Стандарты оформления кода. Основы алгоритмизации.

Определения

- **Стандарт кодирования** — набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования. Наличие общего стиля программирования облегчает понимание и поддержание исходного кода, написанного более чем одним программистом, а также упрощает взаимодействие нескольких человек при разработке программного обеспечения.

Общие рекомендации

- **Правила именования:**
  - имена переменных принято записывать в смешанном регистре, начиная с нижнего (примеры: fileName, currentPoint);
  - именованные константы должны быть записаны в верхнем регистре с использованием подчеркивания (MAX_ITERATIONS, COLOR_RED, PI);
  - названия методов и функций должны быть глаголами, быть записанными в смешанном регистре и начинаться с нижнего (getName(), computeTotalWidth());

– все имена следует записывать, используя слова английского языка (fileName, а не imyaFayla);
– переменные, имеющие большую область видимости, следует называть длинными именами, имеющие небольшую область видимости — короткими. Имена временных переменных, использующихся для хранения временных значений или индексов, лучше всего делать короткими: i, j, k, l, m..

- множественное число следует использовать для представления массивов (коллекций) объектов (int values[10]);
- префикс **n** следует использовать для представления числа объектов (nLines, nPoints);
- переменным-итераторам следует давать имена i, j, k и т. д.;
- префикс is следует использовать только для булевых (логических) переменных и методов (isOpen, isSet);

– симметричные имена должны использоваться для соответствующих операций (min/max, add/remove);
– следует избегать сокращений в именах (не стоит comAv() вместо computeAverage()).

Общие рекомендации

- **Файлы:**
  - класс следует объявлять в заголовочном файле (расширение h) и определять (реализовывать) в файле исходного кода (расширение cpp), имена файлов совпадают с именем класса;
  - содержимое файлов не должно превышать 80 колонок.

Общие рекомендации

- **Выражения:**
  - переменные, относящиеся к циклу, следует инициализировать непосредственно перед ним;
  - избегайте сложных условных выражений. Вместо этого вводите булевы переменные;

Общие рекомендации

- константы с плавающей точкой следует всегда записывать, по крайней мере, с одной цифрой до десятичной точки (double total = 0.0; // НЕ РЕКОМЕНДУЕТСЯ: double total = 0);
- методы рекомендуется отделять тремя пустыми строками – это улучшает их видимость в тексте;
- переменные в объявлениях стоит выравнивать

```
AsciiFile* file;
int        nPoints;
```

Стили записи кода. 1TBS

- Этот стиль был впервые использован Кернинганом и Ричи в своей книге "The C Progamming Language". Расшифровывается как **One True Bracing Style** (единственный правильный стиль расстановки скобок). Иногда его называют **K&R** (Kernighan and Ritchie) или kernel стилем. Преимущество этого стиля – экономия вертикального пространства, но можно запутаться в скобках.

  Отступ в данном стиле равняется 8 или 4-м пробелам.

- Данный стиль использует стиль расстановки скобок при котором скобка переносится на новую строку при определении пространств имен (namespaces), классов (classes), функций, в остальных случаях скобка остается на той же строке, где располагается часть кода, к которой она принадлежит.

Стили записи кода. 1TBS

```
void function(int i)
{
        if(i==0) {
                printf(“Hello”);
        }
}
```

Стиль Алмена

- Впервые был употреблен Эриком Алменом в исходных кодах утилит для ОС BSD, поэтому иногда его называют "стиль BSD". Достаточно нагляден, но требует использования дополнительной строки. Отступы в стиле Алмена обычно (но не всегда) составляют четыре пробела. Аргументом в поддержку такого стиля является тот факт, что область видимости блочного оператора ясна и визуально ассоциируется с управляющим оператором.

Стили записи кода Алмена

```
void function(int i)
{
        if(i==0)
      {
                    printf(“Hello”);
        }
}
```

Стиль Whitesmith

- Одно время существовал C-компилятор, который назывался Whitesmith C. В его документации есть пример форматирования программного кода, который стал прототипом для этого стиля форматирования. Этот стиль имеет преимущество в том, что скобки более тесно ассоциируются с кодом, который они включают и разграничивают.

Стили записи кода Whitesmith

```
void function(int i)
      {
         if(i==0)
            {
                  printf(“Hello”);
               }
      }
```

Стили записи кода GNU

Стандарты кодирования GNU были написаны Ричардом Мэттью Столлманом и другими волонтерами проекта GNU. Последовательное структурированное расположение блоков операторов (с отступами) — отличительная черта форматирования в стиле GNU C; также как и обязательный пробел перед скобками. Во всём отформатированном коде по стилю GNU каждая закрывающая фигурная, круглая или квадратная скобки находятся на одинаковом отступе от начала экрана с соответствующими открывающими скобками.

Стили записи кода GNU

GNU является комбинацией стилей Алмена и Whitesmith

```
void function(int i)
    {
        if(i==0)
            {
                printf(“Hello”);
            }
     }
```

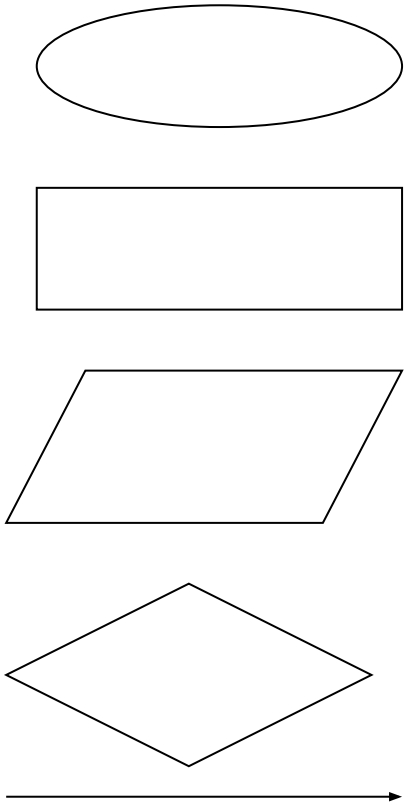
Основы алгоритмизации в блок-схемах

Блок-схемы алгоритмов

Среди универсальных форм представления или записи алгоритмов можно выделить так называемые ***блок-схемы*** алгоритмов. Блоки являются всего лишь шаблоном для описания действий в процессе решения задачи, а связи между блоками определяют последовательность этих действий.

Такая форма часто используется в профессиональной среде программистов. Она позволяет с достаточной степенью свободы описывать решения, получаемые в процессе нисходящего проектирования алгоритмов и соответствующих им программ, абстрагируясь от средств, предоставляемых конкретным языком программирования.

Блок-схемы алгоритмов



метки начала / окончания алгоритма

операторы присваивания

ввод / вывод данных

Условие

соединительная стрелка

Блок-схемы алгоритмов

Операторный блок – это прямоугольник, в который вписывается некоторое действие или выражение. Этот блок может иметь несколько входов и только один выход, что обеспечивает однозначность в определении последовательности выполняемых действий. Исключение составляют начальный и конечный блоки. Первый не имеет входов, второй – выхода.

Условный блок обозначается ромбом, в который вписывается некоторое условие. Поскольку результатом проверки условия может быть либо “да”, либо “нет” (“истина” или “ложь”, “0” или “1”), блок имеет два соответствующих этим ответам выхода.

Программа

- Любая программа всегда состоит из трех условных частей: ввод данных, обработка введенных данных и вывод результата

