**РАЗДЕЛ 2. ОБЗОР ЯЗЫКА ПРОГРАММИРОВАНИЯ GOLANG**

**2.1. Общая характеристика**

Поскольку в этой работе используется интегрированный подход к программной реализации алгоритмов, необходимо хотя бы кратко описать основные концепции и конструкции его составляющих: визуальный алгоритмический язык Drakon и язык программирования Golang. Думается, что такой подход является весьма перспективным, прежде всего, в сфере образования, поскольку способствует формированию и развитию алгоритмического, компьютерного мышления и вполне оправдан для применения в такой области компьютерных наук как структуры данных и алгоритмы. Однако при этом предполагается, что читатель знаком хотя бы с одним из современных языков программирования. В то же время описание языка Golang ограничивается конструкциями, достаточными для понимания алгоритмов структур данных.

Основное внимание в этом разделе уделено особенностям программных конструкций: переменные, массивы, срезы, карты, указатели, логические операторы, циклы, структуры, рекурсии и т. д. Особый акцент делается на описании типов данных как базовых, так и пользовательских. Рассматривается также такая конструкциюя языка как и*нтерфейс*. Все, что выходит за рамки направления «Структуры данных и алгоритмы» представлено в различных материалах, которые приведены как по ходу текста, так и в перечне литературы.

Golang (Go) - язык общего назначения, разработанный с учетом системного программирования. Первоначально язык был разработан в Google в 2007 году Робертом Гриземером, Роб Пайком и Кен Томпсоном. Он сильно и статически типизирован (требуется точное указание типов переменных), обеспечивает встроенную поддержку сбора мусора и поддерживает параллельное программирование [].

Ниже перечислены наиболее важные особенности программирования Go:

* *Простой и понятный синтаксис*. Это делает написание кода приятным занятием.
* *Статическая типизация*. Позволяет избежать ошибок, допущенных по невнимательности, упрощает чтение и понимание кода, делает код однозначным.
* *Скорость и компиляция*. Скорость у Go в десятки раз быстрее, чем у скриптовых языков, при меньшем потреблении памяти. При этом, компиляция практически мгновенна. Весь проект компилируется в один бинарный файл, без зависимостей. Как говорится, «просто добавь воды». И вам не надо заботиться о памяти, есть сборщик мусора.
* *Отход от концепции объектно-ориентированного программирования* (ООП). В языке нет классов, но есть структуры данных с методами. Наследование заменяется механизмом встраивания. Существуют интерфейсы, которые не нужно явно имплементировать, а лишь достаточно реализовать методы интерфейса.
* *Параллелизм*. Параллельные вычисления в языке делаются просто и изящно. Горутины (что-то типа потоков) легковесны, потребляют мало памяти.
* *Богатая стандартная библиотека*. В языке есть все необходимое для веб-разработки и не только. Количество сторонних библиотек постоянно растет. Кроме того, есть возможность использовать библиотеки C и C++.
* *Возможность писать в функциональном стиле*. В языке есть замыкания (closures) и анонимные функции. Функции являются объектами первого порядка, их можно передавать в качестве аргументов и использовать в качестве типов данных.

**2.2. Структура программы на языке Go**

Структура программы на языке Golang представлена на рис. 2.1.

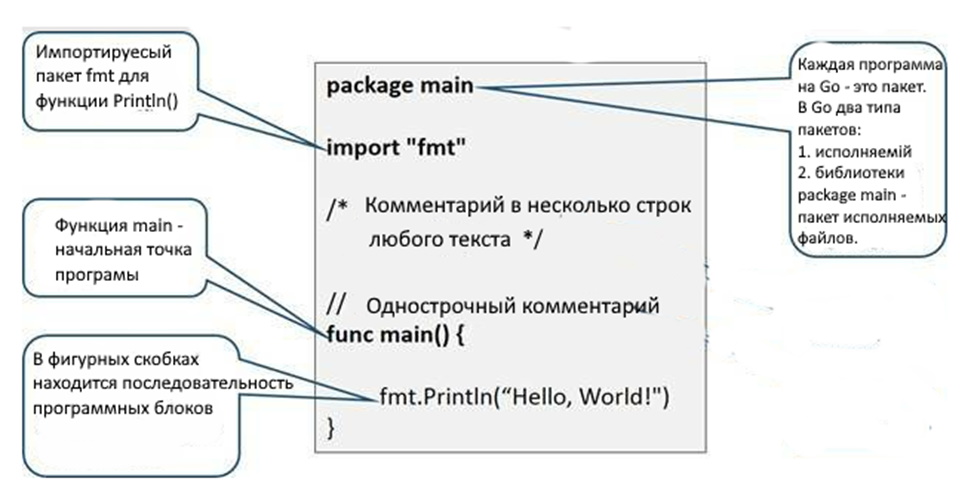


Рис. 2.1. Структура программы на языке Go

Каждая программа на Go — это пакет. В зависимости от его содержимого, этот пакет может стать основным файлом программы, а может — библиотекой с проблемно-ориентированным кодом.

