Rust Basics

CS128H

Goals For Today

- Basic Data Types
- Mutability
- Control Flow
  - Conditionals and loops
- File input

What are Data Types?

- Data types are the method by which we tell the compiler what type of data it is storing
- Built in to the compiler so users can access with predetermined keywords
- There are primarily two types of data types
  - Scalar Data Types
  - Compound Data Types

Scalar Data Types
- Integers
- Floating Points
- Booleans
- Characters

Compound Data Types
- Tuples
- Arrays

Numbers in Rust

- Two ways of expressing numbers
  - Floating Point numbers (used for decimal representations)
  - Integer numbers. Integer will ALWAYS round down
  - Default integer size is 32

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Incorrect

```
fn main() {
    int x = 5;
    println!("{}", x);
}
```

Correct

```
fn main() {
    let x: i32 = 5;
    println!("{}", x);
}
```

```
fn main() {
    let x = 5;
    println!("{}", x);
}
```

Important things to know

- The ***let*** keyword lets the compiler know that a variable is being initialized. It is absolutely necessary
- The size specified in the declaration tells the compiler how much space it needs to allocate in memory
    - Can be used to make your code much more efficient. Not necessary for this class
    - The memory allocated is the length in the table. The range of an integer's value is from -2^(length-1) to 2^(length-1) - 1
    - Ex: a i8 has a range of -128 to 127
    - Unsigned ints cannot be negative so the range starts at 0

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Floating Point

- Floating Points are your decimals
- 2 types of floating points
  - f32: single decimal value precision. Default value
  - f64: double decimal precision.
- Floats work just like ints do
- Remember to declare your floats with ***let***

Mathematical Operations

- Rust supports mathematical operations
- Rust will automatically assign types to variables whose type is not specified
- If the operation was between two floats, the result will also be a float. If it was between two ints, the result will be an int
- Rust does not support operations between floats and integers. You must type cast
-

```
fn main() {
    // addition
    let sum = 5 + 10;

    // subtraction
    let difference = 95.5 - 4.3;

    // multiplication
    let product = 4 * 30;

    // division
    let quotient = 56.7 / 32.2;
    let floored = 2 / 3; // Results in 0

    // remainder
    let remainder = 43 % 5;
}
```

Characters and Strings

- Rust has inbuilt types for both characters and string, but string declaration and usage is slightly more complex than the topics of this video
- Chars or characters are the primary way to store single alphabetic pieces of data
- Chars are represented with the keyword char and are declared with a pair of single quotes
- Rust has a far more diverse and powerful character type than other languages
    - Chars in other languages are 1 byte, in Rust they are 4 bytes!
    - This allows chars to also store emojis or other special characters (like japanese letters)

```
fn main() {
    let c = 'z';
    let z: char = 'Z'; // with explicit type annotation
    let heart_eyed_cat = '😻';
}
```

Boolean

- T/F Data type
- Used in control flow to evaluate expressions
- Stores true and false values
- 1 byte size

```
fn main() {
    let t = true;

    let f: bool = false; // with explicit type annotation
}
```

Mutability

- Rust inherently tries to protect its data at compile time
- It wants to make sure that data that is in use cannot be unintentionally edited by someone by accident
- Thus it, by default, makes all of your declared variables **immutable**
    - This means that once you declare a variable, it is bound to its assigned value forever
- If you want your data to be changeable, you use the keyword ***mut*** when declaring your variable
- This is true for all variable types. If you want to edit the value, use ***mut***

```
let mut x = 5;
```

```
let mut x : f64 = 5.2;
println!("The value of x is: {x}");
x = 6.7;
println!("The value of x is: {x}");
```

Conditionals

- All conditionals are evaluated with boolean logic. You must use a boolean expression
- ***If, else if,*** and ***else***
- All contents of a condition must be within curly braces
- If conditions are fundamentally expressions meaning you can have some creative uses for them

```
fn main() {
    let number = 3;

    if number != 0 {
        println!("number was something other than zero");
    }
}
```

```
fn main() {
    let condition = true;
    let number = if condition { 5 } else { 6 };

    println!("The value of number is: {number}");
}
```

```
fn main() {
    let condition = true;

    let number = if condition { 5 } else { "six" };

    println!("The value of number is: {number}");
}
```

Loops

- 3 different ways to implement loops
  - Loop - used to repeat indefinitely until a break statement is reached
  - While - used to loop until a condition is met
  - For - used to loop through a set number of iterations
    - Enhanced for loops can iterate through elements

```
fn main() {
    let mut number = 3;

    while number != 0 {
        println!("{number}!");

        number -= 1;
    }

    println!("LIFTOFF!!!");
}
```

```
fn main() {
    for i in (1..4)
    {
        println!("{}",i);
    }
}
```

```
fn main() {
    let a = [10, 20, 30, 40, 50];

    for element in a {
        println!("the value is: {element}");
    }
}
```

Reminders:

Rust Extra Practice on PrairieLearn

Common Programming Concepts - The Rust Programming Language (rust-lang.org)

Homework 1 released next week

Add/Drop Deadline is tomorrow