
Е. А. КОНОВА,
Г. А. ПОЛЛАК

АЛГОРИТМЫ И ПРОГРАММЫ. ЯЗЫК C++

Издание второе, стереотипное

ДОПУЩЕНО
УМО по образованию в области прикладной информатики
в качестве учебного пособия для студентов,
обучающихся по направлению
«Прикладная информатика»



САНКТ-ПЕТЕРБУРГ
МОСКВА • КРАСНОДАР
2017

ББК 32.973.26-018.1я73

К 64

Конова Е. А., Поллак Г. А.

К 64 Алгоритмы и программы. Язык С++: Учебное пособие. — 2-е изд., стер. — СПб.: Издательство «Лань», 2017. — 384 с.: ил. — (Учебники для вузов. Специальная литература).

ISBN 978-5-8114-2020-9

При изложении материала авторы используют методику обучения от алгоритмов к программам, поэтому вначале излагаются сведения об алгоритмах с примерами реализации типовых алгоритмов. Изучение основ языка программирования С++ опирается на полученные знания. Примеры можно решать в любой среде разработчика, поддерживающей язык С++, но авторами примеры отлажены в Visual Studio 2013. Коды программ соответствуют стандарту С++11 (ISO/IEC 14882:2011), разработаны в консольных приложениях на основе шаблона «Пустой проект».

В задачах практикума предлагаются как задачи, использующие типовые алгоритмы, так и содержательные, для которых приведено только вербальное описание. Пособие предназначено для студентов направления подготовки «Прикладная информатика» и других, может быть рекомендовано для самостоятельного изучения, так как не требует предварительных знаний о языках программирования.

ББК 32.973.26-018.1я73

Рецензенты:

В. И. ШИРЯЕВ — доктор технических наук, профессор, зав. кафедрой «Системы управления» Южно-Уральского государственного университета; *И. В. САФРОНОВА* — кандидат технических наук, доцент, зав. кафедрой прикладной информатики и математики Уральского социально-экономического института (филиала) Академии труда и социальных отношений.

Обложка

Е. А. ВЛАСОВА

*Охраняется законом РФ об авторском праве.
Воспроизведение всей книги или любой ее части
запрещается без письменного разрешения издателя.
Любые попытки нарушения закона
будут преследоваться в судебном порядке.*

© Издательство «Лань», 2017

© Е. А. Конова, Г. А. Поллак, 2017

© Издательство «Лань»,
художественное оформление, 2017

ВВЕДЕНИЕ

Авторы прекрасно понимают трудности, которые возникают у начинающего программиста, поэтому учебное пособие предназначено, в первую очередь, для тех, кто только начинает изучать программирование.

В разработке методики изложения авторы придерживались определения, данного Н. Виртом [1]: Программа = Алгоритмы + Данные.

Согласно этому определению программы представляют собой конкретные формулировки абстрактных алгоритмов, основанные на конкретных представлениях данных. Именно поэтому авторами уделяется большое внимание как разработке алгоритмов, так и концепции данных языка программирования.

Учебное пособие состоит из трех глав.

В первой главе приводятся сведения о типовых алгоритмах обработки данных безотносительно к языку программирования. Приведены примеры решения некоторых классов задач, где для каждой задачи разработан алгоритм в виде блок-схемы с пояснениями, набор тестовых данных и таблица исполнения алгоритма. Приводимые способы решения задач по возможности являются рациональными, но не претендуют на то, чтобы быть наилучшими.

В качестве языка программирования выбран классический C++. Краткое описание синтаксических правил C++ и механизмов реализации приведено во второй главе. Материал разбит на темы, позволяющие изучать язык по принципу «от простого к сложному». В изложении авторы опираются на материал первой главы, приводя программную реализацию всех алгоритмов. Добавлено мно-

го примеров, иллюстрирующих как особенности языка C++, так и некоторые алгоритмические приемы решения задач. В качестве методологии выбрано структурное программирование на основе функций.

Все приведенные программы проверены на работоспособность авторами в среде проектирования Visual Studio 2013. Коды программ соответствуют стандарту C++11 (ISO/IEC 14882:2011) и разработаны в консольных приложениях на основе шаблона «Пустой проект».

Третья глава — это практикум, который содержит задания для самостоятельного выполнения. В практикуме авторы опираются на материал второй главы. Третья глава состоит из десяти основных тем. В каждой теме есть краткое теоретическое введение, а также примеры программ решения некоторых задач. В каждом примере назван типовой алгоритм и приведен код программы с подробными комментариями.

Примеры и типовые решения помогут начинающим в освоении практических приемов программирования и выработке собственного стиля.

В каждой теме для самостоятельного решения предлагаются по 30 вариантов задач примерно одинакового уровня сложности. Особый интерес, по нашему мнению, представляют содержательные задачи, в которых постановка задачи выполнена на вербальном уровне, т. е. не формализована. Здесь студент должен самостоятельно осмыслить задачу, формализовать ее, предложить структуру данных и выбрать или разработать алгоритм решения. Опыт показывает, что часто именно этот этап в практическом программировании является наиболее трудным.

Материал пособия консолидирует многолетний опыт работы авторов в преподавании различных курсов программирования.

Соответствует ФГОС ВПО третьего поколения для направления «Прикладная информатика».

Не требуется каких-либо предварительных знаний о языках программирования, поэтому пособие может быть рекомендовано для самостоятельного изучения.

ОСНОВЫ АЛГОРИТМИЗАЦИИ

1.1. ОПРЕДЕЛЕНИЕ АЛГОРИТМА И ЕГО СВОЙСТВА

Под алгоритмом понимается точное предписание, задающее последовательность действий, которая ведет от произвольного исходного данного (или от некоторой совокупности возможных для данного алгоритма исходных данных) к достижению полностью определяемого этим исходным данным результата.

Алгоритм должен обладать определенными свойствами, наличие которых гарантирует получение решения задачи исполнителем.

Дискретность. Решение задачи должно быть разбито на элементарные действия. Запись отдельных действий реализуется в виде упорядоченной последовательности отдельных команд, образующих дискретную структуру алгоритма. Это свойство непосредственно отражено в определении алгоритма.

Понятность. На практике любой алгоритм предназначен для определенного исполнителя, и любую команду алгоритма исполнитель должен уметь выполнить.

Определенность (детерминированность). Каждая команда алгоритма должна определять однозначные действия исполнителя. Результат их исполнения не должен зависеть от факторов, не учтенных в алгоритме явно. При одних и тех же исходных данных алгоритм должен давать стабильный результат.

Массовость. Разработанный алгоритм должен давать возможность получения результата при различных исходных данных для однотипных задач.

Например, пользуясь алгоритмом решения квадратного уравнения, можно находить его корни при любых значениях коэффициентов.

Свойство массовости полезное, но не обязательное свойство алгоритма, так как интерес представляют и алгоритмы, пригодные для решения единственной задачи.

Результативность (конечность). Это свойство предполагает обязательное получение результата решения задачи за конечное число шагов. Под решением задачи понимается и сообщение о том, что при заданных значениях исходных данных задача решения не имеет.

Если решить задачу при заданных исходных данных за конечное число шагов не удается, то говорят, что алгоритм «зацикливается».

Смысл условий дискретности, понятности и определенности ясен: их нарушение ведет к невозможности выполнения алгоритма. Остальные условия не столь очевидны. Для сложных алгоритмов выполнить исчерпывающую проверку результативности и корректности невозможно. Это равносильно полному решению задачи, для которой создан алгоритм, вручную.

Можно сформулировать общие правила, руководствуясь которыми следует записывать алгоритм решения задачи.

1. Выделить величины, являющиеся исходными данными для задачи.
2. Разбить решение задачи на такие команды, каждую из которых исполнитель может выполнить однозначно.
3. Указать порядок выполнения команд.
4. Задать условие окончания процесса решения задачи.
5. Определить, что является результатом решения задачи в каждом из возможных случаев.

Хотя алгоритмы обычно предназначены для автоматического выполнения, они создаются и разрабатываются людьми. Поэтому первоначальная запись алгоритма обычно производится в форме, доступной для восприятия человеком.

Самой простой является *словесная* форма записи алгоритмов на естественном языке. В этом виде алгоритм представляет собой описание последовательности этапов обработки данных, изложенное в произвольной форме.

Словесная форма удобна для человеческого восприятия, но страдает многословностью и неоднозначностью.

