«Национальный открытый институт г.Санкт-Петербург»

Боброва Л.В., Рыбакова Е.А

Информатика. Основы алгоритмизации

Методические указания к выполнению практических работ

Рекомендовано Методической комиссией по качеству
Национального открытого института
для студентов, обучающихся по специальности
23.07.01 – Прикладная информатика (по отраслям)

Санкт-Петербург
2016

УДК 004
ББК 32.97
Б72

Методические указания разработаны на основе рабочей программы дисциплины “Работа в приложении Microsoft Office PowerPoint” для специальности Прикладная информатика (по отраслям) в соответствии с требованиями Федерального государственного образовательного стандарта среднего профессионального образования для подготовки специалистов квалификации «Техник-программист».

УДК 004
ББК 32.97

Введение

При изучении дисциплины «Информатика» студенты приобретают навыки пользования не только стандартными прикладными программными средствами, но и учатся выполнять все последовательные этапы решения задач на компьютере: правильно ставить задачу, определять технические и программные средства ее решения, при необходимости – самостоятельно разрабатывать дополнительные программные компоненты на языках высокого уровня. Важнейшим этапом освоения технологии решения задач на компьютере является развитие навыков разработки алгоритмов, их правильного представления в соответствии с общепринятыми стандартами, знание базовых алгоритмических структур.

Для повышения эффективности применения компьютера как инструмента решения задач необходимо освоение основной фундаментальной концепции подхода к использованию цифровых вычислительных средств. В информатике таким фундаментом является алгоритмизация возникающих задач. Алгоритмический стиль мышления позволяет связать воедино функционирование информации в конкретной прикладной среде с требованиями обработки информации на компьютере.

Алгоритмическое мышление помогает сформировать следующие основные навыки решения задач:

- умение правильно планировать структуру предстоящих действий для достижения заданной цели при помощи стандартного набора средств;
- строить информационные структуры для описания объектов и процессов в конкретной предметной области;
- правильно организовывать поиск информации, необходимой для решения задачи;
- четко и однозначно формулировать способ решения задачи в общепринятой форме и правильно понимать способ решения, предложенный другим разработчиком;
- формировать навыки анализа имеющейся информации, умения представлять ее в структурировном виде.

Развитие системного, логического мышления, освоение навыков оперирования формальными понятиями и объектами,

характерными для используемых информационных технологий – это тот необходимый уровень, который формируется на базе правильного понимания алгоритмов и способов алгоритмизации.

1. ПОНЯТИЕ АЛГОРИТМА

Происхождение термина «алгоритм». Слово произошло от имени среднеазиатского ученого Аль-Хорезми (8–9 вв., Хорезм – историческая область на территории современного Узбекистана). В 1857 в библиотеке Кембриджского университета был найден перевод на латинский язык математической работы Аль-Хорезми, в котором имя Аль-Хорезми упоминается как Алгоритми, откуда и появилось слово «алгоритм». Оно стало особенно употребительным с появлением компьютеров для обозначения совокупности действий, составляющих какой-либо вычислительный процесс.

Термин «алгоритм» в бытовом понимании. В повседневной жизни выполнение каждой, даже простой задачи обычно осуществляется в несколько последовательных этапов (шагов). Например, такую цепочку шагов описывает инструкция для получения наличных денег со счета по банковской карте. Подобную инструкцию - четкую последовательность шагов в решении какой-либо жизненной задачи принято называть алгоритмом. Каждое отдельное действие – это шаг алгоритма.

Формализация понятия алгоритма, теория алгоритмов. Приведенное выше общее описание понятия алгоритма не всегда позволяет сравнить какие-либо две таким образом определенные инструкции. Можно, например, сравнивать два алгоритма решения системы уравнений и выбрать из них более подходящий, но невозможно сравнить алгоритм перехода через улицу с алгоритмом извлечения квадратного корня. С этой целью нужно формализовать понятие алгоритма, т.е. отвлечься от существа решаемой алгоритмом задачи и выделить свойства алгоритмов, привлекая к рассмотрению только форму их записи.

Задача нахождения единообразной формы записи алгоритмов, решающих различные задачи, является одной из важнейших в теории алгоритмов. В этой теории предполагается, что каждый шаг алгоритма должен быть таков, что его может выполнить достаточно

простое устройство (например - машина). Так, для уточнения понятия «алгоритм» и получения возможности математического исследования алгоритмов в 30-х г.г. XX века были предложены абстрактные вычислительные машины - машина Поста и машина Тьюринга. Эти механизмы, обладающие свойствами универсальности и простоты своей логической структуры, позволили решить проблему существования алгоритма решения любой практической задачи. В частности, было доказано, что если для решения задачи можно построить машину Поста-Тьюринга, то такая задача алгоритмически разрешима. Также была доказана возможность существования математических задач, для решения которых вообще не может существовать никакой алгоритм.

Вычислительный алгоритм. Это - упорядоченный набор основных математических и логический действий, однозначно определяющий процесс перехода от допустимых исходных данных задачи к конечному результату ее решения и обладающий свойствами массовости, конечности, определенности, детерминированности. Основные особенности вычислительных алгоритмов состоят в следующем.

Массовость – это возможность применять многократно один и тот же алгоритм к различным вариантам его исходных данных. Алгоритм служит, как правило, для решения не одной конкретной задачи, а некоторого класса задач. Так алгоритм сложения столбиком применим к любому конечному набору натуральных чисел. Для каждого алгоритма есть некоторое множество объектов, допустимых в качестве исходных данных. Например, в алгоритме деления вещественных чисел делимое может быть любым, а делитель – тоже любым, но за исключением нуля.

Конечность - это обязательное наличие искомого результата после завершения алгоритма либо четкая фиксация причины, по которой результат не мог быть получен. Следует отметить, что на практике встречаются примеры формально бесконечных алгоритмов, например - алгоритм обработки данных системы абонентских пунктов банка. Этот алгоритм состоит в непрерывном повторении цепочки действий («идентифицировать клиента», «зафиксировать запрос клиента на транзакцию», «проверить допустимость транзакции», «выполнить транзакцию» и т.д.), выполняемых с определенной частотой (через каждую секунду,

минуту, час) во все время функционирования данной банковской системы.

Определенность – это наличие на каждом шаге алгоритма у исполнителя достаточной информации для того, чтобы его можно было выполнить. Исполнителю также нужно четко знать, каким именно образом этот шаг выполняется. Сами шаги инструкции должны быть достаточно простыми, а исполнитель должен однозначно понимать смысл каждого из действий, составляющих алгоритм (например, при вычислении корней квадратного уравнения он должен понимать смысл вычисления дискриминанта).

Детерминированность – это отсутствие элементов случайности при выполнении алгоритма. При многократном применении алгоритма к одним и тем же исходным данным должен получаться всегда один и тот же результат. Поэтому, например, процесс преобразования информации, в котором участвует бросание монеты, не может быть назван алгоритмом.

