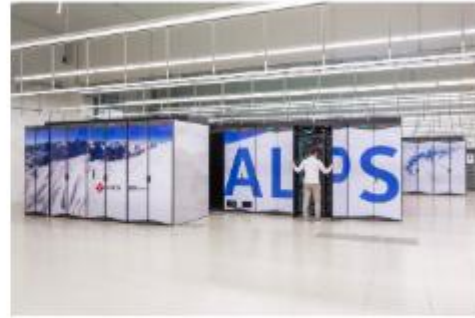
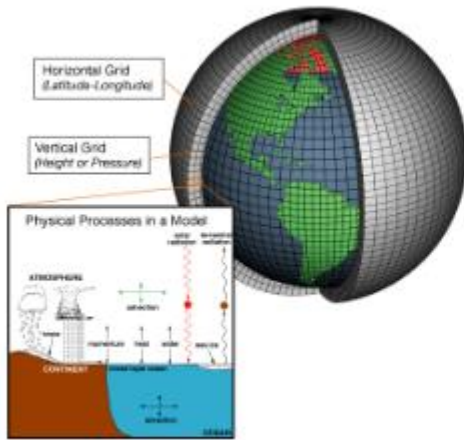


# C introduction part 1

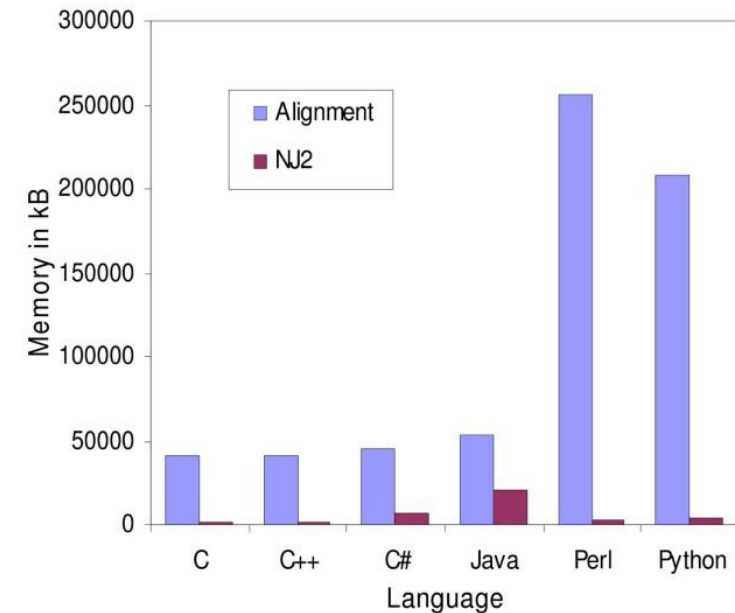
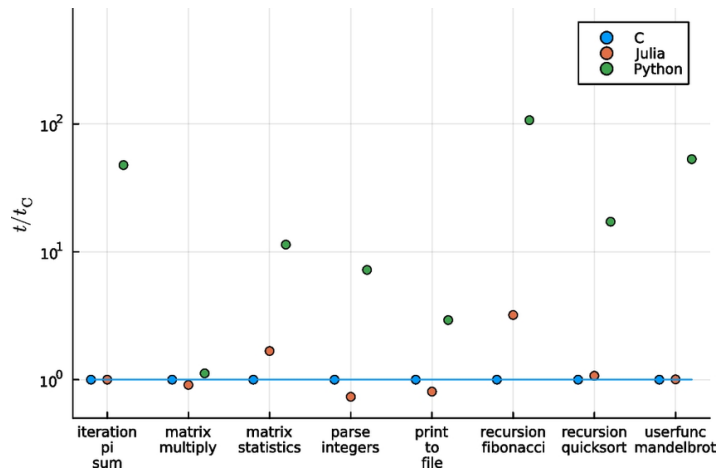
ENG-270