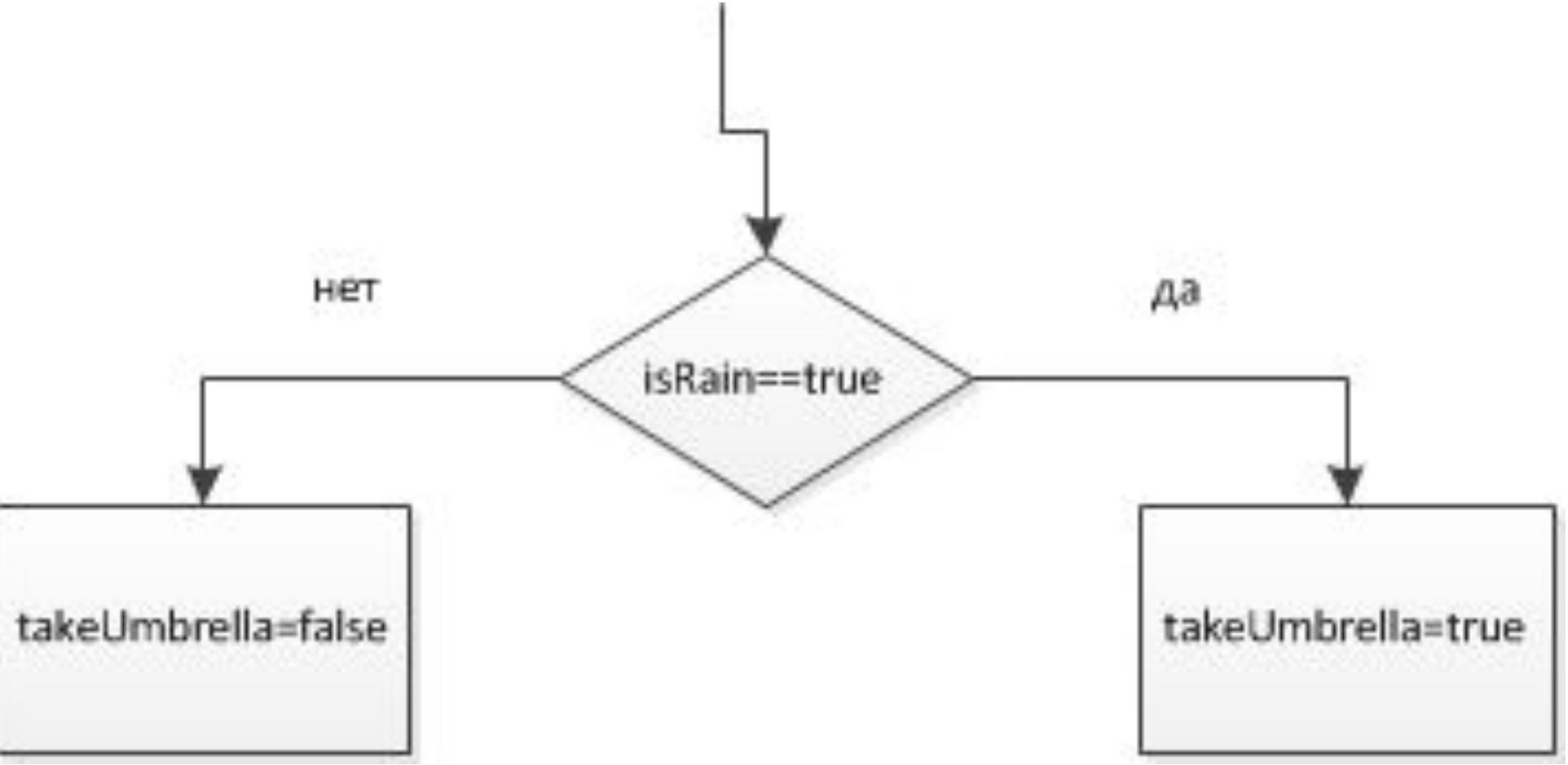
Условные операторы

- Содержат условие, которое может выполниться (true,1) или не выполниться (false,0).
- Для примера рассмотрим булевые переменные:
  - isRain=true //дождь есть
  - isRain=false //дождя нет
  - takeUmbrella=true //брать зонт
  - takeUmbrella=false //не брать зонт

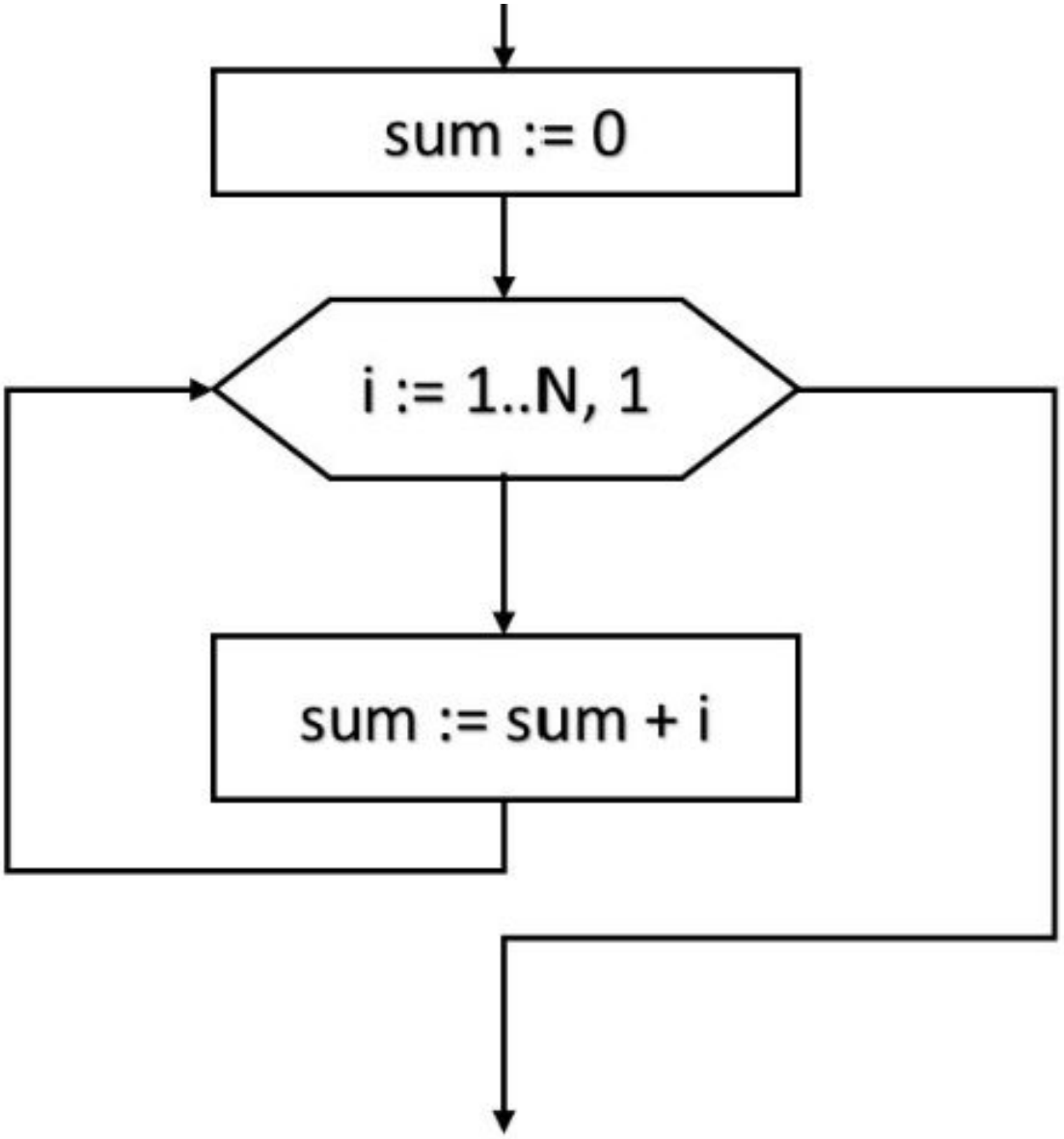
Пример условия

```
….
if (isRain==true)
      takeUmbrella=true;
else
      takeUmbrella=false;
…..
```