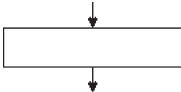
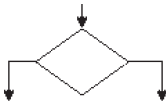

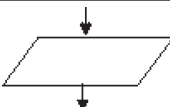

Когда запись алгоритма формализована частично, то используется *псевдокод*. Он содержит как элементы естественного языка, так и формальные конструкции, описывающие базовые алгоритмические структуры. Эти конструкции называются *служебными словами*. Формального определения псевдокода или строгих правил записи алгоритмов в таком формате не существует.

Графическая форма представления алгоритма является более компактной. Алгоритм изображается в виде последовательности связанных между собой блоков, каждый из которых соответствует выполнению одного или нескольких действий. Графическое представление алгоритма называется *блок-схемой*. Блок-схема определяет структуру алгоритма.

Графические обозначения блоков стандартизованы. Некоторые из часто используемых блоков представлены в таблице 1.1.

Таблица 1.1

Изображение основных блоков на блок-схеме

Обозначение блока	Пояснение
	Процесс (вычислительное действие, реализованное операцией присваивания)
	Решение (проверка условия, реализующая условный переход)
	Начало, конец алгоритма
	Ввод-вывод в общем виде
	Модификация (начало цикла с параметром)

Отдельные блоки соединяются линиями переходов, которые определяют очередность выполнения действий. Направление линий сверху вниз или слева направо принимается за основное.

Алгоритм, записанный на языке программирования, называется *программой*. При использовании этих языков запись алгоритма абсолютно формальна и пригодна для выполнения на ЭВМ. Отдельная конструкция языка программирования называется *оператором*. Программа — это упорядоченная последовательность операторов.

1.2. БАЗОВЫЕ АЛГОРИТМИЧЕСКИЕ КОНСТРУКЦИИ

Число реализованных конструкций конечно в любом языке программирования. Структурной элементарной единицей алгоритма является команда, обозначающая один элементарный шаг обработки или отображения информации. Простая команда на языке блок-схем изображается в виде функционального блока «процесс», который имеет один вход и один выход. Из команд проверки условий и простых команд образуются составные команды, имеющие более сложную структуру, но тоже один вход и один выход.

Алгоритм любой сложности может быть представлен комбинацией трех базовых структур:

- следование;
- ветвление (в полной и сокращенной форме);
- цикл (с предусловием или постусловием).

Характерной особенностью этих структур является наличие у них одного входа и одного выхода.

1.2.1. Линейные алгоритмы

Базовая структура «следование» означает, что несколько операторов выполняются последовательно друг за другом, и только один раз за время выполнения программы. Структура «следование» используется для реализации задач, имеющих *линейный* алгоритм решения. Это означает, что такой алгоритм не содержит проверок

условий и повторений, действия в нем выполняются последовательно, одно за другим.

Пример 1.1. Построить блок-схему алгоритма вычисления высот треугольника со сторонами a, b, c по формулам:

$$h_a = \left(\frac{2}{a}\right) \cdot \sqrt{p(p-a)(p-b)(p-c)};$$

$$h_b = \left(\frac{2}{b}\right) \cdot \sqrt{p(p-a)(p-b)(p-c)};$$

$$h_c = \left(\frac{2}{c}\right) \cdot \sqrt{p(p-a)(p-b)(p-c)},$$

где $p = \frac{(a+b+c)}{2}$ — полупериметр.

Для того чтобы не вычислять три раза одно и то же значение, введем вспомогательную величину:

$$t = 2\sqrt{p(p-a)(p-b)(p-c)}.$$

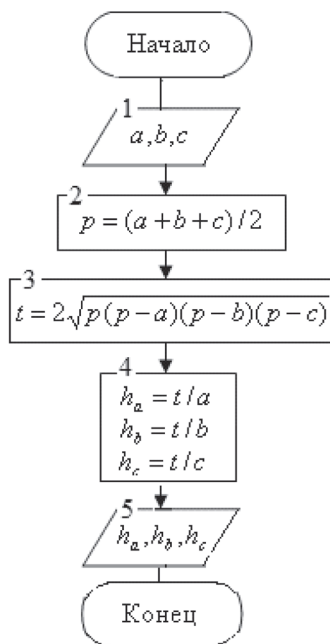
Блок 1. Ввод значений сторон треугольника.

Блок 2. Вычисление полупериметра.

Блок 3. Вычисление вспомогательной величины t .

Блок 4. Вычисление высот, опущенных на стороны a, b, c .

Блок 5. Вывод результатов.



1.2.2. Разветвляющиеся алгоритмы

Второй базовой структурой является «ветвление». Эта структура обеспечивает, в зависимости от результата проверки условия, выбор одного из альтернативных путей работы алгоритма, причем каждый из путей ведет к обще-

му выходу, так что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран.

Существует структура с полным и неполным ветвлением.

Структура с полным ветвлением (*если — то — иначе*) записывается так:

Если <условие>

то <действия 1>

иначе <действия 2>

Все если

Команда выполняется так: если <условие> является истинным, то выполняются <действия 1>, записанные после ключевого слова *то*, если <условие> является ложным, то выполняются <действия 2>, записанные после слова *иначе*.

Структура с неполным ветвлением (*если — то*) не содержит части, начинающейся со слова *иначе*:

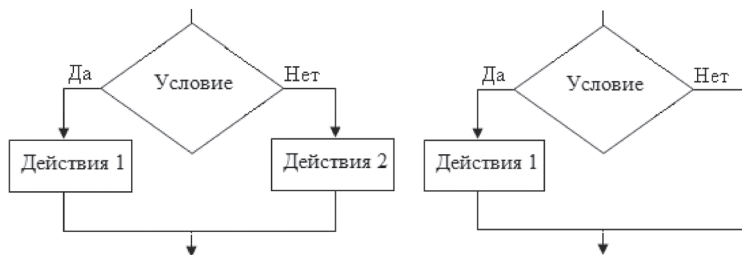
Если <условие>

то <действия 1>

Все если

Команда выполняется так: если <условие> является истинным, то выполняются <действия 1>, записанные после ключевого слова *то*.

Блок-схема алгоритма с ветвлением выглядит так:



Полное ветвление. Структура
Если — То — Иначе

Неполное ветвление.
Структура *Если — То*

Пример 1.2. Вычислить значение функции

$$y = \begin{cases} x + a, & \text{при } x < 10; \\ x + b, & \text{при } 10 \leq x \leq 20; \\ x + c, & \text{при } x > 20. \end{cases}$$

Дано: x, a, b, c — произвольные числа.

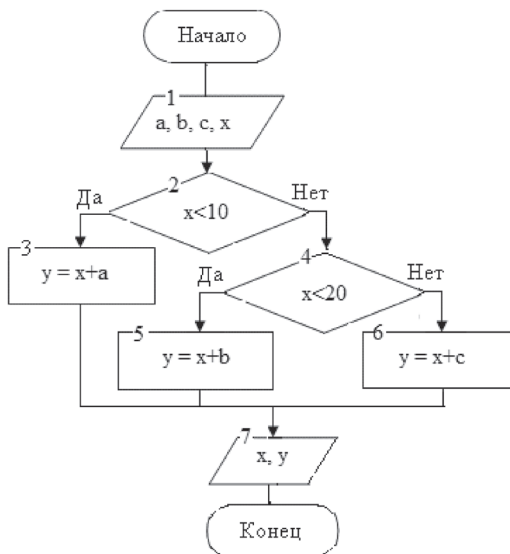
Найти: y .

Представим задачу графически на числовой оси

$x < 10$	$10 \leq x \leq 20$	$x > 20$
10		20
$y = x + a$	$y = x + b$	$y = x + c$
1 интервал	2 интервал	3 интервал

Так как значение переменной x вводится произвольно, то оно может оказаться в любом из трех интервалов.

Приведем блок-схему.



Блок 1. Ввод исходных данных.

Блок 2. Проверка введенного значения. Если $x < 10$ (выход «Да»), то точка находится в первом интервале. В противном случае $x \geq 10$ (выход «Нет»), и точка может оказаться во втором или третьем интервале.

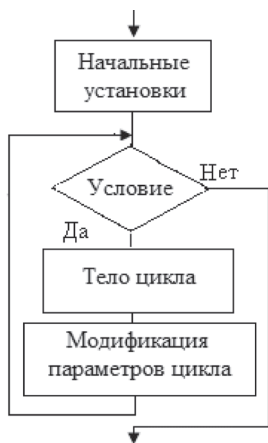
Блок 4. Проверка ограничения значения x справа ($x < 20$). Если условие выполняется (выход «Да»), то x находится во втором интервале, иначе $x \geq 20$, и точка находится в третьем интервале.

