**Шпаргалка по языкам**

**Python**

Фильтр пустых строк в списке строк.

list = [x for x in list if x.strip()!='']

Чтения файла по строкам

with open("/path/to/file") as f:

 for line in f:

 print(line)

Запись в файл строкой за строкой

f = open("/path/tofile", 'w')

for e in aList:

 f.write(e + "\n")

f.close()

Позиционирование строки в тексте

sentence = "this is a test, not testing."

it = re.finditer('\\btest\\b', sentence)

for match in it:

 print("match position: " + str(match.start()) +"-"+ str(match.end()))

Поиск используя регулярные выражения

m = re.search('\d+-\d+', line) # search 123-123 like strings

if m:

 current = m.group(0)

Соединение списка с указанным символом

theList = ["a","b","c"]

joinedString = ",".join(theList)

Фильтр дублируемых элементов

targetList = list(set(targetList))

Удаляем пустые значения из списка

targetList = [v for v in targetList if not v.strip()=='']

# или

targetList = filter(lambda x: len(x)>0, targetList)

Добавление списка к другому списку

anotherList.extend(aList)

Итерация словаря

for k,v in aDict.iteritems():

 print(k + v)

Есть ли строка в списке

myList = ['one', 'two', 'ten']

if 'one' in myList:

 print('Да')

**Str:**

isalpha(str): возвращает True, если строка состоит только из алфавитных символов

islower(str): возвращает True, если строка состоит только из символов в нижнем регистре

isupper(str): возвращает True, если все символы строки в верхнем регистре

isdigit(str): возвращает True, если все символы строки - цифры

isnumeric(str): возвращает True, если строка представляет собой число

startwith(str): возвращает True, если строка начинается с подстроки str

endwith(str): возвращает True, если строка заканчивается на подстроку str

lower(): переводит строку в нижний регистр

upper(): переводит строку в вехний регистр

title(): начальные символы всех слов в строке переводятся в верхний регистр

capitalize(): переводит в верхний регистр первую букву только самого первого слова строки

lstrip(): удаляет начальные пробелы из строки

rstrip(): удаляет конечные пробелы из строки

strip(): удаляет начальные и конечные пробелы из строки

ljust(width): если длина строки меньше параметра width, то справа от строки добавляются пробелы, чтобы дополнить значение width, а сама строка выравнивается по левому краю

rjust(width): если длина строки меньше параметра width, то слева от строки добавляются пробелы, чтобы дополнить значение width, а сама строка выравнивается по правому краю

center(width): если длина строки меньше параметра width, то слева и справа от строки равномерно добавляются пробелы, чтобы дополнить значение width, а сама строка выравнивается по центру

find(str[, start [, end]): возвращает индекс подстроки в строке. Если подстрока не найдена, возвращается число -1

replace(old, new[, num]): заменяет в строке одну подстроку на другую

split([delimeter[, num]]): разбивает строку на подстроки в зависимости от разделителя

join(strs): объединяет строки в одну строку, вставляя между ними определенный разделитель

**list**

Методы:

append(item): добавляет элемент item в конец списка

insert(index, item): добавляет элемент item в список по индексу index

remove(item): удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError

clear(): удаление всех элементов из списка

index(item): возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError

pop([index]): удаляет и возвращает элемент по индексу index. Если индекс не передан, то просто удаляет последний элемент.

count(item): возвращает количество вхождений элемента item в список

sort([key]): сортирует элементы. По умолчанию сортирует по возрастанию. Но с помощью параметра key мы можем передать функцию сортировки.

reverse(): расставляет все элементы в списке в обратном порядке

Функций:

len(list): возвращает длину списка

sorted(list, [key]): возвращает отсортированный список

min(list): возвращает наименьший элемент списка

max(list): возвращает наибольший элемент списка

**C#**

Арифметические операции

static void Main(string[] args)

{

 int a = 5;

 int b = 2;

 int result = a + b;

 System.Console.WriteLine("Сложение, a + b = " + result);

 result = a \* b;

 System.Console.WriteLine("Умножение, a \* b = " + result);

 result = a / b;

 System.Console.WriteLine("Деление, a / b = " + result + " a и b - целые числа, деление только нацело");

 double resultDouble = a / b;