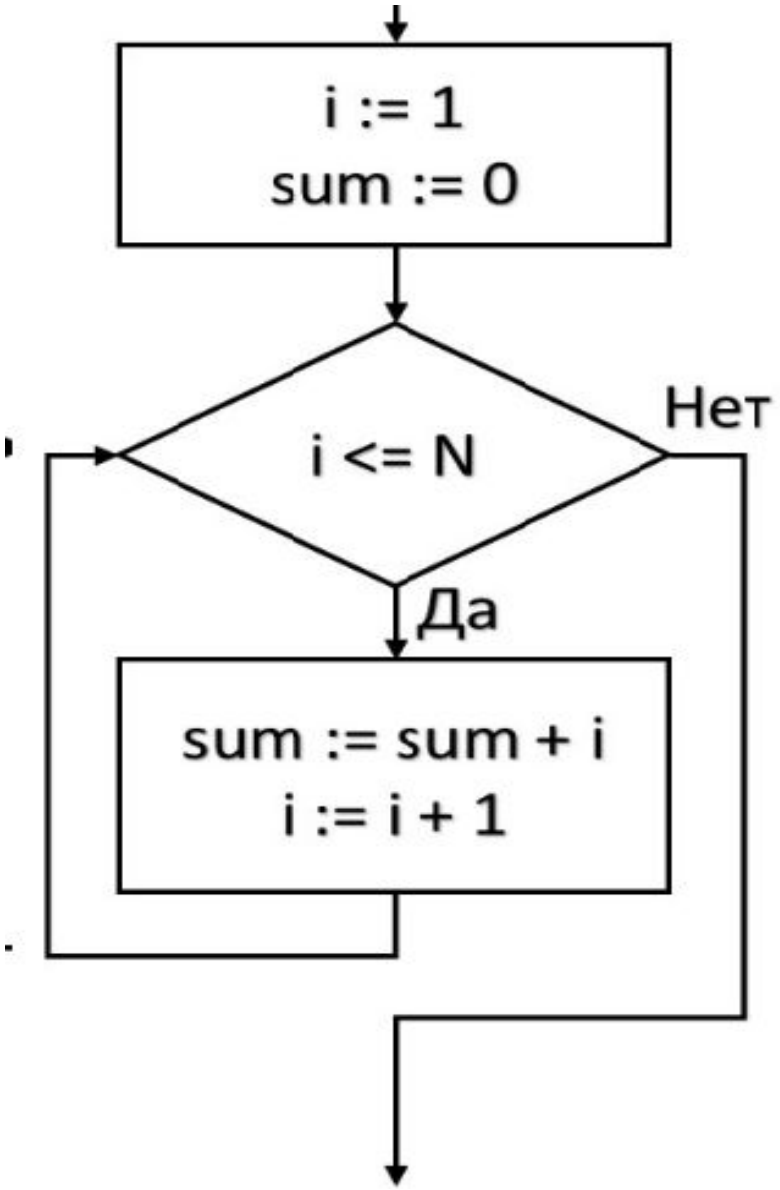
В виде блок-схемы



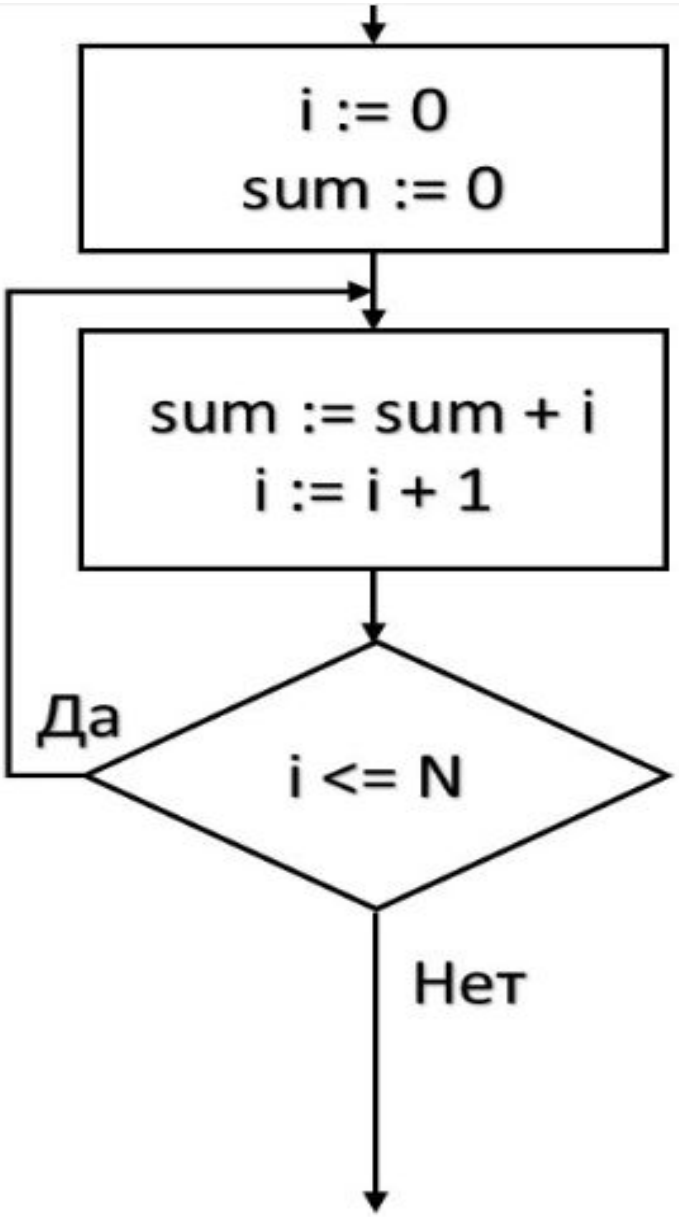
Блок-схема цикла for



Блок-схема цикла while



Блок-схема цикла Do…while



Задача 1

- Определить значение S, вычислив первые N членов последовательности. Вычисления прекратить, если разница между двумя последними членами последовательности не превышает eps=0,01.

$$S = \frac{2}{x+1} + \frac{4}{x+3} + \frac{8}{x+5} + \ldots$$

Решение

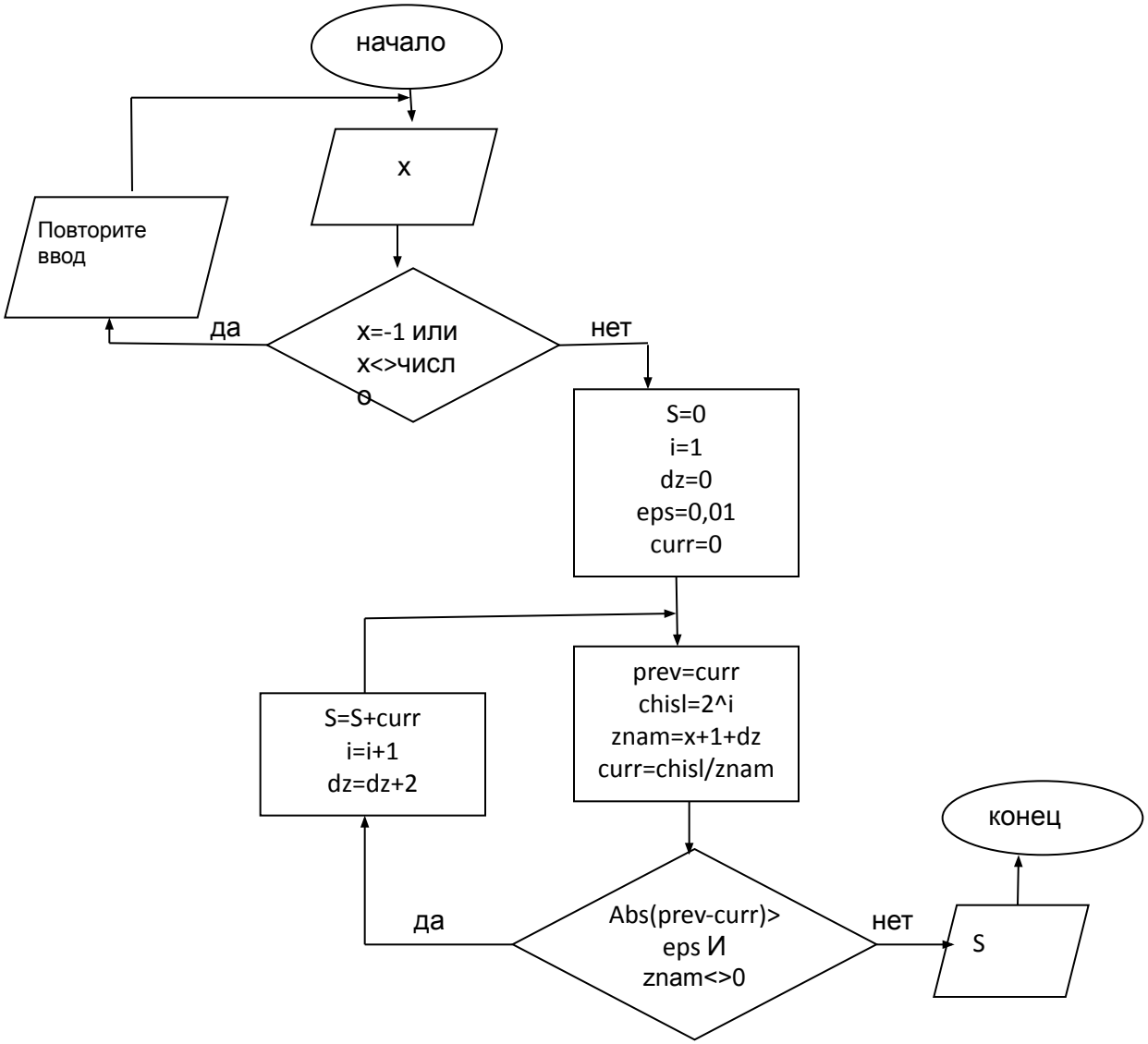
- Запишем отдельно закон вычисления числителя и знаменателя

