. It should output the compiler version as before.
    - If opened directly from VS Code, running `cl.exe` will result in an error: "cl.exe is not recognized as the name of a cmdlet..."

Creating and Running a C++ Program

- Navigate to the folder where you want to create the C++ file and create `main.cpp`.
- Copy and paste the following program:

```
// main.cpp
#include <iostream>

consteval int get_value(){
    return 3;
}

int main(){
    constexpr int value = get_value();
    std::cout << "value : " << value << std::endl;
    return 0;
}
```

▼ Configure Build Task

- Go to **Terminal > Configure Tasks** and choose `cl.exe`.
- A `tasks.json` file will open under the `.vscode` folder.
- Replace the existing `args` key with the following:

```
// tasks.json
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "cppbuild",
            "label": "Build with MSVC",
            "command": "cl.exe",
            "args": [
                "/Zi",
                "/std:c++latest",
                "/EHsc",
                "/Fe:",
                "${fileDirname}\\rooster.exe",
                "${workspaceFolder}\\*.cpp"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$msCompile"
            ],
            "group": "build",
            "detail": "compiler: cl.exe"
        }
    ]
}
```

- This configures the compiler to use **C++20**.
- Change the label to "**Build with MSVC**".
- To run the file: go to **Terminal > Run Task > Build with MSVC**.
- After successful build, a `rooster.exe` file will be created.
- Open a terminal and type `rooster.exe` to run the program.

▼ IntelliSense Configuration

- Go to **View > Command Palette**, search for **C/C++ Edit Configurations (UI)**.
- Configure as follows:
  - **Compiler Path**: Choose the appropriate compiler path (2019 or 2022).
  - **C++ Standard**: Set it to **C++20**.

```
// c_cpp_properties.json
{
    "configurations": [
        {
            "name": "Win32",
            "includePath": [
                "${workspaceFolder}/**"
            ],
            "defines": [
                "_DEBUG",
                "UNICODE",
                "_UNICODE"
            ],
            "windowsSdkVersion": "10.0.19041.0",
            "compilerPath": "C:/Program Files (x86)/Microsoft Visual
            "cStandard": "c17",
            "cppStandard": "c++20",
            "intelliSenseMode": "windows-msvc-x64"
        }
    ],
    "version": 4
}
```

- After setup, return to your code; the red squiggly underlines will be gone.

▼ 3.5 Install GCC and Clang on Windows

1. **Google Winlibs**
   - Go to the first result: https://winlibs.com/
   - Go through the information provided on the site as it will be useful.
   - **GCC**: GNU Compiler Collection, a free and open-source compiler for C and C++.
   - **Mingw-w64**: A free and open-source C library targeting Windows 32-bit and 64-bit platforms.
   - Combining GCC and Mingw-w64 results in a free C/C++ compiler for Windows.
2. **GDB** (GNU Project Debugger): A useful tool for debugging programs written in C, C++, and other languages. It allows you to see what happens inside a program while it runs or what the program was doing at the moment it crashed.
3. **Download and Install GCC and Clang**:
   - Download the Win64 version from the release section of the Winlibs site.
   - Extract the downloaded archive and locate the `bin` folder with all executable binary files.
   - In this folder, you will find **Clang** and **g++** compilers.
   - Copy the extracted folder (e.g., `mingw64`) and paste it into the `C:\` drive.
4. **Set the Environment Variable**:
   - Go to Control Panel > Edit system variables > System Variables > Path > Double-click on it.
   - Add `C:\mingw64\bin` to the list and click OK.
   - To test the installation, open PowerShell and type `g++ —version`.

▼ 3.6 Configure VS Code for GCC

▼ 1. Open a Folder in VS Code

- Create a `main.cpp` file and paste the following code:

```
// main.cpp
#include <iostream>

consteval int get_value(){
    return 3;
}

int main(){
    constexpr int value = get_value();
    std::cout << "value : " << value << std::endl;
    return 0;
}
```

▼ 2. Configure Tasks

- Go to Terminal > Configure Tasks and select `g++.exe`.
- A `tasks.json` file will be created in the `.vscode` folder.
- Replace the `args` in `tasks.json` with the following:

