Rust's std Library

- This is a very large library and you are encouraged to take a peek at it
  - https://doc.rust-lang.org/std
- Here we only look at two things:
  - Strings
  - Basic I/O

Strings

- Rust has TWO string types
  - str – string constants/string literals – these are strings allocated in static memory
  - String – strings allocated on the heap
- Being a system programming language Rust is very concerned with memory usage.
  - Think of an embedded system with limited RAM
  - you don’t want to fill up RAM with string constants
  - therefore Rust allows the developer this additional level of control of how strings are being used
  - Static strings can be stored in ROM rather than RAM

Strings

- Here is an example snippet using the ‘str’ type
- String constants live in static memory therefore you can only get references to them
- You should only use ‘str’ if you are really concerned about memory usage – not the case in our 64-bit memory laptops.
- The library info on this type can be found here:
  - https://doc.rust-lang.org/std/primitive.str.html

```
1 fn main() {
2     let message: &'static str = "Hello, world!";
3     println!("{}", message);
4 }
```

Strings

- Enter the other string type: String
  - https://doc.rust-lang.org/std/string/
- This string type behaves much like the strings we are used to from other languages
  - No references needed
- The only catch is that you will have to explicitly convert string literals into this String type.

```
1 fn main() {
2     let message: String = "Hello World!".to_string();
3     println!("{}", message);
4 }
```

I/O

- We have seen one I/O related operation – the println macro
- Here we take a look at
  - Reading and writing to/from stdin and stdout, respectively
  - Reading and writing files

I/O

- The key to Rust I/O is that we have file handles and traits that these file handles implement.
- Setting up I/O operations is mostly driven by importing the correct file handles and the appropriate traits.

Stdin

- Here we are importing the file handle stdin and the trait Read
- We then call read_to_string on the file handle which is a function the trait Read implements
- All I/O functions return the type Result which is an enum with variants Ok and Err.

```
enum Result<T, E> {
   Ok(T),
   Err(E),
}
```

```
1  use std::io::{stdin, Read};
2
3  fn main() {
4      let mut buffer: String = String::new();
5      // ^D to finish input
6      let status = stdin().read_to_string(&mut buffer);
7
8      match status {
9          Ok(_) => println!("{}", buffer),
10         Err(_) => println!("read error")
11     }
12 }
```

io::Result

- Turns out that I/O functions use an abbreviated version of Result, one which as the second type argument instantiated to io::Error
- You have to be aware of this when trying to return the status of an I/O operation from your own functions.

```
type Result<T> = std::result::Result<T, std::io::Error>;
```

Stdout

- Writing to stdout works analogously to reading from stdin
- Just be aware that the write_all functions expects bytes rather than a string.

```
1  use std::io::{stdout, Write};
2
3  fn main() {
4      let buffer: String = "Hello World!\n".to_string();
5      let status = stdout().write_all(buffer.as_bytes());
6
7      match status {
8          Ok(_) => (),
9          Err(_) => println!("write error")
10     }
11 }
```

Reading from a File

- Reading from a file follows the same patterns as reading from stdin
- The only difference is that we have to open the file first

Reading from a File

```
1  use std::fs::File;
2  use std::io::Read;
3  use std::result::Result;// get back the original definition
4
5  fn read_from_file(fname: String) -> Result<String, String> {
6      let mut f = match File::open(fname.clone()) {
7          Ok(file) => file,
8          Err(_) => return Err(format!("error opening file {}",fname.clone())),
9      };
10
11     let mut s = String::new();
12
13     match f.read_to_string(&mut s) {
14         Ok(_) => return Ok(s),
15         Err(_) => return Err(format!("error reading file {}",fname.clone())),
16     }
17 }
18
19 fn main() {
20     match read_from_file("mytext.txt".to_string()) {
21         Ok(s) => println!("{}",s),
22         Err(s) => panic!(s),
23     }
24 }
```

Writing to a File

```
1  use std::fs::File;
2  use std::io::Write;
3
4  fn main()  {
5      let write_buf = "Hello World!\n".to_string();
6
7      let mut f = match File::create("foo.txt") {
8          Ok(file) => file,
9          Err(e) => panic!("{:?}",e)
10     };
11     match f.write_all(write_buf.as_bytes()) {
12         Ok(_) => (),
13         Err(e) => panic!("{:?}",e)
14     }
15     match f.sync_all() {
16         Ok(_) => (),
17         Err(e) => panic!("{:?}",e)
18     }
19 }
```