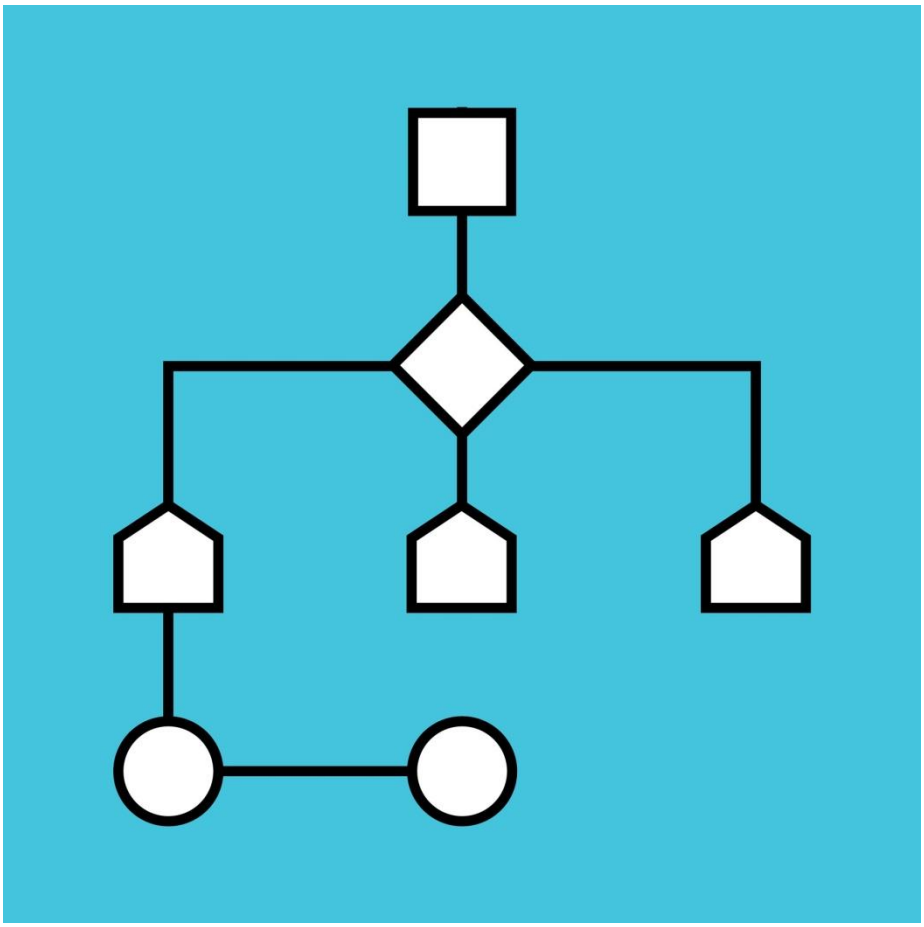
Частное профессиональное образовательное учреждение
**«СЕВЕРО-КАВКАЗСКИЙ КОЛЛЕДЖ ИННОВАЦИОННЫХ ТЕХНОЛОГИЙ
(ЧПОУ «СККИТ»)**

А.М. Жуков

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Учебное пособие



Пятигорск-2022

УДК 004.42

Жуков А.М.

Основы алгоритмизации и программирования[Электронный ресурс]: учеб. пособие / А.М. Жуков – Пятигорск, 2022.

УДК 004.42

Оглавление

Введение

Цель дисциплины – ознакомить студентов с основами современных информационных технологий, тенденциями их развития, обучить студентов принципам построения информационных моделей, проведению анализа полученных результатов, применению современных информационных технологий в профессиональной деятельности и, кроме того, она является базовой для всех дисциплин, использующих автоматизированные методы обработки данных, и так или иначе использующих компьютерную технику.

АЛГОРИТМ

Понятие «алгоритм» является концептуальной основой процессов обработки информации. Поскольку компьютер – универсальная машина для работы с информацией и может работать только по заданному ему алгоритму, то изучению свойств алгоритмов, методов их проектирования, анализа и оценки сложности в информатике уделяется большое внимание.

Понятие алгоритма

Термин «алгоритм» происходит от имени великого ученого средневекового Востока – Мухаммед ибн Муса ал-Хорезми (Магомет, сын Моисея из Хорезма). Он жил приблизительно с 783-го по 850 г. в городе Ургенче (Средняя Азия). В переводах арифметического трактата ал-Хорезми с арабского на латинский его имя транскрибировалось как algorismi. Отсюда и пошло слово «алгоритм» – сначала при изучении арифметических действий с натуральными числами в десятичной арифметике, а затем для обозначений процессов, в которых значения искомых величин последовательно определяются из исходных данных по определенным правилам и инструкциям.

К началу XX в. алгебра и теория чисел позволили дать много примеров алгоритмов. Среди них – алгоритм Евклида нахождения наибольшего общего делителя двух натуральных чисел или двух целочисленных многочленов, алгоритм Гаусса решения системы линейных уравнений, алгоритм нахождения рациональных корней многочленов одного переменного с рациональными коэффициентами, алгоритм Штурма определения числа действительных корней многочлена с действительными коэффициентами на некотором отрезке действительных чисел и т. д. Для решения задач такого типа достаточно интуитивного понимания термина «алгоритм».

В интуитивном (неформальном) понимании «алгоритм» – это описание вычислительного процесса. Такому пониманию термина способствуют многочисленные определения, встречающиеся в литературе и учебниках. Например, в Математическом энциклопедическом словаре читаем:

АЛГОРИТМ, алгорифм, – точное предписание, которое задает вычислительный процесс (называемый в этом случае алгоритмическим), начинающийся с произвольного исходного данного (из некоторой совокупности возможных для данного алгоритма исходных данных) и направленный на получение полностью определяемого этим исходным данным результата.

Формализация понятия «алгоритм»

Используя интуитивное понятие термина «алгоритм», можно доказать существование алгоритма решения задачи. Но в начале XX в. в математике были сформулированы алгоритмические проблемы, решение которых потребовало знать точно, что такое «алгоритм», чтобы доказать не существование алгоритма решения задачи (проблема алгоритмической неразрешимости). Другими словами, возникла задача формального определения термина «алгоритм».

В общем виде формализация понимается как приведение некоторого содержания *(воспринимаемых сигналов, смысла научной теории, содержания текста и т. д.)* к выбранной форме.

Необходимость формализации − то, что создано по формальным правилам, очень просто поддается проверке. Без формализации в принципе невозможно создать и использовать вычислительные машины. Заменив математические утверждения и рассуждения формальными выражениями, получаем возможность сделать сами эти рассуждения предметом компьютерной обработки.

Возможность формализации опирается на фундаментальное положение, которое называют **основным тезисом формализации.** Суть его состоит в ***принципиальной возможности разделения объекта и его обозначения*** *(имени объекта).*

Суть объекта не меняется от того, как мы его назовём. Это значит, что мы можем назвать его как угодно, придать его имени любую форму, которая, по нашему мнению, лучше соответствует данному объекту.

Решение задачи формализации понятия «алгоритм» было предложено в середине XX в. в работах Дж. Гильберта, А. Черча, А. Тьюринга, С. Клини, Э. Поста, А.А. Маркова, А.Н. Колмогорова и др.

Понятие теории алгоритмов

Различные подходы к формализации понятия «алгоритм» являются основой качественной (дескриптивной) теории алгоритмов, содержанием которой является классификация задач по признаку алгоритмической разрешимости, т.е. получение высказываний типа «Задача X алгоритмически разрешима» или «Задача X алгоритмически не разрешима». В рамках этой теории был получен ряд фундаментальных результатов. Например, в 1952 г. советский математик П.С. Новиков доказал алгоритмическую неразрешимость классической проблемы тождества для конечно определенных групп, а отечественный математик Ю.В. Матиясевич в 1970 г. доказал алгоритмическую неразрешимость 10-й проблемы Гильберта.

Теория алгоритмов оказала существенное влияние на развитие ЭВМ и практику программирования. В теории алгоритмов были предопределены концепции, положенные впоследствии в основу аппаратуры ЭВМ и языков программирования. Понятие алгоритма помогло, например, точно определить, что значит эффективно задать последовательность управляющих сигналов.

В свою очередь, применение ЭВМ для обработки информации послужило стимулом развития теории алгоритмов и изучения алгоритмических моделей, изучения алгоритмов с целью их сравнения по характеристикам (числу действий, расходу памяти), а также их оптимизации. В настоящее время теория алгоритмов служит теоретическим фундаментом вычислительных наук. С ее помощью уточняются такие понятия, как доказуемость, эффективность, разрешимость, перечислимость и др. Возникло новое направление в теории алгоритмов – сложность алгоритмов и вычислений.

Начала складываться количественная (метрическая) теория алгоритмов, основным содержанием которой является получение верхних и нижних оценок сложности алгоритмов.

Для получения верхних оценок, достаточно интуитивного понятия алгоритма, «строится» неформальный алгоритм решения конкретной задачи, и затем он формализуется для реализации на подходящей алгоритмической модели. Если удается показать, что сложность (время или память) вычислений для этого алгоритма не превосходит значения подходящей функции при всех значениях аргумента, то эта функция считается верхней оценкой сложности решения рассматриваемой задачи. В получении верхних оценок достигнуто много интересных результатов для конкретных задач. Например, разработаны быстрые алгоритмы умножения целых чисел, многочленов, матриц, решения линейных систем уравнений, которые требуют значительно меньше ресурсов, чем традиционные алгоритмы.

Для получения нижней оценки требуется доказать, что никакой алгоритм не имеет сложности меньшей, чем заданная граница. Для этого необходимо уточнить понятие алгоритма.

Уточнение понятия алгоритма связано с уточнением алфавита данных и формы их представления, памяти и размещения в ней данных, элементарных шагов алгоритма и механизма реализации алгоритма. Все эти понятия сами нуждаются в уточнении. Их словесные определения потребуют введения новых понятий, для которых, в свою очередь, снова необходимы уточнения, и т. д. Поэтому в теории алгоритмов принят другой подход, основанный на конкретной алгоритмической модели, в которой все сформулированные требования выполняются очевидным образом. Используемая алгоритмическая модель универсальна, т.е. моделирует любые другие разумные алгоритмические модели, что позволяет снять возможное возражение против такого подхода: не приводит ли жесткая фиксация алгоритмической модели к потере общности формализации алгоритма? Поэтому алгоритмические модели отождествляются с формальным понятием алгоритма. Известны три основных типа алгоритмических моделей, различающихся исходными трактовками того, что такое алгоритм.

Алан Тьюринг
(Alan Mathison Turing,
1912 – 1954)

Первый тип алгоритмических моделей связывает понятие алгоритма с традиционным представлением – процедурами вычисления значений числовых функций. Основной теоретической моделью этого типа являются рекурсивные функции – исторически первая формализация понятия алгоритма.

Второй тип алгоритмических моделей трактует алгоритм как некоторое детерминированное устройство, способное выполнять в каждый момент лишь строго фиксированное множество операций.

Основной теоретической моделью такого типа является машина А. Тьюринга, оказавшая существенное влияние на понимание логической природы разрабатываемых ЭВМ. Аналогичной теоретической моделью такого типа является машина Э. Поста (1936 г.). Машина А. Поста менее сложна, чем машина А. Тьюринга: ее элементарные действия проще и менее разнообразны способы записи. Но по этим причинам запись и обработка информации на ней требует большего объема «памяти» и большего числа шагов, чем для машины А. Тьюринга.

В конце 1936 г. Алан Тьюринг опубликовал работу «О вычислимых числах с приложением к проблеме разрешимости».

Математический результат этой работы – ***доказательство существования формальной, чисто механической процедуры, позволявшей сделать заключение, выводимо ли данное высказывание из некоторого набора аксиом.***

Эта процедура, получившая название «универсальная машина», а позднее – «машина Тьюринга», стала *первым формальным описанием компьютера как некоей абстрактной цифровой вычислительной машины*, позволяющей имитировать любую другую вычислительную машину, работа которой состоит в переходе из одного дискретного состояния в другое.

Тезис Тьюринга*: Любой алгоритм может быть реализован соответствующей машиной Тьюринга*.

Факт построения воображаемой универсальной машины привел к предположению о целесообразности построения универсальной вычислительной машины, которая могла бы решать любые задачи при условии соответствующей кодировки исходных данных и разработки соответствующей программы действий.

Алгоритм *(по Тьюрингу) – программа для машины Тьюринга, приводящая к решению поставленной задачи.*

Универсальность машины Тьюринга в том, что любой алгоритм над любыми данными можно представить в символьной форме, т.е. объекты описать в виде некоторого текста; действия представить как правила его преобразования, кроме того, можно реализовать любые преобразования любых текстов.

Андрей Андреевич Марков
(1903–1979)

Другой теоретической моделью этого типа является машина произвольного доступа, введенная в 70-х гг. XX в. с целью моделирования реальных вычислительных машин и оценки сложности вычислений.

Третий тип алгоритмических моделей – это преобразования слов в произвольных алфавитах, в которых операциями являются замены частей слов другим словом. Основной теоретической моделью этого типа являются нормальные алгоритмы (алго-

рифмы) А.А. Маркова (1947 г.). А.А. Марков установил эквивалентность понятий нормального алгоритма, рекурсивной функции и машины А. Тьюринга с точки зрения их вычислительной силы.

Все три основных типа алгоритмических моделей математически эквивалентны и различаются сложностями, возникающими при практической реализации алгоритмов. Например, микропрограммирование основывается на идее машины Тьюринга, структурное программирование опирается на теорию рекурсивных функций, а языки символьной обработки данных (ЛИСП, РЕФАЛ, ПРОЛОГ) берут свое начало от нормальных алгоритмов А. А. Маркова.

Универсальная функция

В теории алгоритмов установлен важный факт: во всех алгоритмических моделях всегда существует универсальный алгоритм, т.е. алгоритм, который способен моделировать работу любого другого алгоритма, описанного в этой же модели. Покажем существование универсальной функции для вычислимых функции одного аргумента.

Теорема. Существует вычислимая функция двух аргументов U, являющаяся универсальной функцией для класса вычислимых функций одного аргумента.

Доказательство. Расположим все программы, вычисляющие функции одного аргумента, в вычислимую последовательность p_0, p_1, ... (например, в порядке возрастания их длины).

Пусть U есть вычислимая функция, такая, что $U(i, x)$ равна результату работы i-й программы p_i на входном слове x.

Если программа p_i заканчивает работу на входном слове x, и не определена в противном случае (т.е. программа p_i не применима к входному слову x), то функция U и есть искомая двухместная универсальная вычислимая функция для класса всех одноместных вычислимых функций, что и требовалось доказать.

Алгоритм, вычисляющий саму функцию U, есть, по существу, интерпретатор для используемого языка программирования (если отождествлять программу и ее номер, то он применяет первый аргумент ко второму). Аналогично можно ввести универсальные функции для произвольного числа аргументов.

Важность концепции универсальной функции состоит в том, что еще в 30-е гг. XX в. (т.е. до появления ЭВМ) она показала возможность создания универсального устройства (компьютера), способного выполнять любые алгоритмы.

Нормальные алгорифмы А.А. Маркова

Рассмотрим подробнее нормальные алгоритмы (алгорифмы) А.А. Маркова.

Для формализации и уточнения понятия алгоритм российский математик А.А. Марков предложил использовать ассоциативные исчисления [12].

Ассоциативным исчислением называют совокупность всех слов в данном алфавите вместе с системой допустимых подстановок.

Алфавит – конечный набор различных символов, которые будем называть буквами.

Пример. Пусть задан конечный набор различных символов (букв) *a, b, c.* Следовательно, определен алфавит *A = {a, b, c}.*

Слово – любая конечная последовательность букв (линейный их ряд) в этом алфавите.

Пример. В алфавите *A = {a, b, c}* заданы слова:

N = ab,
M = bcb,
K = abcbcbab.

Рассмотрим два слова *N* и *K* в некотором алфавите *A.* Если слово *N* является частью *K*, то говорят, что *N* входит в *K.*

Зададим в алфавите *A* конечную систему подстановок

N – M,
S – T, ..., где *N, M, S, T,* ... – слова в этом алфавите.

Любую подстановку *N – M* можно применить к некоторому слову *K* следующим способом: если в *K* имеется одно или несколько вхождений слова *N*, то любое из них может быть заменено словом *M* и, наоборот, если имеется вхождение *M,* то его можно заменить словом *N.*

Примеры. В алфавите *A = {a, b, c}* имеются слова:

1) N = ab, *M = bcb, K = abcbcbab.*

Заменив в слове *K*= ***ab****cbcbab* слово *N* на *M,* получим ***bcb****cbcbab* или *abcbcb****bcb*** и, наоборот, заменив *M* на *N*, получим *a****ab****cbab* или *abc****ab****ab;*

2) *S = bacb,* подстановка *ab – bcb* недопустима к слову *S*, так как ни *ab*, ни *bcb* не входит в это слово.

К полученным с помощью допустимых подстановок словам можно снова применить допустимые подстановки и т.д.

Слова *P1* и *P2* в некотором ассоциативном исчислении называются смежными, если одно из них может быть преобразовано в другое однократным применением допустимой подстановки.

Последовательность слов *P, P1, P2, ..., M* называется дедуктивной цепочкой, ведущей от слова *P* к слову *M,* если каждое из двух рядом стоящих слов этой цепочки смежное.

Слова *P* и *M* называют эквивалентными, если существует цепочка от *P* к *M* и обратно.

Пример. Пусть заданы:

алфавит:

{a, b, c, d, e};

система подстановок:

ac – ca;
abac – abace;
ad – da;
eca – ae;
bc – cb;
eda – be;
bd – db;
edb – be.

Слова $P = a\mathbf{bc}de$ и $M = a\mathbf{cb}de$ – смежные (подстановка *bc – cb).*

Слова $P = abcde$ и $M = cadbe$ – эквивалентны.

Любой процесс вывода формул, математические выкладки и преобразования также являются дедуктивными цепочками в некотором ассоциативном исчислении.

Построение ассоциативных исчислений является универсальным методом детерминированной переработки информации и позволяет формализовать понятие алгоритма.

На основе ассоциативного исчисления А.А. Марков ввел понятие алгоритма:

«***Алгорифмом в алфавите А называется точное общепонятное предписание, определяющее потенциально осуществимый процесс последовательного преобразования абстрактных слов в А, процесс, допускающий любое слово в А в качестве исходного»*** [13 , с. 51].

Алгоритм в алфавите *A* задается в виде системы допустимых подстановок, дополненной точным предписанием о том, в каком порядке нужно применять допустимые подстановки и когда наступает останов.

Пример. Заданы:

алфавит:

$A = \{a, b, c\};$

система подстановок *B*:

$cb \rightarrow cc,$ (1)

$cca \rightarrow ab,$ (2)

$ab \rightarrow bca.$ (3)

Предписания о применении подстановок:

в произвольном слове *P* надо сделать возможные подстановки, заменив левую часть подстановок на правую;

повторить процесс с вновь полученным словом.

Применим систему подстановок *B* к слову *babaac:*

*b**ab**aac*→ *b**bca**aac*

так как система подстановок исчерпана, о алгоритм завершен (останов).

Применим систему подстановок *B* к слову *bcacabc:*

*bcac**ab**c* → *bcac**bca**c,* применена подстановка (3)

*bca**cb**cac*→ *bca**cc**cac,* применена подстановка (1)

*bcac**cca**c*→ *bcac**ab**c,* применена подстановка (2)

В результате получили исходное слово, следовательно, процесс бесконечный, т.е. останова нет.

Опираясь на ассоциативное исчисление, А.А. Марков ввел понятие **нормального алгорифма**. Определение всякого нормального алгорифма состоит из определения:

1) алфавита алгорифма, к словам в котором алгорифм будет применяться;

2) схемы алгорифма.

Схемой нормального алгорифма называется ***конечный упорядоченный*** набор формул подстановки, каждая из которых может быть простой или заключительной.

Пусть заданы:

– *алфавит* $A = \{ a_1, ..., a_i \};$

– P и Q – слова в алфавите $A;$

– выражение $P \rightarrow Q$ - *простая формула подстановки* слов в алфавите $A;$

– выражение $P \rightarrow \cdot Q$ - *заключительная формула подстановки* слов в алфавите $A.$

– *конечный список формул подстановки в алфавите A* (группа символов *(·)* – точка в скобках означает, что формула подстановки может быть простой или заключительной):

$$P_1 \rightarrow (\cdot)Q_1,$$
$$P_2 \rightarrow (\cdot)Q_2,$$
$$P_3 \rightarrow (\cdot)Q_3,$$
$$\dots$$
$$P_m \rightarrow (\cdot)Q_m,$$
$$\dots$$
$$P_r \rightarrow (\cdot)Q_r.$$

Условимся предварительно, что слово *T входит* в слово *Q,* если существуют такие (возможно пустые) слова *U* и *V,* что

$$Q = UTV.$$

Алгорифм, определенный таким образом, называется *нормальным алгорифмом (или алгорифмом А.А. Маркова)* в алфавите А.

Работа алгорифма может быть описана следующим образом.

Пусть дано слово *P* в алфавите *A*.

Находим первую в схеме алгорифма формулу подстановки

$P_m \rightarrow (\cdot)Q_m$ такую, что P_m входит в P.

Подставляем слово Q_m вместо самого левого вхождения слова P_m в слово P. Пусть R_1 – результат подстановки.

Если формула подстановки $P_m \rightarrow (\cdot)Q_m$ – простая, то применим к R_1 тот же поиск, который был только что применен к P, и т. д.

Если формула подстановки $P_m \rightarrow (\cdot)Q_m$ – заключительная, то работа алгорифма заканчивается ***(останов)*** и R_1 будет результатом его работы.

Если получим такое слово R_i, что ни одно из слов P_1, ... , P_r не входит в R_i, то работа алгорифма заканчивается и R_i будет результатом его работы.