$$chisl = 2^i, i = 1,...$$

$$znam = x + 1 + dz_i,$$

$$i = 1, dz_1 = 0;$$

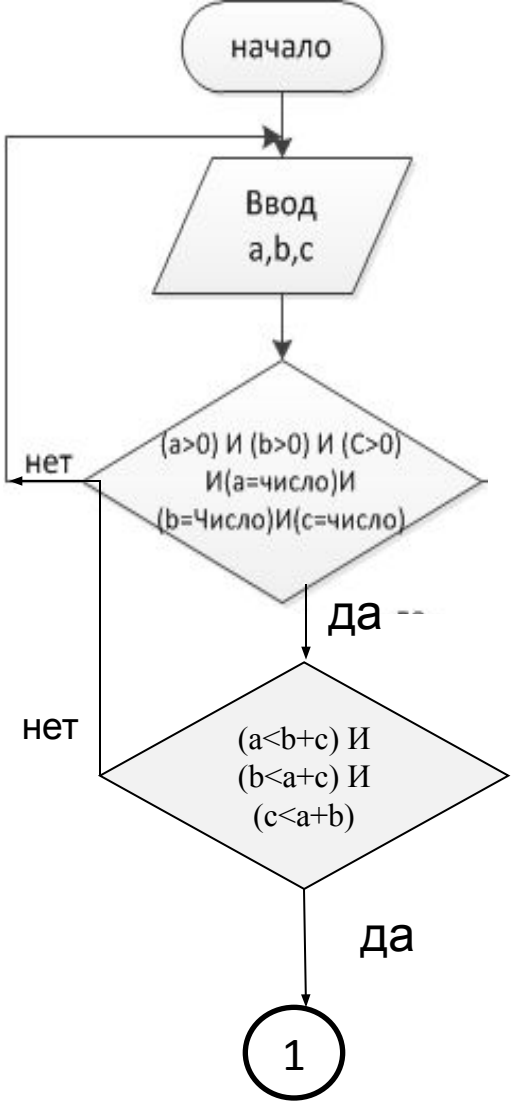
$$i = 2,..., dz_i = dz_{i-1} + 2$$

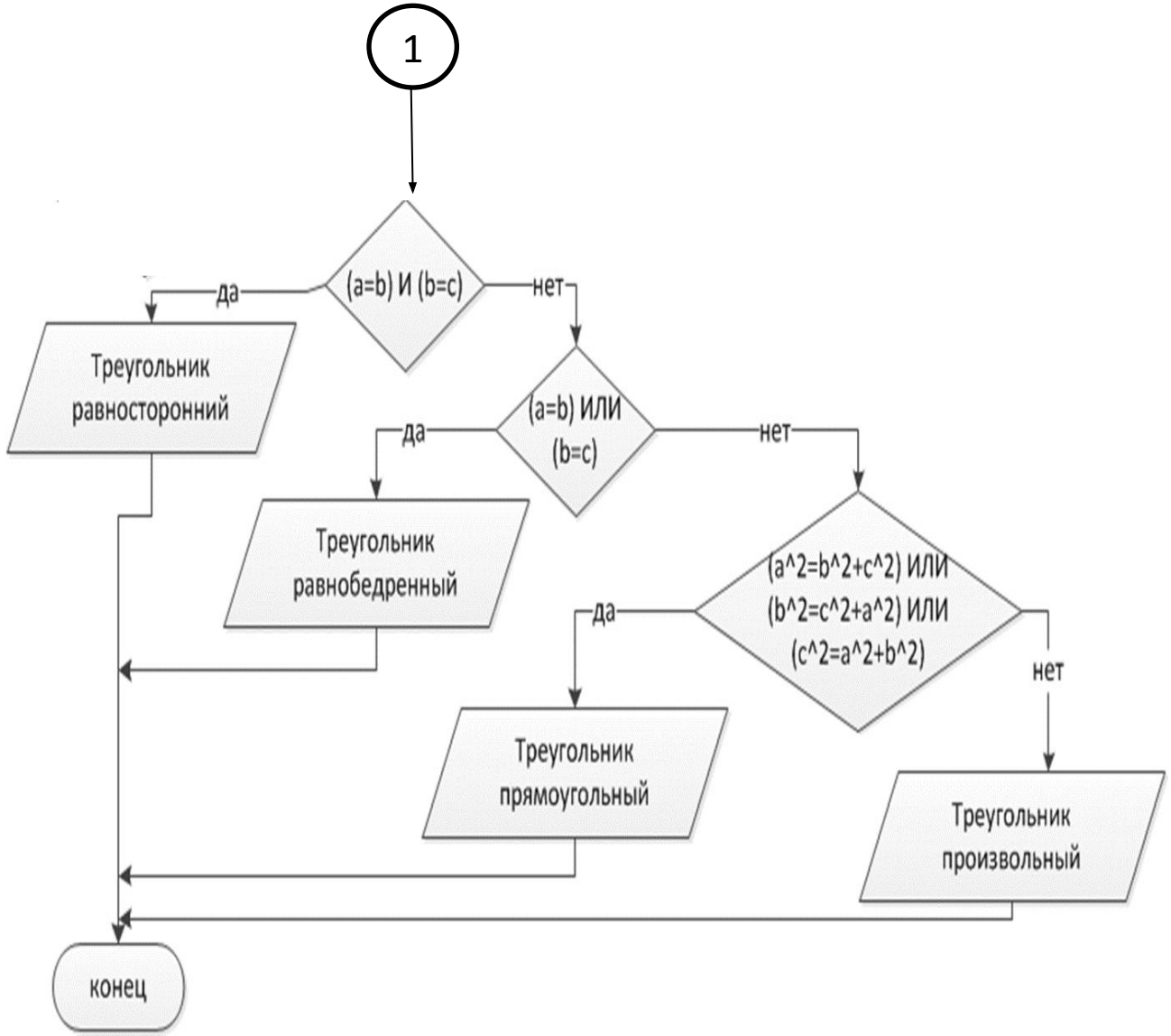

начало
x
Повторите ввод
да
x=-1 или x<>числ о
нет
S=0
i=1
dz=0
eps=0,01
curr=0
prev=curr
chisl=2^i
znam=x+1+dz
curr=chisl/znam
S=S+curr
i=i+1
dz=dz+2
да
Abs(prev-curr)> eps И znam<>0
нет
S
конец

Задача 2

- Определить тип треугольника: равносторонний, равнобедренный, прямоугольный или произвольный. Треугольник задан сторонами ***a,b,c.***

- ***Вначале при вводе необходимо проверить, получится ли из введённых данных треугольник:*** (a<b+c) И (b<a+c) И (c<a+b)


начало
Ввод
a,b,c
(a>0) И (b>0) И (C>0)
И(a=число)И
(b=Число)И(c=число)
нет
да
(a<b+c) И
(b<a+c) И
(c<a+b)
нет
да
1


1
(a=b) И (b=c)
да
нет
Треугольник
равносторонний
(a=b) ИЛИ
(b=c)
да
нет
Треугольник
равнобедренный
(a^2=b^2+c^2) ИЛИ
(b^2=c^2+a^2) ИЛИ
(c^2=a^2+b^2)
да
нет
Треугольник
прямоугольный
Треугольник
произвольный
конец

Массивы

- Массив – набор пронумерованных ячеек, каждая из которых содержит элемент.
- Массив – множество однотипных элементов, объединённых общим именем, доступ к которым осуществляется по индексу

Массивы

Массив имеет следующие характеристики:

Имя – название массива;

Индекс – номер элемента в массиве;

Элемент – значение в массиве;

Размер – количество элементов в массиве.

Houses[1]=-5;
…
Houses[5]=9.

Массивы

Проход по массиву в цикле.
Вводим переменную индекс массива – ***i***.

Итерация 1. ***i=1, Houses[i]=-5***
Итерация 2. ***i=2, Houses[i]=0***
…
Итерация 5. ***i=5, Houses[i]=9***

Найти в массиве первое четное число, вывести его значение

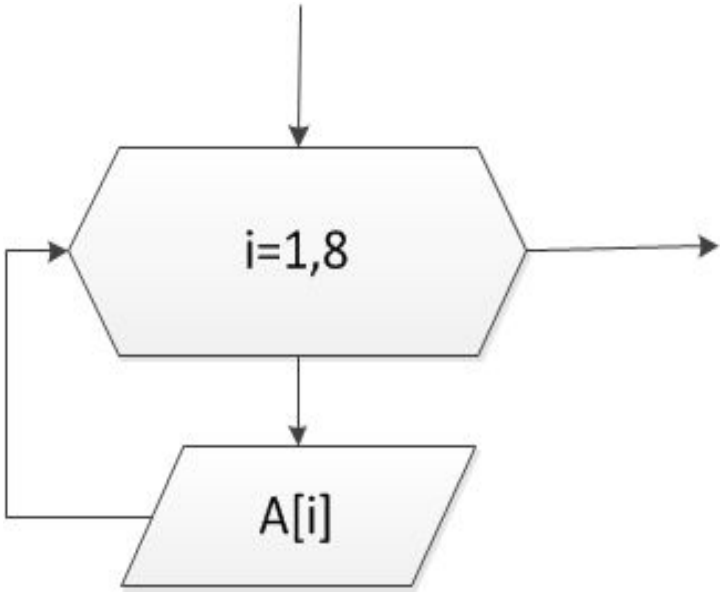
- нужен проход по массиву, пока не встретится четное число;
- нужно ввести переменную ***i***, которая будет индексом элементов массива и будет принимать все значения от 1 до 5;
- при проходе по массиву нужно проверять условие ***остаток(A[i]/2)==0.***

