МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФГБОУ ВО «Брянский государственный технический университет»
(БГТУ)

Политехнический колледж (ПК БГТУ)

УТВЕРЖДАЮ
Ректор ФГБОУ ВО БГТУ

_____О.Н. Федонин
«30» августа 2020 г.

ДОКУМЕНТ ПОДПИСАН
ЭЛЕКТРОННОЙ ПОДПИСЬЮ

Сертификат: 00DF212636081B4F2FEC2C849AFF43F11D
Владелец: Федонин Олег Николаевич
Действителен: с 27.07.2023 до 19.10.2024

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ
по выполнению практических работ
по учебной дисциплине
ОП.08. Теория алгоритмов

Специальность:	**09.02.03 Программирование в компьютерных системах**
Уровень образования выпускника:	среднее профессиональное образование (СПО)
Программа подготовки специалиста среднего звена (ППССЗ):	базовая
Присваиваемая квалификация:	Техник-программист
Форма обучения:	очная
Срок получения СПО по ППССЗ:	3 года 10 месяцев
Уровень образования, необходимый для приема на обучение по ППССЗ:	основное общее образование
Год приема на обучение на 1-й курс:	2020

Брянск 2020

Методические указания по выполнению практических работ
по учебной дисциплине
ОП.08. Теория алгоритмов
(далее — МУ)
для специальности ***09.02.03 Программирование в компьютерных системах***

Разработал(и):

– преподаватель ПК БГТУ С.С. Шепотатьева

МР рассмотрены и одобрены на заседании предметно-цикловой комиссии «Программирование в компьютерных системах» ПК БГТУ (далее — ПЦК) от «30».08.2020г., протокол № 1

Председатель ПЦК Е. С. Трошина

Согласовано:

Заместитель директора ПК БГТУ
по учебно-методической работе Т.Е. Балашова

Общие указания к выполнению работ.

Перед началом выполнения работы внимательно ознакомьтесь с инструкцией, заданием к практической работе.

Отчет оформляется на листах со штампом. В отчет впишите тему, цель работы, оборудование, программное обеспечение. При выполнении работы следуйте приведенным пунктам плана. Номер пунктов и их названия должны соответствовать друг другу.

По мере выполнения работы необходимо сформулировать вывод.

Отчеты оформляются в журнале. На титульном листе должны быть указаны: название предмета, группа и фамилия студента.

Пример оформления титульного листа:

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФГБОУ ВО «Брянский государственный технический университет» (БГТУ)
Политехнический колледж (ПК БГТУ)

ЖУРНАЛ

практических работ по дисциплине

Теория алгоритмов

ПКТУ ОИ.0012.001ПР

Студент Петров Н.Н.

Группа:

Преподаватель

Введение

Методические указания к выполнению практических занятий по дисциплине «Теория алгоритмов» предназначены для закрепления теоретических знаний, полученных на лекциях, а также для овладения студентами умений и навыков применять эти знания при самостоятельной работе.

Перечень практических занятий соответствует рабочей программе по дисциплине «Математические методы»

Выполнение студентами практических занятий по дисциплине проводится с целью:

- закрепления полученных теоретических знаний по дисциплине;
- углубления теоретических знаний в соответствии с заданной темой;
- формирования умений решать практические задачи;
- развития самостоятельности, ответственности и организованности;
- формирования активных умственных действий студентов, связанных с поисками рациональных способов выполнения заданий;
- подготовки к зачёту.

Методические указания выполняют функцию управления самостоятельной работой студента, поэтому каждое занятие имеет унифицированную структуру, включающую определение целей занятия, оснащения занятия, порядок выполнения работы, а также задания и контрольные вопросы для закрепления темы.

Нормы оценки знаний, умений и навыков обучающихся по дисциплине

Оценка практических занятий обучающихся

Ответ оценивается отметкой «5», если:

- работа выполнена полностью;
- в логических рассуждениях и обосновании решения нет пробелов и ошибок;
- в решении нет математических ошибок (возможна одна неточность, описка, которая не является следствием незнания или непонимания учебного материала).

Отметка «4» ставится в следующих случаях:

- работа выполнена полностью, но обоснования шагов решения недостаточны (если умение обосновывать рассуждения не являлось специальным объектом проверки);
- допущены одна ошибка или есть два – три недочёта в выкладках, рисунках, чертежах или графиках (если эти виды работ не являлись специальным объектом проверки).

Отметка «3» ставится, если:

- допущено более одной ошибки или более двух – трех недочетов в выкладках, чертежах или графиках, но обучающийся обладает обязательными умениями по проверяемой теме.

Отметка «2» ставится, если:

- допущены существенные ошибки, показавшие, что обучающийся не обладает обязательными умениями по данной теме в полной мере.

Отметка «1» ставится, если:

- работа показала полное отсутствие у обучающегося обязательных знаний и умений по проверяемой теме или значительная часть работы выполнена не самостоятельно.

Учитель может повысить отметку за оригинальный ответ на вопрос или оригинальное решение задачи, которые свидетельствуют о высоком математическом развитии обучающегося; за решение более сложной задачи или ответ на более сложный вопрос, предложенные обучающемуся дополнительно после выполнения им каких-либо других заданий

Общая классификация ошибок

При оценке знаний, умений и навыков учащихся следует учитывать все ошибки (грубые и негрубые) и недочёты.

Грубыми считаются ошибки:

- незнание определения основных понятий, законов, правил, основных положений теории, незнание формул, общепринятых символов обозначений величин, единиц их измерения;
- незнание наименований единиц измерения;
- неумение выделить в ответе главное;
- неумение применять знания, алгоритмы для решения задач;
  неумение делать выводы и обобщения;
- неумение читать и строить графики;
- неумение пользоваться первоисточниками, учебником и справочниками;
- потеря корня или сохранение постороннего корня;
- отбрасывание без объяснений одного из них;
- равнозначные им ошибки;
- вычислительные ошибки, если они не являются опиской;
- логические ошибки.

К **негрубым ошибкам** следует отнести:

- неточность формулировок, определений, понятий, теорий, вызванная неполнотой охвата основных признаков определяемого понятия или заменой одного - двух из этих признаков второстепенными;
- неточность графика;
- нерациональный метод решения задачи или недостаточно продуманный план ответа (нарушение логики, подмена отдельных основных вопросов второстепенными);
- нерациональные методы работы со справочной и другой литературой;
- неумение решать задачи, выполнять задания в общем виде.

Недочетами являются:

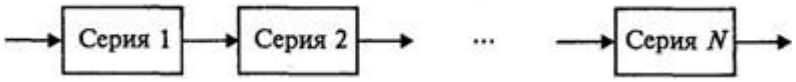
- нерациональные приемы вычислений и преобразований;
- небрежное выполнение записей, чертежей, схем, графиков.

Практическая работа №1. Построение линейного алгоритма

1. Цель работы:
Получение навыков построения алгоритмов линейного типа.

2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Способы записи алгоритмов.
- Преобразование выражений к линейному виду.

3. Теоретический материал
В конце 60-х — начале 70-х гг. XX столетия вырабатывается дисциплина, которая получила название структурного программирования. Ее появление и развитие связаны с именами Э. В. Дейкстры, Х.Д.Милса, Д. Е. Кнута и других ученых. Структурное программирование до настоящего времени остается основой технологии программирования. Соблюдение его принципов позволяет программисту быстро научиться писать ясные, безошибочные, надежные программы.
В основе структурного программирования лежит теорема, которая была строго доказана в теории программирования. Суть ее в том, что алгоритм для решения любой логической задачи можно составить только из структур «следование, ветвление, цикл». Их называют базовыми алгоритмическими структурами.
Следование — это линейная последовательность действий:



Каждый блок может содержать в себе как простую команду, так и сложную структуру, но обязательно должен иметь один вход и один выход.

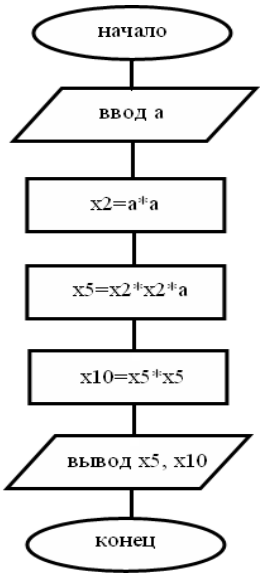
На практике наиболее распространены следующие формы представления алгоритмов:
• **словесная** (запись на естественном языке);
• **графическая** (изображения из графических символов);
• **псевдокоды** (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
• **программная** (тексты на языках программирования).

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном
языке.

Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным.
При графическом представлении алгоритм изображается в виде последовательности
связанных между собой функциональных блоков, каждый из которых соответствует

выполнению одного или нескольких действий.
Такое графическое представление называется схемой алгоритма или **блок-схемой**. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки
и т.п.) соответствует геометрическая фигура, представленная в виде **блочного символа**.
Блочные символы соединяются **линиями переходов**, определяющими очередность

4. Пример
Дано значение *a.* Не используя никаких функций и никаких операций, кроме умножения, получить значение a^5за три операции и a^{10} за четыре операции.



5. Задание
Составить программы, вычисляющие:

1. $\frac{b+\sqrt{b^2+4ac}}{2a} - a^3c + b^{-2}$
2. $\frac{CosX}{\pi - 2\,X} + 16X \cdot Cos(\,XY\,) - 2$
3. Вычислить периметр и площадь прямоугольного треугольника по заданным длинам двух катетов a и b.
4. Полторы кошки съедают за полтора часа полторы мышки. Сколько мышек съедят X кошек за Y часов?

Дополнительные задания
Дана сторона квадрата. Найти его периметр.
Дан радиус окружности. Найти ее диаметр.
Дана длина ребра куба. Найти объем куба и площадь его боковой поверхности.
Дан радиус окружности. Найти длину окружности и площадь круга.
Даны два целых числа. Найти их среднее арифметическое.
Известны объем и масса тела. Определить плотность материала этого тела.
Известны количество жителей в государстве и площадь его территории. Определить

плотность населения в этом государстве.
Даны катеты прямоугольного треугольника. Найти его гипотенузу.
Найти площадь кольца по заданным внешнему и внутреннему радиусам.
Даны два числа. Найти среднее арифметическое их модулей.
Даны два числа. Найти их сумму, разность, произведение, а также частное от деления первого числа на второе,
Известны координаты двух точек на плоскости. Найти расстояние между ними.
Составить программу обмена значениями двух переменных.
Составить программу обмена значениями трех переменных a, b, c по следующей схеме:
а) b присвоить значение c, a присвоить значение b, c присвоить значение a.
б) b присвоить значение a, c присвоить значение b, a присвоить значение c.
Дано вещественное число a. Не пользуясь никакими арифметическими операциями, кроме умножения, получить:
a^3 и a^{10} за четыре операции; a^4 и a^{20} за пять операций; a^5 и a^{13} за пять операций; a^5 и a^{19} за пять операций; a^2, a^5 и a^{17} за шесть операций. a^4, a^{12} и a^{28} за шесть операций.

Практическая работа №2. Построение алгоритма с неполным ветвлением

1. Цель работы:

Получение навыков построения алгоритмов с неполным ветвлением.

2. Темы для предварительной проработки

- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Способы представления алгоритмов.

3. Теоретический материал

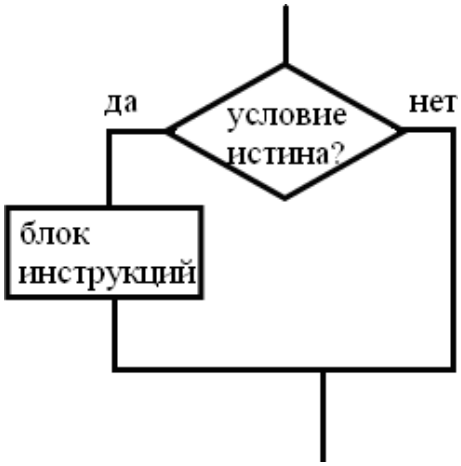
Ветвления в алгоритмах

Легко и просто было бы жить (и даже неинтересно), если бы удалось раз и навсегда расписать, какие поступки и в какой последовательности совершать. На самом деле нам постоянно приходится принимать решения в зависимости от создавшейся ситуации. Если идёт дождь, то мы раскрываем зонтик. Если нам весело, то мы смеёмся. Т.е. наши действия зависят от некоторых условий, возникающих извне, а выбор решения осуществляется как выбор ветви действия, либо одной, либо другой. Только ветвление поможет в сложных условиях сделать выбор.

Ветвление - это алгоритмическая конструкция, в которой в зависимости от условия выполняется та или иная последовательность действий.

Структура алгоритма, содержащая ветвление, называется разветвляющейся. Эта структура обеспечивает выбор между двумя альтернативами. Для определения направления, в котором пойдёт дальнейшее выполнение, делается проверка условия. Каждый из путей ведёт к общей точке слияния, так что выполнение алгоритма будет продолжаться независимо от того, какой путь был выбран.

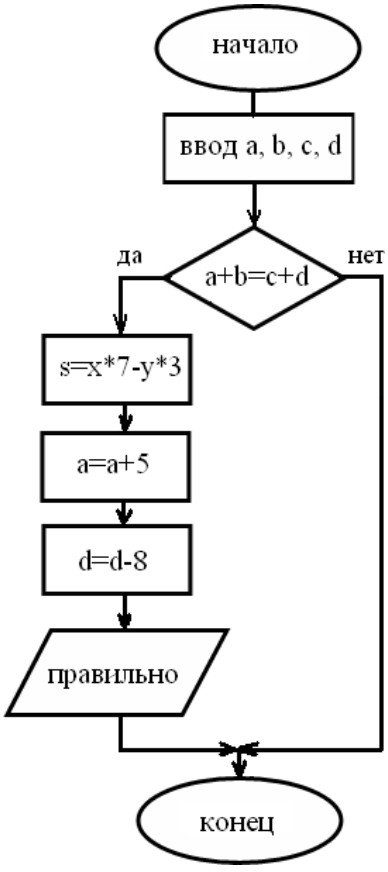
Часто в жизни встречаются ситуации, когда по одному из направлений движения по алгоритму может не совершиться ни одного действия, а по другому - совершится несколько действий. Такое ветвление называется ветвлением в неполной форме. В виде блок-схем такие алгоритмы можно записать так:



Условие – это любое допустимое в языке выражение, которое вычисляется компилятором. Результат вычисления интерпретируется следующим образом: 0 – ложь, все остальные значения – истина. Компилятор не проверяет правильность составления выражения, в любом случае выражение просто вычисляется.
е истина или ложь.

4. Пример

Составить блок-схему решения задачи:
Если a+b=c+d то сделать следующее: напечатать сообщение "Правильно", вычислить s=x*7-y*3, сложить a с пятью, вычесть из d восемь.



5. Задание

1. Задано число p. Если p<0, то y=p+5. Составьте алгоритм решения этой задачи помощью блок-схемы.
2. Составьте алгоритм, определяющий нахождение точки с координатами (x, y) в четвертой четверти на плоскости.
3. Составьте алгоритм, который проверял бы у ученика таблицу умножения по трём вопросам, причём, номер вопроса обозначьте переменной N, а количество правильных ответов - переменной K.

Дополнительные задания

1. Если q+p=c+d то сделать следующее: напечатать сообщение "Исправь", вычислить q=q*10+r*11, сложить t с двадцатью.
2. Подсчитать количество отрицательных среди чисел a, b, c.
3. Подсчитать количество положительных среди чисел a, b, c.
4. Подсчитать количество целых среди чисел a, b, c.
5. Определить, делителем каких чисел a, b, c является число k.
6. Даны три действительных числа. Возвести в квадрат те из них, значения которых неотрицательны.
7. Заданы размеры A, B прямоугольного отверстия и размеры x, y, z кирпича. Определить, пройдет ли кирпич через отверстие.
8. Написать программу, которая по заданным трем числам определяет, является ли

сумма каких-либо двух из них положительной.
9. Выбрать максимальное из двух чисел x, y и присвоить его значение переменной q.
10. Выбрать максимальное из трех чисел x, y, z и присвоить его значение переменной q.

Практическая работа №3. Построение алгоритма с полным ветвлением

1. Цель работы:

Получение навыков построения алгоритмов с полным ветвлением.

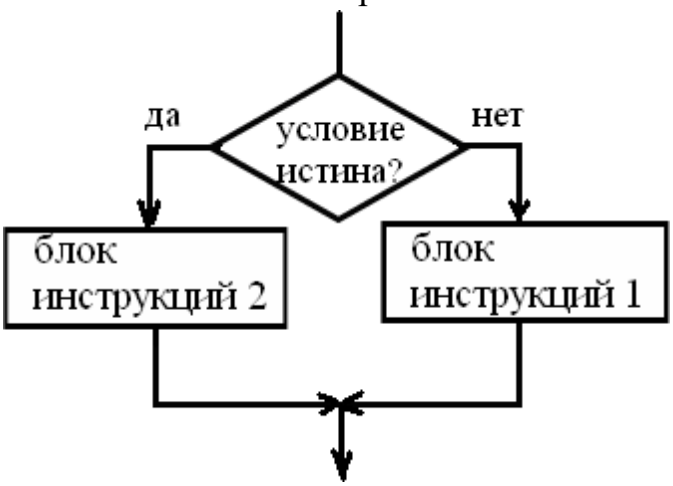
2. Темы для предварительной проработки

- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схемы алгоритма с неполным ветвлением.

3. Теоретический материал

Ветвления в алгоритмах

В практической работе №2 был рассмотрен материал построения алгоритмов с неполными ветвями. Также часто в жизни встречаются ситуации, когда по одному из направлений движения по алгоритму должны совершиться одни действия, а по другому - совершится другие действия. Такое ветвление называется ***ветвлением полной формы***. В виде блок-схем такие алгоритмы можно записать так:



Условия

Запись вида **b<= c>d** в информатике не применяется и может привести к неправильной работе программы.

Условие – выражение, которое компьютер и вычисляет и представляет результат в виде значения (1, 0)

Компьютер не следит за правильностью составления логического выражения, а только вычисляет его.

Условия могут быть простыми и составными.

Простое условие:

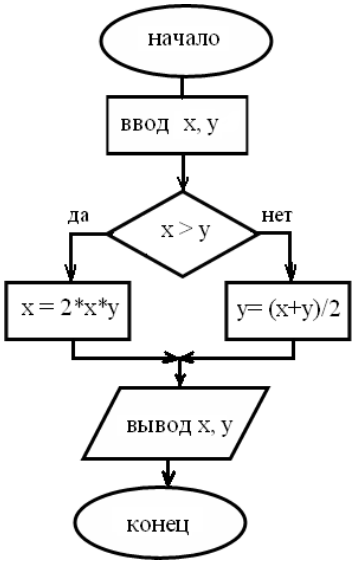
Операнд 1	**Знак операции отношения**	**Операнд 2**
Число	равно = =	Число
Текст	не равно !=>	Текст

Переменная	больше >	Переменная
Выражение	меньше <	Выражение
	больше или равно >=	
	меньше или равно <=	

Примеры:
A>= b + 3 * y
a + b == c*(a*x + d)
Не во всех задачах можно обойтись простым условием.

Составное условие
Составное условие получается из простых условий с помощью логических операций И, ИЛИ, НЕ (&&. || !). Для изменения порядка вычислений применяются скобки, как в алгебраических выражениях.
Примеры составных условий:
(x+y==c) && (z>3)
(b+x==z) || (t==r+q)
!x
! (a+b==v)
((a+b==g) || (s*f>n)) && (c*5==18)

4. Пример
Составить блок-схему решения задачи:
Даны два числа. Замените большее из двух данных чисел удвоенным произведением, а меньшее полусуммой этих чисел;



5. Задание
Даны два числа. Замените:
а) меньшее из двух чисел их суммой, а большее произведением этих чисел;
б) меньшее из двух данных чисел модулем разности, а большее модулем произведения этих чисел;
в) большее из двух данных чисел модулем суммы, а меньшее — модулем полуразности этих чисел.

Дополнительные задания

1. Заданы три числа x, y, z. Если x<0, то p задать как максимальное из x, y. Если x>=0, то p задать как минимальное из x, y.
2. Заданы два числа x, y. Если их сумма положительна, то p задать как x^2+y^2; если отрицательна или равна нулю, то p задать как $(x+y)^2$.
3. Заданы три числа x, y, z. Если x+y>z, положить s=x+y+z. Если x+y<=z, то s=x+y-z.
4. Подсчитать количество положительных и отрицательных чисел среди чисел a, b, c.
5. Даны три действительных числа. Возвести в квадрат те из них, значения которых неотрицательны, и в четвертую степень — отрицательные.
6. Даны целые числа x, y. Если числа не равны, то заменить каждое из них одним и тем же числом, равным большему из исходных, а если равны, то заменить числа нулями.

Практическая работа №4. Построение алгоритма с использованием конструкции выбора

1. Цель работы:

Получение навыков построения алгоритмов с использованием конструкции выбора.

2. Темы для предварительной проработки

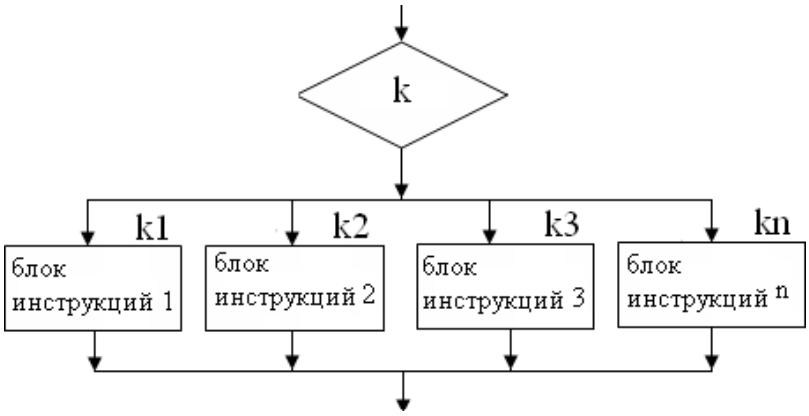
- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схемы алгоритма с использованием конструкций неполной, полной и вложенной ветвей.

3. Теоретический материал

Инструкция выбора обеспечивает многонаправленное ветвление. Она позволяет делать выбор одной из множества альтернатив. Хотя многонаправленное тестирование можно реализовать с помощью последовательности вложенных инструкций полного или неполного ветвления, во многих ситуациях инструкция выбора оказывается более эффективным решением.

Она работает следующим образом. Значение выражения последовательно сравнивается с константами из заданного списка. При обнаружении совпадения для одного из условий сравнения выполняется последовательность инструкций, связанная с этим условием.

Элемент *выражение* инструкции выбора должен при вычислении давать целочисленное или символьное значение. (Выражения, имеющие, например, тип с плавающей точкой, не разрешены.) Очень часто в качестве управляющего выражения используется одна переменная.



Итак, для применения инструкции выбора необходимо знать следующее.