Блоки 3, 5 и 6. Вычисление значения y .

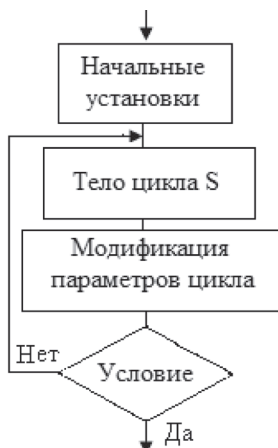
1.2.3. Циклические алгоритмы

При составлении алгоритмов решения большинства задач возникает необходимость в неоднократном повторении одних и тех же команд. Алгоритм, составленный с использованием многократных повторений одних и тех же действий (циклов), называется *циклическим*. Однако слово «неоднократно» не означает «до бесконечности». Организация циклов, никогда не приводящая к остановке в выполнении алгоритма («заикливание» алгоритма), нарушает требование его результативности — получения результата за конечное число шагов.

Блок, для выполнения которого организуется цикл, называется *телом цикла*. Остальные операторы служат для управления процессом повторения вычислений: это начальные установки, проверка условия продолжения цикла и модификация параметра цикла. Один проход цикла называется *итерацией*.



Цикл с предусловием



Цикл с постусловием

Начальные установки служат для того, чтобы до входа в цикл задать значения переменных, которые в нем используются.

Проверка условия продолжения цикла выполняется на каждой итерации либо до тела цикла (*цикл с предусло-*

вием), либо после тела цикла (*цикл с постусловием*). Тело цикла с постусловием всегда выполняется хотя бы один раз. Проверка необходимости выполнения цикла с предусловием делается до начала цикла, поэтому возможно, что он не выполнится ни разу.

При конструировании циклов следует соблюдать обязательное условие результативности алгоритма (т. е. его окончания за конечное число шагов). Практически это означает, что в условии должна быть переменная, значение которой изменяется в теле цикла. Причем, изменяется таким образом, чтобы условие в конечном итоге перестало выполняться. Такая переменная называется управляющей переменной цикла или *параметром* цикла.

Еще один вид циклов — цикл с *параметром*, или *арифметический цикл*. Тело цикла выполняется, пока параметр цикла i пробегает множество значений от начального (I_n) до конечного (I_k).

Переменная i определяет количество повторений тела цикла S . Если шаг изменения значения параметра цикла обозначить через ΔI , то количество повторений тела цикла n можно вычислить по формуле

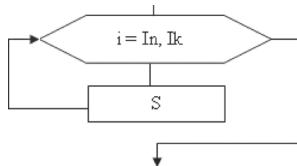
$$n = \frac{I_k - I_n}{\Delta I} + 1.$$

Если параметр цикла i изменяется с шагом 1, то шаг может не указываться.

Цикл выполняется так: начальное значение параметра цикла i равно I_n . Если $i \leq I_k$, выполняется тело цикла S , после чего параметр цикла увеличивается на 1 с помощью оператора присваивания $i = i + 1$, и снова проверяется условие $i \leq I_k$.

Пример 1.3. Дано целое положительное число n . Вычислить факториал этого числа. Известно, что факториал любого целого положительного числа n определяется как произведение чисел от 1 до заданного числа n :

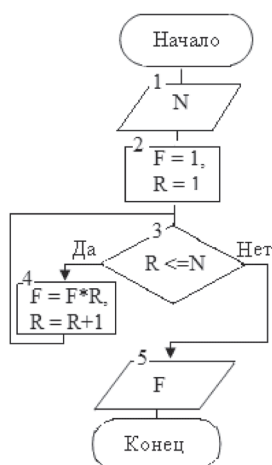
$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n.$$



По определению $0! = 1$ и $1! = 1$.

Задача решается с помощью циклического алгоритма. Введем следующие обозначения: N — заданное число, F — факториал числа, R — параметр цикла. Составим два варианта алгоритма: с использованием цикла с предусловием и цикла с параметром.

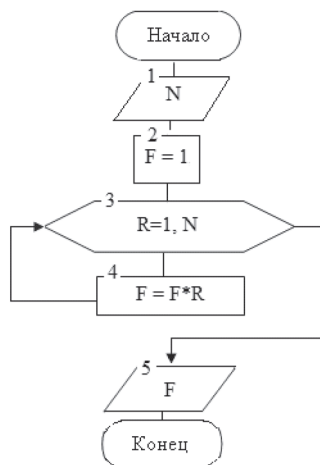
Правильность алгоритма можно проверить, если выполнить его формально «вручную». Выполним алгоритм при $n = 4$.



Цикл с предусловием

Цикл с предусловием

R	$R \leq N$	F
1	Да	$1 \cdot 1 = 1$
2	Да	$1 \cdot 2 = 2$
3	Да	$2 \cdot 3 = 6$
4	Да	$6 \cdot 4 = 24$
5	Нет	— (кц)



Цикл с параметром

Цикл с параметром

R	$R \leq N$	F
1	Да	$1 \cdot 1 = 1$
2	Да	$1 \cdot 2 = 2$
3	Да	$2 \cdot 3 = 6$
4	Да	$6 \cdot 4 = 24$
5	Нет	— (кц)

Итак, при решении данной задачи выполнение цикла с предусловием ничем не отличается от выполнения цикла с параметром. При $R = 5$ произойдет выход из цикла и окончательное значение $4! = 24$.

В чем же отличие? Посмотрим на структуру этих циклов. В цикле *пока* начальное значение параметра цикла R задается перед входом в цикл, в теле цикла организовано изменение параметра цикла командой $R = R + 1$, условие $R \leq N$ определяет условие продолжение цикла. В цикле с заданным числом повторений эти же команды неявно заданы в операторе заголовка цикла.

Пример 1.4. Пусть $a_0 = 1$; $a_k = k \cdot a_{k-1} + 1/k$, $k = 1, 2, \dots$. Дано натуральное число n . Получить a_n .

Дано: a_0 — первый член последовательности; n — номер члена последовательности, значение которого требуется найти.

Найти: a_n — n -й член последовательности.

Математическая модель. Посмотрим, как изменяется значение члена последовательности при изменении значения k . При $k = 1$ $a_1 = 1 \cdot a_0 + 1$. При $k = 2$ $a_2 = 2 \cdot a_1 + 1/2$. При $k = 3$ $a_3 = 3 \cdot a_2 + 1/3$. Выполнив указанные вычисления n раз, получим искомое значение a_n . В задачах такого типа не требуется хранить результаты вычислений на каждом шаге. Поэтому можно использовать простые переменные.

Обозначим через a — произвольный член последовательности. Тогда формула для вычисления члена последовательности будет выглядеть так:

$$a = k \cdot a + 1/k.$$

В этой формуле значение a , стоящее справа от знака «=», определяется на предыдущем шаге вычисления, а значение a , стоящее в левой части выражения, определяется на данном шаге и заменяет в памяти предыдущее значение.

Переменная a — вещественного типа; переменные k и n — целого типа.

Приведем словесное описание и блок-схему алгоритма.

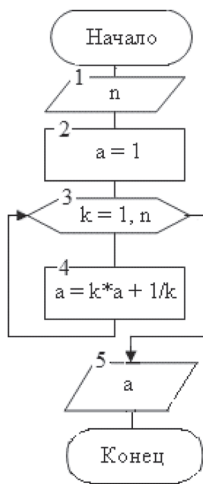
Блок 1. Ввод количества членов последовательности n .

Блок 2. Присваивание начального значения $a = 1$.

Блок 3. Арифметический цикл. Начальное значение параметра цикла k равно 1, конечное значение равно n , шаг изменения параметра — 1.

Блок 4. Выполнение тела цикла. Для каждого значения k вычисляется выражение $a = k * a + 1/k$.

Блок 5. Вывод значения a .



Пример 1.5. Даны натуральное число n , действительное число x . Вычислить сумму:

$$\sum_{i=1}^n \frac{x^i}{i!}.$$

Дано: n — количество слагаемых; x — действительное число.

Найти: S — сумму чисел.

Математическая модель. Обозначим s — слагаемое. Начальное значение суммы $S = 0$.

Количество слагаемых известно и равно n . Следовательно, можно использовать арифметический цикл.