 System.Console.WriteLine("Деление, a / b = " + resultDouble + " все равно что-то не так...");

 double aDouble = 5;

 resultDouble = aDouble / b;

 System.Console.WriteLine("Деление, a / b = " + resultDouble);

 int reminder5 = counter % 5; // остаток

 System.Console.ReadLine();

}

Логика

boolVariable = 2 < 4;

if (boolVariable)

{

 System.Console.WriteLine("boolVariable = 2 < 4; Истина!");

}

else

{

 System.Console.WriteLine("boolVariable = 2 < 4; Ложь!");

}

if (10 != 100)

{

 System.Console.WriteLine("10 != 100! Ваш капитан очевидность!");

}

boolVar1 = true;

boolVar2 = false;

if (boolVar1 && boolVar2)

{

 System.Console.WriteLine("Вернуло true логическое И для true и false");

}

if (boolVar1 || boolVar2)

{

 System.Console.WriteLine("Вернуло true логическое ИЛИ для true и false");

}

Циклы

 int counter = 1;

**while** (counter <= 7)

{

 System.Console.WriteLine("Значение счетчика: " + counter);

 counter++;

}

**for** (int counter = 1; counter <= 7; counter++)

{

 System.Console.WriteLine("Значение счетчика: " + counter);

}

System.Console.ReadLine();

**foreach** (Person currPerson in personsList)

{

 // пропускаем Пушкина и переходим к гончаровой

 if (currPerson.Surname == "Пушкин")

 {

 continue;

 }

 Console.WriteLine(currPerson.FioInitials);

}

Функции и класс, наследования

public static int Multiply(int a, int b)

{ return a \* b; }

public class Cargo

{

 private double \_mass = 0;

 public double Mass

 { get { return \_mass; } }

 public Cargo(double mass)

 { \_mass = mass; }

}

// класс человека

public class Person : Cargo

{ // три строковых переменные-свойства, доступные извне класса - public

 private string \_name = "";

 public string Name

 { get { return \_name; }

 set { \_name = value; } }

 private string \_surname = "";

 public string Surname

 { get { return \_surname; }

 set { \_surname = value; } }

 private string \_otchestvo = "";

 public string Otchestvo

 { get { return \_otchestvo; }

 set { \_otchestvo = value; } }

 public string Fio

 { get {

 string fio = Surname + " " + Name + " " + Otchestvo;

 return fio; }

 }

 // конструктор, специальная функция, которая вызывается при создани экземпляра класса с помощью слова new

 public Person(string surname, string name, string otchestvo, double mass)

 : base(mass)

 { Name = name;

 Surname = surname;

 Otchestvo = otchestvo; }

}

public class Cat : Cargo{

 public string Name = "";

 public Person Master = null;

 public Cat(string name, Person master, double mass)

 : base(mass)

 { Name = name;

 Master = master; }

}

Ошибки

static void Main(string[] args)

{

 List<Person> personsList = new List<Person>();

 personsList.Add(new Person("Пушкин", "Александр", "Сергеевич"));

 personsList.Add(new Person("Гончарова", "Наталья", "Николаевна"));

 for (int counter = 1; counter <= 2; counter++)

 { try

 {

 Console.WriteLine("Код ДО операции, в которой может произойти ошибка");

 Console.WriteLine(personsList[counter].Fio);

 // в случае возникновении ошибки выполнение прервется на этом месте и перейдет в блок catch

 Console.WriteLine("Код ПОСЛЕ операции, в которой может произойти ошибка");

 }

 catch (Exception error)

 { Console.WriteLine("Произошла ошибка! Информация об ошибке " + error.Message); }

 finally {

 Console.WriteLine("Код внутри необязательного блока finally выполнится вне зависимости от того, произошла ошибка или нет");

 Console.WriteLine(); }

 }

 Console.ReadLine();

//Добавим в класс Cat функцию

public override string ToString()

{

 return Name + ", вес " + Mass + " кг., хозяин " + Master.Fio;

}

**List<T>**

**Свойства**

Capacity Возвращает или задает общее число элементов, которые может вместить внутренняя структура данных без изменения размера.

Count Получает число элементов, содержащихся в интерфейсе List<T>.

