# 2 - A Guessing Game



## 用户IO

- 初始化项目：

```
$ cargo new guessing_game
```

- 主代码：

```
use std:: io;

fn main() {
    println!("Guessing the number!");

    println!("Please input your guess");

    let mut guess = String::new();

    io::stdin()
        .read_line(&mut guess)
        .expect("Failed to read line");

    println!("You guess: {}", guess);
}
```

  - `std::io`: Rust Standard Library中的IO Library.
  - `let mut guess = Sring::new()`: 创建一个可变的String变量
  - `.read_line(&mut guess)`: 将guess作为参数传入.read_line( )中，表示用户输入的string将被存储在guess这个变量中。
  - `&mut guess`: 暂时先理解成C++中的reference
  - `read_line()` 实际上也会返回一个变量叫`io::Result`，这个`Result` type是一个enumerations（一个set中有variants）
    - `Result`的variants有`OK`和`Err`, OK表示operation成功了，Err表示O配ration失败了。
    - `io::Result`同时有个**expect method**, 如果Result是Err的话，它就会让程序crash
- 编译、运行：

  output:

```
$ cargo run
   Compiling guessing_game v0.1.0
(/home/ubuntu/Desktop/Rust_Learning/2_A_Guessing_Game/guessing_game)
    Finished dev [unoptimized + debuginfo] target(s) in 2.15s
     Running `target/debug/guessing_game`
Guessing the number!
Please input your guess
3
You guess: 3
```

---

## 生成随机数

- Rust中一个重要的概念叫crate, 我的理解就是一个external library。比如如果我们要用rand的话，就要引入rand crate这个东西
- 在toml文件的dependencies下加上：

```
rand = "0.8.3"
```

- `cargo build`编译，可以看到生成了下载和编译了很多额外的依赖：

  output:

```
$ cargo build
    Updating crates.io index
  Downloaded rand v0.8.5
  Downloaded libc v0.2.121
  Downloaded getrandom v0.2.5
  Downloaded 3 crates (689.5 KB) in 3.52s
   Compiling libc v0.2.121
   Compiling cfg-if v1.0.0
   Compiling ppv-lite86 v0.2.16
   Compiling getrandom v0.2.5
   Compiling rand_core v0.6.3
   Compiling rand_chacha v0.3.1
   Compiling rand v0.8.5
   Compiling guessing_game v0.1.0
(/home/ubuntu/Desktop/Rust_Learning/2_A_Guessing_Game/guessing_game)
    Finished dev [unoptimized + debuginfo] target(s) in 15.33s
```

  - 注意：Rust是从`crate.io`中下载外部依赖的，`crate.io`is where people in the Rust ecosystem post their open source Rust projects for others to use.
  - `cargo.lock`:里面存贮了之前使用的crate的版本，在编译的时候，编译器也会优先去cargo.lock文件中下载之前一直用的版本（防止因为依赖的版本更新了导致不必要的bug）
    - 用`$ cargo update`: 编译器就不会去看lock中的版本，而直接更新到这个series下最新版本（e.g. 0.8.3 -> 0.8.4, 但如果要更新到0.9.x, 就需要手动去toml文件中改）
- 更改代码为：

```
use std::io;
use rand::Rng;

fn main() {
    println!("Guessing the number!");
```

```
    let secret_number = rand::thread_rng().gen_range(1..101);

    println!("The secrete number is {}", secret_number);

    println!("Please input your guess");

    let mut guess = String::new();

    io::stdin()
        .read_line(&mut guess)
        .expect("Failed to read line");

    println!("You guess: {}", guess);
}
```

- `use rand::Rng`: 使用rand中的Rng trait。 `Rng` trait 定义了生成随机数的实现方法。
- `rand::thread_rng().gen_range(1..101)`: 使用`rand::thread_rng()`这个function，`gen_range()`是function下的method。

---

## 比较输入和随机数

- 在代码中加入：

```
use std::cmp::Ordering;

// snip
match guess.cmp(&secret_number){
        Ordering::Less => println!("Too small!"),
        Ordering::Greater => println!("Too big!"),
        Ordering::Equal => println!("You win!"),
}
```

- `use std::cmp::Ordering`: Ordering 和 io.Result一样是个enums类型，包含三个variants：Less, Greater, Equal
- `guess.cmp(&secret_number)`： 比较，并返回一个variant of Ordering.
- `match`: 由很多个arms组成，感觉可以理解为C++中的switch

- 并且我们注意到，guess和secret_number不是同一个数据类型，rand生成的数据类型是`i32`，我们需要把string转化为i32:

```
let guess: i32 = guess.trim().parse.expect("Please type a number!");
```

- 注意，这里guess的值**shadow**了原来guess的值
- `trim()`: 忽略string前后的所有空格，换行符啥的
- `.parse`是转换数据类型，`let guess: i32`: 告知想转换为i32. `.parse`返回的数据是Result（前面的io讲过），交由expect判断。

---

## Loop

- 代码上加上循环：

```
loop{
        println!("Please input your guess");
```

```
        // --snip --

        match guess.cmp(&secret_number){
            Ordering::Less => println!("Too small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal => {
                println!("You win!");
                break;
            }
        }
    }
```

- 在match为`Ordering::Equal`后执行break跳出循环。

---

## 处理非法输入

- 将guess的转换类型改为：

```
let mut guess = String::new();

io::stdin()
    .read_line(&mut guess)
    .expect("Failed to read line");

let guess: i32 = match guess.trim().parse(){
    Ok(num) => num,
    Err(_) => continue,
};
```

- 注意`parse()`返回的类型是Result，如果正确执行的话，转换成的结果将被裹挟在Ok( )中。我们从Ok(num) 中把num取出来。
- `Err(_)`的意思是match所有的Err
- `continue`：重新开始执行loop

---

## 完整代码

```
use std::io;
use rand::Rng;
use std::cmp::Ordering;

fn main() {
    println!("Guessing the number!");

    let secret_number = rand::thread_rng().gen_range(1..101);

    loop{
        println!("Please input your guess");

        let mut guess = String::new();

        io::stdin()
            .read_line(&mut guess)
            .expect("Failed to read line");
```

```
        let guess: i32 = match guess.trim().parse(){
            Ok(num) => num,
            Err(_) => continue,
        };

        println!("You guess: {}", guess);

        match guess.cmp(&secret_number){
            Ordering::Less => println!("Too small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal => {
                println!{"You win!"};
                break;
            }
        }
    }
}
```