Массив одномерный A[1..8]- 1 строка, 8 столбцов

2	-4	5	0	0	1	2	1
1	2	3	4	5	6	7	8

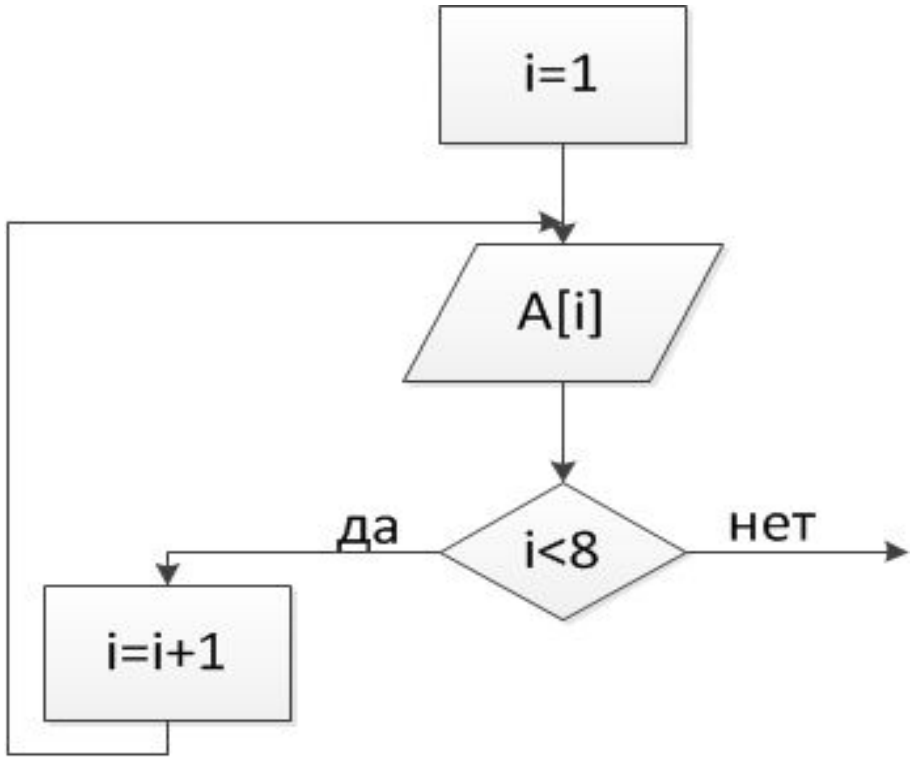
Проход по массиву – ввод элементов

- цикл

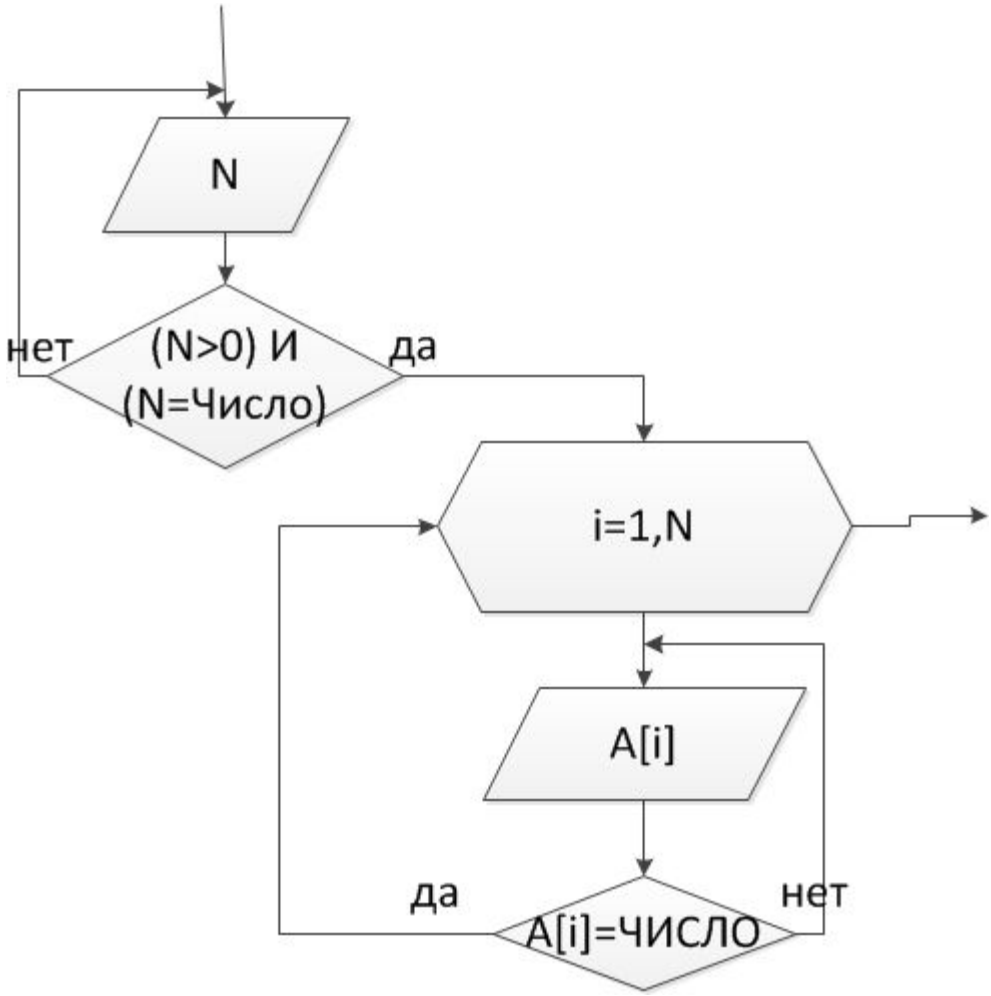


Проход по массиву – ввод элементов

- Ветвление

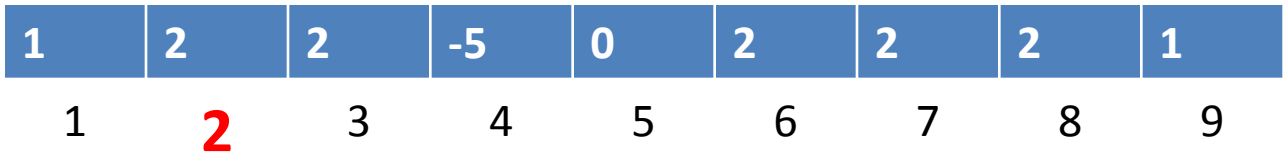


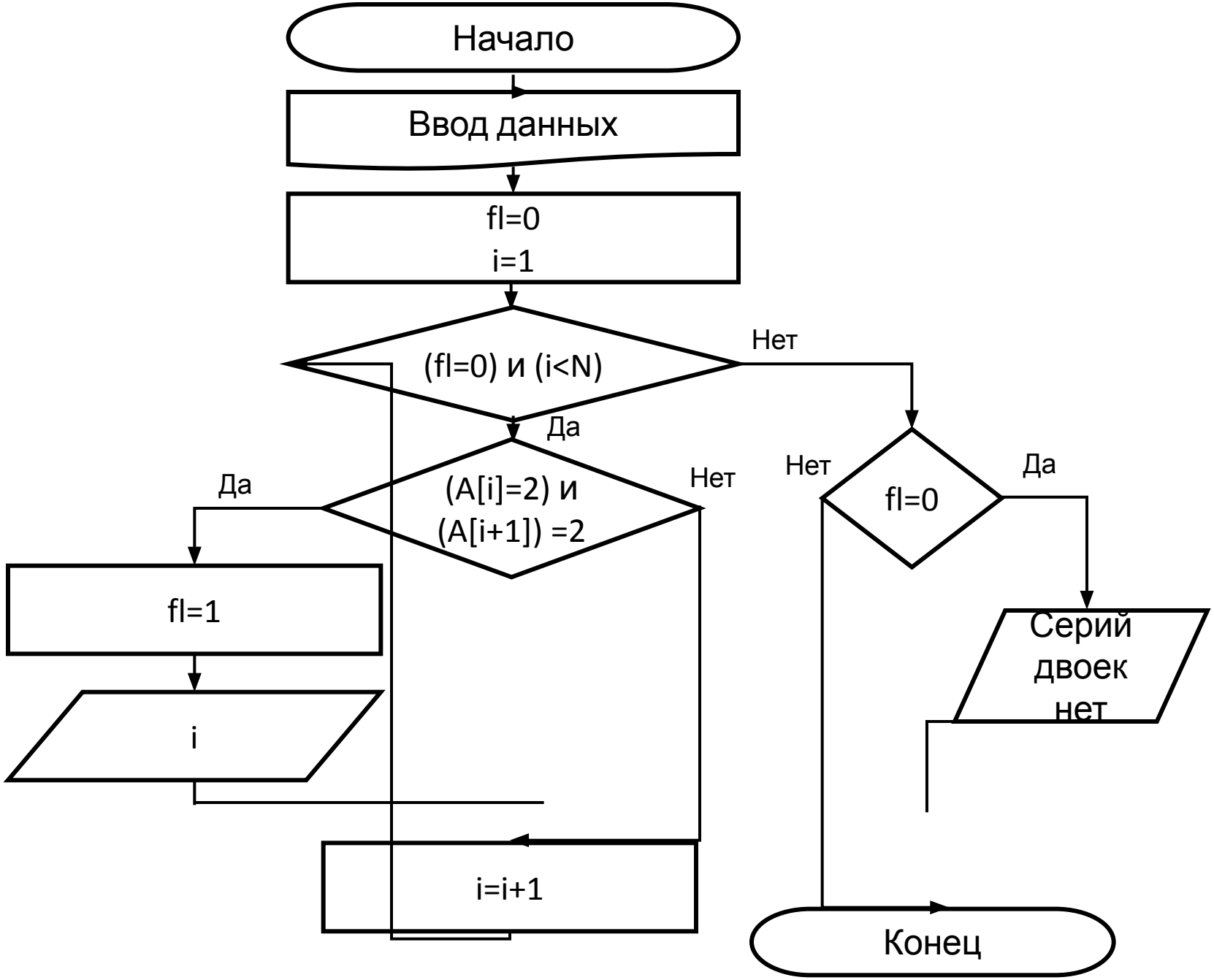
Одномерный массив произвольной длины. Ввод



Задача 3

- В одномерном массиве A[1..N] найти номер первого элемента первой серии двоек. Серией считаем минимум две двойки, стоящие рядом.




Начало
Ввод данных
fl=0
i=1
(fl=0) и (i<N)
Нет
Да
(A[i]=2) и
(A[i+1]) =2
Да
Нет
fl=1
i
i=i+1
Нет
fl=0
Да
Серий
двоек
нет
Конец

Сортировка элементов массива

5	1	6	-2	0	8	8	1

По возрастанию

-2	0	1	1	5	6	8	8

По убыванию

8	8	6	5	1	1	0	-2

Сортировка элементов массива м. «пузырька»

- Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

Сортировка элементов массива м. «пузырька». Пример по возрастанию. Первый проход

5	1	6	-2	0	8	8	1

5>1? – да, меняем “5” и “1” местами

1	5	6	-2	0	8	8	1

5>6? – нет

6>-2? – да, меняем “6” и “-2”

1	5	-2	6	0	8	8	1

6>0? – да, меняем “6” и “0” местами

1	5	-2	0	6	8	8	1

6>8? - нет

8>8? - нет

8>1? – да, меняем “8” и “1”

1	5	-2	0	6	8	1	8

Сортировка элементов массива м. «пузырька». Пример по возрастанию. Второй проход

1	5	-2	0	6	8	1	8

1>5? – нет

5>-2? – да, меняем “5” и

1	-2	5	0	6	8	1	8

5>0? – да, меняем “5” и “0” местами

1	-2	0	5	6	8	1	8

5>6? – нет

6>8? - нет

8>1? – да, меняем «8» и

1	-2	0	5	6	1	8	8

Сортировка элементов массива м. «пузырька». Пример по возрастанию. Третий проход

1	-2	0	5	6	1	8	8

1>-2? – да, меняем “1” и

-2	1	0	5	6	1	8	8

1>0? – да, меняем “1” и

-2	0	1	5	6	1	8	8

1>5? – нет

5>6? – нет

6>1? – да, меняем «6» и «1»

-2	0	1	5	1	6	8	8

Сортировка элементов массива м. «пузырька». Пример по возрастанию. Четвертый проход

