Go run

Go build

Gofmt - w

Package “name” – любая программа начинается с имени Пакета, в нем должна находится функция “name”, которая запускается при старте программы

В пакете FMT – содержит функцию вывода на экран, импортируем ее

Funk “name”(){

}

---- переменные ------

Var – объявление – “name” – “type” 1 method

Var “name” = значение (20) 2 method, компилятор определит тип автоматически

“name” := значение 3 метод, короткий, используется как возвращаемое значение из функции. Можно объявить только новую переменную

Используется camelCase стиль объявления переменной – userPass (user\_pass – under\_score – не приветствуется, может ругатся синтаксический анализатор).

“name”++ - используется поствикс, префикс (++name) – не используется

Var weight, height int = 10, 20 - объявление нескольких переменных

weight, height = 11, 21 - присваивание в существующие переменные

weight, age := 12, 22 - короткое присваивание, должна быть хотя бы одна переменная новой

int – автоматически выбранный, платформозависимый 32/64 -> int(8 / 16 / 32 / 64), unsigned or signed

float – float32, float64 (двойная точность)

bool – по умолчанию false

complex – complex64, complex128 (var c complex128 = -1.1 + 7.12i)

-----

var str string – пустая строка, по умолчанию UTF-8

var str string = “hello\n\t” – строки в двойных кавычках “ “

var str string = ` hello \n\t ` - обратные кавычки, строка без преобразования спец символов, выводится КАК ЕСТЬ.

Строки не изменяемы

Len(str) – количество используемых БАЙТ

RuneCountInString(str) – количество СИМВОЛОВ

Str[: 3] - В строках используется СРЕЗ, из примера взять от 0 до 2 байт

Конвертация в СЛАЙС байт и обратно

byteStr = []byte(str)

str = string(byteStr)

----

Var rawbin byte = ‘ \x27 ’ – одинарные кавычки ‘ ‘ , для БАЙТ

Const “name” = “value” - объявить константу

Const( - объявить блок констант

“name” = “value”

“name1” = “value1”

)

Const(

“name” = iota - iota это автоинкремент для констант

\_ - пустая переменная, пропуск iota

“name1” - =3

)

Const(

Year = 2019 - нетипизированная константа, может быть подставлена ПОТОМ в любой тип данных

yaerTyped int = 2019 – типизированная константа

)

Type UserId int - определить собственный тип данных

Func main(){

Var uid UserId = “value”

}

---

указатели

A := 3

B := &a

\*B = 5 - А = 5

D := new(int) – создаст переменную типа int, указатель на переменную типа int

---

Массивы

Размер массива – это чать типа данных, т.е arr[2] и arr[3] – это два разных типа данных, они не совместимы,

Размер массива задается при компиляции и не может быть изменена динамически

Var “name” [3]int – [0,0,0]

Const size = 2

Varr “name” [size \* 2] bool - [false, false, false, false]

Arr := […] int {1,2,3} - определение размера при объявлении

---

Слайс – тип данных основывающийся на массиве, у слайса есть **длина** (то количество элементов которое там уже ЕСТЬ) и **капасити** (то количество элементов, которое влезет еще без АЛЛОЦИРОВАНИЯ дополнительной памяти)

Var buf0 []int - len=0, cap=0

Var buf0 []int{} - уже инициализирован но значений нету, len=0, cap=0

Buf1 := [] int {42} - len=1, cap=1

**Make** – встроенная функция которая создает массив нужной размерности и капасити

Buf2 := make([] int, 0)

Buf2 := make([] int, 5) - len=5, cap=5

Buf2 := make([] int, 5, 10) - len=5, cap=10

**Append –** добавление элементов в массив, при увеличении размерности **run time делает, х 2**

Var buf [] int

Buf = append(Buf, 9, 10) - len=2, **cap=2**

Buf = append(Buf, 12) - len=3, **cap=4**

**Добавление другого Слайса**

otherBuf := make( [] int, 3) - [0, 0, 0]

buf = append(buf, otherBuf…) - len =6, cap=8

**len** – функция для массива сообщает сколько там элементов, **cap -** функция для массива сообщает сколько там памяти выделено

var bufLen, bufCap int = len(buf), cap(buf)

buf := [] int {1, 2, 3, 4, 5}

“разрез” := buf[1:4] - [2, 3, 4]

“разрез” := buf[:2] - [1, 2]

“разрез” := buf[2:] - [3, 4, 5]

newBuf := buf[:] - скопировать все элементы которые будут ссылатся на туже память что и buf[]

newBuf [0] = 9 - buf [9, 2, 3, 4, 5] , так как та же память

newBuf = append(newBuf, 6) – то теперь newBuf указывает на другие данные, так как создастся другая область памяти и туда скопируются значения

newBuf[0] = 1

buf [9, 2, 3, 4, 5] - не изменится

newBuf [1, 2, 3, 4, 5] - изменится

**копирование Слайса**

var emptyBuf [] int

copied := copy(emptyBuf, buf)

newBuf = make( []int, len(buf), len(buf))

copy(newBuf, buf)

ints := [] int {1, 2, 3, 4}

copy( ints[ 1:3 ], [] int {5, 6} ) - скопировать срез ints = [ 1, **5, 6,** 4]

**map** – он же Хэш таблица, он же ассоциативный массив, суть по **Ключу получить Значение**

var user map[ string (тип ключа)] string (тип данных) = map[ string ] string { - инициализация при создании

“name”: “alex”,

“lastName” : “Chan”

}

Profile := make( map[ string] string, 10) - сразу с нужной емкостью

nName := user [“middleName”] - если нет Ключа – вернет значение по умолчанию ТИПА

nName, nNameExist := user[“middleName”] – проверка на существование ключа

\_, nNameExist := user[“middleName”] – пустая переменная проверка что ключ есть

Delete(user, “lastname”) – удаление ключа

---

**УПРАВЛЯЮЩИЕ СТРУКТУРЫ**

If (условное выражение) **-** условное выражение может быть Только Булева переменная

If keyValue, keyExist := mapVal[“name”]; keyExist {

// условиие с блоком инициализации

}