2. ФОРМЫ ПРЕДСТАВЛЕНИЯ АЛГОРИТМОВ

Для записи алгоритма могут использоваться различные формы его представления.

Вербальная форма представления алгоритма предполагает запись алгоритма на русском языке (или любом другом естественном языке) в виде последовательности пронумерованных инструкций. Как правило, эта форма записи алгоритма громоздка, неудобна и недостаточно наглядна.

Например, в ней описание алгоритма нахождения НОД (наибольшего общего делителя) двух целых положительных чисел *m* и *n* может быть представлено в виде последовательности следующих четырех шагов:

Шаг 1: Сравнить *m* и *n*.

Шаг 2: Если *m* равно *n*, то *m* и есть исходный НОД, расчет окончен. Иначе перейти к шагу 3.

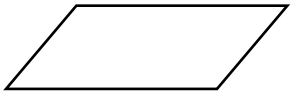
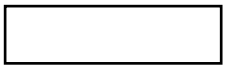
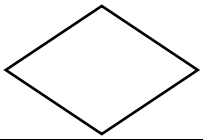
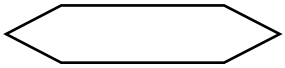
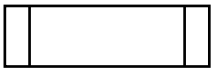
Шаг 3: Если *m* больше *n*, то уменьшить значение *m* на величину *n* и вернуться к шагу 1. Иначе перейти к шагу 4.

Шаг 4: Уменьшить значение *n* на величину *m* и вернуться к шагу 1.

Эта современная запись алгоритма нахождения НОД - весьма упрощенная. Запись, данная первоначально Евклидом, заполняет целую страницу текста, причем последовательность элементарных действий там значительно сложней.

Представление алгоритма в форме блок-схемы реализуется в виде набора геометрических элементов (блоков), соединенных стрелками. Каждый блок – это «шаг» алгоритма, его отдельное действие. Направление стрелок между блоками задает последовательность действий. В табл.1 представлены основные стандартные элементы блок-схем.

Таблица 1

Название элемента	**Обозначение**	**Пояснение**
Пуск-останов		Начало или конец вычислений. Внутри фигуры пишут: «начало» или «конец» соответственно
Ввод-вывод		Операция ввода-вывода данных
Процесс		Вычислительное действие или последовательность действий
Решение		Проверка условия, указанного внутри фигуры
Модификатор		Заголовок цикла с параметром
Предопределенный процесс		Вычисления по подпрограмме
Документ		Вывод данных на печать
Линия перехода		Соединяет между собой блоки, указывая очередность их выполнения

Элементы блок-схемы различаются по своему внешнему виду и назначению. Так, элементы, содержащие инструкции по каким-либо преобразованиям величин, обозначаются прямоугольниками, а элементы, содержащие проверку условий – ромбами. Операции ввода-вывода данных обозначаются параллелограммами. Начало и конец алгоритма обозначаются прямоугольниками с двумя скругленными противоположными сторонами, внутри которых пишут: «начало» или «конец» соответственно.

Из прямоугольника выходит единственная стрелка, входить в него может несколько стрелок. Из обеих острых вершин ромба выходят стрелки: одна из них помечается словом «да», другая – словом «нет», они задают дальнейшее направление вычислений в случае, соответственно, выполнения или невыполнения записанного внутри ромба условия.

Назначение и правила использования всех типов блоков приведены в табл.1 в графах Название элемента и Пояснение.

С использованием приведенных в табл.1 стандартных элементов вышеуказанный алгоритм нахождения НОД двух положительных целых чисел *m* и *n* примет вид блок-схемы на рис.1. Внутри блоков в ней знаком := обозначена операция присваивания переменной величине, указанной слева от знака, ее значения, вычисляемого по формуле, стоящей справа от знака.

Блок-схемы наиболее удобно использовать для отображения уже разработанных, готовых алгоритмов, а также на начальных стадиях изучения программирования, так как они позволяют подключить к восприятию алгоритма наглядные зрительные образы производимых операций. Кроме того, на стадии проектирования сложных программ эта форма представления алгоритмов используется при формировании верхних уровней в иерархической структуре программ, а также - на нижних уровнях, если не полностью определены средства описания программ.

На стадиях детальной разработки сложных алгоритмов использование блок-схем оказывается неэффективным, так как запутанное переплетение многочисленных соединительных стрелок затрудняет понимание разрабатываемого алгоритма.

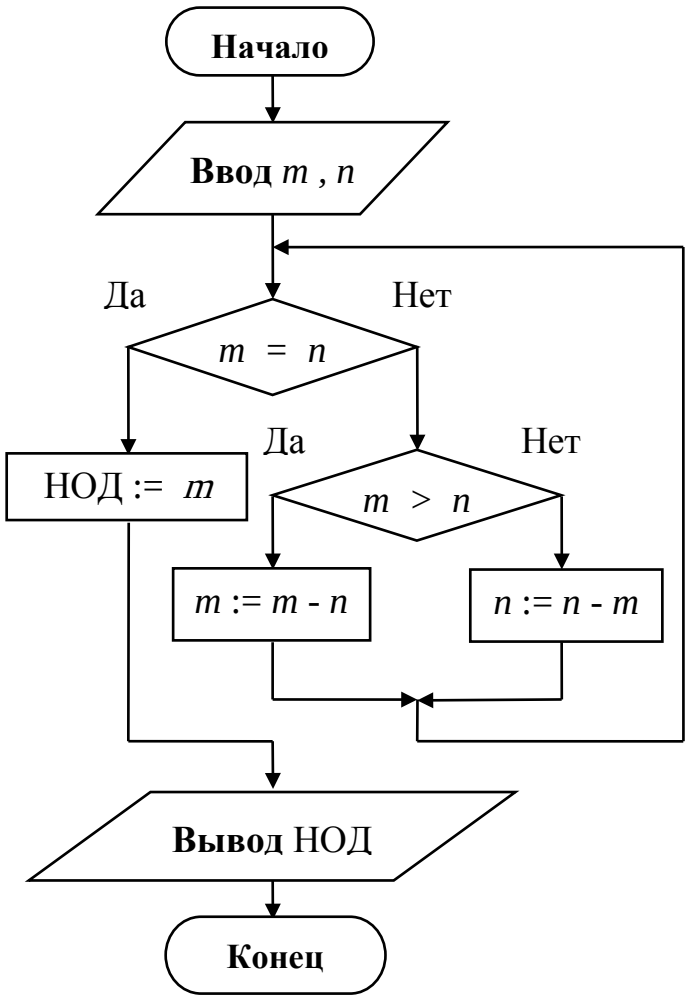


Рис. 1. Блок-схема алгоритма нахождения НОД

Представление алгоритма в форме псевдокода основано на описании этапов решения задачи с помощью ограниченного набора стандартных синтаксических конструкций. В псевдокоде, в частности, могут использоваться отдельные инструкции формальных алгоритмических языков программирования. Например, возведение X в степень A обозначается как X**A, извлечение квадратного корня из X обозначается как Sqrt (X).

