Структура программы

Имена типов данных

- Начинается с буквы или `_`
- Регистрозависимы
- имеется 25 ключевых слов, которые нельзя использовать для именования + ~30 предопределенных имен (которые можно переопределять)

Видимость и правила именования

- Являются локальными внутри функций
- Если обьявлена в области пакета, то видна во всех файлах пакета
- Если переменная пакета начинается прописной буквы, то она видима за пределами пакета.
- Имена пакетов всегдя состоят из строчных букв
- Предпочтение отдается `camelCase`

Обьявления

- Существует 4 разновидности объявлений: `var`, `const`, `type`, `func`
- Каждый файл начинается с объявления `package`, которое говорит, частью какого пакета является файл.
- Затем идут обьявления `import`
- После этого последовательность типов, переменных, констант, функций уровня пакета

```
package main

import "fmt"

const boilingF = 212.0 // константа уровня пакета

func main() { // функция уровня пакета
    var f = boilingF //локальная переменная
    var c = (f - 32) * 5 / 9
    fmt.Printf("точка кипения = %g°F or %g°C\n", f, c)
    // Вывод:
    // точка кипения = 212°F or 100°C
}
```

- функция выполняется или до вызова `return` либо доходит до конца функции и передает управление внешней области.

```
import "fmt"

func main() {
    const freezingF, boilingF = 32.0, 212.0
    fmt.Printf("%g°F = %g°C\n", freezingF, fToC(freezingF)) // "32°F = 0°C"
    fmt.Printf("%g°F = %g°C\n", boilingF, fToC(boilingF))   // "212°F = 100°C"
}

func fToC(f float64) float64 {
    return (f - 32) * 5 / 9
}
```

Переменные

- `var name type = expression` - общий вид, var создает переменную, назначает имя и присваивает начальное значение. Можно пропустить `type` или `expression` но не вместе.
- Значение по умолчанию: `0`, `false`, `""`, `nil`
- в одном обьявлении можно обьявить или определить несколько перменных
- `var i, j, k int // int, int, int`
- `var b, f, s = true, 2.3, "four" // bool, float64, string`
- `var f, err = os.Open(name) // возвращает файл или ошибку`
- Для инициализации локальных переменных можно использовать короткие объявления `name := expression`
- `anim := gif.GIF{LoopCount: nframes}`
- `t := 0.0`
- `i, j := j, i // обмен значениями`
- `f, err := os.Open(name)`
- если в кратком объявлении используется уже обьявленная переменная, то такая операция является присваиванием

Указатели

- переменная это выделенный блок памяти определенного размера содержащщий значение
- значением указателя является адрес переменной, таким образом указатель является местоположением переменной. С попомшью указателя можно считывать или изменять значение переменной не зная ее имени.

- Если переменная обьявлена как `var x int`, то `&x` (адрес x), дает указатель на целочисленную переменную, т.е значение типа `*int`, который произносится как *указатель на int*. Если это значение назвается `p` мы говорим `p` указывает на `x` (содержит адрес `x`).
- Переменная на которую указывает `p`, записывается как `*p`. Выражение `*p` - дает значение этой переменной `int`.
- Так как `*p` обозначает переменную, то мы можем использовать его в левой части для присваивания.

```
 x := 1
 p := &x //p имеет тип *int и указывает на x
 fmt.Println(*p) // "1"
 *p = 2 // равно x = 2
fmt.Println(x) // 2
```

- Каждый компонент переменной составного типа - поле структуры или элемент массива, так-же является переменной, а значит имеет свой адрес.

```
var x, y int
fmt.Println(&x ==&x, &x == &y, &x == nil) // true false false
```

- Функция может возвращать адрес переменной, таким образом переменная созданная внутри функции будет существовать даже после возврата из функции, и указатель будет указывать на нее.

```
var p = f()
func f() *int {
  v :=1
  return &v
}
```

- Каждый вызов `f()` будет возвращать различные значения `fmt.Println(f() == f()) // false`
- Так как указатель содержит адрес переменной, передача указателя в функцию позволяет обновлять значение переменной

```
func incr(p *int) int {
  *p ++ // увеличиваем значение, на которое указывает p
  // не меняет значение p
  return *p
}

v :=1
incr(&v) // v становится равным 2
fmt.Println(incr(&v)) // 3
```

Функция new

- Переменные создавать можно с помошью функции new
- new(T) - создает неименованную переменную типа T, инициализирует значением по умолчанию и возвращает адрес, который представляет собой значение *T

```
p := new(int) // p, имеюший тип *int, указывает на неименованную переменну
ю типа int
fmt.Println(*p) // 0
*p = 2 // устанавливает значение переменной = 2
fmt.Println(*p) // 2
```

