

C introduction part 1

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

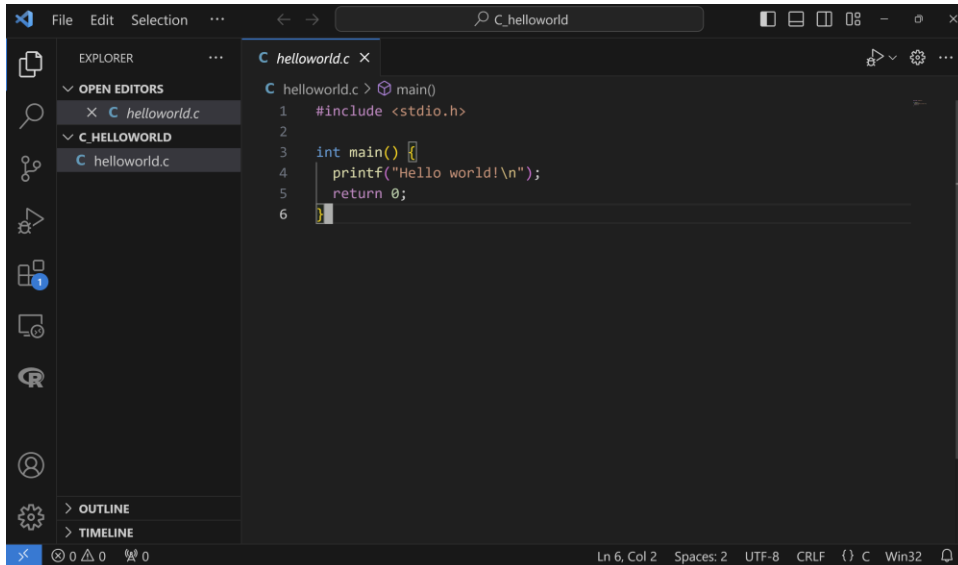
Interpreted language

- 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

First program

VSCode script window

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

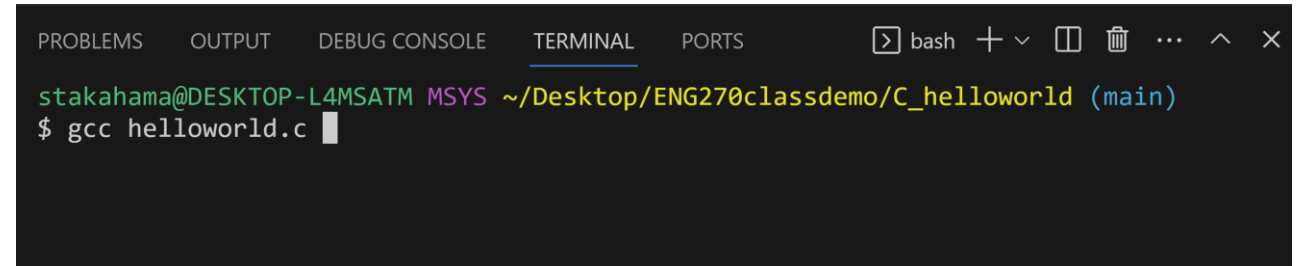


A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'C_HELLOWORLD' with a file 'helloworld.c' selected. The main editor window displays the contents of 'helloworld.c', which is a C program that includes <stdio.h> and has a main function that prints 'Hello world!\n' and returns 0. The status bar at the bottom indicates the cursor is at line 6, column 2, with 2 spaces, UTF-8 encoding, CRLF line endings, and a C language mode.

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

VSCode terminal

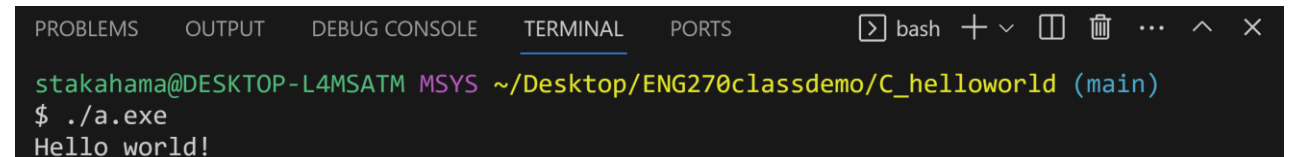
2. Compile the program to generate an executable