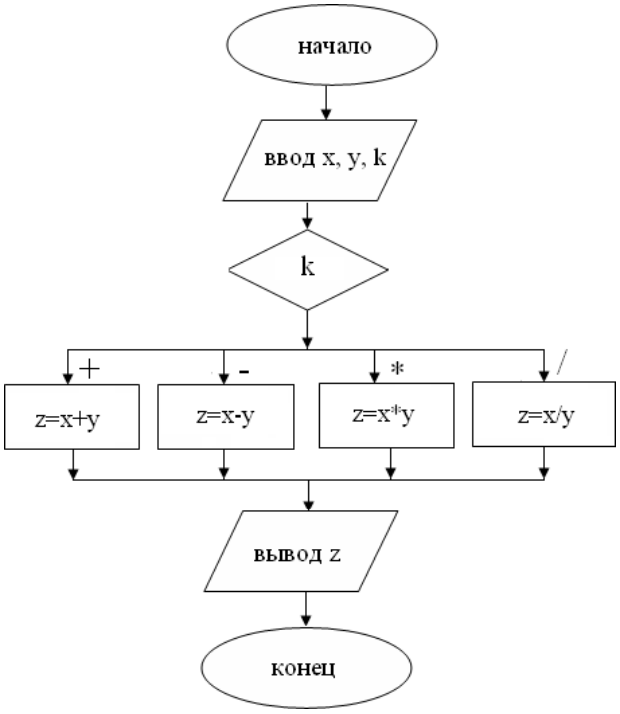
- ✓ Инструкция выбора отличается от инструкций ветвления тем, что выражение можно тестировать только с использованием условия равенства (т.е. на совпадение выражения с заданными константами), в то время как условие может быть любого типа.
- ✓ Никакие две константы в одной инструкции выбора не могут иметь идентичных значений.
- ✓ Инструкция выбора обычно более эффективна, чем вложенные ветвления.

Инструкция выбора может быть использована как часть внешней инструкции выбора. В этом случае она называется *вложенной* инструкцией выбора.

4. Пример

Составить алгоритм следующей задачи. Пользователь вводит два числа и знак операции. Программа в зависимости от введенного знака операции производит действия над числами – сложение, вычитание, умножение, деление.

Указание: k –знак арифметической операции.



5. Задание

1. Написать программу, которая по номеру дня недели (целому числу от 1 до 7) выдает в качестве результата количество уроков в вашем классе в этот день.
2. Составить программу, которая по заданным году и номеру месяца т определяет количество дней в этом месяце.

3. Для каждой введенной цифры (0 — 9) вывести соответствующее ей название на английском языке (0 — zero, 1 — one, 2 — two,...).
4. Составить программу, которая по данному числу (1—12) выводит название соответствующего ему месяца.
5. Составить программу, позволяющую получить словесное описание школьных отметок (1 — «плохо», 2 — «неудовлетворительно», 3 — «удовлетворительно», 4 — «хорошо», 5 — «отлично»).

Дополнительные задания
1. От пользователя вводится цифра английского алфавита. Если это I, V, X, L, C, D, M, то выдать десятичное значение данной римской цифры, иначе сообщить, что это не цифра.
2. Написать программу, позволяющую по последней цифре числа определить последнюю цифру его квадрата.
3. В старояпонском календаре был принят 12-летний цикл. Годы внутри цикла носили названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, курицы, собаки и свиньи. Написать программу, которая вводит номер некоторого года и печатает его название по старояпонскому календарю.
(Справка: 1996 г. — год Крысы — начало очередного цикла.)
4. Для целого числа A: от 1 до 99 напечатать фразу «Мне k лет», учитывая при этом, что при некоторых значениях k слово «лет» надо заменить на слово «год» или «года». Например, 11 лет, 22 года, 51 год.
5. Написать программу, которая бы по введенному номеру единицы измерения
(1 — дециметр, 2 — километр, 3 — метр, 4 — миллиметр, 5 — сантиметр) и длине отрезка L выдавала бы соответствующее значение длины отрезка в метрах.
6. Написать программу, которая по вводимому числу от 1 до 11 (номеру класса) выдает соответствующее сообщение «Привет, k-классник». Например, если k = 1, «Привет, первоклассник»; если k = 4, «Привет, четвероклассник».

Практическая работа №5. Построение алгоритма с использованием арифметического цикла

1. Цель работы:
Получение навыков построения алгоритмов с использованием арифметического цикла.

2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схемы алгоритма с использованием различного типа ветвлений.

3. Теоретический материал
Алгоритмы, которые мы составляли ранее, обладают одним общим свойством: при их выполнении каждое действие совершается один раз или вообще не совершается. Но для многих задач, решаемых на компьютере, характерно многократное выполнение отдельных участков вычислений.
Вычислительные процессы, при реализации которых имеет место многократное

выполнение одного или нескольких процессов вычислений, принято называть циклическими.

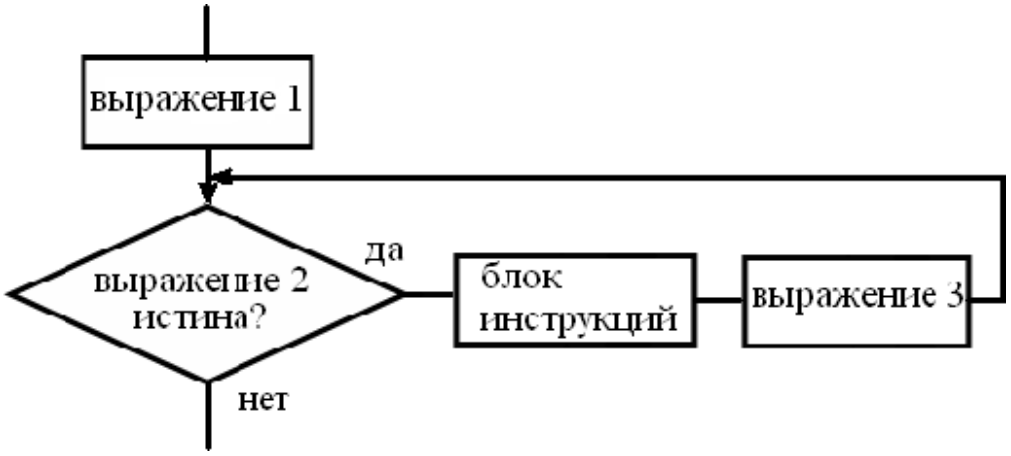
Цикл - это алгоритмическая конструкция, в которой в зависимости от условия повторяется определённая последовательность действий.

Цикл в алгоритме имеет особое значение, т.к. только его использование позволяет с помощью сравнительно коротких алгоритмов записывать длинные последовательности действий, что позволяет значительно уменьшить скорость выполнения программы.

Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.

Цикл называется арифметическим, если число повторений цикла известно заранее или может быть вычислено.

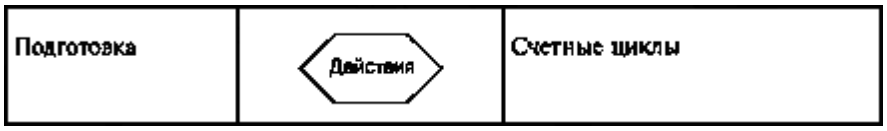
Выражение 1 выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла). Выражение 2 — это условие выполнения цикла. Выражение 3 обычно определяет изменение параметра цикла. Блок инструкций — тело цикла, то есть инструкции, которые должны выполняться заданное количество раз..



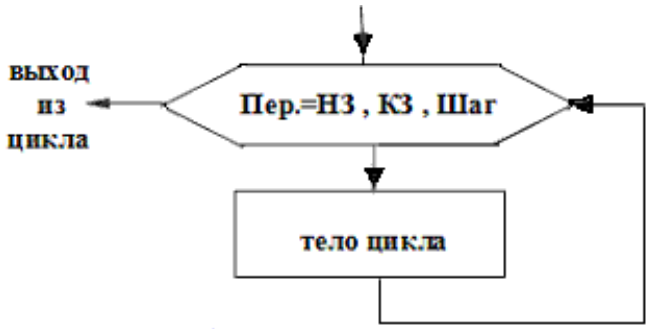
Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.

Цикл с параметрами иначе называется как ***цикл для каждого***.

Есть специальный элемент блок-схем для счетных циклов.



С использованием специального элемента алгоритм ***цикла для каждого*** выглядит так:

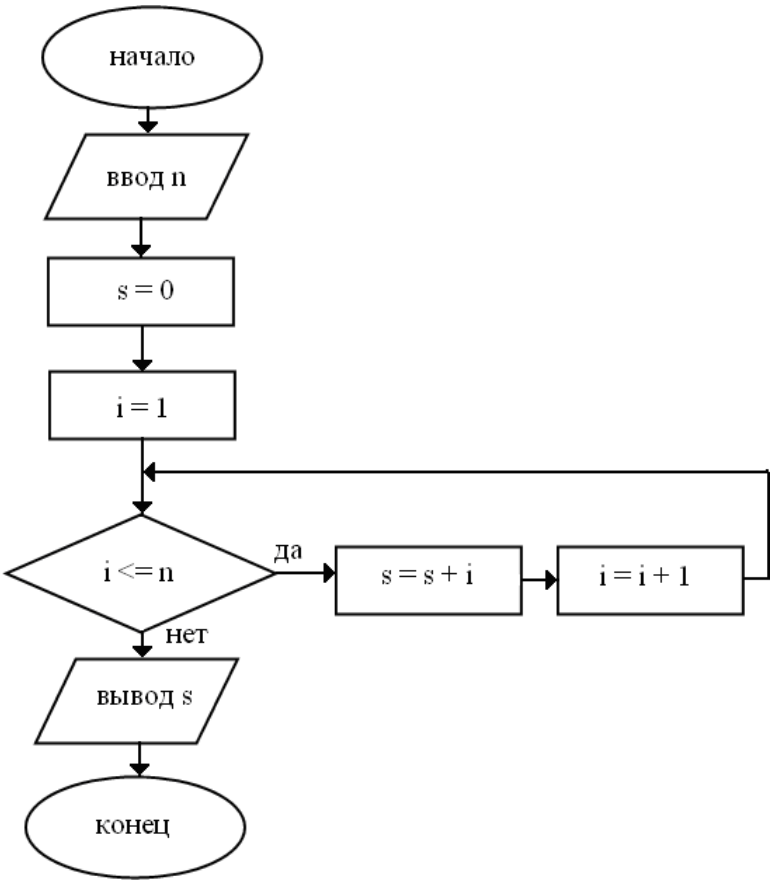


Пер. - переменная (параметр)
НЗ - начальное значение параметра
КЗ - конечное значение параметра
Шаг - величина изменения параметра после каждого выполнения тела цикла.

Тело цикла выполняется столько раз, сколько разных значений примет параметр в заданных пределах.

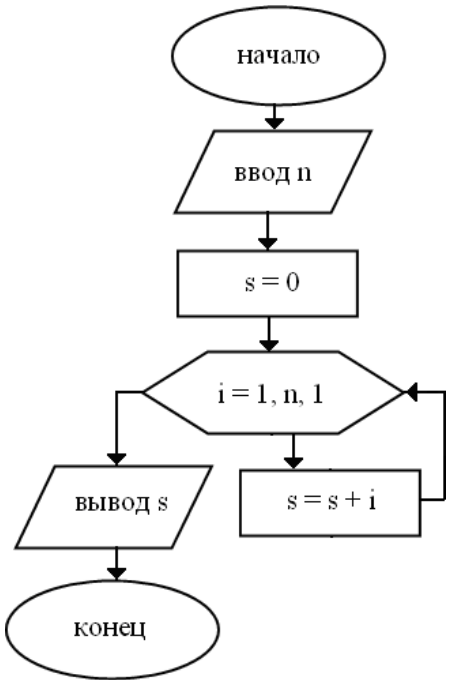
4. Пример

Составить алгоритм нахождения суммы первых n чисел.
Обратите внимание! Переменная s должна быть подготовлена – обнулена, чтобы не было искажения результата.



Выражение 1 i = 1
Выражение 2 i<= n
Выражение 3 i = i + 1
Блок инструкций s = s + i

Блок-схема алгоритма с использованием специального элемента для счетного цикла выглядит так:



Переменная i
Начальное значение параметра 1
Конечное значение параметра n
Величина изменения параметра после каждого выполнения тела цикла 1

5. Задание

1. Составьте алгоритм для задачи: вывести на экран все четные двузначные числа.
2. Вывести на экран все трехзначные числа, кратные восьми.
3. Вывести на экран в убывающем порядке все двузначные нечетные числа.
4. В переменную p по очереди вводятся 10 чисел. В переменной q получить максимальное из этих чисел.
5. Старший брат подарил мне 1 рубль в первый мой день рождения. В каждый следующий день рождения он удваивал свой подарок. Какую сумму я получу на 16-летие?
6. Начав тренировки, спортсмен в первый день пробежал 10 км. Каждый следующий день он пробегал на 10% больше, чем в предыдущий. Сколько километров он пробежит в седьмой день? Сколько всего километров он пробежит за семь дней?
7. Одноклеточная амеба каждые три часа делится на две клетки. Определите, сколько амеб будет через сутки.

Дополнительные задания

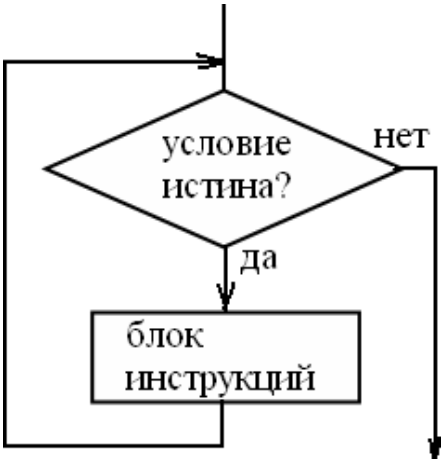
1. Старший брат подарил мне 1 рубль в первый мой день рождения. В каждый следующий день рождения он удваивал свой подарок и прибавлял к нему столько рублей, сколько лет мне исполнилось. Какую сумму я получу на N-летие? К какому дню рождения сумма подарка превысит 1000 рублей?
2. Вычислить сумму целых чисел о n до m.
3. Вычислить сумму кубов целых чисел от n до m.
4. Найти произведение n произвольных чисел.
6. Ввести n чисел. Определить, сколько из них превосходит первое число.
6. Ввести n чисел. Определить, сколько среди них положительных.

Практическая работа №6. Построение алгоритма с использованием цикла с предусловием

1. Цель работы:
Получение навыков построения алгоритмов с использованием цикла с предусловием.

2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схемы алгоритма с использованием арифметического цикла.

3. Теоретический материал
Алгоритмы, которые мы составляли ранее, обладают одним общим свойством: при их выполнении каждое действие совершается один раз или вообще не совершается. Но для многих задач, решаемых на компьютере, характерно многократное выполнение отдельных участков вычислений.
Вычислительные процессы, при реализации которых имеет место многократное выполнение одного или нескольких процессов вычислений, принято называть циклическими.
Цикл - это алгоритмическая конструкция, в которой в зависимости от условия повторяется определённая последовательность действий.
Цикл в алгоритме имеет особое значение, т.к. только его использование позволяет с помощью сравнительно коротких алгоритмов записывать длинные последовательности действий, что позволяет значительно уменьшить скорость выполнения программы.
Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.
Для организации циклов можно применять условия, т.к. циклический алгоритм является частным случаем разветвляющегося.
Тело цикла - это повторяющаяся последовательность действий (блок инструкций).
Цикл, для которого нельзя указать число повторений, и проверка окончания которого происходит по достижению нужного условия, называется ***итерационным***.
Проверка выполнения условия может происходить до выполнения тела цикла и после него, а циклические структуры являются циклами ***с предусловием***и **с *постусловием***.
В цикле с предусловием, называемом еще ***циклом “пока”,*** сначала проверяется условие, а затем выполняются действия.



Цикл с предусловием может не выполниться ни разу.

Решим задачу:
Составить алгоритм выполнения домашнего задания по геометрии, если задано решить несколько задач.

Решение

на разговорном языке:

Алгоритм: "Домашнее задание"
Исходные данные:
несколько нерешённых задач
Конечный результат:
все задачи решены

Последовательность действий:
а) Пока задачи не решены, повторять: реши задачу

на языке блок-схем:

Цикл "пока"

4. Пример

Составить алгоритм для определения наибольшего общего кратного двух целых положительных чисел, используя алгоритм Евклида (определение НОД методом вычитания).

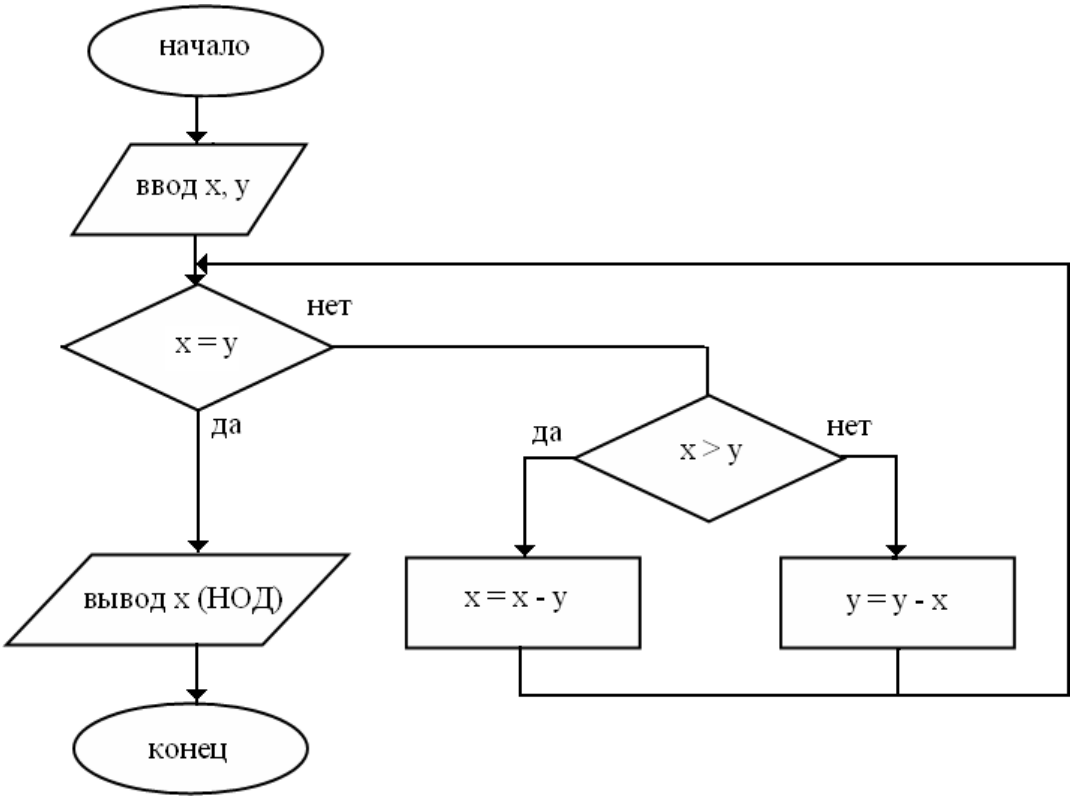
Согласно алгоритму Евклида для нахождения наибольшего общего кратного двух целых положительных чисел, нужно из большего числа вычитать меньшее до тех пор, пока они не станут равны, это число и есть наибольший общий делитель.

Возьмем два числа : 4 и 11.

Шаг	0	1	2	3	4	5
x	4	4	4	1	1	1
y	11	7	3	3	2	1

Таким образом, НОД чисел 4 и 11 равен 1.

Блок-схема решения задачи определения НОД методом Евклида:



5. Задание

1. Составить алгоритм решения задачи определения НОД методом деления. (алгоритм Евклида). Большее из чисел заменяется на остаток от целочисленного деления большего числа на меньшее. Повторяется это действие до тех пор, пока остаток не станет равным нулю, второе число при этом и есть НОД.

2. Начав читать детектив с 30-й страницы, читатель подсчитал сумму номеров прочитанных за день страниц. Она оказалась равна 840. Сколько страниц за день он прочитал?

3. Весной на березе распустилось в первый день A листьев, а в каждый последующий в три раза больше, чем в предыдущий. В какой день распустится на дереве B листьев за один день? Через сколько дней на дереве будет 1000000 листьев?

4. Николай родился A см в длину. За первый год он вырос на B см. В каждый последующий год он подрастал на 17% меньше, чем в предыдущий год. Через сколько лет его рост стал больше, чем 165 см?

5. Стакан кипятка (100 градусов Цельсия) остыл за первую минуту на A градусов, а каждую последующую остывал на 15% меньше, чем в предыдущую. Через сколько минут температура воды станет ниже N градусов Цельсия?

6. Седьмому классу на практике поручено сдуть 1000 одуванчиков. Бригада мальчиков в первый день сдула A одуванчиков, а бригада девочек – в 2 раза меньше. В каждый последующий день производительность мальчиков уменьшалась на 10%, а девочек росла на 15% по сравнению с предыдущим. Через сколько дней класс выполнит норму? Сколько сделает бригада мальчиков и бригада девочек в отдельности?

Дополнительные задания

1. Написать алгоритм Евклида, использующий соотношения НОД(a, b)=НОД(a mod b, b) при a>= b НОД (a, b)=НОД (a, bmoda) при b>= a

2. Написать алгоритм Евклида, использующий соотношения НОД (a, b)=НОД (a, bmoda) при b>= a
3. Даны натуральные a и b, не равные 0 одновременно. Найти d = НОД (a,b) и такие целые x и y, что d = a*x + b*y.

4. Даны натуральные a и b, не равные 0 одновременно. Найти d = НОД (a,b) и такие целые x и y, что d = a*x + b*y.

5. Даны натуральные a и b, не равные 0 одновременно. Найти d = НОД (a,b) и такие целые x и y, что d = a/x + b*y.

6. Даны натуральные a и b, не равные 0 одновременно. Найти d = НОД (a,b) и такие целые x и y, что d = a*x + b/y.

7. Написать вариант алгоритма Евклида, использующий соотношения
НОД(2*a, 2*b) = 2*НОД(a,b)

8. Написать вариант алгоритма Евклида, использующий соотношения
НОД(2*a, b) = НОД(a,b) при нечетном b
9. Написать бинарный алгоритм, использующий соотношения НОД(a, b)=НОД(a mod b, b) при a>= b НОД (a, b)=НОД (a, bmoda) при b>= a
10. Даны натуральные a и b, не равные 0 одновременно. Найти d = НОД (a,b) и такие целые x и y, что d = a*x + b*y, используя бинарный алгоритм.

Практическая работа №7. Построение алгоритма с использованием цикла с постусловием

1. Цель работы:

Получение навыков построения алгоритмов с использованием цикла с постусловием.

2. Темы для предварительной проработки

- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схемы алгоритма с использованием цикла с предусловием.

3. Теоретический материал

Алгоритмы, которые мы составляли ранее, обладают одним общим свойством: при их выполнении каждое действие совершается один раз или вообще не совершается. Но для многих задач, решаемых на компьютере, характерно многократное выполнение отдельных участков вычислений.

Вычислительные процессы, при реализации которых имеет место многократное выполнение одного или нескольких процессов вычислений, принято называть циклическими.

Цикл - это алгоритмическая конструкция, в которой в зависимости от условия повторяется определённая последовательность действий.

Цикл в алгоритме имеет особое значение, т.к. только его использование позволяет с помощью сравнительно коротких алгоритмов записывать длинные последовательности действий, что позволяет значительно уменьшить скорость выполнения программы.

Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.

