Lifetime

20.2.22
김지현

Lifetime

모든 참조는 유효한 범위를 가지는 것을 **참조의 수명(lifetime)**이라고 함

수명은 기본적으로 추론에 의해 결정되지만

그렇게 결정될 수 없는 경우 명시 필요함

Lifetime annotation

-두 개의 문자열을 인자로 받아 길이가 더 긴 것을 반환하는 함수

-함수는 문자열의 소유권을 가져가지 않고 참조(&)함

```
fn main() {
    let string1 : String = String::from( s: "abcd");
    let string2 : &str = "xyz";

    println!("The longest string is {}", longest( x: string1.as_str(), y: string2));
}

fn longest(x: &str, y: &str) -> &str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

```
error[E0106]: missing lifetime specifier
 --> src\main.rs:8:33
  |
8 | fn longest(x: &str, y: &str) -> &str {
  |               ----     ----     ^ expected named lifetime parameter
  |
  = help: this function's return type contains a borrowed value, but the signature does not say whether it is borrowed from `x` or `y`
help: consider introducing a named lifetime parameter
  |
8 | fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
  |           ^^^^    ^^^^^^^     ^^^^^^^     ^^^
```

-x와 y중 무엇이 반환될지 알 수 없으므로 **제네릭 생명주기를 명시**해주어야 함

Lifetime annotation

-수정코드

```
fn main() {
    let string1 : String = String::from( s: "abcd");
    let string2 : &str = "xyz";

    println!("The longest string is {}", longest( x: string1.as_str(), y: string2));
}

fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

```
    Finished dev [unoptimized + debuginfo] target(s) in 0.12s
     Running `target\debug\webserver.exe`
The longest string is abcd
```

-x와 y가 lifetime annotation ‘a을 공유하여 ‘a 가 나타내는 수명은 **x와 y의 수명의 교집합**에 해당하는 범위

-longest함수의 반환값은 **x, y둘 중 하나라도 lifetime을 다하면 끝남**

Lifetime 생략 규칙

-참조를 사용하지만 lifetime **어노테이션을 생략할 수 있는 경우**도 있음

-Rust컴파일러가 유추할 수 있는 경우 생략 가능

규칙1. 각 참조 매개변수는 각자의 수명 매개변수를 가짐

fn foo(x: &i32)는 fn foo<'a>(x: &i32)와 같이 하나의 수명 매개변수를 가짐

fn foo(x: &i32, y:&i32)는 fn foo<'a,'b>(x: &'a i32, y:&'b i32)와 같이 두개의 수명 매개변수를 가짐

Lifetime 생략 규칙

-참조를 사용하지만 lifetime **어노테이션을 생략할 수 있는 경우**도 있음

-Rust컴파일러가 유추할 수 있는 경우 생략 가능

규칙2.입력 수명 매개변수가 하나 존재한다면 그것이 모든 출력 수명 매개변수에 적용됨

fn foo<'a>(x: &'a i32) -> &i32와 같이 하나의 수명 annotation이 명시되어있는 함수는

fn foo<'a>(x: &'a i32) -> &'a i32 와 같이 반환값의 수명에 적용됨

Lifetime 생략 규칙

-참조를 사용하지만 lifetime **어노테이션을 생략할 수 있는 경우**도 있음

-Rust컴파일러가 유추할 수 있는 경우 생략 가능

규칙3. 함수 인자 중에 &self 또는 &mut self가 있으면, 그 인자에 사용된 lifetime이 모든 출력 lifetime에 적용됨

```
fn my_foo(&self, x: &str) -> &Foo { &self.foo }
// becomes
  fn my_foo<'a, 'b>(&'a self, x: &'b str) -> &'a Foo { &self.foo }
```

```
fn my_bar(&mut self, x: &str) -> (&mut Foo, &Bar) { (&mut self.foo, &self.bar) }
// becomes
  fn my_bar<'a, 'b>(&'a mut self, x: &'b str) -> (&'a mut Foo, &'a Bar) { (&mut self.foo, &self.bar) }
```

static lifetime

-모든 문자열 리터럴은 **정적 수명**을 가짐

-’static이라는 수명 어노테이션으로 명시 할 수 있음

-정적 수명을 가진 변수는 **프로그램 전 범위에서 사용가능**함