
1 ATAD | Debugging in VS Code with GDB

This assumes you have all the required software packages installed. See installation notes if not sure (other file).

Important: If you have *cloned* the `CProgram_Template` project, then you can ignore the “Manual set up” section.

1.1 Example program

Consider the following program in `main.c` (the line no. 9 is identified. You’ll need it later on).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {

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

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

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

    return EXIT_SUCCESS;
}
```

1.1.1 Debugging procedure example

1. Considering the previous program, place a *breakpoint* at line 9 (click with the mouse right beside the line number). You should now see a persistent red circle. This indi-

cates that, when debugging, the program will stop here and you'll have control of the program flow afterwards.

2. In the **Run** tab (left side) 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.
3. 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!
4. Add the expression “`str[i]`” to the **Watch** list, before continuing;
5. Now use the **Step Over (F10)** command to proceed line by line, watching the values change as the program executes.

1.2 Manual set up

This is only required for an empty or existing project.

The following steps must be complete in order to debug programs interactively inside Visual Studio Code.

Your `makefile` should look like this (the important part is the usage of the `-g` flag; this will include debug symbols in the executable):

```
default:
    gcc -Wall -o prog main.c
debug:
    gcc -Wall -o prog -g main.c
clean:
    rm -f prog
```

Please note that the executable file in all cases is named `prog`.

Go ahead and compile the program with the `make` command, invoking the `debug` directive:

```
$> make debug
```

1.3 Configuring debugging in VS Code

You must produce the following steps to ensure the correct usage of *gdb* inside *vs code*, together with all the nice interactive features:

1. Create a folder named **.vscode** within your *working directory*.
2. Create and copy the contents for the following files:

tasks.json

```
{
  "tasks": [
    {
      "type": "cppbuild",
      "label": "C/C++: call make debug",
      "command": "/usr/bin/make",
      "args": [
        "debug"
      ],
      "options": {
        "cwd": "${workspaceFolder}"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "detail": "Task generated by Debugger."
    }
  ],
  "version": "2.0.0"
}
```

launch.json

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "gdb - Debug project",
      "type": "cppdbg",
      "request": "launch",
      "program": "${workspaceFolder}/prog",
      "args": [],
      "stopAtEntry": false,
      "cwd": "${workspaceFolder}",
      "environment": [],
      "MIMode": "gdb",
      "setupCommands": [
        {
          "description": "Enable pretty-printing for gdb",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        }
      ],
      /* This calls the debug task in "tasks.json" prior to debugging */
      "preLaunchTask": "C/C++: call make debug",
      "miDebuggerPath": "/usr/bin/gdb"
    }
  ]
}
```

1.4 Author and support

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

Please follow the course *guidelines* when seeking support.