-2	0	1	5	1	6	8	8

-2>0? – нет

0>1? – нет

1>5? – нет

5>1? – да, меняем “5” и “1”

-2	0	1	1	5	6	8	8

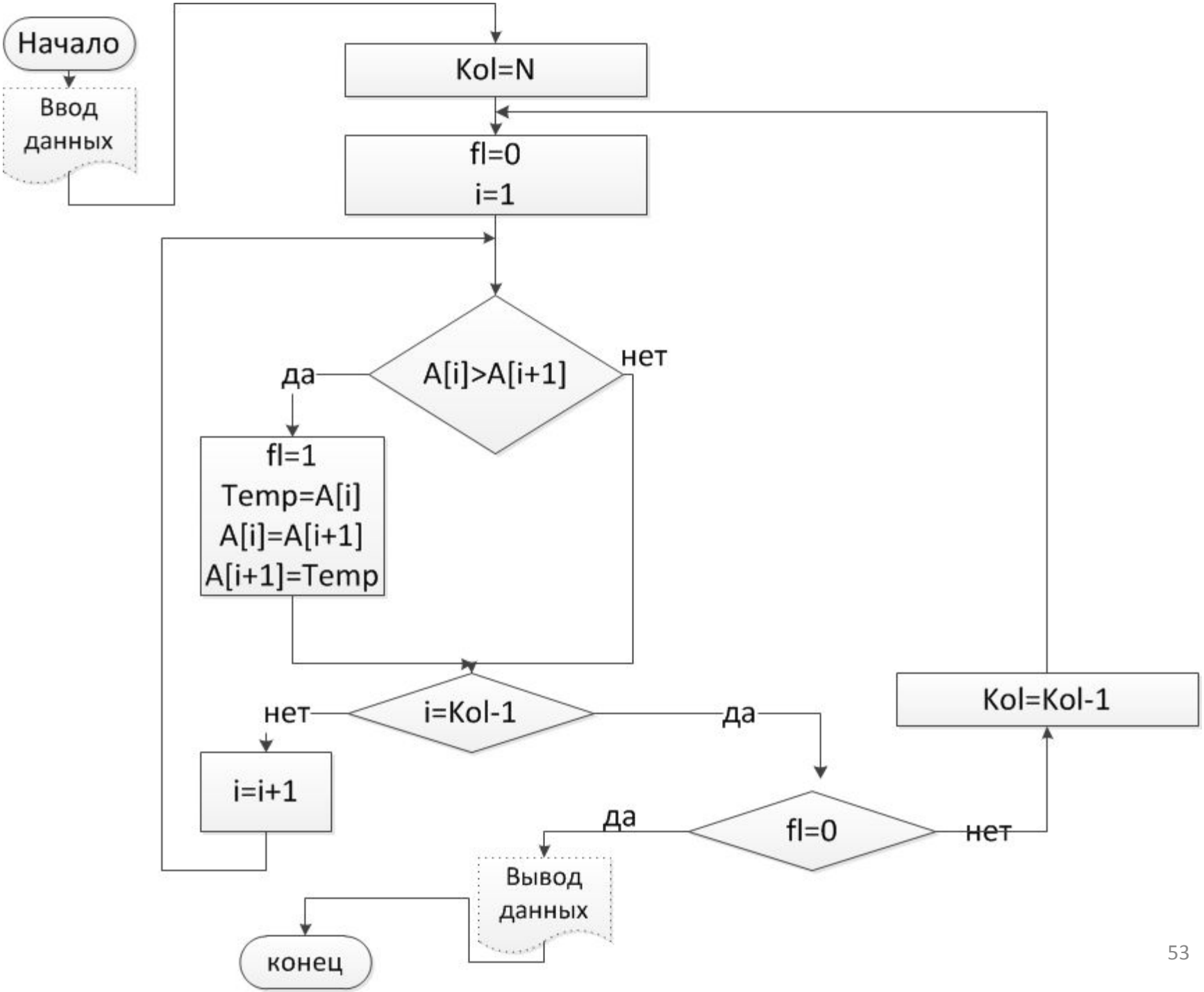
В следующем (пятом) проходе перестановок не будет, значит массив отсортирован.

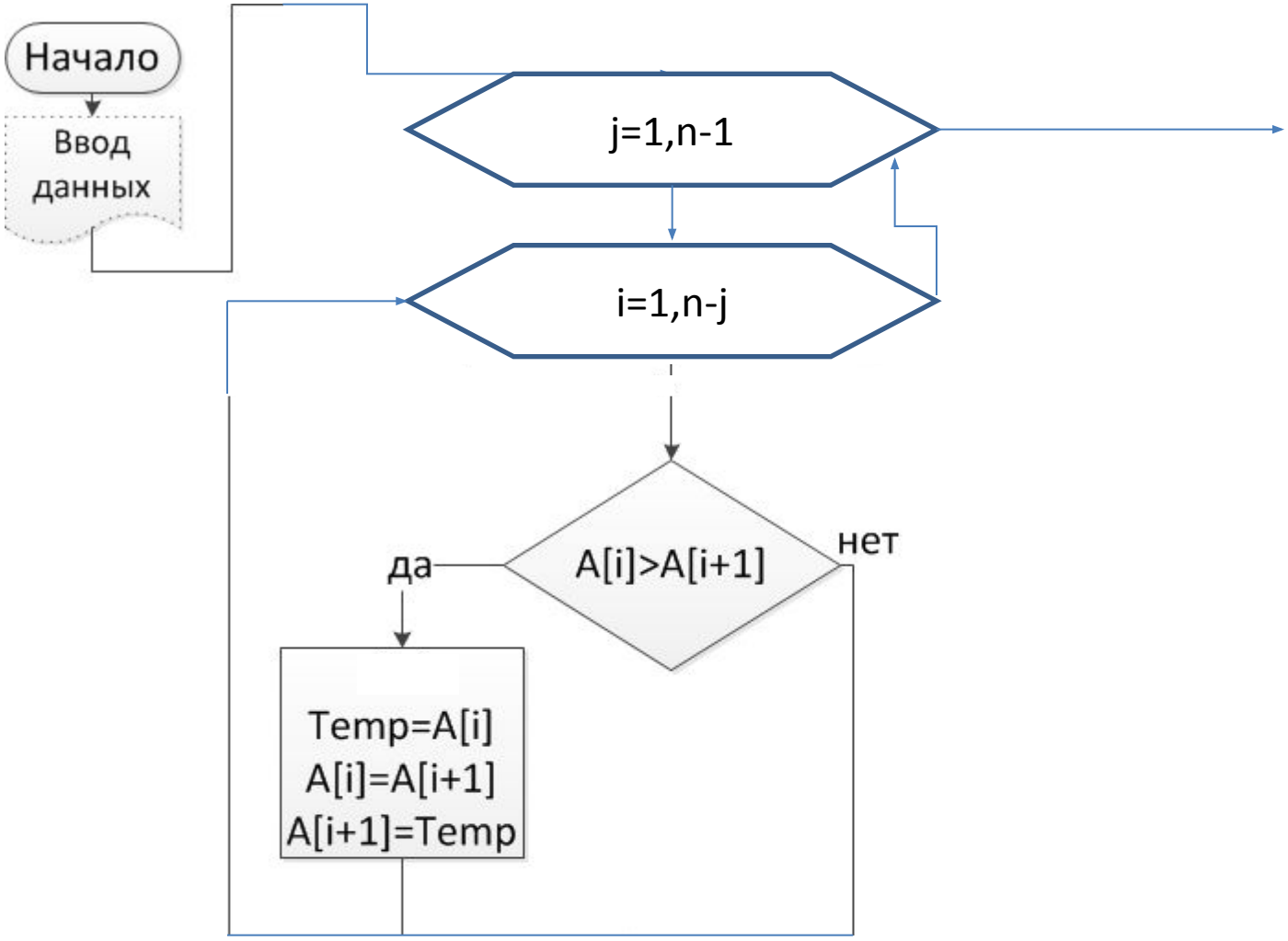
Сортировка элементов массива. Меняем элементы массива местами

- Две переменные нельзя просто перезаписать, необходимо вводить дополнительную переменную.
- Temp=A[i] (записываем значение во временную переменную)
- A[i]=A[i+1] (значение A[i] теряется, но оно есть в Temp)
- A[i+1]=Temp