Так же, как в формальных языках программирования, в псевдокоде присутствуют ключевые слова, смысл которых заранее оговорен и фиксирован. В табл.2 приведен базовый перечень ключевых слов.

В качестве ключевых могут использоваться также соответствующие английские слова-аналоги: **else** вместо **иначе**, **then** вместо **то**, и т.д.

В самом общем виде запись алгоритма в форме псевдокода выглядит следующим образом:

алг *название алгоритма* **(***аргументы - входные параметры и параметры - результаты работы алгоритма***)**

дано | *условия применимости алгоритма*

надо | *цель выполнения алгоритма*

нач *описание промежуточных (внутренних) величин алгоритма*

команда 1;

команда 2;

…

команда N

кон

Здесь часть записи от слова **алг** до слова **нач** называется заголовком алгоритма, а часть, заключенная между словами **нач** и **кон** - его телом. Внутри тела также могут встречаться слова **нач** и **кон,** группа команд между ними образует свой отдельный блок команд. Команды отделяются друг от друга символом «;».

Сразу после названия алгоритма в круглых скобках указываются характеристики (**арг** или **рез**) и тип значения (**цел, вещ, сим, лог** или **таб**) всех входных (**арг**) и выходных (**рез**) величин. При описании массивов служебное слово **таб** дополняется граничными парами значений по каждому индексу элементов массива.

В конце любой строки в текст псевдокода после знака «|» можно вставлять комментарий, облегчающий понимание алгоритма.

Разделы **дано** и **надо** в записи алгоритма не являются обязательными.

Таблица 2

Ключевое слово	**Пояснение**
алг	Начало описания алгоритма
арг	Входные данные (аргументы) алгоритма
вещ	Вещественный тип данных
все	Конец команды ветвления вычислений
да	Логическая константа «TRUE»
дано	Описание условий применимости алгоритма
до	Конечное значение параметра цикла

Ключевое слово	Пояснение
и	Логическая связка «И»
или	Логическая связка «ИЛИ»
иначе	Часть конструкции **если … то … иначе …**
кон	Конец всего алгоритма или блока команд
кц	Конец цикла
лит	Символьный (литерный) тип данных
лог	Логический (булевский) тип данных
надо	Описание цели алгоритма
нач	Начало тела алгоритма или блока команд
нет	Логическая константа «FALSE»
нц	Начало цикла
от	Начальное значение параметра цикла
рез	Выходные данные (результаты) алгоритма
таб	Табличный тип данных (массив данных)
то	Часть конструкции **если … то … иначе**
цел	Целый тип данных
шаг	Шаг изменения параметра цикла

К числу основных действий, составляющих тело алгоритма, относятся команды ввода-вывода, присваивания, перехода, ветвления и циклов. Для обозначения этих команд используются соответствующие ключевые слова.

Команда ввода: ключевое слово **ввод,** за которым указываются имена переменных, для которых вводятся значения. Например, команда **ввод** a,b,c означает ввод значений соответственно для переменных a,b,c.

Команда вывода: ключевое слово **вывод,** за которым следуют имена выводимых переменных, выводимые выражения и тексты (тексты помещаются в кавычки). Например, команда **вывод** "S = ", S означает вывод имени переменной S, за которым после знака равенства = следует вывод текущего значения этой переменной.

Команды присваивания используются для вычисления выражений и присваивания их значений переменным. Общий вид команды: "переменная" := "выражение", где знак ":=" означает команду замены прежнего значения переменной, стоящей в левой части, на вычисленное значение выражения, стоящего в правой части. Например, команда x := y + z означает присваивание переменной x значения суммы величин y и z, а команда k := k+1 означает увеличение текущего значения переменной k на единицу.

Команда перехода: ключевые слова **идти к,** после которых указывается номер той строки, команда которой должна выполняться следующей. Таким образом, команда перехода изменяет естественную последовательность выполнения команд по возрастанию номеров строк. Например, записанная в 5-й строке команда **идти к** 10 означает, что далее должна выполняться команда, записанная в 10-й строке, а не команда, записанная в следующей 6-й строке.

Для организации ветвлений вычислительного процесса применяются команды, начинающиеся с ключевых слов **если** и **выбор,** циклические вычисления организуются с помощью команд, начинающихся с ключевых слов **пока** и **для.** Подробно эти команды рассмотрены далее при описании базовых структур алгоритмов.

С использованием псевдокода вышеуказанный алгоритм нахождения НОД двух положительных целых чисел *m* и *n* примет следующий вид:

```
1. алг Наибольший Общий Делитель (арг цел m, n, рез цел nod)
2.    дано | положительные целые числа m, n
3.    надо | nod – наибольший общий делитель чисел m, n
4. нач
5.    ввод m, n;
6.    если m = n то идти к 9 все;
7.    если m > n то m := m - n иначе n := n – m все;
8.    идти к 6;
9.    nod := m;
10.      вывод “Значение НОД равно”, nod
11.   кон
```

Эта форма представления облегчает запись алгоритма на стадии его проектирования и дает возможность в дальнейшем легко перевести алгоритм в более широкий набор команд формального языка программирования.

3. БАЗОВЫЕ СТРУКТУРЫ АЛГОРИТМОВ

Алгоритм решения любой вычислительной задачи можно описать, используя комбинации из следующих трех стандартных базовых конструкций алгоритмов:

- **последовательного алгоритма** – алгоритма линейной структуры;
- **ветвящегося алгоритма** – алгоритма разветвляющейся структуры;
- **циклического алгоритма** – алгоритма циклической структуры.

3.1. Алгоритм линейной структуры

Алгоритм линейной структуры – это объединение всех действий в единую цепь, в которой каждое последующее действие строго и однозначно следует за предыдущим действием (рис.2). На языке псевдокода соответствующая последовательность команд запишется в виде:

команда 1; команда 2; . . . ; команда N;

3.2. Алгоритм разветвляющейся структуры

Алгоритм разветвляющейся структуры – содержит проверку одного либо нескольких условий, по результатам которой происходит переключение на один из возможных двух либо один из возможных нескольких вариантов дальнейшего развития процесса. Различают четыре вида ветвящегося алгоритма.

Ветвление «**если-то-иначе»** (рис.3) встречается на практике наиболее часто. В нем присутствует проверка одного условия, по результатам которой происходит выбор между двумя возможными цепочками дальнейших действий.

На языке псевдокода соответствующая последовательность команд запишется в виде:

если *Условие*
то *команда 1; . . . ; команда M*
иначе *команда 2; . . . ; команда N*
все;

Ветвление «**если-то»** (рис.4) позволяет при соблюдении проверяемого в нем условия выполнить заданную цепочку действий: *команды 1…N*. При несоблюдении указанного условия *команды 1…N* пропускаются и сразу начинает выполняться следующая за ветвлением *команда N+1*.