Возможно, что описанный процесс никогда не закончится. В таком случае говорят, что алгорифм неприменим к слову P.

Пример нормального алгорифма, описывающего сложение натуральных чисел, представленных наборами единиц.

Задан алфавит:

A={+,1}

Схема алгорифма (Задана конечная система подстановок B):

$1+ \rightarrow +1$ (1)

$+1 \rightarrow \cdot 1$ (2)

т.е. точка перед единицей означает, что эта формула подстановки заключительная.

Применим этот алгоритм к слову: P = 11+11.

Последовательно применяя подстановки к слову P, получаем:

P = 11+11	P = 1***1***+11	применим формулу (1)	P_1 = 1+***1***11
	P_1 = ***1***+111	применим формулу (1)	P_2 = +***1***111
	P_2 = +***1***111	применим формулу (2)	P_3 = 1111

Нормальный алгоритм Маркова можно рассматривать как универсальную форму задания любого алгоритма. *Универсальность нормальных алгоритмов декларируется принципом нормализации: для любого алгоритма в произвольном конечном алфавите А можно построить эквивалентный ему нормальный алгоритм (над алфавитом А).*

Принцип нормализации теперь может быть высказан в видоизмененной форме: все алгоритмы нормализуемы.

Данный принцип *не может быть строго доказан*, поскольку понятие произвольного алгоритма не является строго определенным и основывается на том, что все известные в настоящее время алгоритмы являются нормализуемыми, а способы ком-

позиции алгоритмов, позволяющие строить новые алгоритмы из уже известных, не выводят за пределы класса нормализуемых алгоритмов.

Способы композиции нормальных алгоритмов

Суперпозиция алгоритмов. При суперпозиции двух алгоритмов *A* и *B* выходное слово первого алгоритма *A* рассматривается как входное слово второго алгоритма *B,* результат суперпозиции *C* может быть представлен в виде *C(p) = B(A(p)).*

Объединение алгоритмов. Объединением алгоритмов *A* и *B* в одном и том же алфавите называется алгоритм *C* в том же алфавите, преобразующий любое слово *p,* содержащееся в пересечении областей определения алгоритмов *A* и *B,* в записанные рядом слова *A(p)* и *B(p).*

Разветвление алгоритмов. Разветвление алгоритмов представляет собой композицию *D* трех алгоритмов *A, B* и C, причем область определения алгоритма *D* является пересечением областей определения всех трех алгоритмов *A, B* и *C*, а для любого слова *p* из этого пересечения *D(p) = A(p),* если *C(p) = e, D(p) = B(p),* если *C(p) = e,* где *e* – пустая строка.

Итерация алгоритмов. Итерация (повторение) представляет собой такую композицию *C* двух алгоритмов *A* и *B,* что для любого входного слова *p* соответствующее слово *C(p)* получается в результате последовательного многократного применения алгоритма *A* до получения слова, преобразуемого алгоритмом *B.*

Нормальные алгоритмы Маркова являются не только средством теоретических построений, но и основой специализированного языка программирования РЕФАЛ, применяемого как язык символьных преобразований при разработке систем искусственного интеллекта. Это один из немногих языков, разработанных в России и получивших известность во всем мире.

Существует строгое доказательство того, что по возможностям преобразования нормальные алгоритмы Маркова эквивалентны машинам Тьюринга.

Точное определение понятия «алгоритм» позволяет сказать о задаче, для решения которой нам не известен алгоритм, существует ли он вообще. В практике под алгоритмом обычно понимают программу для компьютера, т.е. разрабатывают алгоритм как программу на каком-либо языке программирования. Ограниченность такого подхода состоит в следующем:

1) все, что можно запрограммировать с помощью какого-либо языка программирования, представляется любой из алгоритмических моделей (А. Черч, А. Пост, А. Тьюринг, А.А. Марков),

2) формализация понятия алгоритм позволила установить, какие задачи вообще нельзя запрограммировать.

В теории алгоритмов доказано существование двух алгоритмически не разрешимых задач:

1) по описанию алгоритма и его входным данным необходимо выяснить, завершит ли он работу. Эта не разрешимая задача известна как «проблема останова».

Отсюда следует, что невозможно создать универсальный (пригодный для любой программы) алгоритм отладки программ;

2) проблема эквивалентности – существует ли «машина Т» (алгоритм), для которой исходными данными являются результаты работы «машины T_1» и «машины T_2» (алгоритмов), которая определяет эквивалентность результатов работы «машин T_1 и T_2» (алгоритмов). Отсюда следует, что на практике каждый программист должен доказать, действительно ли разработанная им программа решает поставленную задачу.

В качестве рабочего определения понятия «алгоритм» используем определение Международной организации по стандартизации (документ СТ ИСО 2382/1 – 84):

АЛГОРИТМ (ALGORITHM) – конечный набор предписаний, определяющий решение задачи посредством конечного количества операций.

Пример. Полная формулировка последовательности арифметических операций, с помощью которых производят вычисление синуса с заданной точностью.

Характерные свойства алгоритма:

1) дискретность – алгоритм состоит из отдельных команд,

2) определенность – каждая команда алгоритма определена и указан способ ее выполнения,

3) результативность – алгоритм всегда приводит к получению результата,

4) правильность – сам алгоритм правильный и применение правильное,

5) массовость – алгоритм разработан для решения класса однотипных задач,

6) конечность – результат достигается за конечное число шагов.

Совокупность задач, к которым применим алгоритм, называется областью применимости алгоритма.

Исполнитель алгоритмов

Алгоритм всегда разрабатывают для конкретного исполнителя.

Исполнитель алгоритмов – это объект (кто или что), формально исполняющий алгоритмы. Исполнитель характеризуется системой команд (СКИ) и средой, в которой исполняет алгоритмы. Если исполнитель получает команду, не входящую в его систему команд, то он сообщает «не понял». Если исполнитель получает команду, которую не может исполнить по какой-либо причине, то он сообщает «не могу».

В теории алгоритмов исполнители: машины Поста и Тьюринга. В практике – исполнители: ЭВМ, роботы. В обучении исполнителями служат различные учебные программы: Кукарача, Перевозчик, Кенгуренок, ГРИС, Стрелочка и т.д.

Величины

Информация предоставляется исполнителю в форме величин. Величина в информатике – это объект любой природы: число, текст, таблица, график и т. д.

Основные характеристики величин:

1) имя величины,
2) вид величины,
3) тип величины,
4) значение величины.

Имя величины

С точки зрения человека имя величины – это ее обозначение в алгоритме. Для исполнителя имя величины – это место в памяти, где хранится значение величины. Имя величины – статическая характеристика, т. е. имя не изменяется в процессе исполнения алгоритма.

Именем может быть любая последовательность символов (букв, цифр и знаков подчеркивания), начинающаяся с буквы. Не принято использовать в имени символы: , ? ! * % ; : №

Правильные имена: VEL15, VEL_15, КНИГА

Неправильные имена: ?VEL15, *VEL_15, \КНИГА

Виды величин

Вид величины характеризует ее использование в алгоритме. Вид величины – статическая характеристика, т. е. он не изменяется в процессе исполнения алгоритма. Различают: аргументы, результаты, вспомогательные величины.

Типы величин

Тип величины задает множество допустимых значений величины и множество выполняемых над ней операций. Тип величины – статическая характеристика, т. е. тип величины не изменяется в процессе исполнения алгоритма (см. табл. 1).

Таблица 1. Типы величин

Типы величин	Обозначение	Описание	Например
Числовые	нат	Натуральный ряд чисел	**нат** *S* Описана величина *S* натурального типа
	цел	Целые числа	**цел** *M* Описана величина *M* целого типа
	вещ	Вещественные (действительные) числа	**вещ** *P* Описана величина *P* действительного типа
Логические	лог	Множество значений величины логического типа состоит из двух элементов: **да** и **нет**	**лог** *F* Описана величина *F* логического типа
Табличные	таб	Таблица с числовыми, литерными или символьными элементами	**таб цел** *T* [1..10] Описана величина *T* табличного типа, содержащая 10 элементов целого типа
Символьные	сим	Символы алфавитов	**сим A** Описана величина A символьного типа
Литерные	лит	Строка символов	**лит L** Описана величина L литерного типа

Таблица 2. Операции, выполняемые над числовыми величинами

Операция	Тип результата
Арифметические операции	
+ сложение	Определяется типом величин слагаемых
– вычитание	Определяется типом величин
* умножение	Определяется типом величин сомножителей
/ деление	**вещ**
^ возведение в степень	Определяется типом величин сомножителей
Операции сравнения	
= равно	**лог**
< меньше	
> больше	
< > не равно	
<= меньше или равно	
>= больше или равно	

Кроме того, для числовых величин определен ряд стандартных функций. Укажем две из них:

div(x,y) – целая часть частного от деления x на y (x, y – тип **цел**);

mod(x,y) – остаток от деления x на y (x, y – тип **цел**).

Над величинами логического типа можно выполнять известные логические операции (НЕ, И, ИЛИ, импликация и т.д.) и две операции сравнения = (равно), <> (не равно); результат всех операций имеет тип **лог** (см. табл. 2)

Значение величины

Значение величины – это ее содержание. Значение величины – динамическая характеристика, т. е. оно может изменяется в процессе исполнения алгоритма сколько угодно раз.

Изменить значение величины можно командой ввода значения величины в память исполнителя или командой присваивания в процессе исполнения алгоритма.

Обозначение команды присваивания (двоеточие равно)

:=

Формат команды присваивания:

имя величины **:=** выражение

Например:

A:= 6, читается: величине A присвоить значение 6;

B:=A–1, читается: величине B присвоить значение разности значения величины A и единицы.

Запомните! Тип величины в левой части команды присваивания и тип выражения (или константы, или величины) в правой части команды присваивания должны быть одинаковыми!

Способы представления алгоритмов

Наиболее известны следующие способы представления алгоритмов:

– на естественном языке (словесное описание);

– на псевдокоде, например на школьном алгоритмическом языке (АЯ);

– в виде графической схемы алгоритма (блок-схемы);

– в виде N-S диаграммы (структурограмма, диаграмма Насси-Шнейдермана, структурная блок-схема) [19];

– на языке программирования (в виде программы).

Алгоритмический язык

Способ записи алгоритмов, в котором используются предложения естественного и математического языков, называется *псевдокодом.* Для наших (учебных) целей

наиболее пригоден псевдокод, известный под названием *школьный алгоритмический язык* – АЯ.

Алгоритмический язык – это набор символов, соглашений и правил для записи алгоритмов.

В качестве набора символов можно использовать любые известные исполнителю символы.

Соглашения и правила записи алгоритмов. Для записи алгоритмов используются служебные слова. Их назначение, смысл и способ применения строго определены. Служебные слова в тексте алгоритма выделяют каким-либо способом, обычно подчеркиванием.

1. Минимально необходимый набор служебных слов:

алг (алгоритм) – указывает заголовок алгоритма;

дано указывает на исходные данные и условия применимости алгоритма;

надо указывает на выходные данные (цель исполнения алгоритма);

нач (начало) – указывает начало тела алгоритма;

кон (конец) – указывает конец алгоритма.

2. Комбинации служебных слов:

если то иначе все;

пока нц кц;

выполнять до;

для от до шаг нц кц

используются для записи управляющих структур. Подробно они будут описаны в разделе «Основные управляющие структуры».

3. Общий вид алгоритма:

алг ИМЯ_(**арг**ументы, **рез**ультаты)
    дано условия применимости алгоритма
    надо цель исполнения алгоритма
нач
    команда или управляющая структура №1
    команда или управляющая структура №2
    …
    команда или управляющая структура №N
кон

Графические схемы алгоритмов

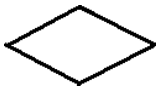
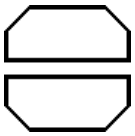
Графическая схема алгоритма представляется в соответствии с ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения» (разработанный методом прямого применения международного стандарта ИСО 5807-85 «Обработка информации. Символы и условные обозначения блок-схем данных, программ и систем, схем программных сетей и системных ресурсов»).

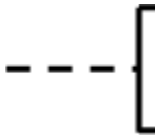
Для построения графической схемы алгоритма используют блоки (имеющие свое наименование и обозначение) и управляющие структуры, состоящие из этих блоков. Минимальный набор графических символов, необходимый для проектирования и анализа алгоритмов, приведен в табл. 3. Любая управляющая структура имеет один вход и один выход. Алгоритм, построенный из таких структур, называется *структурированным.*

Структурное кодирование – это метод разработки алгоритмов, основанный на использовании *управляющих структур: следование, ветвление, цикл.* Правильность любой структуры легко проанализировать и установить. При этом одни управляющие структуры могут быть вложены в другие.

Таблица 3. Применение графических символов для изображения графических схем алгоритмов в соответствии с ГОСТ 19.701-90 (ИСО 5807-85)

Наименование символа	Обозначение символа	Функции
Данные		Символ отображает данные, носитель данных не определен. Используется для обозначения операций ввода и вывода данных
Процесс		Символ отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации). Используется для обозначения операций присваивания
Предопределенный процесс		Символ отображает предопределенный процесс, состоящий из одной или нескольких операций или шагов, которые определены в другом месте (в подпрограмме, модуле). Используется для обозначения неэлементарных блоков
Подготовка		Символ отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию (установка переключателя, модификация индексного регистра или инициализация программы). Может быть использован для обозначения заголовка цикла

Наименование символа	Обозначение символа	Функции
Решение		Символ отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа. Соответствующие результаты вычисления могут быть записаны по соседству с линиями, отображающими эти пути. Используется для обозначения оператора условного перехода или оператора варианта
Граница цикла		Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т.д. помещаются внутри символа в начале или в конце в зависимости от типа цикла
Соединитель		Символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения ее в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение
Терминатор		Символ отображает выход во внешнюю среду и вход из внешней среды. Используется для обозначения начала или окончания алгоритма
Линия		Символ отображает поток данных или управления. Направления справа налево и снизу вверх обозначаются стрелками. Используется для соединения символов в алгоритме
Параллельные действия		Символ отображает синхронизацию двух или более параллельных операций
Пунктирная линия		Символ отображает альтернативную связь между двумя или более символами. Кроме того, символ используется для обведения аннотированного участка при записи комментариев

Наименование символа	Обозначение символа	Функции
Комментарий		Символ используется для добавления описательных комментариев или пояснительных записей с целью объяснений или примечаний. Пунктирные линии в символе комментария связаны с соответствующим символом или могут обводить группу символов. Текст комментариев или примечаний должен быть помещен около ограничивающей фигуры
Пропуск		Символ (три точки) используется в схемах для отображения пропуска символа или группы символов, в которых не определены ни тип, ни число символов. Символ используется только в символах линий или между ними. Он применяется главным образом в схемах, изображающих общие решения с неизвестным числом повторений

Основные управляющие структуры

Управляющая структура «Следование»

Данная управляющая структура используется в тех случаях, когда исполнитель должен выполнить последовательно одну или несколько операций.

Общий вид управляющей структуры «Следование»

в виде блок-схемы:

на псевдокоде:

нач
<серия команд>
кон

В соответствии с парадигмой структурного программирования первоначально любая, даже очень сложная задача, представляется с использованием структуры «Следование» и затем каждая <серия команд> последовательно уточняется.

Пример использования управляющей структуры «Следование»

Задача. Разработайте алгоритм, который увеличивает на 2 (целое число) введенное пользователем значение величины целого типа *A* и выводит на экран полученное значение *A*.

Решение

Проектирование

1. Постановка задачи:
   - входные данные: величина A целого типа, **цел** A;
   - выходные данные: величина A целого типа, **цел** A;
   - связь между входными и выходными данными: A:=A+2.
2. Проект решения задачи сведен в табл. 4.

Таблица 4. Проект решения задачи

	Выполняемые действия	Наименование графического символа
1	Начало алгоритма	Терминатор
2	Описание используемых переменных	Процесс
3	Ввод данных (ввод значения переменной *A*, тип целый)	Данные
4	Присваивание значения (переменной *A* значения *A*+2)	Процесс
5	Вывод данных (вывод значения переменной *A*, тип целый)	Данные
6	Конец алгоритма	Терминатор

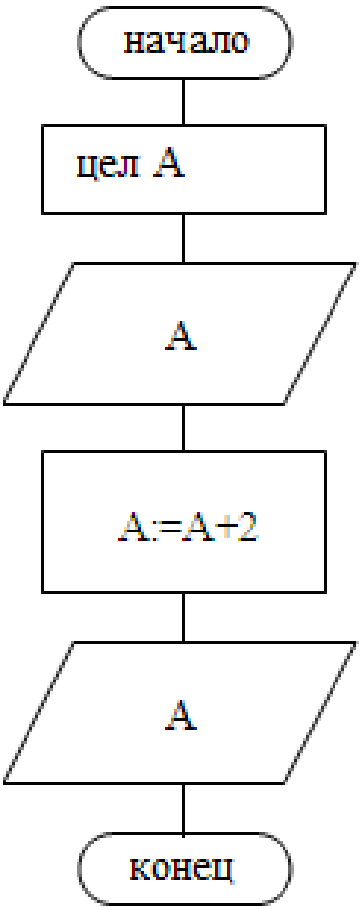
Реализация

Представим алгоритм в виде графической схемы.

Используемые в алгоритме графические символы описаны в табл. 4 (правый столбец).

Для представления потока данных и управления используется стандартный символ «Линия».

Таким образом, получим следующую графическую схему алгоритма решения задачи:



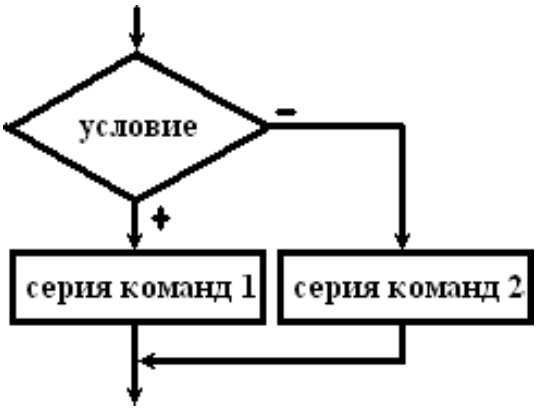
Управляющая структура «Ветвление»

Управляющая структура «Ветвление» используется в тех случаях, когда исполнитель должен выбрать одно из двух возможных продолжений исполнения алгоритма.

Общий вид управляющей структуры «Ветвление»

Полная форма:

в виде блок-схемы:

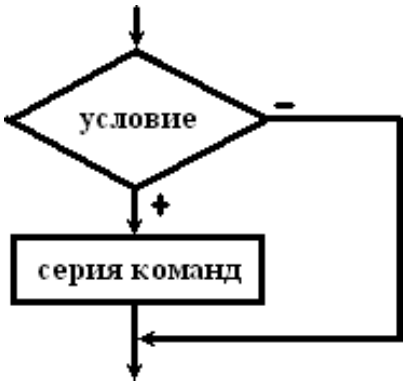


на псевдокоде:

<u>если</u> <условие >
  <u>то</u> <серия команд 1>
  <u>иначе</u> <серия команд 2>
<u>все</u>

Сокращенная форма:

в виде блок-схемы:



на псевдокоде:

<u>если</u> <условие >
  <u>то</u> <серия команд>
<u>все</u>

Условие – это логическое выражение, принимающее значение истина (True) или ложь (False).

В зависимости от значения истинности условия исполнитель исполняет одну и только одну серию команд.

Пример использования управляющей структуры «Ветвление» (сокращенная форма)

Задача. Разработайте алгоритм, который проверяет введенное пользователем значение величины целого типа *A,* и если оно отрицательное, то увеличивает введенное значение на 2, а затем выводит на экран полученное значение *A*.

Решение

Проектирование

1. Постановка задачи:
   - входные данные: величина A целого типа, **цел** A;
   - выходные данные: величина A целого типа, **цел** A;
   - связь между входными и выходными данными:

     если A<0

     то A:=A+2.

2. Проект решения задачи сведен в табл. 5.

Таблица 5. Проект решения задачи

	Выполняемые действия	Наименование графического символа
1	Начало алгоритма	Терминатор
2	Описание используемых переменных	Процесс
3	Ввод данных (ввод значения переменной *A*, тип целый)	Данные
4	Анализ введенных данных (значение переменной *A* отрицательное)	Решение
5	Присваивание значения (переменной *A* присваивается значение *A*+2)	Процесс
6	Вывод данных (полученное значение переменной *A,* тип целый)	Данные
7	Конец алгоритма	Терминатор

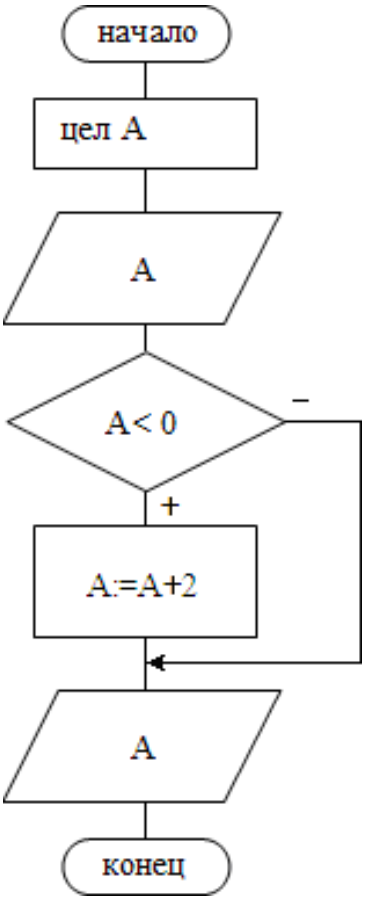
Реализация

Представим алгоритм в виде графической схемы.

Используемые в алгоритме графические символы представлены в табл. 5 (правый столбец).