Выведем рекуррентное соотношение, определяющее формулу, по которой вычисляется слагаемое на очередном шаге выполнения цикла.

Вспомним, что факториал числа n определяется как произведение чисел от 1 до n включительно. Следовательно, $i! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \dots \cdot i$, по определению $0! = 1$, $1! = 1$. Для вывода рекуррентного соотношения вычислим несколько значений слагаемых.

$$\begin{aligned}i = 1, \quad c &= x^1/1! = x; \\i = 2, \quad c &= x^2/2! = x^2/(1 \cdot 2) = c \cdot x/2 = c \cdot x/i; \\i = 3, \quad c &= x^3/3! = x^3/(1 \cdot 2 \cdot 3) = c \cdot x/3 = c \cdot x/i; \\i = 4, \quad c &= x^4/4! = x^4/(1 \cdot 2 \cdot 3 \cdot 4) = c \cdot x/4 = c \cdot x/i.\end{aligned}$$

В правой части всех формул значение c равно значению, вычисленному на предыдущем шаге. Итак, получаем следующее соотношение, которое будем использовать в алгоритме $c = c \cdot x/i$, где начальное значение $c = 1, i = 1, 2, \dots, n$.

Сумму накапливаем, используя формулу $S = S + c$. Записанное соотношение определяет, что значение суммы увеличивается на величину слагаемого при каждом повторении цикла. Начальное значение суммы равно 0.

Переменные i, n — целого типа, переменные c, x, S — вещественного типа.

Приведем словесное описание и блок-схему алгоритма.

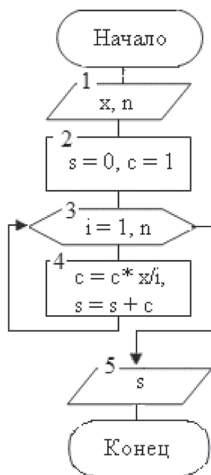
Блок 1. Ввод x, n .

Блок 2. Присваивание начальных значений $S = 0, c = 1$.

Блок 3. В цикле для всех значений параметра цикла i вычисляем тело цикла (блок 4):

$$\begin{aligned}c &= c * x/i; \\S &= S + c.\end{aligned}$$

Блок 5. Вывод суммы S .



1.2.4. Итерационные циклы

Все циклические алгоритмы можно разделить на две группы: циклы с заранее известным количеством повторений и циклы с заранее неизвестным количеством повторений. Циклы с заранее неизвестным количеством повторений называются *итерационными*. Итерационный процесс — это последовательное приближение к результату за некоторое количество шагов.

Рассмотрим блок-схемы итерационных циклов на примере.

Пример 1.6. Даны действительные числа x , ε ($x \neq 0$, $\varepsilon > 0$). Вычислить сумму с точностью ε :

$$\sum_{k=0}^{\infty} \frac{x^k}{2^k \cdot k!}$$

Вычисление бесконечной суммы с заданной точностью ε означает, что требуемая точность достигнута, когда вычислена сумма нескольких первых слагаемых и очередное слагаемое оказалось по модулю меньше, чем ε , — это и все последующие слагаемые можно не учитывать.

Пусть c — слагаемое, S — сумма, F — факториал числа k .

Математическая модель. Выведем рекуррентное соотношение, используя способ, показанный в примере 1.5.

Для вычисления факториала числа можно воспользоваться формулой $F = F \cdot k$, которую мы вывели ранее. Начальное значение $F = 1$. Проверьте правильность этой формулы, последовательно вычисляя значение $k!$.

Выведем общую формулу для вычисления одного слагаемого (без учета факториала числа k):

$$\begin{aligned} k=0, \quad c &= x^0/2^0 = 1; \\ k=1, \quad c &= x^1/2^1 = x/2; \\ k=2, \quad c &= x^2/2^2 = (x/2)(x/2) = c \cdot x/2; \\ k=3, \quad c &= x^3/2^3 = (x^2/4)(x/2) = c \cdot x/2; \\ k=4, \quad c &= x^4/2^4 = (x^3/8)(x/2) = c \cdot x/2. \end{aligned}$$

Обобщая, для произвольного значения k можно записать $c = c \cdot x/2$, где значение переменной c в правой части

формулы вычисляется на предыдущем шаге. Начальное значение $c = 1$ при $k = 0$.

Сумму вычисляем по формуле $S = S + c / F$ (см. пример 1.5). Начальное значение $S = 0$.

В данной задаче следует использовать цикл с предусловием, потому что количество слагаемых будет зависеть от введенного значения x и требуемой точности вычислений ε .

Переменная k — целого типа, переменные x , ε , c , S — вещественного типа. Переменную F следует взять вещественного типа, так как диапазон целых чисел ограничен, а факториал быстро возрастает с ростом значения k .

Приведем словесное описание и блок-схему алгоритма.

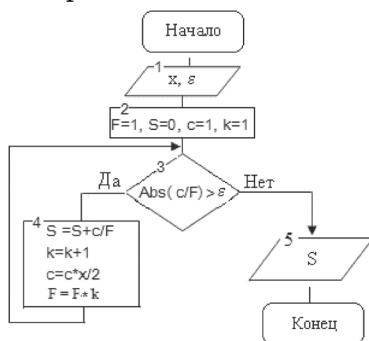
Блок 1. Ввод исходных данных x и ε .

Блок 2. Присваивание начальных значений $F = 1$, $S = 0$, $c = 1$. Параметр цикла $k = 1$.

Блок 3. Прежде чем вычислять сумму, определим выполнение условия продолжения цикла. По условию, когда очередное слагаемое окажется по модулю меньше заданной точности ε , вычисления следует прекратить. Слагаемым в нашем случае является дробь c/F . Так как может быть введено отрицательное значение x , то использована функция $Abs()$, определяющая модуль числа.

Блок 4. Если требуемая точность вычислений не достигнута (выход «Да» блока 3), выполняем тело цикла, определяем новые значения переменных и продолжаем цикл.

Блок 5. Как только требуемая точность вычисления достигнута (выход «Нет» блока 3), завершаем цикл и печатаем значение переменной S .



Цикл, который мы использовали при решении задачи, называется *итерационным*. Особенностью такого цикла является то, что число его повторений зависит от выполнения условия, записанного при входе в цикл. В итерационных алгоритмах необходимо обеспечить обязательное достижение условия выхода из цикла (сходимость итерационного процесса). В противном случае произойдет «зацикливание» алгоритма. Управление итерационным циклом организует программист, который должен позаботиться и об инициализации управляющей переменной, и об ее приращении, и об условии завершения цикла.

1.3. АЛГОРИТМЫ, ИСПОЛЬЗУЮЩИЕ ОДНОМЕРНЫЕ МАССИВЫ

Массив — это упорядоченный набор однотипных значений (элементов массива). Каждый массив имеет имя, что дает возможность различать массивы между собой и обращаться к ним по имени.

Каждый элемент массива имеет три характеристики:

- 1) *имя*, совпадающее с именем массива;
- 2) *индекс* — целое число или множество целых чисел, однозначно определяющее местоположение элемента в массиве. В качестве индекса может использоваться также переменная или арифметическое выражение целого типа. Примеры индексов: 3, 15, i , j , $i - 1$, $j + 2$;

- 3) *значение* — фактическое значение элемента, определенное его типом.

Элементы массива могут использоваться произвольно и являются одинаково доступными. Доступ к элементам массива производится по его индексу.

Массивы могут быть *одномерными* и *многомерными*. В этом подпараграфе рассмотрим некоторые алгоритмы на одномерных массивах.

1.3.1. Ввод и вывод элементов массива

Одномерный массив определяется именем и числом элементов (размером), и мы обозначим его $a[n]$, где a — имя массива; n — число элементов массива. Например,

$a[10]$, где a — имя массива; 10 — число элементов в массиве.

Каждый элемент одномерного массива имеет один индекс, равный порядковому номеру элемента в массиве. Например, массив из 10 элементов выглядит так:

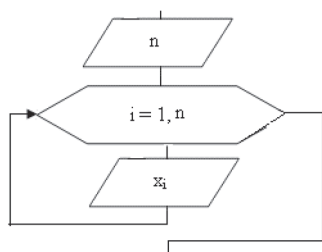
Индекс	1	2	3	4	5	6	7	8	9	10
Значение	3	0	15	4	6	-2	11	0	-9	7

$a[1] = 3$; $a[5] = 6$; $a[7] = 11$; $a[9] = -9$; $a[10] = 7$.