На языке псевдокода вариант «**если-то»** запишется в виде:

если *Условие*
то *команда 1; . . . ; команда N*
все;
команда N+1;

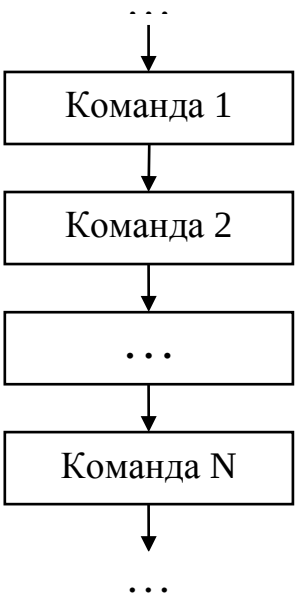


Рис. 2. Алгоритм линейной структуры

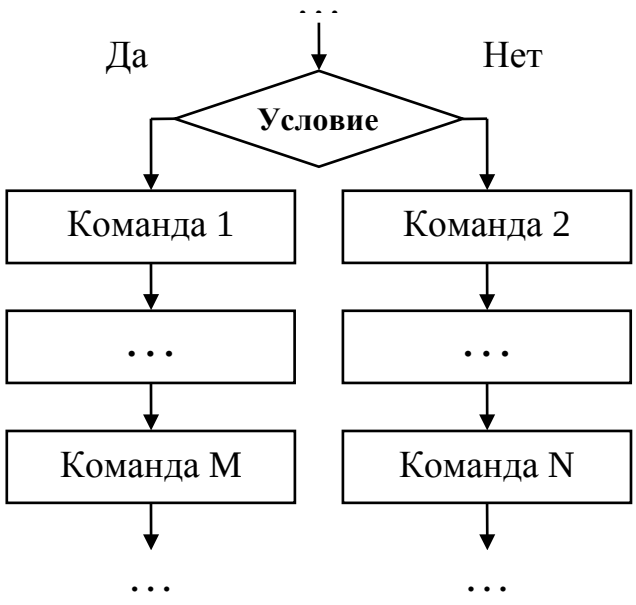


Рис. 3. Алгоритм разветвляющейся структуры (вариант «если-то-иначе»)

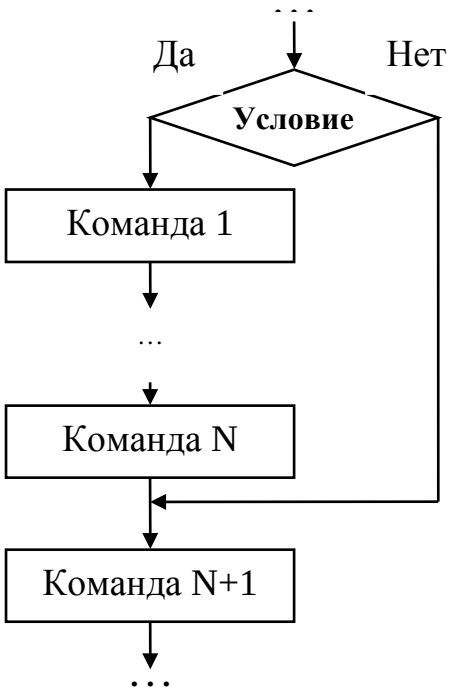


Рис. 4. Алгоритм разветвляющейся структуры (вариант «если-то»)

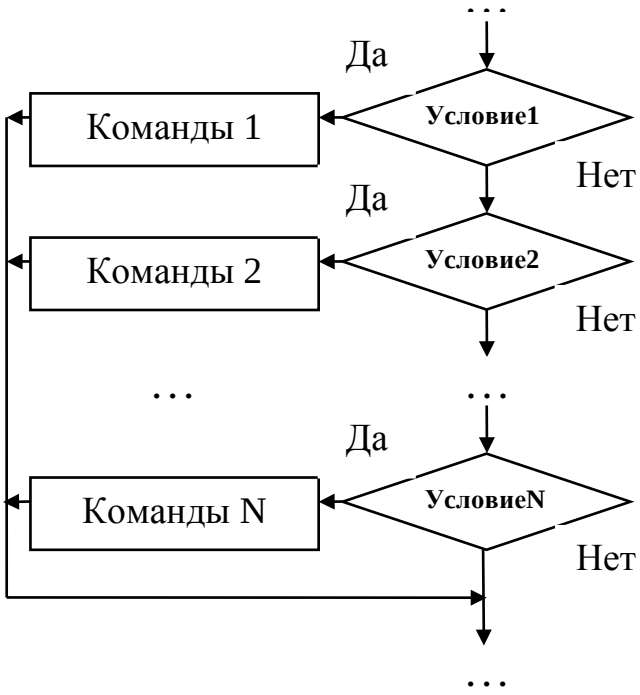


Рис. 5. Алгоритм разветвляющейся структуры (вариант «выбор»)

Ветвление «**выбор**» (рис.5) позволяет выбрать одну из N имеющихся альтернатив - цепочек команд. Для каждой альтернативы сначала проверяется соответствующее ей условие, срабатывает та первая альтернатива, у которой оно выполнится. Если не выполнится ни одно из условий выбора – ни одна из N цепочек команд не сработает, а управление перейдет к следующей после ветвления команде.

На языке псевдокода вариант «**выбор**» запишется в виде:

выбор
  при *условие 1: команды 1*
  при *условие 2: команды 2*
  . . . . . .
  при *условие N: команды N*
все. То

Ветвление «**выбор-иначе»** (рис.6) предусматривает проверку условий только у первых N-1 альтернатив, а у последней N-й цепочки команд условие отсутствует. Если не сработает ни одна из первых N-1 альтернатив, то тогда автоматически выполнится N-я цепочка команд. Таким образом, в отличие от ветвления «**выбор»,** здесь обязательно произойдет срабатывание одной из имеющихся N цепочек команд.

На языке псевдокода вариант «**выбор-иначе»** запишется в виде:

выбор
  при *условие 1: команды 1*
  при *условие 2: команды 2*
  . . . . . .
  при *условие N-1: команды N-1*
  иначе *команды N*
все;

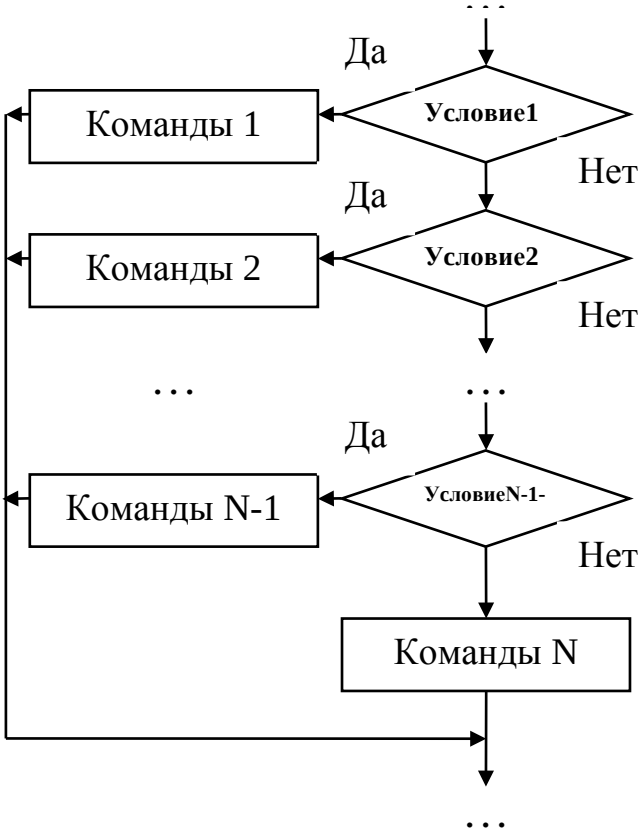


Рис. 6. Алгоритм разветвляющейся структуры (вариант «выбор-иначе»)