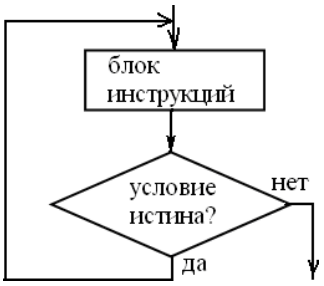
Для организации циклов можно применять условия, т.к. циклический алгоритм является частным случаем разветвляющегося.

Тело цикла - это повторяющаяся последовательность действий (блок инструкций).

Цикл, для которого нельзя указать число повторений, и проверка окончания которого происходит по достижению нужного условия, называется ***итерационным***.

Проверка выполнения условия может происходить до выполнения тела цикла и после него, а циклические структуры являются циклами ***с предусловием***и **с *постусловием***.

В цикле с постусловием, называемом ***циклом “до”,*** наоборот: сначала выполняется действие, а лишь потом проверяется условие.



Цикл “до” функционирует следующим образом: до тех пор, пока условие не выполнится, выполняется тело цикла. Характерно, что тело цикла в цикле ”до” выполняется хотя бы один раз.

Цикл с постусловием выполняется хотя бы один раз.

Решим задачу:

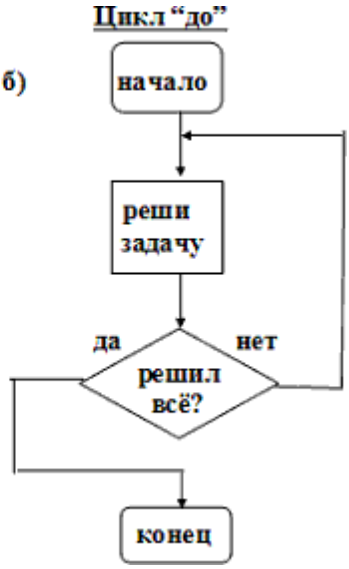
Составить алгоритм выполнения домашнего задания по геометрии, если задано решить несколько задач.

Решение

на разговорном языке:

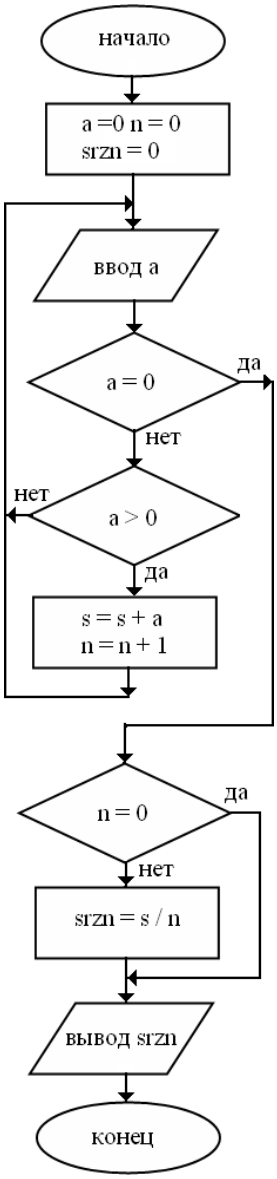
б) до тех пор, пока не решил задачу повторять: реши задачу

на языке блок-схем:



4. Пример

Вычислить среднее значение положительных чисел, введенных пользователем. Признак окончания ввода чисел – ввод значения ноль.



5. Задание

1. Определить максимальное число в последовательности положительных чисел, вводимых пользователем. Признак окончания ввода – ввод числа ноль.

2. Начальный вклад в банке 1000 рублей. Через каждый месяц вклад увеличивается на p процентов (p=7%). Определить, через сколько месяцев размер вклада превысит 1100 рублей. Вывести количество месяцев и сумму вклада. Пояснение. Процент начисляется на первоначальную сумму вклада.

Дополнительные задания

1. Составить алгоритм, определяющий, является ли введенное число простым (т.е делится на 1 и на само себя0.

2. Составить алгоритм программы, которая запрашивает пароль (например, четырехзначное число) до тех пор, пока он не будет правильно введен.

3. Старший брат подарил мне 1 рубль в первый мой день рождения. В каждый следующий день рождения он удваивал свой подарок и прибавлял к нему столько рублей, сколько лет мне исполнилось. К какому дню рождения сумма подарка превысит 1000 рублей?

Практическая работа №8. Построение алгоритма с использованием вложенного цикла

1. Цель работы:

Получение навыков построения алгоритмов с использованием вложенного цикла.

2. Темы для предварительной проработки

- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схемы алгоритма с использованием цикла с предусловием и постусловием.
- Построение блок-схемы алгоритма с использованием арифметического цикла.

3. Теоретический материал

Алгоритмы, которые мы составляли ранее, обладают одним общим свойством: при их выполнении каждое действие совершается один раз или вообще не совершается. Но для многих задач, решаемых на компьютере, характерно многократное выполнение отдельных участков вычислений.

Вычислительные процессы, при реализации которых имеет место многократное выполнение одного или нескольких процессов вычислений, принято называть циклическими.

Цикл - это алгоритмическая конструкция, в которой в зависимости от условия повторяется определённая последовательность действий.

Цикл в алгоритме имеет особое значение, т.к. только его использование позволяет с помощью сравнительно коротких алгоритмов записывать длинные последовательности действий, что позволяет значительно уменьшить скорость выполнения программы.

Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.

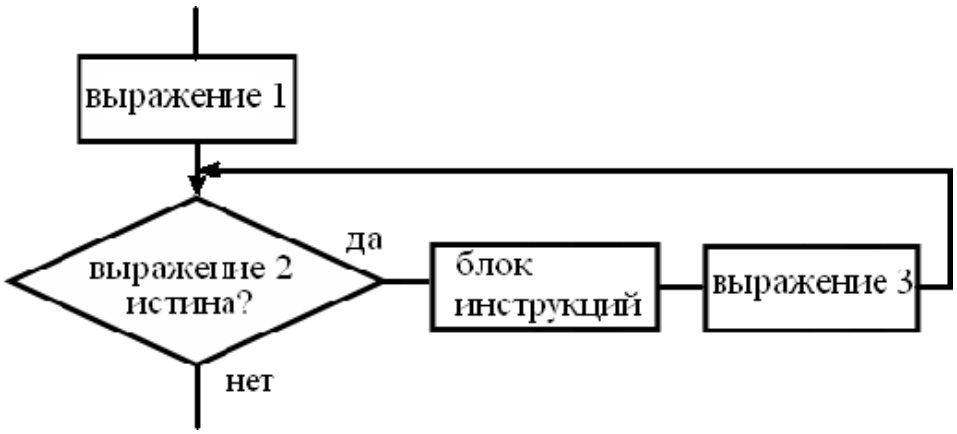
Для организации циклов можно применять условия, т.к. циклический алгоритм является частным случаем разветвляющегося.

Тело цикла - это повторяющаяся последовательность действий (блок инструкций).

Цикл называется ***арифметическим***, если число повторений цикла известно заранее или может быть вычислено.

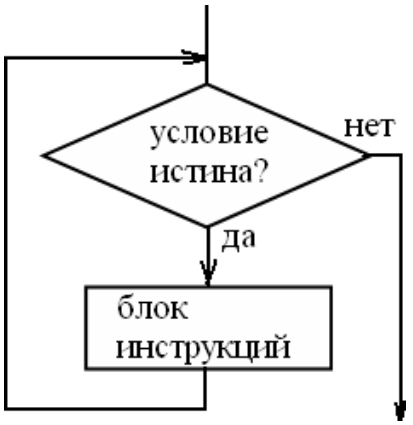
Цикл называется арифметическим, если число повторений цикла известно заранее или может быть вычислено.

Выражение 1 выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла). Выражение 2 — это условие выполнения цикла. Выражение 3 обычно определяет изменение параметра цикла. Блок инструкций — тело цикла, то есть инструкции, которые должны выполняться заданное количество раз.



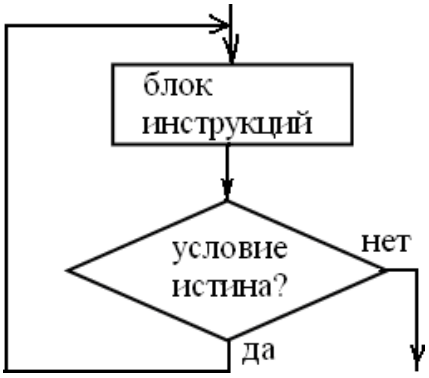
Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.

Цикл, для которого нельзя указать число повторений, и проверка окончания которого происходит по достижению нужного условия, называется ***итерационным***.
Проверка выполнения условия может происходить до выполнения тела цикла и после него, а циклические структуры являются циклами ***с предусловием*** и **с *постусловием***.
В цикле с предусловием, называемом еще ***циклом "пока",*** сначала проверяется условие, а затем выполняются действия.



Цикл с предусловием может не выполниться ни разу.

В цикле с постусловием, называемом ***циклом "до",*** наоборот: сначала выполняется действие, а лишь потом проверяется условие.



Цикл "до" функционирует следующим образом: до тех пор, пока условие не выполнится, выполняется тело цикла. Характерно, что тело цикла в цикле "до" выполняется хотя бы один раз.
Цикл с постусловием выполняется хотя бы один раз.

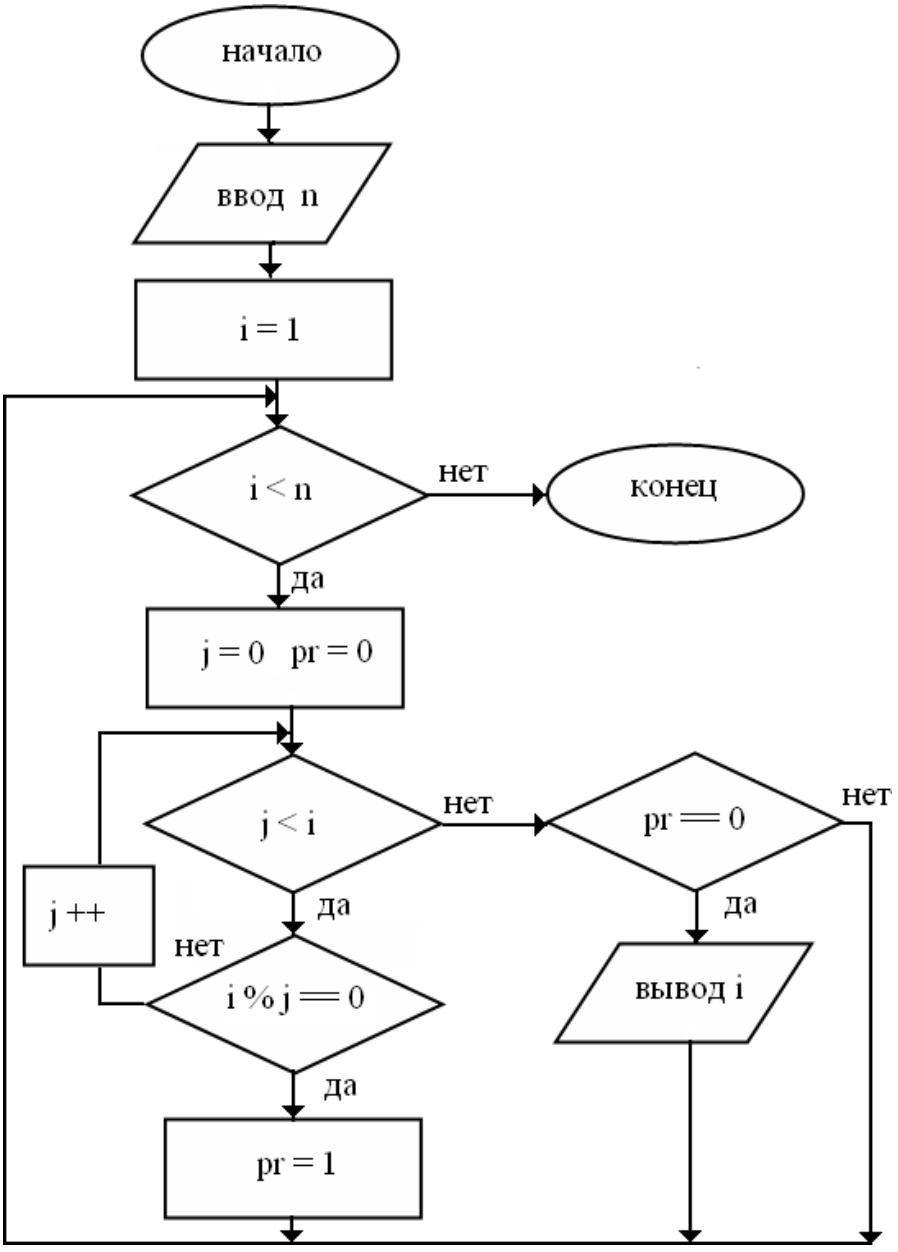
В теле цикла вновь может быть цикл, в таком случае говорят о ***вложенном цикле***.

4. Пример

Найти все простые числа в диапазоне от 1 до n.

Простым называется число, которое делится только на единицу и на само себя, например 1, 2, 3, 5, 7, 11, 13, 17, 19, 23 и т.д.

Для решения надо вначале получить очередное число (с помощью любого типа цикла), а затем проверить – на какие числа делится данное число, формируя делители от 2 до значения самого этого числа. Если таких делителей нет – число простое.



5. Задание

1) Оценки каждого из 18 учеников по трем предметам представлены в виде таблицы. Составить программу, в результате работы которой на экране будет отражена эта таблица. При этом программа должна запрашивать ввод каждой оценки. Задачу решить в двух вариантах:

а) ввод – вывод оценок осуществляется по строкам;

б) ввод – вывод оценок осуществляется по столбцам.

2) Оценки каждого из 18 учеников по трем предметам представлены в виде таблицы, в которой строки соответствуют ученикам, а столбцы – оценкам по трем предметам.

Составить программу, в результате работы которой на экране будет отражена информация:
а) количество «двоек» для каждого ученика;
б) средняя оценка по каждому предмету.
3) Напечатать полную таблицу умножения

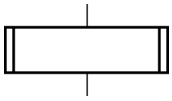
Дополнительные задания
1) Найти все целые числа из промежутка от1 до 300, у которых ровно пять делителей.
2) Найти натуральное число от a до b с максимальной суммой делителей.
3) Найти все целые числа из промежутка от 1 до 100, в десятичной записи которых есть цифра «7».
4) Найти сумму делителей каждого из целых чисел от 50 до 70.

Практическая работа №9. Построение алгоритма с использованием рекурсии

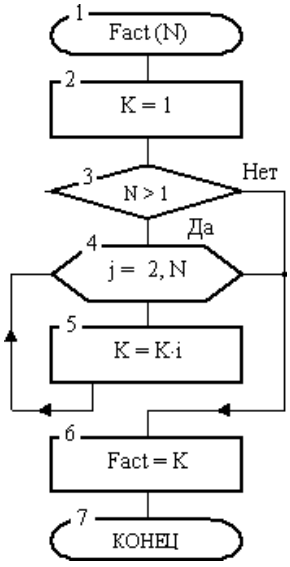
1. Цель работы:
Получение навыков построения алгоритмов с использованием рекурсии.

2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схемы алгоритма с использованием вложенного цикла.

3. Теоретический материал
При создании программы для решения сложной задачи программист обычно разбивает ее на подзадачи, чтобы для каждой такой подзадачи написать подпрограмму, а затем объединить все написанные подпрограммы в программу. Если подзадача достаточно сложная, то для решения может оказаться полезным разбить ее еще на более мелкие подзадачи и программировать каждую из них отдельно и т. д.
Блок-схема предопределенного процесса – обращение к подзадаче (в C++ к функции):



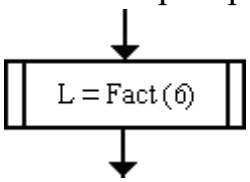
Рассмотрим задачу вычисления факториала числа N! = 1·2·3· · ·N. Результатом будет одно число, поэтому лучше алгоритм оформить в виде функции.
Ее блок-схема показана на рисунке. Переменная К используется для накопления произведения и, поскольку 0! = 1 и1! = 1, то в блоке 2 ей сразу присваивается значение 1. Далее, если N>1, то в цикле, образованном блоками 4-5, накапливается искомое произведение и помещается в переменную К. В блоке 6 имя Fact функции получает значение вычисленного произведения из ячейки К. Для процедур действия, размещенного в блоке 6, не может быть, а для функций оно должно быть обязательно, поскольку иначе значение функции на выходе окажется неопределенным.



Обращение к функции в других алгоритмах (головных, процедурах, функциях) производится по ее имени.

При этом оно может входить в состав выражений. В качестве фактических параметров могут быть использованы как переменные, константы, так и целые выражения. Важно только, чтобы фактический параметр был совместим по типу с формальным, который содержится в заголовке описания алгоритма.

Пример использования функции Fact показан на рисунке. В операторе присваивания используется обращение к функции для N = 6. После передачи этого значения в алгоритм и вычислений внутри него результат будет сначала присвоен имени функции, т. е. переменной Fact, а затем в операторе присваивания - переменной L.



Часто бывает, что в процессе такого разбиения задача сводится к самой себе. Если при этом исходные данные становятся проще, то этот процесс можно продолжать до тех пор, пока исходные данные не окажутся настолько простыми, что решение задачи для них станет тривиальным.

Процедура или функция может содержать вызов других процедур или функций. В том числе процедура может вызвать саму себя. Никакого парадокса здесь нет – компьютер лишь последовательно выполняет встретившиеся ему в программе команды и, если встречается вызов процедуры, просто начинает выполнять эту процедуру. Без разницы, какая процедура дала команду это делать.

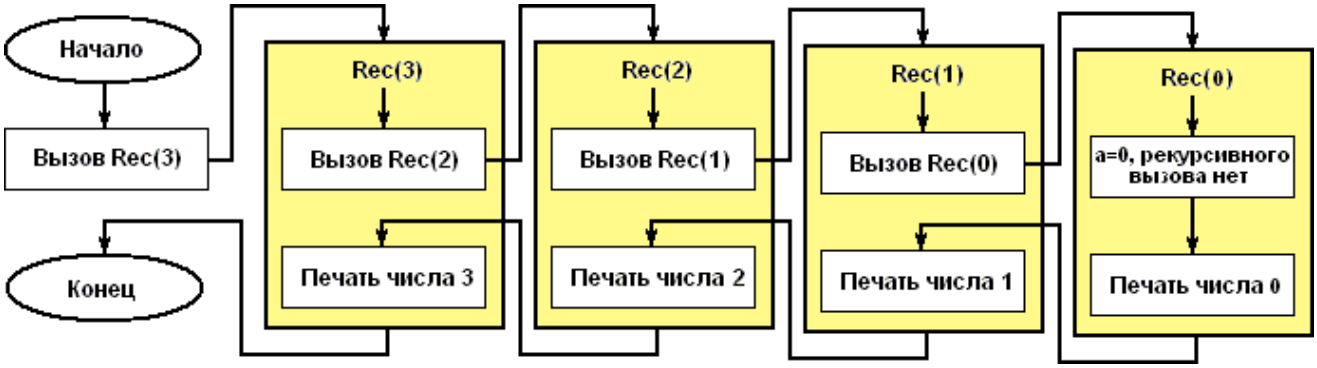
```
voidRec(inta){
if (a>0)Rec(a-1);
  cout<<"\n a="<<a;
}
```

Рассмотрим, что произойдет, если в основной программе поставить вызов, например, вида Rec(3).

```
void main(){clrscr();
int x;
cout<<"\nx=";cin>>x;
Rec(x);
```

getch(); }

Ниже представлена блок-схема, показывающая последовательность выполнения операторов.



Результат работы программы:

```
/*
x=3
a=0
a=1
a=2
a=3
*/
```

Процедура Rec вызывается с параметром a = 3. В ней содержится вызов процедуры Rec с параметром a = 2. Предыдущий вызов еще не завершился, поэтому можете представить себе, что создается еще одна процедура и до окончания ее работы первая свою работу не заканчивает. Процесс вызова заканчивается, когда параметр a = 0. В этот момент одновременно выполняются 4 экземпляра процедуры. Количество одновременно выполняемых процедур называют *глубиной рекурсии*.

Четвертая вызванная процедура (Rec(0)) напечатает число 0 и закончит свою работу. После этого управление возвращается к процедуре, которая ее вызвала (Rec(1)) и печатается число 1. И так далее пока не завершатся все процедуры. Результатом исходного вызова будет печать четырех чисел: 0, 1, 2, 3.

Еще один визуальный образ происходящего представлен на рисунке.

Выполнение процедуры Rec с параметром 3 состоит из выполнения процедуры Rec с параметром 2 и печати числа 3. В свою очередь выполнение процедуры Rec с параметром 2 состоит из выполнения процедуры Rec с параметром 1 и печати числа 2. И т. д.

4. Пример

Опишем рекурсивную функцию вычисления факториала. В функции один параметр — натуральное число *n.* Тривиальный случай — это вычисление значения 1!. Заметим, что верно следующее соотношение *n! = n* x *(n* — 1)!, поэтому можно свести задачу вычисления *n!* к решению той же задачи, но с другим, более "простым" параметром.

```
//rec03
//рекурсия вычисление факториала
#include <iostream.h>
#include <conio.h>
long fact(int n) {
int f;
  if (n==1) f=1;
else
  f=n*fact(n-1);
  return f;
  }

void main(){
clrscr(); int a;
cout<<"a="; cin>>a;
cout<<"\n Factorial="<<fact(a);
getch();
}
/*
a=7
Factorial=5040
*/
```

Пусть n=3. Как будет выполняться вызов fact(n) ? На рисунке представлена схема выполнения вызова функции. Стрелки указывают порядок вычисления. Сначала происходит движение по стрелкам вниз, указывающее на то, что происходит временное прерывание выполнения текущего вызова, и переход к выполнению вызова более низкого уровня, пока не будет получен тривиальный случай. Затем происходит подъем вверх, что означает возобновление прерванных вызовов. Первым возобновится выполнение такого вызова, который был прерван последним.

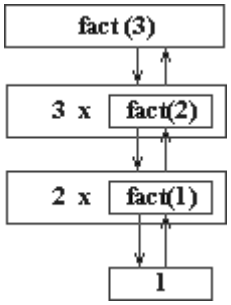


Схема выполнения вызова рекурсивной функции

5. Задание

1. Составить алгоритм вычисления НОД двух чисел с помощью рекурсивной функции.
2. Составить алгоритм вычисления значения очередного числа Фибоначчи с помощью

рекурсивной функции.
3. Напишите рекурсивную функцию, переворачивающую заданное натуральное число.

Дополнительные задания
1. Напишите рекурсивную функцию, которая по заданным натуральным числам *m* и *n* выводит все различные представления числа *n* в виде суммы *m* натуральных слагаемых. Представления, различающиеся лишь порядком слагаемых, считаются одинаковыми.
Напишите рекурсивную функцию, вычисляющую *n!!*.
2. Напишите рекурсивную функцию, определяющую количество единиц в двоичном представлении натурального числа.
3. Написать функцию сложения двух чисел, используя только прибавление единицы.
4. Написать функцию умножения двух чисел, используя только операцию сложения.
5. Проверить, является ли фрагмент строки с i-го по j-й символ палиндромом.
6. Подсчитать количество цифр в заданном числе.