Item[Int32] Возвращает или задает элемент по указанному индексу.

**Методы**

Add(T) Добавляет объект в конец очереди List<T>.

AddRange(IEnumerable<T>) Добавляет элементы указанной коллекции в конец списка List<T>.

AsReadOnly() Возвращает для текущей коллекции оболочку ReadOnlyCollection<T>, доступную только для чтения.

BinarySearch(Int32, Int32, T, IComparer<T>) Выполняет поиск элемента в диапазоне элементов отсортированного списка List<T>, используя указанную функцию сравнения, и возвращает индекс элемента, отсчитываемый от нуля.

BinarySearch(T) Выполняет поиск элемента по всему отсортированному списку List<T>, используя компаратор по умолчанию, и возвращает индекс элемента, отсчитываемый от нуля.

BinarySearch(T, IComparer<T>) Выполняет поиск элемента по всему отсортированному списку List<T>, используя указанный компаратор, и возвращает индекс элемента, отсчитываемый от нуля.

Clear() Удаляет из коллекции List<T> все элементы.

Contains(T) Определяет, входит ли элемент в коллекцию List<T>.

ConvertAll<TOutput>(Converter<T,TOutput>) Преобразует элементы текущего списка List<T> в другой тип и возвращает список преобразованных элементов.

CopyTo(Int32, T[], Int32, Int32) Копирует диапазон элементов из списка List<T> в совместимый одномерный массив, начиная с указанного индекса конечного массива.

CopyTo(T[]) Копирует весь список List<T> в совместимый одномерный массив, начиная с первого элемента целевого массива.

CopyTo(T[], Int32) Копирует List<T> целиком в совместимый одномерный массив, начиная с указанного индекса конечного массива.

Equals(Object) Определяет, равен ли заданный объект текущему объекту. (Inherited from Object)

Exists(Predicate<T>) Определяет, содержит ли List<T> элементы, удовлетворяющие условиям указанного предиката.

Find(Predicate<T>) Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает первое найденное вхождение в пределах всего списка List<T>.

FindAll(Predicate<T>) Извлекает все элементы, удовлетворяющие условиям указанного предиката.

FindIndex(Int32, Int32, Predicate<T>) Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает отсчитываемый от нуля индекс первого вхождения в диапазоне элементов списка List<T>, начинающемся с заданного индекса и содержащем указанное число элементов.

FindIndex(Int32, Predicate<T>) Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает отсчитываемый от нуля индекс первого вхождения в диапазоне элементов списка List<T>, начиная с заданного индекса и заканчивая последним элементом.

FindIndex(Predicate<T>) Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает отсчитываемый от нуля индекс первого найденного вхождения в пределах всего списка List<T>.

FindLast(Predicate<T>) Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает последнее найденное вхождение в пределах всего списка List<T>.

FindLastIndex(Int32, Int32, Predicate<T>) Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает отсчитываемый от нуля индекс последнего вхождения в диапазоне элементов списка List<T>, содержащем указанное число элементов и заканчивающемся элементом с заданным индексом.

FindLastIndex(Int32, Predicate<T>) Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает отсчитываемый от нуля индекс последнего вхождения в диапазоне элементов списка List<T>, начиная с первого элемента и заканчивая элементом с заданным индексом.

FindLastIndex(Predicate<T>) Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает отсчитываемый от нуля индекс последнего найденного вхождения в пределах всего списка List<T>.

ForEach(Action<T>) Выполняет указанное действие с каждым элементом списка List<T>.

GetEnumerator() Возвращает перечислитель, осуществляющий перебор элементов списка List<T>.

GetHashCode() Служит хэш-функцией по умолчанию. (Inherited from Object)

GetRange(Int32, Int32) Создает неполную копию диапазона элементов исходного списка List<T>.

GetType() Возвращает объект Type для текущего экземпляра. (Inherited from Object)

IndexOf(T) Осуществляет поиск указанного объекта и возвращает отсчитываемый от нуля индекс первого вхождения, найденного в пределах всего списка List<T>.

IndexOf(T, Int32) Осуществляет поиск указанного объекта и возвращает отсчитываемый от нуля индекс первого вхождения в диапазоне элементов списка List<T>, начиная с заданного индекса и до последнего элемента.