3.3. Алгоритм циклической структуры

Алгоритм циклической структуры обеспечивает повторение операции или группы операций при выполнении некоторого условия, называемого условием цикла. Такое повторение может быть многократным, но не должно быть бесконечным. Если повторение продолжается сколь угодно много раз, то говорят о зацикливании алгоритма. При реализации на компьютере зацикливание приводит к необходимости прервать цикл, не дожидаясь его формального завершения.

На рис.7 представлены цикл с предусловием ***(а)*** и цикл с постусловием ***(б)***. Повторяющееся тело цикла составляют команды 1…N, команда N+1 в цикл не входит. В цикле с предусловием тело цикла не выполнится ни разу, если первая проверка условия цикла покажет его несоблюдение. В цикле с постусловием тело цикла обязательно выполнится хотя бы один раз.

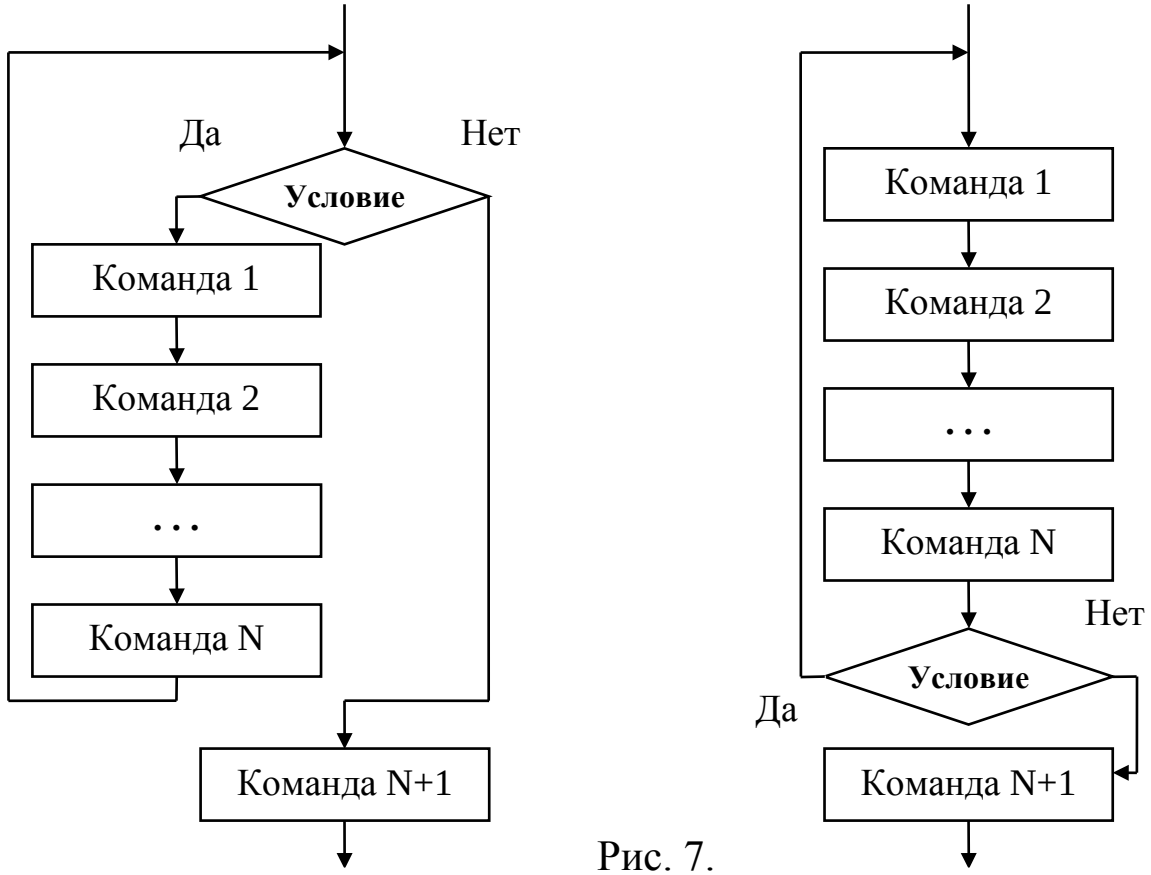


Рис. 7. Алгоритмы циклической структуры

На языке псевдокода циклы с предусловием или с постусловием отображаются соответственно следующими вариантами команды **пока**:

(а)
нц пока *Условие*
команда 1; команда 2; . . . ; команда N
кц;
команда N+1;

(б)
нц *команда 1; команда 2; . . . ; команда N*
пока *Условие*
кц;
команда N+1;

В тех случаях, когда число повторов выполнения тела цикла заранее известно, для отображении цикла удобно использовать блок модификатор (табл.1), в котором размещается заголовок цикла (рис.8).

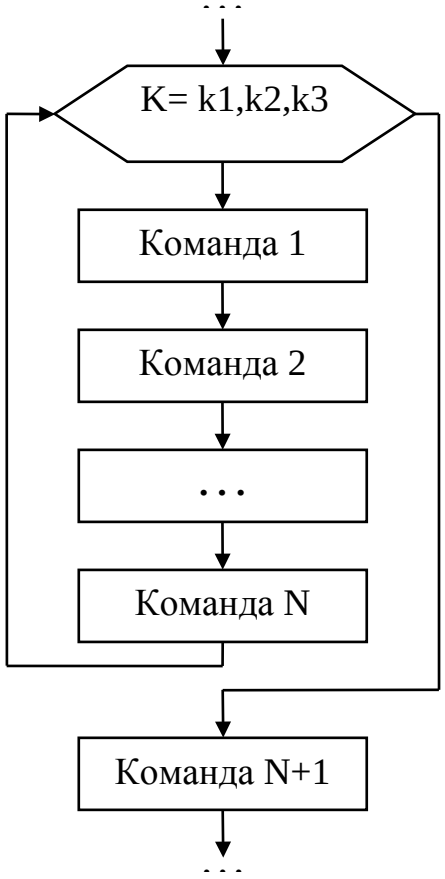


Рис. 8. Цикл с параметром

В заголовке цикла использованы следующие обозначения:

K – имя параметра цикла - переменной, выполняющей функцию счетчика числа повторов выполнения тела цикла;

k1 – начальное значение параметра цикла;

k2 – конечное значение параметра цикла;

k3 – шаг изменения параметра цикла после очередного выполнения тела цикла. Если этот параметр не указан, шаг изменения полагается равным 1.