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "cppbuild",
            "label": "Build with GCC",
            "command": "C:\\mingw64\\bin\\g++.exe",
            "args": [
                "-g",
                "-std=c++20",
                "${workspaceFolder}\\*.cpp",
                "-o",
                "${fileDirname}\\rooster.exe"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "compiler: C:\\mingw64\\bin\\g++.exe"
        }
    ]
}
```

3. Build the File:

- Go to Terminal > Run Task > Select "Build with GCC".
- This will generate a `rooster.exe` executable file.

4. Configure IntelliSense:

- Follow the IntelliSense configuration guide from the previous section to eliminate squiggly lines in the code.

3.7 Configure VS Code for Clang

1. Create a New Folder

- Inside the folder, create a `main.cpp` file and copy the code below:

```
// main.cpp
#include <iostream>

consteval int get_value(){
    return 3;
}

int main(){
    constexpr int value = get_value();
    std::cout << "value : " << value << std::endl;
    return 0;
}
```

2. Configure Tasks for Clang

- Go to Terminal > Configure Tasks and select `C/C++ Clang++.exe build active file`.
- This will create `.vscode/tasks.json` for compiler-related information and `.vscode/c_cpp_properties.json` for IntelliSense.
- Open `tasks.json` and replace the `args` with the following:

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "cppbuild",
            "label": "Build with Clang",
            "command": "C:\\mingw64\\bin\\clang++.exe",
            "args": [
                "-g",
                "-std=c++20",
                "${workspaceFolder}\\*.cpp",
                "-o",
                "${fileDirname}\\rooster.exe"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "compiler: C:\\mingw64\\bin\\clang++.exe"
        }
    ]
}
```

3. Build with Clang:

- Change the label to "Build with Clang" for readability.
- To compile, go to Terminal > Run Task > Select "Build with Clang".

4. Configure IntelliSense:

- Follow the IntelliSense configuration steps from the previous section to remove squiggly lines.

3.8 Windows template project - All compilers

1. Open a Folder in Developer PowerShell for VS 2022:

- Launch "Developer PowerShell for VS 2022".
- Navigate to your desired project folder.
- Type `code .` in the terminal to open VS Code from the Developer PowerShell, allowing access to the MSVC compiler.

2. Check Compiler Accessibility:

- Ensure you can access the **MSVC compiler** from the terminal by running the following commands:
    - `cl.exe`
    - `g++ --version`
    - `clang --version`
- Verify that all compilers (MSVC, GCC, and Clang) are properly installed and accessible.

▼ 3. Compiler Configuration

▼ **Configure Tasks for GCC**:

- Go to **Terminal > Configure Tasks** and select `g++`.
- Replace the `args` value in the `tasks.json` file using the same settings as described in the previous GCC configuration.

▼ **Reuse Previous Tasks Configuration**:

- Since we've already configured `tasks.json` for both **GCC** and **Clang**, you can copy and paste those `tasks.json` configurations directly.

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "cppbuild",
            "label": "Build GCC",
            "command": "C:\\mingw64\\bin\\g++.exe",
            "args": [
                "-g",
                "-std=c++20",
                "${workspaceFolder}\\*.cpp",
                "-o",
                "${fileDirname}\\rooster.exe"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "compiler: C:\\mingw64\\bin\\g++.exe"
        },
        {
            "type": "cppbuild",
            "label": "Build with MSVC",
            "command": "cl.exe",
            "args": [
                "/Zi",
                "/std:c++latest",
                "/EHsc",
                "/Fe:",
                "${fileDirname}\\rooster.exe",
                "${workspaceFolder}\\*.cpp"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$msCompile"
            ],
            "group": "build",
            "detail": "compiler: cl.exe"
        },
        {
            "type": "cppbuild",
            "label": "Build with Clang",
            "command": "C:\\mingw64\\bin\\clang++.exe",
            "args": [
                "-g",
                "-std=c++20",
                "${workspaceFolder}\\*.cpp",
                "-o",
                "${fileDirname}\\rooster.exe"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "compiler: C:\\mingw64\\bin\\clang++.exe"
        }

    ]
}
```

▼ **Access All Compilers**

▼ After setting this up, you will be able to access all three compilers (MSVC, GCC, and Clang) via **Terminal > Run Task** in VS Code.