IndexOf(T, Int32, Int32) Выполняет поиск указанного объекта и возвращает отсчитываемый от нуля индекс первого вхождения в диапазоне элементов списка List<T>, начинающемся с заданного индекса и содержащем указанное число элементов.

Insert(Int32, T) Вставляет элемент в коллекцию List<T> по указанному индексу.

InsertRange(Int32, IEnumerable<T>) Вставляет элементы коллекции в список List<T> в позиции с указанным индексом.

LastIndexOf(T) Осуществляет поиск указанного объекта и возвращает отсчитываемый от нуля индекс последнего вхождения, найденного в пределах всего списка List<T>.

LastIndexOf(T, Int32) Осуществляет поиск указанного объекта и возвращает отсчитываемый от нуля индекс последнего вхождения в диапазоне элементов списка List<T>, начиная с первого элемента и до позиции с заданным индексом.

LastIndexOf(T, Int32, Int32) Выполняет поиск указанного объекта и возвращает отсчитываемый от нуля индекс последнего вхождения в диапазоне элементов списка List<T>, содержащем указанное число элементов и заканчивающемся в позиции с указанным индексом.

MemberwiseClone() Создает неполную копию текущего объекта Object. (Inherited from Object)

Remove(T) Удаляет первое вхождение указанного объекта из коллекции List<T>.

RemoveAll(Predicate<T>) Удаляет все элементы, удовлетворяющие условиям указанного предиката

RemoveAt(Int32) Удаляет элемент списка List<T> с указанным индексом.

RemoveRange(Int32, Int32) Удаляет диапазон элементов из списка List<T>.

Reverse() Изменяет порядок элементов во всем списке List<T> на обратный.

Reverse(Int32, Int32) Изменяет порядок элементов в указанном диапазоне.

Sort() Сортирует элементы во всем списке List<T> с помощью функции сравнения по умолчанию.

Sort(Comparison<T>) Сортирует элементы во всем списке List<T> с использованием указанного Comparison<T>.

Sort(IComparer<T>) Сортирует элементы во всем списке List<T> с помощью указанной функции сравнения.

Sort(Int32, Int32, IComparer<T>) Сортирует элементы в диапазоне элементов списка List<T> с помощью указанной функции сравнения.

ToArray() Копирует элементы списка List<T> в новый массив.

ToString() Возвращает строку, представляющую текущий объект. (Inherited from Object)

TrimExcess() Задает емкость, равную фактическому числу элементов в списке List<T>, если это число меньше порогового значения.

TrueForAll(Predicate<T>) Определяет, все ли элементы списка List<T> удовлетворяют условиям указанного предиката.

**JAVA**

Простейшая программа на Java

public class FirstSample

{ public static void main(String[] args)

 { System.out.println("Hello"); }

}

Типы данных

Целочисленные: int — 4 байта

short — 2 байта

long — 8 байт

byte — 1 байт

С плавающей точкой: float — 4 байта

double — 8 байт

POSITIVE\_INFINITY

NEGATIVE\_INFINITY

NaN (not a number)

Символы

char — кодовая единица в кодировке UTF-16

Логический

boolean — true/false, преобразование в целочисленный тип и наоборот невозможно.

(Переменную рекомендуется объявлять, как можно ближе к тому месту кода, где предполагается ее использовать.)

Константы

Для обозначения констант используется ключевое слово final. В названиях констант желательно использовать только прописные буквы. Константы, доступные только нескольким методам класса называются константами класса. Константы класса объявляются как static final.

Операции

/ — целочисленное деление если оба аргумента являются целыми числами. В противном случае — деление чисел с плавающей запятой.

% — остаток от деления целых чисел.

++/— — инкремент/декремент имеет префиксную и постфиксную форму. Различие проявляется только в выражениях. В префиксной форме сначала изменяется значение переменной, и для дальнейших вычислений уже используется новое значение, а в постфиксной форме используется старое значение этой переменной и лишь после данной операции оно изменяется.

Тернарная операция условия

условие ? выражение\_1 : выражение\_2 — если условие истинно, то вычисляется первое выражение, а если ложно — второе выражение.