// название программы (пакета)

package main

……………………….

// подгружаем нужные пакеты, если команды из этих пакетов понадобятся в нашей программе

import (

"fmt",

“flag”

)

// основная функция — main — означает, что её содержимое и будет выполняться после запуска программы

// кроме неё в программе может быть сколько угодно других функций

func main() {

// содержимое функции

fmt.Println("Hello World")

}

Комментарии в Go существуют в двух видах: однострочные и многострочные.

// - однострочный комментарий;

/\* \*/ многострочный комментарий, расположенный между этими символами

**2.3. Переменные и константы**

Переменная в любом языке программирования назначает место хранения значения, связанного с символическим именем или идентификатором. Одна переменная или список переменных, используемых в программе, объявляется через ключевое слово – var. Присваивание значений переменным осуществляется знаком «=».

При объявлении переменной обязательно указівается ее тип, чтобы компилятор знал, как обрабатывать эти данные. Именно поэтому язык Go называется строго типизированным.

var i int

var s string

var f bool

здесь i – целочисленная переменная; s - строковая переменная, f – логическая переменная.

При объявлении переменной допускается присвоение начальных значений, которые в дальнейшем могут изменяться:

var a int = 25

var a, b, c int 100, 200 300

var y = float32 = 32.5

var z string = "Введите новое значение "

При отсутствии начальных значений переменные либо обнуляются (для числовых типов), либо заполняются пустыми строками (для строковых данных).

Возможно короткое объявление переменных через через оператор :=, например, a := 2.5. Однако в этом случае переменная является локальной, то есть доступной только в отдельном фрагменте программы.

Присвоение имен переменным должно ориентироваться на определенные правила и стиль.

* имена переменных могут состоять только из одного слова (без пробелов).
* имена переменных могут состоять только из букв, цифр и символов подчеркивания (\_).
* имена переменных не могут начинаться с цифр.

В переменных учитывается регистр, однако регистр первой буквы имени переменной в Go имеет особое значение. Если имя переменной начинается с заглавной буквы, это означает, что данная переменная доступна за пределами пакета, где она была декларирована (экспортируемая переменная). Если имя переменной начинается со строчной буквы, она будет доступна только в том пакете, где она декларирована.

Константы представляют собой объекты программы, значения которых в программном коде не меняются. Константы объявляются через ключевое слово: const:

const item string

const n int

const y float64

При этом допускается присвоение начальных значений без объявления типа:

const item = "name"

const n = 25

const  y = 45.5

**2.4.. Ввод и вывод**

Ввод данных осуществляется с помощью функций *Scan(&имя\_переменной)* чтобы просто поместить введённое значение в переменную или *Scanf(%формат, &имя\_переменной)* чтобы заранее указать тип данных, которые будем вводить.

Для реализации вывода данных в начало программы (после package main) импортируется соответствующий пакет fmt, в котором находится множество функций вывода в зависимости от типа переменных, расположения и поясняющей строки, заключенной в двойные кавычки (\n – перевод строки). Для связи с соответствующим пакетом используется префикс fmt.

fmt.Print (“Привет”) // вывод без перевода на новую строку

fmt.Println () // вывод с переводом курсора на новую строку

fmt.Printf() // вывод значений переменных в текст

fmt.fmt.Printf("Hello %d\n", 23) // вывод строки и числа целого типа

fmt.fmt.Fprint("Hello ", 23, "\n") //то же с переводом на новую строку

fmt.fmt.Println("Hello", 23) // вывод строки и числа

fmt.Printf("В этом году я проехал %v городов, ", &output)

Например,

package main

import “fmt”

func main() {

    var name string

    var salary int

    fmt.Println("Введите фамилию")

    fmt.Scanf("%s \n", &name)

    fmt.Println("Введите зарплату")

    fmt.Scanf("%d \n", &salary)

    fmt.Printf("%s \n",name)

    fmt.Printf("%d \n",salary)

    fmt.Printf("Зарплата %s составляет %d рублей", name, salary)

}

Результат:

Введите зарплату

Степанова

Введите зарплату

32000

Степанова

32000

Зарплата Степанова составляет 32000 рублей

**2.5. Операторы принятия решений**

При решении множества задач возникает проблема выбора дальнейшего пути решения в зависимости от некоторых условий. В языке Go такая возможность реализуется следующими конструкциями:

* выбор по условию: if else; switch case; select ;
* повтор команд с помощью итераций: for (range);
* изменение поведения в процессе итераций: break и continue.

