

Программирование на языке Go

Лекция 6

Go

Структуры

Постановка задачи

- Хранить в программе описание характеристик некоторого объекта

Решение I

```
var aliceBirthYear int
var aliceBirthMonth int
var aliceBirthDay int
var aliceHeight float64
var aliceWeight float64
```

```
var bobBirthYear int
var bobBirthMonth int
var bobBirthDay int
var double bobHeight float64
var double bobWeight float64
```

Решение I - Проблемы

- Для каждого человека нужно создавать по пять отдельных переменных – **долго, могут быть опечатки**
- Чтобы передать в функцию, нужно перечислить все аргументы – **можно перепутать порядок**

```
print(aliceBirthYear, aliceBirthMonth,  
      aliceBirthDay, aliceHeight, aliceWeight);
```

- Как вернуть из функции?

Решение II - Структуры

```
type human struct{    // Свой тип данных
    aliceBirthYear int
    aliceBirthMonth int
    aliceBirthDay int
    aliceHeight float64
    aliceWeight float64
}

var alice, bob human // Создаём переменные
```

Решение II - Структуры

```
var alice, bob struct {  
    aliceBirthYear  int  
    aliceBirthMonth int  
    aliceBirthDay   int  
    aliceHeight     float64  
    aliceWeight     float64  
}
```

Решение II - Структуры

```
alice := struct {  
    aliceBirthYear  int  
    aliceBirthMonth int  
    aliceBirthDay   int  
    aliceHeight     float64  
    aliceWeight     float64  
}{  
    // Тут обязательно значения  
}
```


Где можно объявлять структуры?

- Внутри функций

```
func foo() {  
    type num struct {  
        i int  
    }  
}
```
- Вне функций

```
type num struct {  
    i int  
}  
func foo() {  
}
```
- Внутри других структур

```
type num struct {  
    j struct {  
        val int  
    }  
    i int  
}
```

Что может быть членом структуры?

Если можно создать переменную этого типа, то это может быть членом структуры

Например:

- Примитивные типы: `int`, `float64`, `byte` ...
- Другие структуры;
- Массивы;
- Строки;
- ...

Имена полей

```
type DataOut struct {  
    Year  int  
    Month int  
    Day   int  
}
```

```
type DataIn struct {  
    year  int  
    month int  
    day   int  
}
```

Поля структуры `DataIn` видны только в этом пакете, т.е. их можно назвать приватными полями.
Поля структуры `DataOut` видны за пределами пакета, т.е. их можно назвать публичными полями.

Как работать со структурой

```
type Data struct {  
    Year  int  
    Month int  
    Day   int  
}
```

```
var now Data  
now.Year = 2018  
now.Day  = 9  
now.Month = 11
```

Как работать со структурой (указатель)

```
type Data struct {  
    Year  int  
    Month int  
    Day   int  
}
```

```
var now *Data = new(Data)  
now.Year = 2018  
now.Day = 9  
now.Month = 11
```

Как работать со структурой (указатель II)

```
type Data struct {  
    Year  int  
    Month int  
    Day   int  
}
```

```
var now *Data = &Data{}  
now.Year = 2018  
now.Day = 9  
now.Month = 11
```

Как работать со структурой

```
now.Year = now.Year + 1 // 2019
```

```
fmt.Print(now.Day) // 9
```

```
now.Month = now.Day + now.Year // 2028
```

```
var p *int = &now.Month
```

Инициализация структуры

```
type Employee struct {  
    id    int  
    age   int  
    wage  float64  
}
```

// Значения по умолчанию 0. Поменять нельзя

```
var Kate Employee  
var frank = Employee{}  
var joe = Employee{1, 32, 60000.0} // Нужны все  
var john = Employee{id: 2, wage: 60000.0}
```


Инициализация слайса структур

```
type location struct {  
    name string  
    lat  float64  
    long float64  
}
```

```
locations := []location{  
    {name: "Bradbury Landing", lat: -4.5895, long: 137.4417},  
    {name: "Columbia", lat: -14.5684, long: 175.472636},  
    {name: "Challenger", lat: -1.9462, long: 354.4734},  
}
```