Для отображения потока данных и управления используется стандартный символ «Линия».

Таким образом, на основе табл. 5 получим следующую графическую схему алгоритма решения задачи.



Пример использования управляющей структуры «Ветвление» (полная форма)

Задача. Разработайте алгоритм, который проверяет введенное пользователем значение величины целого типа *A,* и если оно отрицательное, то увеличивает введенное значение на 2, иначе уменьшает его на 4, а затем выводит на экран полученное значение *A*.

Решение.

Проектирование

1. Постановка задачи.

Входные данные: величина A целого типа, **цел** A;

Выходные данные: величина A целого типа, **цел** A;

Связь между входными и выходными данными:

если A<0

то A:=A+2

иначе A:=A-4

2. Проект решения задачи сведен в табл. 6.

Таблица 6. Проект решения задачи

	Выполняемые действия	Наименование графического символа
1	Начало алгоритма	Терминатор
2	Описание используемых переменных	Процесс
3	Ввод данных (ввод значения переменной *A*, тип целый)	Данные
4	Анализ введенных данных (значение переменной *A* отрицательное)	Решение
5	Присваивание значения (переменной *A* присваивается значение *A*+2)	Процесс
6	Присваивание значения (переменной *A* присваивается значение *A*-4)	Процесс
7	Вывод данных (вывод полученного значения переменной *A*, тип целый)	Данные
8	Конец алгоритма	Терминатор

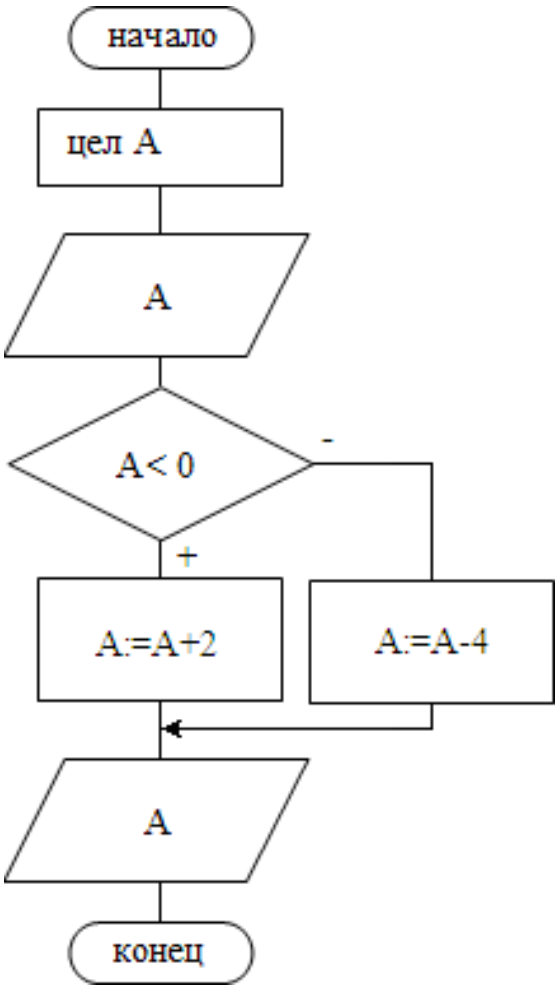
Реализация

Представим алгоритм в виде графической схемы.

Используемые в алгоритме графические символы описаны в табл. 6 (правый столбец).

Для отображения потока данных и управления используется стандартный символ «Линия».

Таким образом, на основании табл. 6 получим следующую графическую схему алгоритма решения задачи:



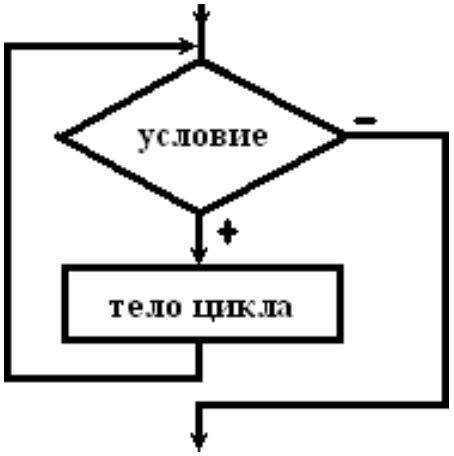
Управляющая структура «Цикл ПОКА»

Управляющая структура «Цикл ПОКА» используется в тех случаях, когда исполнитель должен многократно исполнять последовательность (тело цикла) в зависимости от значения истинности некоторого условия.

Особенность управляющей структуры «Цикл ПОКА» состоит в том, что разработчику алгоритма не известно, сколько раз необходимо исполнять последовательность (тело цикла).

Общий вид управляющей структуры «Цикл ПОКА»

в виде блок-схемы:



на псевдокоде:

пока <условие >
нц
<тело цикла>
кц

Условие – это логическое выражение, принимающее значение истина (True) или ложь (False).

Исполнитель проверяет условие (вычисляет значение истинности) только перед исполнением тела цикла, поэтому «Цикл ПОКА» называется циклом с предусловием, а само условие называется условием продолжения исполнения тела цикла. Изменение значения истинности условия разработчик алгоритма предусматривает командой внутри тела цикла.

Возможны три случая исполнения управляющей структуры «Цикл ПОКА»:

1. Если при первой проверке значение истинности условия ложь, то исполнитель не исполняет тело цикла ни одного раза.

2. Если условие принимает значение истина конечное количество раз, то и тело цикла исполняется столько же раз.

3. Если условие всегда принимает значение истина, то тело цикла исполняется бесконечное количество раз (зацикливание).

Пример использования управляющей структуры «Цикл ПОКА»

Задача. Разработайте алгоритм, который вычисляет сумму S первых n натуральных чисел и выводит на экран полученное значение S.

Решение

Проектирование:

1. Постановка задачи:
   - входные данные: величина *n* целого типа, **цел** *n*;
   - выходные данные: величина S целого типа, **цел** S;
   - связь между входными и выходными данными:

     пока *i* <= *n*.

     нц *S* := *S* + *i*; *i*:=*i*+*1* **кц**

2. Проект решения задачи сведен в табл. 7.

Таблица 7. Проект решения задачи

	Выполняемые действия	Наименование графического символа
1	Начало алгоритма	Терминатор
2	Описание используемых переменных (целые *n, S, i*)	Процесс
3	Ввод данных (ввод значения переменной *n*, тип целый)	Данные
4	Присваивание значения (переменной *S* присваивается значение 0)	Процесс
5	Присваивание значения (переменной *i* присваивается значение 1)	Процесс
6	Анализ данных (значение переменной *i* не превосходит *n*)	Граница цикла
	Присваивание значения (переменной *S* присваивается значение *S* + *i*)	Процесс
	Присваивание значения (переменной *i* присваивается значение *i*+1)	Процесс
		Граница цикла
7	Вывод данных (вывод вычисленного значения переменной *S*, тип целый)	Данные
8	Конец алгоритма	Терминатор

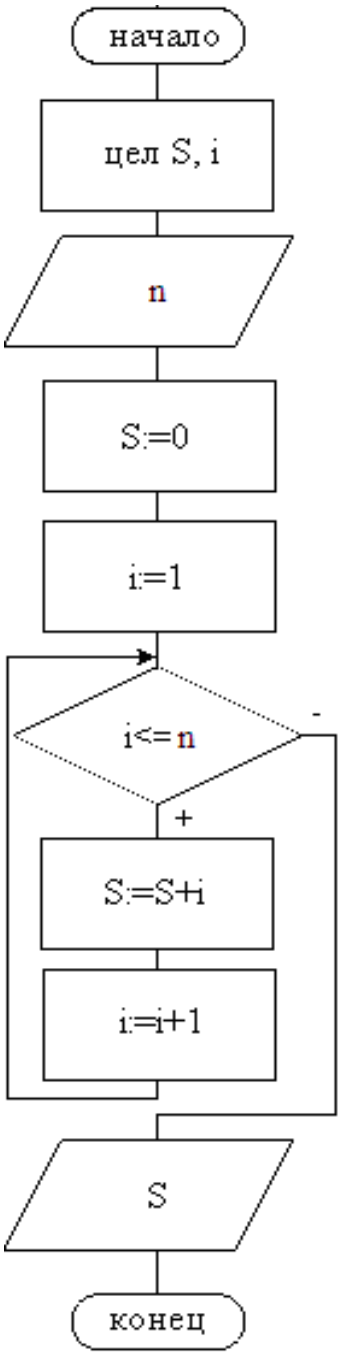
Реализация

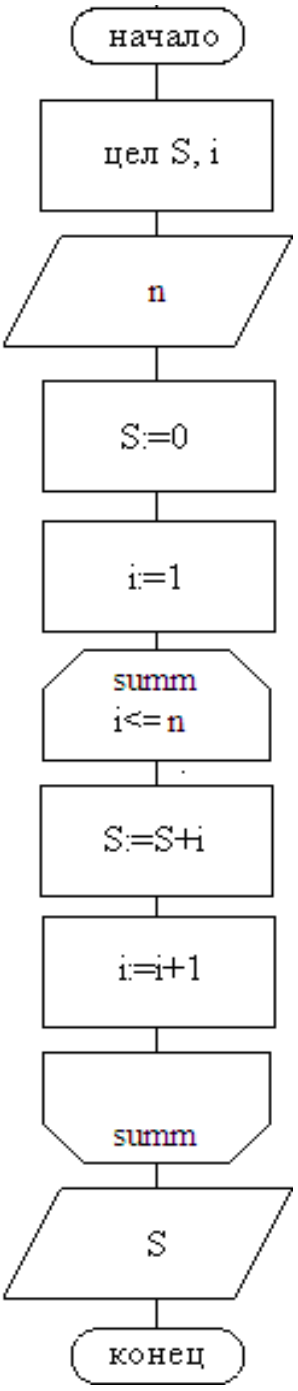
Представим алгоритм в виде графической схемы.

Используемые в алгоритме графические символы представлены в табл. 7 (правый столбец).

Для отображения потока данных и управления используется стандартный символ «Линия».

Таким образом, на основании табл. 7 получим следующую графическую схему алгоритма решения задачи:





Управляющая структура «Цикл ВЫПОЛНЯТЬ ДО»

Данная управляющая структура используется в тех случаях, когда исполнитель должен многократно исполнять последовательность команд тела цикла в зависимости от некоторого условия.

Особенность управляющей структуры «Цикл ВЫПОЛНЯТЬ ДО»:

1) исполнитель обязательно один раз исполнит последовательность команд тела цикла,

2) разработчику алгоритма не известно, сколько раз необходимо исполнять последовательность команд тела цикла.

Общий вид управляющей структуры «Цикл ВЫПОЛНЯТЬ ДО»

в виде блок-схемы:

на псевдокоде:

выполнять
<тело цикла (серия команд)>
до <условие >

Условие – это логическое выражение, принимающее значение истина (True) или ложь (False).

Исполнитель проверяет условие (вычисляет значение истинности) только после исполнением тела цикла, поэтому «Цикл ВЫПОЛНЯТЬ ДО» называется циклом с постусловием, а само условие называется условием завершения исполнения тела цикла. Изменение значения условия разработчик алгоритма предусматривает операцией внутри цикла.

Возможны три случая исполнения управляющей структуры «Цикл ВЫПОЛНЯТЬ ДО».

1. Если при первой проверке значения истинности условия истина (True), то исполнитель уже исполнил тело цикла один раз и завершает исполнение.

2. Если условие принимает значение истинности ложь (False) конечное количество раз, то и тело цикла исполняется столько же раз и +1 раз.

3. Если условие принимает значение истинности ложь (False), то и тело цикла исполняется бесконечное количество раз (зацикливание).

Пример использования управляющей структуры
«Цикл ВЫПОЛНЯТЬ ДО»

Задача. Разработайте алгоритм, который вычисляет сумму *S* первых *n* натуральных чисел и выводит на экран полученное значение *S*.

Решение.

Проектирование

1. Постановка задачи.

Входные данные: величина *n* целого типа, **цел** *n*;

Выходные данные: величина S целого типа, **цел** S;

Связь между входными и выходными данными:

выполнять
S:= S+i
i:=i+1
до i>n

2. Проект решения задачи сведен в таблицу 8.

Таблица 8. Проект решения задачи

	Выполняемые действия	Наименование графического символа
1	Начало алгоритма	Терминатор
2	Описание используемых переменных (целые *n, S, i*)	Процесс
3	Ввод данных (ввод значения переменной *n*, тип целый)	Данные
4	Присваивание значения (переменной S присваивается значение 0)	Процесс
5	Присваивание значения (переменной i присваивается значение 1)	Процесс
		Граница цикла
	Присваивание значения (переменной S присваивается значение S+i)	Процесс
	Присваивание значения (переменной i присваивается значение i+1)	Процесс
6	Анализ данных (значение переменной i превосходит n)	Граница цикла
7	Вывод данных (вывод полученного значения переменной S, тип целый)	Данные
8	Конец алгоритма	Терминатор

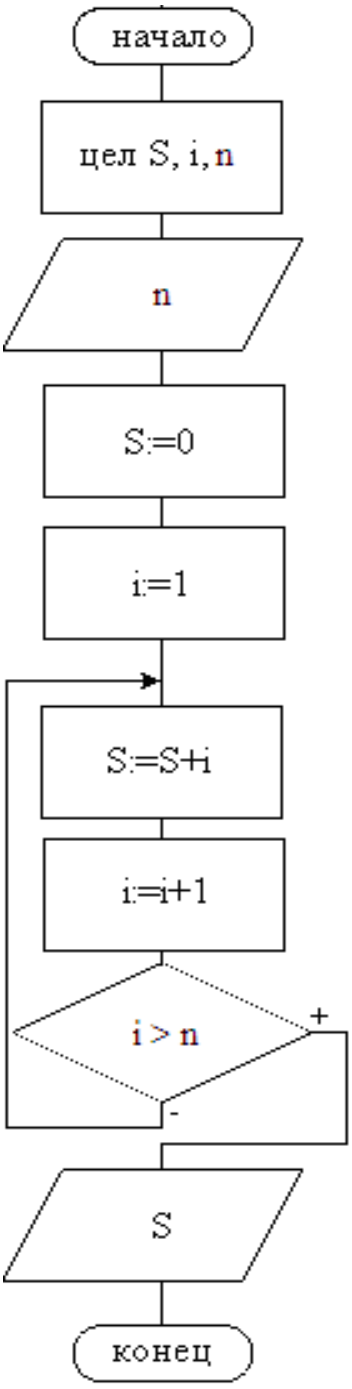
Реализация

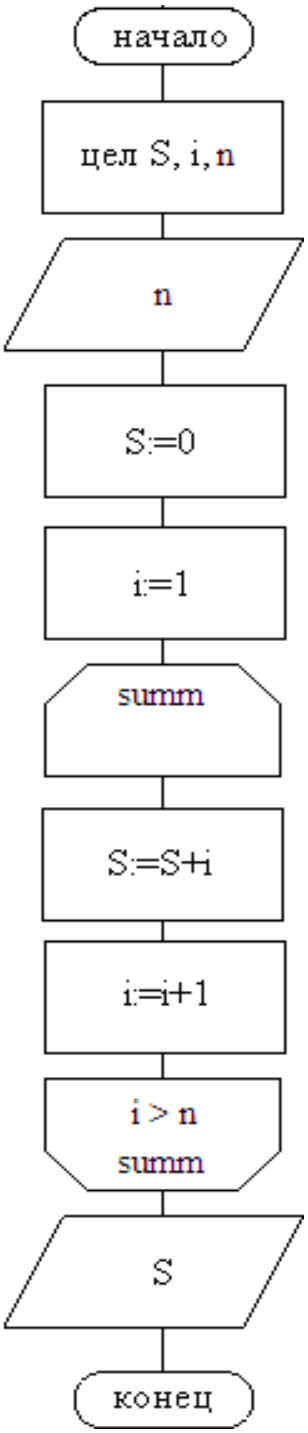
Представим алгоритм в виде графической схемы.

Используемые в алгоритме графические символы описанные в табл. 8 (правый столбец).

Для отображения потока данных и управления используется стандартный символ «Линия».

Таким образом, на основании табл. 8 получим следующую графическую схему алгоритма решения задачи:





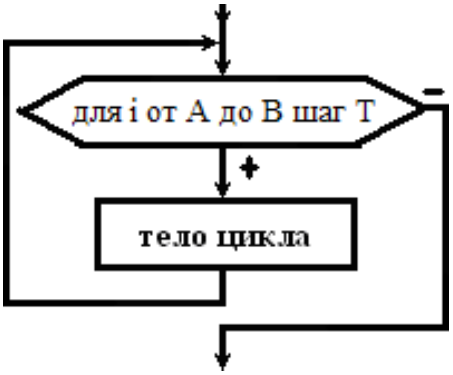
Управляющая структура «Цикл ДЛЯ»

Управляющая структура «Цикл ДЛЯ»используется в тех случаях, когда исполнитель должен многократно исполнять последовательность команд тела цикла в зависимости от значения истинности некоторого условия.

Особенность управляющей структуры «Цикл ДЛЯ» – разработчику алгоритма заранее известно, сколько раз необходимо выполнить последовательность команд тела цикла.

Общий вид управляющей структуры «Цикл ДЛЯ»

в виде блок-схемы:



на псевдокоде:

<u>для</u> *i* **<u>от</u>** *A* **<u>до</u>** *B* **<u>шаг</u>** *T*
<u>нц</u>
<тело цикла (серия команд)>
<u>кц</u>

В управляющей структуре «Цикл ДЛЯ»:

i – управляющая переменная цикла (другое название – параметр цикла), величина целого типа.

T – шаг; исполнитель изменяет текущее значение управляющей переменной цикла на значение величины *T* перед очередным исполнением серии команд (увеличивает или уменьшает). Если *T* задано выражением, то исполнитель вычисляет его значение только один раз перед первым исполнением серии команд. Значение *T* не может быть равно нулю.

A – начальное значение управляющей переменной цикла; исполнитель присваивает управляющей переменной цикла *i* значение *A* в момент начала исполнения управляющей структуры. Если *A* задано выражением, то исполнитель вычисляет его значение только один раз перед первым исполнением серии команд.

B – конечное значение управляющей переменной цикла; исполнитель сравнивает с ним текущее значение управляющей переменной цикла *i* перед очередным исполнением серии команд. Если *B* задано выражением, то исполнитель вычисляет его значение только один раз перед первым исполнением серии команд.

УСЛОВИЕ продолжения исполнения тела цикла:

$$(T>0)\ \text{AND}\ (i<=B)\ \text{OR}\ (T<0)\ \text{AND}\ (i>=B).$$

Исполнитель вычисляет значение истинности УСЛОВИЯ только перед исполнением тела цикла, поэтому цикл «ДЛЯ» исполняется также, как цикл «ПОКА».

Пример использования управляющей структуры
«Цикл ДЛЯ»

Задача. Разработайте алгоритм, который вычисляет сумму *S* первых *n* натуральных чисел и выводит на экран полученное значение *S*.

Решение.

Проектирование

1. Постановка задачи.

Входные данные: величина n целого типа, **цел** n;

Выходные данные: величина S целого типа, **цел** S;

Связь между входными и выходными данными:

для *i* **от** *1* **до** *n* **шаг** *1*

нц

S:=S+i

кц

2. Проект решения задачи сведен в табл. 9.

Таблица 9. Проект решения задачи

	Выполняемые действия	Наименование графического символа
1	Начало алгоритма	Терминатор
2	Описание используемых переменных (целые *n*, *S*, *i*)	Процесс
3	Ввод данных (ввод значения переменной *n*, тип целый)	Данные
4	Присваивание значения (переменной *S* присваивается значение 0)	Процесс
5	Анализ данных (значение переменной *i* от 1 до *n*)	Граница цикла
	Присваивание значения (переменной *S* присваивается значение *S* + *i*)	Процесс
		Граница цикла
6	Вывод данных (вывод вычисленного значения переменной величины *S*, тип целый)	Данные
7	Конец алгоритма	Терминатор

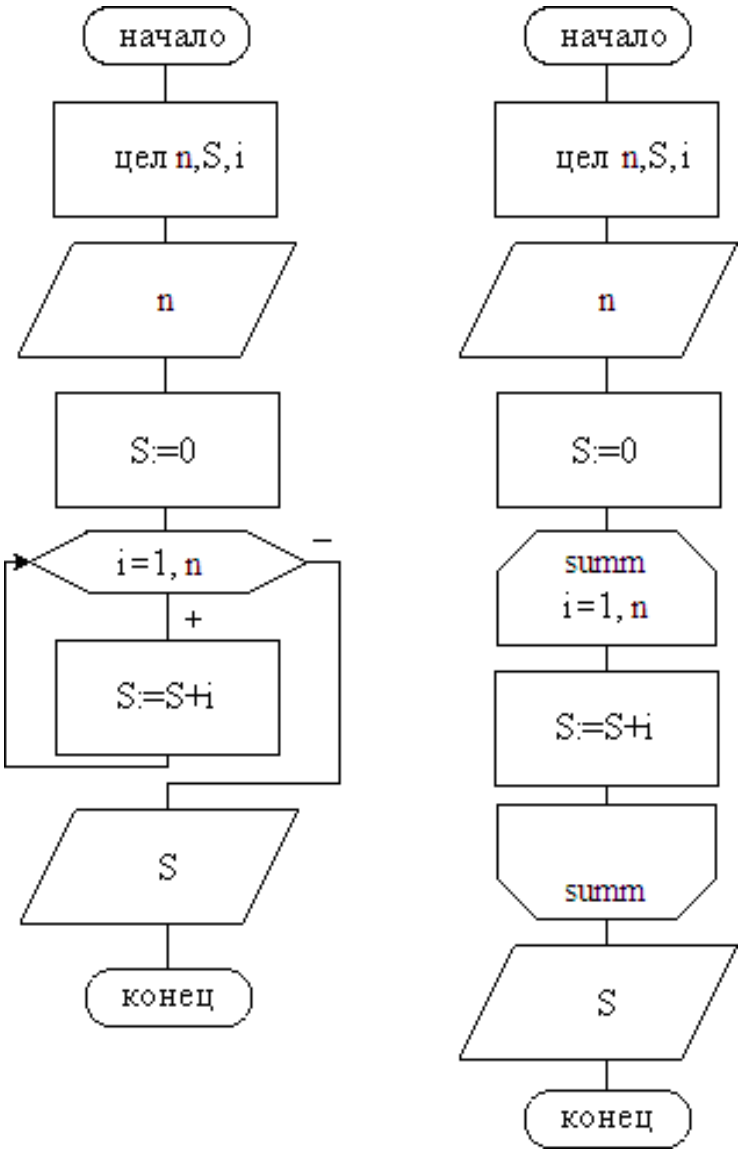
Реализация

Представим алгоритм в виде графической схемы.