- Можно переопределить `func delta(old, new int) int { return new - old }`

Время жизни переменных

- Время жизни переменной уровня пакета - равно времени работы всей программы.
- Локальные переменные имеют динамическое время жизни, всякий раз при вызове функции или блока переменная пересоздается.

```
var global *int
func f() {
  var x int
  x = 1
  global = &x // значение переменной x выходит за пределы функции
}

func g() {
  y := new (int)
  *y = 1 // значение переменной y не выходит за пределы функции
}
```

Присваивания

```
x = 1                       // Именованная переменная
*p = true                   // Косвенная переменная
person.name = "bob"         // Поле структуры
count[x] = count[x] * scale // Элемент массива
count[x] *=scale

v :=1
v++
v--
```

- присваивание кортежу : прежде чем любая из переменных в левой части получит новое значение, вычисляются все выражения в левой части, это позволяет делать обмен значений без необходимости создавать промежуточные переменные
  - `x, y == y, x`
  - `a[i], a[j] = a[j], a[i]`
  - `i, j, k = 2, 3, 5`
  - `v, ok = m[key]` - поиск в отображении
  - `v, ok = x.(T)` - утвержение о типе
  - `v, ok = <-ch` - получение из канала
  - `_, err = io.Copy(dst,src)` - Отбрасываем количество байт

Обьявления типов

- Тип переменной или выражения определяет характеристики значений, которые он может принимать (размер,внутреннее представление, операции которые могут быть над ними выполнены и связанные методы)
- Обьявление `type` определяет новый именованный тип, который имеет тот же базовый тип, что и существующий.
- Дает возможность отличать различные типы базового типа. `type имя базовый_тип`

```
package tempconv

import "fmt"

type Celsius float64 // новый тип
type Fahrenheit float64  // новый тип

const (
  AbsoluteZeroC Celsius = -273.15
  FreezingC     Celsius = 0
  BoilingC      Celsius = 100
)

func (c Celsius) String() string    { return fmt.Sprintf("%g°C", c) }
func (f Fahrenheit) String() string { return fmt.Sprintf("%g°F", f) }
```

- преобразование одного типа в другой разрешено, если они имеют одинаковый базовый тип, или являются неименованными указателями на переменные одного базового типа

```
var c Celsius
var f Farhenheit
fmt.Println(c == 0) // "true"
fmt.Println(f >=0) // "true"
fmt.Println(c == f) // Ошибка компиляции: несоответствие типа
fmt.Println(c == Celsius(f)) // "true"
```

- Именованные типы позволяют задавать поведение начений этого типа. Такое поведение выражается в виде набора функций доступных данному типу, именуемых методами типа.

```
func( c Celsius) String() string {return fmt.Sprintf("%gC",c)}

c:= FToC(212.0)
fmt.Println(c.String()) // 100C
fmt.Println("%v\n",c)   // 100C; явный вызов String() не нужен
fmt.Println("%s\n",c)   // 100C
fmt.Println(c)          // 100C
fmt.Println(%g\n)       // 100; не вызывает Srting
fmt.Println(%g\n)       // 100; не вызывает Srting
fmt.Println(float64(c)) // 100; не вызывает String
```

Пакеты и файлы

- Исходный текст пакета, хранится в одном или нескольких файлах `.go`, обычно в каталоге, имя которого является окнчанием пути импорта
- Каждый пакет служит отдельным пространством имен для своих обьявлений.

- Чтбы обратиться к функции за пределами пакета, надо квалифицировать идентификатор, явно указав, имя пакета `image.Decode` или `utf16.Decode`
- В Go экспортируются идентификаторы, которые начинаются с прописной буквы
    - tempconv.go

```
package tempconv

import "fmt"

type Celsius float64
type Fahrenheit float64

const (
  AbsoluteZeroC Celsius = -273.15
  FreezingC     Celsius = 0
  BoilingC      Celsius = 100
)

func (c Celsius) String() string    { return fmt.Sprintf("%g°C", c) }
func (f Fahrenheit) String() string { return fmt.Sprintf("%g°F", f) }
```

    - conv.go

```
package tempconv

// CToF converts a Celsius temperature to Fahrenheit.
func CToF(c Celsius) Fahrenheit { return Fahrenheit(c*9/5 + 32) }

// FToC converts a Fahrenheit temperature to Celsius.
func FToC(f Fahrenheit) Celsius { return Celsius((f - 32) * 5 / 9) }
```

- Имена констант начинаются с прописных букв, они доступны вне пакета `tempconv.AbsoluteZeroC`
- Чтобы преобразовать температуру во внешнем пакете можно вызвать `tempconv.CToF(tempconv.BoilingC)` / 212 F