а). Оператор выбора *if-else*

В языке Go используются три конструкции принятия решений или, иными словами, выбора варианта дальнейшего вычислительно процесса: *if-else,*  *switch* и select*.* Синтаксис первого оператора имеет вид:

if condition {

// исполняемый код при условии condition == true

} else {

// исполняемый код при условии condition == false

}

При одиночном сравнении фигурные скобки можно не ставить. Открывающая скобка остаётся на той же строке, что и условие:

package main

import "fmt"

func main() {

// если условие верно

if ID == "Apple" {

// то выполняется то, что заключено в скобках:

fmt.Println("Введите свой логин и пароль")

// если условие не выполнилось, то можно сразу проверить ещё одно условие

// так можно делать сколько угодно раз

} else if ID =="Google" {

// выводим второй ответ

fmt.Println("Ваша операционная система не поддерживается")

// выполняем то, что относится к последнему if

} else {

fmt.Println("Ошибка ввода")

}

}

б). Оператор переключения switch

Переключатель выражений *switch* предоставляет собой простую конструкцию обращения к различным частям программы на основе значения выражения. Синтаксис оператора *switch:*

switch optstatement; optexpression{

case expression1: Statement..

case expression2: Statement..

...

default: Statement..

}

После первого найденного варианта оператор выполняет нужные действия и прекращает работу.

switch ID {

// проверяется первое значение

case "Apple":

fmt.Println("Введите свой логин и пароль")

// проверяется второе значение

case "Google":

fmt.Println("Ваша операционная система не поддерживается")

// если ничего нужного не нашлось

default:

fmt.Println("Ошибка ввода")

}

в). Оператор выбора select

Оператор select позволяет выбирать для выполнения одно из нескольких выражений. Главное различие между select и switch заключается в том, что select работает по принципу ожидания. Это означает, что команда select не будет выполняться до тех пор, пока не будет выполнено сообщение, связанное с отправкой и получением по любому каналу. Следует отметить, что в визуальном алгоритмическом языке DRAKON в реализации редактора Drakon Web Editor используется подобный оператор Select, что будет рассмотрено в следующем разделе.

**2.6. Циклы**

В языке Golang принят только один формат цикла - *for*, имеющий следующие варианты.

а). Классический, С-подобный цикл с переменной, условием и шагом цикла:

for i := 0; i < 8; i++ { Тело цикла }

б). Цикл с предусловием.

Самый простой цикл —объявить только условие, а остальное разместить внутри цикла:

package main

import "fmt"

func main() {

// переменная для цикла

var count = 10

// пока переменная больше 0 — цикл работает

for count > 0 {

// выводим текущее значение переменной

fmt.Println(count)

// уменьшаем её на единицу

count = count - 1

}

в). Цикл в диапазоне

Существует еще одна разновидность цикла for, выполняющая итерации для диапазона значений для типа данных. Ключевое слово range используется в цикле для итерации по элементам массива, среза или карты. При использовании массива и среза цикл возвращает индекс элемента в виде целого числа. При использовании карт он возвращает ключ следующей пары ключ-значение:

package main

import "fmt"

func main() {

    nums := []int{2, 3, 4}

    sum := 0

    for \_, num := range nums {

        sum += num

    }

    fmt.Println("sum:", sum)

    for i, num := range nums {

        if num == 3 {

            fmt.Println("index:", i)

        }

    }

    kvs := map[string]string{"ca": "Paris", "co": "France"}

    for k, v := range kvs {

        fmt.Printf("%s -> %s\n", k, v)

    }

    for k := range kvs {

        fmt.Println("key:", k)

    }

}

The following table provides a complete list of Golang keywords:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| break | case | chan | const | continue |
| default | defer | else | fallthrough | for |
| func | go | goto | if | import |
| interface | map | package | range | return |
| select | struct | switch | type | var |

**2.7. Типы данных**

Типы данных определяют виды значений, которые сохраняются переменными при написании программы. Типы данных также помогают определить операции, которые можно выполнять с использованием данных. Раздичают следующие виды типов данных.