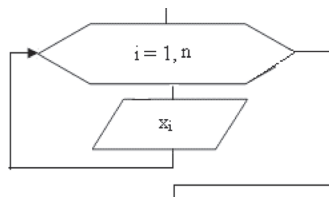
Так как всегда известно количество элементов в массиве, то для ввода и вывода его элементов используется цикл с заданным числом повторений.

Мы рассматриваем массив, состоящий из произвольного числа элементов. Поэтому, прежде чем задать значения элементов массива, требуется ввести количество элементов массива n .

Вывод элементов также производится с использованием цикла с заданным числом повторений.



Ввод элементов массива



Вывод элементов массива

Здесь и далее будем обозначать через i — текущий индекс элемента массива. Он же будет являться параметром цикла, так как количество повторений цикла зависит от количества элементов в массиве.

Далее во всех задачах будем считать, что элементы массива заданы тем или иным способом, и показывать только реализацию алгоритма решения задачи, опуская команды ввода данных и вывода результата.

Также не будем обозначать блоки начала и конца выполнения алгоритма. Подразумевается, что все пере-

численные блоки должны всегда присутствовать в алгоритме.

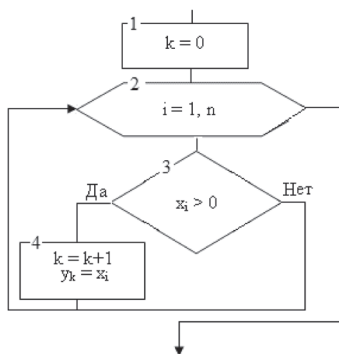
Пример 1.7. Сформировать новый одномерный массив из положительных элементов заданного массива.

Дано: n — размер массива произвольный; массив $X[n]$.

Найти: массив $Y[k]$.

Обозначим: i — текущий индекс элементов массива X , k — текущий индекс элементов массива Y .

Ясно, что для реализации алгоритма необходимо использовать цикл с заданным числом повторений, так как количество элементов в массиве X известно. Приведем фрагмент блок-схемы алгоритма



и ее словесное описание.

Блок 1. $k = 0$.

Блок 2. В цикле для всех значений параметра i от 1 до n выполняется тело цикла.

Блок 3. Если условие $X[i] > 0$ выполняется (выход «Да»), то вычисляется группа операторов в блоке 4:

$$k = k + 1; Y[k] = X[i].$$

Тест

Данные	Результат
$n = 5$ $X = (-1, 2, 0, 4, -3)$	$k = 2$ $Y = (2, 4)$

Выполнение алгоритма.

k	i	$x_i > 0?$	$y_k = x_i$
0	1	$x_1 > 0?$ «Нет»	
	2	$x_2 > 0?$ «Да»	
1			$y_1 = x_2 = 2$
	3	$x_3 > 0?$ «Нет»	
	4	$x_4 > 0?$ «Да»	
2			$y_2 = x_4 = 4$
	5	$x_5 > 0?$ «Нет»	
	6	Конец цикла	

Мы используем цикл с заданным числом повторений. Напомним, что такой цикл (арифметический цикл) применяется, когда число повторений цикла известно к началу его выполнения.

1.3.2. Вычисление суммы и количества элементов массива

Пример 1.8. Вычислить сумму элементов одномерного массива.

Дано: n — размер массива; массив $A = (a_1, a_2, \dots, a_n)$.

Найти: S — сумму элементов массива.

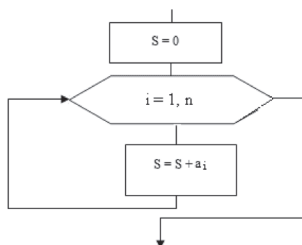
Начальное значение суммы равно нулю. В предыдущих подпараграфах мы говорили о том, что значение суммы накапливается с каждым шагом выполнения алгоритма. Вычисляем сумму по формуле $S = S + a_i$.

Тест

Данные		Результат
$N = 4$	$A = (3, 5, -2, 8)$	$S = 14$

Исполнение алгоритма

i	S
	0
1	$S + a_1 = 0 + 3 = 3$
2	$S + a_2 = a_1 + a_2 = 3 + 5 = 8$
3	$S + a_3 = a_1 + a_2 + a_3 = 8 - 2 = 6$
4	$S + a_4 = a_1 + a_2 + a_3 + a_4 = 6 + 8 = 14$



Фрагмент алгоритма

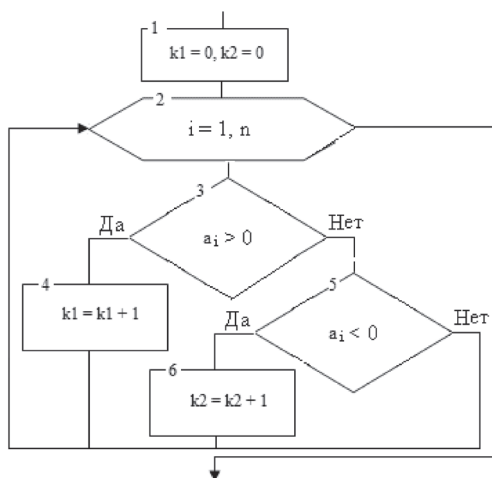
Пример 1.9. Найти количество положительных и отрицательных чисел в данном массиве.

Дано: n — размер массива; массив $A = (a_1, a_2, \dots, a_n)$. Обозначим $k1$ — количество положительных чисел, $k2$ — количество отрицательных чисел.

Найти: $k1, k2$ — количество положительных и отрицательных чисел массива.

Математическая модель. Пусть $k1 = 0$ и $k2 = 0$. Если $a[i] > 0$, то $k1 = k1 + 1$. Если $a[i] < 0$, то $k2 = k2 + 1$. Процесс повторяется до окончания просмотра всех чисел массива.

Приведем фрагмент блок-схемы алгоритма и ее словесное описание.



Блок 1. Задание начальных значений переменным $k1$ и $k2$.

Блок 2. Заголовок арифметического цикла.

Блок 3. Проверка условия. Если очередное значение элемента массива положительное (выход «Да» блока 3), то увеличиваем количество положительных чисел (блок 4).

Блок 5. Если очередное значение элемента массива отрицательное (выход «Да» блока 5), то увеличиваем количество отрицательных чисел массива (блок 6).

Указанные вычисления выполняем для всех n чисел массива.

В примере нам понадобилось два условных оператора, так как в массиве могут встретиться нулевые элементы, количество которых нам считать не надо.

1.3.3. Определение наибольшего элемента массива

Пример 1.10. Дан массив произвольной длины. Найти наибольший элемент массива и определить его номер.

Дано: n — размер массива; массив $A = (a_1, a_2, \dots, a_n)$.

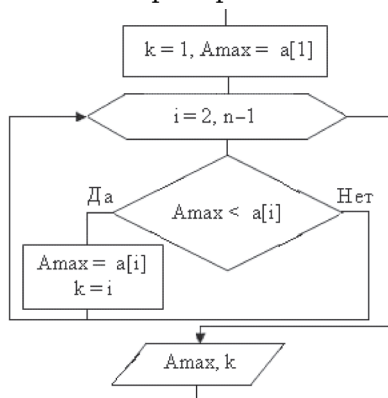
Найти: A_{\max} — наибольший элемент массива, k — его номер.

Математическая модель. Пусть $A_{\max} = a[1]$; $k = 1$. Если $A_{\max} < a[i]$, то $A_{\max} = a[i]$, $k = i$, для $i = 2, 3, \dots, n$.

Тест

Данные		Результат	
$n = 5$	$A = (3, -1, 10, 1, 6)$	$A_{\max} = 10$	$k = 3$

Приведем фрагмент блок-схемы алгоритма и его выполнение для тестового примера.



i	$i \leq n?$	$A_{\max} < A[i]?$	A_{\max}	k
			3	1
2	$2 \leq 5?$ «Да»	$-1 < 3?$ «Да»		
3	$3 \leq 5?$ «Да»	$10 < 3?$ «Нет»	10	3
4	$3 \leq 5?$ «Да»	$1 < 10?$ «Да»		
5	$5 \leq 5?$ «Да»	$6 < 10?$ «Да»		
6	$6 \leq 5?$ «Нет» (кц)			

1.3.4. Удаление и вставка элементов массива

Пример 1.11. Удалить из массива, в котором все элементы различны, наибольший элемент.

Дано: n — размер массива; $A[n]$ — массив вещественных чисел.