# Why C?

## Blazing fast speed



## Low memory footprint



# Why is C more efficient / faster?

- Explicit allocation of resources (memory)
- Type declarations lets compiler know what will be computed
- Compiler can optimize computation
  
- Dynamic, interpreted languages (e.g, Python) cannot optimize because they don't know what data will be fed to the program until runtime

Language	Type	Compiled/Interpreted
C	statically typed	compiled
Python	dynamically typed	interpreted
MATLAB	dynamically typed	JIT (compiled piecewise)

# Steps for code execution

## **Compiled language**

- Write code
- Compile in terminal
- Fix compilation errors
- Execute
- Check results
- Debug

## **Interpreted language**

- Write code
- Execute parts of your code in REPL
- Check results
- Write more code
- Execute more parts of your code in REPL
- Check results
- Debug

# Anatomy of a C program

Sum of natural numbers up to 10

$sum = 1 + 2 + 3 + \dots + 10$

C code

```
#include <stdio.h>
int main() {
    int n, i, sum = 0;

    printf("Enter a positive integer: ");
    scanf("%d", &n);

    for (i = 1; i <= n; ++i) {
        sum += i;
    }

    printf("Sum = %d", sum);
    return 0;
}
```

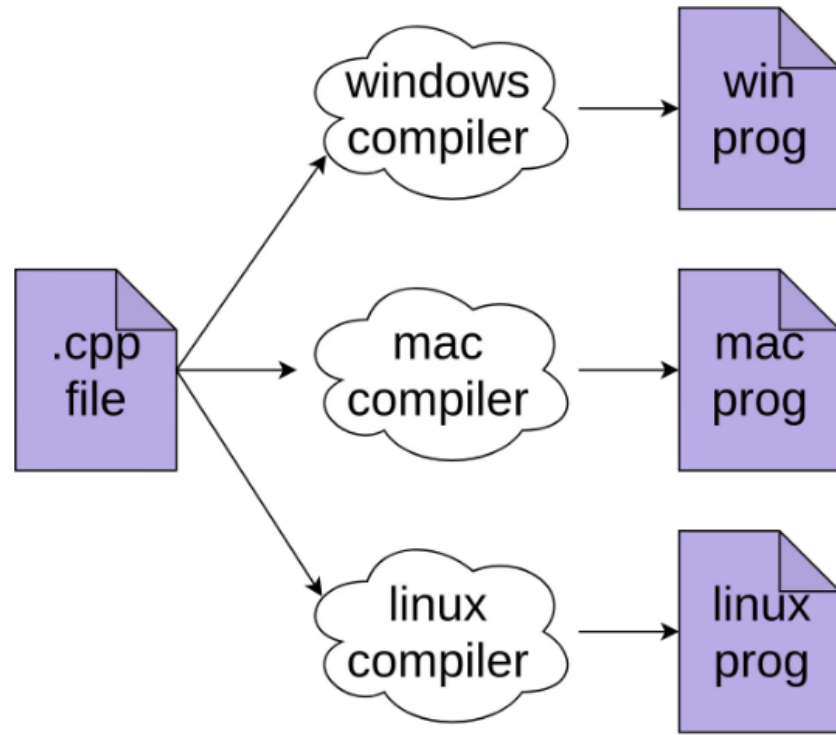
Basic elements

- `#include` statement to import libraries
- `main()` function with return value of 0
- required type annotations everywhere
- semicolon is *required* at the end of each statement

# Compiler vs. Interpreter

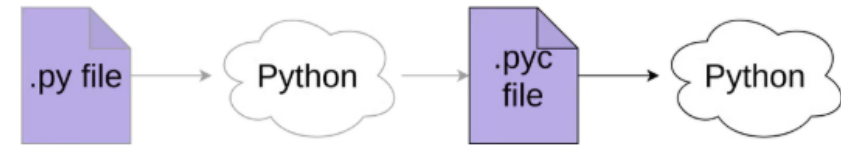
## C compiler

once program is compiled, can run on machine it was compiled for without user having to install C