* Базовый тип (Fundamental types). К этой категории относятся *числа, строки и логические (булевы)* значения.
* Агрегированный тип (Aggregate types). К этой категории относятся *массив и структуры*.. struct
* Ссылочный тип (Reference types). К этой категории относятся *указатели, срезы, ассоциативный массив (карта) и, функции*.
* Встраиваемый тип (Embedding types). Go поддерживает определяемые пользователем типы в виде типов псевдонимов или структур.

**2.7.1. Базовые типы**

а). Целочисленные и вещественные типы переменных и констант

Целочисленные типы Go делятся на целые со знаком и целые без знака. Целочисленные типы со знаком включают int, int8, int16, int32 и int64; целые типы без знака включают uint, uint8, uint16, uint32, uint64, uintptr. Диапазон значений и объем занимаемой памяти целочисленных типов представлены в табл. 2.1:

Табл. 2.1

| Тип | Диапазон | Занимаемая память |
| --- | --- | --- |
| int8 | –128 — 127 | 8 бит (1 байт) |
| uint8 | 0 — 255 |
| int16 | –32 768 — 32 767 | 16 бит (2 байта) |
| uint16 | 0 — 65535 |
| int32 | –2 147 483 648 — 2 147 483 647 | 32 бита (4 байта) |
| uint32 | 0 — 4 294 967 295 |
| int64 | –9 223 372 036 854 775 808 — 9 223 372 036 854 775 807 | 64 бита (8 байт) |
| uint64 | 0 — 18 446 744 073 709 551 615 |

В языке Go принято два формата чисел с плавающей запятой:.

• float32 - Самым большим float32 является константа math.MaxFloat32, которая составляет около 3,4e38. Наименьшее положительное значение - 1,4e-45/.

• float64 - Самым большим float64 является константа math.MaxFloat64, которая составляет около 1,8e308. Наименьшее положительное значение 4,9e-324.

Float32 обеспечивает примерно шесть десятичных знаков точности, в то время как float64 предоставляет около 15 цифр. Тип float64 является более предпочтительным для большинства задач, поскольку при использовании типа float32 во многих итерационных алгоритмах быстро накапливается ошибка округления.

Для определения к какому типу данных компилятор Go относит ту или иную переменную используется функция Printf(), в параметрах которой присутствует специальный символ %T:

а := 234,45

fmt.Printf("Type %T for %v\n", a, a)

Результат: Type float64 for 234.45

б) Строковые переменные и константы

Строка - это неизменяемая последовательность байтов. Строки могут содержать произвольные данные, включая байты со значением 0, но обычно они содержат читабельный текст. Текстовые строки обычно интерпретируются как UTF-8-кодированные последовательности кодовых точек. Встроенная функция len возвращает число байт в строке, а оператор печати по спецификации %x возвращает один байт строки s для латиницы и два байта для кириллицы:

func main() {

    s := "Hello, World"

    fmt.Println("Строка >> ", s)

    fmt.Println("Длина строки = ", len(s))

    fmt.Println("Hex bytes: ")

    for i := 0; i < len(s); i++ {

        fmt.Printf(" %x",s[i])

    }

    fmt.Println()

    q := "Привет, Мир"

    fmt.Println()

    fmt.Println("Строка >> ", q)

    fmt.Println("Длина строки = ", len(q))

    for i := 0; i < len(q); i++ {

        fmt.Printf(" %x",q[i])

        }

Результат:

Строка >> Hello, World

Длина строки = 12

Hex bytes:

48 65 6c 6c 6f 2c 20 57 6f 72 6c 64

Строка >> Привет, Мир

Длина строки = 20

d0 9f d1 80 d0 b8 d0 b2 d0 b5 d1 82 2c 20 d0 9c d0 b8 d1 80

в) Логические переменные и константы

Логический тип данных (bool) может иметь одно из двух значений: **true** (истина) или **false** (ложь). Булевы операторы используются в программировании для сравнения и для контроля потока данных:

func main() {

x := 5

     y := 8

    fmt.Println("x == y:", x == y) // равно

    fmt.Println("x != y:", x != y) // не равно

    fmt.Println("x < y:", x < y) // меньше

     fmt.Println("x > y:", x > y) // больше

    fmt.Println("x <= y:", x <= y) // меньше или равно

    fmt.Println("x >= y:", x >= y) // больше или равно

    }

Результат:

x != y: true

x < y: true

x > y: false

x <= y: true

x >= y: false

**2.7.2. Аггрегированные типы**

а) Массив (Array).