Найти: новый массив A размера $n - 1$.

Для решения задачи необходимо:

1) найти номер k наибольшего элемента массива. Эта задача решена в примере 1.10;

2) сдвинуть все элементы массива, начиная с k -го, на один элемент влево.

Для того чтобы разработать алгоритм, рассмотрим конкретный пример. Пусть дан одномерный массив, состоящий из 7 элементов:

$$A = (6, 3, 7, 11, 2, 8, 1).$$

Номер наибольшего элемента равен 4 ($k = 4$), т. е. начиная с четвертого элемента будем сдвигать элементы на один влево: четвертому элементу присвоим значение пятого, пятому — значение шестого, а шестому — значение седьмого. На этом сдвиг заканчивается. Таким образом, сдвиг начинается с k -го элемента и заканчивается элементом с номером n , где n — количество элементов в массиве. После сдвига массив будет такой: $A = (6, 3, 7, 8, 1, 1)$. Уменьшим количество элементов в массиве, так как после удаления количество элементов в массиве станет на один элемент меньше. Массив примет вид $A = (6, 3, 7, 8, 1)$.

В примере 1.10 уже рассматривался алгоритм поиска наибольшего элемента в одномерном массиве. Так как в данной задаче нас не интересует конкретное значение наибольшего элемента, то модифицируем рассмотренный алгоритм и сохраняем только номер наибольшего элемента.

Приведем фрагмент блок-схемы алгоритма и ее словесное описание.

Блок 1. Пусть первый элемент максимальный ($k = 1$).

Блок 2. В цикле просматриваем все элементы массива, начиная со второго до последнего.

Блок 3. Сравниваем элемент с номером k с очередным элементом массива.

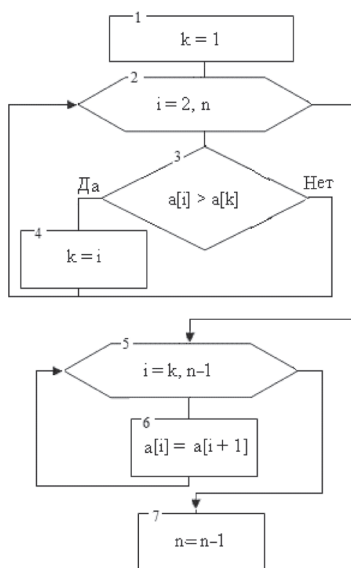
Блок 4. Запоминаем номер элемента, который на очередном шаге выполнения цикла больше k -го.

По окончании цикла (блок 2) переменная k сохранит номер наибольшего элемента.

Начинаю сдвиг.

Блок 5. В цикле, начиная с k -го элемента по $n - 1$ -й, выполняем присваивание (блок 6), тем самым заменяя элемент с номером i на элемент с номером $i + 1$.

Блок 7. Размер массива уменьшаем на 1, выполняя присваивание $n = n - 1$.



Мы уже рассматривали подробно выполнение алгоритма нахождения наибольшего элемента массива и его номера для тестового примера (см. пример 1.10). Покажем, как выполняется вторая часть (сдвиг элементов массива) для тестового примера.

Тест

Данные		Результат
$n = 7$	$A = (6, 3, 7, 11, 2, 8, 1)$	$A = (6, 3, 7, 2, 8, 1) \quad n = 6$

В массиве номер наибольшего элемента $k = 4$. Удаляем этот элемент.

i	$i \leq n - 1?$	$a_i = a_{i+1}$	Результат
4	$4 \leq 6?$ «Да»	$a_4 = a_5$	$A = (6, 3, 7, 2, 2, 8, 1)$
5	$5 \leq 6?$ «Да»	$a_5 = a_6$	$A = (6, 3, 7, 2, 8, 8, 1)$
6	$6 \leq 6?$ «Да»	$a_6 = a_7$	$A = (6, 3, 7, 2, 8, 1, 1)$
7	$7 \leq 6?$ «Нет» (кц)		
$n = 6$			$A = (6, 3, 7, 2, 8, 1)$

Пример 1.12. Вставить произвольное число в одномерный массив $A[n]$ после элемента с заданным номером.

Дано: n — размер массива; $A[n]$ — числовой вещественный массив; k — номер элемента, после которого вставляется число, равное m .

Найти: новый массив A размера $n + 1$.

Словесное описание алгоритма. Вставка нового элемента осуществляется следующим образом:

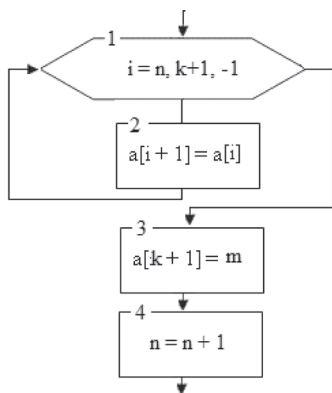
1) первые k элементов массива остаются без изменения;

2) все элементы, начиная с $(k + 1)$ -го необходимо сдвинуть вправо, чтобы освободить место для вставляемого элемента;

3) элементу с номером $(k + 1)$ присвоить значение m . Количество элементов массива увеличить на 1.

Рассмотрим пример. Пусть дан массив $A = (3, -12, 5, 14, 27)$, где $n = 5$.

Вставим элемент со значением 10 после второго элемента массива. Получим массив $A = (3, -12, 10, 5, 14, 27)$. Количество элементов $n = 6$.



Приведем фрагмент блок-схемы алгоритма и ее словесное описание.

Блок 1. Начинаем просматривать массив с конца, с n -го элемента до $(k + 1)$ -го.

Блок 2. В цикле сдвигаем все элементы массива вправо.

Блок 3. На освободившееся место вставляем новый элемент.

Блок 4. Увеличиваем количество элементов в массиве.

Тест

Данные		Результат
$n = 5; k = 2;$ $m = 10$	$A = (3, -12, 5, 14, 27)$	$A = (3, -12, 10, 5, 14, 27); n = 6$

Покажем выполнение алгоритма для тестового примера.

i	$a[i + 1] = a[i]$	$a[k + 1] = m$	n	Массив
5	$a[6] = a[5]$		5	$A = (3, -12, 5, 14, 27, 27)$
4	$a[5] = a[4]$			$A = (3, -12, 5, 14, 14, 27)$
3	$a[4] = a[3]$ (кц)			$A = (3, -12, 5, 5, 14, 27)$
		$a[3] = 10$	6	$A = (3, -12, 10, 5, 14, 27)$

В рассмотренных примерах мы использовали арифметический цикл, количество повторений которого всегда известно до работы начала алгоритма.

1.3.5. Определение первого элемента массива, имеющего заданное свойство

Пример 1.13. Определить, является ли заданная последовательность различных чисел a_1, a_2, \dots, a_n монотонно убывающей.

По определению последовательность монотонно убывает, если $a[i] \geq a[i + 1]$. Если хотя бы для одной пары чисел это условие нарушается, то последовательность не является монотонно убывающей. Следовательно, при построении алгоритма мы должны зафиксировать именно этот факт.

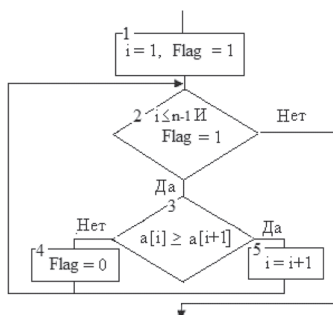
Пусть переменная *Flag* принимает значение равное 1, если последовательность является монотонно убывающей, и 0 — в противном случае.

Дано: n — размер массива; $A[n]$ — числовой вещественный массив.

Найти: $Flag = 1$, если последовательность монотонно убывает; $Flag = 0$ в противном случае.

Словесное описание алгоритма. Предположим, что последовательность монотонно убывает, и присвоим переменной *Flag* начальное значение, равное 1. В цикле последовательно сравниваем значения двух соседних элементов. Выход из цикла возможен в двух случаях:

- просмотрены все элементы заданной последовательности. Это означает, что условие $a[i] \geq a[i + 1]$ выполняется для всех пар соединений элементов, и последовательность является монотонно убывающей. Тогда $Flag = 1$;
- условие $a[i] \geq a[i + 1]$ не выполнилось для пары соседних элементов, тогда $Flag = 0$. Цикл прерывается. Сле-



довательно, последовательность не является монотонно убывающей.

Приведем фрагмент блок-схемы алгоритма и ее словесное описание.