Используемые в алгоритме графические символы описаны в табл. 9 (правый столбец).

Для отображения потока данных и управления используется стандартный символ «Линия».

Таким образом, на основании табл. 9 получим следующую графическую схему алгоритма решения задачи:



Контрольные вопросы

1. Что такое алгоритм?
2. Перечислите характерные свойства алгоритма и поясните их смысл.
3. Что такое алгоритмические модели? Перечислите их типы.
4. Что достаточно для получения оценок сложности алгоритмов?
5. Чем характеризуется исполнитель алгоритмов?
6. Перечислите способы представления алгоритмов.
7. Перечислите основные управляющие структуры.
8. В чем различие полной и сокращенной формы управляющей структуры «Ветвление»?
9. Назовите особенность управляющей структуры «Цикл ПОКА».
10. Назовите особенность управляющей структуры «Цикл ВЫПОЛНЯТЬ ДО».
11. Назовите особенность управляющей структуры «Цикл ДЛЯ».

Разработка алгоритмов

Общепринято, что разработка алгоритма состоит из следующей последовательности работ:

1. Разработка требований – определена условием задачи.
2. Проектирование – определение основных операций с последующим уточнением каждой из них.
3. Реализация – выбор способа записи (представления) алгоритма и запись по правилам этого способа.
4. Тестирование и отладка – выявление ошибок и их устранение.

Разработка требований

Требования к алгоритму, процессу ввода и вывода данных, как правило, определены решаемой задачей.

Проектирование

Проектирование начинают с постановки задачи. Необходимо выяснить: Что дано? Что требуется? Что будет считаться результатом (решением задачи)? Каким условиям должны удовлетворять исходные данные? Каким условиям должен удовлетворять результат?

Затем следует теоретический анализ задачи:

− построение модели предметной области (выделение объектов, установление их свойств и отношений между ними);

− определение структуры входных и выходных данных;

− определение и описание процессов и действий, допустимых в предметной области;

− выявление известных методов решения задачи;

− определяется точная последовательность выполнения предписаний, определяемых методом решения.

Реализация

Среди современных инструментальных средств разработки программ существуют системы, допускающие реализацию алгоритма до получения программного кода. На лабораторных работах для реализации разрабатываемых алгоритмов (а также их тестировании и отладки) будем использовать интегрированную систему «Конструктор схем алгоритмов» («Schemes»), которая разработана в ВЦ-ЦНТО Пермского государственного национального исследовательского университета.

Тестирование и отладка

«Тестирование может показать наличие ошибок в программе, но никогда не докажет их отсутствие»

Эдсгер Дейкстра

После разработки алгоритма переходят к этапу «Тестирование».

Главная цель тестирования – обнаружить факт наличия в алгоритме синтаксических и семантических (смысловых) ошибок и определить их место, т. е. необходимо проверить, делает ли алгоритм то, для чего он предназначен, а также, не делает ли он того, чего не должен делать.

Тестирование делится на этапы: выбор подхода к тестированию, проектирование и разработка тестовых наборов, проверка тестов, исполнение алгоритма на тестовых наборах, анализ результатов.

Планировать тестирование нужно еще при разработке алгоритма. Разрабатывая какой-либо фрагмент алгоритма, необходимо представлять себе, как он будет тестироваться, исходя из предположения, что в алгоритме есть ошибки. Это позволяет:

- обеспечить непрерывность этапов создания алгоритма: сначала определяются входные данные, затем они используются при разработке, затем выполняется готовый алгоритм, давая ожидаемые результаты;
- лучше понимаются ограничения на входные данные;
- устраняется возможность подгонки тестов к уже разработанному алгоритму;
- уменьшается вероятность внесения одних и тех же ошибок и в алгоритм, и в тестовые данные.

На этапе «Тестирование» несколько раз исполняют алгоритм с такими тестовыми наборами входными данными, для которых результаты работы легко проверить.

Тестовый набор – совокупность исходных данных и ожидаемых результатов. Создавая набор тестов, необходимо для каждого теста описать ожидаемые значения выходных данных или результатов их обработки. Нельзя упрощать алгоритм, чтобы облегчить тестирование. Очень важно учитывать общеизвестный факт: ошибки имеют свойство накапливаться. Отсюда парадоксальный вывод – чем больше ошибок найдено в некотором фрагменте алгоритма, тем больше шансов, что там есть еще. Поэтому этот фрагмент алгоритма нужно протестировать еще раз.

Тестирование выполняется с помощью:

1. использование спецификации алгоритма (другие названия – метод *черного ящика*, метод *управление по данным*);
2. использование логики алгоритма (другие названия метод *белого ящика,* метод *управление по логике*).

При тестировании с использованием спецификации алгоритм рассматривается, как черный ящик, о внутреннем устройстве которого ничего не известно. Этот подход предназначен для выяснения обстоятельств, в которых поведение алгоритма не соот-

ветствует его спецификации. Крайним случаем такого подхода является *исчерпывающее входное тестирование* – в качестве тестов используются все возможные наборы данных. Однако это практически невозможно, поэтому нужно искать наборы входных данных, которые ограничены разумными пределами, но достаточно представительны.

При тестировании с использованием логики алгоритма анализируется его логика. Этот подход предназначен для выявления путей исполнения алгоритм, на которых он «дает отказ». Крайний случай такого подхода – *исчерпывающее тестирование путей.* Однако даже в простых алгоритмах число путей может быть так велико, что все их трудно перебрать, а тем более, построить для них наборы тестовых данных.

При создании тестового набора для любого метода тестирования необходимо:

1) учесть область применения программы, т.е. входные данные необходимо подбирать с учетом предметной области;

2) определить ограничения на входные данные;

3) указать «особые» ситуации (например, значения которые могут привести к ошибкам при вычислениях значений по формулам);

4) формировать тестовые наборы с учетом типа входных данных;

5) совокупность тестовых наборов должна исчерпывать все возможные «варианты прохождения» алгоритма (для алгоритмов с ветвлениями и циклическими структурами).

Получив результаты тестирования, внимательно их изучите. Каждое несоответствие полученных и ожидаемых выходных данных дает информацию об ошибках в алгоритме, необходимую для его отладки.

Отладка состоит во внесении изменений и исправлений в алгоритм для устранения найденных ошибок. При этом следует помнить, что:

- исправления могут привести к появлению новых ошибок;
- где есть одна ошибка, там может быть и еще одна;
- вносите изменения по одному и поочередно.

Тестирование и отладку выполняют неоднократно друг за другом до тех пор, пока не будут устранены все обнаруженные ошибки.

Контрольные вопросы

1. Перечислите этапы проектирования алгоритмов.
2. На каком этапе решения задачи разрабатывают информационную модель?
3. В чем заключается построение алгоритма?
4. Назовите два основных подхода, используемых на этапе тестирования.
5. Что называют отладкой? Перечислите принципы отладки.
6. Что называют тестированием? Перечислите этапы тестирования.
7. Что является главной целью тестирования?
8. Что происходит в процессе тестирования?

9. Что такое тестовый набор? Перечислите требования, предъявляемые к тестовому набору.
10. Какие подходы используются при разработке тестов?
11. Что такое метод белого и черного ящика?
12. Что такое исчерпывающее входное тестирование? В каком случае его применяют?

Классы сложности задач

В основе классификации задач по сложности лежит вопрос о том, как быстро растет время, необходимое для решения любой задачи, с ростом ее объема. Эта зависимость измеряется числом элементарных шагов, входящих в алгоритм решения задачи.

Например, перемножение двух ***n***-разрядных чисел можно выполнить за время, пропорциональное квадрату числа разрядов. Принято считать, что это время «*полиномиально по* ***n***». Но разложение числа на множители даже самыми современными методами требует времени растущего с числом разрядов как $2^{\sqrt[3]{n}}$. Принято считать, что это время *«экспоненциально по* ***n****»*.

Современные компьютеры эффективно решают задачи, число шагов в алгоритме которых растет в постоянной степени, например

$$n^2 \text{ или } n^{(2,5)},$$

где ***n*** **–** размер входных данных.

Такие алгоритмы называют эффективными, а задачи, решаемые с их помощью, относят к *классу сложности P,* где *P* обозначает «полиномиальное время».

Примеры задач класса ***P***:

1) простая задача: по заданной сетке дорог определить, можно ли из одного города добраться до любого другого;

2) задачи, решение которых не столь очевидно:

- раскладывается ли данное число на множители или является простым;
- возможно ли, имея определенный список мужчин и женщин, желающих вступить в брак друг с другом, подобрать каждому человеку желанного партнера?

3) более сложные задачи:

- о размещении коробок различных размеров в багажнике автомобиля;
- как раскрасить страны на карте тремя различными цветами, при условии, что государства, окрашенные одним и тем же цветом, нигде не должны граничить друг с другом;
- «задача коммивояжера», в которой требуется объехать группу городов, соединенных между собой дорогами – не посетить ни один из городов больше одного раза.

Все три разновидности задачи являются «одинаковыми» в том смысле, что эффективный алгоритм решения любой одной из них будет эффективным и для всех остальных

В класс ***NP*** (*NP* означает «*недетерминированное полиномиальное время*») входят задачи, для которых решение, если оно будет предложено, может быть проверено за полиномиальное время, даже если найти само решение, – очень сложная задача. Класс ***NP*** охватывает огромное число задач, представляющих практический интерес.

Например, известна карта с множеством островов и мостов, для нахождения способа объехать их все таким образом, чтобы ни один из них не посетил больше одного раза, могут потребоваться годы. Но если такой маршрут уже предложен, проверить, правилен ли он, будет несложно.

Самыми сложными в классе ***NP*** являются переборные, или ***NP***-полные задачи, обладающие свойством: если найден эффективный алгоритм решения одной из них, то он может быть использован для решения всех остальных ***NP*** – задач.

Создание такого алгоритма означало бы, что современные представления о классах ***P***, ***NP*** и ***NP***-полных задачах совершенно неверны, и все задачи класса ***NP***, включая переборные, в действительности относятся к классу ***P***, что, в свою очередь, означает тождественность классов ***P*** и ***NP***.

Очевидно, что все задачи класса ***P*** входят также в класс ***NP,*** так как на проверку правильности решения каждой уходит не больше времени, чем на ее решение.

Сложность алгоритма

Вычислительным процессом, порожденным алгоритмом, называется последовательность шагов алгоритма, пройденных при исполнении этого алгоритма.

Сложность алгоритма – это количественная характеристика, свидетельствующая о том, сколько времени он работает, либо – какой объем памяти ему необходим.

Виды сложности алгоритмов:

1) вычислительная (временная);
2) объемная (емкостная) сложность – определяется количеством скалярных переменных, элементов массивов, элементов записей или просто количеством байт);
3) сложность текста алгоритма – характеризует исполнителя (например ЭВМ), его язык, а не метод решения задачи;
4) логическая сложность – определяется количеством человеко-месяцев, которое нужно потратить на создание алгоритма (программы для ЭВМ), поскольку не представляется возможным дать объективные количественные характеристики.

Вычислительная сложность алгоритма – количество элементарных шагов в вычислительном процессе этого алгоритма. Обратите внимание: именно в вычислительном процессе, а не в самом алгоритме. Очевидно, для сравнения сложности разных алгоритмов необходимо, чтобы сложность подсчитывалась в одних и тех же элементарных действиях.

Временная сложность алгоритма – это время T, необходимое для его выполнения. Оно равно произведению числа элементарных действий на среднее время выполнения одного действия:

$$T = k \cdot t.$$

Поскольку ***t*** зависит от исполнителя, реализующего алгоритм, то естественно считать, что сложность алгоритма в первую очередь зависит от ***k***. Очевидно, что в наибольшей степени количество операций при выполнении алгоритма зависит от количества обрабатываемых данных. Действительно, для упорядочивания по алфавиту списка из 100 фамилий требуется существенно меньше операций, чем для упорядочивания списка из 100 000 фамилий. Поэтому сложность алгоритма выражают в виде функции от объема входных данных.

Пусть задан алгоритм ***A***. Для него существует параметр ***n***, характеризующий объем обрабатываемых алгоритмом данных и называемый *размерностью задачи*.

Обозначим:

T(n) – время выполнения алгоритма,

f – некую функцию от ***n***.

Будем говорить, что

T(n) – время выполнения алгоритма ***A*** имеет порядок роста ***f(n)*** при $n \to \infty$

или, по-другому,

алгоритм имеет **теоретическую сложность *O((f (n))*** (читается «о большое от ***f(n)»***),

если найдется такая константа $c > 0$ и число n_0, что

$$T(n) <= c\,f(n) \quad \text{при всех} \quad n >= n_0 .$$

Здесь предполагается, что ***f(n)*** неотрицательно, по крайней мере при $n >= n_0$.

Например:

– алгоритм, выполняющий только операции чтения данных из файла в оперативную память, имеет ***линейную*** сложность ***O(n)***;

– алгоритм сортировки методом «пузырька» (см. «*Операции с массивами*») имеет ***квадратичную*** сложность $O(n^2)$, так как при сортировке любого массива надо выполнить

$$(n^2 - n)/2$$

операций сравнения (при этом операций перестановок вообще может не быть, например, на упорядоченном массиве).

Для решения задачи могут быть разработаны алгоритмы различной сложности. Логично воспользоваться лучшим среди них, т.е. имеющим наименьшую сложность.

Объемная (емкостная) сложность алгоритма – определяется количеством ячеек памяти, используемых в процессе его работы. Число шагов алгоритма может сколь угодно сильно превосходить объем памяти за счет циклов по одним и тем же объектам. Поэтому временная сложность считается основной характеристикой алгоритма.

Сложность текста алгоритма может быть оценена на основе анализа его текста. При этом для определенности полагают, что алгоритм представлен на конкретном языке программирования (типа Паскаля) и содержит явно записанные операции пересылки и сравнения значений величин, арифметические операции, управляющие структуры ветвления и циклов, а также их композиции.

Пример. Алгоритм содержит цикл:

для *i* **от** 1 **до** *x* **шаг** 1
нц
<тело цикла>
кц

где *x* – входная переменная, $x \in [1, 100]$.

Тело цикла выполняется *x* раз, худший случай при *x* = 100. Если предположить равновероятность различных значений *x*, то среднее количество выполнений тела цикла равно

$$(1/100)(1+2+3+ \ldots+100) = 50{,}5.$$

Пример. Алгоритм содержит вложенный цикл:

для *i* **от** 1 **до** *x* **шаг** 1
нц
для *j* **от** *i* **до** *x* **шаг** 1
нц
<тело цикла>
кц
кц

где *x* – входная переменная, $x \in [1, 5]$.

Тело цикла выполняется

$$x + (x\text{-}1)+(x\text{-}2)+ \ldots+ 1 = x\,(x+1)/2 \text{ раз.}$$

Верхняя граница сложности

$$max = (x\,(x+1)/2) = 13.$$

Среднее значение сложности подсчитать несколько труднее. Предположим, что все 5 возможных значений *x* равновероятны. Тогда среднее значение сложности равно:

$$\sum_{x=1}^{5} \frac{1}{5} \frac{x(x+1)}{2} = \frac{1}{10} \sum_{x=1}^{5} x(x+1) = 7$$

Управляющая структура «Цикл ПОКА»

Циклы **while** и **repeat** анализировать значительно сложнее, чем цикл «ДЛЯ», поскольку количество исполнений тела цикла зависит от значения истинности или ложности условия и не исключены изменения в теле цикла значений переменных, входящих в состав условия.

Сложность алгоритма можно оценить и на основе анализа его графической схемы.

Пример анализа сложности по схеме алгоритма. На рис. 1–3 представлен алгоритм поиска наибольшего из трех чисел *a, b, c*.

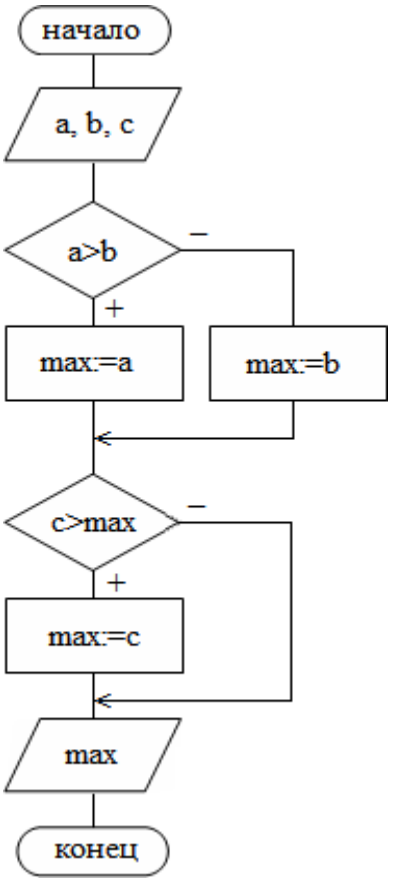


Рис. 1.Схема алгоритма поиска наибольшего из трех чисел. Метод 1.

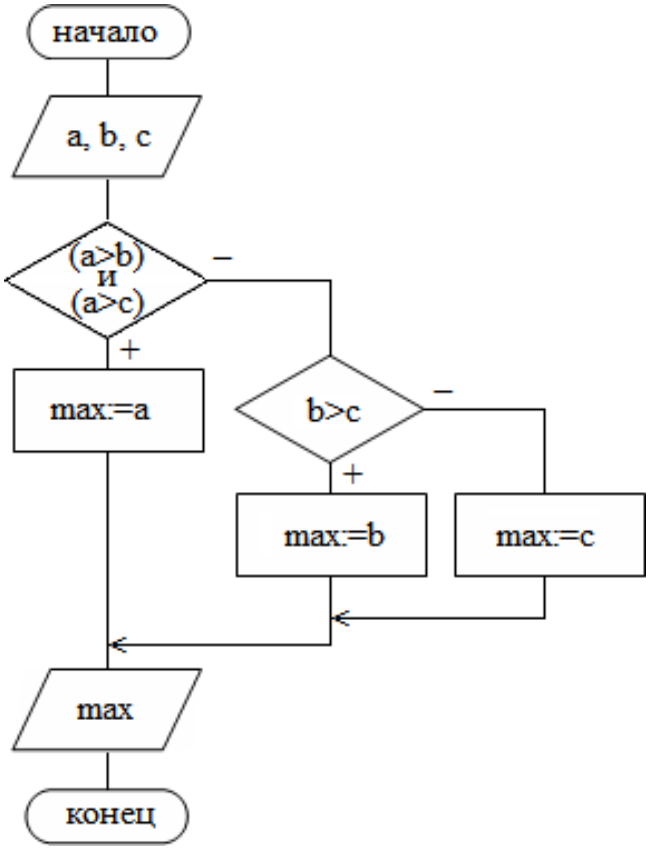


Рис. 2. Схема алгоритма поиска наибольшего из трех чисел. Метод 2.

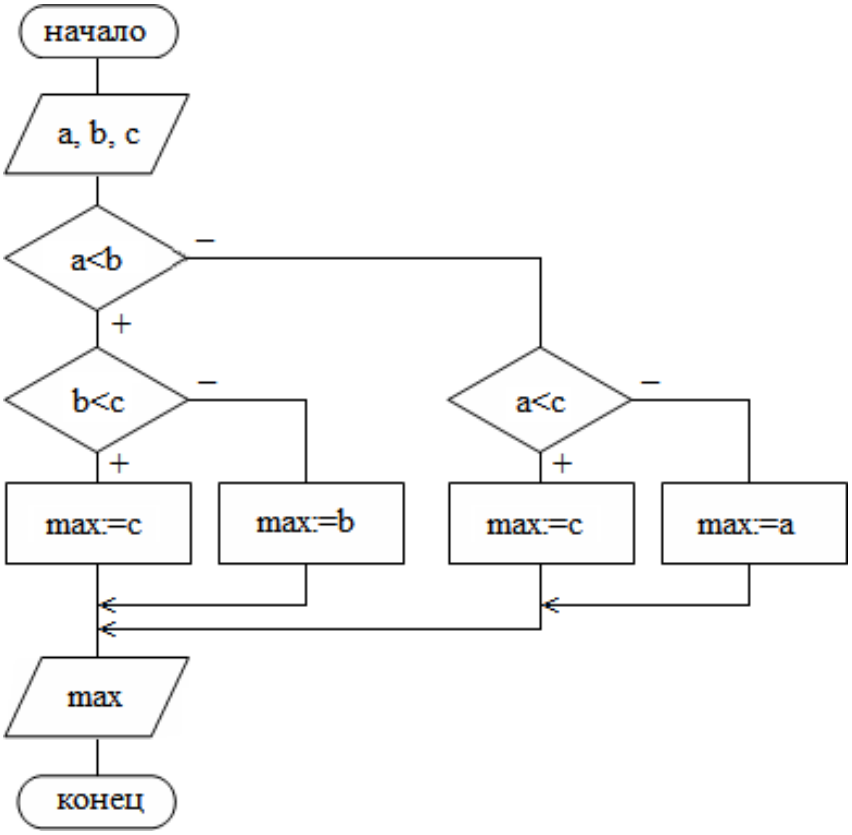


Рис. 3. Схема алгоритма поиска наибольшего из трех чисел. Метод 3.

