Chapter 11 for문

날짜	@November 10, 2021
발표자	장예서

11.1 for문 동작 원리

Go 언어는 반복문으로 for문 하나만 지원하지만, 여러 형태가 있기 때문에 각 형태를 적재적소에 잘 사용해야함.

```
for 초기문; 조건문; 후처리 {
  코드 블록  //조건문이 true인 경우 수행됩니다.
}
```

for문이 실행될 때 초기문 먼저 실행 → 조건문 검사 → 조건문 결과가 true이면 for문 {} 안쪽 코드 블록 수행 → 후처리 구문 → 다시 조건문 검사

만약 조건문이 false이면 후처리 없이 for문 종료

```
package main

import "fmt"

func main() {
  for i := 0; i < 10; i++ { // 초기문; 조건문; 후처리
    fmt.Print(i, ", ") //i 값 출력
  }

  //  fmt.Println(i)              // Error - i는 이미 사라졌다.
}
```

출력문

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

- i 변수 선언하고 0으로 초기화, i를 10과 비교, 조건문 true → {} 코드 블록 실행 → 후처리 i++ 실행
- 반복 후 i가 10이 되면 조건문 10<10은 false가 되어 for문 종료
- for문 안에서 선언된 i가 for문이 종료되면서 메모리에서 제거되기 때문에 for문 밖에서 i를 호출하면 에러 발생

11.1.1 초기문 생략

```
for ; 조건문; 후처리 {
  코드 블록
}
```

11.1.2 후처리 생략

```
for 초기문; 조건문; {
  코드 블록
}
```

11.1.3 조건문만 있는 경우

```
for ; 조건문; {
  코드 블록
}
```

혹은

```
for 조건문 {
  코드 블록
}
```

11.1.4 무한 루프

- 조건문이 true이면 코드 블록이 무한 반복되는 무한 루프가 됨.

```
for true {
 코드 블록
}
```

switch문에서 조건문을 생략하면 true가 되듯이 for문에서도 true 생략 가능

```
for {
 코드 블록
}
```

- 무한 루프는 프로그램이 강제 종료되거나 break를 사용해 for문을 종료하지 않으면 계속 반복되니 사용에 주의를 기울여야 함.

```
package main

import (
  "fmt"
  "time"
)

func main() {
  i := 1
  for {
    time.Sleep(time.Second)
    fmt.Println(i)
    i++
  }
}
```

출력문

```
1
2
3
4
5
...
```

- 위 예제는 프로그램을 강제 종료하지 않으면 계속 숫자 출력. → Ctrl + C를 눌러 강제 종료하기!

11.2 continue와 break

continue와 break는 반복문을 제어하는 키워드.

continue: 이후 코드 블록을 수행하지 않고 곧바로 후처리 → 조건문 검사 → ...

break: for문에서 곧바로 빠져나옴

```
package main

import (
  "bufio"
  "fmt"
  "os"
)

func main() {
  stdin := bufio.NewReader(os.Stdin)
  for { // 무한 루프
    fmt.Println("입력하세요")
    var number int
    _, err := fmt.Scanln(&number) // 한줄 입력 받기
    if err != nil {               // 숫자가 아닌 경우
      fmt.Println("숫자를 입력하세요")

      // 키보드 버퍼를 비웁니다.
      stdin.ReadString('\n') // 키보드 버퍼 지우기
      continue
    }
    fmt.Printf("입력하신 숫자는 %d입니다\n", number)
    if number%2 == 0 { // 짝수 검사
      break // for문 종료
    }
  }
  fmt.Println("for문이 종료되었습니다.")
}
```

출력문

```
입력하세요
ff
숫자를 입력하세요
입력하세요
31
입력하신 숫자는 31입니다
입력하세요
32
입력하신 숫자는 32입니다
for문이 종료되었습니다.
```

- 입력받은 수가 짝수일 때까지 계속 입력을 받아서 출력하는 예제

- 만약 입력이 숫자가 아닌 경우에는 err에 에러값이 들어감 (err ≠ nil이라면 에러가 발생했다는 뜻)
- 줄바꿈까지 문자열을 다시 읽어서 키보드 버퍼를 없애줘야함
- continue를 통해서 이후 코드 블록을 건너뜀. 무한 루프라서 후처리와 조건검사가 없기 때문에 바로 다음 반복 시작

11.3 중첩 for문

```
package main

import "fmt"

func main() {
  for i := 0; i < 3; i++ { // 3번 반복
    for j := 0; j < 5; j++ { //5번 반복
      fmt.Print("*") // * 출력
    }
    fmt.Println()
  }
}
```

출력문

```
*****
*****
*****
```

- i가 3보다 작으면 반복, j가 5보다 작으면 반복
- j를 0으로 초기화하므로 총 5회 반복 → *를 5번 출력
- 안쪽 반복문 종료 후 줄바꿈
- 이중 중첩된 for문에서 각 for문이 i번, j번 반복하면 i x j번 반복하게 됨. 만약 삼중 중첩 for문을 사용하면? 총 i x j x k번 반복. 중첩 반복문을 사용하면 연산량을 크게 증가시키므로 반복 횟수가 많을 때는 사용에 특히 주의해야 함.

```
package main

import "fmt"
```

```
func main() {
  for i := 0; i < 5; i++ { //5번 반복
    for j := 0; j < i+1; j++ { //현재 i값+1만큼 반복
      fmt.Print("*") // * 출력
    }
    fmt.Println()
  }
}
```