Практическая работа №10. Построение алгоритма обработки одномерного массива

1. Цель работы:
Получение навыков построения алгоритмов с использованием одномерного массива.

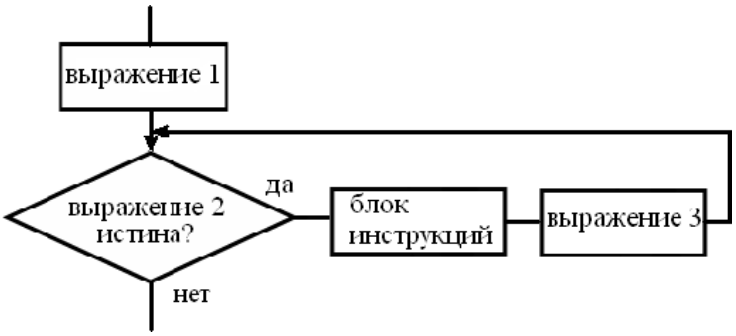
2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схем циклических алгоритмов.

3. Теоретический материал
До сих пор мы говорили о переменных, хранящих только одно значение, и рассматривали возможности различного представления и использования этого значения при решении конкретных задач.
Огромное количество алгоритмов требует одновременного хранения в памяти целых наборов однородных объектов, причем длина этих наборов может быть неизвестна.
Например, пусть необходимо обрабатывать данные о среднесуточной температуре за год для вычисления максимальной и минимальной температур, среднемесячной и среднегодовой температуры и т.п. Для реализации таких алгоритмов необходимо обеспечить хранение каждого отдельного значения среднесуточной температуры. Если иметь при этом в виду переменные вещественного тира, то таких переменных потребовалось бы 365.
Для решения подобных задач применяются структурированные типы данных.
Структурированные типы описывают наборы однотипных разнотипных данных, с которыми алгоритм должен работать как с одной именованной переменной.
Наиболее широко известная структура данных – массив.
Массив представляет собой упорядоченную структуру однотипных данных, которые называются элементами массива.
Доступ к каждому элементу массива осуществляется с помощью индекса – в общем случае порядкового номера элемента в массиве.
Массивы могут быть как одномерными (адрес каждого элемента определяется

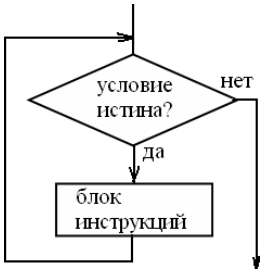
значением одного индекса), так и многомерными (адрес каждого элемента определяется значением нескольких индексов).
Массивы занимают смежные ячейки памяти.
(Другими словами, элементы массива в памяти расположены последовательно друг за другом.) Ячейка с наименьшим адресом относится к первому элементу массива, а с наибольшим — к последнему. Предположим, мы используем массив a из семи элементов. После заполнения массив будет выглядеть следующим образом.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
5	1	2	4	6	3	9

Для работы с одномерными массивами применяются циклические алгоритмы.
Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.
Тело цикла - это повторяющаяся последовательность действий (блок инструкций).
Цикл называется ***арифметическим***, если число повторений цикла известно заранее или может быть вычислено.
Цикл называется арифметическим, если число повторений цикла известно заранее или может быть вычислено.
Выражение 1 выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла). Выражение 2 — это условие выполнения цикла. Выражение 3 обычно определяет изменение параметра цикла. Блок инструкций — тело цикла, то есть инструкции, которые должны выполняться заданное количество раз.

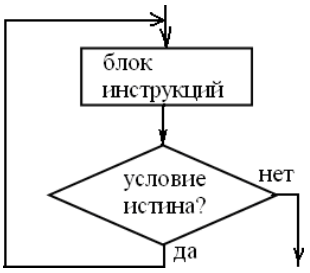


Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.
Цикл, для которого нельзя указать число повторений, и проверка окончания которого происходит по достижению нужного условия, называется ***итерационным***.
Проверка выполнения условия может происходить до выполнения тела цикла и после него, а циклические структуры являются циклами ***с предусловием***и **с *постусловием***.
В цикле с предусловием, называемом еще ***циклом “пока”,*** сначала проверяется условие, а затем выполняются действия.



Цикл с предусловием может не выполниться ни разу.

В цикле с постусловием, называемом ***циклом “до”,*** наоборот: сначала выполняется действие, а лишь потом проверяется условие.



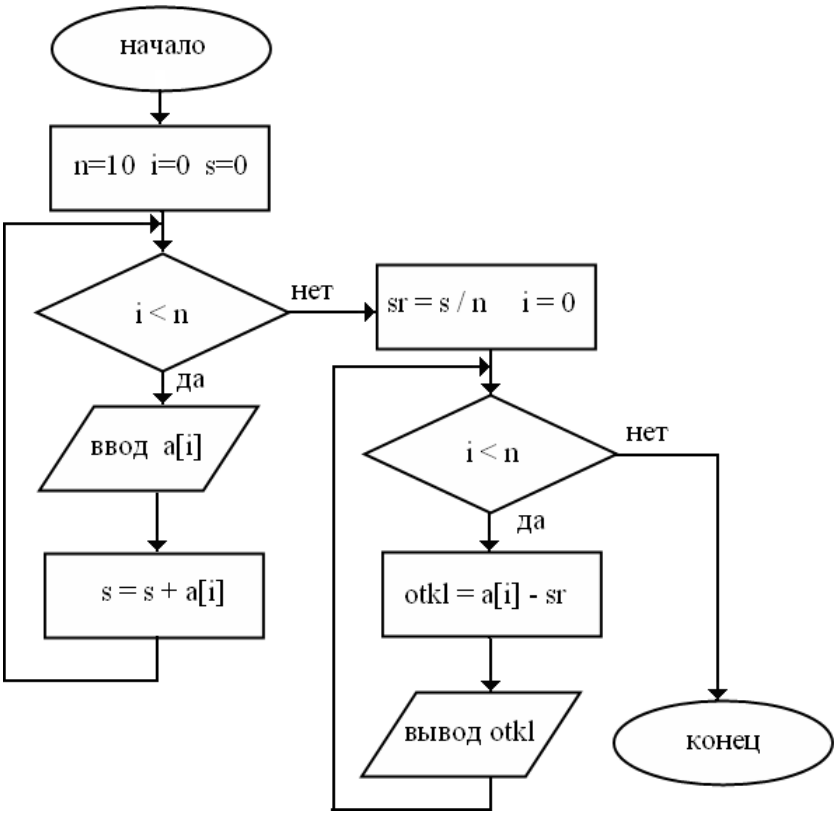
Цикл “до” функционирует следующим образом: до тех пор, пока условие не выполнится, выполняется тело цикла. Характерно, что тело цикла в цикле ”до” выполняется хотя бы один раз.

Цикл с постусловием выполняется хотя бы один раз.

В теле цикла вновь может быть цикл, в таком случае говорят о ***вложенном цикле***.

4. Пример

Ввести значения элементов массива с клавиатуры. Вычислить среднее значение элементов массива, вывести на экран отклонение каждого элемента массива от среднего значения.



5. Задание

Составить блок-схему алгоритма решения задачи.

1) Дан массив целых чисел. Выяснить:

а) является ли *p-й* элемент массива положительным числом;

б) является ли f-й элемент массива четным числом;

в) какой элемент массива больше, n-й или m-й.

2) В массиве хранятся сведения о количестве осадков, выпавших за каждый день сентября. Определить, сколько осадков выпадало в среднем за один день в первую, вторую и

третью декады этого месяца.
3) В массиве хранится информация о численности книг в каждом из 35 разделов библиотеки. Выяснить, верно ли, что общее число книг в библиотеке есть шестизначное число.

Дополнительные задания
1) В массиве хранится информация о количестве людей, живущих на каждом из 15 этажей дома (на первом этаже — в первом элементе массива, на втором — во втором и т. д.). Определить два этажа, на которых проживает меньше всего людей
2) Дан массив. Определить:
а) количество максимальных элементов в массиве;
б) количество минимальных элементов в массиве.
3) Найти число элементов массива, которые больше своих «соседей», т. е. предшествующего и последующего.

Практическая работа №11. Построение алгоритма заполнения одномерного массива по заданному закону

1. Цель работы:
Получение навыков построения алгоритмов с использованием одномерного массива.

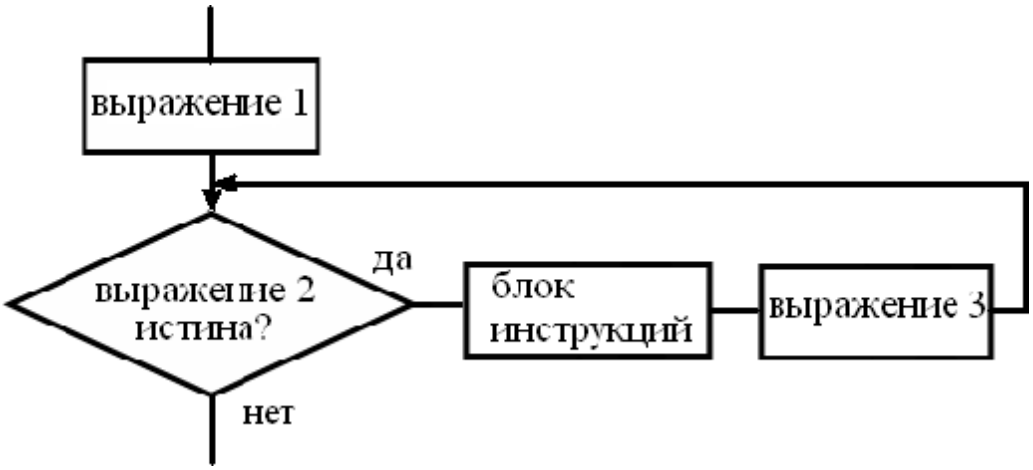
2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схем циклических алгоритмов.

3. Теоретический материал
До сих пор мы говорили о переменных, хранящих только одно значение, и рассматривали возможности различного представления и использования этого значения при решении конкретных задач.
Огромное количество алгоритмов требует одновременного хранения в памяти целых наборов однородных объектов, причем длина этих наборов может быть неизвестна.
Например, пусть необходимо обрабатывать данные о среднесуточной температуре за год для вычисления максимальной и минимальной температур, среднемесячной и среднегодовой температуры и т.п. Для реализации таких алгоритмов необходимо обеспечить хранение каждого отдельного значения среднесуточной температуры. Если иметь при этом в виду переменные вещественного тира, то таких переменных потребовалось бы 365.
Для решения подобных задач применяются структурированные типы данных.
Структурированные типы описывают наборы однотипных разнотипных данных, с которыми алгоритм должен работать как с одной именованной переменной.
Наиболее широко известная структура данных – массив.
Массив представляет собой упорядоченную структуру однотипных данных, которые называются элементами массива.
Доступ к каждому элементу массива осуществляется с помощью индекса – в общем случае порядкового номера элемента в массиве.
Массивы могут быть как одномерными (адрес каждого элемента определяется значением одного индекса), так и многомерными (адрес каждого элемента

определяется значением нескольких индексов).
Массивы занимают смежные ячейки памяти.
(Другими словами, элементы массива в памяти расположены последовательно друг за другом.) Ячейка с наименьшим адресом относится к первому элементу массива, а с наибольшим — к последнему. Предположим, мы используем массив a из семи элементов. После заполнения массив будет выглядеть следующим образом.

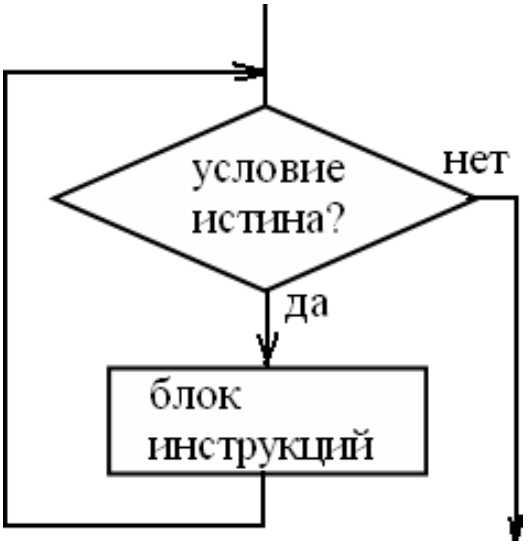
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
5	1	2	4	6	3	9

Для работы с одномерными массивами применяются циклические алгоритмы.
Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.
Тело цикла - это повторяющаяся последовательность действий (блок инструкций).
Цикл называется ***арифметическим***, если число повторений цикла известно заранее или может быть вычислено.
Цикл называется арифметическим, если число повторений цикла известно заранее или может быть вычислено.
Выражение 1 выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла). Выражение 2 — это условие выполнения цикла. Выражение 3 обычно определяет изменение параметра цикла. Блок инструкций — тело цикла, то есть инструкции, которые должны выполняться заданное количество раз..



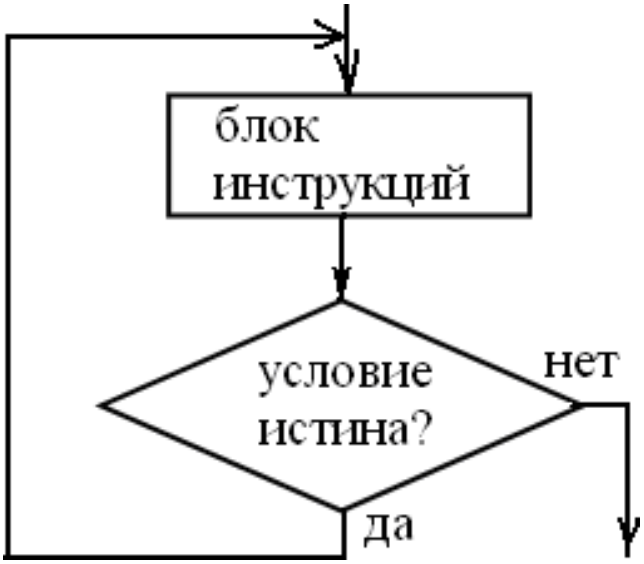
Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.

Цикл, для которого нельзя указать число повторений, и проверка окончания которого происходит по достижению нужного условия, называется ***итерационным***.
Проверка выполнения условия может происходить до выполнения тела цикла и после него, а циклические структуры являются циклами ***с предусловием*** и **с *постусловием***.
В цикле с предусловием, называемом еще ***циклом “пока”,*** сначала проверяется условие, а затем выполняются действия.



Цикл с предусловием может не выполниться ни разу.

В цикле с постусловием, называемом ***циклом "до",*** наоборот: сначала выполняется действие, а лишь потом проверяется условие.



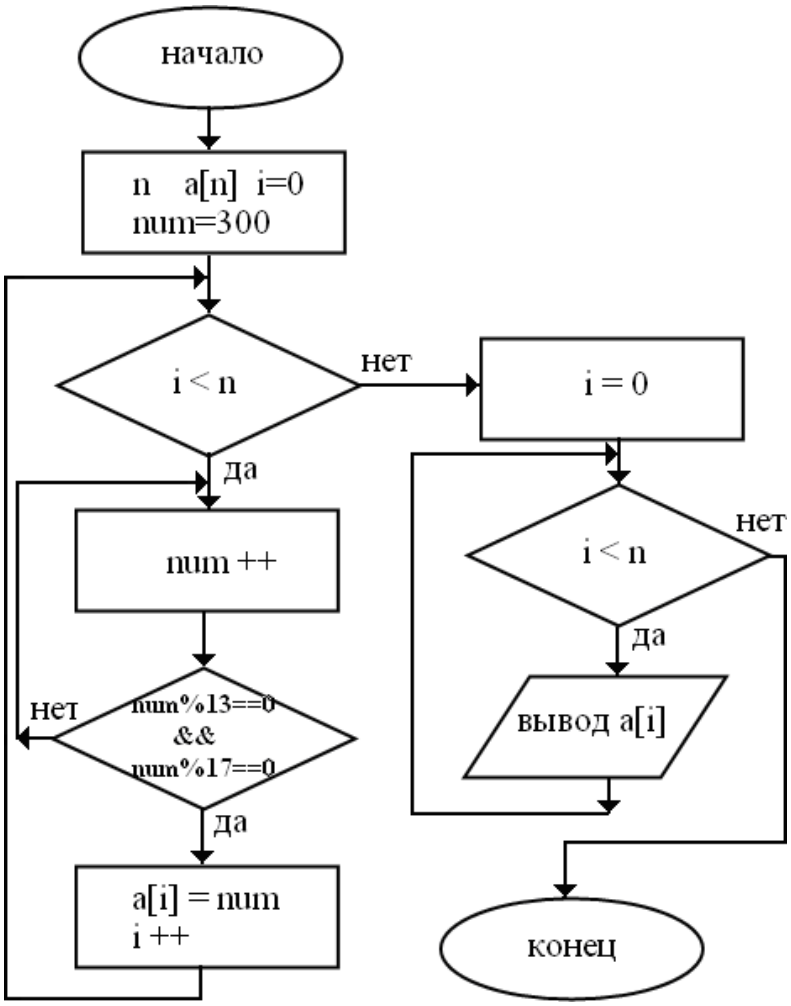
Цикл "до" функционирует следующим образом: до тех пор, пока условие не выполнится, выполняется тело цикла. Характерно, что тело цикла в цикле "до" выполняется хотя бы один раз.

Цикл с постусловием выполняется хотя бы один раз.

В теле цикла вновь может быть цикл, в таком случае говорят о ***вложенном цикле***.

4. Пример

Заполнить массив двадцатью первыми натуральными числами, делящимися нацело на 13 или на 17 и большими 300;



5. Задание

Составить блок-схему алгоритма решения задачи.

1) Заполнить массив тридцатью первыми простыми числами.

2) Вывести элементы массива на экран в обратном порядке.

3) Массив предназначен для хранения значений ростов двенадцати человек. С помощью датчика случайных чисел заполнить массив целыми значениями, лежащими в диапазоне от 163 до 190 включительно.

Дополнительные задания

1) Дано натуральное число *n (n*<999999). Заполнить массив его цифрами, расположенными в обратном порядке (первый элемент равен последней цифре, второй — предпоследней и т. д.). Незаполненные элементы массива должны быть равны нулю. Элементы массива, являющиеся цифрами числа *n,* вывести на экран.

2) Заполнить массив двенадцатью первыми членами последовательности Фибоначчи (последовательности, в которой первые два члена равны 1, а каждый следующий равен сумме двух предыдущих).

3) Используя датчик случайных чисел, заполнить массив из двадцати элементов неповторяющимися числами.

Практическая работа №12. Построение алгоритма пузырьковой сортировки

1. Цель работы:

Получение навыков построения алгоритмов с использованием пузырьковой сортировки.

2. Темы для предварительной проработки

- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схем циклических алгоритмов.
- Построение блок-схемы алгоритма заполнения одномерного массива .

3. Теоретический материал

Сортировка применяется во всех без исключения областях программирования, будь то базы данных или математические программы.

Практически каждый алгоритм сортировки можно разбить на три части:

- сравнение, определяющее упорядоченность пары элементов;
- перестановку, меняющую местами пару элементов;
- собственно сортирующий алгоритм, который осуществляет сравнение и перестановку элементов до тех пор, сока все элементы множества не будут упорядочены.

Пузырьковая сортировка

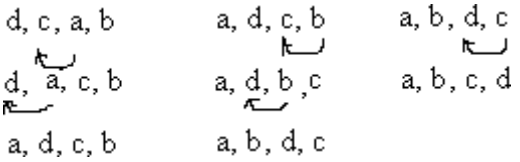
Самый простой алгоритм этой группы получил название пузырьковой сортировки. Не самый удачный алгоритм. Заключается в сравнении соседних элементов и, при необходимости их перестановке. При неубывающем упорядочении элементы "выплывают" как пузырьки каждый до своего уровня.

Описание алгоритма:

Используется два цикла. Внешний проходит N-1 раз, это гарантирует, что даже в худшем случае каждый элемент будет находиться на своем месте.

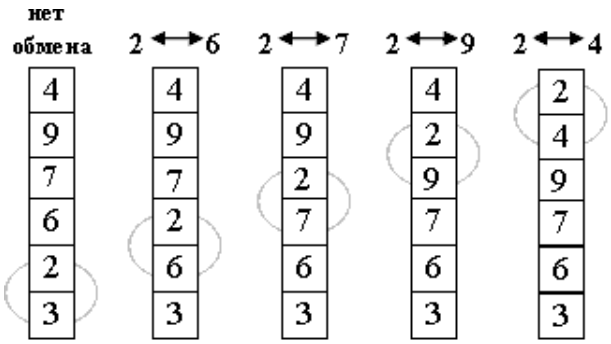
Исходный массив d, c, a, b.

В процессе работы программы массив будет изменяться следующим образом:



Расположим цифровой массив 4, 9, 7, 6, 2, 3 сверху вниз, от нулевого элемента - к последнему.

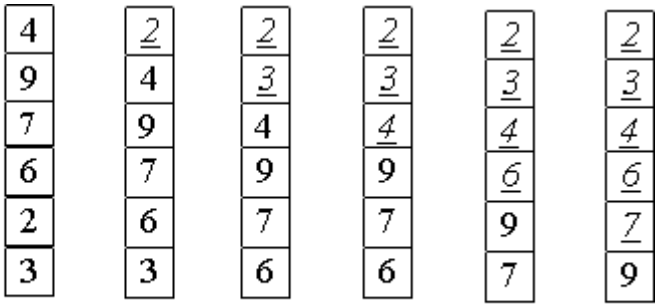
Идея метода: шаг сортировки состоит в проходе снизу вверх по массиву. По пути просматриваются пары соседних элементов. Если элементы некоторой пары находятся в неправильном порядке, то меняем их местами.



Нулевой проход, сравниваемые пары выделены

После нулевого прохода по массиву "вверху" оказывается самый "легкий" элемент - отсюда аналогия с пузырьком. Следующий проход делается до второго сверху элемента, таким образом второй по величине элемент поднимается на правильную позицию...

Делаем проходы по все уменьшающейся нижней части массива до тех пор, пока в ней не останется только один элемент. На этом сортировка заканчивается, так как последовательность упорядочена по возрастанию.