Тело цикла составляют команды 1…N.

Такой тип цикла называется циклом с параметром.

На языке псевдокода цикл с параметром отображается с помощью команды **для:**

Нц

для К **от** k1 **до** k2 **шаг** k3

команда 1; команда 2; . . . ; команда N

кц;

команда N+1;

Контрольные вопросы

1. Каково происхождение слова «алгоритм»?
2. Какой цели служит формализация понятия «алгоритм»?
3. Каково определение понятия «вычислительный алгоритм»?
4. В чем состоит свойство массовости вычислительного алгоритма?
5. В чем состоит свойство конечности вычислительного алгоритма?
6. В чем состоит свойство определенности вычислительного алгоритма?
7. В чем состоит свойство детерминированности вычислительного алгоритма?
8. Каковы формы представления вычислительного алгоритма?
9. В чем заключается вербальная форма записи алгоритма, каковы ее недостатки?
10. Что собой представляет запись алгоритма в форме блок-схемы?
11. Какие геометрические фигуры используются для обозначения блоков на блок-схемах алгоритмов, какой тип действий связан с каждой из фигур?
12. В чем состоят преимущества и недостатки представления алгоритма в виде блок-схемы?
13. Что собой представляет запись алгоритма в форме псевдокода?
14. В чем состоит назначение основных ключевых слов псевдокода?
15. Какие разделы в записи псевдокода являются обязательными?
16. На какой стадии разработки алгоритма эффективно использование псевдокода?

17. Какая форма представления алгоритма в наибольшей степени соответствует его реализации в виде компьютерной программы?

18. Каковы базовые структуры вычислительных алгоритмов?

19. Какой алгоритм называется последовательным?

20. Какой алгоритм называется ветвящимся?

21. Каким образом реализуется стандартная конструкция ветвления «**если-то**»?

22. Каким образом реализуется стандартная конструкция ветвления «**если-то-иначе**»?

23. Каким образом реализуется стандартная конструкция ветвления «**выбор**»?

24. Каким образом реализуется стандартная конструкция ветвления «**выбор-иначе**»?

25. В каких случаях результатом ветвления становится пропуск команды или блока команд?

26. Какой алгоритм называется циклическим?

27. Каким образом реализуется цикл с предусловием?

28. Каким образом реализуется цикл с постусловием?

29. Каким образом реализуется цикл с параметром?

30. В каком из базовых типов цикла возможна ситуация, когда ни разу не выполнится тело цикла?

31. В каком из базовых типов цикла число повторов выполнения тела цикла точно задается заранее?

32. В каких базовых типах циклических алгоритмов возможно зацикливание?

Контрольные задания по составлению алгоритмов

Содержание заданий: составить алгоритм решения предлагаемой задачи на основе использования трех стандартных базовых конструкций алгоритмов: линейной, ветвящейся, циклической. Алгоритм представить в виде блок-схемы и в виде псевдокода. В тексте псевдокода привести комментарии, поясняющие смысл используемых команд.

Примеры выполнения контрольных заданий

Пример 1.

Ввести три различных вещественных числа a, b, c и определить среди них максимальное и минимальное значения.

Решение. При составлении алгоритма решения задачи целесообразно воспользоваться стандартной конструкцией ветвления «**выбор-иначе**».

Всего имеется 7 альтернативных вариантов расчета, из которых первые 6 связаны соответственно с выполнением условий a>b>c, a>c>b, b>a>c, b>c>a, c>a>b, c>b>a, а последний 7-й вариант отражает случай ввода некорректных данных, когда среди величин a, b, c по крайней мере для двух были введены одинаковые значения.

Блок-схема алгоритма решения задачи представлена на рис.9. Соответствующий текст псевдокода с комментариями имеет вид:

```
1.  алг Макс и мин трех чисел (арг вещ a, b, c; рез вещ maxz, minz)
2.  дано | a,b,c – все значения различны
3.  надо | maxz= максимум из a,b,c ; minz= минимум из a,b,c
4.  нач
5.    ввод a,b,c ; | ввод исходных данных
6.      выбор
7.        при a>b и b>c : maxz:=a; minz:=c | случай a>b>c
8.        при a>c и c>b : maxz:=a; minz:=b | случай a>c>b
9.        при b>a и a>c : maxz:=b; minz:=c | случай b>a>c
10.       при b>c и c>a : maxz:=b; minz:=a | случай b>c>a
11.       при c>a и a>b : maxz:=c; minz:=b | случай c>a>b
12.       при c>b и b>a : maxz:=c; minz:=a | случай c>b>a
13.       иначе вывод "Ошибка ввода данных"; идти к 17 | ошибка:
14.     | имеются одинаковые введенные значения для a,b,c
15.     все
16.   вывод "max=", maxz, "min=", minz | вывод результатов расчета
17.  кон
```