Присваивание значений структурам I

```
type Employee struct {  
    id int  
    age int  
    wage float64  
}
```

```
var joe Employee  
joe.id = 1  
joe.age = 32  
joe.wage = 60000.0
```

Присваивание значений структурам II

```
type Employee struct {  
    id    int  
    age   int  
    wage  float64  
}
```

```
var joe, mike = Employee{1, 20, 3.0}, Employee{}  
mike = joe // Копирование значений joe в mike  
joe = Employee{2, 22, 6.3}
```

Передача структуры как параметр в функцию

```
type Employee struct {  
    id    int  
    age   int  
    wage  float64  
}
```

```
func printInformation(employee Employee) {  
    fmt.Println("ID: ", employee.id)  
    fmt.Println("Age: ", employee.age)  
    fmt.Println("Wage: ", employee.wage)  
}
```

Передача структуры как параметр в функцию (указатель)

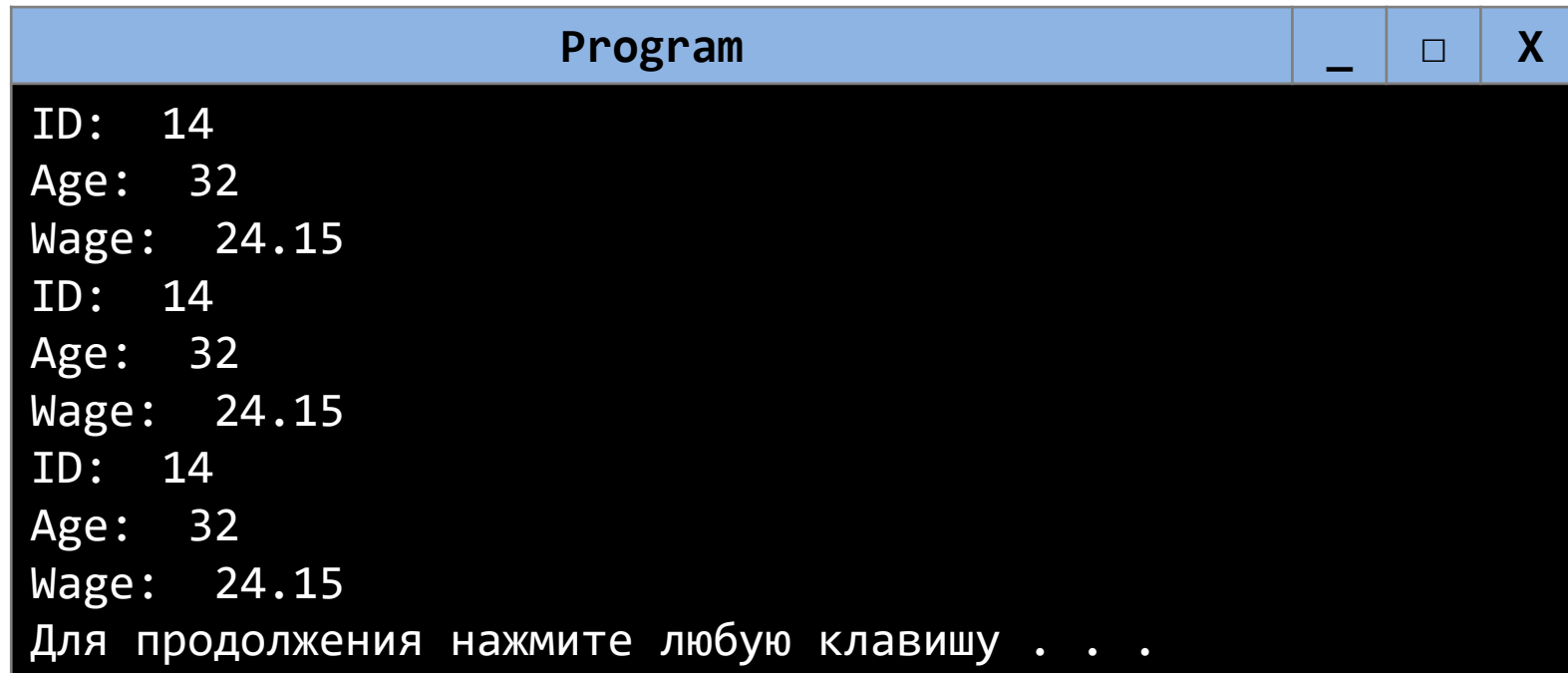
```
type Employee struct {  
    id    int  
    age   int  
    wage  float64  
}
```

```
func printInformationPtr(employee *Employee) {  
    fmt.Println("ID: ", employee.id)  
    fmt.Println("Age: ", employee.age)  
    fmt.Println("Wage: ", employee.wage)  
}
```

Передача структуры как параметр в функцию

```
func main() {  
    var john = Employee{14, 32, 24.15}  
  
    printInformation(john)  
    printInformation(Employee{14, 32, 24.15})  
    printInformationPtr(&john)  
}
```

Передача структуры как параметр в функцию



```
Program
ID:  14
Age:  32
Wage: 24.15
ID:  14
Age:  32
Wage: 24.15
ID:  14
Age:  32
Wage: 24.15
Для продолжения нажмите любую клавишу . . .
```

Возврат структур из функций

```
type Point3d struct {  
    x, y, z float64  
}  
  
func getZeroPoint() Point3d {  
    return Point3d{0.0, 0.0, 0.0}  
}  
  
func main() {  
    zero := getZeroPoint()  
    fmt.Print(zero)  
}
```


Метод String

```
type Point3d struct {  
    x, y, z float64  
}  
  
func (b Point3d) String() string {  
    return "Point3d"  
}  
  
func main() {  
    var zero Point3d  
    fmt.Print(zero)  
}
```

Дополнительные сведения

Разные типы

```
type Point3d struct {  
    x, y, z float64  
}
```

```
type Vector3d struct {  
    x, y, z float64  
}
```

```
p := Point3d{ 0.0, 0.0, 0.0 }
```

```
var v Vector3d
```

```
v = p; // Ошибка. У v и p разные типы, но можно преобразовать
```

Вложенные структуры

```
type Employee struct {  
    id    int16  
    age   int32  
    wage  float64  
}  
  
type Company struct {  
    CEO           Employee // CEO – это структура  
    numberOfEmployees int  
}  
  
var myCompany = Company{Employee{1, 42, 60000.0}, 5}  
fmt.Print(myCompany.CEO.id)
```

Размер структуры и выравнивание I

```
type Point3d struct {  
    x int16    // Sizeof: 2  
    y int32    // Sizeof: 4  
    z float64  // Sizeof: 8  
}
```

```
var zero Point3d  
fmt.Print(unsafe.Sizeof(zero)) // 16  != ( 2 + 4 + 8 )
```

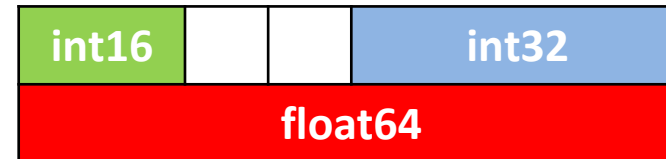
Размер структуры и выравнивание II

```
type Point3d struct {  
    x int16    // Sizeof: 2  
    z float64  // Sizeof: 8  
    y int32    // Sizeof: 4  
}
```

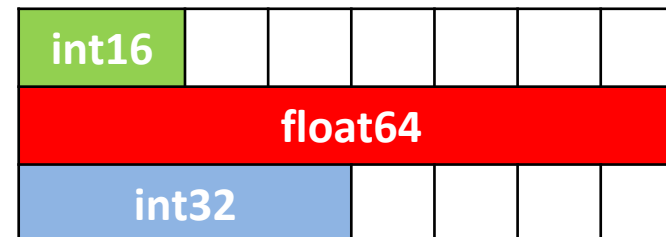
```
var zero Point3d  
fmt.Print(unsafe.Sizeof(zero)) // 24  != ( 2 + 4 + 8 )
```

Размер структуры и выравнивание II

```
type Point3d struct {  
    x int16  
    y int32  
    z float64  
}
```



```
type Point3d struct {  
    x int16  
    z float64  
    y int32  
}
```



Объединения

Что такое объединение?

В Go нет типа данных "объединение" или аналога

Перечисления

Что такое перечисление

Перечисление – это пользовательский тип данных, определяющий набор целочисленных констант.

Зачем нужен:

- Сделать код более читабельным путём замены «магических чисел» на элементы перечисления;
Пример: `return 0;` `return SUCCESS;`
- Как дополнительный контроль, защищающий от случайных, автоматических преобразований типов.

Что такое перечисление

В Go нет типа "перечисление", но можно создать аналог

Объявление I

```
type Weekday int
```

```
const (  
    Sunday      Weekday = iota + 1 // 1  
    Monday           // 2  
    Tuesday          // 3  
    Wednesday        // 4  
    Thursday         // 5  
    Friday           // 6  
    Saturday         // 7  
)
```

Объявление II

```
type Weekday int
```

```
const (  
    Sunday      Weekday = iota + 1 // 1  
    Monday           // 2  
    Tuesday         // 3  
    Wednesday Weekday = 1          // 1  
    Thursday        // 1  
    Friday          // 1  
    Saturday        // 1  
)
```

Переменные

```
type Direction int
```

```
const (  
    North Direction = iota // 0  
    East             // 1  
    South           // 2  
    West            // 3  
)
```

```
var a Direction
```

```
var b Direction = 0
```

```
var c Direction = North
```

Ввод / Вывод

```
type Direction int
```

```
const (  
    North Direction = iota // 0  
    East             // 1  
    South            // 2  
    West             // 3  
)
```

```
var a Direction = North  
fmt.Println(a) // 0  
fmt.Scan(&a)   // ВВОДИТЬ нужно число
```


Ввод / Вывод

```
type Direction int

const (
    North Direction = iota // 0
    East                  // 1
    South                 // 2
    West                  // 3
)

func (d Direction) String() string {
    return [...]string{"North", "East", "South", "West"}[d]
}

func (d Direction) EnumIndex() int {
    return int(d)
}

func main() {
    var d Direction = West
    fmt.Println(d)           // West
    fmt.Println(d.String())  // West
    fmt.Println(d.EnumIndex()) // 4
}
```

Операции

Для перечислений работают все операции базового типа, но при этом тип результата остаётся новый:

```
var d Direction = West
var i int = d + 1          // Ошибка. Результат выражения НЕ int

var d Direction = West
var i Direction = d + 1    // Ошибки нет
```

Переменной перечисляемого типа можно присвоить только перечислитель соответствующего типа:

```
var d Direction = West
var e Direction = NewWest // Ошибка. Значение как у West, но другой тип
var f Direction = Direction(NewWest) // Нормально
```

Перечисления и функции

```
type Direction int

const (
    North Direction = 1
    South Direction = -1
    East  Direction = -2
    West  Direction = 2
)

func move(xpos int, ypos int, d Direction) {
    switch d {
    case North, South:
        xpos += int(d)
    case East, West:
        ypos += int(d) / 2
    }
}

move(0, 0, West)
move(0, 0, 1)
var dir int = 1
move(0, 0, dir) // Ошибка
```

Множество

Что такое множество

Множество — структура данных, которая является реализацией математического объекта множество.

Множество хранит элементы в единственном числе, без дублей. Как правило элементы множества не упорядочены (но это зависит от реализации), поэтому множество относится к ассоциативным типам данных.

В Go нет типа данных "множество", но его аналог можно реализовать при помощи словаря

Множество

```
set := make(map[string]bool)
set["apple"] = true    // Добавляем элементы
set["orange"] = true
set["mango"] = true
fmt.Println(set)

delete(set, "apple")  // Удаляем элемент
fmt.Println(set)

_, ok := set["mango"] // Проверяем присутствие
fmt.Println("mango in set:", ok)
```

Словарь

(карта, мэп, ассоциативный массив)

Что такое словарь

Словарь — структура данных, как и массив, предназначен для хранения набора данных, но вместо одиночных значений словарь хранит пары — ключ и значение.

В словаре ключи не могут дублироваться, а значения могут. Как правило элементы словаря не упорядочены (но это зависит от реализации), поэтому словарь относится к ассоциативным типам данных.

Как определить словарь

```
// имя := make(map[тип ключа] тип значения, длина)
```

```
dict1 := make(map[string]string, 3)
```

```
dict2 := make(map[string]string)
```

```
var dict3 map[string]string // Нельзя использовать
```

```
dict3 = map[string]string{} // Теперь можно
```

Тип значения может быть любым.

Тип ключа должен иметь неизменный хэш.

Инициализация элементов

```
package main

import "fmt"

func main() {
    sammy := map[string]string{
        "name":      "Sammy",
        "animal":    "shark",
        "color":     "blue",
        "location": "ocean",
    }
    fmt.Println(sammy)
}
```

Обращение к элементам

```
package main

import "fmt"

func main() {
    sammy := map[string]string{
        "name":    "Sammy",
        "animal":  "shark",
        "color":   "blue",
        "location": "ocean",
    }
    sammy["cost"] = "10000$"

    fmt.Println(sammy["animal"]) // shark
    fmt.Println(sammy["color"])  // blue
    fmt.Println(sammy["location"]) // ocean
    fmt.Println(sammy["cost"])   // "10000$"
}
```

Перебор элементов

```
package main

import "fmt"

func main() {
    sammy := map[string]string{
        "name":      "Sammy",
        "animal":    "shark",
        "color":     "blue",
        "location":  "ocean",
    }

    for key, value := range sammy {
        fmt.Printf("%q is the key for the value %q\n", key, value)
    }
}
```

Удаление элементов

```
package main

import "fmt"

func main() {
    sammy := map[string]string{
        "name":    "Sammy",
        "animal":  "shark",
        "color":   "blue",
        "location": "ocean",
    }

    delete(sammy, "location") // Удаляем ключ "location"

    for key, value := range sammy {
        fmt.Printf("%q is the key for the value %q\n", key, value)
    }
}
```

Размер словаря

```
package main

import "fmt"

func main() {
    sammy := map[string]string{
        "name":      "Sammy",
        "animal":    "shark",
        "color":     "blue",
        "location":  "ocean",
    }

    fmt.Print(len(sammy))
}
```

Проверка наличия элемента

```
package main

import "fmt"

func main() {
    sammy := map[string]string{
        "name":    "Sammy",
        "animal":  "shark",
        "color":   "blue",
        "location": "ocean",
    }

    cost, ok := sammy["cost"]
    if ok {
        fmt.Printf("Sammy has a cost of %s\n", cost)
    } else {
        fmt.Println("Sammy cost was not found")
    }
}
```

Копирование словаря

```
package main

import "fmt"

func main() {
    sammy := map[string]string{
        "name":    "Sammy",
        "animal":  "shark",
        "color":   "blue",
        "location": "ocean",
    }

    newSammy := sammy           // Указывают на один словарь
    newSammy["cost"] = "10000$" // Меняются оба

    fmt.Print(sammy, newSammy)
}
```

Если нужна копия, то нужно копировать вручную, например в цикле.

Указатель на значение в словаре

```
package main

import "fmt"

func main() {
    sammy := map[string]string{
        "name":      "Sammy",
        "animal":    "shark",
        "color":     "blue",
        "location":  "ocean",
    }

    // Словари не упорядочены и могут перестроиться
    // при добавлении/удалении элементов, поэтому
    // так нельзя
    var color *string = sammy["color"]
}
```

Передача словаря в функцию

```
package main

import "fmt"

func setCost(obj map[string]string) {
    obj["cost"] = "10000$"
}

func main() {
    sammy := map[string]string{
        "name":    "Sammy",
        "animal":  "shark",
        "color":   "blue",
        "location": "ocean",
    }

    setCost(sammy) // Изменится
    fmt.Print(sammy)
}
```