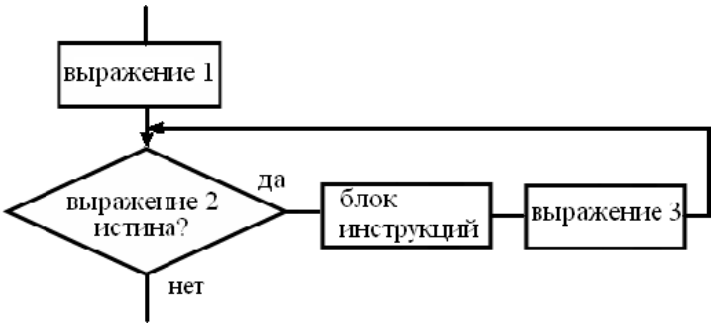


Теоретический материал для повторения

Массив представляет собой упорядоченную структуру однотипных данных, которые называются элементами массива.
Доступ к каждому элементу массива осуществляется с помощью индекса – в общем случае порядкового номера элемента в массиве.
Массивы могут быть как одномерными (адрес каждого элемента определяется значением одного индекса), так и многомерными (адрес каждого элемента определяется значением нескольких индексов).
Массивы занимают смежные ячейки памяти.
(Другими словами, элементы массива в памяти расположены последовательно друг за другом.) Ячейка с наименьшим адресом относится к первому элементу массива, а с наибольшим — к последнему. Предположим, мы используем массив a из семи элементов. После заполнения массив будет выглядеть следующим образом.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
5	1	2	4	6	3	9

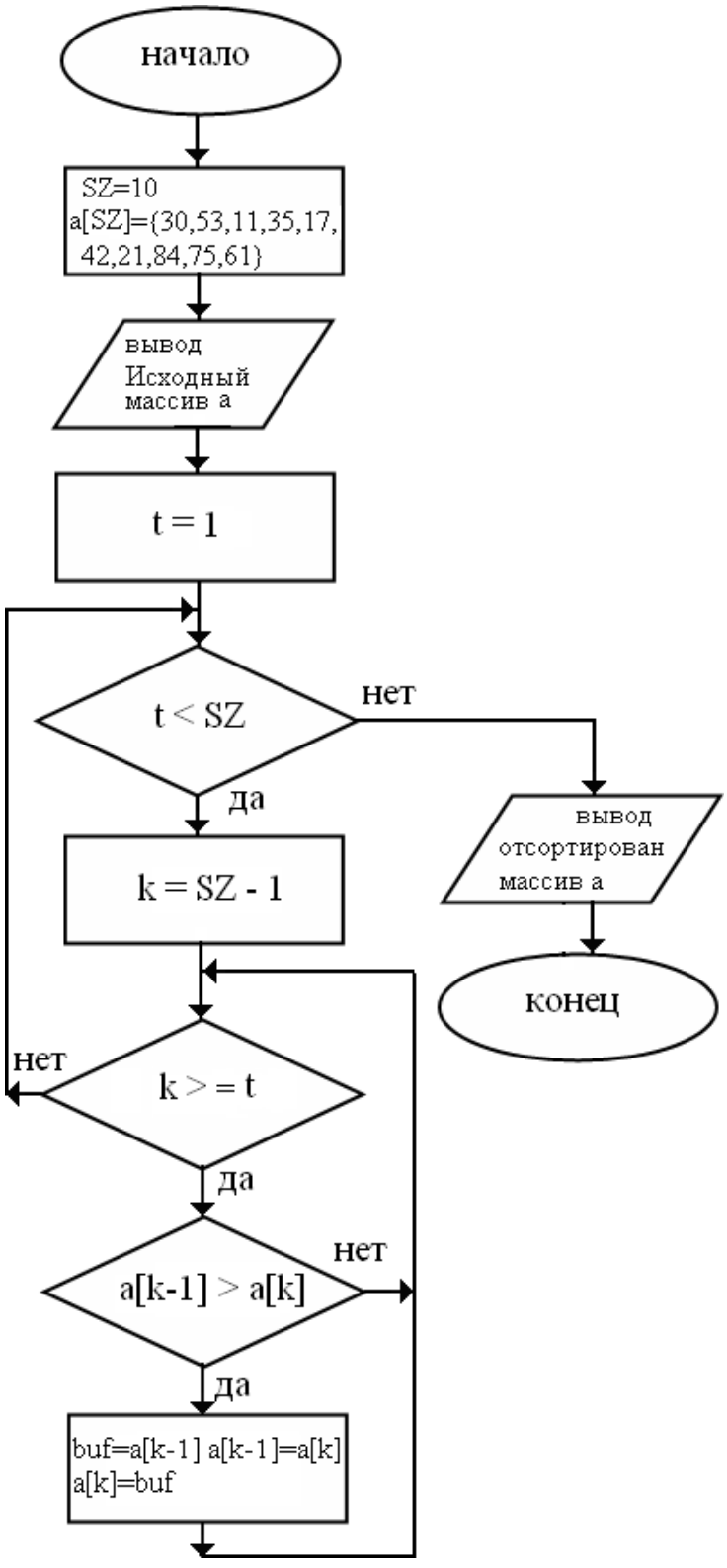
Для работы с одномерными массивами применяются циклические алгоритмы.
Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.
Тело цикла - это повторяющаяся последовательность действий (блок инструкций).
Цикл называется ***арифметическим***, если число повторений цикла известно заранее или может быть вычислено.
Цикл называется арифметическим, если число повторений цикла известно заранее или может быть вычислено.
Выражение 1 выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла). Выражение 2 — это условие выполнения цикла. Выражение 3 обычно определяет изменение параметра цикла. Блок инструкций — тело цикла, то есть инструкции, которые должны выполняться заданное количество раз..



Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.

4. Пример

Заполнить массив десятью значениями 30,53,11,35,17,42,21,84,75,61. Отсортировать данную последовательность методом выбора.



Чтобы не загромождать блок-схему, вывод исходного и отсортированного массивов обозначен одним блоком «вывод» (надо помнить, что для вывода на экран массива используется цикл с параметром).

Результат работы алгоритма:
Исходный массив: 30 53 11 35 17 42 21 84 75 61

11 30 53 17 35 21 42 61 84 75
11 17 30 53 21 35 42 61 75 84
11 17 21 30 53 35 42 61 75 84

11 17 21 30 35 53 42 61 75 84
11 17 21 30 35 42 53 61 75 84
11 17 21 30 35 42 53 61 75 84
11 17 21 30 35 42 53 61 75 84
11 17 21 30 35 42 53 61 75 84
11 17 21 30 35 42 53 61 75 84
**********Отсортированный массив
11 17 21 30 35 42 53 61 75 84

5. Задание
Составить блок-схему алгоритма решения задачи.

1) Изменить процедуру сортировки так, чтобы сортировка производилась по убыванию элементов.
2) Проверить, является ли данная последовательность целых чисел упорядоченной по убыванию. Если нет, упорядочить ее.
3) Отсортировать четные элементы массива с помощью пузырьковой сортировки.

Дополнительные задания
1) Составьте алгоритм, упорядочивающий элементы массива, стоящие на нечетных местах, в возрастающем порядке, а на четных - в убывающем.
2) В неупорядоченном массиве могут быть совпадающие элементы. Из каждой группы одинаковых элементов оставить только один, удалив остальные и «поджав» массив к его началу.
3) В массиве X(N) каждый элемент равен 0, 1 или 2. Переставить элементы массива так, чтобы сначала располагались все единицы, затем все двойки и, наконец, все нули (дополнительного массива не заводить).

Практическая работа №13. Построение алгоритма последовательного поиска

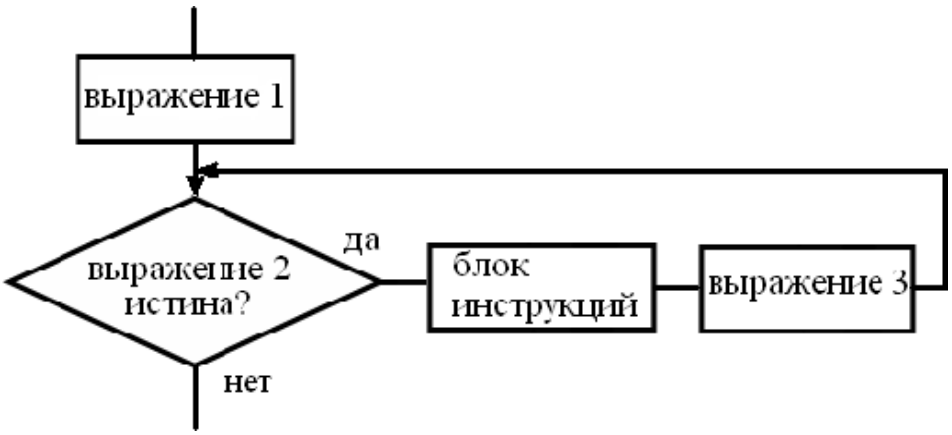
1. Цель работы:
Получение навыков построения алгоритмов последовательного поиска.
2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схем циклических алгоритмов.
- Построение блок-схемы алгоритма заполнения одномерного массива .

3. Теоретический материал
Теоретический материал для повторения
Массив представляет собой упорядоченную структуру однотипных данных, которые называются элементами массива.
Доступ к каждому элементу массива осуществляется с помощью индекса – в общем случае порядкового номера элемента в массиве.
Массивы могут быть как одномерными (адрес каждого элемента определяется значением одного индекса), так и многомерными (адрес каждого элемента определяется значением нескольких индексов).
Массивы занимают смежные ячейки памяти.

(Другими словами, элементы массива в памяти расположены последовательно друг за другом.) Ячейка с наименьшим адресом относится к первому элементу массива, а с наибольшим — к последнему. Предположим, мы используем массив a из семи элементов. После заполнения массив будет выглядеть следующим образом.

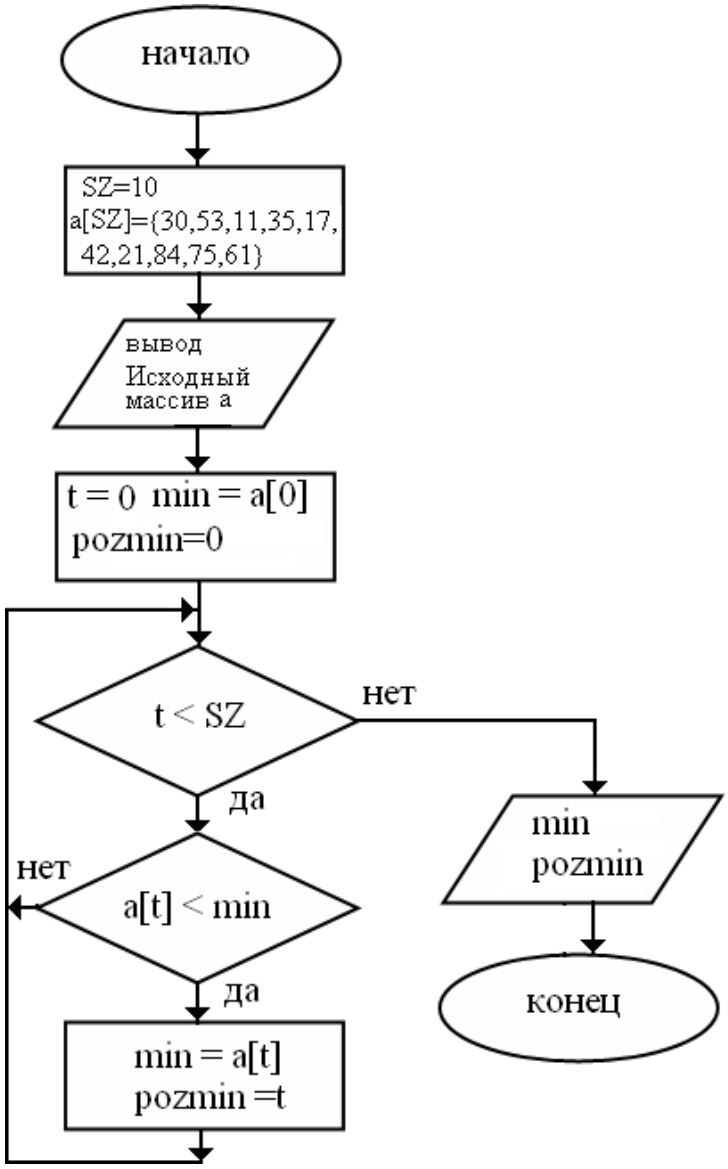
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
5	1	2	4	6	3	9

Для работы с одномерными массивами применяются циклические алгоритмы.
Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.
Тело цикла - это повторяющаяся последовательность действий (блок инструкций).
Цикл называется ***арифметическим***, если число повторений цикла известно заранее или может быть вычислено.
Цикл называется арифметическим, если число повторений цикла известно заранее или может быть вычислено.
Выражение 1 выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла). Выражение 2 — это условие выполнения цикла. Выражение 3 обычно определяет изменение параметра цикла. Блок инструкций — тело цикла, то есть инструкции, которые должны выполняться заданное количество раз..



Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.

Поиск минимального (максимального) элемента массива
Задачу поиска минимального элемента массива рассмотрим на примере массива целых чисел. Алгоритм поиска минимального (максимального) элемента массива довольно очевиден: сначала делается предположение, что первый элемент массива является минимальным (максимальным), затем остальные элементы массива последовательно сравниваются с этим элементом. Если во время очередной проверки обнаруживается, что проверяемый элемент меньше (больше) принятого за минимальный (максимальный), то этот элемент становится минимальным (максимальным) и продолжается проверка оставшихся элементов.



Чтобы не загромождать блок-схему, вывод исходного массива обозначен одним блоком «вывод» (надо помнить, что для вывода на экран массива используется цикл с параметром).

Поиск в массиве заданного элемента методом простого перебора

При решении многих задач возникает необходимость определить, содержит ли массив определенную информацию или нет. Например, проверить, есть ли в списке студентов фамилия Петров. Задачи такого типа называются поиском в массиве.

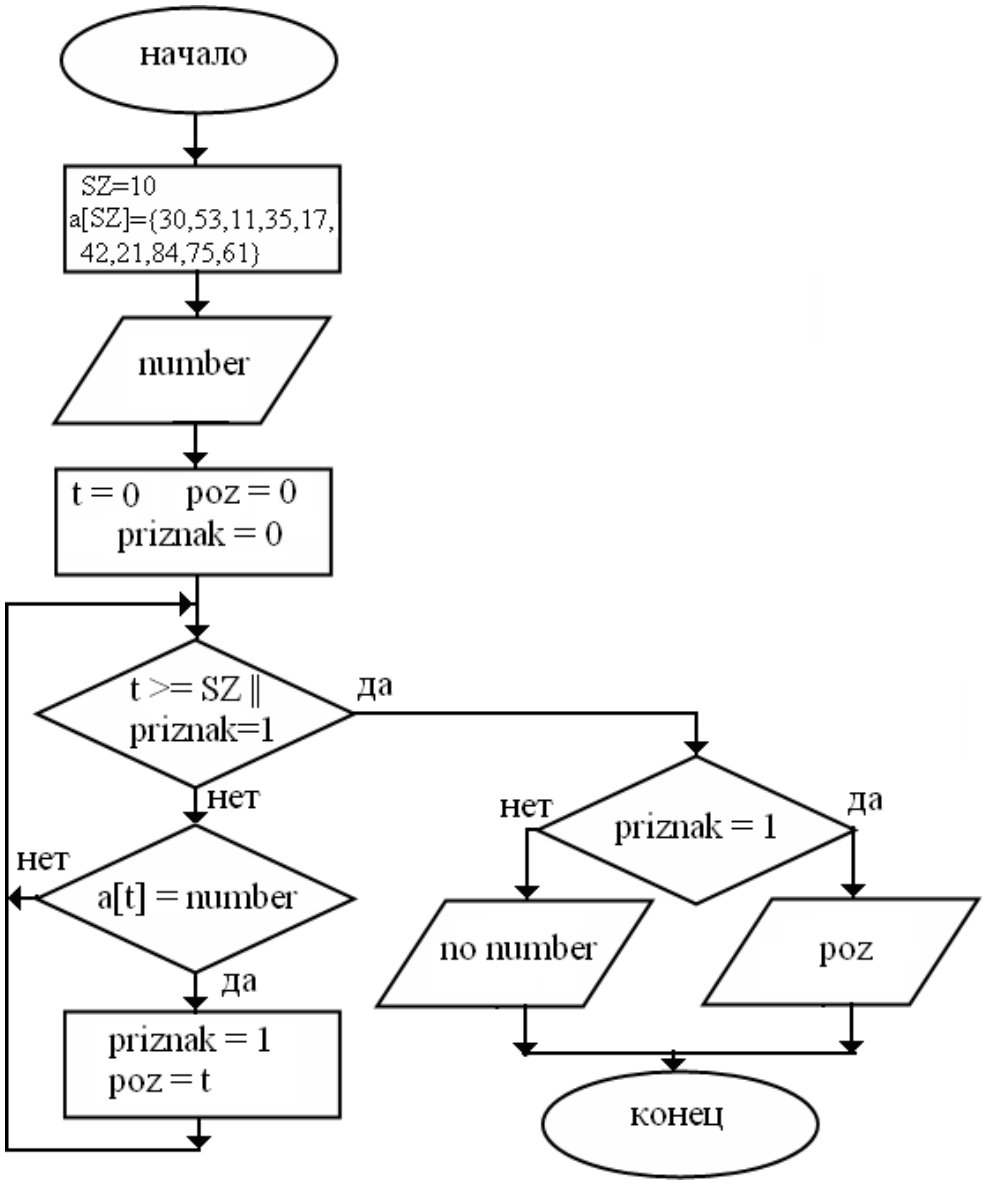
Для организации поиска в массиве могут быть использованы различные алгоритмы. Наиболее простой — это алгоритм простого перебора. Поиск осуществляется последовательным сравнением элементов массива с образцом до тех пор, пока не будет найден элемент, равный образцу, или не будут проверены все элементы. Алгоритм простого перебора применяется, если элементы массива не упорядочены.

Цикл просмотра элементов массива завершается, если в массиве обнаружен элемент, равный образцу (в этом случае значение специальной переменной-признака равно единице), или если проверены все элементы массива. По завершении цикла по значению переменной-признака можно определить, успешен поиск или нет.

Очевидно, что чем больше элементов в массиве и чем дальше расположен нужный элемент от начала массива, тем дольше программа будет искать необходимый элемент.

4. Пример

Заполнить массив десятью значениями 30,53,11,35,17,42,21,84,75,61. Составить алгоритм, который по введенному пользователем числу определит – есть это число в данной последовательности и выведет на экран позицию в массиве этого числа.



5. Задание

Составить блок-схему алгоритма решения задачи.

1) Определить число максимумов в заданной последовательности чисел.
2) Заменить все минимумы в последовательности чисел их утроенным значением.
3) Сжать числовой массив, удалив из него все максимумы.

Дополнительные задания

1) Найти повторяющиеся элементы в заданной последовательности чисел.
2) Найти неповторяющиеся элементы в заданной последовательности чисел.
3) Составить однопроходный алгоритм подсчета минимумов в последовательности чисел.

Практическая работа №14. Построение алгоритма бинарного поиска в упорядоченной последовательности.

1. Цель работы:

Получение навыков построения алгоритма бинарного поиска в упорядоченной последовательности.

2. Темы для предварительной проработки

- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схем циклических алгоритмов.
- Построение блок-схемы алгоритма заполнения одномерного массива .

3. Теоретический материал

Теоретический материал для повторения

Массив представляет собой упорядоченную структуру однотипных данных, которые называются элементами массива.

Доступ к каждому элементу массива осуществляется с помощью индекса – в общем случае порядкового номера элемента в массиве.

Массивы могут быть как одномерными (адрес каждого элемента определяется значением одного индекса), так и многомерными (адрес каждого элемента определяется значением нескольких индексов).

Массивы занимают смежные ячейки памяти.

(Другими словами, элементы массива в памяти расположены последовательно друг за другом.) Ячейка с наименьшим адресом относится к первому элементу массива, а с наибольшим — к последнему. Предположим, мы используем массив a из семи элементов. После заполнения массив будет выглядеть следующим образом.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
5	1	2	4	6	3	9

Для работы с одномерными массивами применяются циклические алгоритмы.

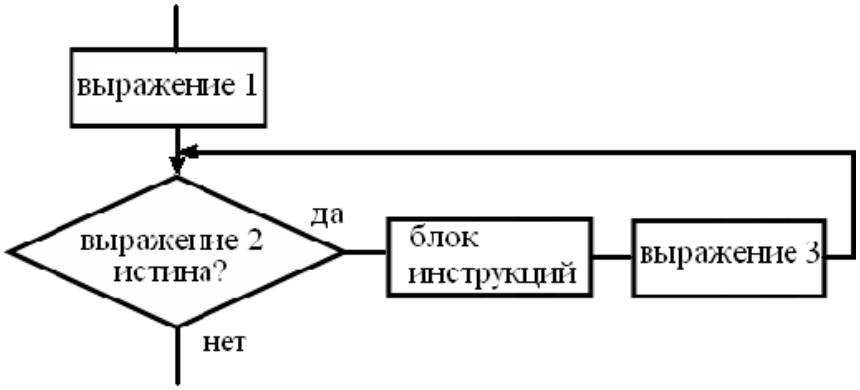
Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.

Тело цикла - это повторяющаяся последовательность действий (блок инструкций).

Цикл называется ***арифметическим***, если число повторений цикла известно заранее или может быть вычислено.

Цикл называется арифметическим, если число повторений цикла известно заранее или может быть вычислено.

Выражение 1 выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла). Выражение 2 — это условие выполнения цикла. Выражение 3 обычно определяет изменение параметра цикла. Блок инструкций — тело цикла, то есть инструкции, которые должны выполняться заданное количество раз..



Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.

Для организации поиска в массиве могут быть использованы различные алгоритмы. Наиболее простой — это алгоритм простого перебора. Поиск осуществляется последовательным сравнением элементов массива с образцом до тех пор, пока не будет найден элемент, равный образцу, или не будут проверены все элементы. Алгоритм простого перебора применяется, если элементы массива не упорядочены.

Метод бинарного поиска

На практике довольно часто производится поиск в массиве, элементы которого упорядочены по некоторому критерию (такие массивы называются упорядоченными). Например, массив фамилий, как правило, упорядочен по алфавиту, массив данных о погоде — по датам наблюдений. В случае, если массив упорядочен, то применяют другие, более эффективные по сравнению с методом простого перебора алгоритмы, один из которых — метод бинарного поиска.

Он известен также под именами *метод дихотомии* или *метод половинного деления*. Как обычно, за скорость взимается плата: массив должен быть упорядочен. Сам по себе этап предварительного упорядочения, или *сортировки*, обходится недешево, во всяком случае - дороже однократного линейного поиска.

Пусть есть упорядоченный по возрастанию массив целых чисел. Нужно определить, содержит ли этот массив некоторое число (образец).

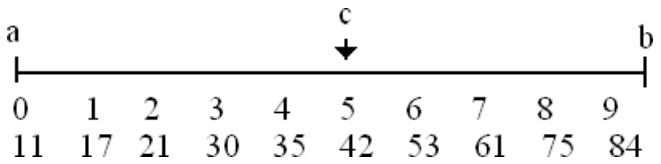
Идея алгоритма состоит в том, чтобы на каждом, очередном, шаге выделять из интервала индексов - для дальнейшего поиска - лишь его "половину", отбросив вторую.

1. Определяем середину интервала поиска.
2. Сравниваем образец с элементом, расположенным посередине. Если образец оказался больше, то областью дальнейшего поиска становится правая половина; в противном случае - левая половина интервала, но в любом случае индексный интервал уменьшается вдвое. (Если осуществить еще одну проверку, то можно установить и совпадение, после чего дальнейшие шаги не обязательны.)
3. Если остался интервал единичной длины, то переходим к заключительному шагу 4, в противном случае - к шагу 1.
4. Либо единственный элемент интервала совпадает с образцом, либо - искомого элемента в массиве нет.

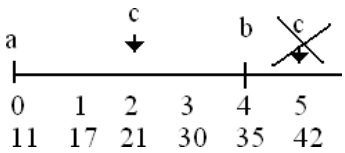
4. Пример

Заполнить массив десятью значениями 11, 17, 21, 30, 35, 42, 53, 61, 75, 84. Массив должен быть упорядочен! Составить алгоритм, который по введенному пользователем числу определит – есть это число в данной последовательности и выведет на экран

позицию в массиве этого числа.
Пусть число, введенное пользователем, к = 21.