Блок 1. Берем первое число последовательности $i = 1$. Предполагаем, что последовательность монотонно убывает ($Flag = 1$).

Блок 2. Цикл с предусловием. Пока не проверены все элементы и пока последовательность монотонно убывает (проверка условия в блоке 2), выполняется тело цикла (блоки 3–5).

Блок 3. Если последовательность не монотонная (выход «Нет»), то фиксируем это и $Flag = 0$ (блок 4).

Если условие не нарушается (выход «Да» блока 3), то присваиваем номеру элемента следующее значение (блок 5) и возвращаемся в цикл.

Система тестов

№ теста	Проверяемый случай	Данные		Результат
		n	Массив A	$Flag$
1	Является	3	(3, 2, 1)	1
2	Не является	3	(2, 3, 1)	0

Выполнение алгоритма.

№ теста	i	$Flag$	$(i \leq n - 1) \text{ и } (Flag = 1)?$	$a[i] > a[i + 1]?$
1	1	1	«Да»	$a[1] > a[2]?$ «Да»
	2		«Да»	$a[2] > a[3]?$ «Да»
	3		«Нет»	
			Последовательность монотонно убывает	
2	1	1	«Да»	$a[1] > a[2]?$ «Нет»
		0	«Нет»	
			Последовательность не монотонно убывающая	

Пример 1.14. Определить, есть ли в одномерном массиве хотя бы один отрицательный элемент.

Пусть переменная $Flag = 1$, если в массиве есть отрицательный элемент, и $Flag = 0$ — в противном случае.

Дано: n — размер массива; $A[n]$ — числовой вещественный массив.

Найти: $Flag = 1$, если отрицательный элемент есть; $Flag = 0$ — в противном случае.

Словесное описание алгоритма. Начинаем просматривать массив с первого элемента ($i = 1$). Пока не просмотрен последний элемент ($i \leq n$) и не найден отрицательный элемент ($a[i] \geq 0$), будем переходить к следующему элементу ($i = i + 1$). Таким образом, мы закончим просмотр массива в одном из двух случаев:

1) просмотрели все элементы и не нашли отрицательного, тогда $Flag = 0$ и $i > n$;

2) нашли нужный элемент, при этом $Flag = 1$.

Приведем фрагмент блок-схемы алгоритма и ее словесное описание.

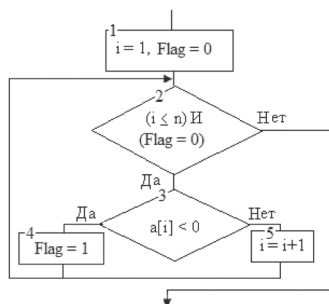
Блок 1. Берем первый элемент массива. Предполагаем, что в массиве нет отрицательных чисел и $Flag = 0$.

Блок 2. Пока не просмотрены все числа и пока не встретилось отрицательное число (выход «Да» блока 2) выполняем тело цикла (блоки 3–5).

Блок 3. Если встретилось отрицательное число (выход «Да» блока 3), запоминаем это и $Flag = 1$. Если очередное число положительное или равно нулю (выход «Нет» блока 3), то увеличиваем номер числа (блок 5) и переходим к следующему числу.

Система тестов

№ теста	Проверяемый случай	Данные		Результат
		n	Массив A	$Flag$
1	Есть	3	(3, -2, 1)	1
2	Нет	3	(2, 3, 1)	0



Исполнение алгоритма.

№ теста	i	$Flag$	$(i \leq n)$ и $(Flag = 0)$?	$a[i] < 0$?
1	1	0	$(1 \leq 3)$ и $(Flag = 0)$? «Да»	$a[1] < 0$? «Нет»
	2	0	$(2 \leq 3)$ и $(Flag = 0)$? «Да»	$a[2] < 0$? «Да»
		1	Найдено отрицательное число (кц)	
2	1	0	$(1 \leq 3)$ и $(Flag = 0)$? «Да»	$a[1] < 0$? «Нет»
	2	0	$(2 \leq 3)$ и $(Flag = 0)$? «Да»	$a[2] < 0$? «Нет»
	3	0	$(3 \leq 3)$ и $(Flag = 0)$? «Да»	$a[3] < 0$? «Нет»
	4	0	$(4 \leq 3)$ и $(Flag = 0)$? «Нет» (кц)	
			Отрицательных чисел нет	

1.4. АЛГОРИТМЫ, ИСПОЛЬЗУЮЩИЕ ДВУМЕРНЫЕ МАССИВЫ

1.4.1. Понятие двумерного массива

Понятие «двумерный массив» определим на примере. Пусть имеются данные о продажах некоторого товара по месяцам:

Месяц	Объем продаж, пар	Цена продажи, руб.	Себестоимость, руб.
1	4500	100	50
2	3900	110	55
3	3100	120	60

Таблица представляет собой множество из двенадцати однородных величин — это массив. Ее элементы расположены в 3 строки по 4 столбца в каждой.

Подобного рода таблицы из нескольких строк с равным числом элементов в каждой называют в информатике двумерными массивами или матрицей.

Двумерный массив определяется именем, числом строк и столбцов и обозначается, например, так: $A[n, m]$, где A — произвольное имя массива; n — число строк; m — число столбцов. Обратите внимание на то, что сначала всегда указывается количество строк, а потом — количество столбцов массива.

Если имеются данные о продажах за 3 мес., то нашу таблицу можно обозначить так: $A[3, 4]$, т. е. массив состоит из 3 строк и 4 столбцов.

Строки двумерных массивов нумеруются по порядку сверху вниз, а столбцы — слева направо.

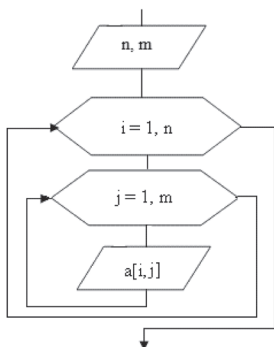
Элементы таблицы, представленной в примере, получают такие обозначения:

$$A[3,4] = \begin{pmatrix} A[1,1] & A[1,2] & A[1,3] & A[1,4] \\ A[2,1] & A[2,2] & A[2,3] & A[2,4] \\ A[3,1] & A[3,2] & A[3,3] & A[3,4] \end{pmatrix} = \begin{pmatrix} 1 & 4500 & 100 & 50 \\ 2 & 3900 & 110 & 55 \\ 3 & 3100 & 120 & 60 \end{pmatrix}.$$

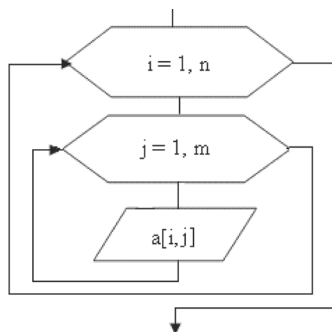
Каждый элемент двумерного массива определяется номерами строки и столбца, на пересечении которых он находится, и в соответствии с этим обозначается именем массива с двумя индексами: первый — номер строки, второй — номер столбца. Например, $A[1, 3]$ — элемент находится в первой строке и третьем столбце.

Таким образом, $A[1, 1] = 1$, $A[2, 3] = 110$ и т. д. Произвольный элемент двумерного массива мы будем обозначать $A[i, j]$, где i — номер строки, j — номер столбца.

Поскольку положение элемента в двумерном массиве описывается двумя индексами, алгоритмы для решения большинства задач с их использованием строятся на основе вложенных циклов. Обычно внешний цикл организуется по строкам массива, т. е. в нем выбирается требуемая строка, а внутренний — по столбцам, в котором выбирается элемент внутри строки.



Ввод двумерного массива



Вывод двумерного массива

В отличие от одномерных массивов для ввода и вывода данных в двумерные массивы необходимо использовать вложенные циклы. Циклы являются арифметическими, так как количество строк и столбцов известно.

Мы ввели и вывели элементы произвольного массива, имеющего n строк и m столбцов. Понятно, что при вводе необходимо задать количество строк и столбцов.

Напомним, что во всех задачах будем считать, что элементы массива заданы тем или иным способом, и показывать только реализацию алгоритма решения задачи, опуская команды ввода данных и вывода результата.

Также не будем обозначать блоки начала и конца выполнения алгоритма. Подразумевается, что все перечисленные блоки должны всегда присутствовать в алгоритме.

Следует понимать, что все алгоритмы, приведенные ранее для одномерных массивов, можно использовать и для двумерных массивов, применив вложенный цикл. Приведем алгоритмы решения некоторых задач.