Поразрядные операции: & — И

| — ИЛИ

^ — исключающее ИЛИ

~ — НЕ

<</>> — сдвиг ВЛЕВО/ВПРАВО (>> — заполняет старший бит нулями (>> — восстанавливает в старших разрядах знаковый бит))

Перечислимый тип

enum Size { SMALL, MEDIUM, LARGE, EXTRA\_LARGE };

Size s = Size.MEDIUM;

Символьные строки

В JAVA отсутствует встроенный тип для символьных строк. Вместо этого в стандартной библиотеке содержится класс String. Каждая символьная строка, заключенная в кавычки, представляет собой экземпляр класса String. Объекты типа String являются неизменяемыми, т.е. постоянными.

Для проверки символьных строк на равенство нельзя применять операцию ==. Она лишь определяет, хранятся ли обе строки в одной и той же области памяти.

Чтение вводимых данных

Scanner in = new Scanner(System.in);

String name = in.nextLine();

int age = in.netxInt();

Управляющие конструкции

**if** (a > b) { оператор; }

else if (a < c) { оператор; }

else { оператор; }

**while** (условие) оператор

**do** оператор **while** (условие);

**for** (int i = 1; i <= 10; i++) { … }

// В качестве метки ветви case может быть char, byte, short, int, классы-оболочки Character, Bte, Short, Integer, перечисляемые константы, строковые литералы.

switch (choice) {

case 1:

....

break;

case 2:

.....

break;

default:

....

break; }

В отличие от C++ перегрузка операций в JAVA **не** поддерживается.

for (переменная : коллекция) оператор; // При выполнении этого цикла его переменной

// последовательно присваивается каждый элемент коллекции, после чего выполняется оператор

// или блок. В качестве коллекции может использоваться массив или экземпляр класса,

// реализующего интерфейс Iterable, например ArrayList.

Копирование массивов

int[] destinationArray = sourceArray;

destinationArray[6] = 13; // sourceArray[6] теперь тоже равен 13

// Переменные sourceArray и destinationArray ссылаются на один и тот же массив

Если необходимо скопировать все элементы одного массива в другой, то для этого необходимо вызвать метод copyTo() из класса Arrays.

int destinationArray[] = Arrays.copyOf(sourceArray, sourceArray.length \* 2);

// Первая половина элементов заполняется значениями sourceArray

// Оставшиеся элементы заполняются нулями

Параметры командной строки

java paramExamples -x arg1 arg2

args[0] = «-x»

args[1] = «arg1»

В отличие от C++ при запуске программы на Java ее имя не сохраняется в массиве args.

Сортировка массивов

Сортировка массивов осуществляется путем путем вызова метода sort() из класса Arrays.

**List:**

void add(int index, E obj): добавляет в список по индексу index объект obj

boolean addAll(int index, Collection<? extends E> col): добавляет в список по индексу index все элементы коллекции col. Если в результате добавления список был изменен, то возвращается true, иначе возвращается false

E get(int index): возвращает объект из списка по индексу index

int indexOf(Object obj): возвращает индекс первого вхождения объекта obj в список. Если объект не найден, то возвращается -1

int lastIndexOf(Object obj): возвращает индекс последнего вхождения объекта obj в список. Если объект не найден, то возвращается -1

ListIterator<E> listIterator (): возвращает объект ListIterator для обхода элементов списка

static <E> List<E> of(элементы): создает из набора элементов объект List

E remove(int index): удаляет объект из списка по индексу index, возвращая при этом удаленный объект

E set(int index, E obj): присваивает значение объекта obj элементу, который находится по индексу index

void sort(Comparator<? super E> comp): сортирует список с помощью компаратора comp

List<E> subList(int start, int end): получает набор элементов, которые находятся в списке между индексами start и end

**Map**:

void clear(): очищает коллекцию

boolean containsKey(Object k): возвращает true, если коллекция содержит ключ k

boolean containsValue(Object v): возвращает true, если коллекция содержит значение v

Set<Map.Entry<K, V>> entrySet(): возвращает набор элементов коллекции. Все элементы редставляют объект Map.Entry

boolean equals(Object obj): возвращает true, если коллекция идентична коллекции, передаваемой через параметр obj

boolean isEmpty: возвращает true, если коллекция пуста

V get(Object k): возвращает значение объекта, ключ которого равен k. Если такого элемента не окажется, то возвращается значение null