Массив представляет собой последовательность данных фиксированной длины, которая используется для хранения в памяти однородных элементов. Массивы в Go практически аналогичны массивам в других языках программирования. Элементы массива индексируются с помощью оператора [] index с их нулевой позицией, что означает, что индекс первого элемента равен array[0], а индекс последнего элемента — array[len(array)-1], где len(array) – длина массива. Однако из-за фиксированной длины массивы не очень популярны в отличие от конструкции Slice (Срез), которая несравненно чаще используется в Go. Синтаксис массива выглядит таким образом:

[N]T{value1, value2, value3, ...value n}, где N – число элементов

б) Структура (Structure)

Конструкция Structure представляет собой тип данных, определяемый разработчиком и служащий для представления каких-либо реальных объектов. Структуры содержат набор полей, которые представляют различные атрибуты объекта. Для определения структуры применяются ключевые слова **type** и **struct**.

type имя\_структуры struct{

    поля\_структуры

}

Ниже представлен пример структуры данных о сотрудниках предприятия:

package main

import "fmt"

type Employee struct {

firstName, lastName, address string // поля строкового типа

age, phone, salary int // поля целого типа

}

func main() {

// инициализация структуры

emp := Employee{firstName: "Евгений", lastName: "Смирнов", age: 42, phone: 123456789, salary: 34000, address: "Курск"}

fmt.Println("Имя и фамилия сотрудника: ", emp.firstName, emp.lastName)

fmt.Println("Возраст сотрудника: ", emp.age)

fmt.Println("Зарплата сотрудника : ", emp.salary)

fmt.Println("Телефон сотрудника : ", emp.phone)

fmt.Println("Адрес сотрудника : ", emp.address)

}

Память для новой переменной типа struct выделяется с помощью функции new, которая возвращает указатель на выделенную память:

var w \*T = new(T),

или в разных строчках, если структура объявлена в области пакета:

var w \*T

w = new(T)

При использовании краткой формы присвоения значения переменной (:=), то есть, w := new(T) переменная z является указателем на участок памяти, в котором поля структуры содержат нулевые значения в соответствии с их типами:

type structEmpl struct {

name string

age int

salary float64

}

func main() {

 ms := new(structEmpl)

 fmt.Println(ms)

 ms.name = "Рыжов"

 ms.age = 45

 ms.salary = 1200.5

 //fmt.Printf("Фамилия: %s\n", ms.name)

 fmt.Printf("Возраст: %d\n", ms.age)

 fmt.Printf("Зарплата: %8.1f\n", ms.salary)

 fmt.Println(ms)

}

**2.7.3. Ссылочные типы**

а) Указатели (Pointer)

Указатели в языке программирования  [Golang](https://www.geeksforgeeks.org/go-programming-language-introduction/)— это переменные, которые используются для хранения адреса памяти другой переменной. [Переменные](https://www.geeksforgeeks.org/go-variables/) используются для хранения некоторых данных по определенному адресу памяти в системе. Адрес памяти всегда представляется в шестнадцатеричном формате (начиная с 0x, например, 0xFFAAF и т. д.) (рис. 2.2):

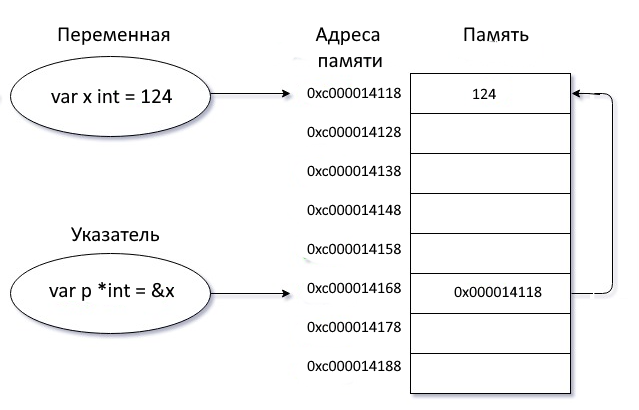


Рис. 2. Переменная, указатель, память

Указатель обычно называют особым видом переменной. Основными и единственными операторами указателей являются: оператор разыменования (\*) и оператор адреса (&). Оператор разыменования (\*) используется для объявления переменной указателя и доступа к значению, хранящемуся в адресе. Оператор адреса (&) используется для возврата адреса переменной или для доступа к адресу переменной указателю.

Синтаксис объявления указателя:

var pointer\_name \*Data\_Type,

где Data\_Type - любой допустимый тип данных, например, var pos \*string.