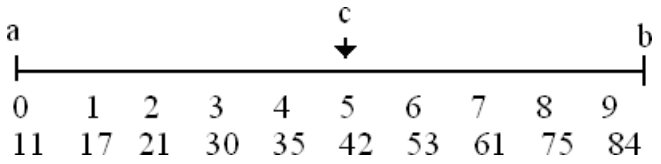


c = (a+b)/2=(0+9)/2=5 m[5]=42 21 < 42 поиск будет вестись в левой части a - c

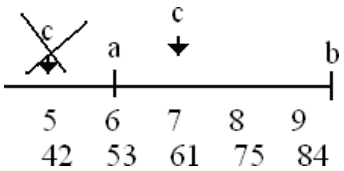


b=c-1=5-1=4 a=0 c=(a+b)/2=(0+4)/2=2 m[2]=21 Число найдено. Позиция в массиве равна 2.

Пусть число, введенное пользователем, к = 61.



c = (a+b)/2=(0+9)/2=5 m[5]=42 61 > 42 поиск будет вестись в правой части c - b

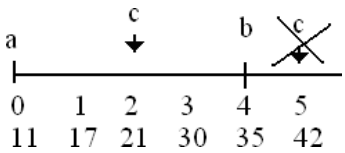


a=c+1=5=1=6; c=(a+b)/2=(6+9)/2=7 m[7]=61 Число найдено. Позиция в массиве равна 7.

Проверим, как проводится поиск чисел, которых нет в массиве.

Пусть число, введенное пользователем, к = 28.

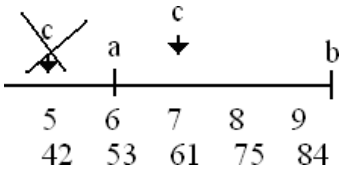
c = (a+b)/2=(0+9)/2=5 m[5]=42 28 < 42 поиск будет вестись в левой части a - c



b=c-1=5-1=4 c=(a+b)/2=(0+4)/2=2 m[2]=21 28 > 21 поиск будет вестись в правой части c – b
a=c+1=2=1=3 b=4 c=(a+b)/2=(3+4)/2=3 m[3]=30 28<30 поискслева a – c
b=c-1=3-1=2 a=3 c=(a+b)/2=(3+2)/2=2 a>c поиск прекращен, числа 28 в массиве нет

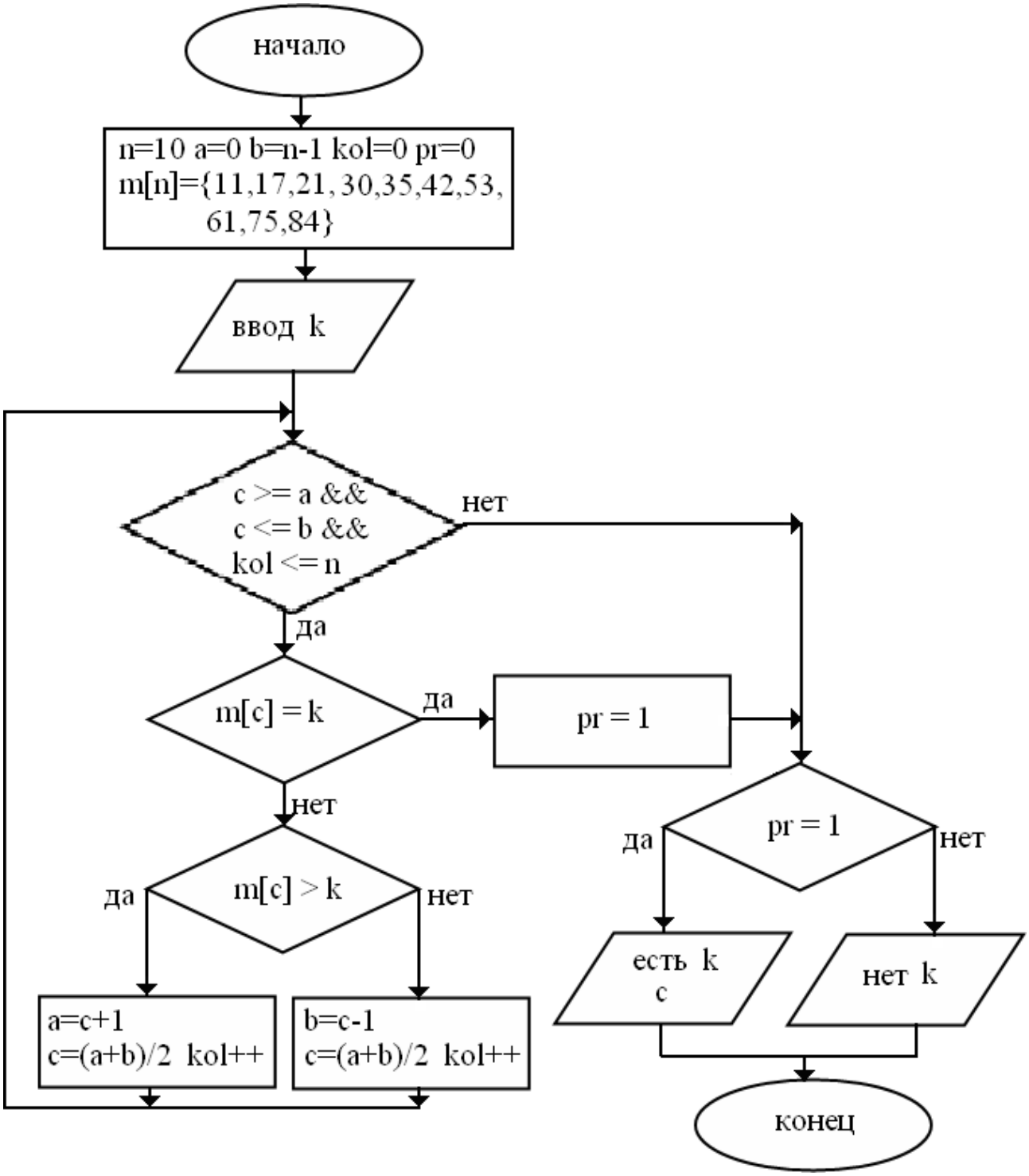
Пусть число, введенное пользователем, к = 65.

c = (a+b)/2=(0+9)/2=5 m[5]=42 65 > 42 поиск будет вестись в правой части c - b



a=c+1=5=1=6 b= 9 c=(a+b)/2=(6+9)/2=7 m[7]=61 65 > 61 поиксправа c - b
a=c=1=7+1=8 b=9 c=(a+b)/2=(8+9)/2=8 m[8]=75 65 < 75 поисклева a – c
b=c-1=8-1=7 a=8 c=(a+b)/2=(7+8)/2=7 a>c поиск прекращен, числа 65 в массиве нет

Аналогично можно проследить выполнение алгоритма бинарного поиска для чисел, которые выходят за границы диапазона значений элементов массива, например 5 и 100.



5. Задание

Составить блок-схему алгоритма решения задачи.

1) Использовать для поиска массив, упорядоченный по убыванию.

2) Составить игру. Пользователь должен отгадать число в заданном диапазоне.

3) Даны натуральные числа a1, a2, a3,…an и b1, d2, b3,…bm. a1>a2>a3….>an. Подсчитать количество тех b_i, 1<=i<=m, для которых нет равных среди a1, a2, a3,…a50.

Дополнительные задания

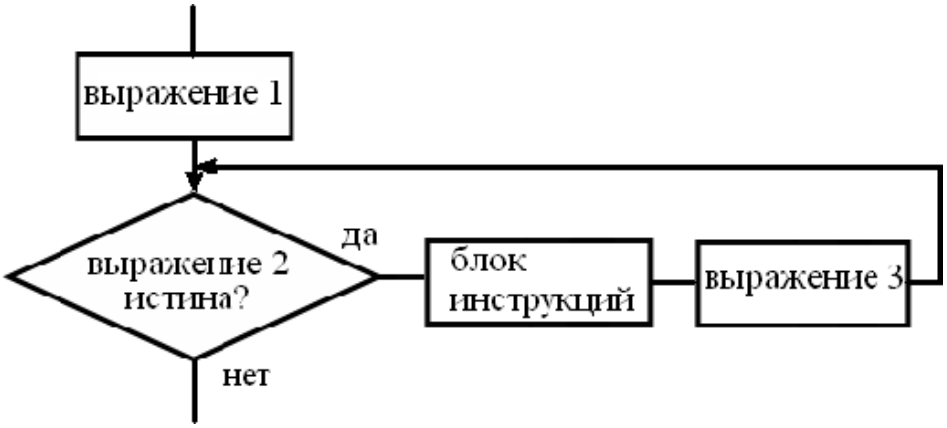
Имеется железнодорожное расписание, содержащее номер рейса поезда, времена отправления и прибытия на станцию прибытия. Организовать поиск номера поезда, время отправления и прибытия, если задана станция.

Практическая работа №15. Построение алгоритма заполнения двумерного массива

1. Цель работы:
Получение навыков построения алгоритмов заполнения двумерного массива.

2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схемы алгоритма с использованием вложенного цикла.

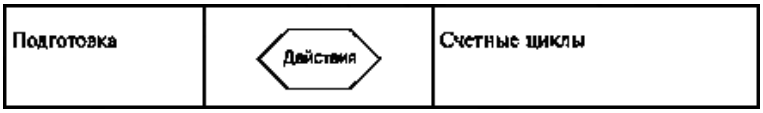
3. Теоретический материал
Цикл - это алгоритмическая конструкция, в которой в зависимости от условия повторяется определённая последовательность действий.
Цикл в алгоритме имеет особое значение, т.к. только его использование позволяет с помощью сравнительно коротких алгоритмов записывать длинные последовательности действий, что позволяет значительно уменьшить скорость выполнения программы.
Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.
Для организации циклов можно применять условия, т.к. циклический алгоритм является частным случаем разветвляющегося.
Тело цикла - это повторяющаяся последовательность действий (блок инструкций).
Цикл называется ***арифметическим***, если число повторений цикла известно заранее или может быть вычислено.
Цикл называется арифметическим, если число повторений цикла известно заранее или может быть вычислено.
Выражение 1 выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла). Выражение 2 — это условие выполнения цикла. Выражение 3 обычно определяет изменение параметра цикла. Блок инструкций — тело цикла, то есть инструкции, которые должны выполняться заданное количество раз..



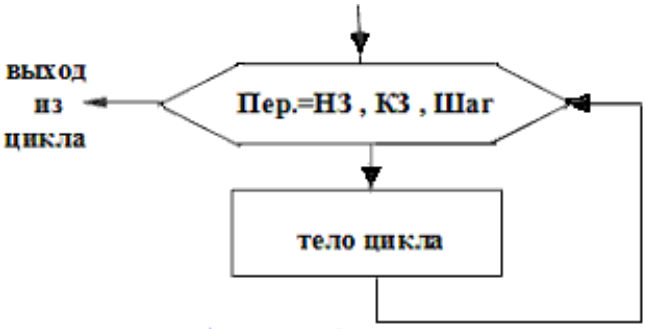
Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.

Цикл с параметрами иначе называется как ***цикл для каждого***.

Есть специальный элемент блок-схем для счетных циклов.



С использованием специального элемента алгоритм ***цикла для каждого*** выглядит так:



Пер. - переменная (параметр)
НЗ - начальное значение параметра
КЗ - конечное значение параметра
Шаг - величина изменения параметра после каждого выполнения тела цикла.

Тело цикла выполняется столько раз, сколько разных значений примет параметр в заданных пределах.

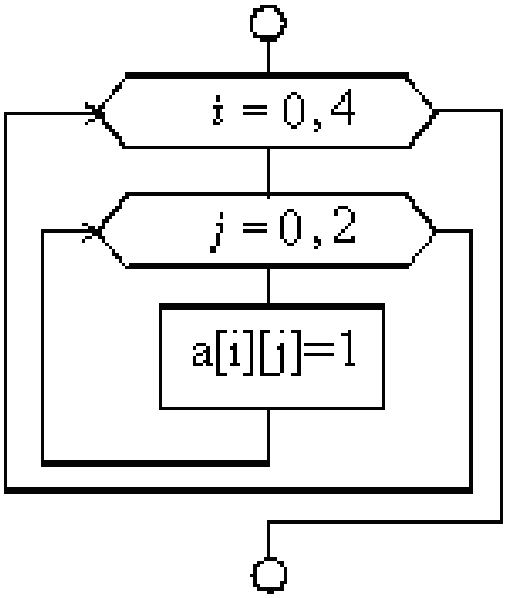
Возможны случаи, когда внутри тела цикла необходимо повторять некоторую последовательность операторов, т. е. организовать внутренний цикл. Такая структура получила название **цикла в цикле** или **вложенных циклов**. Глубина вложения циклов (то есть количество вложенных друг в друга циклов) может быть различной.
При использовании такой структуры для экономии машинного времени необходимо выносить из внутреннего цикла во внешний все операторы, которые не зависят от параметра внутреннего цикла.

4. Пример

1. Заполнить двумерный массив из пяти строк и трех столбцов единицами.

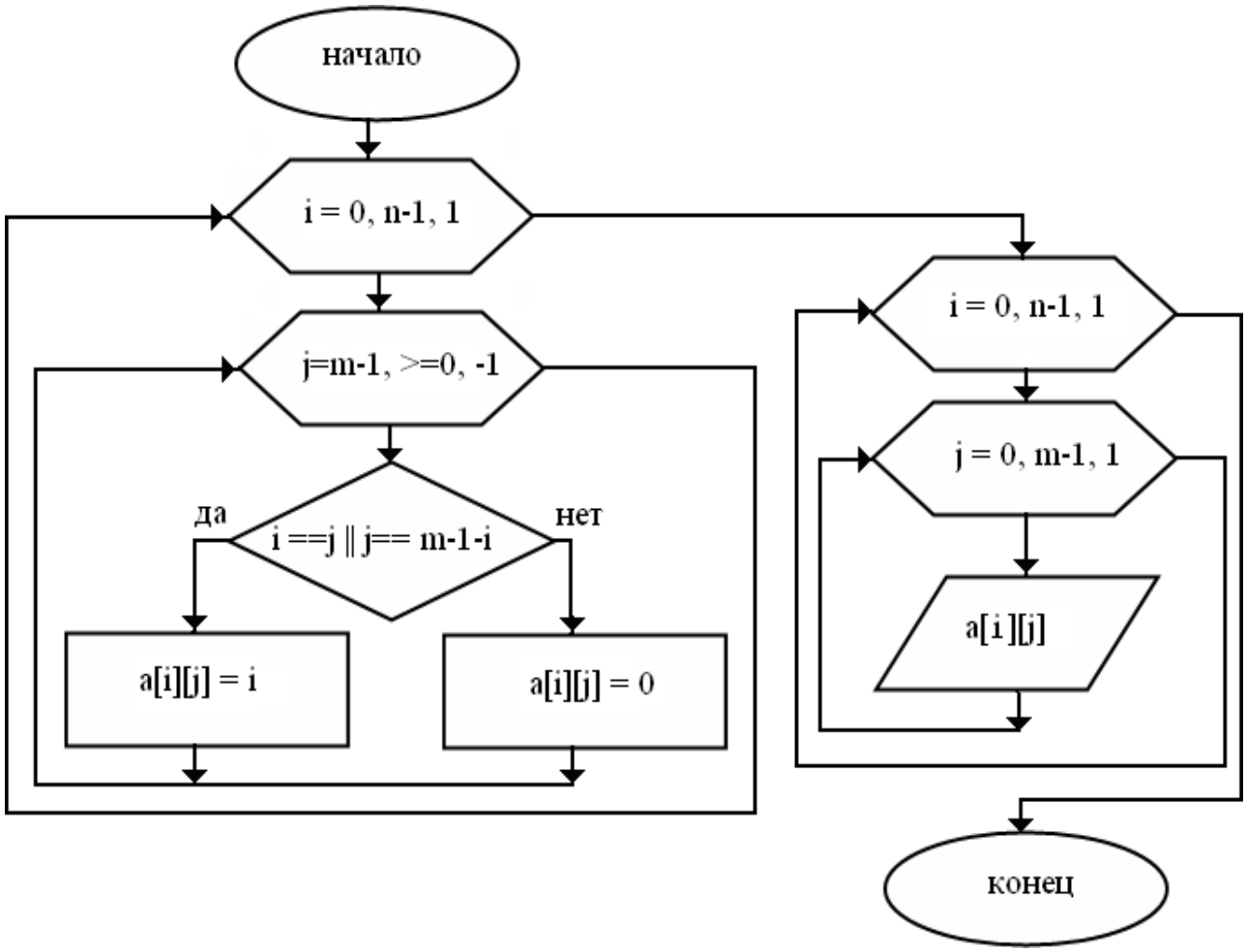
	0	1	2
0	1	1	1
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1

Для этого нужны два цикла. Внешний – по строкам, параметр i, изменяется от 0 до 4 с шагом 1, внутренний – по столбцам, параметр j, изменяется от 0 до 2 с шагом 1. Можно организовать обработку и наоборот.



2. Заполнить двумерный массив из 10 строк и 10 столбцов следующим образом:

0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 1 0
0 0 2 0 0 0 0 2 0 0
0 0 0 3 0 0 3 0 0 0
0 0 0 0 4 4 0 0 0 0
0 0 0 0 5 5 0 0 0 0
0 0 0 6 0 0 6 0 0 0
0 0 7 0 0 0 0 7 0 0
0 8 0 0 0 0 0 0 8 0
9 0 0 0 0 0 0 0 0 9



5.

Задание

1) Заполнить двумерный массив m[5][5] следующим образом: позиции, где оба индекса четные поставить 5, в остальные 1.

2) Заполнить двумерный массив m[7][7] произведениями индексов строки и столбца.

3) Заполнить двумерный массив m[4][4] следующим образом:

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

Дополнительные задания

1) Сформировать двумерный массив по следующему правилу

а).

0	0	0	0	0
0	1	0	0	0
0	0	2	0	0
0	0	0	3	0
0	0	0	0	4

б).

1	1	1	1	1
2	3	1	1	1
4	5	6	1	1
7	8	9	10	1
11	12	13	14	15

в).

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

2) Заполнить двумерный массив m[7][7] по спирали следующим образом:

1	2	3	4	5	6	7
24	25	26	27	28	29	8
23	40	41	42	43	30	9
22	39	48	49	44	31	10
21	38	47	46	45	32	11
20	37	36	35	34	33	12
19	18	17	16	15	14	13

3) Заполнить двумерный массив m[5][5] по спирали следующим образом:

25	24	23	22	21
10	9	8	7	20
11	2	1	6	19
12	3	4	5	18
13	14	15	16	17

4) Заполнить матрицу заданным образом:
0 1 1 1 1 1 0
1 0 1 1 1 0 1
1 1 0 1 0 1 1
1 1 1 0 1 1 1
1 1 0 1 0 1 1
1 0 1 1 1 0 1
0 1 1 1 1 1 0
5) Заполнить матрицу заданным образом:
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
6)Заполнить матрицу заданным образом:
0 0 0 0 0 0 0 0 0 9
0 0 0 0 0 0 0 0 8 0
0 0 0 0 0 0 0 7 0 0
0 0 0 0 0 0 6 0 0 0
0 0 0 0 0 5 0 0 0 0
0 0 0 0 4 0 0 0 0 0
0 0 0 3 0 0 0 0 0 0
0 0 2 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
7) Заполнить матрицу заданным образом (песочные часы):
1 0 0 0 0 0 1

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

8) Составить программу, которая заполняет квадратную матрицу порядка n натуральными числами 1, 2, 3, ..., n2, записывая их в нее «по спирали».

Например, для n = 5 получаем следующую матрицу:

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

9) Заполнить матрицу заданным образом:

1 1 1 1 1 1 1
3 1 1 1 1 1 4
3 3 1 1 1 4 4
3 3 3 2 4 4 4
3 3 2 2 2 4 4
3 2 2 2 2 2 4
2 2 2 2 2 2 2

Практическая работа №16. Построение алгоритма обработки двумерного массива

1. Цель работы:

Получение навыков построения алгоритмов обработки двумерного массива.

2. Темы для предварительной проработки

- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схемы алгоритма с использованием вложенного цикла.

3. Теоретический материал

Цикл - это алгоритмическая конструкция, в которой в зависимости от условия повторяется определённая последовательность действий.

Цикл в алгоритме имеет особое значение, т.к. только его использование позволяет с помощью сравнительно коротких алгоритмов записывать длинные последовательности действий, что позволяет значительно уменьшить скорость выполнения программы.

Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.

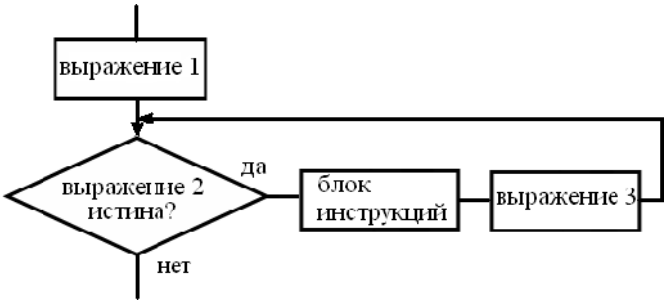
Для организации циклов можно применять условия, т.к. циклический алгоритм является частным случаем разветвляющегося.

Тело цикла - это повторяющаяся последовательность действий (блок инструкций).

Цикл называется ***арифметическим***, если число повторений цикла известно заранее или может быть вычислено.

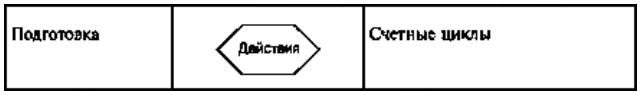
Цикл называется арифметическим, если число повторений цикла известно заранее или

может быть вычислено.
Выражение 1 выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла). Выражение 2 — это условие выполнения цикла. Выражение 3 обычно определяет изменение параметра цикла. Блок инструкций — тело цикла, то есть инструкции, которые должны выполняться заданное количество раз..

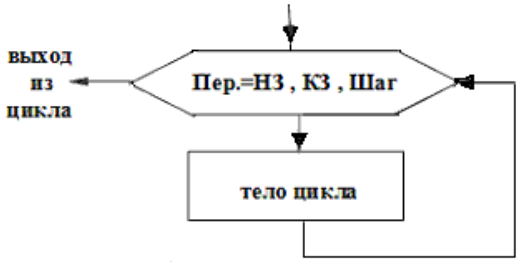


Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.

Цикл с параметрами иначе называется как ***цикл для каждого***.
Есть специальный элемент блок-схем для счетных циклов.