V getOrDefault(Object k, V defaultValue): возвращает значение объекта, ключ которого равен k. Если такого элемента не окажется, то возвращается значение defaultVlue

V put(K k, V v): помещает в коллекцию новый объект с ключом k и значением v. Если в коллекции уже есть объект с подобным ключом, то он перезаписывается. После добавления возвращает предыдущее значение для ключа k, если он уже был в коллекции. Если же ключа еще не было в коллекции, то возвращается значение null

V putIfAbsent(K k, V v): помещает в коллекцию новый объект с ключом k и значением v, если в коллекции еще нет элемента с подобным ключом.

Set<K> keySet(): возвращает набор всех ключей отображения

Collection<V> values(): возвращает набор всех значений отображения

void putAll(Map<? extends K, ? extends V> map): добавляет в коллекцию все объекты из отображения map

V remove(Object k): удаляет объект с ключом k

int size(): возвращает количество элементов коллекции

**GO**

Переменные

 var v1 int в С++ int v1;

 var v2 string ====> const std::string v2; (примерно)

 var v3 [10]int int v3[10];

 var v4 []int int\* v4; (примерно)

 var v5 struct { f int } struct { int f; } v5;

 var v6 \*int int\* v6; (но нет арифметики для указателей)

 var v7 map[string]int std::unordered\_map\* v7; (примерно)

 var v8 func(a int) int int (\*v8)(int a);

var v1 int = 100

var v2 string = "Hello!"

var v3 [10]int = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

var v4 []int = {1000, 2000, 12334}

var v5 struct { f int } = { 50 }

var v6 \*int = \*v1

var v7 map[string]int = {"one":1, "two":2, "three":3}

var v8 func(a int) int = func(a int) int { return a+1 }

срезы:

Тип []T — срез с элементами типа T.

a[0:5] — срез 5 элементов массива a.

var numbers []int /\* срез неопределённого размера \*/

/\* numbers = []int{0,0,0,0,0} \*/

numbers = make([]int,5,5) /\* срез длиной и ёмкостью равной 5 \*/

 primes := [6]int{2, 3, 5, 7, 11, 13}

 var s []int = primes[1:4] // [3 5 7]

Присваивание

 a = b // Присвоить переменной a значение b

 v1 := v2 // аналог var v1 = v2 (создать переменную и присвоить значение)

a := 10 // Объявление и инициализация целой переменной a.

b := 20 // Объявление и инициализация целой переменной b.

 ...

a := b // ОШИБКА! Попытка повторного определения a.

i, j = j, i // Поменять местами значения i и j.

Функции

func f(a, b int) (int, string) { return a+b, "сложение" }

package main

import "fmt"

func add(a int, b int) int { return a + b }

func main() {

 fmt.Println("Сумма равна ", add(10, 19))

}

func incTwo(a, b int) (c, d int) {

 c = a+1

 d = b+1

 return

}

first, second := incTwo(1, 2) // first = 2, second = 3

first := incTwo(1, 2) // НЕВЕРНО - нет переменной, которой присваивается второй результат

first, \_ := incTwo(1, 2) // ВЕРНО, второй результат не используется

Условия

true — выполняется всегда;

a < 10 — выполняется, когда a меньше 10;

(a < b) || (a < c) — выполняется, когда a меньше b или a меньше c;

(a < b) && (a < c) — выполняется, когда a меньше b и a меньше c.

 **if** i >=0 && i < len(arr) {

 println(arr[i])

 }

...

 **for** i := 0; i < 10; i++ { … }

**switch** value {

case 1:

 fmt.Println("One")

 fallthrough // Далее будет выполнена ветвь "case 0:"

case 0:

 fmt.Println("Zero")

}

**switch** chars[code].category {

case "Lu", "Ll", "Lt", "Lm", "Lo":

 ...

case "Nd":

 ...

default:

 ...