## Python interpreter

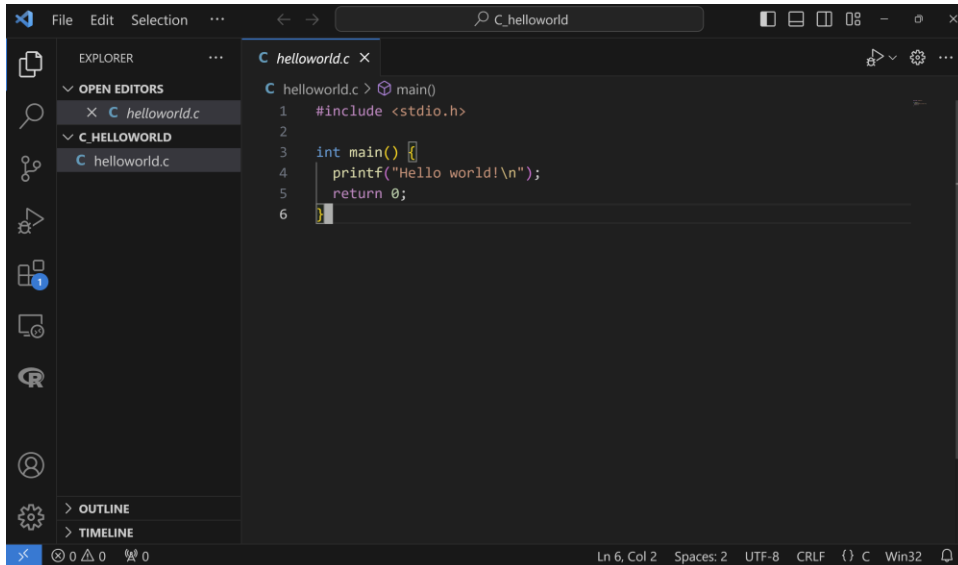
user needs to install Python to run Python scripts



# First program

## VSCoDe script window

1. Write the program and save it to a file (“helloworld.c”)

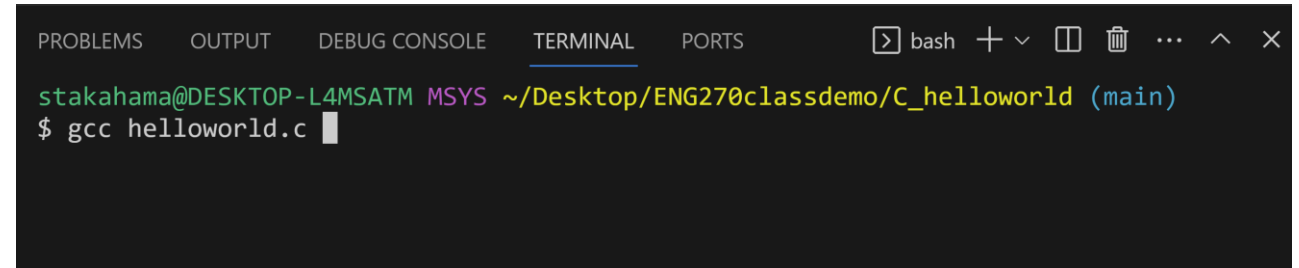


The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'C\_HELLOWORLD' with a file 'helloworld.c' open. The main editor window displays the following C code:

```
helloworld.c > main()
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello world!\n");
5     return 0;
6 }
```

## VSCoDe terminal

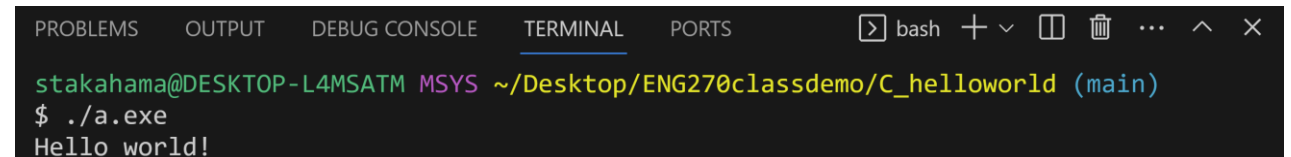
2. Compile the program to generate an executable



