Essential Extensions:

C/C++ by Microsoft: This is the foundational extension, providing crucial features like IntelliSense (code completion, parameter info), debugging, code navigation, and basic formatting for C and C++. Clangd: An alternative to the Microsoft C/C++ extension's language server, Clangd offers fast and accurate code completion, diagnostics, navigation, and refactoring capabilities, often preferred for larger projects due to its performance. CMake Tools: If using CMake for your build system, this extension provides seamless integration, allowing you to configure, build, and debug CMake-based projects directly within VS Code. Code Runner: A lightweight extension for quickly executing snippets of C++ code or entire files without needing to set up a full build system, ideal for testing algorithms or small functions. codeLLDB: For debugging, codeLLDB offers a powerful and flexible debugger experience, especially useful for projects that require advanced debugging features.

Helpful Additions:

GitLens: Integrates Git functionality deeply into VS Code, providing insights into code authorship, commit history, and more, which is invaluable for version control. GitHub Pull Requests and Issues: Streamlines the workflow for managing GitHub pull requests and issues directly within the VS Code environment. Live Share: Facilitates collaborative coding by allowing you to share your workspace with others in real-time for pair programming or remote assistance. Todo Tree: Collects and displays all "TODO" comments and similar tags in your codebase, providing a centralized overview of pending tasks.

All Extensions :-

https://terminalroot.com/top-10-vscode-extensions-for-c-cpp/

Terminal › External: Linux Exec Customizes which terminal to run on Linux.

x-terminal-emulator konsole gnome-terminal

Installing Tools

download and install VS-Code

```
sudo apt install  ./Downloads/code.........
```

Install Compiler ---> Gcc Compiler

```
sudo apt install build-essential
```

- **what it is :** build-essential is A meta-package that installs the basic tools required to compile C and C++ programs on Linux.
- **It includes:**
  - gcc — GNU Compiler Collection (for C)
  - g++ — C++ compiler
  - make — Tool to automate the build process using Makefiles.
  - Other necessary libraries and header files

Install debugger

```
sudo apt install gdb
```

- **What it is:** GDB = GNU Debugger - A powerful debugging tool for C and C++ programs.
- **What it does:**
  - Lets you pause program execution (breakpoints)
  - Step through your code line by line
  - Inspect variable values while the program is running
  - Find bugs like segmentation faults or logic errors
- **Why you need it:**
  - It’s essential for diagnosing and fixing bugs in your code. It helps you understand exactly what's happening when your program runs.

Install CMake

```
sudo apt install cmake
```

- **What it is:**
  - A family of tools designed to build, test, and package software. It is an open-source, cross-platform build system generator.
  - CMake is a cross-platform build system generator.
- **Purpose:**
  - For small "hello world" projects, you can compile with a single compiler command. For larger, real-world projects with many files, libraries, and configurations (e.g., building on Windows vs. Linux), a build system is essential. CMake generates native build files (like Makefiles or Visual Studio projects) that the compiler uses to compile the entire project correctly, making the build process consistent across different operating systems and environments.
  - Creates platform-specific build files (like Makefiles or Visual Studio projects)

    - Simplifies building large projects
    - Allows modular configuration of dependencies
- **Why you need it:**
    - It’s widely used in open-source and large-scale C/C++ projects. Instead of writing Makefiles manually, you write a CMakeLists.txt file and let CMake handle the rest.

Install git

```
sudo apt install git
```

- **What it is:** The most widely used Version Control System (VCS).
- **Purpose:** It is used to track changes in your source code over time. Key functionalities include:
    - Collaboration: Allows multiple developers to work on the same project without overriding each other's changes.
    - History: You can view a complete history of changes and revert to a previous, stable version of the code if needed.
    - Branching: Allows developers to work on new features or bug fixes in isolation before merging them into the main codebase.
- **What it does:**
    - Tracks changes in your code over time
    - Lets you go back to previous versions
    - Makes collaboration easy by syncing code between multiple developers
    - Integrates with platforms like GitHub, GitLab, Bitbucket
- **Why you need it:**
    - It’s essential for managing code in any real-world project, especially when working in teams.

Install clang-format

```
sudo apt install clang-format
```

- **What is clang-format?**
    - clang-format is a tool that automatically formats your source code according to a defined coding style.
    - It works with several programming languages, especially: C , C++ , Java , JavaScript , Objective-C
    - It’s part of the LLVM/Clang project.
- **What Does clang-format Do?**

    - It takes messy or inconsistently formatted code and reformats it to follow a consistent, readable style.