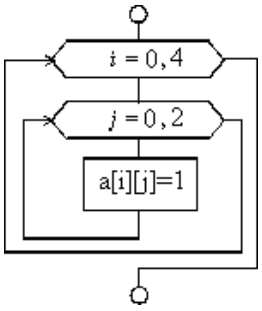


С использованием специального элемента алгоритм ***цикла для каждого*** выглядит так:



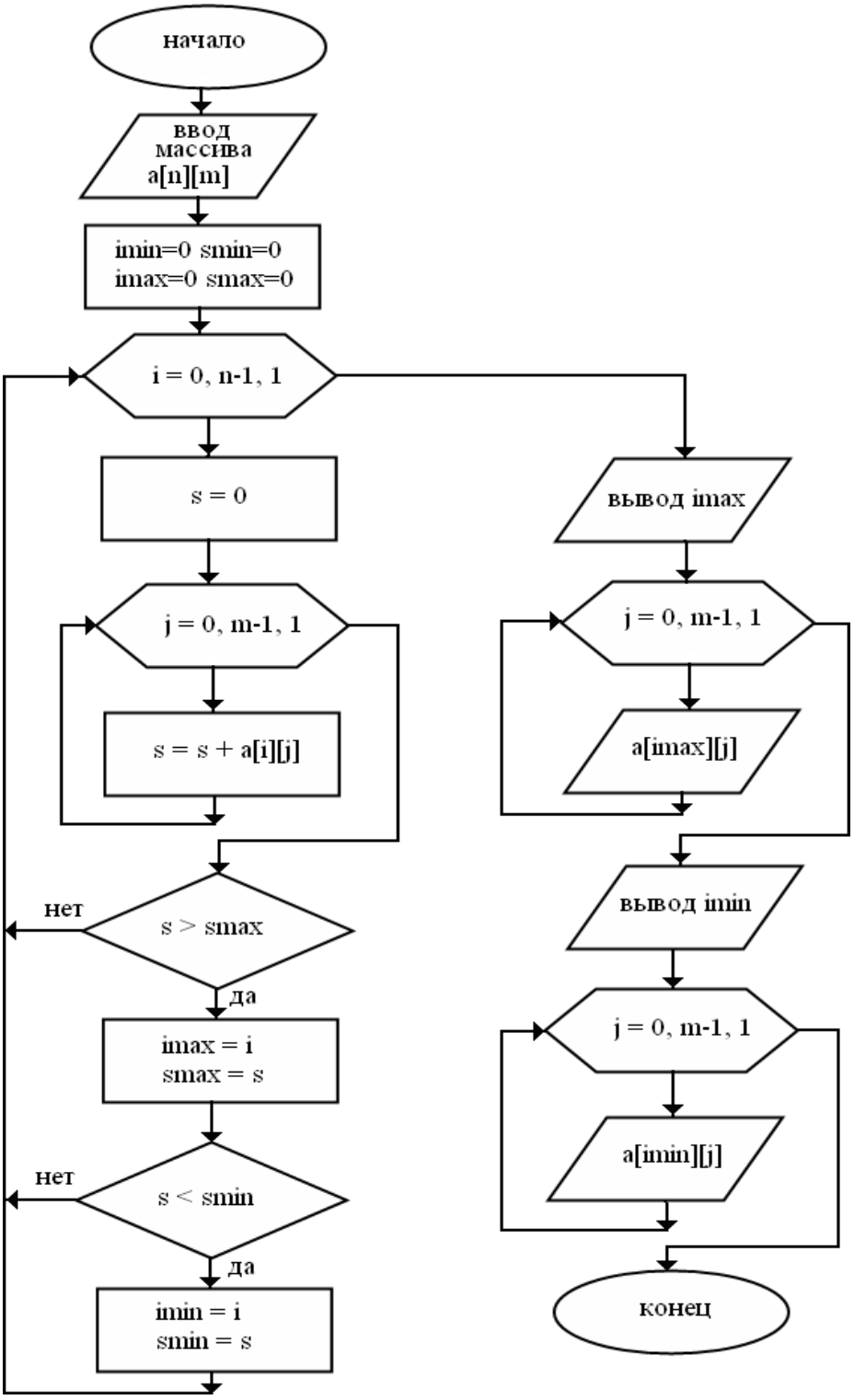
Пер. - переменная (параметр)
НЗ - начальное значение параметра
КЗ - конечное значение параметра
Шаг - величина изменения параметра после каждого выполнения тела цикла.

Тело цикла выполняется столько раз, сколько разных значений примет параметр в заданных пределах.
Возможны случаи, когда внутри тела цикла необходимо повторять некоторую последовательность операторов, т. е. организовать внутренний цикл. Такая структура получила название**цикла в цикле** или **вложенных циклов**. Глубина вложения циклов (то есть количество вложенных друг в друга циклов) может быть различной.
При использовании такой структуры для экономии машинного времени необходимо выносить из внутреннего цикла во внешний все операторы, которые не зависят от параметра внутреннего цикла.
Например, заполнить матрицу из 5 строк и 3 столбцов единицами.



4. Пример

Дан двумерный массив. Вывести на экран строки с максимальной и минимальной суммой, а также индексы этих строк.



5. Задание

1) Вычислить суммы элементов столбцов заданной матрицы A(N, M).
2) Подсчитать, сколько раз встречается в заданной целочисленной матрице A(N, M) максимальное по величине число.
3) Определить, имеется ли среди элементов главной диагонали заданной целочисленной матрицы A(N, N) хотя бы один положительный нечётный элемент.

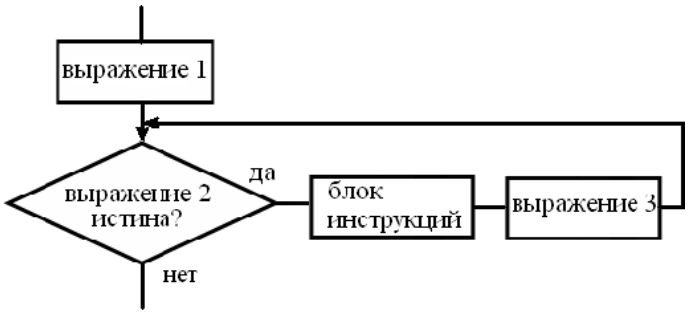
Дополнительные задания
1) Проверить, является ли заданная целочисленная матрица A(N, N) "магическим квадратом" (это значит, что суммы чисел во всех её строках, всех столбцах и двух диагоналях одинаковы).
2) Даны сведения о количестве забитых голов каждого футболиста команды в каждом из матчей чемпионата. Проверить, сколько в команде есть футболистов, забивавших голы в каждом матче.
3) Подсчитать количество различных (не повторяющихся) чисел, встречающихся в заданной целочисленной матрице $A(N, M)$.

Практическая работа №17. Построение алгоритма преобразования двумерного массива по заданному закону

1. Цель работы:
Получение навыков построения алгоритмов преобразования двумерного массива.

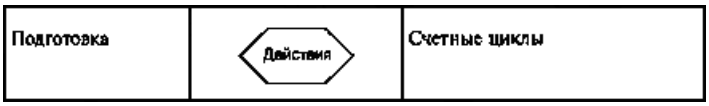
2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схемы алгоритма с использованием вложенного цикла.

3. Теоретический материал
Цикл - это алгоритмическая конструкция, в которой в зависимости от условия повторяется определённая последовательность действий.
Цикл в алгоритме имеет особое значение, т.к. только его использование позволяет с помощью сравнительно коротких алгоритмов записывать длинные последовательности действий, что позволяет значительно уменьшить скорость выполнения программы.
Алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными, называется циклическим.
Для организации циклов можно применять условия, т.к. циклический алгоритм является частным случаем разветвляющегося.
Тело цикла - это повторяющаяся последовательность действий (блок инструкций).
Цикл называется ***арифметическим***, если число повторений цикла известно заранее или может быть вычислено.
Цикл называется арифметическим, если число повторений цикла известно заранее или может быть вычислено.
Выражение 1 выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла). Выражение 2 — это условие выполнения цикла. Выражение 3 обычно определяет изменение параметра цикла. Блок инструкций — тело цикла, то есть инструкции, которые должны выполняться заданное количество раз..

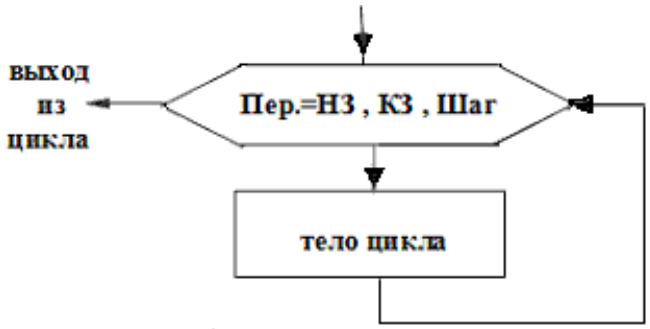


Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.

Цикл с параметрами иначе называется как ***цикл для каждого***.
Есть специальный элемент блок-схем для счетных циклов.



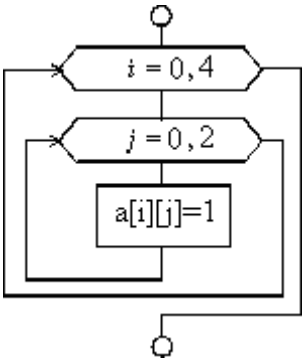
С использованием специального элемента алгоритм ***цикла для каждого*** выглядит так:



Пер. - переменная (параметр)
НЗ - начальное значение параметра
КЗ - конечное значение параметра
Шаг - величина изменения параметра после каждого выполнения тела цикла.

Тело цикла выполняется столько раз, сколько разных значений примет параметр в заданных пределах.

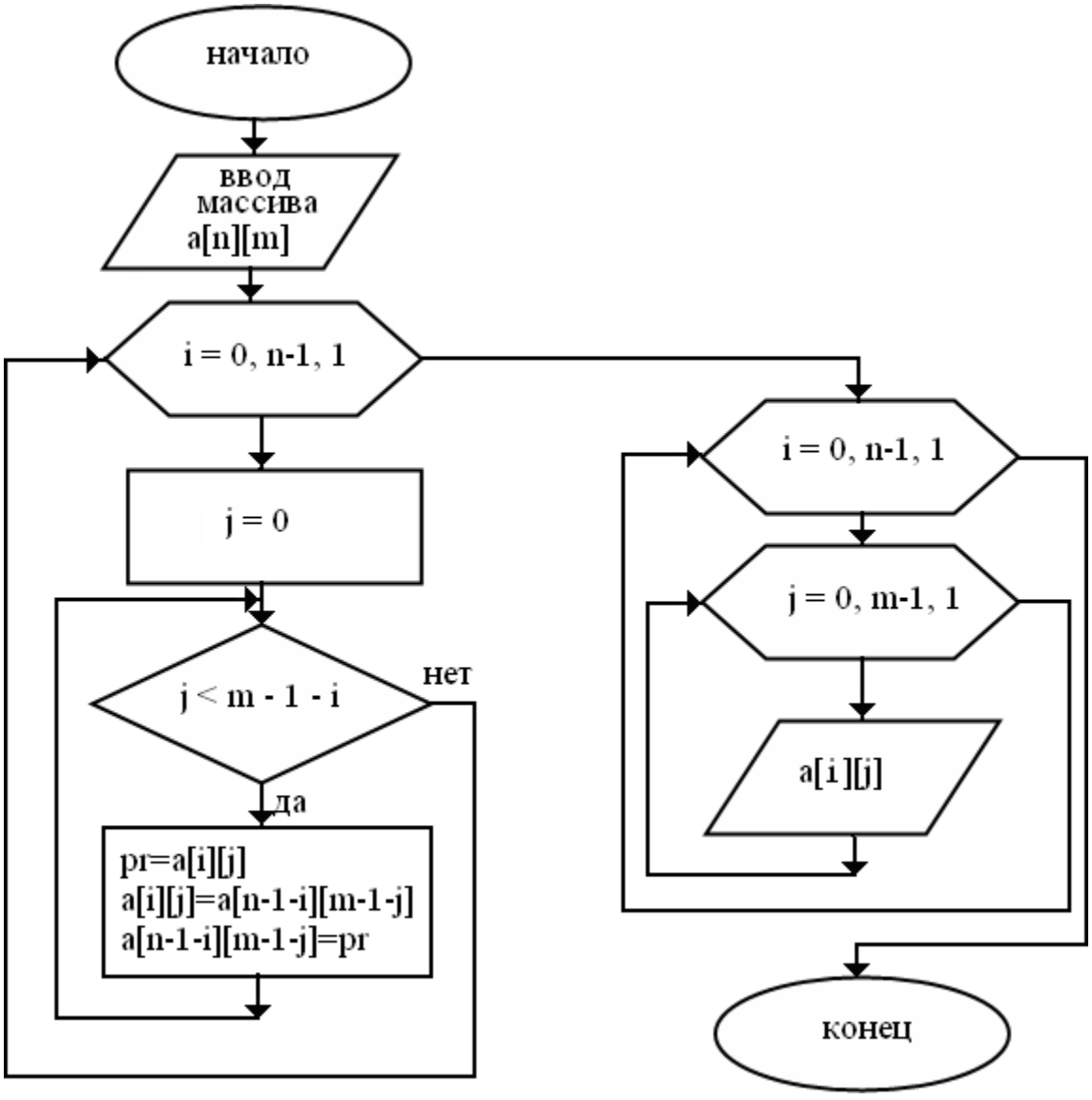
Возможны случаи, когда внутри тела цикла необходимо повторять некоторую последовательность операторов, т. е. организовать внутренний цикл. Такая структура получила название **цикла в цикле** или **вложенных циклов**. Глубина вложения циклов (то есть количество вложенных друг в друга циклов) может быть различной.
При использовании такой структуры для экономии машинного времени необходимо выносить из внутреннего цикла во внешний все операторы, которые не зависят от параметра внутреннего цикла.
Например, заполнить матрицу из 5 строк и 3 столбцов единицами.



4. Пример

Отразить заданный двумерный массив с равным числом строк и столбцов относительно побочной диагонали.

1 1 1 1 0 2 2 2 2 0
1 1 1 0 2 2 2 2 0 1
1 1 0 2 2 =>2 2 0 1 1
1 0 2 2 2 2 0 1 1 1
0 2 2 2 2 0 1 1 1 1



5. Задание

1) Найти наибольший и наименьший элементы прямоугольной матрицы и поменять их местами.
2) Дана действительная квадратная матрица порядка n. Преобразовать матрицу по следующему правилу: строку с номером n сделать столбцом с номером n, а столбец с номером n — строкой с номером n.
3) Пусть дана действительная матрица размером a x b. Требуется преобразовать матрицу следующим образом: поэлементно вычесть последнюю строку из всех строк, кроме последней.

Дополнительные задания
Сгенерировать и вывести на экран массив 10x10 из единиц и двоек. Определить и вывести массив B как одно из геометрических преобразований массива A:

а) разворот на 90 градусов по часовой стрелке;
б) разворот на 90 градусов против часовой стрелки;
в) разворот на 180 градусов;
г) зеркальное отражение по горизонтали;
д) зеркальное отражение по вертикали;
е) зеркальное отражение по главной диагонали;

Практическая работа №18. Построение алгоритма для решения рекуррентных соотношений (экологическая задача).

1. Цель работы:
Получение навыков построения алгоритмов для решения рекуррентных соотношений.

2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Правила построения алгоритмов в виде блок-схем.
- Построение блок-схемы алгоритма с использованием ветвления, цикла.

3. Теоретический материал
В век Фибоначчи возрождение было еще далеко, однако история даровала Италии краткий промежуток времени, который вполне можно было назвать репетицией надвигающейся эпохи Ренессанса.
Наибольший интерес представляет для нас сочинение "Книга абака".
Эта книга представляет собой объемный труд, содержащий почти все арифметические и алгебраические сведения того времени и сыгравший значительную роль в развитии математики в Западной Европе в течении нескольких следующих столетий. В частности, именно по этой книге европейцы познакомились с индусскими ("арабскими") цифрами.
Сообщаемый в "Книге абака" материал поясняется на большом числе задач, составляющих значительную часть этого тракта.
На стр. 123-124 данной рукописи, Фибоначчи поместил следующую задачу о кроликах.
Некто поместил пару кроликов в некоем месте, огороженном со всех сторон стеной, чтобы узнать, сколько пар кроликов родится при этом в течение года, если природа кроликов такова, что через месяц пара кроликов производит на свет другую пару, а рождают кролики со второго месяца после своего рождения.

Ясно, что если считать первую пару кроликов новорожденными, то на второй месяц мы будем по прежнему иметь одну пару; на 3-й месяц – 1 + 1 = 2; на 4-й – 2 + 1 = 3 пары (ибо из двух имеющихся пар потомство дает лишь одна пара); на 5-й месяц – 3 + 2 = 5 пар (лишь 2 родившиеся на 3-й месяц пары дадут потомство на 5-й месяц); на 6-й месяц – 5 + 3 = 8 пар (ибо потомство дадут только те пары, которые родились на 4-м месяце) и т. д. Т.о., если обозначить число пар кроликов, имеющихся на n-м месяце через Fk , то F1 = 1, F2 = 1, F3 = 2, F4 = 3, F5 = 5, F6 = 8, F7 = 13, F8 = 21 и т. д., причем образование этих чисел регулируется общим законом: Fn = Fn – 1 + Fn – 2 при всех n > 2, ведь число пар кроликов на n-м месяце равно числу Fn – 1 пар кроликов на предшествующем месяце плюс число вновь родившихся пар, которое совпадает с числом Fn – 2 пар кроликов, родившихся на (n – 2)-м месяце (ибо лишь эти пары кроликов дают потомство).
Решение задачи

Месяцы	1	2	3	4	5	6	7	8	9	…
Пары кроликов	1	1	2	3	5	8	13	21	34	….

Французский математик Люка впервые назвал числовую последовательность
1, 1, 2, 3, 5, 8, 13…
числами Фибоначчи
и открыл не менее фундаментальную последовательность
2, 1, 3, 4, 7, 11…,
которая тоже связана с золотой пропорцией. Отношение соседних чисел Люка по мере удаления от начала последовательности в пределе стремится к золотой пропорции.

Последовательности чисел Фибоначчи F(n) = 1, 1, 2, 3, 5, 8, 13,… и чисел Люка: L(n) = 2, 1, 3, 4, 7, 11,… ученые все чаще встречают во многих явлениях окружающего мира.

Рекуррентными называются соотношения, в которых очередной член последовательности выражен через один или несколько предыдущих. (от латинского "recurrere" – возвращаться").

Пример:
формулы для нахождения n-го элемента
$a_1 = 6$, $a_i = a_{i-1} + 3$ 6, 9, 12, 15, 18, 21, 24, 27, 31
$a_1 = 1$, $a_i = a_{i-1} + 10$ 1, 11, 21, 31, 41, 51, 61, 71,81
$a_1 = 3$, $a_i = a_{i-1} * 2 – 1$ 5, 9, 17, 33, 65, 129, 257, 513

4. Пример
ЭКОЛОГИЧЕСКАЯ ЗАДАЧА

В результате сброса промышленных стоков возрос уровень загрязнения реки. Каким он будет через сутки, двое и т.д. и когда уровень загрязнения станет допустимым, если известно, что за сутки он уменьшается в определенное количество раз?

Обозначим:

C_0 – начальная концентрация вредной примеси.
$C_{доп}$ – предельно допустимая концентрация вредной примеси.
К – коэффициент суточного уменьшения концентрации вредного вещества.

Вещество	C_0 (мг/л)	$C_{доп}$ (мг/л)	К
Свинец	5	0.03	1.12
Мышьяк	1.5	0.05	1.05
Фтор	0.2	0.05	1.01

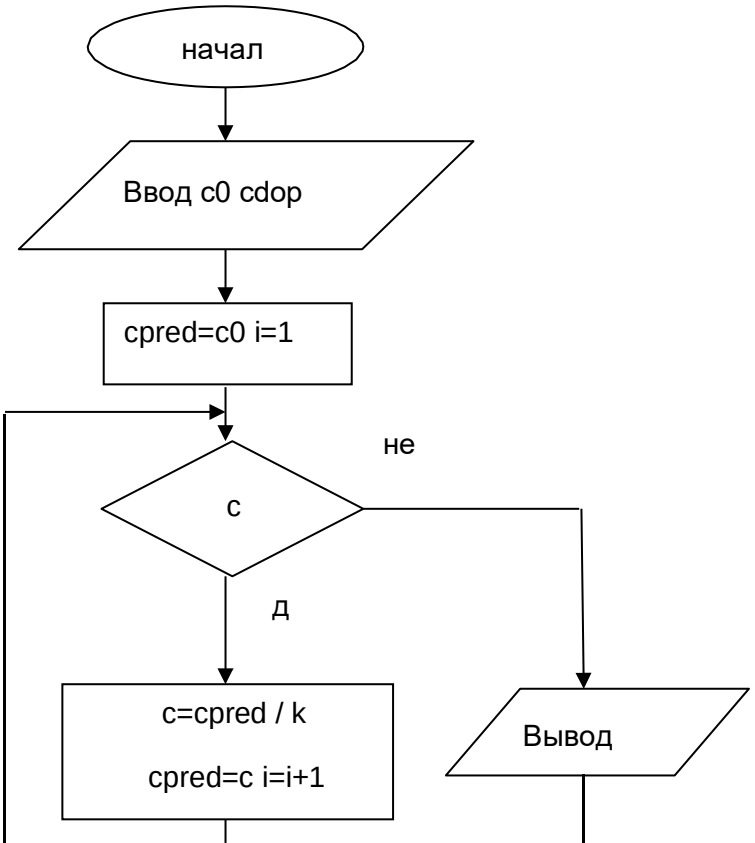
Математическая формулировка задачи:
$C_1 = C_0 / K$
$C_2 = C_1 / K = C_0 / K^2$
$C_3 = C_2 / K = C_1 / K^3$
……
$C_n = C_{n-1} / K = C_0 / K^n$ или
$C_n = C_0 / K^n$ или
$C_n = C_{n-1} / K$

Анализируем последнюю формулу – состояние загрязнения реки в текущий день зависит от состояния загрязнения в предыдущий день, т.е. имеем рекуррентное соотношение.

Для создания алгоритма представим рекуррентную формулу в виде:

c=cpred / k.

Обозначим предельно допустимое значение концентрации вредной примеси cdop, начальное значение концентрации c0 , коэффициент уменьшения загрязнения в сутки k, i – число суток.
Блок-схема решения задачи



5. Задание
Изменить алгоритм так, чтобы расчет производился с помощью трех видов циклов (с параметром, с предусловием, с постусловием), а данные могли вводиться пользователем.

Изменить алгоритм так, чтобы пользователь мог получить информацию в следующих случаях:
1. Нерадивый хозяйственник не имеет очистных сооружений. Известно, что комиссия по проверке уровня загрязнения воды приедет через 10 дней. Какой степени загрязнения он может сделать выброс?
2. Начиная с 10-го дня применен специальный состав, снижающий уровень загрязнения. После применения этого состава коэффициент суточного снижения загрязнения по свинцу 1.35, по мышьяку 1.25, по фтору 1.3

Дополнительные задания
Задан линейный массив из n элементов. Составьте алгоритм для решения задачи, предварительно составив рекуррентное соотношение.

