УЧЕБНИК ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ ГО

ЕФИМОВ А. И.

РИГА 2025

Оглавление

ПРЕДИСЛОВИЕ

Начались внезапные занятия по Го. Вашему вниманию предоставляются материалы этих занятий с пояснениями.

ОПЕРАТОР УСЛОВНОГО ПЕРЕХОДА

Занятие посвящено оператору ЕСЛИ. Начнём с обширных примеров.

```go
package main

import (
  "fmt"
)

func example1() {
  a := 4
  b := 6

  x := a * a + 2 * b
  if x := a - b; x > 10 {
    a := 234
    fmt.Println("x is too big, x =", x, ", a =", a)
  } else {
    fmt.Println("x is OK, x =", x)
  }

  fmt.Println("Now x =", x, ", a =", a)
}

func example2() {
  a := 4
  b := 6

  if x := a * a + 2 * b; x > 10 {
    fmt.Println("x is too big, x =", x)
  } else {
    fmt.Println("x is OK, x =", x)
  }

  x := 123
  _ = x

  //fmt.Println("Now x =", x) // error: x undefined
}

func main() {
  example1()
  example2()
```

```
}
```

Пример «Шахматы»

```
package main

import (
  "fmt"
)

func main() {
  var x1, y1, x2, y2 int

  fmt.Println("Введите позицию первой фигуры: ")
  fmt.Scan(&x1, &y1)
  fmt.Println("Введите позицию второй фигуры: ")
  fmt.Scan(&x2, &y2)

  dx := x1 - x2
  if dx < 0 {
    dx = -dx
  }

  dy := y1 - y2
  if dy < 0 {
    dy = -dy
  }

  if dx == 0 && dy == 0 {
    fmt.Println("Одинаковые!")
  } else {
    if dx == 0 || dy == 0 {
      fmt.Println("Ладья бьёт")
    }
    if dx <= 1 && dy <= 1 {
      fmt.Println("Король бьёт")
    }
    if dx == dy  {
      fmt.Println("Слон бьёт")
    }
    if dx == 0 || dy == 0 || dx == dy {
      fmt.Println("Ферзь бьёт")
    }
    if dx == 1 && dy == 2 || dx == 2 && dy == 1 {
      //Вариант 2: if (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 -
      ↪  y2) == 5 {
      //Вариант 3: if dx * dx + dy * dy == 5 {
      fmt.Println("Конь бьёт")
    }
  }

}
```

```go
package main

import (
  "fmt"
)

func main() {
  fmt.Println("WHILE:")
  i := 0
  for i < 10 {
    fmt.Print(i)
    i++
  }
  fmt.Println(" Значение i после цикла =", i)
  fmt.Println()

  fmt.Println("FOR с пустым последействием:")
  for i := 0; i < 5; {
    fmt.Print(i)
    i++
  }
  fmt.Println(" Значение i после цикла =", i)
  fmt.Println()

  fmt.Println("FOR:")
  for i := 0; i < 5; i++ {
    fmt.Print(i)
  }
  fmt.Println(" Значение i после цикла =", i)
  fmt.Println()

  fmt.Println("LOOP:")
  i = 0
  for {
    if (i == 7) { break }
    fmt.Print(i)
    i++
  }
  fmt.Println(" Значение i после цикла =", i)
  fmt.Println()

  /*      i := 0;
          REPEAT
            Out.Int(i, 0);
            INC(i)
          UNTIL i = 8 */

  fmt.Println("REPEAT:")
  i = 0
  for {
    fmt.Print(i)
    i++
    if (i == 8) { break }
```

```
  }
  fmt.Println(" Значение i после цикла =", i)
  fmt.Println()

  fmt.Println("FOR INTEGER RANGE:")
  //for j := 0; j < 10; j++ {
  for j := range i {
    fmt.Print(j)
  }
  fmt.Println()

  fmt.Println("Slice:")
  // s is a slice
  s := []int{22, 0, 77, 44, 19, 123, 6, 11, 23}
  for i, x := range s {
    fmt.Println(i, x)
  }

}
```

Примеры на серзы (начало)

```
package main

import (
  "fmt"
)

func main() {
  // m is an array of 9 integers
  m := [...]int{22, 0, 77, 44, 19, 123, 6, 11, 23}

  // x is a slice that points to m[2], length=6-2=4
  x := m[2:6]

  fmt.Println("m:", m, len(m))
  fmt.Println("x:", x, len(x), cap(x))

  // If there is enough space in the underlying array,
  // then items are replaced in the array. Otherwise,
  // a new larger (unnamed) array is created.
  x = append(x, -47, -48) // There is space for 2 items
  x = append(x, -49, -50) // And now - not enough space

  fmt.Println("After x = append(x, ....):")
  fmt.Println("x:", x, len(x), cap(x))
  fmt.Println("m:", m, len(m))

  // Point to x[1]. x[0] is unreachable forever
  x = x[1:cap(x)]
  fmt.Println("массив под слайсом:", x[0:cap(x)])
}
```

ПЕРЕМЕННЫЕ

В языке Го переменные объявляются вот так:

```
var x int
```

или вот так:

```
x := 5
```

То есть оператор «=» — это присваивание, а оператор «:=» — это краткое объявление переменной с одновременным присваиванием. Во втором случае тип выводится из значения.

Можно так:

```
var x, y, z int
a, b, c := 1, 2, 3
```

Значения по умолчанию

Объявленные через var переменные (даже локальные) получают нулевое значение соответствующего типа (это только в Го так среди компилируемых языков).

Локальные и глобальные переменные

Переменные можно объявлять в функциях или перед функциями (тогда они глобальные).

Ещё есть такой вариант (чаще используется для объявления глобальных переменных):

```
var (
  y int
  z int = 7
  x = 51
  w = int(3)
)
```

Глобальные переменные объявляем крайне редко.

Кортежное присваивание

(не картёжное, а от слова кортеж — touple)

```
x := 5
y := 4
a, b := 2, 3
x = x + 1
x, y = a, b // то же, что и x = a; y = b
x, y = y, x // поменять местами
```

Последнее присваивание — не то же самое, что x = y; y = x (тут бы потерялось изначальное значение x)

Оператор условного перехода

(также известный как ИФ)

Ничего необычного. Но нет круглых скобок, а фигурные обязательны во всех случаях. «{» в Го вместо THEN в Обероне, а «}» вместо END — без них никак.

Нельзя заканчивать строку сразу после слова else — лексический анализатор Го поставит «;» в конец такой строки, и ничего не сработает. Поэтому формат только такой:

```
if a < b {
  x = 5
} else if a > b {
  x = 6
} else {
  x = 7
}
```

Ну или в строчку

```
if a < b { x = 6 } else { x = 7 }
```

Такая запись считается допустимой, но IDE'шки, бывает, всё равно такое разворачивают.

Инициализация в ифе

В Го можно непосредственно перед условием ифа (или элс-ифа) выполнять присваивание или даже объявление переменных.

```go
if x := a * b + c; x < 10 {
  //...
} else if x > 10 {
  //...
}
```

При этом объявленная переменная существует от места её объявления до конца всего ифа. Её можно использовать в других ветвях (но не выше места объявления) и в других условиях.

Область видимости

(по-английски — scope)

Переменные в Го видны в рамках того блока, где были объявлены, начиная от места объявления и до конца блока — до соответствующей этому блоку «}».

То есть переменная, объявленная в ифе (не перед условием), будет длить своё жалкое существование только в этой ветви.

```go
a, y := 1, 0
if a == 1 {
  x := 7
  y = x + a
}
fmt.Println(x) // Ошибка: x не определена
```

Такое Го не даст скомпилировать.

Но если переменная с тем же именем была объявлена раньше, то она вернётся вместе со своим значением.

```go
a, x, y := 1, 25, 0
if a == 1 {
  x := 7
  y = x + a
}
fmt.Println(x) // Выведет 25
```

В общем, переменные «затеняются».

В Го все переменные обязательно использовать. Неиспользованная переменная вызывает ошибку компиляции.

```go
if a == 1 {
  x := 5 // Ошибка: переменная не используется
}
```

Можно хотя бы «положить её в мусорник». Это не производственный код, а временное дело.

```
if a == 1 {
  x := 5
  _ = x
}
```

Автоопределение типа

При объявлении переменных без указания типа

```
a := 5 // тип int
b := 5.0 // тип float64
c := "ы" // тип string
d := true // тип bool
```

Более сложный пример: объявляем массив сразу со значениями. Тут тип в общем-то не указан, а указано значение, но зато значение синтаксически включает в себя описание тип. Так тоже работает.

```
x := [3]int{26, 92, 70}
```

Поэтому, если нужна переменная float32, а не float64, то не обязательно писать так:

```
var a float32 = 4.0
```

можно и так:

```
a := float32(4.0)
```

Технически, это не указание типа, а приведение типа. Но так тоже работает.

Кроме типа int есть ещё int8, int16, int32, int64, а также uint8, uint16, uint32, uint64.

Тип int фактически int64 (но теоретически может быть и int32 на каких-то системах). Логически он отделён от их обоих, для присваивания между int и int32/int64 надо приводить типы. Зато нет таких проблем при кроссплатформенном программировании, как в языке Си.

Тип byte — синоним uint8. Тип rune — синоним int32. Это в Го вместо CHAR.

int с приставкой u означает беззнаковое целое (unsigned integer). Диапазон значений от 0 до $2^(n) - 1, n$ — $(32 int 32)$.

Без приставки u целые числа имеют сдвинутый диапазон, чтобы поместились отрицательные числа: от $-2^(n-1) 2^(n-1) - 1$.

Пример:

```
uint8 = 0..255
int8 = -128..127
```

$(2^8 = 256, 2^7 = 128)$

ЗАДАНИЕ НА ПРАКТИКУ

Шахматы

Вводятся 4 натуральных числа — координаты двух клеток на шахматной доске 8 на 8 (вариант: миллион на миллион). Программа определяет, соединены ли эти клетки ходом: ладьи, короля, слона, ферзя, коня.

Например:

```
Введите координаты первой клетки (x, y): 1 4
Введите координаты второй клетки (x, y): 3 6
Ладья  не бьёт
Король не бьёт
Слон   бьёт
Ферзь  бьёт
Конь   не бьёт
```

Подсказка: все шахматные фигуры во все стороны ходят одинаково. Кроме того, не важно, где именно на доске находятся обе клетки, а важно их взаимное расположение относительно друг друга.

Подсказка 2: Одну из клеток можно мысленно перенести в начало координат (разумеется, параллельно перенеся и вторую клетку). Тогда координаты первой клетки будут (0; 0) и потеряют смысл. Координаты второй клетки можно привести к неотрицательным (взять модуль). От этого ответ не изменится. В итоге получаем два неотрицательных числа, дальше уже просто.

В Го есть, якобы, только один цикл — for. Но на самом деле это 5 разных циклов: while, for (в сишном стиле), loop (бесконечный цикл), repeat until, и foreach (здесь он for range). Теперь ещё есть и for range над целочисленными значениями, что напоминает уже паскалевский цикл фор.

В циклах часто объявляют переменные. Их существование также ограничено всем оператором цикла.

Наиболее полная форма:

```
for i := 0; i < 10; i++ {
  //...
}
```

Здесь: i := 0 — инициализация i < 10 — условие продолжения i++ — последействие

i := 0 — сработает один раз перед началом цикла. Похоже на такую же штуку в ифе.

i < 10 — очередной шаг цикла (включая и самый первый) будет выполнен только в том случае, если перед его началом это условие истинно.

i++ — будет происходить всякий раз в конце каждого шага цикла.

Можно и не объявлять переменную в цикле, а сделать это заранее.

```
i := 0;
for ; i < 10; i++ {
  //...
}
fmt.Println(i) // Выведет 10
```

Тогда переменная будет существовать (и сохранит своё приобретённое в цикле новое значение) и после цикла.

Форма типа WHILE:

```
i := 0
for i < 10 {
  //...
  i++
}
```

В этой форме вообще нет точек с запятыми. Это просто обычный цикл WHILE.

Если точки с запятыми есть, их должно быть две:

```
i := 0
// это глупость, но так можно
for ; i < 10; {
  //...
  i++
}
```

«Операторы инициализации и последействия можно опустить. В этом случае не нужны и точки с запятыми: сишная конструкция while в Го пишется вот как for» (Тур по Го).

```
// s is a slice
s := []int{22, 0, 77, 44, 19, 123, 6, 11, 23}
for i, x := range s {
  fmt.Println(i, x)
}
\begin{gocode}2
i будет принимать значения 0, 1, 2, 3...

x будет принимать значения 22, 0, 77, 44...

Если что-то из этого не нужно, надо положить его в мусорник:

\begin{gocode}
for _, x := range s {
  fmt.Println(x)
}
```

Если нужны только индексы (i), можно так:

```
for i := range s {
  fmt.Println(i)
}
```

ЗАДАЧИ НА ПРАКТИКУ (ДОМАШНЕЕ ЗАДАНИЕ)

Задачи на линейные программы

1) Вычислить квадрат, куб, четвертую и пятую степень введённого числа. Примечание: Сэкономить — не вычислять каждый раз всё заново. В этой задаче не предполагается использование if и for.

Задачи на оператор ЕСЛИ

2) Влазит ли кирпич в коробку. Вводятся шесть чисел: размеры кирпича и размеры коробки. Считать, что в коробку того же размера кирпич со скрипом, но помещается.

3) Високосный ли год?

Подсказка:

Формулировка критерия високосности следующая: Число должно нацело делиться на 4, но если оно делится на 100, то оно должно также делиться и на 400.

Или, если перефразировать, получается следующее:

Чтобы считать год високосным, должны выполняться следующие условия:

1. Число должно делиться нацело на 4. Если оно не делится на 4, то год не високосный — и на этом всё, остальные проверки не требуются.

2. Дальше смотрим, делится ли число на 100. Если не делится, к нему больше вопросов нет. Получается, оно делится на 4, но не на 100 — замечательно. Год високосный. Вопрос закрыт.

3. Однако если число-таки делится на 100, то к нему появляется следующее требование: оно должно делиться на 400.

Вот попробуйте это всё объединить в одно составное условие с операциями И (&&) и ИЛИ (||).

Задачи на циклы

4) Угадайка на диапазоне [1..100]. Человек загадывает число. Программа отгадывает его максимум за 7 шагов половинного деления.

Примечание: Машина угадывает число, задавая серию вопросов вида «Ваше число больше 50?». Человек отвечает да или нет. (В данном примере, если загадано число 50, ответ будет «нет»).

Подход к решению: Две переменные L и R задают начало и конец промежутка, в котором идёт поиск правильного ответа. Рекомендуется задавать L не включительно, R — включительно. Промежуток 1..100 будет поэтому выражен как L = 0, R = 100. Каждый шаг цикла либо L сдвигается вправо, либо R сдвигается влево. Цикл заканчивается, когда в промежутке остаётся ровно одно число — оно и есть ответ.

5) Вывести все простые числа, не превосходящие N = 1000.

6) Найти сумму цифр натурального числа.

7) Вводятся числа, причём каждое следующее не меньше предыдущего. Как только введённое число окажется меньше предыдущего, ввод заканчивается (последнее введённое число не считаем членом введённой последовательности). Программа выводит: а) сколько всего чисел введено, б) сколько различных чисел введено, в) каково максимальное количество повторов одного числа.