Before formatting:	After running clang-format:

int main(){std::cout<<"Hello";return 0;}	int main() {
	std::cout << "Hello";
	return 0;
	}

- **Purpose**
    - Main purpose: Automatically formats code
    - Keeps your code clean and consistent
    - Saves time by automating formatting
    - Makes it easier to work in teams, since everyone follows the same style
    - Helps with code reviews — focus on logic, not formatting

Install clang-tidy

```
sudo apt install clang-tidy
```

- **What is clang-tidy?**
    - clang-tidy is a C/C++ "linter" and static analysis tool that helps you:
        - Catch bugs and potential issues early
        - Enforce coding standards
        - Apply modern C++ best practices
        - Automatically refactor or fix parts of your code
    - It's part of the LLVM/Clang tooling suite, just like clang-format, but while clang-format focuses on formatting, clang-tidy focuses on code quality and correctness.
- **Purpose :** Linting, static analysis, bug detection, code improvements.

Feature	Example
🐛 Detect bugs	Use of uninitialized variables, memory issues
🔒 Catch undefined behavior	Dangerous pointer operations, bad casts
🎯 Enforce coding standards	Google style, LLVM style, custom rules
🚀 Suggest modern C++ replacements	Replace for loops with range-based loops, use auto smartly

Feature	Example
🔧 Auto-fix code (when safe)	Add missing includes, simplify expressions, fix naming conventions

- clang-tidy vs clang-format

Tool	Purpose	Modifies Code?
clang-format	Code style & formatting	Yes
clang-tidy	Code quality & analysis	Yes (optional auto-fix)

Install all in one Command

```
sudo apt install build-essential gdb cmake git clang-format clang-tidy
```

Tools :-

```
# Update package list
sudo apt update

# Install build-essential (includes g++, gcc, make, etc.)
sudo apt install build-essential

# Verify installation
g++ --version
gcc --version
make --version
#What's included in build-essential:
#g++ - C++ compiler
#gcc - C compiler
#make - Build automation tool
#libc-dev - C standard library
#dpkg-dev - Package building tools
#==================================================================
#Install GDB - GNU Debugger
sudo apt install gdb
# Verify installation
gdb --version
# Basic usage
gdb ./your_program #-tui
# Common commands: run, break, next, step, print, backtrace
#==================================================================
# CMake - Build System Generator
sudo apt install cmake
# Verify installation
cmake --version
```

```
# Optional: Install latest version via snap if needed
sudo snap install cmake --classic
#==================================================================
sudo apt install git
sudo apt install clang-format
sudo apt install clang-tidy


#🚀 Advanced Setup Options
# Code coverage
sudo apt install lcov gcovr

# Static analysis
sudo apt install clang-tidy cppcheck

# Profiling tools
sudo apt install valgrind

# Documentation
sudo apt install doxygen graphviz
```

📁 Sample Project Structure

```
my_project/
├── CMakeLists.txt
├── src/
│   ├── main.cpp
│   └── utils.cpp
├── include/
│   └── utils.h
└── build/
```

```
// main.cpp
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, Build Tools!" << endl;
    return 0;
}


//-------------------# Compile directly-------------------------
/*
g++ -o hello main.cpp

# Compile with debugging info
```

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

# Compile with optimizations
g++ -O2 -o hello_optimized main.cpp

# Run the program
./hello
*/

//--------------------#CMake Example------------------------------
// CMakeLists.txt
/*
cmake_minimum_required(VERSION 3.10)
project(MyProject)

set(CMAKE_CXX_STANDARD 14)

# Add executable
add_executable(hello src/main.cpp src/utils.cpp)

# Include directories
target_include_directories(hello PRIVATE include)
*/


//----------------# Compile with debug symbols--------------
/*
g++ -g -o debug_example debug_example.cpp

# Debug with GDB
gdb ./debug_example

# Inside GDB:
# (gdb) break main
# (gdb) run
# (gdb) next
# (gdb) print result
# (gdb) backtrace
*/
```

VS Code Extension

- C/C++ Microsoft : microsoft.com
- Clang-Format v1.9.0 : Xaver Hellauer
- Clang-tidy
- CMake Tools v1.21.36 Microsoft :microsoft.com

VS Setting

- clang setting
    - Check format on save

- Rulers
- Whitespace Display
    - View -> Appearance -> Render Whitespace.
    - workbench.colorCustomizations in settings.

    ```
    "workbench.colorCustomizations": {
      "editorWhitespace.foreground": "#ffa500"
    }
    ```

~~~
- makeFile
  - touch CMakeLists.txt
    ```c
      project(cpp_course)
      add_executable(ex_1 ex1.cc)
    ~~~
- Build
``` bash
g++ file.cc -o main
```

- Build with Debugging

```
g++ -g file.cc -o main

gdb -tui main #: Starts the GNU Debugger in text user interface mode for the executable main.
    - file main #: Loads the main executable file into GDB.
    - layout asm #: Switches the TUI display to show assembly code.
    - layout src #: Switches the TUI display back to show the source code.
    - b 5 #: Sets a breakpoint at line 5 of the source code.
    - r #: Runs the program under the debugger, stopping at breakpoints.
    - p varname #: Prints the value of the variable varname at the current point of execution.
```