Для работы с **указателем** его необходимо инициализировать с адресом памяти другой переменной с помощью оператора адреса &, как показано в следующем примере:

var a = 124

var s \*int = &a

Неинициализированный указатель всегда будет иметь нулевое значение <nil>. Ниже приведен пример инициализации и обработки указателя:

func main() {

var z1, z2 int = 64, 128

var p1, p2 \*int

p1 = &z1 // указатель p1 инициализирован

fmt.Println("Значение переменной z1 = ", z1)

fmt.Println("Адрес z1 = ", &z1)

fmt.Println("Значение переменной z2 = ", z2)

fmt.Println("Адрес z2 = ", &z2)

// указатель p2 неинициализирован

fmt.Println("Значение, сохраненное в переменной p1 = ", p1)

fmt.Println("Значение, сохраненное в переменной p2 = ", p2)

}

Результат:

Значение переменной z1 = 64

Адрес z1 = 0xc000086080

Значение переменной z2 = 128

Адрес z2 = 0xc000086088

Значение, сохраненное в переменной p1 = 0xc000086080

Значение, сохраненное в переменной p2 = <nil>

Оператор разыменования \* используется для объявления переменной указателя и для доступа к значению, хранящемуся в переменной, на которую указывает указатель:

func main() {

    var y = 157

    var p = &y

    fmt.Println("Значение переменной в y = ", y)

    fmt.Println("Адрес переменной y = ", &y)

    fmt.Println("Значение, сохраненное в p = ", p)

    fmt.Println("Значение, сохраненное в  y(\*p) = ", \*p)

}

Результат:

Значение переменной в y = 157

Адрес переменной y = 0xc000014078

Значение, сохраненное в p = 0xc000014078

Значение, сохраненное в y(\*p) = 157

б) Срез (Slice)

Срезы представляют собой последовательности переменной длины, все элементы которых имеют один и тот же тип. Записывается тип среза [] T, где элементы имеют тип T.

Cрез состоит из трех компонентов:

Указатель: указатель указывает на элемент массива, доступный через срез, который не обязательно является первым элементом массива.

Длина: количество элементов среза. Он не может превышать емкость.

Емкость: емкость обычно представляет собой количество элементов между началом среза и концом базового массива, представляющего может меняться.

Встроенные функции *len* и*cap* содержатзначения длины и емкости среза соответственно. Несколько срезов могут совместно использовать один и тот же базовый массив и могут ссылаться на перекрывающиеся части этого массива. **Первая позиция индекса в срезе всегда равна 0, а последняя равна длине среза минус единица (len– 1)**. Срез объявляется так же, как массив, но он не содержит размера среза. Таким образом, он может расти или уменьшаться в соответствии с тем или иным алгоритмом.

Синтаксис конструкции среза имеет вид:

[]T или []T{} или []T{value1, value2, value3, ...value n}

Здесь T — это тип элементов. Например: var my\_slice [] int

На рисунке (рис. 2.3) показано визуальное представление среза через его компоненты – *указатель (ptr), длину (len) и емкости (cap):*

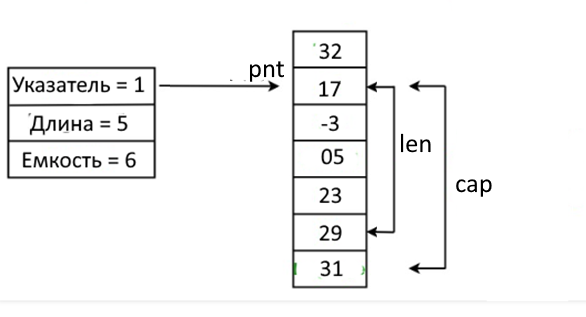
**

Рис.2.3. Представление среза в памяти через компоненты

Срезы можно создавать из уже существующих срезов, указав соответствующие индексы (рис. 2.4):

package main

import (

    "fmt"

)

func main() {

    var sr = []int{17, -21, 5, 62, 24, 48, 78, -43}

    sr0 := sr[2:6]

    fmt.Println("sr0 = ", sr0)

    sr1 := sr[2:]

    fmt.Println("sr1 = ", sr1) // prints [5 7 9 11 13 15]

    sr2 := sr1[:3]

    fmt.Println("sr2 = ", sr2)