The screenshot shows the VS Code terminal window with the following text:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash + v [ ] [ ] ... ^ x
stakahama@DESKTOP-L4MSATM MSYS ~/Desktop/ENG270classdemo/C_helloworld (main)
$ gcc helloworld.c
```

3. Run the program



The screenshot shows the VS Code terminal window with the following text:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash + v [ ] [ ] ... ^ x
stakahama@DESKTOP-L4MSATM MSYS ~/Desktop/ENG270classdemo/C_helloworld (main)
$ ./a.exe
Hello world!
```

```
int main() {
```

**or**

```
int main( int argc, char *argv[] ) {
```

**?**

see [https://stakahama.gitlab.io/sie-eng270/intro\\_C.html#hello-world](https://stakahama.gitlab.io/sie-eng270/intro_C.html#hello-world)

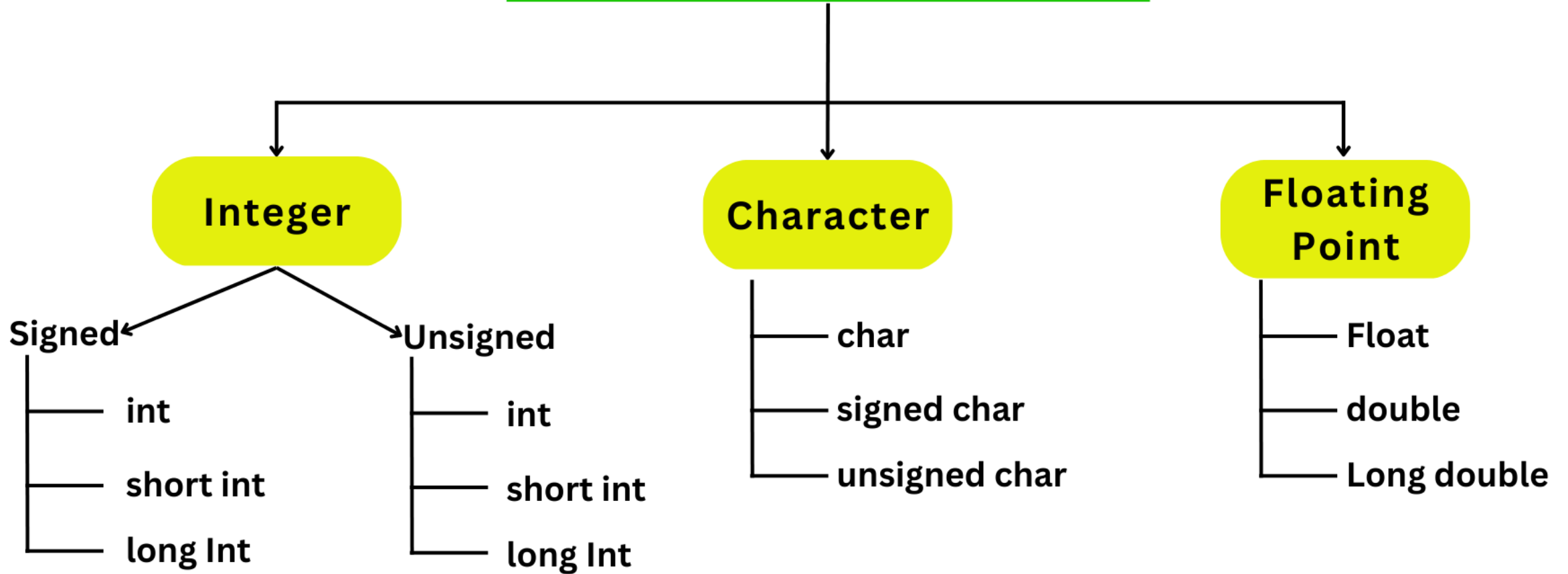
# Terminal

- Interact with the operating system. Typical commands (non-exhaustive!):  
<https://sieprog.ch/#terminal>
  - `cd`: change directory (“.” current directory, “..” up one directory)
  - `pwd`: check current working directory
  - `gcc`: compile program with gcc (run compiled file “*program*” with “./*program*”)
  - `gdb`: run gdb
  - `echo`: print
  - `ls`: list contents of directory
- POSIX-compliant shell
  - macOS, Linux – default