출력문

```
*
**
***
****
*****
```

- 바깥 for문: i < 5가 참인 동안 반복
- 안쪽 for문: j < i + 1이 참인 동안 반복
- i값이 0일 때 안쪽 for문의 조건문은 j < 0 + 1 → 1번 실행
- j가 1, 2, 3, 4, 5보다 작을 때 참 → *를 1, 2, 3, 4, 5개 출력

```
package main

import "fmt"

func main() {
  dan := 2
  b := 1
  for {
    for {
      fmt.Printf("%d * %d = %d\n", dan, b, dan*b)
      b++
      if b == 10 { //
        break // 안쪽 for문 종료
      }
    }
    b = 1
    dan++
    if dan == 10 {
      break // 바깥쪽 for문 종료
    }
  }
  fmt.Println("for문이 종료되었습니다.")
}
```

출력문

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
3 * 1 = 3
... 중략 ...
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
for문이 종료되었습니다.
```

- b값이 10이 되면 break → 안쪽 for문 종료
- dan값이 10이 되면 break → 바깥쪽 for문 종료

11.4 중첩 for문과 break, 레이블

모든 for문을 빠져나가고 싶을 때? 첫번째 방법은 불리언 변수 사용.

```
package main

import "fmt"

func main() {
  a := 1
  b := 1

  found := false
  for ; a <= 9; a++ {
    for b = 1; b <= 9; b++ {
      if a*b == 45 {
        found = true // 찾았음을 표시하고 break
        break
```

```
      }
    }
    if found { // 바깥쪽 for문에서 찾았는지 검사해서 break
      break
    }
  }
  fmt.Printf("%d * %d = %d\n", a, b, a*b)
}
```

출력문

```
5 * 9 = 45
```

- 1~9 사이의 두 수를 곱했을 때 45가 되는 수를 찾음
- 안쪽 for문에서 두 수의 곱이 45가 되는 경우 found 변수 → true 후 break
- 바깥쪽 for문은 found가 true인지 검사

플래그 변수를 사용하는 게 번거롭다면? 레이블을 이용한 방법. for문을 시작할 때 레이블을 정의하고 break할 때 앞서 정의한 레이블을 적어주면 그 레이블에서 가장 먼저 속한 for문까지 모두 종료하게 됨.

```
package main

import "fmt"

func main() {
  a := 1
  b := 1

OuterFor:               // 레이블 정의
  for ; a <= 9; a++ {
    for b = 1; b <= 9; b++ {
      if a*b == 45 {
        break OuterFor  // 레이블에 가장 먼저 포함된 for문까지 종료
      }
    }
  }
  fmt.Printf("%d * %d = %d\n", a, b, a*b)
}
```

출력문

```
5 * 9 = 45
```

- 레이블은 레이블 이름을 적고 콜론 :을 적어서 정의
- break할 때 레이블 이름을 적으면 그 레이블에서 가장 먼저 포함된 for문까지 종료

레이블이 편리할 수 있으나 혼동을 불러일으킬 수 있고 자칫 잘못 사용하면 예기치 못한 버그 발생 가능. 그래서 **되도록 플래그를 사용하고 레이블은 꼭 필요한 경우에만 사용하기를 권장함.**

클린 코드를 지향하려면 중첩된 내부 로직을 함수로 묶어 중첩을 줄이고, 플래그 변수나 레이블 사용을 최소화해야함.

```
package main

import "fmt"

func find45(a int) (int, bool) {
  for b := 1; b <= 9; b++ {
    if a*b == 45 {
      return b, true
    }
  }
  return 0, false
}

func main() {
  a := 1
  b := 0

  for ; a <= 9; a++ {
    var found bool
    if b, found = find45(a); found { // 함수 호출
      break
    }
  }
  fmt.Printf("%d * %d = %d\n", a, b, a*b)
}
```

출력문

```
5 * 9 = 45
```

- 중첩된 for문 내부 코드를 find45() 함수로 묶음. 두번째 반환값으로 a와 곱해서 45가 되는 값을 찾았는지 여부 반환
- 중첩하지 않고 함수 호출 → 플래그 변수나 레이블이 필요없음. 코드 읽기에도 편리

연습문제

1.

```
package main

import "fmt"

func main() {
  for i := 10; i > 0; i-- {
    fmt.Print(i, " ")
  }
}
```

2.

```
package main

import "fmt"

func main() {
  for i := 2; i < 10; i++ {
    if i >= 3 && i <= 6 {
      continue
    }
    for j := 1; j < 10; j++ {
      fmt.Println(i, "*", j, "=", i*j)
    }
    fmt.Println()
  }
}
```

3.

```
package main

import "fmt"
```

```
func main() {
  for i := 1; i < 10; i += 2 {
    fmt.Println(i, "*", i, "=", i*i)
  }
}
```

4.

```
package main

import "fmt"

func main() {
  for i := 0; i < 5; i++ {
    for j := 0; j < 5-i; j++ {
      fmt.Print("*")
    }
    fmt.Println()
  }
}
```