Сортировка элементов массива. Переменные

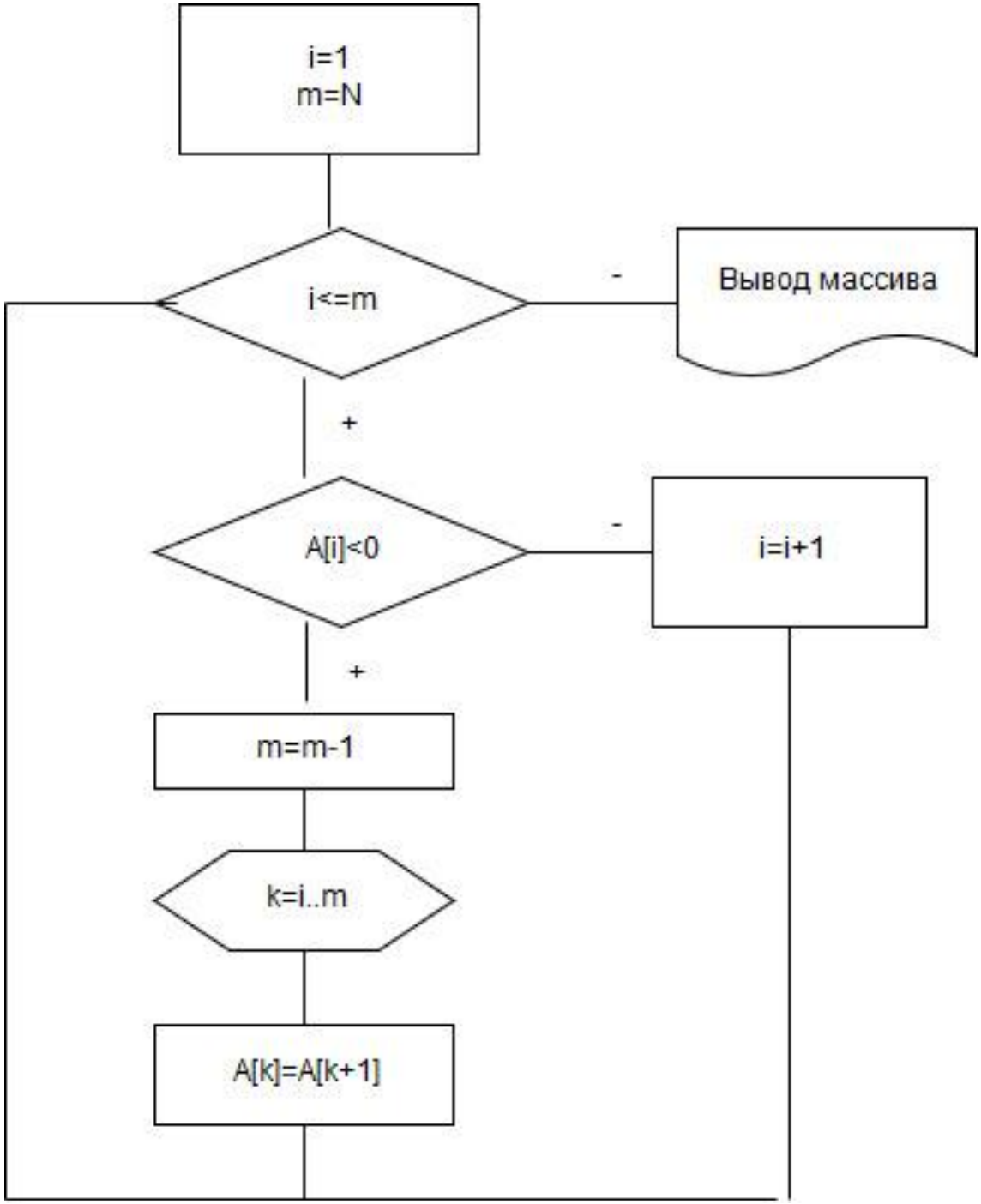
- fl – индикатор перестановок, fl=0 – не было перестановок, fl=1 – были перестановки.
- Temp-дополнительная переменная для обмена элементов массива.
- Kol – количество элементов для сравнения (уменьшается с каждым шагом, так как один элемент «всплывает»).
- i– индекс массива.
- A[1..N]-исходный массив из N элементов.


Начало
Ввод данных
Kol=N
fl=0
i=1
A[i]>A[i+1]
да
нет
fl=1
Temp=A[i]
A[i]=A[i+1]
A[i+1]=Temp
i=Kol-1
нет
да
i=i+1
Kol=Kol-1
fl=0
да
нет
Вывод данных
конец


Начало
Ввод данных
j=1,n-1
i=1,n-j
A[i]>A[i+1]
да
нет
Temp=A[i]
A[i]=A[i+1]
A[i+1]=Temp

Задача

Удалить из массива все отрицательные элементы


i=1
m=N
i<=m
-
Вывод массива
+
A[i]<0
-
i=i+1
+
m=m-1
k=i..m
A[k]=A[k+1]

Задача

Найти самую длинную серию, состоящую из одинаковых элементов. Вывести количество элементов самой длинной серии и номер элемента, который является её началом.

Решение

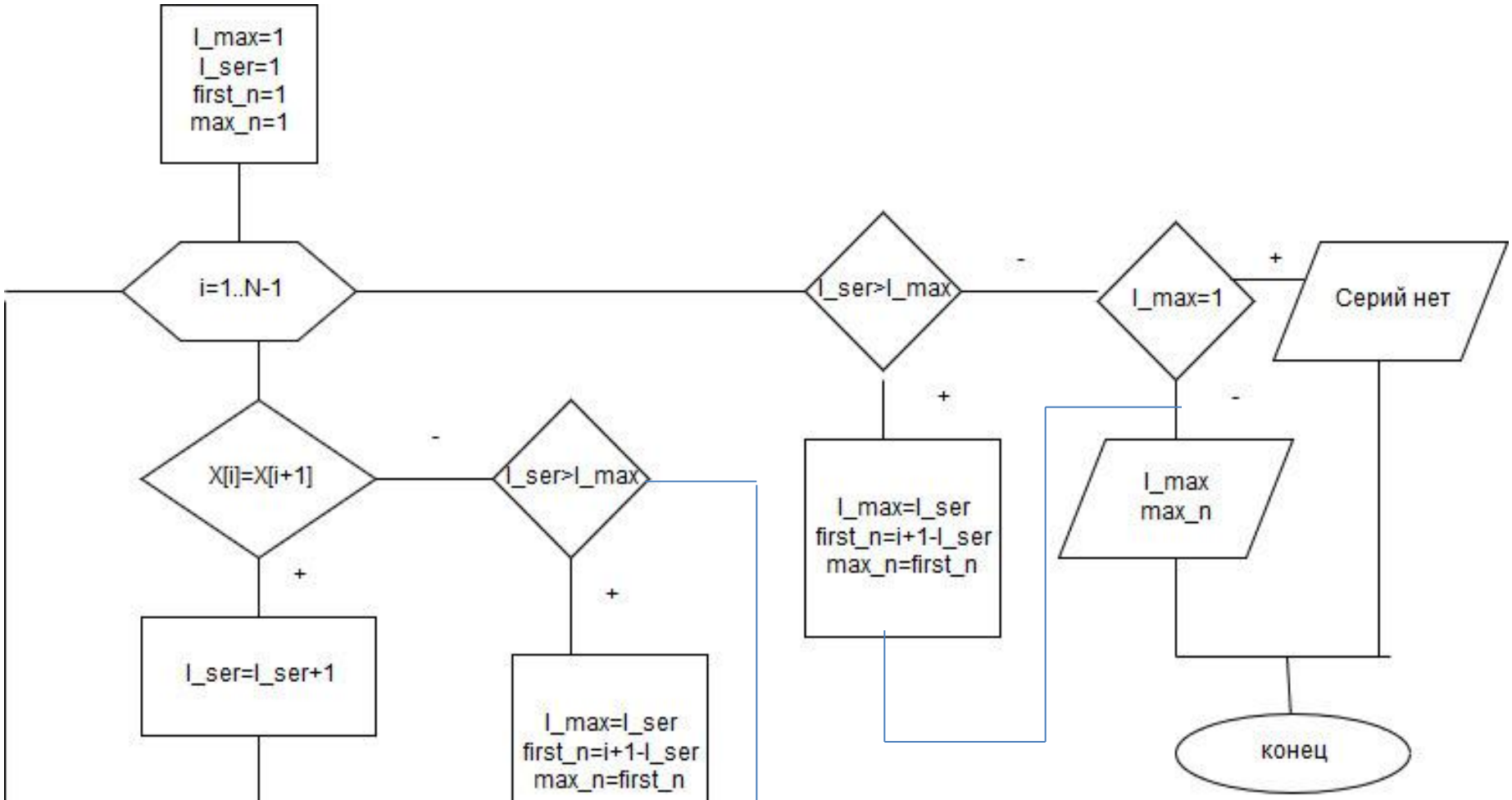
l_max – длина самой большой серии;

l_ser – длина текущей серии;

first_n – номер 1-го элемента для текущей серии;

max_n – номер 1-го элемента самой длинной серии

Необходима дополнительная проверка для случая, если последний элемент серии, является последним элементом массива (например, 1101111 , т.е. после последней проверки X[6]=X[7] мы сделаем l_ser=4, но не обновим l_max, т.к. выйдем из цикла, из-за того, что следующий шаг будет равен N.


l_max=1
l_ser=1
first_n=1
max_n=1
i=1..N-1
X[i]=X[i+1]
+
-
l_ser=l_ser+1
l_ser>l_max
+
l_max=l_ser
first_n=i+1-l_ser
max_n=first_n
l_ser=1
first_n=1
l_ser>l_max
+
-
l_max=l_ser
first_n=i+1-l_ser
max_n=first_n
l_max=1
+
-
Серий нет
l_max
max_n
конец