A screenshot of the VS Code terminal window. The terminal shows the user 'stakahama@DESKTOP-L4MSATM' in the 'MSYS' shell, located in the directory '~/Desktop/ENG270classdemo/C_helloworld'. The user has entered the command 'gcc helloworld.c' to compile the program.

```
stakahama@DESKTOP-L4MSATM MSYS ~/Desktop/ENG270classdemo/C_helloworld (main)
$ gcc helloworld.c
```

3. Run the program



A screenshot of the VS Code terminal window showing the execution of the program. The user has entered the command './a.exe' to run the executable. The terminal output shows 'Hello world!'.

```
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/C_intro.html#org0bae946

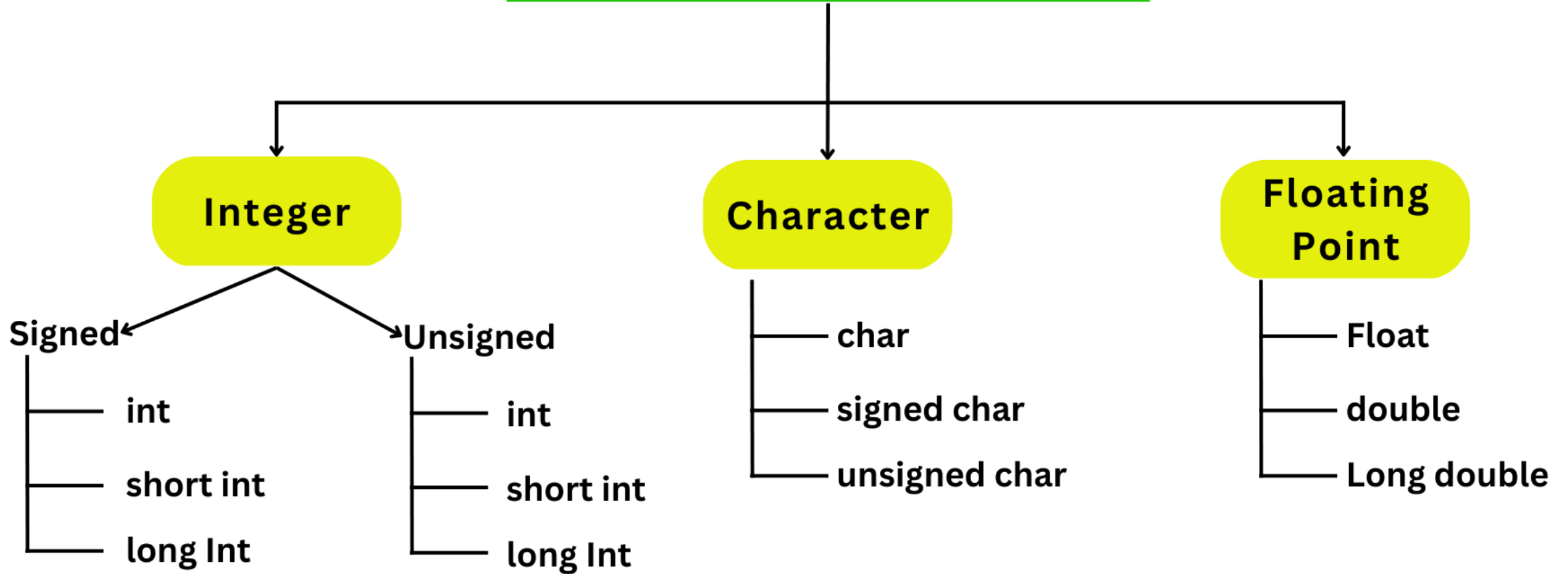
Terminal

- 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

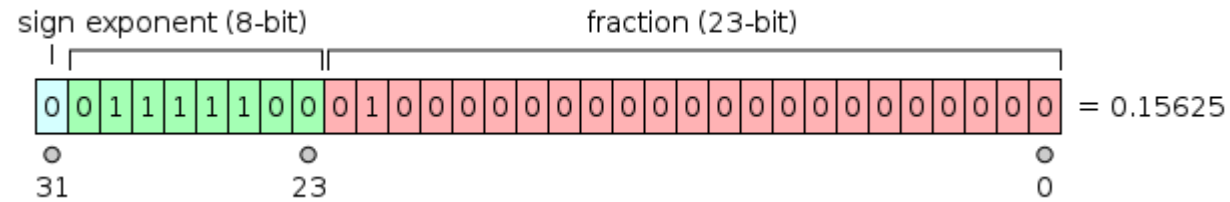
- You must declare data types for all variables and functions.
 - <https://sieprog.ch/#c/variables>
 - <https://sieprog.ch/#c/fonctions>
- Where?
 - Close to where it is used
 - At the beginning of the function
 - Within a loop (if used only in the loop)
- Do not forget to initialize variables when you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

Primary/ Basic Data Types



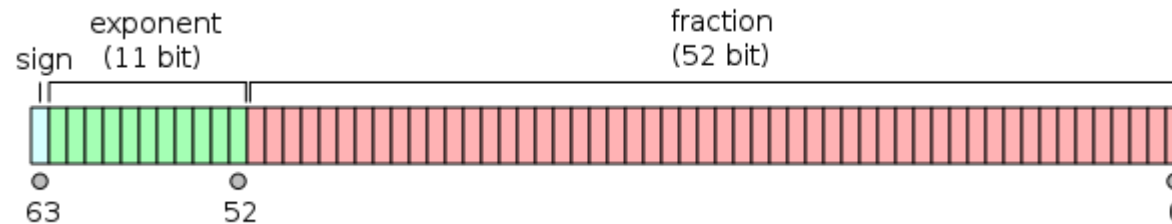
Floating point representation

32-bit
(32 binary digits)



float
single precision

64-bit
(64 binary digits)



double
double precision

Size and range of values

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
- See next lesson

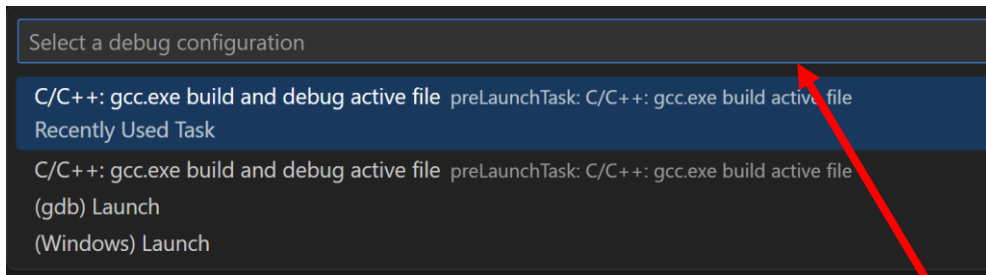
Debugging

- Use print statements or debugger
- gdb (GNU debugger)
 - from command line
 - <https://sieprog.ch/#gdb>
 - https://www.tutorialspoint.com/gnu_debugger/gdb_commands.htm
 - 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

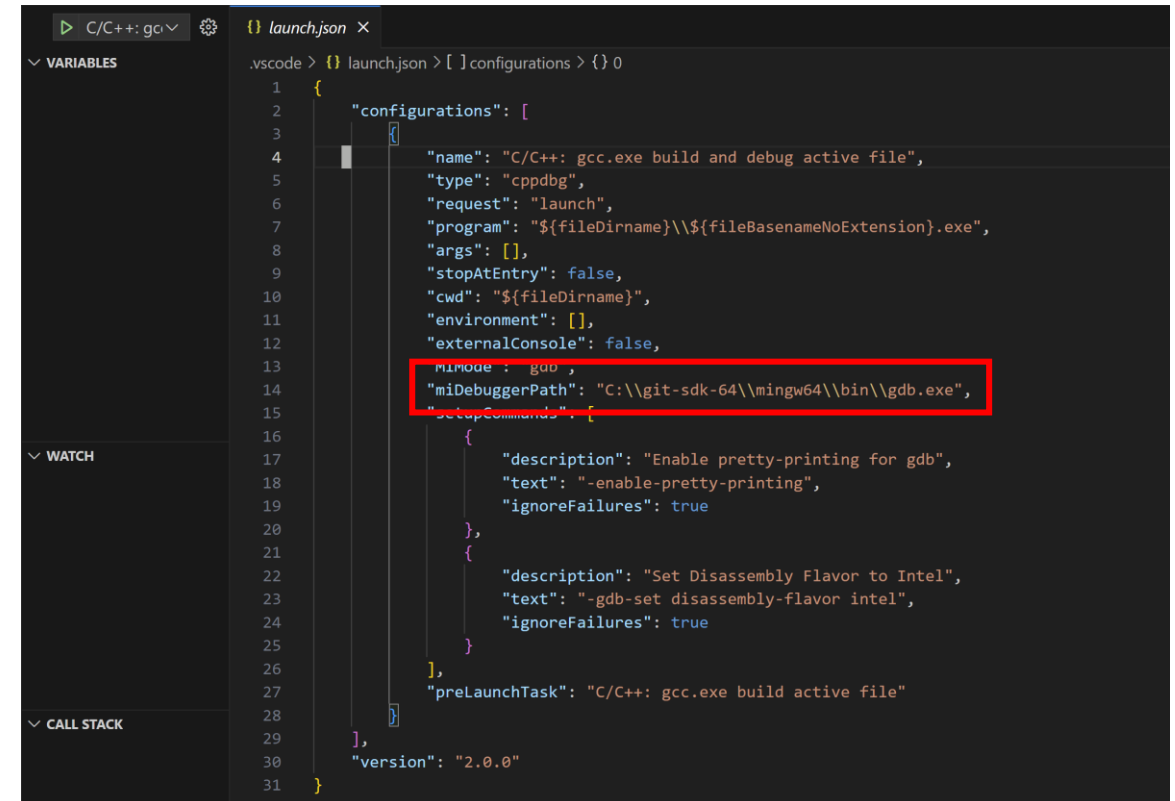
step 2 check that the debugger is set to right path (save)



step 1 click on the gear box



gear box

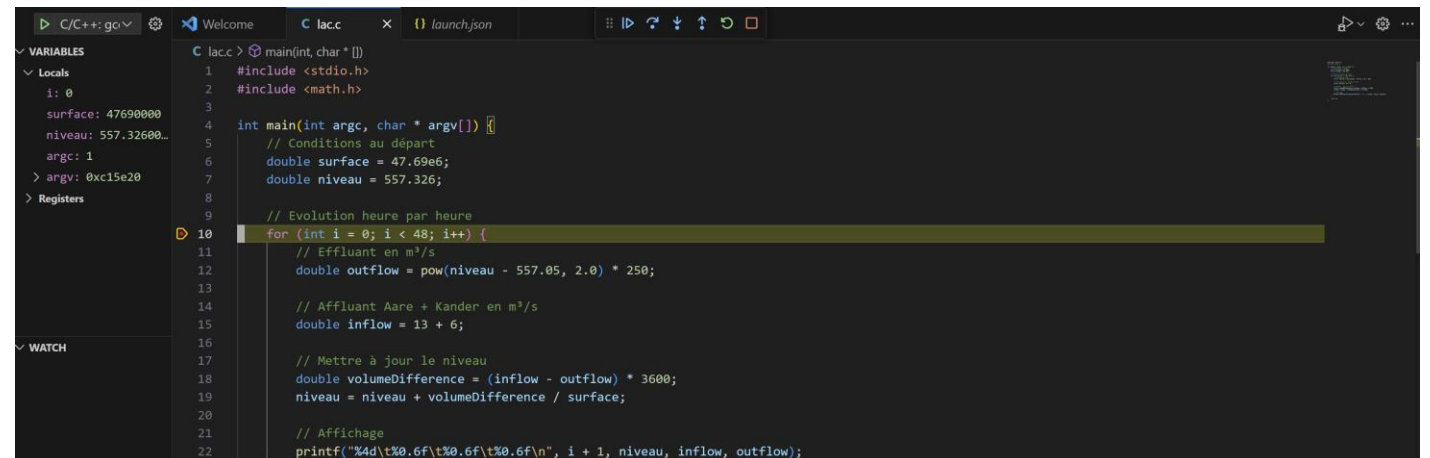


step 3 select debug C/C++ file



```
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     }
```

step 4 enjoy!



```
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     }
```

VARIABLES

- Locals
 - i: 0
 - surface: 47690000
 - niveau: 557.326000
 - argc: 1
 - argv: 0xc15e20
- Registers

WATCH

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 (“where gcc” on Windows cmd)

Am I in the right directory?

→ type “pwd” without quotes, and then “ls” to see what’s in the directory