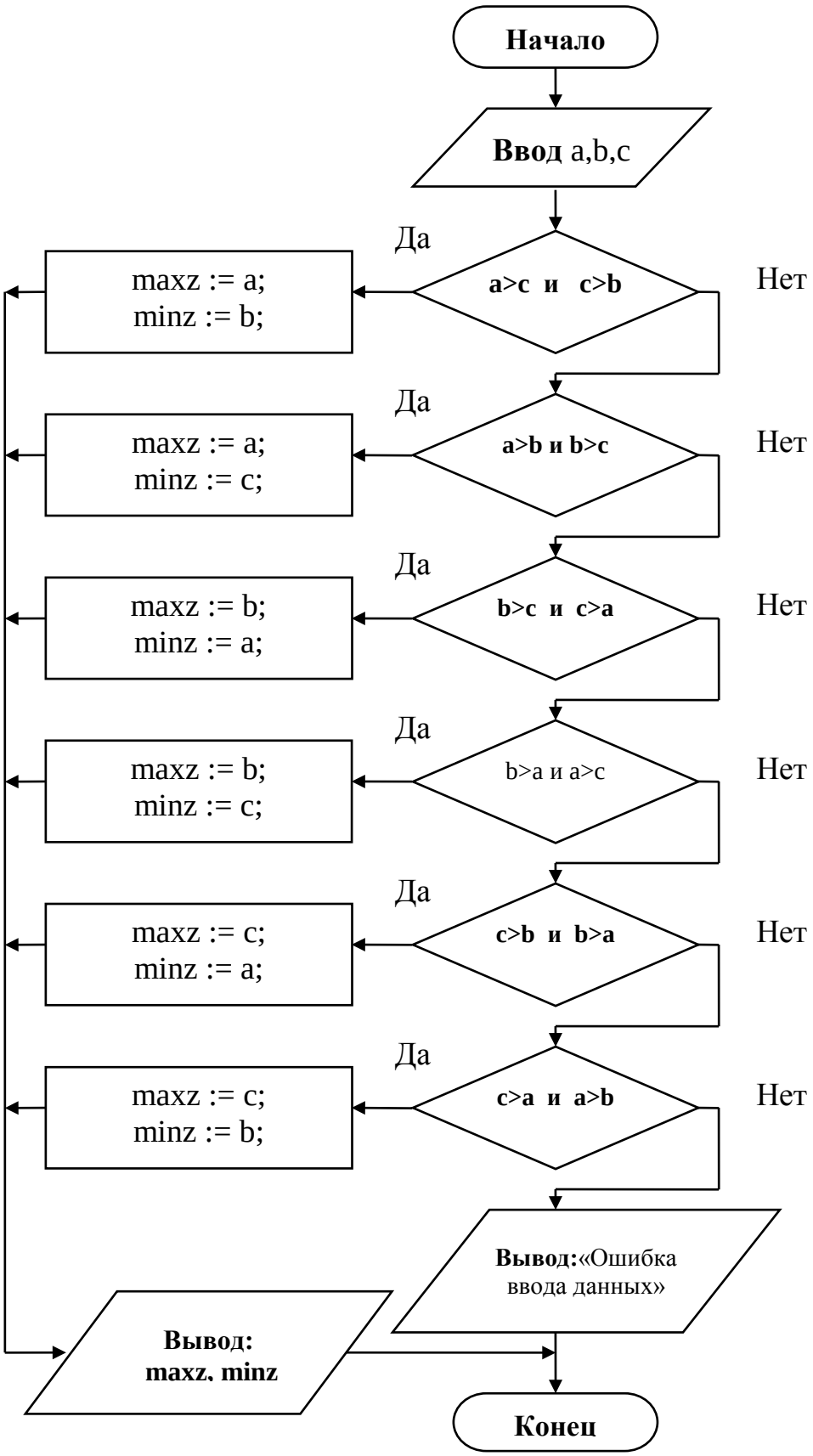


Рис. 9.
Алгоритм поиска максимума maxz
и минимума minz из трех различных чисел a, b, c

Пример 2.

Ввести значения элементов таблицы вещественных чисел X[i,j] из m строк и n столбцов (m>0, n>0) и подсчитать сумму S всех отрицательных элементов таблицы.

Решение. При составлении алгоритма решения задачи целесообразно воспользоваться стандартной конструкцией цикла с параметром. Данную стандартную конструкцию можно применить дважды: один раз во внешнем цикле, где параметром служит номер строки таблицы, и второй раз - во внутреннем цикле, где параметром служит номер столбца таблицы.

В начальный момент перед вводом значений элементов таблицы искомая сумма S полагается равной нулю. В дальнейшем после ввода каждого очередного элемента таблицы X[i,j] происходит проверка значения элемента на отрицательность и, при необходимости – добавление этого значения в сумму S.

Блок-схема алгоритма решения задачи представлена на рис.10. Соответствующий текст псевдокода с комментариями имеет вид:

```
1.  алг Сумма отрицательных элементов (арг цел m,n;
2.        вещ таб X[1:m,1:n];    рез вещ S)
3.   дано | m,n > 0;  X[i,j], i=1,2...m, j=1,2...n
4.   надо | S = сумма всех X[i,j] таких, что X[i,j]<0.0
5.   нач
6.        цел i;  |  i - счетчик цикла по строкам,
7.        цел j;  |  j - счетчик цикла по   столбцам
8.      ввод m,n;  | ввод размеров таблицы
9.      S:=0.0;  | присваивание начального значения сумме S
10.     Нц
11.       для i от 1 до m шаг 1 |заголовок цикла по строкам
12.         для j от 1 до n шаг 1 |заголовок цикла по столбцам
13.           ввод X[i,j];  | ввод очередного элемента таблицы
14.           если X[i,j]<0.0 то S:=S + X[i,j] все  | добавление
15.                  |  в сумму S отрицательного элемента X[i,j]
16.     Кц;
17.     вывод "S = ", S  |  вывод результата расчетов
18.  кон
```

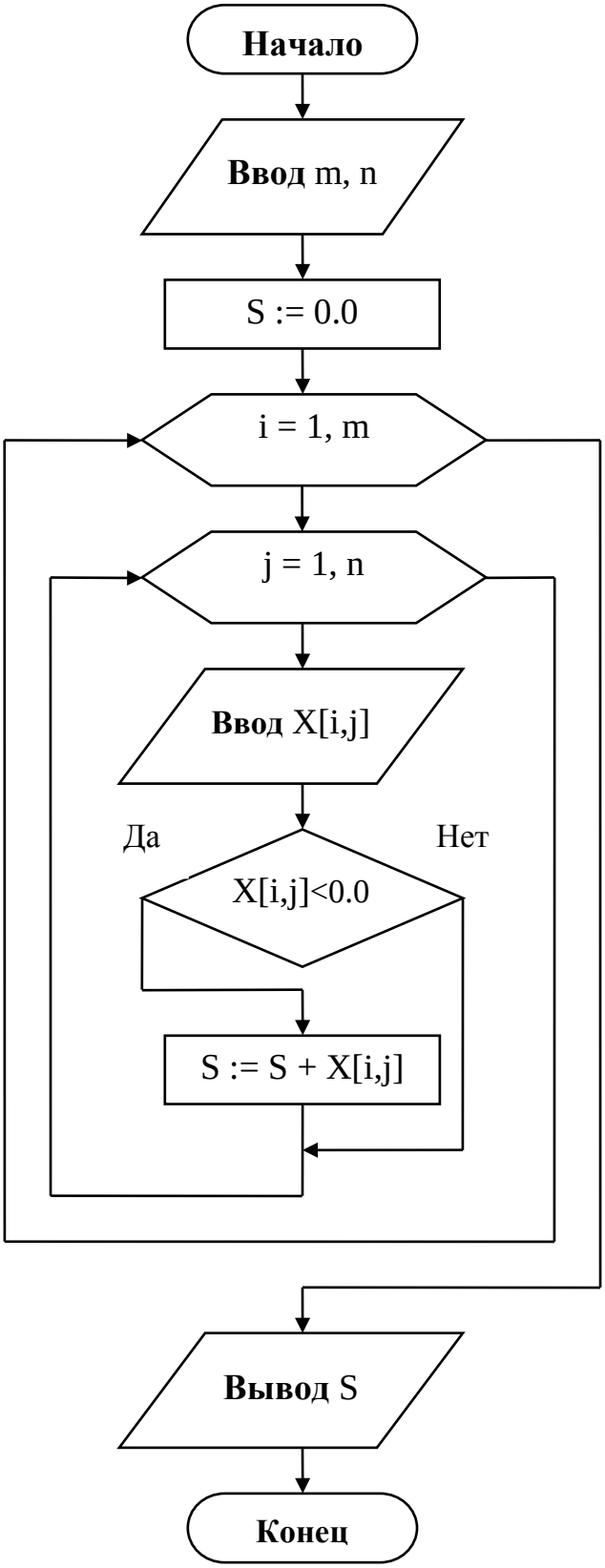


Рис. 10.
Алгоритм поиска суммы S отрицательных элементов таблицы x [1:m,1:n]

Индивидуальные задания

Необходимо выполнить триа задания.
Варианты заданий выбираются по последней цифре пин-кода.

