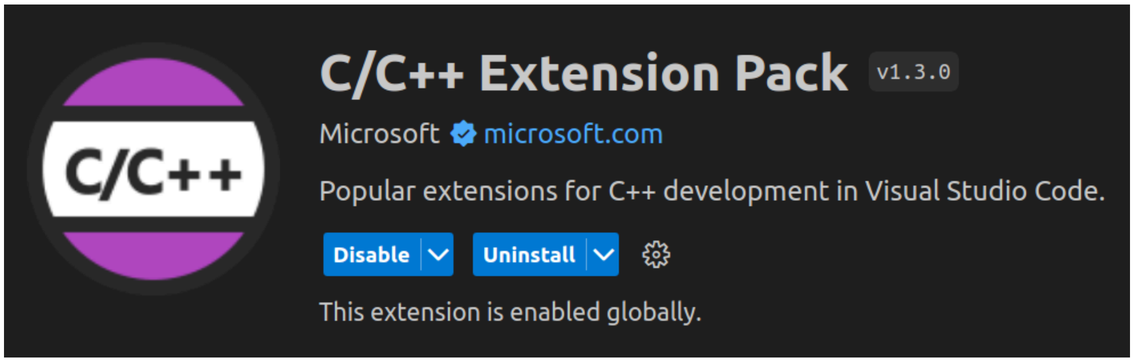
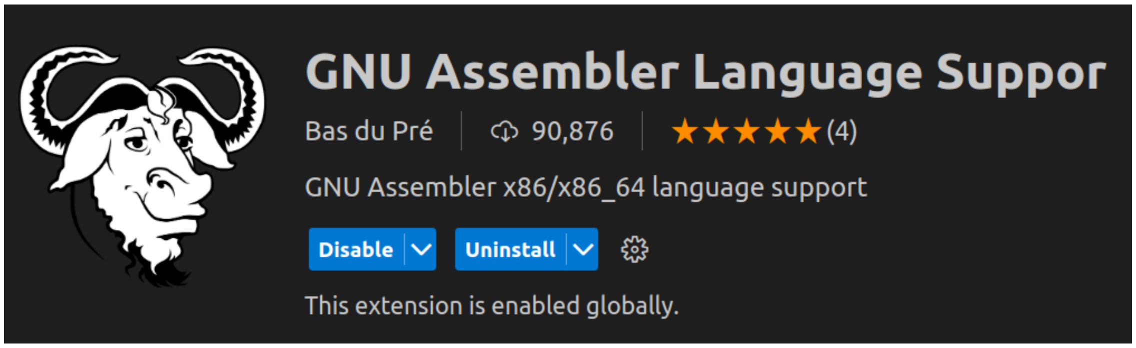
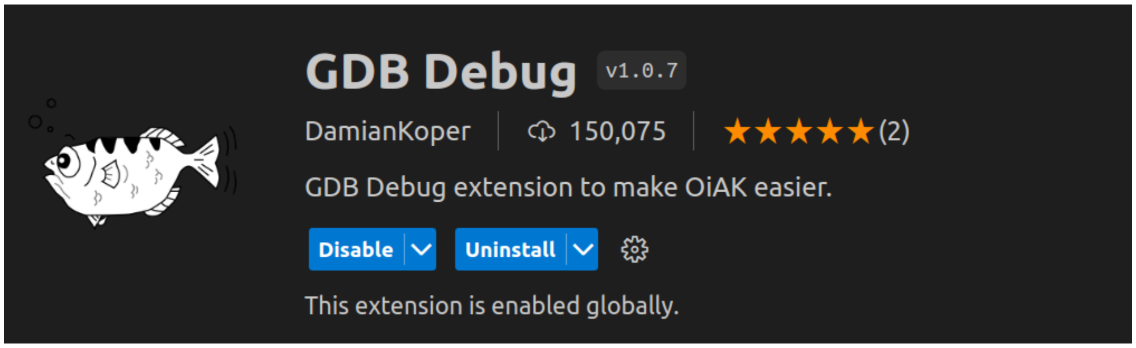
Debugging Assembly Using VSCode

Prerequisites

Make sure you have the following extensions downloaded:


C/C++ Extension Pack v1.3.0
Microsoft microsoft.com
Popular extensions for C++ development in Visual Studio Code.
This extension is enabled globally.