}

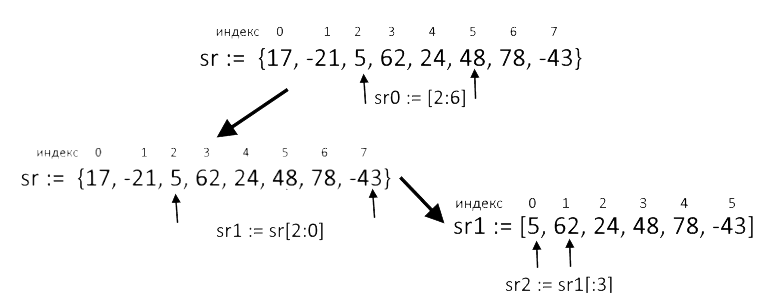


Рис. 2.4. Создание новых срезов из существующих срезов

Важно отметить, что каждому созданному срезу sr0, sr1, sr2 отводится свой участок памяти, то есть, все созданные срезы имеют различные адреса, даже в том случае, когда они являются фрагментами одного среза.:

sr --> 0xc000004078; sr0 --> 0xc000004090; sr1 --> 0xc0000040c0;

sr2 > 0xc0000040f0

2.7.3. Ассоциативные массивы (Map)

Ассоциативные массивы (в дальнейшем *карты*) представляют собой контейнеры хранения пары *"ключ-значение*". Карты являются одной из самых распространенных и полезных структур данных. В отличие от таких структур данных как массив или срез карта обепечивает быстрый и эффективный поиск элементов. Карта не допускает дублирования ключей, но может иметь повторяющиеся значения.

Синтаксис объявления карты, в которой ключ имеет тип *int,* а значение *- string:*

var names map[int]string

Инициализация карты может быть выполнена двумя способами:

С помощью функции *make()*: var names = make(map[int]string);

С помощью синтаксиса литералов: var films = map[int]string;

Ниже приведен пример, иллюстрирующий оба метода инициализации карты:

func main() {

  names := map[int]string{0: "Жаров", 1: "Лавров", 2: "Ульянов", 3: "Матвеев"}

    fmt.Println(names)

    films := make(map[string]string)

    films["Жаров"] = "Путевка в жизнь"

    films["Лавров"] = "Так победим"

    films["Ульянов"] = "Председатель"

    films["Матвеев"] = "Любовь земная"

    fmt.Sprintln(films)

    for key, value := range films {

        fmt.Println(" \n", key, “>”,value)

    }

}

}

Результат:

Матвеев > Любовь земная

Жаров > Путевка в жизнь

Лавров > Так победим

Ульянов > Председатель

**2.8. Функции и методы**

2.8.1. Функции

Функция представляет собой группу операторов, совместно выполняющих ту или иную задачу. С помощью функций можно многократно вызывать ее блок операторов как единое целое в других частях программы. Функции в Go могут быть назначены переменным, переданы в качестве аргумента и могут быть возвращены из другой функции. При объявлении функции нужно указать какого типа переменные передаются в функцию (*параметры*) и какого типа данные возвращает функция (*возвращаемые тивы*). В Go необходимо указывать [тип данных](https://www.digitalocean.com/community/tutorials/understanding-data-types-in-go) для каждого параметра. Для объявления функции используется ключевое слово func. Общая структура объявления функции показана ниже:

func Имя функции [Параметры) (Возвращаемые типы)

{

Тело функции

}

Функция умножения двух вещественных чисел multiply(x,y) выглядит следующим образом:

package main

import "fmt"

func multiply(x,y float64) float64 {

var res

res = x \* y

return res // возвращение результата

}

func main() { // выполняемая функция

var x float64 = 15

var y float64 = 120.6

var mult float64 // переменная для результата

mult = multiply(x,y) // результат работы функции

fmt.Print(mult) // вывод результата

}

Функция может возвращать столько значений, сколько требуется:

func main() {

    var a, b int = 60, 20

    vAdd, vSub := addSub(a, b)

    fmt.Printf("a + b = %d\n", vAdd) // prints "35 + 25 = 60"

    fmt.Printf("a - b = %d\n", vSub) // prints "35 - 25 = 10"

}

func addSub(x, y int) (int, int) { // multiple return values (int, int)

    return x + y, x - y

}

Важной особенностью функций в Go является возможность принимать параметры по значению или по ссылке [].

2.8.2. Методы (Methods)

Методы вместе с функциями предоставляют различные способы организации программного кода. Метод в языке Go - это функция, связанная с определенным типом, которая действует на переменную определенного типа, называемую приемником (получателем). Язык Golang не поддерживает концепцию Class, общепринятую в других языках. Именно с помощью метода объект предоставляет свои свойства, включая поведение объекта. Метод должен удовлетворять следующим условиям: он должен быть определенного типа и должен быть определен в том же пакете.