Определим, какой алгоритм эффективнее. Поскольку размерность данных во всех алгоритмах одинаковая, то объемная сложность алгоритма одинаковая.

Оценим эффективность по вычислительной сложности (см. табл. 10).

Таблица 10. Оценка эффективности по вычислительной сложности

Вычислительная сложность	Операция	Метод 1	Метод 2	Метод 3
В лучшем случае	сравнение	2	2	2
	присваивание	1	1	1
В худшем случае	сравнение	2	3	2
	присваивание	2	1	1

Сравнение алгоритмов показывает, что алгоритмы примерно равноценны, но при многократном использовании с разными исходными данными метод 3 даст наименьшее (в среднем) количество операций.

Контрольные вопросы

1. Что значит высказывание «Задача алгоритмически разрешима»?
2. Что значит высказывание «Задача алгоритмически не разрешима»?
3. Что понимают под сложностью алгоритма в общем случае?
4. Перечислите характеристики сложности алгоритмов.
5. Приведите пример алгоритма и объясните, как найти объемную и временную сложность.

Примеры выполнения заданий

Под заданием понимается вариант – набор задач по теме (темам), для которых студент должен разработать алгоритмы, отладить и протестировать их в интегрированной системе «Конструктор схем алгоритмов» («Schemes»). По результатам работы составить отчет.

Отчет о выполнении варианта для каждой задачи должен содержать:

1) условие задачи;

2) алгоритм решения задачи в виде графической схемы алгоритма;

3) тестовый набор, содержащий варианты значений входных данных и ожидаемых результатов (необходимо учесть все варианты выполнения алгоритма);

4) оценку объемной сложности разработанного алгоритма. Учитывать только объем памяти необходимый для хранения входных данных и результатов;

5) оценку временной сложности разработанного алгоритма.

Рассмотрим пример оформления отчета о выполнении варианта, содержащего три задачи:

− задачу №1 на управляющую структуру «следование»,

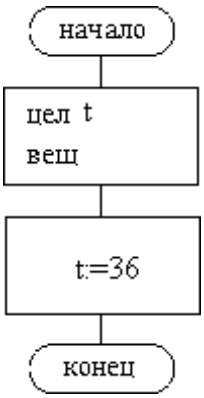
− задачу №2 на управляющую структуру «ветвление»,

− задачу №3 на управляющую структуру «цикл пока».

Задача №1 на управляющую структуру «следование».

1. Разработайте алгоритм для присваивания переменной величине *t* целого типа значения 36.

2. Алгоритм решения задачи в виде графической схемы алгоритма



3. Тестовые наборы

№ набора	Исходные данные	Результаты	
		ожидаемые	полученные
	цел *t*	**цел** *t*	**цел** *t*
1	36	36	

4. Оценка объемной сложности разработанного алгоритма

№ набора	Данные	Тип	Объем, байт
1	*t*	**цел**	2
		Итого	2

5. Оценку временной сложности разработанного алгоритма.

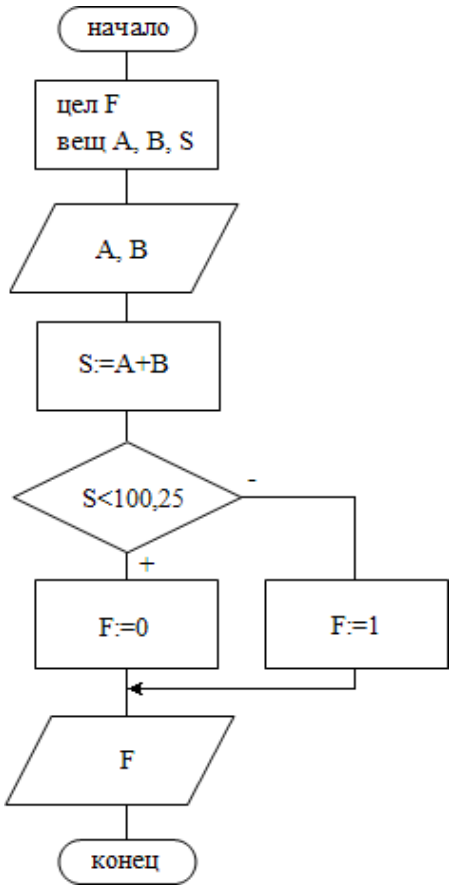
Операции	Тест 1
Присваиваний	1
Сравнений	0

Таким образом, алгоритм имеет сложность *O(n)*, т.е. линейную.

Задача №2 на управляющую структуру «ветвление»

1. Разработайте алгоритм, который проверяет сумму *S* введенных пользователем значений двух величин вещественного типа *A, B* и, если ее значение меньше 100,25, то значению флага *F* присвоить 0, иначе – 1. Затем алгоритм выводит на экран полученное значение флага *F*. Величина *F* целого типа.

2. Алгоритм решения задачи в виде графической схемы алгоритма



3. Тестовые наборы

№ набора	Исходные данные		Результаты			
			промежуточные		ожидаемые	полученные
			ожидаемые	полученные		
	вещ *A*	**вещ** *B*	**вещ** *S*	**вещ** *S*	**цел** *F*	**цел** *F*
1	- 4,2	- 1,6	- 5,8		0	
2	- 1	0,5	- 0,5		0	
3	28,6	- 130	- 101,4		0	
4	13,5	56,7	70,2		0	
5	56	61,2	117,2		1	

Тестовый набор составлен так, чтобы смоделировать работу алгоритма по обеим ветвям управляющей структуры «Ветвление».

4. Оценка объемной сложности разработанного алгоритма

№ набора	Данные	Тип	Объем, байт
1	*A*	**вещ**	4
2	*B*	**вещ**	4
3	*S*	**вещ**	4
4	*F*	**цел**	2
		Итого	14

5. Оценку временной сложности разработанного алгоритма

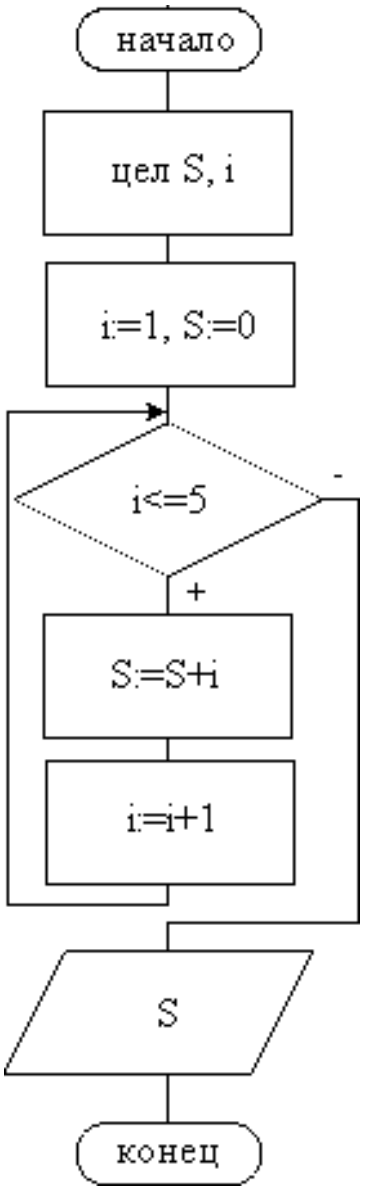
Операция	Тест			
	1	2	3	4
Присваивание	2	2	2	2
Сравнение	1	1	1	1

Таким образом, алгоритм имеет сложность *O(n)*, т.е. линейную.

Задача №3 на управляющую структуру «цикл пока»

1. Найти сумму пяти первых натуральных чисел. Результат вывести на экран.

2. Алгоритм решения задачи в виде графической схемы алгоритма.



3. Тестовые наборы

Исходные данные		Результат
цел i	цел S	цел S
1	0	15

4. Оценка объемной сложности разработанного алгоритма

№ набора	Данные	Тип	Объем, байт
1	S	**цел**	2
2	*i*	**цел**	2
		Итого	4

5. Оценку временной сложности разработанного алгоритма

Операции	Тест 1
Присваиваний	12
Сравнений	6

Таким образом, алгоритм имеет сложность *O(n)*, т.е. линейную.

Задачи для самостоятельного выполнения

Некоторые из приведенных задач заимствованы из интегрированной системы «Конструктор схем алгоритмов» («Schemes»), разработанной в ВЦ–ЦНТО ПГНИУ.

Управляющая структура «Следование»

Уровень А

1. Разработайте алгоритм для присваивания переменной m действительного типа значения 36,6.
2. Разработайте алгоритм для присваивания переменной summa целого типа значения 0.
3. Разработайте алгоритм для присваивания переменной A целого типа значения 15.
4. Разработайте алгоритм для присваивания переменной X действительного типа значение -1,5.
5. Разработайте алгоритм для присваивания величине i целого типа значения 1.
6. Разработайте алгоритм для присваивания величине S целого типа значения 0.
7. Разработайте алгоритм для присваивания величине P целого типа значения 1.
8. Разработайте алгоритм для присваивания величине S действительного типа значения 10,5.
9. Разработайте алгоритм для присваивания величине M действительного типа значения -0,5.
10. Разработайте алгоритм для присваивания величине h действительного типа значения 126,5.
11. Разработайте алгоритм для ввода в память ЭВМ значений величины R целого типа и вывода этих значений из памяти.
12. Разработайте алгоритм для ввода в память ЭВМ значений величины B целого типа и вывода этих значений из памяти.
13. Разработайте алгоритм для ввода в память ЭВМ значений величины C целого типа и вывода этих значений из памяти.
14. Разработайте алгоритм для ввода в память ЭВМ значений величины D целого типа и вывода этих значений из памяти.
15. Разработайте алгоритм для ввода в память ЭВМ значений величины E целого типа и вывода этих значений из памяти.
16. Разработайте алгоритм для ввода в память ЭВМ значений величины F целого типа и вывода этих значений из памяти.
17. Разработайте алгоритм для ввода в память ЭВМ значений величины Z целого типа и вывода этих значений из памяти.
18. Разработайте алгоритм для ввода в память ЭВМ значений величины A целого типа и вывода этих значений из памяти.
19. Разработайте алгоритм для ввода в память ЭВМ значений величины F действительного типа и вывода этих значений из памяти.

20. Разработайте алгоритм для ввода в память ЭВМ значений величины G действительного типа и вывода этих значений из памяти.
21. Разработайте алгоритм для ввода в память ЭВМ значений величины P действительного типа и вывода этих значений из памяти.
22. Разработайте алгоритм для ввода в память ЭВМ значений величины R действительного типа и вывода этих значений из памяти.
23. Разработайте алгоритм для ввода в память ЭВМ значений величины S действительного типа и вывода этих значений из памяти.
24. Разработайте алгоритм для ввода в память ЭВМ значений величины Y действительного типа и вывода этих значений из памяти.
25. Разработайте алгоритм для ввода в память ЭВМ значений величины W действительного типа и вывода этих значений из памяти.
26. Разработайте алгоритм для присваивания величине S целого типа значения 0 и вывода ее значения из памяти ЭВМ.
27. Разработайте алгоритм для присваивания величине B целого типа значения -5 и вывода ее значения из памяти ЭВМ.
28. Разработайте алгоритм для присваивания величине C целого типа значения 12 и вывода ее значения из памяти ЭВМ.
29. Разработайте алгоритм для присваивания величине D действительного типа значения -9 и вывода ее значения из памяти ЭВМ.
30. Разработайте алгоритм для присваивания величине P целого типа значения 1 и вывода ее значения из памяти ЭВМ.
31. Разработайте алгоритм для присваивания величине S действительного типа значения 1,1 и вывода ее значения из памяти ЭВМ.
32. Разработайте алгоритм для увеличения в половину введенного значения переменной действительного типа t.
33. Разработайте алгоритм для увеличения на единицу введенного значения переменной целого типа n.
34. Разработайте алгоритм для уменьшения в два с половиной раза введенного значения переменной действительного типа u.
35. Разработайте алгоритм для уменьшения на два введенного значения переменной действительного типа counter.
36. Разработайте алгоритм для присваивания величине F действительного типа значения 5.2, уменьшения его на два и вывода ее значения из памяти ЭВМ.
37. Разработайте алгоритм для ввода значений величин X, Y действительного типа, вычисления значений величины Z=X/Y, полученное значение величины Z сообщить пользователю.
38. Разработайте алгоритм для ввода значений величин X, Y целого типа, вычисления значений величины Z=X/Y, полученное значение величины Z сообщить пользователю.

39. Разработайте алгоритм для ввода значений величины X действительного типа и величины Y целого типа и вычисления значений величины Z=X/Y, полученное значение величины Z сообщить пользователю.
40. Разработайте алгоритм для ввода значений величины X целого типа и величины Y действительного типа и вычисления значений величины Z=X/Y, полученное значение величины Z сообщить пользователю.
41. Разработайте алгоритм для ввода значений величины X целого типа, величины Y действительного и вычисления значения величины Z = X - Y, полученное значение величины Z сообщить пользователю.
42. Разработайте алгоритм для присваивания величине действительного типа F значения 5.2, увеличения его в два раза и вывода ее значения из памяти ЭВМ.
43. Разработайте алгоритм для присваивания величине действительного типа K значения -20.2, уменьшения его в два раза и вывода ее значения из памяти ЭВМ.
44. Разработайте алгоритм для присваивания величине действительного типа A значения -2.2, увеличения ее значения на два, а затем - вывода из памяти ЭВМ.
45. Разработайте алгоритм для ввода значений величины X действительного типа и величины Y действительного типа и вычисления значений величины Z=X/Y, полученное значение величины Z сообщить пользователю.
46. Разработайте алгоритм для ввода значений величины X целого типа, присваивания величине Y действительного типа значения 5.5, вычисления значения величины Z = X - Y и вывода значения величины Z.
47. Разработайте алгоритм для ввода значений величины X целого типа, присваивания величине Y действительного типа значения 2.5, вычисления значений величины Z=X/Y и вывода значения величины Z.
48. Разработайте алгоритм для ввода значений величины X целого типа, присваивания величине Y действительного типа значения 5.5, вычисления значения величины Z = X + Y и вывода значения величины Z.
49. Разработайте алгоритм для ввода значений величины X целого типа, присваивания величине Y действительного типа значения 5.5, вычисления значений величины Z = X*Y и вывода значения величины Z.
50. Разработайте алгоритм для ввода значений величины X действительного типа, присваивания величине Y действительного типа значения 2.5, вычисления значений величины Z=X/Y и вывода значения величины Z.
51. Разработайте алгоритм пересчета введенного веса из фунтов в килограммы (один фунт равен 409,5 г).
52. Разработайте алгоритм пересчета введенного расстояния из километров в версты (одна верста равна 1066,8 м).
53. Разработайте алгоритм для вычисления среднего арифметического переменных целого типа x1 и x2.
54. Разработайте алгоритм для вычисления значения функции $y = (x+5)/x^2$ в заданной точке x (значение переменной x вводит пользователь).

55. Разработайте алгоритм для вычисления значения функции $y = 1/x^2$ в заданной точке x (значение переменной x вводит пользователь).
56. Разработайте алгоритм вычисления значения функции $y = x+5/x^2$ в заданной точке x (значение переменной x вводит пользователь).
57. Разработайте алгоритм вычисления значения функции $y = -2,7x^3 + 0,23 x^2 - 1,4$ в заданной точке x (значение переменной x вводит пользователь).
58. Разработайте алгоритм вычисления значения функции $y = -4x^4 + 2,7 x^3$ в заданной точке x (значение переменной x вводит пользователь).
59. Разработайте алгоритм вычисления суммы S двух введенных величин целого типа S и i.
60. Разработайте алгоритм вычисления произведения P двух введенных величин действительного P и целого i.

Уровень B

61. Разработайте алгоритм для вычисления площади прямоугольника.
62. Разработайте алгоритм для вычисления площади куба.
63. Разработайте алгоритм для пересчета сопротивления электрической цепи из Омов в килоОмы.
64. Разработайте алгоритм для пересчета расстояния из метров в километры.
65. Разработайте алгоритм для пересчета веса из килограммов в центнеры.
66. Разработайте алгоритм для пересчета веса из центнеров в тонны.
67. Разработайте алгоритм для нахождения дискриминанта D квадратного уравнения с коэффициентами a, b, c.
68. Разработайте алгоритм для вычисления среднего арифметического S четырех целых чисел A, B, C, D.
69. Разработайте алгоритм для вычисления сопротивления электрической цепи, состоящей из двух параллельно соединенных резисторов: $r = (r_1 r_2)/(r_1 + r_2)$.
70. Разработайте алгоритм для вычисления площади треугольника: $s = 1/2ah$, где a – длина основания треугольника, h – его высота.
71. Разработайте алгоритм для вычисления площади трапеции:
    $s = h * (a+b)/2$, где a и b – длины оснований, a – высота трапеции.
72. Разработайте алгоритм для вычисления площади поверхности цилиндра:
    $s = 2 * \pi * r(h + r)$; где r – радиус основания, h – его высота.
73. Разработайте алгоритм для вычисления объема цилиндра:
    $v = \pi * r^2 * h$, где r – радиус основания, h – его высота.
74. Разработайте алгоритм для вычисления площади крута: $s = \pi * r^2$.
75. Разработайте алгоритм для вычисления объема цилиндра.
76. Разработайте алгоритм для вычисления объема полого цилиндра по формуле $v = \pi h (r_1^2 - r_2^2)$, где r_1 – радиус цилиндра, r_2 – радиус отверстия, h — высота цилиндра.
77. Разработайте алгоритм для вычисления объема параллелепипеда.
78. Разработайте алгоритм для вычисления объема куба.

79. Разработайте алгоритм для вычисления объема конуса $s = 1/3\ \pi\ r^2\ h$, где r – радиус основания, h – высота.
80. Разработайте алгоритм для вычисления объема шара по формуле $v=3/4\ \pi\ r^3$.
81. Разработайте алгоритм для вычисления площади поверхности шара по формуле $S= 4 * \pi *r^2$.
82. Разработайте алгоритм для пересчета расстояния из километров K в версты W (одна верста равна 1066,8 м).
83. Разработайте алгоритм для пересчета расстояния из верст W в километры K (одна верста равна 1066,8 м).
84. Разработайте алгоритм для пересчета веса из фунтов F в килограммы K (один фунт равен 405,9 г).
85. Разработайте алгоритм для пересчета веса из килограммов K в фунты F (один фунт равен 405,9 г).
86. Разработайте алгоритм для вычисления среднего арифметического S четырех действительных чисел A, B, C, D.
87. Разработайте алгоритм для вычисления площади треугольника, если известны длины двух его сторон и величина угла между этими сторонами.
88. Разработайте алгоритм для вычисления дискриминанта (D/4) квадратного уравнения $ax^2 + bx + c = 0$, в котором b – четное число.
89. Из железной полосы длиной L метров нужно изготовить обруч. На соединение концов уходит D метров полосы. Разработайте алгоритм для вычисления радиуса R обруча.
90. Заданы величины X, Y, Z целого типа. Разработайте алгоритм для обмена значений величин по схеме X→Y, Y → Z, Z →X. Необходимо использовать вспомогательную величину T.
91. Заданы величины X, Y, Z действительного типа. Разработайте алгоритм для обмена значений величин и затем уменьшения значений величин X,Y в два раза. Необходимо использовать вспомогательную величину T.
92. Заданы величины X, Y, Z целого типа. Разработайте алгоритм для обмена значений величин и затем увеличения на два значений величин Y, Z. Необходимо использовать вспомогательную величину T.
93. Заданы величины X, Y целого типа. Разработайте алгоритм для обмена значений величин.
94. Заданы величины X, Y действительного типа. Разработайте алгоритм для обмена значений величин и уменьшения величины X в полтора раза.
95. Заданы величины X, Y, Z целого типа. Разработайте алгоритм для обмена значений величин и затем уменьшения значений величины Y в три раза. Необходимо использовать вспомогательную величину T.
96. Заданы величины X, Y, Z действительного типа. Разработайте алгоритм для обмена значений величин по схеме X → Z, Z → Y, Y→ X. Необходимо использовать вспомогательную величину T.
97. Разработайте алгоритм для вычисления среднего арифметического S четырех

величин A, B – действительного типа, C, D – целого типа.

98. Разработайте алгоритм для вычисления среднего арифметического S четырех величин A, B – целого типа, C, D – действительного типа.
99. Разработайте алгоритм для вычисления реального расстояния R (от пункта A до пункта B) по карте с масштабом M.
100. Разработайте алгоритм для пересчета реального расстояния R (от пункта A до пункта B) для нанесения на карту с масштабом M.