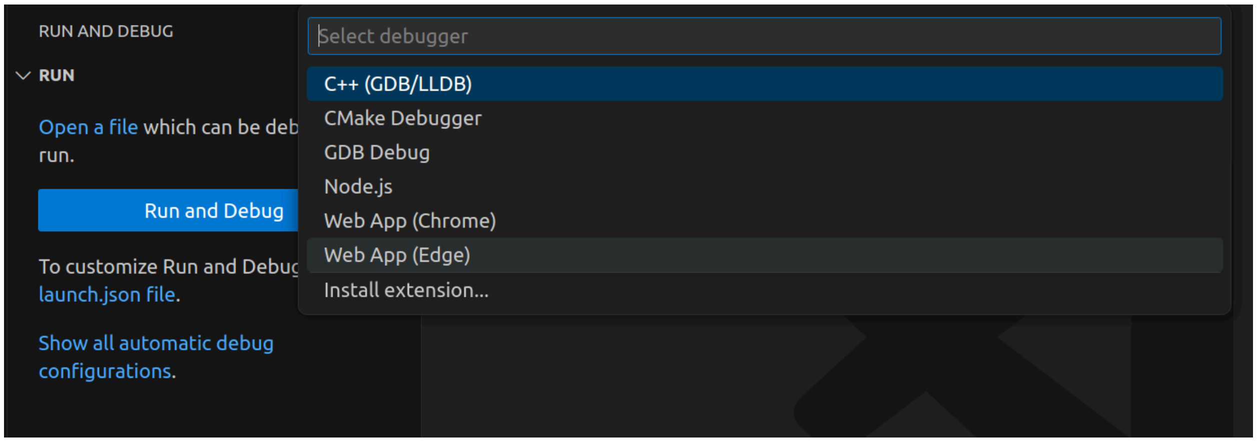

GNU Assembler Language Suppor
Bas du Pré | 90,876 | ★★★★★ (4)
GNU Assembler x86/x86_64 language support
This extension is enabled globally.


GDB Debug v1.0.7
DamianKoper | 150,075 | ★★★★★ (2)
GDB Debug extension to make OiAK easier.
This extension is enabled globally.

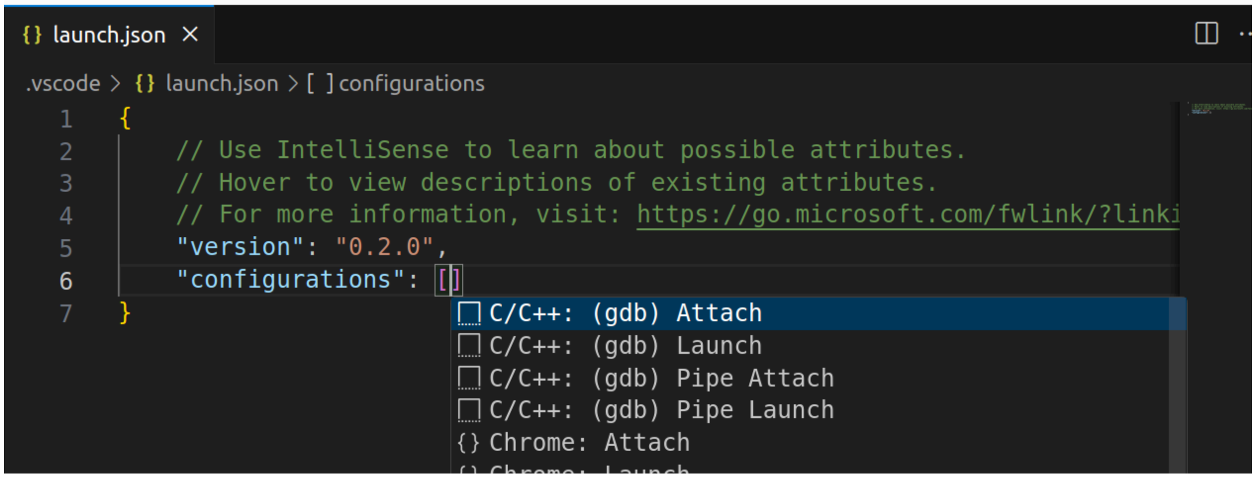
Creating a configuration file

Just like we saw in class, go to the “Run and Debug” section in VSCode, and click on “create a launch.json file”

From the suggested options, choose “C++ (GDB/LLDB)”