  - Windows – MSYS2 / Git Bash. Note path separator “/” for POSIX-compliant, “\” for Windows cmd
- **USE TAB COMPLETION!!! (to save yourself keystrokes)**

# Back to C: Data types

- You must declare data types for all variables and functions.
  - <https://sieprog.ch/#c/variables>
  - <https://sieprog.ch/#c/fonctions>
- Where?
  - Close to where it is used
  - At the beginning of the function
  - Within a loop (if used only in the loop)
- Do not forget to initialize variables when you declare them.

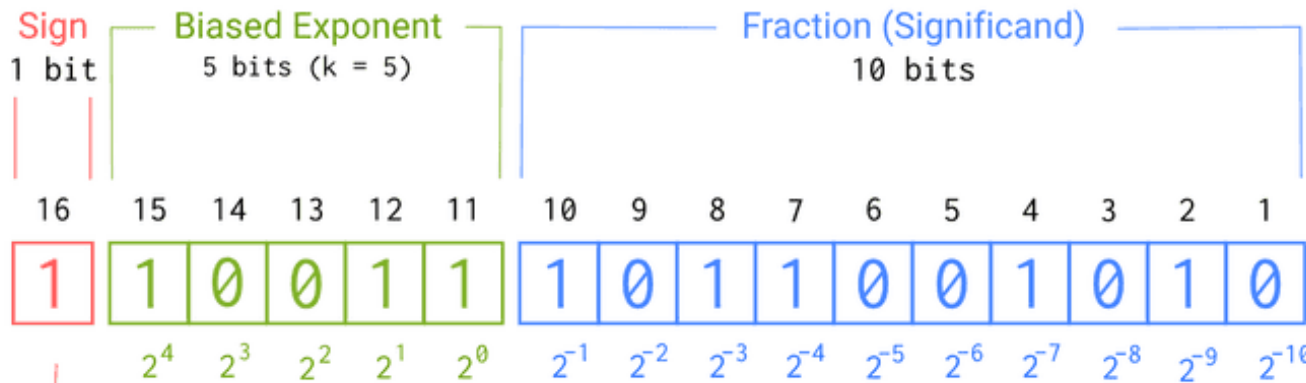
# Primary/ Basic Data Types





# Half-precision 16 bits

trekhele.dev



= -27.15625

exponent =  $2^4 + 2^1 + 2^0 = 19$

bias =  $2^{k-1} - 1 = 2^{5-1} - 1 = 15$

biased\_exponent = exponent - bias = 19 - 15 = 4

fraction =  $2^{-1} + 2^{-3} + 2^{-4} + 2^{-7} + 2^{-9} =$

$= 0.5 + 0.125 + 0.0625 + 0.0078125 + 0.001953125 =$

$= 0.697265625$

$-1^1 \times 2^4 \times (2^0 + 0.697265625) = -27.15625$

# Size and range of values

For reference – your computer typically has 8-16 GB ( $10^9$  bytes) of memory (*RAM: Random Acces Memory*)

Data Type	Range	Bytes	Format
signed char	-128 to + 127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

Note: The sizes and ranges of int, short and long are compiler dependent. Sizes in this figure are for 16-bit compiler.