Уровень C

101. Дано натуральное число X. Вычислить $Y = 1 - 2*X + 3*X^2 - 4*X^3$. Разрешается использовать не более 8 арифметических операций. Допустимы операции: сложение, вычитание, умножение.
102. Дано натуральное число X. Вычислить $Y = X^5$. Разрешается использовать только три операции умножения.
103. Заданы величины X, Y действительного типа. Разработайте алгоритм для обмена значений величин. Использовать вспомогательные величины нельзя.
104. Разработайте алгоритм вычисления величины дохода по вкладу. Процентная ставка (в процентах годовых) и время хранения (в днях) задаются пользователем.
105. Разработайте алгоритм вычисления силы тока по известным значениям напряжения и сопротивления электрической цепи.
106. Разработайте алгоритм вычисления сопротивления электрической цепи по известным значениям напряжения и силы тока.
107. Разработайте алгоритм вычисления сопротивления электрической цепи, состоящей из трех последовательно соединенных резисторов.
108. Разработайте алгоритм вычисления стоимости покупки некоторого количества (по весу) помидоров, огурцов и яблок.
109. Разработайте алгоритм вычисления стоимости покупки нескольких тетрадей, карандашей и линейки.
110. Разработайте алгоритм для вычисления объема V параллелепипеда длиной L м, шириной G м и высотой H м.
111. Разработайте алгоритм для вычисления объема V цилиндра высотой H м и радиусом основания R м.
112. Разработайте алгоритм для вычисления падения напряжения U(B) на участке электрической цепи сопротивлением R (Ом) по известному значению силы тока I (A).
113. Разработайте алгоритм для вычисления площади поверхности S параллелепипеда длиной L м, шириной G м и высотой H м.
114. Разработайте алгоритм для вычисления силы тока I (A) по известным значениям напряжения U(B) и сопротивления R (Ом) электрической цепи.
115. Разработайте алгоритм для вычисления сопротивления R (Ом) участка электрической цепи по известным значениям напряжения U(B) и силы тока I (A) на

этом участке.

116. Разработайте алгоритм для вычисления сопротивления R (Ом) электрической цепи, состоящей из двух последовательно соединенных резисторов сопротивлениями R1 (Ом) и R2 (кОм).
117. Разработайте алгоритм пересчета величины временного интервала, заданного в минутах, в величину, выраженную в часах и минутах.
118. Разработайте алгоритм расчета величины ежемесячной кредитной выплаты, если погашение происходит ежемесячно равными долями. Процентная ставка, срок и сумма кредита определяются пользователем.
119. Разработайте алгоритм, который преобразует введенное с клавиатуры дробное число в денежный формат.
120. Смешали V1 литров воды температуры T1 градусов Цельсия с V2 литрами температуры T2 градусов Цельсия. Вычислить объем V и температуру T образовавшейся смеси.

Управляющая структура «Ветвление»

Уровень А

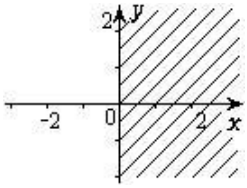
1. Дано целое число A. Если значение A > 0, то необходимо увеличить его на единицу. Разработайте схему алгоритма для решения этой задачи.
2. Дано целое число A. Если значение A < 0, то необходимо удвоить его. Разработайте схему алгоритма для решения этой задачи.
3. Дано целое число A. Если значение A = 0, то необходимо увеличить его на 3. Разработайте схему алгоритма для решения этой задачи.
4. Дано целое число A. Если значение A <> 0, то необходимо увеличить его в 4 раза. Разработайте схему алгоритма для решения этой задачи.
5. Дано целое число A. Если значение A > 0, то необходимо увеличить его в 5 раз. Разработайте схему алгоритма для решения этой задачи.
6. Дано целое число A. Если значение A < 0, то необходимо уменьшить его на 10. Разработайте схему алгоритма для решения этой задачи.
7. Дано целое число A. Если значение A >= 10, то необходимо уменьшить его на 3. Разработайте схему алгоритма для решения этой задачи.
8. Дано целое число A. Если значение A <> -20, то необходимо увеличить его в 15 раз. Разработайте схему алгоритма для решения этой задачи.
9. Даны целые числа A, B. Если значение B >= 10, то необходимо присвоить величине A значение суммы введенных чисел. Разработайте схему алгоритма для решения этой задачи.
10. Даны действительные числа A, B. Если значение A <= -10.5, то необходимо присвоить величине B значение суммы введенных чисел. Разработайте схему алгоритма для решения этой задачи.
11. Даны целые числа S, P. Если значение S < 0, то необходимо присвоить ей значение суммы введенных чисел. Разработайте схему алгоритма для решения этой задачи.

12. Даны действительные числа S, P. Если значение P <= 0, то необходимо присвоить ей значение произведения введенных чисел. Разработайте схему алгоритма для решения этой задачи.
13. Дано целое число A. Если значение A > 0, то необходимо увеличить его на единицу, иначе присвоить A значение равное 1. Разработайте схему алгоритма для решения этой задачи.
14. Дано целое число A. Если значение A < 0, то необходимо удвоить его, иначе присвоить A значение равное -1. Разработайте схему алгоритма для решения этой задачи.
15. Даны целые числа A, B. Если значение B >= 10, то необходимо присвоить величине A значение суммы введенных чисел, иначе – разности. Разработайте схему алгоритма для решения этой задачи.
16. Даны действительные числа A, B. Если значение A <= -10.5, то необходимо присвоить величине B значение суммы введенных чисел, иначе – разности. Разработайте схему алгоритма для решения этой задачи.
17. Даны целые числа S, P. Если значение величины S < 0, то необходимо присвоить ей значение суммы введенных чисел, иначе – величине P присвоить значение произведение введенных величин. Разработайте схему алгоритма для решения этой задачи.
18. Даны действительные числа S, P. Если значение величины S = 0, то необходимо присвоить ей значение суммы введенных чисел, иначе – величине P присвоить произведение. Разработайте схему алгоритма для решения этой задачи.
19. Дано целое число A. Если значение A = 0, то необходимо увеличить его на 3, иначе присвоить A значение равное -3. Разработайте схему алгоритма для решения этой задачи.
20. Дано целое число A. Если значение A <> 0, то необходимо увеличить его в 4 раза, иначе присвоить A значение равное 4. Разработайте схему алгоритма для решения этой задачи.
21. Дано целое число A. Если значение A > 100, то необходимо увеличить его на единицу, иначе уменьшить на 10. Разработайте алгоритм для решения этой задачи.
22. Дано целое число A. Если значение A <= 0, то необходимо удвоить его, иначе присвоить A значение равное 1. Разработайте схему алгоритма для решения этой задачи.
23. Дано целое число A. Если значение A = 0, то необходимо увеличить его на 3, иначе присвоить A значение равное 0. Разработайте схему алгоритма для решения этой задачи.
24. Дано целое число A. Если значение A <> 0, то необходимо увеличить его в 4 раза, иначе присвоить A значение равное -8. Разработайте схему алгоритма для решения этой задачи.

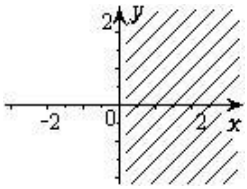
25. Дано целое число A. Если значение A < 0, то необходимо утроить его, иначе присвоить A значение равное -10. Разработайте алгоритм для решения этой задачи.
26. Дано действительное число A. Если значение A<100, то необходимо уменьшить его в два раза, иначе увеличить в два раза. Разработайте схему алгоритма для решения этой задачи.
27. Разработайте алгоритм вычисления частного от деления двух чисел. Необходимо проверять корректность введенных данных и, если они некорректные (делитель равен нулю), вывести сообщение об ошибке.
28. Разработайте алгоритм вычисления площади кольца, проверьте корректность исходных данных.
29. Разработайте алгоритм вычисления площади прямоугольника, проверьте корректность исходных данных.
30. Разработайте алгоритм вычисления длины окружности, проверьте корректность исходных данных.
31. Разработайте алгоритм вычисления стоимости покупки с учетом скидки. Скидка 10% предоставляется, если сумма покупки больше 1000 руб.
32. Разработайте алгоритм для вычисления повышающего коэффициента к стипендии. Если процент успеваемости более 75%, повышающий коэффициент – 1,3, иначе – 1.
33. Разработайте алгоритм для анализа успеваемости. Если процент успеваемости более 60%, студент признан успевающим, иначе – неуспевающий.

На координатной плоскости определена заштрихованная область. Точка A задана координатами X, Y (вводит пользователь). Разработайте схему алгоритма, который устанавливает значение флага F=1, если точка принадлежит заштрихованной области и устанавливает значение флага F=0 в противном случае, а затем выводит значение F.

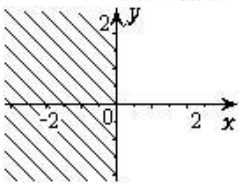
34.



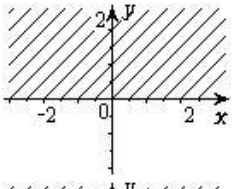
35.



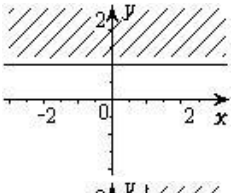
36.



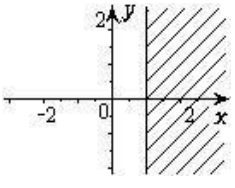
37.

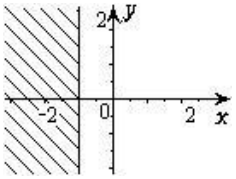


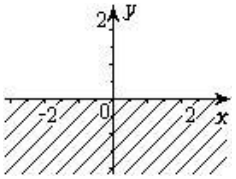
38.

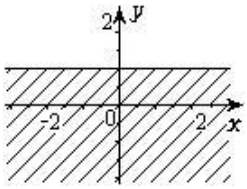


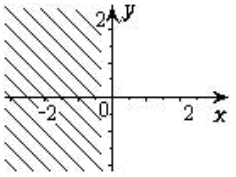
39.

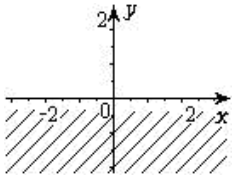


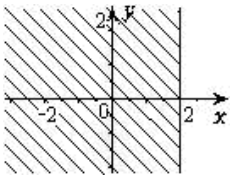
40. 

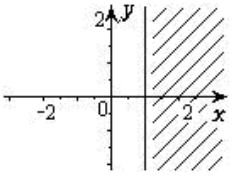
41. 

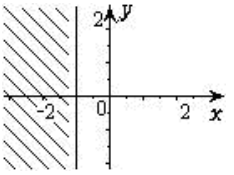
42. 

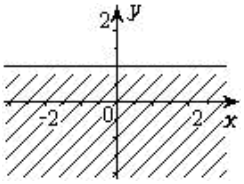
43. 

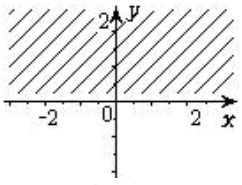
44. 

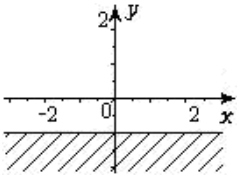
45. 

46. 

47. 

48. 

49. 

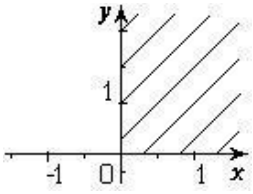
50. 

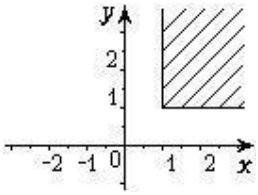
Уровень В

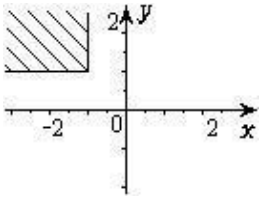
51. Разработайте алгоритм нахождения корней квадратного уравнения с коэффициентами a, b, c (a, b, c – не нулевые коэффициенты).
52. Разработайте схему алгоритма для подсчета количества n положительных чисел среди введенных целых чисел a, b, c.
53. Разработайте схему алгоритма для подсчета количества n отрицательных чисел среди введенных целых чисел a, b, c.
54. Разработайте схему алгоритма для подсчета количества n нулей среди введенных целых чисел a, b, c.
55. Разработайте алгоритм, который проверяет, не приводит ли суммирование двух целых чисел A и B к переполнению разрядной сетки, т. е. значение суммы S не должно быть более 32767. Если переполнение отсутствует, установите значение флага F=0, иначе установите значение флага F=1. Вывести значение F.
56. Разработайте алгоритм, который проверяет, попадают ли введенные целые числа A и B в интервал (-∞; -32) U (164; ∞). Если значение суммы принадлежит отрезку, установите значение флага F=1, иначе установите значение флага F=0. Вывести значение F.

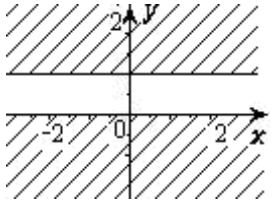
57. Даны два целых числа A и B. Если значение разности введенных чисел отрицательное, то установите значение флага F=0, иначе установите значение флага F=1. Вывести значение F.
58. Даны два действительных числа A и B. Если частное введенных чисел превосходит 100, то установите значение флага F=1, иначе установите значение флага F=0. Вывести значение F.
59. Даны два целых числа A и B. Если оба введенных значения больше нуля, то установите значение флага F=1, иначе установите значение флага F=0. Вывести значение F.
60. Даны два действительных числа A и B. Если оба введенных значения отрицательные, то установите значение флага F=0, иначе установите значение флага F=1. Вывести значение F.
61. Пользователь ввел три целых числа a, b, c. Если все введенные значения отрицательные, то установите значение флага F=0, иначе установите значение флага F=1. Вывести значение F.
62. Пользователь ввел три действительных числа a, b, c. Если все введенные значения больше 100, то установите значение флага F=1, иначе установите значение флага F=0. Вывести значение F.
63. Пользователь ввел три целых числа. Разработайте алгоритм, который проверяет, сколько из введенных значений не превышает 10.
64. Пользователь ввел четыре целых числа. Разработайте алгоритм, который проверяет, сколько из введенных значений не меньше -21.
65. Разработайте алгоритм для вычисления стоимости покупки с учетом скидки. Скидка в 3% предоставляется в том случае, если сумма покупки больше 500 руб., в 5% — если сумма больше 1000 руб.
66. Разработайте алгоритм для сравнения двух действительных чисел, введенных с клавиатуры, вывести большее число или, если числа равны, вывести любое из них.
67. Разработайте алгоритм для сравнения двух целых чисел, введенных с клавиатуры, вывести меньшее число или, если числа равны, вывести любое из них.
68. Разработайте алгоритм для вычисления оптимального веса пользователя, сравнения его с реальным и вывода рекомендации о необходимости поправиться или похудеть на недостающее (лишнее) количество килограммов. Оптимальный вес вычисляется по формуле: рост (в сантиметрах)-100.
69. Разработайте схему алгоритма для нахождения min(a, b, c) целых чисел a, b, c.
70. Разработайте схему алгоритма для нахождения max(a, b, c) целых чисел a, b, c.
71. Разработайте схему алгоритма, который проверяет, не превосходит ли сумма двух действительных чисел A и B числа 100. Если не превосходит, установите значение флага F=0, иначе установите значение флага F=1. Вывести значение F.
72. Разработайте алгоритм для подсчета количества n нулей среди целых чисел a, b, c.

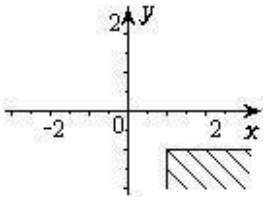
На координатной плоскости определена заштрихованная область. Точка A задана координатами X,Y. Разработайте схему алгоритма, который устанавливает значение флага F=1, если точка принадлежит заштрихованной области и устанавливает значение флага F=0 в противном случае, а затем выводит значение F.

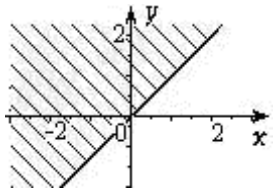
73. 

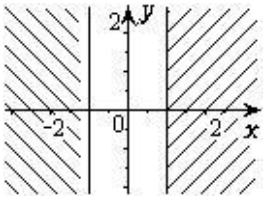
74. 

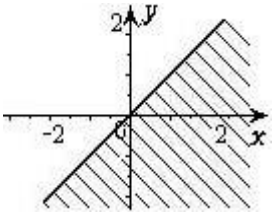
75. 

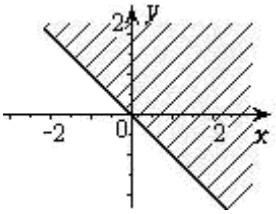
76. 

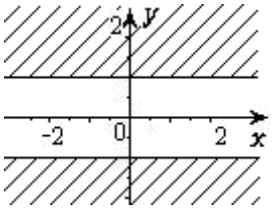
77. 

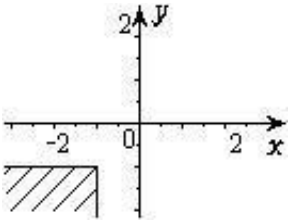
78. 

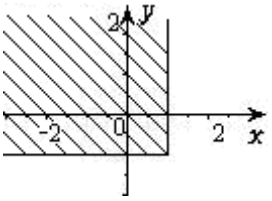
79. 

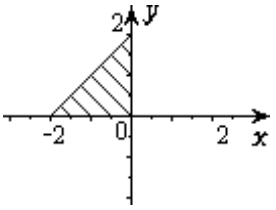
80. 

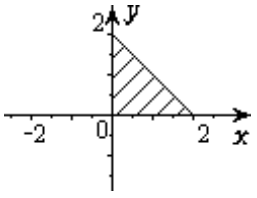
81. 

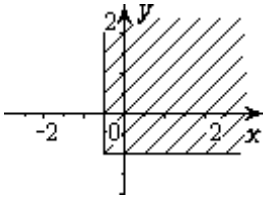
82. 

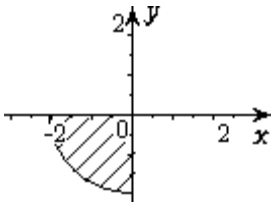
83. 

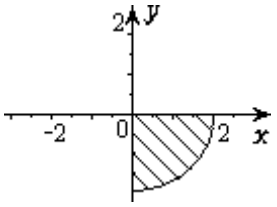
84. 

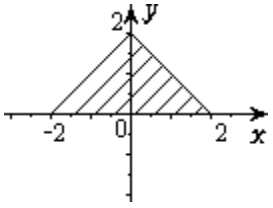
85. 

86. 

87. 

88. 

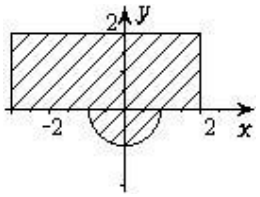
89. 

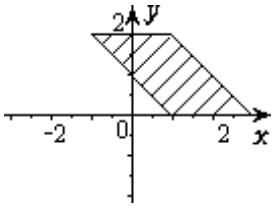
90. 

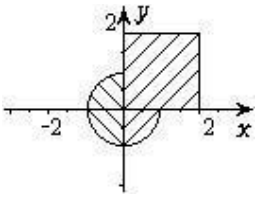
Уровень С

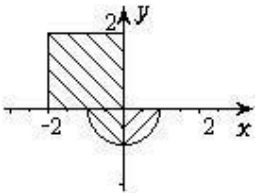
91. Треугольник задан длинами сторон A, B, C. Величина F (флаг) хранит признак существования треугольника. Значение F=1, если треугольник существует, в противном случае значение F=0. Разработайте схему алгоритма для решения этой задачи, используя сложные логические условия.
92. Треугольник задан длинами сторон A, B, C. Величина F (флаг) хранит признак. Значение F=1, если треугольник равносторонний, в противном случае значение F=0. Разработайте схему алгоритма для решения этой задачи, используя сложные логические условия.
93. Треугольник задан длинами сторон A, B, C. Величина F (флаг) хранит признак. Значение F=1, если треугольник египетский (равнобедренный), в противном случае значение F=0. Разработайте схему алгоритма для решения этой задачи, используя сложные логические условия.
94. Треугольник задан длинами сторон A, B, C. Если треугольник египетский (равнобедренный), то вычислите его площадь; в противном случае вычислите его полупериметр. Разработайте алгоритм для решения этой задачи, используя сложные логические условия.
95. Разработайте алгоритм для отыскания max(min(a,b), min(c,d)), не используя сложные логические условия и вложенные ветвления. Числа a, b, c, d – целые.
96. Разработайте алгоритм для отыскания min(max(a,b), max(c,d)), не используя сложные логические условия и вложенные ветвления. Числа a, b, c, d – целые.
97. Даны три целых числа A, B, C. P – количество одинаковых пар среди них. Разработайте схему алгоритма для подсчета P.

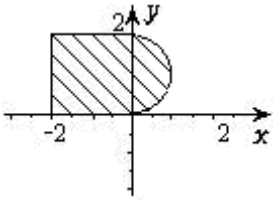
На координатной плоскости определена заштрихованная область. Точка A задана координатами X,Y. Разработайте схему алгоритма, который устанавливает значение флага F=1, если точка принадлежит заштрихованной области и устанавливает значение флага F=0 в противном случае, а затем выводит значение F.

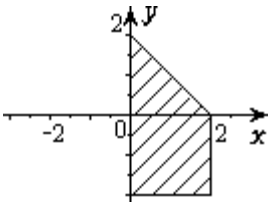
98. 

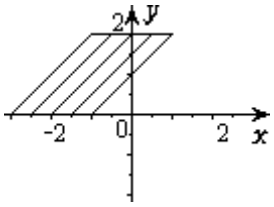
99. 

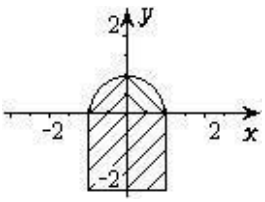
100. 

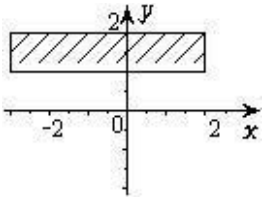
101. 

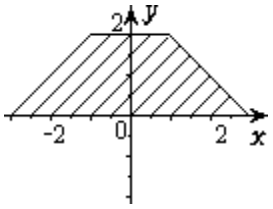
102. 

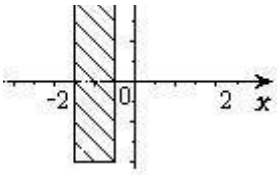
103. 