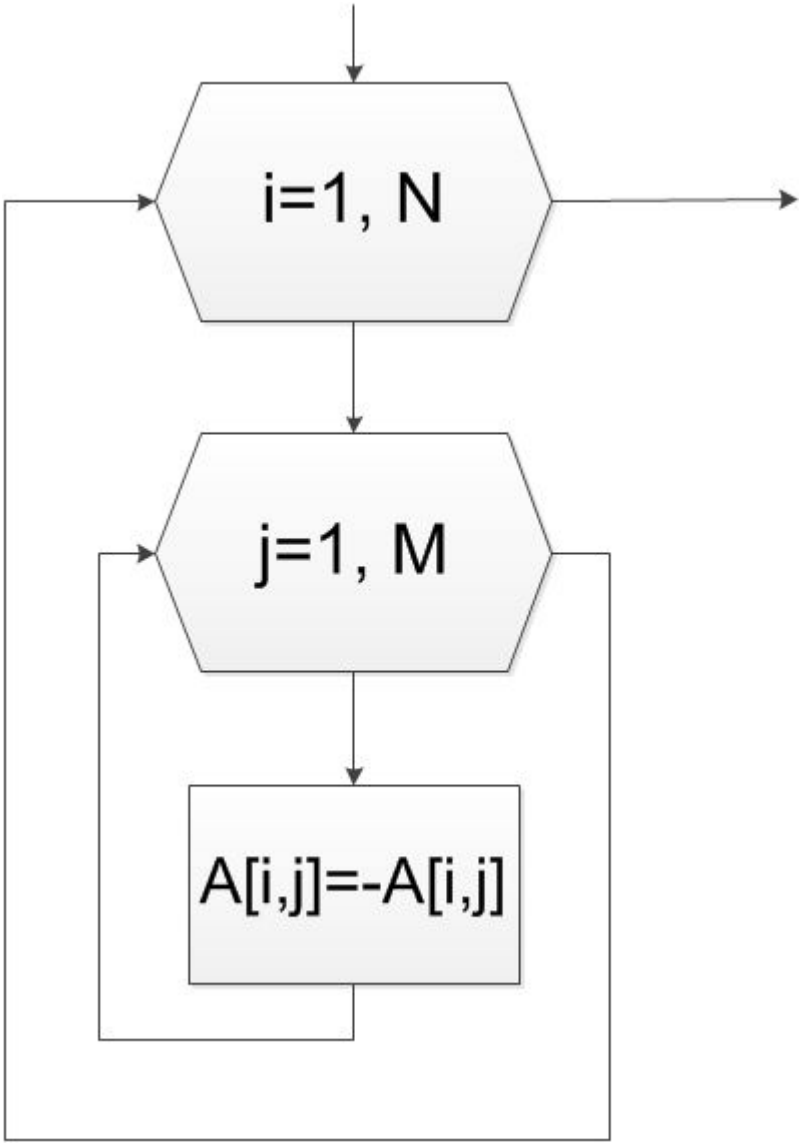
Многомерные массивы, матрицы

- Многомерные массивы – многоэтажные дома, у которых каждый этаж имеет свою нумерацию – от 1 и до общего количества квартир.
- Наиболее распространенный случай – матрица, у которой N строк и M столбцов.
- Для прохода по матрице используют два индекса: i – по строкам, j – по столбцам.

Матрицы

5	**-5**	**9**	**10**	**2**
1	**-2**	**0**	**5**	**6**
12	**-9**	**7**	**3**	**2**
14	**1**	**9**	**0**	**0**
5	**-6**	**-32**	**-1**	**4**
78	**8**	**9**	**0**	**3**

A[1,1]=5
A[3,2]=-9



i=1,j=1, A[1,1]=-5;
i=1,j=2, A[1,2]=5;
i=1,j=3, A[1,3]=-9;
i=1,j=4, A[1,4]=-10;
i=1,j=5, A[1,5]=-2;
i=2,j=1, A[2,1]=-1;
….

Литература по основам алгоритмизации

1. Вирт Н. Алгоритмы и структуры данных. — М.: Мир, 1989.
2. Могилев А. В., Пак Н.И., Хеннер Е.К. Информатика: Учеб. пособие для студ. пед. вузов / Под ред. Е. К. Хеннера. — М.: Изд. центр «Академия», 1999.
3. Бондарев В.М., Рублинецкий В.И., Качко Е.Г. Основы программирования. —Харьков: Фолио, Ростов н/Д: Феникс, 1997.

Литература (продолжение)

4. Алгоритмы: построение и анализ. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. – Вильямс. – 2012. - 1296 стр.
5. Б. Керниган, Р. Пайк. Практика программирования. – Вильямс. – 2004. – 288 с.

Функции в программировании

- **Функция** в программировании — это проименованная часть программы, которая может вызываться из других частей программы столько раз, сколько необходимо. Функция, в отличие от процедуры, обязательно возвращает значение.

Функции в программировании

- **Функция** в программировании — отдельная система (подсистема, подпрограмма), на вход которой поступают управляющие воздействия в виде значений аргументов. На выходе функция возвращает результат, который может быть как скалярной величиной, так и векторным значением (структура, индексный массив и т.п.).

Функции в программировании

К функции можно обращаться очень часто и с любой точки программы, но даже если оформленный в виде функций фрагмент кода будет вызываться лишь один раз, такой подход к организации программы повысит ее структурированность и,как результат, улучшит ее читабельность и облегчит процесс модификации программы.

Общий синтаксис

```
тип имя_функции(тип1 аргумент1,…)
{
    …
    возврат значения_типа_функции;
}
```

Замечания

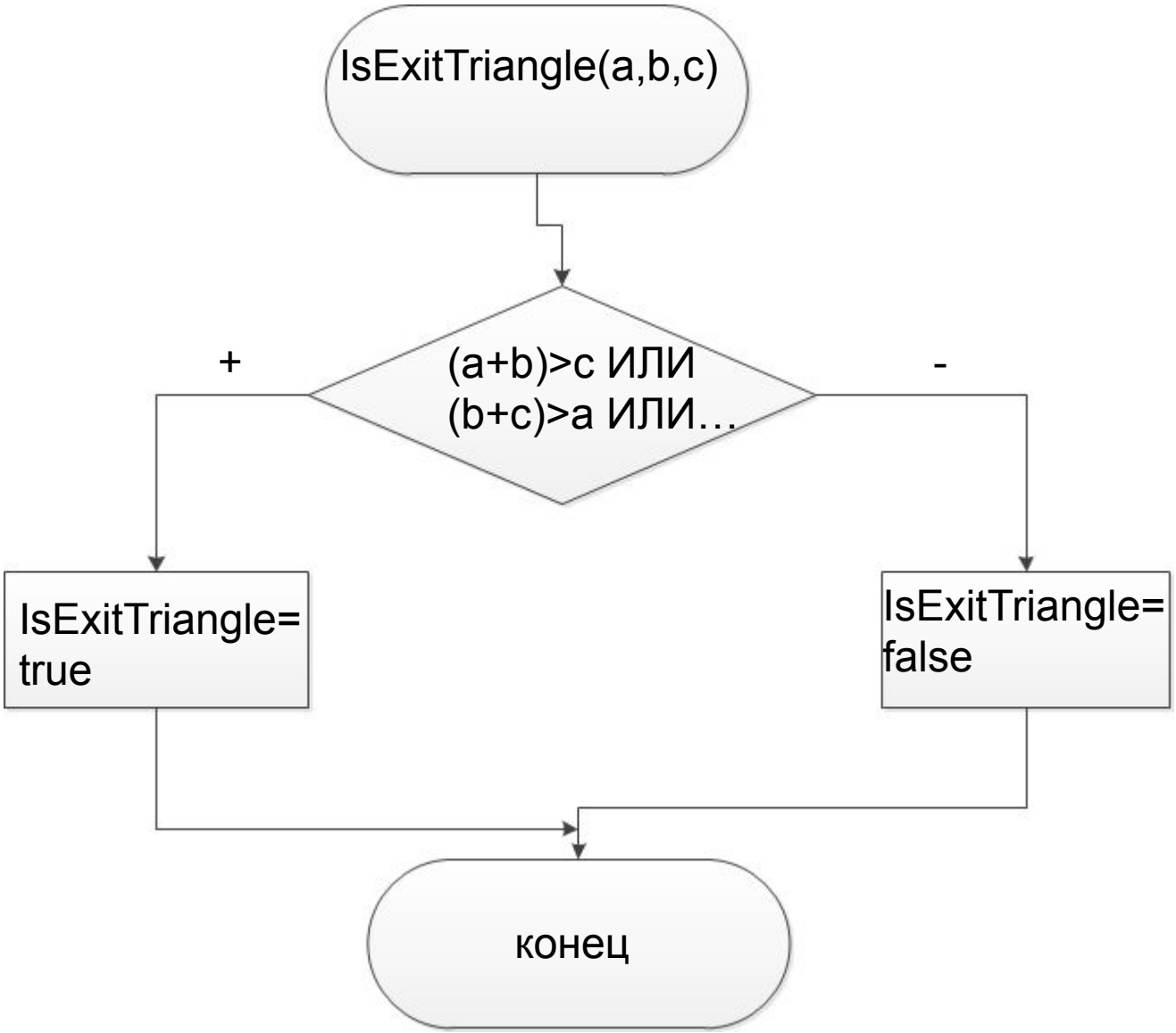
- В Си:
  - тип **void** – функция ничего не возвращает, имеем частный случай функции, которая является процедурой.
- В Паскале:
  - существуют отдельно функции **function**, отдельно процедуры **procedure**.

Примеры

- bool IsExitTriangle(int a, int b, int c)

Вызов функции на блок-схеме

Блок-схема функции



Глобальные и локальные переменные

- Если в процессе работы функции мы изменяем переменные основной программы, то мы изменяем глобальные переменные.
- Область видимости локальных переменных внутри функции.

Задача

- Нарисовать блок-схему алгоритма поиска минимального отрицательного элемента в массиве длины N. Ввод элементов массива оформить в виде отдельной функции.

Задача

1. Нарисовать блок-схему алгоритма поиска корней квадратных уравнений, составленных из элементов массива длины N.

Пример:

-3	7	8	10	-1	2

Получаем два квадратных уравнения с коэффициентами:
a=-3, b=7, c=8
a=10, b=-1, c=2