}

Структуры:

type Vertex struct {

 X int

 Y int }

func main() {

 v := Vertex{1, 2}

 v.X = 4

 fmt.Println(v.X) // 4

}

type myType struct { i int }

// Здесь p - получатель в методах типа myType.

func (p \*myType) get() int { return p.i }

func (p \*myType) set(i int) { p.i = i }

Каналы

Go дает возможность создать новый поток выполнения программы с помощью ключевого слова **go**, которое запускает анонимную или именованную функцию в заново созданной go-процедуре (термин, используемый в Go для обозначения сопрограмм). Все go-процедуры в рамках одного процесса используют общее адресное пространство, выполняясь над ОС-потоками, но без жёсткой привязки к последним, что позволяет выполняющейся go-процедуре покидать поток с заблокированной go-процедурой (ждущей, например, отправки или приема сообщения из канала) и продолжать работу далее.

in := make(chan string, 0) // Создание небуферизованного канала in

out := make(chan int, 10) // Создание буферизованного канала out

...

in <- arg // запись значения в канал in

r1 := <- out // чтение из канала out

r2, ok := <- out // чтение с проверкой закрытия канала

if ok { // если ok == true - канал открыт

 ...

} else { // если канал закрыт, делаем что-то ещё

 ...

}

// Функция запускает параллельное чтение из входного канала in целых чисел и запись

// в выходной канал только тех из них, которые положительны.

// Возвращает выходной канал.

func positives(in <-chan int64) <-chan int64 {

 out := make(chan int64)

 go func() { // Цикл далее будет выполняться, пока канал in не закрыт

 for next := range in {

 if next > 0 { out <- next }

 }

 close(out)

 }()

 return out

}

**Pascal**

Операторы цикла

Оператор цикла с управляющей переменной - форма "to"

Пример

for имя:=значение1 to значение2 do for i:=1 to n do

оператор begin read(a); s:=s+a end

Оператор цикла с управляющей переменной - форма "downto"

Пример

for имя:=значение1 downto значение2 do for i:=n downto 0 do

оператор write(i)

Оператор цикла с логическим пред-условием

Пример

while условие-повторения do while i<=n do

оператор begin read(a); s:=s+a; i:=i+1 end

Оператор цикла с логическим пост-условием

Пример

Repeat repeat

операторы write(i); i:=i-1

until условие-выхода until i=0

Типы и записи

Описание типа записи Пример

имя типа = record type data = record

имя-1: тип-1; x:integer;

имя-2: тип-2; y:integer;

имя-N: тип-N; t:string;

end; end;

 var a1, a2, a3: data;

Объекты и классы

Описание типа объекта (класс) Пример

имя класса = object (класс-родитель) type donkey = object (animals)

свойство-1: тип-1; x:integer;

свойство-2: тип-2; y:integer;

свойство-N: тип-N; t:string;

метод-1(параметры); function run(m:string):boolean;

метод-2(параметры); procedure show;

метод-N(параметры); procedure hide;

end; end;

 var a1, a2, a3: donkey;

**Примеры программ:**

Подсчет различных букв в слове

var s:string;

 r:real;

 i,j,n:integer;

begin

 r:=0;

 readln(s);

 for i:=1 to length(s) do begin

 n:=0;

 for j:=1 to length(s) do begin

 if s[i]=s[j] then inc(n);

 end;

 r:=r+1/n;

 end;

 writeln('количество различных букв = ', r:1:0);

end.

Перестановка букв в слове (циклический сдвиг вправо)

var s:string;

 i,j,n:integer;

begin

 readln(s);

 s:=s[length(s)] + copy(s,1,length(s)-1);

 writeln(s);

end.

Поиск минимального элемента в массиве?

var a:array[1..10] of integer;

 min:integer;

 i:integer;

begin

 writeln('введите 10 элементов массива');

 min:=MAXINT;

 for i:=1 to 10 do begin

 readln( a[i] );

 if min>a[i] then min:=a[i];

 end;

 writeln( 'Максимальный элемент массива = ', min );

end.

Сортировка массива "пузырьком" по возрастанию

const n = 10; { количество элементов в массиве }

var a:array[1..n] of integer;

 i,j,x:integer;

begin

 writeln('введите ',n,' элементов массива');

 for i:=1 to n do readln( a[i] );

 for i:=1 to n-1 do begin

 for j:=i+1 to n do begin

 if a[i]>a[j] then begin

 x:=a[i]; a[i]:=a[j]; a[j]:=x;

 end;

 end;

 end;

 writeln('после сортировки:');

 for i:=1 to n do writeln( a[i] );

end.