Например, если последняя цифра 1, выполняются задания 1, 11 и 21.

При последней цифре 0 выполняются задания 10,20 и 30.

1. Составить алгоритм, вводящий положительные целые числа K, L, M, N и, считая их массами гирь, выясняющий, можно ли все эти гири как-либо положить на обе чаши весов таким образом, чтобы весы оказались в равновесии.
2. Составить алгоритм, вводящий вещественные числа A, B, C, D и, рассматривая эти числа как координаты четырех точек на прямой, определяющий расстояние между двумя самыми близкими друг к другу точками.
3. Составить алгоритм, вводящий значения элементов таблицы вещественных чисел X[i,j] из m строк и n столбцов (m>0, n>0 вводятся в начале работы) и определяющий число Q всех отрицательных элементов этой таблицы.
4. Составить алгоритм, вводящий три пары вещественных чисел, который считает их координатами трех точек на плоскости и проверяет, образуют ли они равнобедренный треугольник.
5. Составить алгоритм, вводящий значения элементов K[i] последовательности n целых чисел (n>0 вводится в начале работы) и определяющий число maxpodp элементов в ее наиболее длинной невозрастающей подпоследовательности.
6. Составить алгоритм, вводящий четыре пары вещественных чисел, который считает их координатами четырех точек на плоскости и находит наибольшую длину ломаной, проходящей через все эти точки по одному разу.
7. Составить алгоритм, вводящий значения элементов таблицы целых чисел K[i,j] из m строк и n столбцов (m>0, n>0 вводятся в начале работы) и определяющий номер imin той строки, в которой находится минимальный элемент таблицы.
8. Составить алгоритм, вводящий положительные целые числа K, L, M, N и, считая их массами гирь, выясняющий, можно

ли как-либо уравновесить гирю с массой K, если гири с массами L, M, N можно класть каждую на любую из чаш весов.

9. Составить алгоритм, вводящий значения элементов таблицы целых чисел K[i,j] из m строк и n столбцов (m>0, n>0 вводятся в начале работы) и определяющий номер imax той строки, в которой находится максимальный элемент таблицы.

10. Составить алгоритм, вводящий три пары вещественных чисел, который считает их координатами трех точек на плоскости и находит среди них такую точку, что сумма расстояний от окружности единичного радиуса с центром в ней до двух остальных точек максимальна.

11. Составить алгоритм, вводящий три целых числа, который находит второе по величине число, если такое число существует, а в противном случае печатает фразу «требуемое число отсутствует».

12. Составить алгоритм, вводящий значения элементов таблицы вещественных чисел X[i,j] из m строк и n столбцов (m>0, n>0 вводятся в начале работы) и определяющий число P всех положительных элементов этой таблицы.

13. Составить алгоритм, вводящий вещественные числа A, B, C, D и, рассматривая эти числа как координаты точек на прямой, определяющий расстояние между двумя наиболее удаленными друг от друга точками.

14. Составить алгоритм, вводящий четыре пары вещественных чисел, который считает их координатами четырех точек на плоскости и находит среди этих точек три, образующие треугольник наибольшей площади.

15. Составить алгоритм, вводящий значения элементов K[i] последовательности n целых чисел (n>0 вводится в начале работы) и печатающий все элементы последовательности в порядке их невозрастания.

16. Составить алгоритм, вводящий четыре пары вещественных чисел, который считает их координатами четырех точек на плоскости и находит наименьшую длину ломаной, проходящей через все эти точки по одному разу.

17. Составить алгоритм, вводящий значения элементов таблицы целых чисел K[i,j] из m строк и n столбцов (m>0, n>0 вводятся в начале работы) и определяющий номер jmin того столбца, в котором находится минимальный элемент таблицы.

18. Составить алгоритм, вводящий три пары вещественных чисел, который считает их координатами трех точек на плоскости и находит среди них такую точку, что сумма расстояний от окружности единичного радиуса с центром в ней до двух остальных точек минимальна.

19. Составить алгоритм, вводящий значения элементов K[i] последовательности n целых чисел (n>0 вводится в начале работы) и печатающий все элементы последовательности в порядке их неубывания.

20. Составить алгоритм, вводящий четыре пары вещественных чисел, который считает их координатами четырех точек на плоскости и находит среди этих точек три, образующие треугольник наименьшей площади.

21. Составить алгоритм, вводящий значения элементов K[i] последовательности n целых чисел (n>0 вводится в начале работы) и определяющий такой ее элемент K[sredn], что количество элементов, меньших K[sredn], совпадает с количеством элементов больших K[sredn]. Если же такого элемента K[sredn] в последовательности K[i] не окажется – напечатать фразу «требуемый элемент последовательности отсутствует».

22. Составить алгоритм, вводящий три пары вещественных чисел, который считает их координатами трех точек на плоскости и проверяет, образуют ли они прямоугольный треугольник.

23. Составить алгоритм, вводящий значения элементов таблицы целых чисел K[i,j] из m строк и n столбцов (m>0, n>0 вводятся в начале работы) и определяющий номер jmax того столбца, в котором находится максимальный элемент таблицы.

24. Составить алгоритм, вводящий четыре пары вещественных чисел, который считает их координатами четырех точек на плоскости и находит среди них такую, что сумма расстояний от нее до трех остальных точек максимальна.

25. Составить алгоритм, вводящий значения элементов K[i] последовательности n целых чисел (n>0 вводится в начале работы) и определяющий число lokmax локальных максимумов в последовательности (элемент является локальным максимумом, если он не имеет ближайших соседей, больших, чем он сам).

26. Составить алгоритм, вводящий четыре пары вещественных чисел, который считает их координатами четырех

точек на плоскости и находит среди этих точек три, образующие треугольник наибольшего периметра.

27. Составить алгоритм, вводящий значения элементов K[i] последовательности n целых чисел (n>0 вводится в начале работы) и определяющий число maxpodp элементов в ее наиболее длинной неубывающей подпоследовательности.

28. Составить алгоритм, вводящий четыре пары вещественных чисел, который считает их координатами четырех точек на плоскости и находит среди этих точек такую, что сумма расстояний от нее до трех остальных точек минимальна.

29. Составить алгоритм, вводящий значения элементов K[i] последовательности n целых чисел (n>0 вводится в начале работы) и определяющий число lokmin локальных минимумов в последовательности (элемент является локальным минимумом, если он не имеет ближайших соседей, меньших, чем он сам).

30. Составить алгоритм, вводящий четыре пары вещественных чисел, который считает их координатами четырех точек на плоскости и находит среди этих точек три, образующие треугольник наименьшего периметра.

ОГЛАВЛЕНИЕ