Импорт

- Каждый пакет идентифицируетя уникальной строкой, которая называется *путем импорта*. *Путь импорта* обозначает каталог, содержащий один или несколько файлов совместно образующщих пакет.
- В дополнении к *пути импорта* пакет имеет *имя пакета* которое находится в обьявлении `package`
- По соглашению имя пакета является последней частью пути импорта
    - cf

```
package main

import (
  "fmt"
  "os"
  "strconv"

  "../tempconv"
)

func main() {
  for _, arg := range os.Args[1:] {
    t, err := strconv.ParseFloat(arg, 64)
    if err != nil {
      fmt.Fprintf(os.Stderr, "cf: %v\n", err)
      os.Exit(1)
    }
    f := tempconv.Fahrenheit(t)
    c := tempconv.Celsius(t)
    fmt.Printf("%s = %s, %s = %s\n",
      f, tempconv.FToC(f), c, tempconv.CToF(c))
  }
}
```

- импорт пакета без последующего его использования считается ошибкой
- инструмент `golang.org/x/tools/cmd/goimports` автоматически добавляет и удаляет пакеты из объявлений импорта.

Инициализация пакетов

- Инициализация пакетов начинается с инициализации переменых уровня пакета - в том порядке, в котором они объявлены, с тем исключением что разрешаются зависимости:

```
var a = b + c // инициализируется третьей, значением 3
var b = f()   // инициализируется второй, значением 2 из вызова f()
var c = 1     // инициализируется первой, значением 1

func f() int { return c + 1 }
```

- Если пакет состоит из нескольких файлов, они инициализируются в том порядке, в котором передаются компилятору, перед компиляцией они упорядочиваются в алфавитном порядке
- Каждая переменная уровня пакета, начинает свое существование с тем, значением которым оно было инициализировано
- Любой файл может содержать функцию `init`, любой файл может иметь любое количество таких функций: `init() { /*...*/ }`, такие функции нельзя

вызывать или обращаться к ним, в таком файле функции `init` выполняются автоматически при запуске программы в порядке их обьявления.

- Инициализация пакетов происходит в порядке обьявления, первыми обрабатываются зависимости.
- Последним инициализируется пакет `main()`

```
package popcount

  //pc[i] количество единичных битов в i.
  var pc [256]byte

  func init() {
    for i := range pc {
      pc[i] = pc[i/2] + byte(i&1)
    }
  }

   //PopCount возвращает степень заполнения (количество установленных бито
в) значения x.
  func PopCount(x uint64) int {
    return int(pc[byte(x>>(0*8))] +
      pc[byte(x>>(1*8))] +
      pc[byte(x>>(2*8))] +
      pc[byte(x>>(3*8))] +
      pc[byte(x>>(4*8))] +
      pc[byte(x>>(5*8))] +
      pc[byte(x>>(6*8))] +
      pc[byte(x>>(7*8))])
  }
```

Область видимости

- Область видимости - это область исходного текста программы; это свойство времени компиляции. Временем же жизни переменной называют диапазон времени выполнения, когда к переменной можно обращаться из других частей программы; это своейство времени выполнения.
- Синтаксический блок `{ }`, имя обьявленное внутри него не видимо из вне блока. Блок определяет область видимости локальных переменных.
- Программа может иметь несколько одинаковых обьявлений одноименных переменных, они должны находится в раличных областях.
- Переменные обьявленные во внешних областях видимости доступны во входящих в них внутренних, при условии что они не переобьявлены в последних.

```
func f() {}
var g = "g"

func main() {
  f :="f"
  fmt.Println(f) // переменная f "затемняет" внешнюю функцию f()
  fmt.Println(g) // переменная уровня пакета
  fmt.Println(h) // Ошибка компиляции
}
```

- Внутри функции лексические блоки могут быть вложенными произвольной глубиной, поэтому одно локальное обявление может затенять другое.

```
func main() {
  x := "hello!"
  for i :=0; i < len(x); i ++ {
    x := x[i]
    if x != '!' {
      x := x + 'A' - 'a'
      fmt.Println("%c",x) // HELLO (по букве за итерацию)
    }
  }
}
```

- В примере обьявлены три x каждая в своей области видимости

```
func main() {
  x := "hello"
  for _, x := range x {
    x := x + 'A' - 'a'
    fmt.Printf("%c",x) // HELLO по букве за итерацию
  }
}
```

- В программе обслатью видимости `f` является блок `if`, программа вызовет ошибку:

```
if f, err :=os.Open(fname); err !=nil {
  // ошибка f неиспользуется
}

f.Stat()  // Ошибка компиляции: неопределенная переменная f
f.Close() // Ошибка компиляции: неопределенная переменная f
```

- Решение

```
f, err := os.Open(fname)
if err != nil {
  return err
}

f.Stat()
f.Close()
```

- Еще одно

```
if f, err :=os.Open(fname); err !=nil {
  return err
} else {
  f // здесь видима
  f.Stat()
  f.Close()
}
```