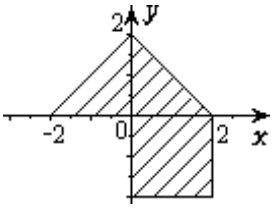
104. 

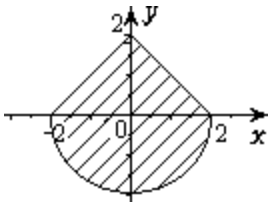
105. 

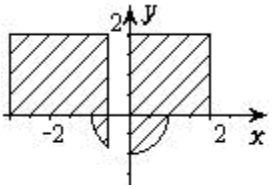
106. 

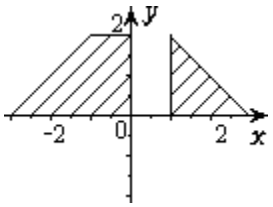
107. 

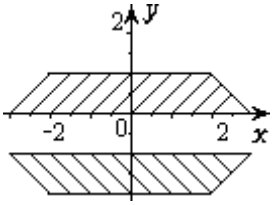
108. 

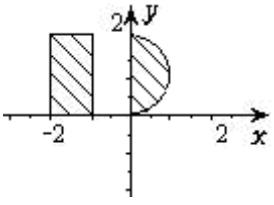
109. 

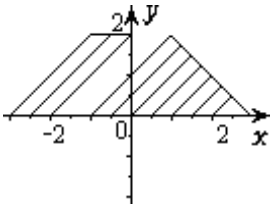
110. 

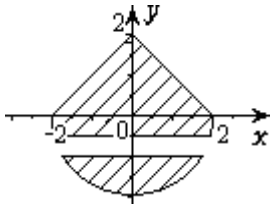
111. 

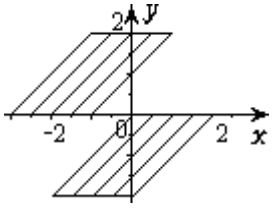
112. 

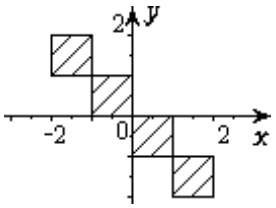
113. 

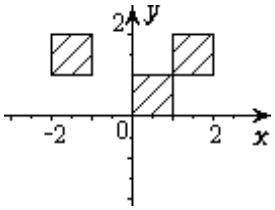
114. 

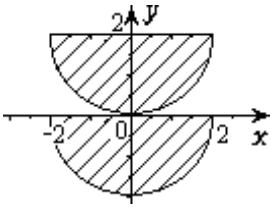
115. 

116. 

117. 

118. 

119. 

120. 

Управляющая структура «Цикл»

Уровень А

1. Разработайте алгоритм вычисления значения функции $y = -2{,}7x^3 + 0{,}23\ x^2 - 1{,}4$ на отрезке [a; b] (значение a, b вводит пользователь).
2. Разработайте схему алгоритма для вычисления произведения P первых n целых чисел. Предусмотрите ввод значения n с клавиатуры. Текущее значение целого числа имеет имя i (i = 1, 2, .., n).
3. Разработайте схему алгоритма для вычисления произведения P всех однозначных целых чисел. Текущее значение целого числа имеет имя i (i = 1, 2, .., n).
4. Разработайте схему алгоритма для вычисления произведения P всех однозначных четных целых чисел. Текущее значение целого числа имеет имя i (i = 1, 2, .., n).
5. Разработайте схему алгоритма для вычисления произведения P всех однозначных нечетных целых чисел. Текущее значение целого числа имеет имя i (i = 1, 2, .., n).
6. Разработайте схему алгоритма для вычисления суммы S всех целых четных двузначных чисел. Текущее значение целого числа имеет имя i.
7. Разработайте схему алгоритма для вычисления суммы S всех целых двузначных чисел, кратных десяти. Текущее значение целого числа имеет имя i.
8. Разработайте схему алгоритма для вычисления суммы S всех целых четных двузначных чисел. Текущее значение целого числа имеет имя i.
9. Разработайте схему алгоритма для вычисления суммы S всех целых двузначных чисел, кратных десяти. Текущее значение целого числа имеет имя i.
10. Разработайте схему алгоритма для вычисления суммы S первых n целых чисел. Предусмотрите ввод значения n с клавиатуры. Текущее значение целого числа имеет имя i (i = 1, 2, .., n).
11. Разработайте схему алгоритма для вычисления суммы S целых нечетных чисел на отрезке [a; b]. Текущее значение целого числа имеет имя i.

12. Разработайте схему алгоритма для вычисления суммы S целых четных чисел на отрезке [a; b]. Текущее значение целого числа имеет имя i.
13. Разработайте схему алгоритма для вычисления суммы S всех целых нечетных двузначных чисел. Текущее значение целого числа имеет имя i.
14. Разработайте схему алгоритма для вычисления суммы S квадратов однозначных целых чисел.
15. Разработайте схему алгоритма для вычисления суммы S первых n действительных чисел с шагом 0,5. Предусмотрите ввод значения n с клавиатуры.
16. Разработайте алгоритм вычисления значения функции $y = 1/x^2$ на отрезке [a; b] (значение a, b вводит пользователь).
17. Разработайте алгоритм для вычисления произведения P всех однозначных нечетных целых чисел. Текущее значение целого числа имеет имя i (i = 1, 2, .., n).
18. Разработайте алгоритм для вычисления произведения P всех однозначных целых чисел. Текущее значение целого числа имеет имя i (i = 1, 2, .., n).
19. Разработайте алгоритм для вычисления произведения P всех однозначных четных целых чисел. Текущее значение целого числа имеет имя i (i = 1, 2, .., n).
20. Разработайте алгоритм для вычисления произведения P первых n целых чисел. Предусмотрите ввод значения n с клавиатуры. Текущее значение целого числа имеет имя i (i = 1, 2, .., n).
21. Разработайте алгоритм для вычисления суммы S всех целых двузначных чисел на отрезке [a; b]. Текущее значение целого числа имеет имя i.
22. Разработайте алгоритм для вычисления суммы S всех целых трехзначных чисел на отрезке [a; b].
23. Разработайте алгоритм для вычисления суммы S всех целых четных трехзначных чисел на отрезке [a; b].
24. Разработайте алгоритм для вычисления суммы S степеней числа 2, до n-ой степени.
25. Разработайте схему алгоритма для вычисления суммы S всех целых нечетных двузначных чисел. Текущее значение целого числа имеет имя i.
26. Разработайте схему алгоритма для вычисления суммы S всех целых четных двузначных чисел на отрезке [a; b]. Текущее значение целого числа имеет имя i.
27. Разработайте схему алгоритма для вычисления суммы S целых чисел на отрезке [a; b]. Текущее значение целого числа имеет имя i.
28. Разработайте схему алгоритма для вычисления суммы S первых n целых чисел. Предусмотрите ввод значения n с клавиатуры. Текущее значение целого числа имеет имя i (i = 1, 2, .., n).

Уровень B

29. Разработайте алгоритм для вычисления произведения P цифр четырехразрядного целого числа M. Текущее значение цифры целого числа M имеет имя i (i = 1, 2, .., n).
30. Разработайте схему алгоритма для вычисления произведения $P = (-1)\cdot 2\cdot (-3)\cdot 4\cdot (-5)\cdot \ldots \cdot n$. Используйте два цикла: один для вычисления произведе-

ния нечетных чисел P1, второй – для вычисления произведения четных чисел P2.

31. Разработайте схему алгоритма для вычисления суммы S = 1 - 2 + 3 - 4 + ... + n. Используйте два цикла: один для вычисления суммы нечетных чисел S1, второй - для вычисления суммы четных чисел S2.
32. Разработайте схему алгоритма для вычисления и вывода значений функции $y = |x + 2|$. Диапазон изменения аргумента – от -4 до 4, шаг приращения аргумента – 0,5
33. Разработайте схему алгоритма для вычисления и вывода значений функции $y = |x- 2|+ |x + 1|$. Диапазон изменения аргумента – от -4 до 4, шаг приращения аргумента – 0,5.
34. Разработайте схему алгоритма для вычисления и вывода табл. умножения на 7.
35. Разработайте схему алгоритма для вычисления суммы S по правилу: $1+2\cdot x+3\cdot x^2+ \dots +n\cdot x^{(n-1)}$. Значения переменных x, n вводит пользователь.
36. Разработайте схему алгоритма для вычисления суммы S по правилу: $1-2\cdot(-x)-3\cdot(-x)^2- \dots -n\cdot(-x)^{(n-1)}$.
37. Разработайте схему алгоритма для вычисления суммы первых n положительных целых чисел.
38. Разработайте схему алгоритма для вычисления суммы степеней двойки от нулевой до n, введенного пользователем.
39. Разработайте схему алгоритма для вычисления факториала числа n, введенного с клавиатуры.
40. Разработайте схему алгоритма для заполнения таблицы значений функции $y = - 2,4x^2 + 5x - 3$ в диапазоне от -2 до 2 с шагом 0,5.
41. Разработайте схему алгоритма для нахождения суммы квадратов первых пяти целых положительных нечетных чисел.
42. Разработайте схему алгоритма для нахождения суммы цифр введенного целого трехзначного числа.
43. Разработайте схему алгоритма для преобразования введенного целого двузначного числа по правилу: обменять местами цифры десятков и единиц.
44. Разработайте схему алгоритма для преобразования введенного целого двузначного числа по правилу: вычислить разность десятков и единиц, полученный результат записать на место единиц, цифру десятков сохранить.
45. Разработайте схему алгоритма для преобразования введенного целого двузначного числа по правилу: удвоить цифры введенного числа, например, введено 24, получено 2244.
46. Разработайте схему алгоритма для вычисления произведения P цифр четырехразрядного целого числа M. Текущее значение цифры целого числа M имеет имя i (i = 1, 2, .., n).
47. Разработайте схему алгоритма для вычисления суммы S цифр четырехразрядного целого числа M. Текущее значение цифры целого числа M имеет имя i (i = 1, 2, .., n).
48. Разработайте схему алгоритма для отыскания наибольшей цифры Q в записи пятиразрядного целого числа M. Текущее значение цифры целого числа M имеет

имя i (i = 1, 2, .., n).

49. Разработайте схему алгоритма для отыскания наименьшей цифры W в записи пятиразрядного целого числа M. Текущее значение цифры целого числа M имеет имя i (i = 1, 2, .., n).

Уровень С

50. Разработайте алгоритм для вычисления суммы S n целых чисел. При вычислении суммы учесть требование: если число четное, то вычитать его из суммы, иначе – прибавлять. Текущее значение целого числа имеет имя i.
51. Разработайте алгоритм для вычисления суммы S n целых двузначных чисел. При вычислении суммы учесть требование: если число четное, то вычитать его из суммы, иначе – прибавлять. Текущее значение целого числа имеет имя i.
52. Разработайте алгоритм для вычисления суммы S n целых чисел. При вычислении суммы учесть требование: если число четное, то вычитать его из суммы, иначе – прибавлять. Текущее значение целого числа имеет имя i.
53. Разработайте алгоритм для вычисления суммы S целых чисел на отрезке [a; b]. При вычислении суммы учесть требование: если число четное, то вычитать его из суммы, иначе - прибавлять. Текущее значение целого числа имеет имя i.
54. Разработайте алгоритм для вычисления произведения P действительных чисел на отрезке [a; b]. При вычислении произведения учесть требование: если число кратно 10, то разделить на него, иначе – умножить. Текущее значение действительного числа имеет имя i.
55. Разработайте алгоритм для преобразования введенного целого четырехзначного числа по правилу: поменять цифры числа местами в порядке возрастания.
56. Разработайте алгоритм для преобразования введенного целого четырехзначного числа по правилу: поменять цифры числа местами в порядке убывания.
57. Разработайте алгоритм для преобразования введенного целого четырехзначного числа по правилу: получить из четырехзначного введенного числа трехзначное путем исключения из записи цифры 5.
58. Разработайте алгоритм для преобразования введенного целого четырехзначного числа по правилу: заменить число сотен разностью тысяч и единиц взятых по абсолютной величине.
59. Разработайте алгоритм для преобразования введенного целого четырехзначного числа по правилу: заменить число десятков разностью единиц и сотен взятых по абсолютной величине.
60. Разработайте алгоритм для преобразования введенного целого четырехзначного числа по правилу: заменить нечетные цифры на ближайшие четные, найти сумму цифр полученного числа. Вывести полученное число и сумму его цифр.
61. Разработайте алгоритм для преобразования введенного целого четырехзначного числа по правилу: заменить четные цифры на ближайшие нечетные, найти сумму цифр полученного числа. Вывести полученное число и сумму его цифр.

Варианты индивидуальных заданий

Приведенные ниже наборы заданий рекомендуется использовать совместно с интегрированной системой «Конструктор схем алгоритмов» («Schemes»), разработанной в ВЦ ЦНТО ПГНИУ.

Набор 1

Уровни	Структура следование			Структура ветвление			Циклические структуры		
	A	B	C	A	B	C	A	B	C
Вариант 1	33	86	104	22	87	95	17	41	50
Вариант 2	6	77	116	14	68	101	18	35	51
Вариант 3	43	85	105	50	56	100	12	45	52
Вариант 4	10	81	112	28	86	98	7	46	51
Вариант 5	1	66	115	42	78	93	22	32	53
Вариант 6	2	94	111	24	73	97	16	33	54
Вариант 7	52	61	101	33	51	102	25	37	55
Вариант 8	26	64	102	18	52	105	13	38	56
Вариант 9	46	65	119	10	75	107	15	40	57
Вариант 10	8	76	108	45	81	110	21	47	58
Вариант 11	58	80	106	39	72	106	11	31	59
Вариант 12	18	97	109	32	77	99	8	44	60
Вариант 13	36	70	114	48	76	104	20	49	61
Вариант 14	55	78	118	31	89	103	6	30	50
Вариант 15	42	73	107	26	90	109	14	34	51
Вариант 16	51	99	110	29	61	96	9	39	52
Вариант 17	4	75	117	40	66	91	10	29	53
Вариант 18	16	79	103	6	83	108	4	36	55
Вариант 19	20	98	113	30	54	92	28	43	59
Вариант 20	30	62	120	20	58	94	5	42	61

Набор 2

Уровни	Структура следование			Структура ветвление			Циклические структуры		
	A	B	C	A	B	C	A	B	C
Вариант 1	20	87	108	21	62	91	3	42	51
Вариант 2	10	62	111	34	72	103	17	39	50
Вариант 3	3	69	116	38	55	120	12	38	58
Вариант 4	4	66	119	3	73	104	13	46	56
Вариант 5	46	72	104	2	88	97	15	29	60
Вариант 6	39	70	105	39	78	93	1	30	53
Вариант 7	51	61	109	33	61	96	6	49	59

Уровни	Структура следование			Структура ветвление			Циклические структуры		
	A	B	C	A	B	C	A	B	C
Вариант 8	44	98	115	11	58	102	16	41	52
Вариант 9	18	73	101	50	68	109	24	43	54
Вариант 10	21	63	117	49	51	107	9	32	51
Вариант 11	58	65	112	6	56	110	25	31	50
Вариант 12	56	99	120	14	74	119	5	45	55
Вариант 13	52	67	110	30	60	98	11	40	60
Вариант 14	50	78	107	44	54	106	2	37	59
Вариант 15	45	84	106	1	65	108	23	47	54
Вариант 16	47	85	118	16	59	92	10	35	57
Вариант 17	34	71	114	43	53	101	14	36	55
Вариант 18	54	64	103	5	87	95	20	33	56
Вариант 19	35	91	102	42	84	105	8	44	52
Вариант 20	38	68	113	27	89	94	7	34	58

Набор 3

Уровни	Структура следование			Структура ветвление			Циклические структуры		
	A	B	C	A	B	C	A	B	C
Вариант 1	15	91	119	50	88	93	21	45	61
Вариант 2	11	85	113	23	51	116	10	41	55
Вариант 3	60	70	108	25	65	92	20	44	58
Вариант 4	14	65	112	40	63	99	2	48	55
Вариант 5	56	88	118	37	78	107	4	33	60
Вариант 6	23	80	103	43	86	110	27	29	50
Вариант 7	27	72	102	45	60	97	9	47	59
Вариант 8	20	84	109	41	89	106	7	30	57
Вариант 9	29	86	120	48	56	104	23	39	54
Вариант 10	31	81	117	39	73	108	25	46	59
Вариант 11	37	87	105	24	67	115	24	36	53
Вариант 12	6	79	101	32	74	101	8	43	50
Вариант 13	13	92	107	16	52	96	16	40	54
Вариант 14	34	96	106	34	87	102	12	37	57
Вариант 15	40	89	114	9	54	103	19	42	58
Вариант 16	21	75	111	8	77	105	22	35	60
Вариант 17	18	74	116	42	69	94	3	34	53
Вариант 18	38	99	104	47	64	100	28	49	61
Вариант 19	42	71	110	14	66	109	1	38	52
Вариант 20	7	100	115	46	90	98	26	31	51

Набор 4

Уровни	Структура следование			Структура ветвление			Циклические структуры		
	A	B	C	A	B	C	A	B	C
Вариант 1	18	100	117	19	85	104	28	42	57
Вариант 2	11	62	112	14	69	102	2	31	60
Вариант 3	13	73	110	4	75	108	22	30	50
Вариант 4	43	92	113	11	54	91	13	33	55
Вариант 5	15	61	118	30	51	100	6	37	52
Вариант 6	34	86	106	40	63	99	10	48	59
Вариант 7	35	97	107	9	53	106	21	29	51
Вариант 8	29	64	111	25	55	98	24	49	59
Вариант 9	38	91	119	8	88	107	16	38	60
Вариант 10	17	90	103	2	82	92	20	40	52
Вариант 11	44	74	101	24	87	101	7	39	58
Вариант 12	40	96	116	1	67	96	17	44	61
Вариант 13	41	93	115	13	71	113	9	41	61
Вариант 14	14	84	105	10	72	94	18	34	53
Вариант 15	1	63	104	29	79	105	4	43	55
Вариант 16	33	89	102	6	68	93	26	36	56
Вариант 17	31	66	114	28	60	97	11	35	56
Вариант 18	48	65	108	12	65	95	23	32	53
Вариант 19	3	99	120	34	70	110	19	45	58
Вариант 20	56	88	109	20	80	109	5	47	50

Набор 5

Уровни	Структура следование			Структура ветвление			Циклические структуры		
	A	B	C	A	B	C	A	B	C
Вариант 1	41	89	115	20	51	93	23	39	58
Вариант 2	51	73	104	37	89	102	21	44	54
Вариант 3	38	71	101	49	80	100	28	32	59
Вариант 4	14	92	102	1	56	109	22	41	51
Вариант 5	13	91	112	9	88	104	19	38	50
Вариант 6	10	90	114	25	70	106	18	31	53
Вариант 7	5	67	111	42	79	110	6	42	59
Вариант 8	6	85	113	47	72	103	11	48	58
Вариант 9	47	65	119	2	54	95	25	49	52
Вариант 10	8	78	116	34	90	94	20	40	55
Вариант 11	46	79	117	44	86	91	24	46	52
Вариант 12	16	69	108	4	73	101	2	34	56

Уровни	Структура следование			Структура ветвление			Циклические структуры		
	A	B	C	A	B	C	A	B	C
Вариант 13	49	87	110	36	71	108	27	45	53
Вариант 14	12	99	107	24	69	92	1	33	51
Вариант 15	28	66	109	30	67	105	26	43	60
Вариант 16	25	98	120	8	53	99	14	47	57
Вариант 17	3	88	118	18	81	97	3	29	60
Вариант 18	50	75	105	3	74	98	17	35	56
Вариант 19	20	68	103	29	65	96	4	30	50
Вариант 20	7	86	106	14	52	107	15	37	54

Набор 6

Уровни	Структура следование			Структура ветвление			Циклические структуры		
	A	B	C	A	B	C	A	B	C
Вариант 1	25	71	103	3	68	107	10	46	56
Вариант 2	57	68	104	27	87	108	27	45	60
Вариант 3	12	85	108	21	63	109	16	39	52
Вариант 4	37	81	107	4	65	105	20	29	52
Вариант 5	20	93	117	2	51	98	28	48	50
Вариант 6	23	91	112	26	77	91	25	47	59
Вариант 7	22	87	120	11	67	92	6	35	56
Вариант 8	1	61	106	15	62	104	12	42	58
Вариант 9	4	63	115	16	53	97	18	31	50
Вариант 10	15	89	109	37	89	93	22	49	59
Вариант 11	41	73	101	31	64	99	24	36	58
Вариант 12	28	78	102	17	78	100	23	41	60
Вариант 13	14	98	118	39	72	94	9	37	51
Вариант 14	19	100	114	23	73	101	7	30	51
Вариант 15	17	99	110	5	83	102	3	40	53
Вариант 16	8	84	111	19	85	106	4	44	55
Вариант 17	19	80	116	7	80	96	17	33	54
Вариант 18	16	82	119	41	56	103	13	34	53
Вариант 19	7	70	105	44	76	95	21	43	54
Вариант 20	18	95	113	1	75	110	15	32	55

СТРУКТУРЫ ДАННЫХ
ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Принято различать целые, вещественные и логические данные, множества, последовательности, векторы, матрицы (таблицы) и т.д. В обработке данных с использованием технических средств тип и структура данных играют большую роль.

Любая константа, переменная, выражение или функция относятся к некоторому *типу.*

Тип данных определяет диапазон допустимых значений и операций, которые могут быть применены к этим значениям. Тип величины может быть установлен при ее описании.

Структура данных задает формат представления объектов в памяти компьютера, так как любые данные будут представлены в виде последовательности двоичных цифр (нулей и единиц).