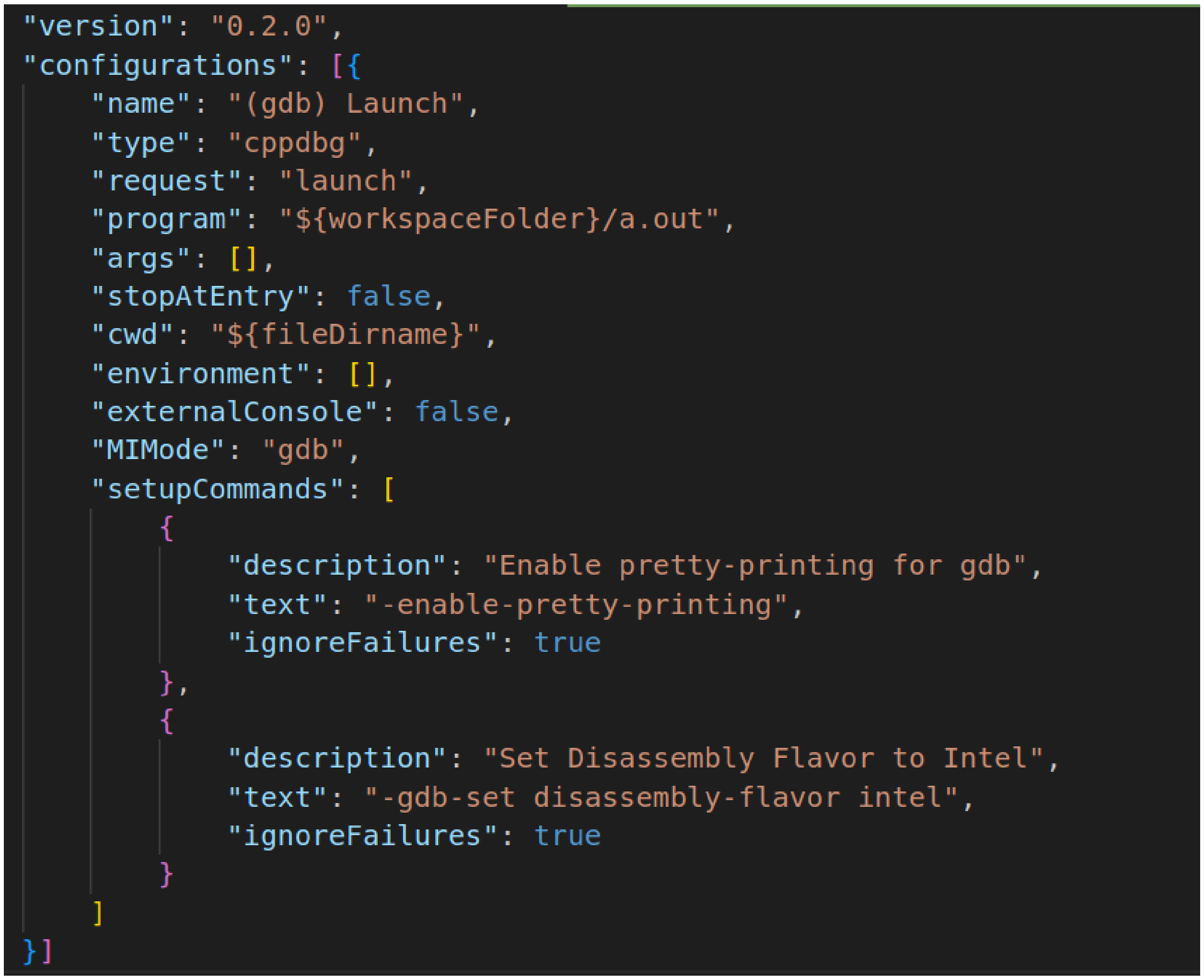


Now, you should have a “launch.json” file inside your “.vscode” directory. Open launch.json, place your cursor inside the “configuration” key, and press “ctrl + space” on your keyboard.



```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linki
    "version": "0.2.0",
    "configurations": []
}
```

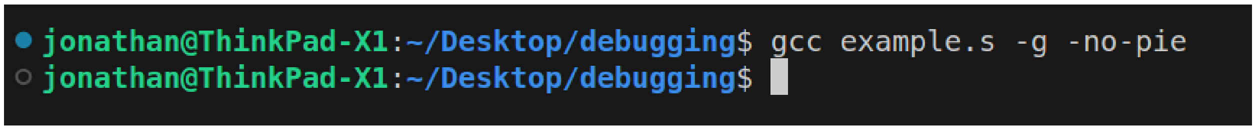
Choose “C/C++: (gdb) Launch”. Again, like we saw in class, you can change the configuration name, add command line arguments etc. But most importantly, change the “program” key to the path of your executable!