1.4.2. Вычисление суммы элементов двумерного массива

Пример 1.15. Вычислить сумму элементов строк заданного двумерного массива.

Дано: n, m — количество строк и столбцов массива соответственно; массив $A[n, m]$.

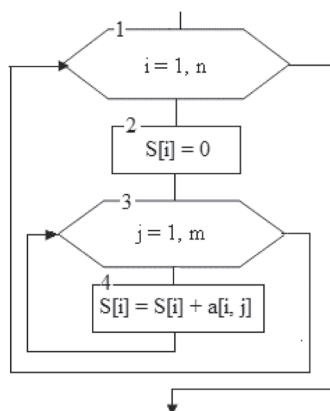
Найти: $S[n]$ — сумму элементов строк массива.

Тест

Данные			Результат
N	M	A	S
2	2	$\begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix}$	(3, 7)

Приведем фрагмент блок-схемы алгоритма и выполнение тестового примера.

Так как в массиве 2 строки, то сумма — это одномерный массив, состоящий из 2 элементов.



Исполнение алгоритма.

i	$i \leq n?$	j	$j \leq m?$	S
1	$1 \leq 2?$ «Да»			$S_1 = 0$
		1	$1 \leq 2?$ «Да»	$S_1 = S_1 + a_{1,1} = 0 + 2 = 2$
		2	$2 \leq 2?$ «Да»	$S_1 = S_1 + a_{1,2} = 2 + 1 = 3$
		3	$3 \leq 2?$ «Нет» (кц)	
2	$2 \leq 2?$ «Да»			$S_2 = 0$
		1	$1 \leq 2?$ «Да»	$S_2 = S_2 + a_{2,1} = 0 + 4 = 4$
		2	$2 \leq 2?$ «Да»	$S_2 = S_2 + a_{2,2} = 4 + 3 = 7$
		3	$3 \leq 2?$ «Нет» (кц)	
3	$3 \leq 2?$ «Нет» (кц)			

В этом примере мы нашли сумму элементов каждой строки матрицы. Количество таких сумм равно количеству строк матрицы, поэтому для сохранения значений сумм использован одномерный массив $S[n]$, количество элементов в котором равно количеству строк матрицы.

Обратите внимание на то, где присваивается начальное значение переменной S . Если находится сумма всех элементов матрицы, то команда $S = 0$ записывается перед началом внешнего цикла. В результате получается одно значение $S = 10$. Если ставится задача нахождения суммы элементов строк матрицы, то команда $S[i] = 0$ записывается внутри цикла по строкам матрицы, тогда результат представляет собой одномерный массив, в котором количество элементов равно количеству строк матрицы, и тогда $S = (3, 7)$. Если требуется найти сумму элементов столб-

цов матрицы, то команда $S[j] = 0$ записывается внутри цикла по столбцам матрицы, тогда результат представляет собой одномерный массив, в котором количество элементов равно количеству столбцов матрицы, и $S = (6, 4)$.

1.4.3. Вычисление наибольшего элемента двумерного массива

Пример 1.16. Найти наибольший элемент двумерного массива и его индексы.

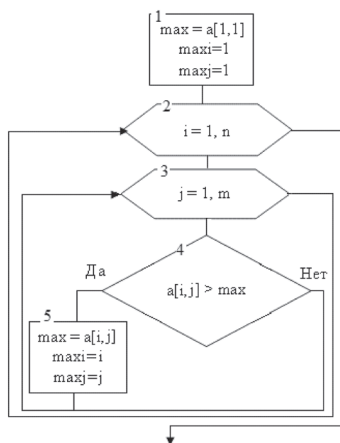
Дано: массив $A[n, m]$, где n, m — количество строк и столбцов массива соответственно.

Найти: \max — наибольший элемент массива, а также $\max i$ и $\max j$ — номер строки и столбца соответственно, на пересечении которых находится искомый элемент.

Словесное описание алгоритма. Пусть первый элемент двумерного массива является наибольшим. Запоминаем его значение и индексы. Сравниваем наибольшее значение со всеми оставшимися элементами. Если запомненное значение меньше очередного элемента массива, то запоминаем новое значение и его индексы.

Так как значения элементов в массиве могут повторяться, то договоримся, что будем запоминать только индексы первого наибольшего элемента.

Приведем фрагмент блок-схемы алгоритма и ее описание.



Блок 1. Присваиваем начальные значения переменным \max , $\max i$, $\max j$.

Блок 2. Берем очередную строку матрицы.

Блок 3. На i -й строке матрицы последовательно выбираем каждый элемент.

Блок 4. Сравниваем выбранный элемент со значением переменной \max .

Блок 5. Если значение очередного элемента больше \max (выход «Да» блока 4),

то запоминаем его значение и индексы. Возвращаемся в блок 3 и выбираем следующий элемент строки.

Блок 6. Если в строке нет элементов, то выбираем новую строку и повторяем вычисления (блоки 3–5).

Тест

Данные			Результат
n	m	A	
2	3	$\begin{pmatrix} 1 & 3 & 5 \\ 4 & 3 & 5 \end{pmatrix}$	$\max = 5$ $\max i = 1$ $\max j = 3$

Выполните тестирование алгоритма самостоятельно.

1.4.4. Вставка и удаление строк и столбцов двумерного массива

Мы уже рассматривали задачи вставки и удаления элементов в одномерном массиве. Обобщим эти алгоритмы на двумерный массив.

Пример 1.17. Вставить в двумерный массив строку, состоящую из нулей, после строки с номером k .

Для решения этой задачи необходимо:

- 1) первые k строк оставить без изменения;
- 2) все строки после k -й сдвинуть на одну вниз. Сдвиг лучше всего начать с последней строки и идти до $(k + 1)$ -й строки;
- 3) присвоить новые значения элементам $(k + 1)$ -й строки и увеличить количество строк в двумерном массиве.

Дано: массив $A[n, m]$, где n, m — количество строк и столбцов массива соответственно; k — номер строки, после которой вставляется новая строка.

Найти: новую матрицу A , содержащую $n + 1$ строку и m столбцов.

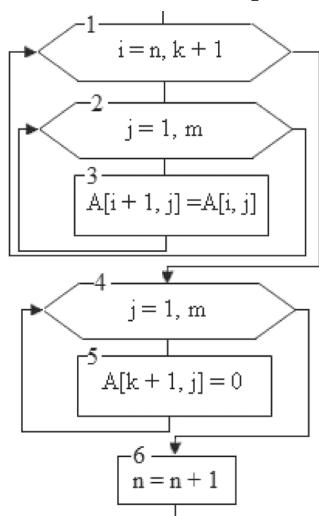
Приведем фрагмент блок-схемы алгоритма.

В блоке 3 в цикле выполняется сдвиг строк матрицы вниз, начиная со строки с номером k . В блоке 5 происходит вставка новой строки.

Тест

n	m	k	Исходная матрица	Результат
2	3	1	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 4 & 5 & 6 \end{pmatrix}$

Так как $k = 1$, то необходимо сдвинуть строку 2 вниз и вставить после первой строки новую строку, состоящую из нулей.



Исполнение алгоритма.

i	j	A
Сдвигаем строки матрицы вниз (блоки 1–3)		
2	1	$A[3, 1] = A[2, 1] = 4$
	2	$A[3, 2] = A[2, 2] = 5$
	3	$A[3, 3] = A[2, 3] = 6$
Промежуточный результат		
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 4 & 5 & 6 \end{pmatrix}$		
Вставка строки		
2	1	$A[2, 1] = 0$
	2	$A[2, 2] = 0$
	3	$A[2, 3] = 0$

Примечания

1. Если необходимо вставить новую строку после строки, удовлетворяющей некоторому условию, то надо найти номер этой строки — и задача сведется к решению уже рассмотренной.

2. Если надо вставить новые строки после всех строк с заданным условием, то надо учесть это при описании массива. Заметим, что удобнее рассматривать строки, начиная с последней строки, и после вставки очередной строки увеличивать количество строк.

3. Вставка новой строки перед строкой с номером k изменится только тем, что сдвигать назад надо не с $(k + 1)$ -й строки, а с k -й.

4. Если надо вставлять столбцы, то размерность массива увеличивается по столбцам, а все остальное практически не меняется: надо сдвинуть столбцы вправо и на данное место записать новый столбец.

Пример 1.18. Удалить из двумерного массива строку с номером k .

Для удаления строки с номером k необходимо: