



CoE 164

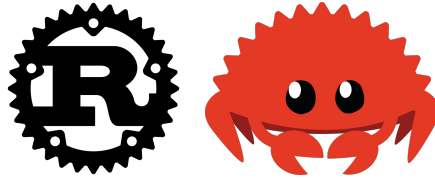
Computing Platforms

01a: About Rust

RUST PROGRAMMING LANGUAGE

The **Rust programming language** is a high-level language emphasizing performance, type-safety, and concurrency.

Rust is popular in systems programming but can be used as a general-purpose programming language.



RUST: HISTORY

Rust was developed in 2006 at Mozilla Research and appeared to the public in 2010. Since 2021, it is now managed by the Rust Foundation.

Since then, Rust has enjoyed considerable usage among programmers who wanted to write memory-safe code.



◆ RUST: FEATURES



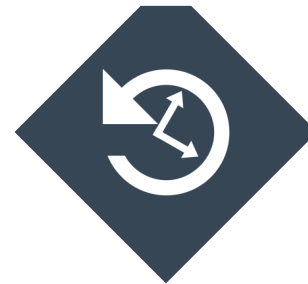
Memory-safe

Rust enforces checks at compile-time to ensure that each memory manipulation is done deliberately and properly.



Performance

Because of a lack of runtime checks, Rust programs can be fast and memory efficient compared to other programs.



Concurrency

Writing concurrent code is much easier and safer. Data temporality is much more predictable.

PLAYGROUND

1. Visit the Rust language playground at <https://play.rust-lang.org>
2. Write your program, noting that Rust will start the program by executing the `main()` function.
3. Click “Run” to compile and run the program.



The screenshot shows the Rust playground interface. The code editor contains the following Rust code:

```
1 fn main() {  
2     println!("Hello, world!");  
3 }
```

The interface includes a top bar with buttons for "RUN", "DEBUG", "STABLE", "SHARE", "TOOLS", and "CONFIG". The code editor is split into two panes, with the code on the left and the output on the right.



◆ INSTALLATION (WINDOWS)

1. Download and install the [Visual Studio C++ build tools](#) before proceeding.
 - a. Install the "MSBuild Tools" and "Desktop development with C++" components.
2. Visit the Rust language website at <https://www.rust-lang.org>
3. Find the links leading to downloading `rustup-init` and download it into your computer.
4. Run the installer and follow the onscreen instructions.
5. Note that the installer is command-line based, so you need to input the selected option and press "Enter" to proceed. When in doubt, select the "default" setting.



◆ INSTALLATION (LINUX/MACOS)

1. Open your preferred terminal. Go to your “Desktop” or “Downloads” folder - preferably where you will place the `rustup` installation script.
2. Type the following to download the installation script.

```
wget https://sh.rustup.rs > rustup.sh
```
3. Type the following to run the installer

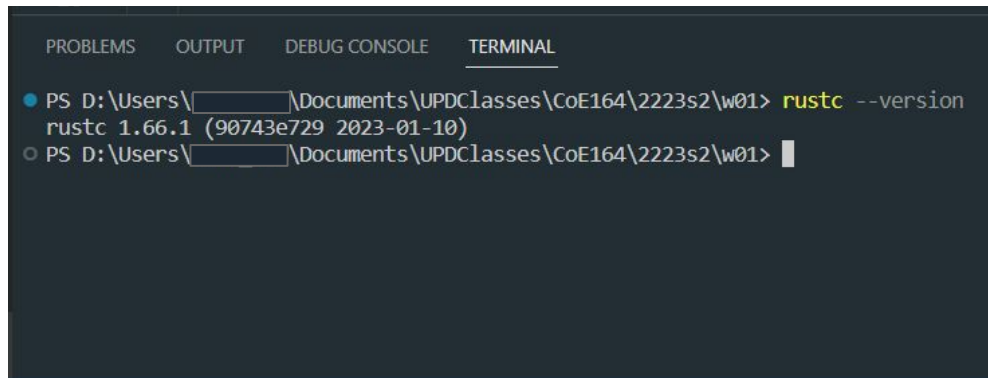
```
sh ./rustup.sh
```

and follow the onscreen instructions.
4. Note that the installer is command-line based, so you need to input the selected option and press “Enter” to proceed. When in doubt, select the “default” setting.

INSTALLATION TEST

Open your terminal and type the following to view the version of `rustc` and `cargo` that you have. A version string should appear.

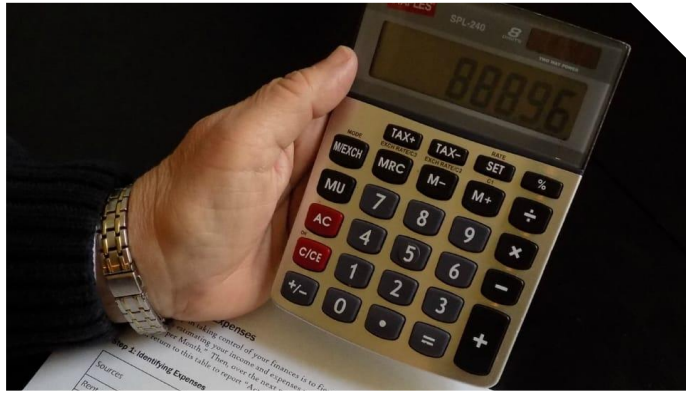
```
rustc --version  
cargo --version
```



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
● PS D:\Users\[redacted]\Documents\UPDClasses\CoE164\2223s2\w01> rustc --version  
rustc 1.66.1 (90743e729 2023-01-10)  
○ PS D:\Users\[redacted]\Documents\UPDClasses\CoE164\2223s2\w01> |
```


CONSIDER...

... a program that takes in two space-separated integers and adds them together.



◆ ADDER: INPUT

Example

```
use std::io;

fn main() {
    println!("Please input two \
            space-separated integers");

    // Get a line
    let mut str_in = String::new();
    io::stdin()
        .read_line(&mutstr_in)
        .expect("Failed to read input");

    // Split into two
    let str_in_split: Vec <&str> = str_in
        .split(' ')
        .collect();
```

◆ ADDER: INPUT + ROUTINE

Example

```
if str_in_split.len() != 2 {
    panic!("Input does not contain two integers!");
}

let a: u64 = str_in_split[0]
    .trim()
    .parse()
    .expect("Input is not an integer!");

let b: u64 = str_in_split[1]
    .trim()
    .parse()
    .expect("Input is not an integer!");

let ans = a + b;
println!("Calculation: {a} + {b} = {c}");
}
```



RUNNING RUST CODE

1. Open your preferred program editor. Something that has syntax highlighting is required.
2. Copy and paste the `adder.rs` code to the editor and save it as `adder.rs` somewhere in your computer.
3. Open the terminal and go to the directory where `adder.rs` is located.
4. Type the following in the terminal to compile the program
`rustc adder.rs`
5. You should have an executable file named `adder` in that same directory. Run it by typing the following in the terminal
`adder.exe` # Windows
`./adder` # Linux/MacOS

ADDER: COMPILE ERRORS

Compiling the previous code as *is* will yield compile errors. Rust will inform you where and what the errors are and may hint you on how to fix them.

As an exercise, try to fix the code using the messages below.

```
PS D:\Users\<redacted>\Documents\UPDClasses\CoE164\2223s2\w01> rustc .\adder.rs
error[E0425]: cannot find value `mutstr_in` in this scope
  --> .\adder.rs:10:21
   |
10 |         .read_line(&mutstr_in)
   |                   ^^^^^^^^^^ help: a local variable with a similar name exists: `str_in`

error[E0425]: cannot find value `c` in this scope
  --> .\adder.rs:32:41
   |
32 |         println!("Calculation: {a} + {b} = {c}");
   |                                         ^ help: a local variable with a similar name exists: `a`

error: aborting due to 2 previous errors

For more information about this error, try `rustc --explain E0425`.
PS D:\Users\<redacted>\Documents\UPDClasses\CoE164\2223s2\w01> █
```

FIXED ADDER: INPUT

Example

```
use std::io;

fn main() {
    println!("Please input two \
            space-separated integers");

    // Get a line
    let mut str_in = String::new();
    io::stdin()
        .read_line(&mut str_in)
        .expect("Failed to read input");

    // Split into two
    let str_in_split: Vec <&str> = str_in
        .split(' ')
        .collect();
}
```

FIXED ADDER: INPUT + ROUTINE

Example

```
if str_in_split.len() != 2 {
    panic!("Input does not contain two integers!");
}

let a: u64 = str_in_split[0]
    .trim()
    .parse()
    .expect("Input is not an integer!");

let b: u64 = str_in_split[1]
    .trim()
    .parse()
    .expect("Input is not an integer!");

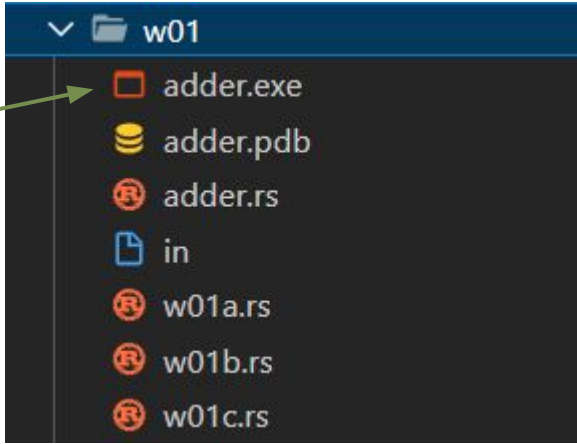
let ans = a + b;
println!("Calculation: {a} + {b} = {ans}");
}
```

ADDER: COMPILED

Compiling the previous code will yield nothing in the terminal. However, there will be an `.exe` (Windows) or ABI file (Linux/MacOS) generated.

```
PS D:\Users\<redacted>\Documents\UPDClasses\CoE164\2223s2\w01> rustc adder.rs  
PS D:\Users\<redacted>\Documents\UPDClasses\CoE164\2223s2\w01> |
```

Executable



- adder.exe
- adder.pdb
- adder.rs
- in
- w01a.rs
- w01b.rs
- w01c.rs

◆ ADDER: COMPILE WARNINGS

Sometimes compiling will result to compile warnings. It will still generate an executable file, but Rust will inform you where it thinks you should action on it.

The Rust compiler below complains about an unused variable in the program.

```
● PS D:\Users\[redacted]\Documents\UPDClasses\CoE164\2223s2\w01> rustc adder.rs
warning: unused variable: `x`
  --> adder.rs:7:9
7 |     let x = 3;
  |         ^ help: if this is intentional, prefix it with an underscore: `_x`
= note: `#[warn(unused_variables)]` on by default

warning: 1 warning emitted

○ PS D:\Users\[redacted]\Documents\UPDClasses\CoE164\2223s2\w01> █
```

RESOURCES

- [The Rust Book](#)





CoE 164

Computing Platforms

01a: About Rust