Решение уравнения: A\*x^2 + B\*x + C = 0

var a,b,c,d,x:real;

begin

 writeln('введите A,B,C');

 readln( a,b,c );

 d:=sqr(b)-4\*a\*c;

 if d<0 then begin

 writeln('действительных корней нет');

 end else if d=0 then begin

 x:=(-b)/2\*a;

 writeln('корень уравнения: ',x);

 end else begin

 x:=(-b+sqrt(d))/2\*a;

 writeln('1-й корень уравнения: ',x);

 x:=(-b-sqrt(d))/2\*a;

 writeln('2-й корень уравнения: ',x);

 end

end.

Перевод двоичного числа в десятичное

var a : string;

function BIN\_DEC(x:string):longint;

const digits:array [0..1] of char = ('0','1');

var res,ves:longint; i,j:byte;

begin

 res:=0; ves:=1;

 for i:=length(x) downto 1 do begin

 j:=0;

 while (digits[j]<>x[i]) do inc(j);

 res:=res+ves\*j;

 ves:=ves\*2;

 end;

 BIN\_DEC:=res;

end;

begin { основная программа }

 readln( a );

 writeln( BIN\_DEC(a) );

end.

Перевод в маленькие буквы (нижний регистр)

var s:string;

function SmallAlpha(ps:string):string;

var i:integer;

begin

 for i:=1 to length(ps) do begin

 case ps[i] of

 'A'..'Z','А'..'П': inc(ps[i],32);

 'Р'..'Я' : inc(ps[i],80);

 end;

 end;

 SmallAlpha:=ps;

end;

procedure write\_results; { Запись результатов в файл }

var i : integer;

begin

 assign(output,'TOWN.OUT'); rewrite(output);

 writeln(ml);

 if (ml>0) then begin

 for i:=1 to ml do writeln(store[i]);

 end;

 close(output);

end;

begin

 writeln('Введите любой текст'); readln(s);

 writeln('Этот же текст маленькими буквами:');

 writeln(SmallAlpha(s));

end.

**Swfit**

let maximumNumberOfLoginAttempts = 10 // констант

var currentLoginAttempt = 0 // переменная

var x = 0.0, y = 0.0, z = 0.0

var welcomeMessage: String // переменная может хранить String

welcomeMessage = "Hello"

var red, green, blue: Double

var friendlyWelcome = “Hello!”

friendlyWelcome = “Bonjour!”

print(friendlyWelcome) // “Bonjour!”

Границы целыйх чисел

let minValue = UInt8.min // minValue равен 0, а его тип UInt8

let maxValue = UInt8.max // maxValue равен 255, а его тип UInt8

Преобразование типов

let three = 3

let pointOneFourOneFiveNine = 0.14159

let pi = Double(three) + pointOneFourOneFiveNine // pi равно 3.14159, и для него выведен тип Double

let integerPi = Int(pi) // integerPi равен 3, и для него выведен тип Int

Логические типы

let orangesAreOrange = true

let turnipsAreDelicious = false

if turnipsAreDelicious {

 print("Mmm, tasty turnips!")

} else {

 print("Eww, turnips are horrible.") // Выведет "Eww, turnips are horrible."

}

Кортежи

let http404Error = (404, "Not Found")

// http404Error имеет тип (Int, String), и равен (404, "Not Found")

let http200Status = (statusCode: 200, description: "OK") //присвоение имени при объявлении

print("The status code is \(http200Status.statusCode)")

// Выведет "The status code is 200"

print("The status message is \(http200Status.description)")

// Выведет "The status message is OK"

Логика

 == - равенство

 != - неравенство

 **if** let actualNumber = Int(possibleNumber) {

 print("\(possibleNumber) has an integer value of \(actualNumber)")

} **else** { // } else if age > 0 {

 print("\(possibleNumber) could not be converted to an integer")

}

// Выведет "123" has an integer value of 123

Ошибки

 func canThrowAnError() throws {

// эта функция может сгенерировать ошибку

}

do {

 try canThrowAnError()

 // ошибка не была сгенерирована

} catch {

 // ошибка сгенерирована

}