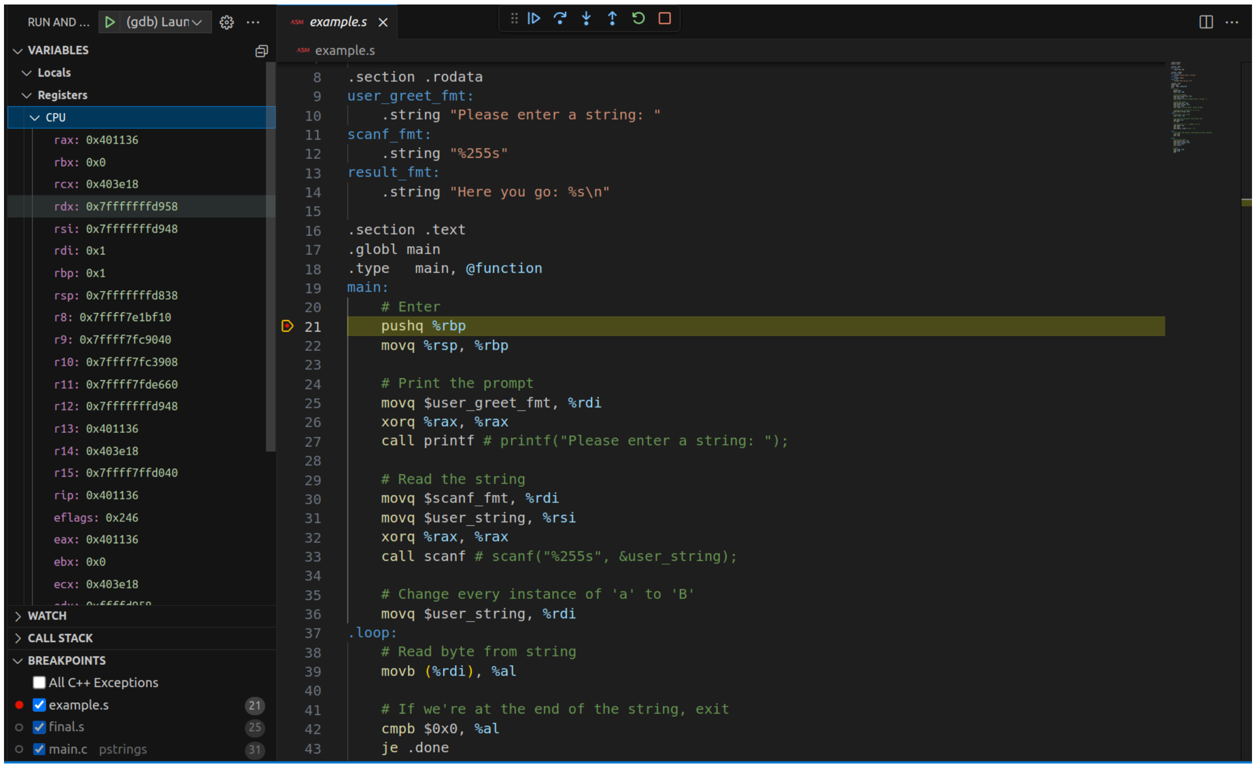
```
"version": "0.2.0",
"configurations": [{
    "name": "(gdb) Launch",
    "type": "cppdbg",
    "request": "launch",
    "program": "${workspaceFolder}/a.out",
    "args": [],
    "stopAtEntry": false,
    "cwd": "${fileDirname}",
    "environment": [],
    "externalConsole": false,
    "MIMode": "gdb",
    "setupCommands": [
        {
            "description": "Enable pretty-printing for gdb",
            "text": "-enable-pretty-printing",
            "ignoreFailures": true
        },
        {
            "description": "Set Disassembly Flavor to Intel",
            "text": "-gdb-set disassembly-flavor intel",
            "ignoreFailures": true
        }
    ]
}]
```

Compiling and Debugging

Compile your program using the “-g” file and place breakpoints wherever you wish. Then, go to the “Run and Debug” section again, and start debugging :).



```
jonathan@ThinkPad-X1:~/Desktop/debugging$ gcc example.s -g -no-pie
jonathan@ThinkPad-X1:~/Desktop/debugging$
```



```
rax: 0x401136
rbx: 0x0
rcx: 0x403e18
rdx: 0x7fffffffd958
rsi: 0x7fffffffd948
rdi: 0x1
rbp: 0x1
rsp: 0x7fffffffd838
r8: 0x7ffff7e1bf10
r9: 0x7ffff7fc9040
r10: 0x7ffff7fc3908
r11: 0x7ffff7fde660
r12: 0x7fffffffd948
r13: 0x401136
r14: 0x403e18
r15: 0x7ffff7ffd040
rip: 0x401136
eflags: 0x246
eax: 0x401136
ebx: 0x0
ecx: 0x403e18
```

```
.section .rodata
user_greet_fmt:
    .string "Please enter a string: "
scanf_fmt:
    .string "%255s"
result_fmt:
    .string "Here you go: %s\n"

.section .text
.globl main
.type   main, @function
main:
    # Enter
    pushq %rbp
    movq %rsp, %rbp

    # Print the prompt
    movq $user_greet_fmt, %rdi
    xorq %rax, %rax
    call printf # printf("Please enter a string: ");

    # Read the string
    movq $scanf_fmt, %rdi
    movq $user_string, %rsi
    xorq %rax, %rax
    call scanf # scanf("%255s", &user_string);

    # Change every instance of 'a' to 'B'
    movq $user_string, %rdi
.loop:
    # Read byte from string
    movb (%rdi), %al

    # If we're at the end of the string, exit
    cmpb $0x0, %al
    je .done
```