Объявление метода похоже на объявление функции, но оно имеет дополнительную часть объявления параметра. Дополнительный параметр может содержать один и только один параметр типа получателя метода. Параметр получателя должен быть заключен в круглые скобки  и объявлен между ключевым словом func и именем метода.

Func (Имя\_приемника Type) Имя\_метода(Параметры)(Возвращаемый тип){

// Блок операторов

}

В языке Go разрешено определять метод, приемник которого имеет тип структуры. Этот *приемник* доступен внутри метода, как показано в приведенном ниже примере. Объявлена структура-приемник *rec,* описывающая объект *rect*:

type rect struct {

var width, height int

}

С этой структурой связываются два метода, определяющих поведение объекта, описываемого структурой *rect*:

|  |
| --- |
| func (r rect) area() int {  return r.width \* r.height  } |
| func (r rect) perim() int {  return 2\*(r.width + r.height)  } |
|  |

**2.9. Интерфейсный тип (Interface type)**

Golang не является классическим объектно-ориентированным языком, то есть, он не поддерживает реализацию в прямом виде концепции «классов» и «наследования». Однако Go содержит очень гибкую концепцию интерфейсов, которая обеспечивает многие аспекты объектно-ориентированного программирования.

Интерфейсы в Go обеспечивают способ указания поведения объекта с помощью набора методов, определяемых типом интерфейса. Переменная типа интерфейса может хранить значение любого типа с набором методов, который является любым надмножеством интерфейса. Концепция интерфейса позволяет организовывать разные группы методов, применяемых к объектам разной природы. Иными словами, интерфейсы представляют собой коллекции сигнатур методов, декларирующих имя, параметры типов и возвращаемые типы методов в интерфейсе.

Синтаксис интерфейса имеет вид:

type Namer interface { // Namer – интерфейсный тип

Method1(param\_list) return\_type

Method2(param\_list) return\_type

Например, нужно создать метод для определения площади геометрических фигур – круга и прямоугольника.

type shape interface {

    area() float64

    perimeter() float64

}

Этот код определяет интерфейс для фигур и объявляет две функции, area() и perimeter() с возвращаевым типом float64.

package main

import "fmt"

// Объявление интерфейса

type shape interface {

    area() float64

    perimeter() float64

}

// Объявление структуры «прямоугольник»

type rectangle struct{

  length, height float64

}

// Объявление структуры «круга»

type circle struct{

  radius float64

}

// Объявление методов для прямоугольника

func (r rectangle) area() float64 {

    return r.length \* r.height

}

func (r rectangle) perimeter() float64 {

    return 2 \* r.length + 2 \* r.height

}

// Объявление методов для круга

func (c circle) area() float64 {

    return 3.142 \* c.radius \* c.radius

}

func (c circle) perimeter() float64 {

    return 2 \* 3.142 \* c.radius

}

func main() {

    r := rectangle{length: 10.0, height: 5.0}

    c := circle{radius: 5.0}

    fmt.Printf("Площадь прямоугольника - %8.1f\n", r.area())

    fmt.Printf("Париметр прямоугольника - %8.1f\n", r.perimeter())

    fmt.Printf("Площадь круга - %8.1f\n", c.area())

    fmt.Printf("Париметр круга - %8.1f\n", c.perimeter())

}

Результат:

Площадь прямоугольника - 50.0

Париметр прямоугольника - 30.0

Площадь круга - 78.5

Пeриметр круга - 31

Заключение

Следует еще раз отметить, что в данном разделе приведены элементарные сведения по языку программирования Golang, достаточные для понимания материала, изложенного в дальнейшем. За пределами раздела остались такие сведения как Go-подпрограммы и каналы, параллельность и совместно используемые переменные, пакеты и инструменты Go, рефлексия и т.д. Далее, в соответствующих разделах будут более подробно объясняться инструменты и конструкции, изложенные в этом разделе. Кроме того, будет детально пояснен инструмент рекурсии в силу его особого значения для итерационных процессов в алгоритмах обработки структур данных.

Кроме того, необходимо отметить, что в данной книге, реализующей концепцию гибридного программирования, линейные конструкции языка Go используются как заполнение графоэлементов визуального алгоритмического языка DRAKON. Другие конструкции, в частности, циклы в диапазоне или оператор Select отличается от Go-конструкций. Более того, некоторые конструкции языка DRAKON в ходе автоматической генерации программного кода кода преобразуются в другие конструкции. В частности, программная реализация оператора Select преобразуется в составной оператор выбора if-else. Это связано с стремлением авторов языка DRAKON сделать генерируемый код более эффективным, повышая его быстродействие и экономя компьютерную память.