ATAD | Debugging with GDB

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

Debugging can be performed in two ways:

1. Within VS Code (will attach process to `gdb` with UI interaction).
    - This method requires some manual configuration, but if you have *cloned* the `CProgram_Template` project, then you can ignore the "Appendix | VS Code project manual set up" section. It will "just work".
2. Directly in the terminal, using `gdb` command, or;

Example program

Consider the following program in `main.c`. The line no. 9 is identified; we'll set a *breakpoint* here later.

```c
#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;
}
```

Compile with debug symbols

To make debugging possible, your program must be compiled with *debug symbols*, using the `-g` flag.

> Regardless of the debugging method, we'll assume there is a *makefile* that is used to compile your program.

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. Debugging in VS Code

This method assumes you have a *makefile* in your project with a directive named `debug` and also a `.vscode` folder with the contents described in "Appendix | VS Code project manual set up". Again, if you are using the `CProgram_Template` base VS project, the folder and makefile already exist.

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

2. Open the **Run and Debug** tab (left side, see image below):

Figure 1: Icon Image

3. 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.
4. 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!
5. Add the expression "`str[i]`" to the **Watch** list, before continuing;
6. Now use the **Step Over (F10)** command to proceed line by line, watching the values change as the program executes.

2. Debugging in Terminal

`gdb` comes with a graphical mode called TUI. It can be used to see the current instruction being executed during the debugging process.

1. Compile your program with debug symbols, e.g.:

    ```
    $> make debug;
    ```

2. Run `gdb` with your program as the input:

    ```
    $> gdb ./prog
    ```

    Accept (ENTER) all questions until you reach the `(gdb)` prompt.

3. Turn on the graphical source viewer (TUI):

    ```
    (gdb) layout src
    ```

    This will show the source code. If it ever gets "scambled", use de Control+L shortcut to refresh.

4. Set your *breakpoint*:

   ```
   (gdb) break main.c:9
   ```

   The sintax is `<file>.c:<line_number>`. If your project has several source files, you can put a breakpoint in any of them. You can also set several breakpoints.

5. Start the program execution:

   ```
   (gdb) run
   ```

   The program will stop at the first breakpoint.

6. At any point you can print the current value of a variable, e.g.:

   ```
   (gdb) print str
   (gdb) print i
   ```

7. To "watch" a variable, you can use:

   ```
   (gdb) watch i
   ```

   Whenever `i` changes, you'll be notified.

8. To execute the program step-by-step you can use `step` (also, just `s`) or `next` (also, just `n`) commands, e.g.:

   ```
   (gdb) next
   ```

9. If you just press ENTER, the previous commando will be repeated (in this case, `next`).

10. To terminate the *debugging* process, press Control+C and then quit `gdb`:

    ```
    (gdb) quit
    ```

More information at:

- https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf
- https://www.youtube.com/watch?v=MTkDTjdDP3c

Appendix | VS Code project manual set up

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 you *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"
        }
    ]
}
```

Author and support

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

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