1 byte = 8 bits

# Pointer data types

- Essential for programming in C
- Covered in future lesson

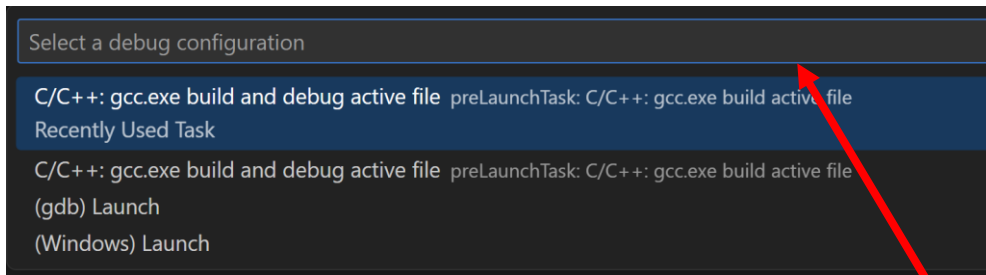
# Debugging

- Use print statements or debugger
- gdb (GNU debugger)
  - from command line
    - <https://sieprog.ch/#gdb>
    - [https://www.tutorialspoint.com/gnu\\_debugger/gdb\\_commands.htm](https://www.tutorialspoint.com/gnu_debugger/gdb_commands.htm)
  - through VS Code
    - <https://code.visualstudio.com/docs/cpp/cpp-debug>
    - <https://code.visualstudio.com/docs/cpp/config-mingw> (Windows)
- lldb is also available, but gdb is recommended. lldb is part of the clang compiler suite and gdb is part of the gcc compiler suite, but, in principle, the two can be used with either compiler.
- Note that C++ is a superset of C; many tutorials on C++ (e.g., installation, debugging) will apply to C.

# GDB in VS Code

step 0 click where you want your program to stop to inspect variables – you should see a red dot appear

step 2 check that the debugger is set to right path (save)



step 1 click on the gear box

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(int argc, char * argv[]) {
5     // Conditions au départ
6     double surface = 47.69e6;
7     double niveau = 557.326;
8
9     // Evolution heure par heure
10    for (int i = 0; i < 48; i++) {
11        // Effluent en m³/s
12        double outflow = pow(niveau - 557.05, 2.0) * 250;
13
14        // Affluent Aare + Kander en m³/s
15        double inflow = 13 + 6;
16
17        // Mettre à jour le niveau
18        double volumeDifference = (inflow - outflow) * 3600;
19        niveau = niveau + volumeDifference / surface;
20
21        // Affichage
22        printf("%4d\t%0.6f\t%0.6f\t%0.6f\n", i + 1, niveau, inflow, outflow);
23    }
24 }
```

gear box

