VSCode C Debugging — Quick Reference & Checklist

Purpose: Minimal, practical guide to the files & settings you must create/edit to debug C programs in Visual Studio Code. Focus: simple projects (2–5 files), Makefiles, and projects using static libraries (.a).

---

1. Files VSCode uses for debugging

- `launch.json` — tells VSCode how to start the debugger (executable path, debugger type, args, working dir, preLaunchTask).
- `tasks.json` — builds your project (compile/link). The debugger usually calls this via `preLaunchTask`.
- `c_cpp_properties.json` — includes include paths and compiler settings used by the C/C++ extension for IntelliSense (not required for debugging but very helpful).
- `Makefile` (optional) — if you use make, tasks.json can call `make`. Keeps multi-file projects organized.
- **Static library files** (`libfoo.a`) — if you link a static library, your linker flags must include it (e.g. `-Lpath -lfoo`).

  Location: put all VSCode config under `.vscode/` (create if missing): - `.vscode/launch.json` - `.vscode/tasks.json` - `.vscode/c_cpp_properties.json`

---

2. Minimal `launch.json` (gdb) — what you will edit most

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "(gdb) Launch",
      "type": "cppdbg",
      "request": "launch",
      "program": "${workspaceFolder}/bin/myprog",    // <--- set to your built executable
      "args": [],                                    // command-line args
      "stopAtEntry": false,
      "cwd": "${workspaceFolder}",
      "environment": [],
      "externalConsole": false,
      "MIMode": "gdb",
      "miDebuggerPath": "/usr/bin/gdb",            // change if on Windows/MinGW
      "preLaunchTask": "build"
```

```
    }
  ]
}
```

Most edits per project: - `program` — path to your compiled executable - `args` — runtime arguments you want to test - `cwd` — working directory for the program - `miDebuggerPath` — path to gdb (especially on Windows) - `preLaunchTask` — must match a task in `tasks.json` that builds your program

---

3. Minimal `tasks.json` — what you will edit most

Option A: use `gcc` directly (small projects):

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "build",
      "type": "shell",
      "command": "gcc",
      "args": [
        "-g", "-O0", "-Wall", "src/*.c", "-o", "bin/myprog"
      ],
      "group": { "kind": "build", "isDefault": true }
    }
  ]
}
```

Option B: call `make` (recommended when you have a Makefile):

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "build",
      "type": "shell",
      "command": "make",
      "args": [],
      "group": { "kind": "build", "isDefault": true }
    }
  ]
}
```

Most important flags: - `-g` — include debug symbols (required for meaningful source-level debugging). - `-O0` — disable optimization for easier debugging (optimizations reorder/inline code).

---

4. `c_cpp_properties.json` (intellisense)

Example minimal:

```
{
  "configurations": [
    {
      "name": "Linux",
      "compilerPath": "/usr/bin/gcc",
      "includePath": ["${workspaceFolder}/include", "/usr/include"],
      "cStandard": "c11"
    }
  ],
  "version": 4
}
```

Edit when: - you add new headers or library include directories - you switch compilers

---

5. Makefile tips (small multi-file projects)

Basic Makefile for 2–5 C files that produce `bin/myprog` and use `libfoo.a`:

```
CC = gcc
CFLAGS = -g -O0 -Wall -Iinclude
LDFLAGS = -Llibs -lfoo
SRCS = src/main.c src/a.c src/b.c
OBJS = $(SRCS:.c=.o)
BIN = bin/myprog

all: $(BIN)

$(BIN): $(OBJS)
    $(CC) $(CFLAGS) -o $@ $(OBJS) $(LDFLAGS)

%.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@

clean:
    rm -f $(OBJS) $(BIN)
```

```
.PHONY: all clean
```

Important things to change per project: - `SRCS` (source files) - `CFLAGS` (include dirs or standards) - `LDFLAGS` (library dirs `-L` and `-l` flags) - `BIN` and output locations

---

6. Static library (`.a`) notes — what to edit

When using a static library `libfoo.a`: - Ensure the linker sees it: use `-L/path/to/libs -lfoo` in the link step. Or list the full path: `/abs/path/to/libfoo.a`. - Order matters: place library flags **after** object files when using `gcc` (e.g. `gcc -g -o myprog a.o b.o -Llibs -lfoo`). - If the static lib depends on other system libs, include them as well. - You usually do *not* need `-static` unless you want fully static executable (rare for debugging).

Makefile example lines to edit: set `LDFLAGS` to include `-Llibs -lfoo` or use `LIBS = libs/libfoo.a` and add `$(LIBS)` to link command.

---

7. Quick example: 2-file project

Files: - `src/main.c` - `src/helper.c` - headers in `include/`

tasks.json: simple `gcc -g -O0 src/*.c -Iinclude -o bin/myprog`

launch.json: `program` -> `${workspaceFolder}/bin/myprog`, `preLaunchTask` -> `build`.

Checklist (short): ensure `-g`, ensure `preLaunchTask` matches task label, ensure `program` path exists after build.

---

8. Common gotchas & troubleshooting

- **No symbols / can't set breakpoint** — you compiled without `-g` or with `-O2` optimizations; recompile with `-g -O0`.
- **Source line mismatch** — compiler optimizations or building different binary than the one launched. Confirm `program` points to the exact binary that `tasks.json` produces.
- **Missing includes in IntelliSense** — update `c_cpp_properties.json` includePath.
- **GDB not found on Windows** — set `miDebuggerPath` to your MinGW or Cygwin gdb and use correct MIMode.
- **Breakpoints ignored** — confirm binary is rebuilt and debugger reloads; try `Clean` and re-build.

---

9. One-page checklist to repeat for each new project

1. Create folder structure: `src/`, `include/`, `bin/`, `.vscode/`.
2. Write/confirm `Makefile` or simple `gcc` build command.
3. Ensure build includes: `-g -O0 -Wall`.
4. Add library flags if using static libs: set `-L` and `-l` or add `.a` full path.
5. Create `.vscode/tasks.json` with a task labeled `build` that builds the project (or calls `make`).
6. Create `.vscode/launch.json` and set `program` to `bin/<your-exe>`, set `preLaunchTask` to `build`, set `miDebuggerPath` if needed.
7. Update `.vscode/c_cpp_properties.json` includePath to include `include/` and any library headers.
8. Build once, then run the Debug -> Start Debugging (F5). Fix paths if it fails.
9. If debugging static lib code, ensure that the static lib was compiled with `-g` as well.

---

10. Minimal recommended workflow (every project)

1. `make clean` (or delete bin/)
2. Build with `build` task (Ctrl+Shift+B or let `preLaunchTask` run)
3. Start debugger (F5)
4. If breakpoint not hit, re-check `program` path and `-g` flag

---

11. Want a printable PDF?

If you want, I can produce this document as a clean, print-ready PDF (one page or two). I can export it and give you a downloadable file.

End — short summary

- You mainly edit `launch.json` (program, args, preLaunchTask), `tasks.json` (how to build), and `c_cpp_properties.json` (includes). For static libs: set linker `-L/-l` or reference `.a` path in your Makefile or task. Always compile with `-g -O0` for debugging.