Классификация типов данных

Любые данные могут быть отнесены к одному из двух типов: простому (основному), форма представления которого определяется архитектурой ЭВМ, или сложному, конструируемому пользователем для решения конкретных задач. Данные простого типа – это символы, числа и т.п. элементы, дальнейшее дробление которых не имеет смысла. Из таких элементарных данных формируются структуры (сложные типы) данных.

Принято различать следующие типы данных:

1. Простые:
   - 1.1. Числовые типы:
     - 1.1.1. Целочисленные.
     - 1.1.2. Вещественные.
   - 1.2. Символьный тип:
     - 1.2.1. Логический тип.
     - 1.2.2. Перечислимый тип.
     - 1.2.3. Множество.
     - 1.2.4. Указатель.
2. Комбинированные:
   - 2.1. Массив.
   - 2.2. Строковый тип.
   - 2.3. Запись.
   - 2.4. Последовательность.

Простые типы

Числовые типы. Значениями переменных таких типов являются числа. К ним могут применяться обычные арифметические операции, операции сравнения (в результате получается логическое значение). Принципиально различны в компьютерном представлении целые и вещественные типы.

Целочисленные типы данных делятся, в свою очередь, на *знаковые* и *беззнаковые.* Целочисленные со знаком могут принимать как положительные, так и отрицательные значения, а беззнаковые – только неотрицательные значения. Диапазон значений при этом определяется количеством разрядов, отводимых на представление конкретного типа в памяти компьютера.

Вещественные типы бывают: с фиксированной точкой, т. е. хранятся знак и цифры целой и дробной частей (в настоящее время в языках программирования реализуются редко), и с плавающей точкой, т. е. число приводится к виду m x 2^e, где m – мантисса, *а* e – порядок числа, причем 1/2 <= <= m <= 1, e – целое число. В данном случае хранятся знак, число e и двоичные цифры дробной части числа m, которые умещаются в отведенную для этого память. Говорят, что вещественные числа представимы с некоторой точностью.

Символьный тип. Элемент этого типа хранит один символ. При этом могут использоваться различные кодировки, которые определяют, какому коду (двоичному числу) какой символ (знак) соответствует. К значениям этого типа могут применяться операции сравнения (в результате получается логическое значение). Символы считаются упорядоченными согласно своим кодам (номерам в кодовой таблице).

Логический тип. Данные этого типа имеют два значения: истина (true) и ложь (false). К ним могут применяться логические операции. Используется в условных выражениях, операторах ветвления и циклах. В некоторых языках, например C, является подтипом числового типа, при этом ложь = 0, истина = 1 (или истинным считается любое значение, отличное от нуля).

Перечислимый тип. Отражает самый прямолинейный способ описания простого типа — перечисление всех значений, относящихся к этому типу. Каждая константа такого типа получает свой порядковый номер, что позволяет реализовать ряд простых операций над этим типом, таких как *получить следующее по порядку значение данного типа.*

Множество как тип данных в основном совпадает с обычным математическим понятием множества. Допустимы стандартные операции с множествами и проверка на принадлежность элемента множеству. В некоторых языках рассматривается как составной тип (массив логических значений, i-й элемент которого указывает, находится ли i в множестве), однако эффективней реализовывать множество как машинное слово (или несколько слов), каждый бит которого характеризует наличие соответствующего элемента в множестве.

Указатель (тип данных). Если описанные выше типы данных представляли какие-либо объекты реального мира, то указатели представляют объекты компьютерного мира, т. е. являются исключительно компьютерными терминами. Переменная-указатель хранит адрес в памяти компьютера, указывающий на какую-либо информацию, как правило, – на другую переменную.

Комбинированные типы

Комбинированные типы формируются на основе комбинаций простых типов.

Массив является индексированным набором элементов одного типа, простого или составного. Одномерный массив предназначен для компьютерной реализации такой структуры, как вектор, двухмерный массив – таблицы.

Строковый тип. Хранит строку символов. Вообще говоря, может рассматриваться как массив символов, но иногда рассматривается в качестве простого типа. Часто используется для хранения фамилий людей, названий предметов и т.п. К элементам этого типа может применяться операция конкатенации (сложения) строк. Обычно реализованы также операции сравнения над строками, в том числе операции "<" и ">", которые интерпретируются как сравнение строк согласно алфавитному порядку (алфавитом здесь является набор символов соответствующей кодовой таблицы). Во многих языках реализованы и специальные операции над строками: поиск заданного символа (подстроки), вставка символа, удаление символа, замена символа.

Запись. Наиболее общий метод получения составных типов из простых заключается в объединении элементов произвольных типов. Причем сами эти элементы могут быть, в свою очередь, составными. Так, человек описывается с помощью нескольких различных характеристик, таких как имя, фамилия, дата рождения, пол и т.д. Записью называется набор различных элементов (полей записи), хранимый как единое целое. При этом возможен доступ к отдельным полям записи. К полю записи применимы те же операции, что и к базовому типу, к которому это поле относится (тип каждого поля указывается при описании записи).

Последовательность. Данный тип можно рассматривать как массив бесконечного размера (память для него может выделяться в процессе выполнения программы по мере роста последовательности). Зачастую такой тип данных обладает лишь *последовательным доступом* к элементам. Под этим подразумевается, что последовательность просматривается от одного элемента строго к следующему, формируется же она путем добавления элементов в ее конец. В языке Pascal подобному типу соответствуют *файловые типы* данных.

Преимущества использования типов данных

Типы данных защищают программы, по крайней мере, от следующих ошибок:

1. Некорректное присваивание. Пусть переменная объявлена как имеющая числовой тип. Тогда попытка присвоить ей символьное или какое-либо другое значение приведет к ошибке еще на этапе компиляции. Такого рода ошибки трудно отследить обычными средствами.

2. Некорректная операция. Типизация позволяет избежать попыток применения выражений вида "Hello world" + 1. Поскольку все переменные в памяти хранятся как наборы битов, то при отсутствии типов подобная операция была выполнима (и могла дать результат вроде "Hello worlde"!). С использованием типов такие ошибки отсекаются опять же на этапе компиляции.

3. Некорректная передача параметров в процедуры и функции. Если функция "синус" ожидает, что ей будет передан числовой аргумент, то передача ей в качестве параметра строки "Hello world" может иметь непредсказуемые последствия. При помощи контроля типов такие ошибки также отсекаются на этапе компиляции или при-

водят к ошибкам выполнения программы, если значения параметра вводятся с клавиатуры или файла.

Структуры данных

Необходимым условием построения алгоритма является *формализация данных,* т.е. приведение информации к некоторой уже исследованной *информационной модели*. Когда такая модель найдена, говорят, что определена *абстрактная структура данных.*

Абстрактная структура данных описывает признаки и свойства объекта, *взаимосвязь* между элементами объекта, а также возможные *операции* над данным объектом или классом объектов.

Одной из задач информатики является нахождение форм представления информации, удобных для компьютерной обработки. Информатика как точная наука работает с формальными (описанными математически строго) объектами. Такими объектами – базовыми **абстрактными структурами данных,** используемыми в информатике, являются:

- целые числа;
- вещественные числа;
- символы;
- логические значения.

Для компьютерной обработки этих объектов в языках программирования существуют соответствующие *тины данных*. Базовые объекты можно объединять в более сложные структуры, добавляя операции уже над структурой в целом и правила доступа к отдельным элементам этой абстрактной структуры данных.

К таким абстрактным структурам данных относятся:

- векторы (конечные массивы);
- таблицы (матрицы), а в общем случае – многомерные массивы;
- графы;
- динамические структуры:
  - последовательности символов, чисел;
  - строки;
  - списки;
  - очереди;
  - стеки;
  - деревья;
  - графы.

Удачный выбор структуры данных часто является залогом создания эффективного алгоритма и программы, его реализующей: используя аналогию структур данных и реальных объектов, можно находить эффективные решения задач.

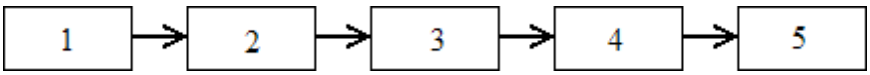
Заметим, что перечисленные структуры существуют независимо от их реализации при программировании. С этими структурами данных работали и в XVIII, и в XIX веках, когда еще не придумали вычислительную машину. Мы можем разрабаты-

вать алгоритм в терминах абстрактной структуры данных, но для реализации алгоритма в конкретном языке программирования необходимо найти способ ее представления в терминах *типов данных* и *операторов,* поддерживаемых данным языком программирования. Для компьютерного представления абстрактных структур используются **структуры данных,** которые представляют собой набор переменных, возможно различных типов данных, объединенных определенным образом. Для конструирования таких структур, как вектор, таблица, строка, последовательность, в большинстве языков программирования присутствуют стандартные *типы данных:* одномерный массив, двухмерный массив, строка, файл (реже список) соответственно. Организацию остальных структур данных, в первую очередь **динамических структур,** размер которых меняется во время выполнения программы, программисту приходится осуществлять самостоятельно, используя базовые типы данных. Рассмотрим такие структуры подробнее.

Списки

Линейный список – последовательность линейно связанных элементов, для которых разрешены операции добавления элементов в произвольное место списка и удаление любого элемента. Линейный список однозначно задается указателем на начало списка. Типовыми операциями над списками являются: обход списка, поиск заданного элемента, вставка элемента сразу после или перед определенным элементом, удаление заданного элемента, объединение двух списков в один, разбиение одного списка на два и более списков и т.п.

В линейном списке для каждого элемента, кроме *первого,* есть *предыдущий* элемент; для каждого элемента, кроме *последнего,* есть *следующий* элемент. Таким образом, все элементы списка упорядочены. Однако обработка линейного односвязного списка не всегда удобна, т.к. отсутствует возможность движения в противоположную сторону — от конца списка к началу. В линейном списке можно обойти все элементы, только двигаясь последовательно от текущего элемента к следующему, начиная с первого, прямой доступ к i-му по счету элементу невозможен.



Линейные списки, в которых операции вставки, удаления и доступа к значениям элементов выполняются только с крайними элементами (первым или последним), получили специальные названия.

Стек – частный случай линейного односвязного списка, для которого определены две операции: добавление элемента в вершину стека (перед первым элементом) и удаление элемента из вершины стека (удаление первого элемента).

Пример. Рассмотрим задачу определения сбалансированности скобок различных видов в арифметическом выражении. Например, требуется проанализировать, сбалансированы ли скобки в выражении, содержащем круглые и квадратные скобки:

[(a + b) -(a- c)] + ([a- b)])? Для решения этой задачи будем использовать динамическую структуру данных *стек.* Приведем алгоритм решения этой задачи по шагам.

Будем использовать следующие обозначения:

i – номер анализируемого символа;

n – количество символов в выражении.

1. *i*= 0.

2. *i*=*i*+ l.

3. Если i <= *n,* то переход на п. (4), иначе если стек пуст, то выдаем сообщение "скобки сбалансированы", в противном случае выдаем сообщение *"скобки не сбалансированы".* Конец алгоритма.

4. Если i-й символ отличен от символов скобок, то переход на п. (2).

5. Если i-й символ равен "(" или " [", то помещаем его в стек, переход на п. (2).

6. Если i-й символ равен ")", то проверяем вершину стека: если в вершине стека находится "(", то извлекаем ее из стека; переход на п. (2), иначе выдаем сообщение *"скобки не сбалансированы".* Конец алгоритма.

7. Если i-й символ равен "] ", то проверяем вершину стека: если в вершине стека находится " [", то извлекаем ее из стека; переход на п. (2), иначе выдаем сообщение *"скобки не сбалансированы".* Конец алгоритма.

Очередь – частный случай линейного односвязного списка, для которого разрешены только две операции: добавление элемента в конец (хвост) очереди и удаление элемента из начала (головы) очереди.

Понятие очереди действительно очень близко к бытовому термину "очередь". Очередь покупателей в магазине хорошо описывается в терминах этой структуры данных.

Дерево – это совокупность элементов, называемых *узлами,* в которой выделен один элемент *(корень),* а остальные элементы разбиты на непересекающиеся множества (поддеревья), каждое из которых является деревом, при этом корень каждого поддерева является *потомком* корня дерева, т.е. все элементы связаны между собой отношением (предок–потомок). В результате образуется иерархическая структура узлов. Узлы, которые не имеют ни одного потомка, называются *листьями.* Над деревом определены следующие операции: добавление элемента в дерево, удаление элемента из дерева, обход дерева, поиск элемента в дереве.

Пример. Дерево является наиболее удобной структурой данных для представления генеалогического дерева, с помощью которого можно решать задачи определения степени родства между двумя людьми.

Используются деревья и для определения выигрышной стратегии в играх, и для построения различных информационных моделей.

Особенно важную роль в информатике играют так называемые *бинарные деревья.*

Двоичное (бинарное) дерево – частный случай дерева, в котором каждый узел может иметь не более двух потомков, являющихся корнями левого и правого поддерева.

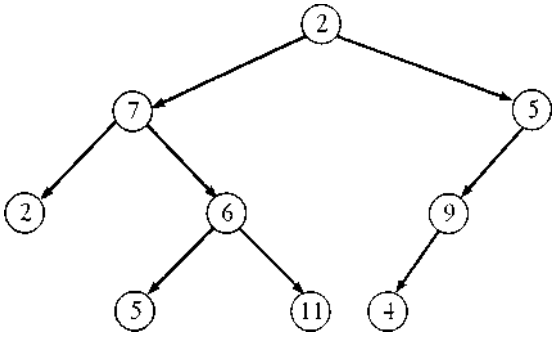


Рис. 4. Пример двоичного дерева

Если дополнительно для узлов дерева выполняется условие, что все значения элементов левого поддерева меньше значения корня дерева, а все значения элементов правого поддерева больше значения корня, то такое дерево называется *деревом бинарного поиска* и предназначено для быстрого поиска элементов. Алгоритм поиска в таком дереве работает так: искомое значение сравнивается со значением корня дерева, и в зависимости от результата сравнения поиск либо заканчивается, либо продолжается только в левом или только в правом поддереве соответственно. Общее количество операций сравнения не будет превосходить так называемую *высоту дерева* – максимальное количество элементов на пути от корня дерева к одному из листьев. Так, высота изображенного на рисунке дерева равна 4.

Графы

Граф – это множество элементов, называемых *вершинами* графа вместе с набором отношений между этими вершинами, называемыми *ребрами* графа. Графической интерпретацией этой структуры данных является множество точек, соответствующих вершинам, некоторые пары из которых соединены линиями или стрелками, которые соответствуют ребрам. В последнем случае граф называется *ориентированным*.

В силу того, что с помощью графов можно описывать объекты произвольной структуры, графы являются основным средством для описания структур сложных объектов и функционирования систем, например, для описания вычислительной сети, транспортной системы, иерархической структуры (дерево является одной из разновидностей графа). Блок-схемы алгоритмов также представляют собой графы.

Если каждому ребру к тому же приписано некоторое число (*вес),* то такой граф называют *взвешенным.* Например, при описании с помощью графа системы дорог России важным является длина дороги (вес ребра графа), соединяющей те или иные населенные пункты (вершины графа). При этом на рисунке длины соответствующих ребер не обязаны соответствовать приписанным им весам в отличие от карты дорог.

Пример. В терминах взвешенного графа удобно решить задачу: Правительство России составляет план строительства современных автомагистралей, соединяющих города, население которых превышает миллион человек. Какие именно дороги следу-

ет построить, чтобы из любого такого города можно было добраться в любой другой по новым автомагистралям, а общая длина дорог была бы минимальной?

Эта задача в теории графов имеет простое и точное решение. Планирование сети дорог можно начать с любого города, например Санкт-Петербурга. Соединим его с ближайшим городом-миллионником. Далее на каждом шаге к имеющейся сети добавляется кратчайшая дорога, которая может соединить город, еще не присоединенный к сети, с одним из городов, уже включенных в сеть. Количество дорог будет на единицу меньше, чем число городов.

Абстрактную структуру данных *граф* в программе можно представить несколькими способами, т.е. используя разные типы данных. Например, граф можно описывать с помощью списка ребер, задавая каждое ребро парой вершин и, при необходимости, весом. Чаще используют табличное хранение графа, называемое также *матрицей смежности,* т.е. двухмерный массив, скажем, *A,* где для невзвешенного графа *A[i] [j] = true* (или 1), если ребро между вершинами *i* и *j* существует, и *A[i] [j] = false* (или 0) в противном случае. Для взвешенного графа A *[i] [j]* равно весу соответствующего ребра, а отсутствие ребра в ряде задач удобно обозначать бесконечностью. Для неориентированных графов матрица смежности всегда симметрична относительно главной диагонали (i = j). С помощью матрицы смежности легко проверить, существует ли в графе ребро, соединяющее вершину i с вершиной *j*. Основной же ее недостаток заключается в том, что матрица смежности требует, чтобы объем памяти был достаточен для хранения N^2 значений для графа, содержащего N вершин, даже если ребер в графе существенно меньше, чем N^2.

Массив – это конструкция данных. Данные (величины), входящие в состав массива, называются **элементами массива**. Массив имеет общее для всех элементов **имя** и общий для всех элементов **тип**. **Значение** массива – это совокупность (набор) значений его элементов. Другое название массива – **таблица**.

Индекс элемента – это порядковый номер элемента в массиве.

Типовые задачи: заполнение массива значениями, упорядочивание значений элементов, поиск элемента с заданными свойствами, поиск элемента по индексу, значению элемента с заданными свойствами и т.д.

Одномерные массивы

Описать одномерный массив – значит задать имя массива, тип массива и начальное и конечное значение индекса.

Обратиться к элементу одномерного массива – значит указать имя массива и значение индекса элемента.

Пример

Опишем массив A целых чисел на 10 элементов.

цел A[1..10]

1	2	3	4	5	6	7	8	9	10

Работа с массивом сводится к работе с его элементами. Для этого, как правило, используются циклы.

Двумерные массивы

Если индексов два, то массив называется **двумерным**.

Описать двумерный массив – значит задать имя массива, тип массива и начальные и конечные значения индексов.

Пример

Опишем массив B целых чисел на 25 элементов.

цел B[1..5, 1..5]

	1	2	3	4	5
1					
2					
3					
4					
5					

Обратиться к элементу двумерного массива – означает указать имя массива и значения индексов элемента.

Пример

Элемент массива B[2,4].

Работа с массивом сводится к работе с его элементами. Для этого, как правило, используются циклы.

О пользе быстрых алгоритмов

Часто разница между плохим и хорошим алгоритмом более существенна, чем между быстрым и медленным компьютером. Пусть необходимо отсортировать массив из миллиона чисел. Что быстрее: сортировать его вставками на суперкомпьютере (100 миллионов операций в секунду) или слиянием на домашнем компьютере (1 млн операций в секунду)? Пусть к тому же сортировка вставками написана на ассемблере чрезвычайно экономно, и для сортировки n чисел нужно, скажем, лишь $2n^2$ операций. В то же время алгоритм слиянием написан без особой заботы об эффективности и требует $50n \log(n)$ операций. Для сортировки миллиона чисел получаем

$(2*(10^6)^2$ опер.) / (10 опер. в секунду) = 20 000 с,

т.е. примерно 5,56 ч для суперкомпьютера

и $(50*(10^6)\log(10^6)$ опер.) / (10 опер. в секунду) = 1000 с,

т.е. примерно 17 мин для типового «домашнего» компьютера.

Очевидно, что разработка эффективных алгоритмов не менее важна, чем разработка быстрой электроники.

Контрольные вопросы

1. Что такое одномерный массив?
2. Для чего в программах используются двухмерные массивы?
3. Для чего используются одномерные массивы?
4. Как в программе использовать значение конкретного элемента двухмерного массива?
5. Как в программе использовать значение конкретного элемента одномерного массива?
6. Как можно заполнить двухмерный массив?
7. Как можно заполнить одномерный массив?
8. Как называется номер элемента одномерного массива?
9. Как описываются элементы одномерного массива?
10. Как описываются элементы двухмерного массива?
11. Как описываются элементы двухмерного массива?
12. Сколько индексов характеризуют элемент двухмерного массива?

Примеры выполнения заданий

Одномерные массивы

Задача №1. Описать одномерный массив C[1..5] целых чисел и ввести значения.

Решение.

1. Алгоритм решения задачи в виде графической схемы алгоритма.

2. Тестовые наборы:

3. Оценка объемной сложности разработанного алгоритма

	Данные	Тип	Объем, байт
1	*t*	**цел**	2
		Итого	2

4. Оценка временной сложности разработанного алгоритма

Операции	Тест 1
Присваиваний	1
Сравнений	0

Таким образом, алгоритм имеет сложность *O(n)*, т.е. линейную.

Задача №2. В массиве A[1..10] найти индекс максимального элемента и вывести его.

Решение.

1. Алгоритм решения задачи в виде графической схемы алгоритма

```
алг
цел n, i цел таб A[1..10]
нач
z:=1
для i от 2 до n
        если a[i]>a[z]
            то z:=i
вывод z
кон
```

2. Тестовые наборы:

3. Оценка объемной сложности разработанного алгоритма

	Данные	Тип	Объем, байт
1	*t*	**цел**	2
		Итого	2

4. Оценка временной сложности разработанного алгоритма.

Операции	Тест 1
Присваиваний	1
Сравнений	0

Таким образом, алгоритм имеет сложность *O(n)*, т.е. линейную.

Двумерные массивы

Задача №3. Разработайте алгоритм для формирования двумерного массива по образцу:

1 1 1 1 1
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0

Решение

1. Алгоритм решения задачи в виде графической схемы алгоритма.

2. Тестовые наборы:

3. Оценка объемной сложности разработанного алгоритма.

	Данные	Тип	Объем, байт
1	*t*	**цел**	2
		Итого	2

4. Оценку временной сложности разработанного алгоритма.

Операции	Тест 1
Присваиваний	1
Сравнений	0

Таким образом, алгоритм имеет сложность *O(n)*, т.е. линейную.