```
1 {
2     "configurations": [
3         {
4             "name": "C/C++: gcc.exe build and debug active file",
5             "type": "cppdbg",
6             "request": "launch",
7             "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
8             "args": [],
9             "stopAtEntry": false,
10            "cwd": "${fileDirname}",
11            "environment": [],
12            "externalConsole": false,
13            "miDebuggerPath": "C:\\git-sdk-64\\mingw64\\bin\\gdb.exe",
14            "setupCommands": [
15                {
16                    "description": "Enable pretty-printing for gdb",
17                    "text": "-enable-pretty-printing",
18                    "ignoreFailures": true
19                },
20                {
21                    "description": "Set Disassembly Flavor to Intel",
22                    "text": "-gdb-set disassembly-flavor intel",
23                    "ignoreFailures": true
24                }
25            ],
26            "preLaunchTask": "C/C++: gcc.exe build active file"
27        }
28    ],
29    "version": "2.0.0"
30 }
31 }
```

## step 3 select debug C/C++ file

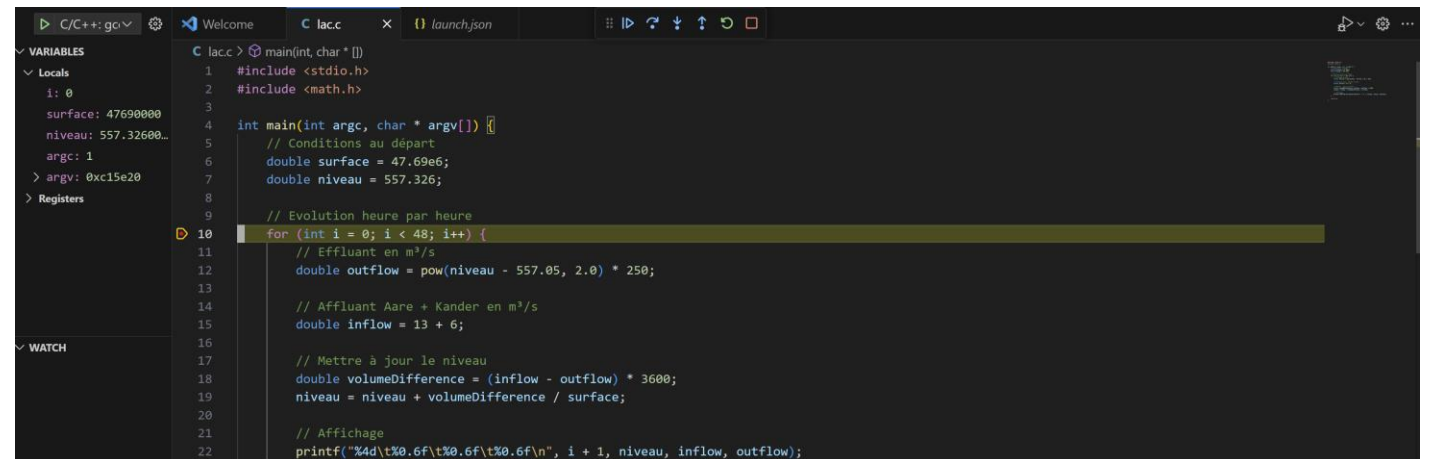


The screenshot shows the Visual Studio Code editor with a C program open in the file 'lacc.c'. The code is as follows:

```
C lacc > main(int, char * [])
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(int argc, char * argv[]) {
5     // Conditions au départ
6     double surface = 47.69e6;
7     double niveau = 557.326;
8
9     // Evolution heure par heure
10    for (int i = 0; i < 48; i++) {
11        // Effluent en m³/s
12        double outflow = pow(niveau - 557.05, 2.0) * 250;
13
14        // Affluent Aare + Kander en m³/s
15        double inflow = 13 + 6;
16
17        // Mettre à jour le niveau
18        double volumeDifference = (inflow - outflow) * 3600;
19        niveau = niveau + volumeDifference / surface;
20
21        // Affichage
22        printf("%d\t%.6f\t%.6f\t%.6f\n", i + 1, niveau, inflow, outflow);
23    }
24 }
```

In the top right corner, a context menu is open, with the option 'Debug C/C++ File' highlighted by a red rectangle. Below it, the option 'Run C/C++ File' is visible.

## step 4 enjoy!



The screenshot shows the Visual Studio Code editor with the same C program open. The program is running in debug mode, and the 'VARIABLES' pane on the left is expanded to show the current state of the program. The 'Locals' pane shows the following values:

Variable	Value
i	0
surface	47690000
niveau	557.326000
argc	1
argv	0xc15e20

The 'Registers' pane is also visible, showing the value of the 'eax' register as 0xc15e20. The 'WATCH' pane is currently empty. The code in the editor is highlighted at line 10, which is the start of the for loop: `for (int i = 0; i < 48; i++) {`.

# Testing installation

Is the problem with the compiler, or with VS Code calling the compiler?

→ USE THE TERMINAL

Is the right compiler being used?

→ type `which gcc` without quotes

Am I in the right directory?

→ type `pwd`, and then `ls` to see what's in the directory