1. Найти сумму, слагаемыми которой являются попарные произведения соседних элементов массива. (первый-второй, третий-четвертый..., если в массиве нечетное количество элементов, то считаем, что последний элемент массива умножается на единицу).
2. Найти сумму, слагаемыми которой являются попарные произведения соседних элементов массива. (первый-второй, второй-третий...).
3. Найти сумму, слагаемыми которой являются произведения трех соседних элементов массива. (первый-второй-третий, второй-третий-четвертый...).
4. Найти максимальное значение среди попарных произведений соседних элементов массива. (первый-второй, второй-третий...).
5. Найти максимальное значение среди произведений трех соседних элементов массива. (первый-второй-третий, второй-третий-четвертый...).
6. Найти максимальное значение среди минимальных из двух соседних элементов массива. (первый-второй, второй-третий...).
7. Найти минимальное значение среди максимальных из трех соседних элементов массива. (первый-второй-третий, второй-третий-четвертый...).
8. Найти минимальное значение среди максимальных и максимальное значение среди минимальных из k соседних элементов массива.

Практическая работа №19. Определение сложности алгоритмов обработки циклов, вложенных циклов.

1. Цель работы:
Получение навыков определения сложности алгоритмов обработки циклов.

2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Способы записи алгоритмов.

3. Теоретический материал
Сложность вычислений
Что такое сложность вычислений?
Центральная задача теории алгоритмов — выяснить, существует ли алгоритм решения той или иной задачи. Если да, то возникает следующий вопрос: а можно ли им воспользоваться на практике, при современном уровне развития вычислительной техники? То есть способен ли компьютер за приемлемое время получить результат? Например, в игре в шахматы возможно лишь конечное количество позиций и, значит, только конечное количество различных партий. Значит, теоретически можно перебрать все возможные партии и выяснить, кто побеждает при правильной игре — белые или черные. Однако количество вариантов настолько велико, что современные компьютеры не могут выполнить такой перебор за приемлемое время.

Что мы хотим от алгоритма? Во-первых, чтобы он работал как можно быстрее. Во-вторых, чтобы
объем необходимой памяти был как можно меньше. В-третьих, чтобы он был как можно более прост и понятен, что позволяет легче отлаживать программу. К сожалению, эти требования противоречивы, и в серьезных задачах редко удается найти алгоритм, который был бы лучше остальных по всем показателям.

Часто говорят о *временной сложности* алгоритма (быстродействии) и *пространственной сложности*, которая определяется объемом необходимой памяти. Поскольку память постоянно дешевеет, а быстродействие компьютеров растет медленно, мы будем рассматривать главным образом временную сложность — время выполнения программы, работающей по данному алгоритму.
В общем случае, говоря о сложности алгоритма, нужно уточнить, о каком исполнителе идет речь, какие элементарные операции мы используем. Как правило, это один из универсальных исполнителей (во многих случаях универсальным исполнителем можно считать компьютер). Временем работы алгоритма называется количество выполненных им элементарных операций T. Такой подход позволяет оценивать именно качество алгоритма, а не свойства исполнителя (например, быстродействие компьютера, на котором выполняется алгоритм).
Как правило, величина T будет существенно зависеть от объема исходных данных: поиск в списке из 10 элементов завершится гораздо быстрее, чем в списке из 10 000 элементов. Поэтому сложность алгоритма обычно связывают с размером входных данных n и определяют как функцию $T(n)$. Например, для алгоритмов обработки массивов в качестве размера n используют длину массива. Функция $T(n)$ называется

временной сложностью алгоритма.

Примеры
Рассмотрим алгоритмы выполнения различных операций с массивом **A** длины *n*, который может быть объявлен в программе на алгоритмическом языке как
цел A[1:n]

Пример 1. Вычислить сумму первых трех элементов массива (при $n \geq 3$).
Решение этой задачи содержит всего один оператор:
S := A[1] + A[2] + A[3]
Этот алгоритм включает две операции сложения и одну операцию записи значения в память, поэтому его сложность $T(n) = 3$ не зависит от размера массива вообще.

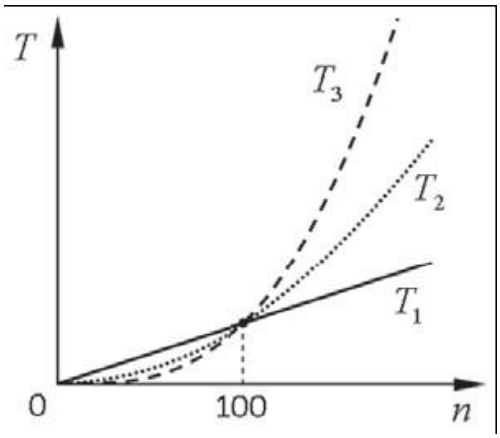
Пример 2. Вычислить сумму всех элементов массива.
В этой задаче уже не обойтись без цикла:
S := A[1]
нц для i от 2 до n
S := S + A[i]
кц
Здесь выполняется $n - 1$ операций сложения и n операций записи в память4, поэтому его сложность $T(n) = 2n - 1$ возрастает линейно с увеличением длины массива5.
Пример 3. Отсортировать все элементы массива по возрастанию методом выбора.
Напомним, что метод выбора предполагает поиск на каждом шаге минимального из оставшихся неупорядоченных значений (здесь **i**, **j**, **nMin** и **c** — целочисленные переменные):
нц для i от 1 до n-1
nMin:= i;
нцдля j от i+1 до n
если A[i] < A[nMin] то nMin:= i все
кц
если nMin <> i то
c:= A[i]; A[i]:= A[nMin]; A[nMin]:= c
все
кц
Подсчитаем отдельно количество сравнений и количество перестановок. Количество сравнений не зависит от данных и определяется числом шагов внутреннего цикла:

$$T_c(n) = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n .$$

Число перестановок зависит от данных. Например, если массив уже отсортирован в нужном порядке, перестановок не будет вообще. В худшем случае на каждом шаге основного цикла происходит перестановка, всего их будет $Tp(n) = n - 1$.

Что такое асимптотическая сложность?
Допустим, что нужно выбрать между несколькими алгоритмами, которые имеют разную сложность. Какой из них лучше (работает быстрее)? Оказывается, для этого

необходимо знать размер массива данных, которые нужно обрабатывать. Сравним, например, три алгоритма, сложность которых $T1(n) = 10\,000 \cdot n$, $T2(n) = 100 \cdot n2$ и $T3(n)$
$= n3$.
Построим эти зависимости на графике (см. *рис.* 15). При$n \le 100$ получаем$T3(n) < T2(n) < T1(n)$,
при $n = 100$ количество операций для всех трех алгоритмов совпадает, а при больших n имеем $T3(n) > T2(n) > T1(n)$.



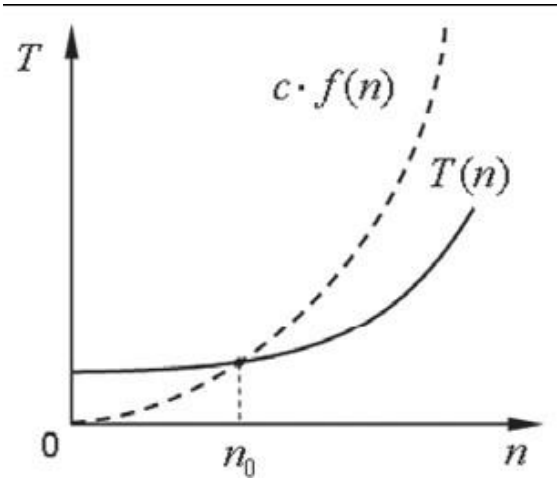
Обычно в теоретической информатике при сравнении алгоритмов используется их *асимптотическая сложность*, то есть скорость роста количества операций при больших значениях n. При этом запись $O(n)$ (читается “О большое от n”) обозначает, что, начиная с некоторого значения $n = n0$, количество операций ограничено функцией $c \cdot n$, где c — некоторая константа: $T(n) \le c \cdot n$ для $n \ge n0$.

Такие алгоритмы имеют *линейную сложность*, то есть при увеличении размера данных в 10 раз объем вычислений увеличивается тоже примерно в 10 раз.
Пусть, например, $T(n) = 2n - 1$, как в алгоритме поиска суммы элементов массива. Очевидно, что при этом $T(n) \le 2n$ для всех $n \ge 1$, поэтому алгоритм имеет линейную сложность.
Многие известные алгоритмы имеют *квадратичную сложность* $O(n2)$. Это значит, что сложность алгоритма ограничена функцией $c \cdot n2$: $T(n) \le c \cdot n2$ для $n \ge n0$.
При этом если размер данных увеличивается в 10 раз, то количество операций (и время выполнения) увеличивается примерно в 100 раз. Пример такого алгоритма — сортировка методом прямого выбора, для которой число сравнений

Алгоритм имеет **асимптотическую сложность** $O(f(n))$, если найдется такая постоянная c, что, начиная с некоторого $n = n0$, выполняется условие $T(n) \le c \cdot f(n)$.
Это значит, что график функции $c \cdot f(n)$ идет выше, чем график функции $T(n)$, по крайней мере при $n \ge n0$ (см. *рис.* 16).
Если количество операций не зависит от размера данных, то говорят, что сложность алгоритма $O(1)$, то есть количество операций меньше некоторой постоянной при любых n.

Существует также немало алгоритмов с кубичной сложностью, $O(n3)$. При больших значениях n алгоритм с кубичной сложностью требует большего количества вычислений, чем алгоритм со сложностью $O(n2)$, а тот, в свою очередь, работает

дольше, чем алгоритм с линейной сложностью.



Заметьте, что при малых значениях *n* все может быть наоборот, это зависит от постоянной *c* для каждого из алгоритмов.
Известны и алгоритмы, для которых количество операций растет быстрее, чем любой полином, например, как $O(2n)$ или $O(n!)$. Они встречаются чаще всего в задачах оптимизации, которые решаются только методом полного перебора. Самая известная задача такого типа — это *задача коммивояжера* (бродячего торговца), который должен посетить по одному разу каждый из указанных городов и вернуться в начальную точку. Для него нужно выбрать оптимальный маршрут, при котором стоимость поездки (или общая длина пути) будет минимальной.

Еще один пример сложной задачи, которая решается только полным перебором всех вариантов — *задача выполнимости*. Дано логическое выражение, которое содержит только имена логических переменных, скобки, а также операции “И”, “ИЛИ” и “НЕ”. Требуется определить, существует ли набор значений логических переменных, при котором заданное выражение истинно.

4. Пример

1). При сравнении различных алгоритмов важно знать, как их сложность зависит от объёма входных данных. Допустим, при сортировке одним методом обработка тысячи чисел занимает 1 с., а обработка миллиона чисел – 10 с., при использовании другого алгоритма может потребоваться 2 с. и 5 с. соответственно. В таких условиях нельзя однозначно сказать, какой алгоритм лучше.
В общем случае сложность алгоритма можно оценить по порядку величины. Алгоритм имеет сложность O(f(n)), если при увеличении размерности входных данных N, время выполнения алгоритма возрастает с той же скоростью, что и функция f(N). Рассмотрим код, который для матрицы A[NxN] находит максимальный элемент в каждой строке.

```
for (i=0; i<N; i++){
max:=A[i,0];
for (j=0; j<N; j++)
if A[i,j]>max max:=A[i,j];
cout<<max;
}
```

В этом алгоритме переменная i меняется от 0 до N. При каждом изменении i,

переменная j тоже меняется от 0 до N. Во время каждой из N итераций внешнего цикла, внутренний цикл тоже выполняется N раз. Общее количество итераций внутреннего цикла равно N*N. Это определяет сложность алгоритма O(N^2). Оценивая порядок сложности алгоритма, необходимо использовать только ту часть, которая возрастает быстрее всего. Предположим, что рабочий цикл описывается выражением N^3+N. В таком случае его сложность будет равна O(N^3). Рассмотрение быстро растущей части функции позволяет оценить поведение алгоритма при увеличении N. Например, при N=100, то разница между N^3+N=1000100 и N=1000000 равна всего лишь 100, что составляет 0,01%.
При вычислении O можно не учитывать постоянные множители в выражениях. Алгоритм с рабочим шагом 3N^3 рассматривается, как O(N^3). Это делает зависимость отношения O(N) от изменения размера задачи более очевидной.

2). Наиболее сложными частями программы обычно является выполнение циклов и вызов процедур. В предыдущем примере весь алгоритм выполнен с помощью двух циклов.
Если одна процедура вызывает другую, то необходимо более тщательно оценить сложность последней. Если в ней выполняется определённое число инструкций (например, вывод на печать), то на оценку сложности это практически не влияет. Если же в вызываемой процедуре выполняется O(N) шагов, то функция может значительно усложнить алгоритм. Если же процедура вызывается внутри цикла, то влияние может быть намного больше. В качестве примера рассмотрим две процедуры: Slow со сложностью O(N^3) и Fast со сложностью O(N^2).

```
void Slow(){
int i,j,k;
for (i=1; i<=N; i++)
for (j=1; j<=N; j++)
for (k=1; k<=N; k++)
//какое-то действие
}
voidFast(){
int i,j
for (i=1; i<=N; i++)
for (j=1; j<=N; j++)
Slow();
}
voidmain(){
Fast();
}
```

Если во внутренних циклах процедуры Fast происходит вызов процедуры Slow, то сложности процедур перемножаются. В данном случае сложность алгоритма составляет O(N^2)*O(N^3)=O(N^5).
Если же основная программа вызывает процедуры по очереди, то их сложности складываются: O(N^2)+O(N^3)=O(N^3). Следующий фрагмент имеет именно такую сложность:

```
voidSlow(){
inti,j,k;
```

```
for (i=1; i<=N; i++)
for (j=1; j<=N; j++)
for (k=1; k<=N; k++)
//какое-то действие
}
void Fast(){
int i,j;
for (i=1; i<=N; i++)
for (j=1; j<=N; j++)
//какое-то действие
}
voidmain(){
Fast();
Slow();
}
```

5. Задание

Оцените количество операций для алгоритмов:
а) поиска всех делителей числа;
б) нахождения минимального и максимального элементов массива;
в) определения количества положительных элементов массива;
г) проверки числа на простоту.
В каждом случае опишите набор используемых элементарных операций. Определите асимптотическую сложность этих алгоритмов.

Дополнительные задания

1. *Предложите алгоритм, позволяющий найти и вывести на экран те символы, которые встречаются в строке более одного раза. Оцените его асимптотическую сложность.

2. *Алфавит языка племени “тумба-юмба” содержит *k* символов. Предложите алгоритм построения всех возможных слов этого языка длиной *n* символов и оцените его асимптотическую сложность.

3. В массиве n целых чисел найти все пары элементов, сумма которых четна и сформировать новый массив из этих сумм.

4. Вычислить среднее значение элементов массива из n чисел.

5. Вычислить максимальное значение в массиве чисел.

6. В массиве n целых чисел найти все тройки элементов, в которых сумма двух элементов равна третьему, подсчитать количество таких троек.

7. В массиве n целых чисел найти все элементы, равные квадрату другого элемента

массива и составить массив из этих элементов.

Практическая работа №20. Определение сложности алгоритмов обработки массивов.

1. Цель работы:

Получение навыков определения сложности алгоритмов линейного и бинарного поиска в массиве.

2. Темы для предварительной проработки

- Общее понятие об алгоритме.
- Способы записи алгоритмов.

3. Теоретический материал

Общие функции оценки сложности

Перечислим некоторые функции, которые чаще всего используются для вычисления сложности. Функции перечислены в порядке возрастания сложности. Чем выше в этом списке находится функция, тем быстрее будет выполняться алгоритм с такой оценкой.

1. C – константа
2. log(log(N))
3. log(N)
4. N^C, 0<C<1
5. N
6. N*log(N)
7. N^C, C>1
8. C^N, C>1
9. N!

Если мы хотим оценить сложность алгоритма, уравнение сложности которого содержит несколько этих функций, то уравнение можно сократить до функции, расположенной ниже в таблице. Например, O(log(N)+N!)=O(N!).

Если алгоритм вызывается редко и для небольших объёмов данных, то приемлемой можно считать сложность O(N^2), если же алгоритм работает в реальном времени, то не всегда достаточно производительности O(N).

Обычно алгоритмы со сложностью N*log(N) работают с хорошей скоростью. Алгоритмы со сложностью N^C можно использовать только при небольших значениях C. Вычислительная сложность алгоритмов, порядок которых определяется функциями C^N и N! очень велика, поэтому такие алгоритмы могут использоваться только для обработки небольшого объёма данных.

В заключение приведём таблицу, которая показывает, как долго компьютер, осуществляющий миллион операций в секунду, будет выполнять некоторые медленные алгоритмы.

	N=10	N=20	N=30	N=40	N=50
N^3	0.001 с	0.008 с	0.027 с	0.064 с	0.125 с
2^N	0.001 с	1.05 с	17.9 мин	1. 29 дней	35.7 лет
3^N	0.059 с	58.1 мин	6.53 лет	$3.86 * 10^5$ лет	$2.28 * 10^{10}$ лет
$N!$	3.63 с	$7.71 * 10^4$ лет	$8.41 * 10^{18}$ лет	$2.59 * 10^{34}$ лет	$9.64 * 10^{50}$ лет

4. Пример

Оценка сложности алгоритма до порядка является верхней границей сложности алгоритмов. Если программа имеет большой порядок сложности, это вовсе не означает, что алгоритм будет выполняться действительно долго. На некоторых наборах данных выполнение алгоритма занимает намного меньше времени, чем можно предположить на основе их сложности.

Например, рассмотрим код, который ищет заданный элемент в векторе A.

```
int Locate(int data){
int i;
int fl;
fl=0; i=1;
while ((not fl) && (i<=N)){
if (A[i]==data)fl=1;
else i=i+1;
}
if (not fl) i=0;
returni;
}
```

Если искомый элемент находится в конце списка, то программе придётся выполнить N шагов. В таком случае сложность алгоритма составит O(N). В этом наихудшем случае время работы алгоритма будем максимальным.

С другой стороны, искомый элемент может находится в списке на первой позиции. Алгоритму придётся сделать всего один шаг. Такой случай называется наилучшим и его сложность можно оценить, как O(1).

Оба эти случая маловероятны. Нас больше всего интересует ожидаемый вариант. Если элемента списка изначально беспорядочно смешаны, то искомый элемент может оказаться в любом месте списка. В среднем потребуется сделать N/2 сравнений, чтобы найти требуемый элемент. Значит сложность этого алгоритма в среднем составляет O(N/2)=O(N).

В данном случае средняя и ожидаемая сложность совпадают, но для многих

алгоритмов наихудший случай сильно отличается от ожидаемого.

Давайте оценим алгоритм бинарного поиска в массиве - дихотомию.
Суть алгоритма: идем к середине массива и ищем соответствие ключа значению срединного элемента. Если нам не удается найти соответствия, мы смотрим на относительный размер ключа и значение срединного элемента и затем перемещаемся в нижнюю или верхнюю половину списка. В этой половине снова ищем середину и опять сравниваем с ключом. Если не получается, снова делим на половину текущий интервал.

```
int search(int low, int high, int key){
int mid, data;
while (low<=high){
mid=(low+high) % 2;
data=a[mid];
if (key==data)
return mid;
else {
if (key < data)
  high:=mid-1;
else
  low=mid+1;
}
return -1;
}
```

Решение:
Первая итерация цикла имеет дело со всем списком. Каждая последующая итерация делит пополам размер подсписка. Так, размерами списка для алгоритма являются
$n\ n/2^1\ n/2^2\ n/2^3\ n/2^4 \dots n/2^m$
В конце концов будет такое целое m, что $n/2^m<2$ или $n<2^{m+1}$
Так как m - это первое целое, для которого $n/2^m<2$, то должно быть верно
$n/2^{m-1}>=2$ или $2^m=<n$
Из этого следует, что
$2^m=<n<2^{m+1}$
Возьмем логарифм каждой части неравенства и получим
$m=<\log_2 n=x<m+1$
Значение m - это наибольшее целое, которое =<x.
Итак, $O(\log_2 n)$.
Обычно решаемая задача имеет естественный "размер" (обычно количество данных ею обрабатываемых) которое мы называем N. В конечном итоге нам бы хотелось получить выражение для времени, необходимого программе для обработки данных размера N, как функцию от N. Обычно на интересует средний случай - ожидаемое время работы программы на "типичных" входных данных, и худший случай - ожидаемое время работы программы на самых плохих входных данных.

5. Задание
Оцените сложность алгоритма определения является ли число палиндромом.

Дополнительные задания

Оцените сложность программы "Тройки Пифагора".

Практическая работа №21. Определение сложности алгоритмов сортировки.

1. Цель работы:

Получение навыков определения сложности алгоритмов сортировки.

2. Темы для предварительной проработки

- Общее понятие об алгоритме.
- Способы записи алгоритмов.

3. Теоретический материал

Алгоритм сортировки

Алгоритм сортировки — это алгоритм для упорядочения элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

Оценка алгоритма сортировки

Алгоритмы сортировки оцениваются по скорости выполнения и эффективности использования памяти:

- **Время** — основной параметр, характеризующий быстродействие алгоритма. Называется также вычислительной сложностью. Для упорядочения важны *худшее*, *среднее* и *лучшее* поведение алгоритма в терминах мощности входного множества *A*. Если на вход алгоритму подаётся множество *A*, то обозначим $n = |A|$. Для типичного алгоритма хорошее поведение — это $O(n \log n)$[1] и плохое поведение — это $O\left(n^2\right)$. Идеальное поведение для упорядочения — $O(n)$. Алгоритмы сортировки, использующие только абстрактную операцию сравнения ключей всегда нуждаются по меньшей мере в $\Omega(n \log n)$сравнениях. Тем не менее, существует алгоритм сортировки Хана (Yijie Han) с вычислительной сложностью $O(n \cdot \log\log n \cdot \log\log\log n)$, использующий тот факт, что пространство ключей ограничено (он чрезвычайно сложен, а за O-обозначением скрывается весьма большой коэффициент, что делает невозможным его применение в повседневной практике). Также существует понятие сортирующих сетей. Предполагая, что можно одновременно (например, при параллельном вычислении) проводить несколько сравнений, можно отсортировать *n* чисел за $O\left(\log^2 n\right)$операций. При этом число *n* должно быть заранее известно;
- **Память** — ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных. Как правило, эти алгоритмы требуют $O(\log n)$ памяти. При оценке не учитывается место, которое занимает исходный массив и независящие от входной последовательности затраты, например, на хранение кода программы (так как всё это потребляет $O(1)$). Алгоритмы сортировки, не потребляющие дополнительной памяти, относят к *сортировкам на месте*.

Классификация алгоритмов сортировки

- **Устойчивость** (stability) — устойчивая сортировка не меняет взаимного расположения равных элементов.
- **Естественность поведения** — эффективность метода при обработке уже упорядоченных, или частично упорядоченных данных. Алгоритм ведёт себя естественно, если учитывает эту характеристику входной последовательности и работает лучше.
- **Использование операции сравнения.** Алгоритмы, использующие для сортировки сравнение элементов между собой, называются основанными на сравнениях. Минимальная трудоемкость *худшего случая* для этих алгоритмов составляет $O(n \log n)$, но они отличаются гибкостью применения. Для специальных случаев (типов данных) существуют более эффективные алгоритмы.

Ещё одним важным свойством алгоритма является его сфера применения. Здесь основных типов упорядочения два:

- **Внутренняя сортировка** оперирует с массивами, целиком помещающимися в оперативной памяти с произвольным доступом к любой ячейке. Данные обычно упорядочиваются на том же месте, без дополнительных затрат.
  - В современных архитектурах персональных компьютеров широко применяется подкачка и кэширование памяти. Алгоритм сортировки должен хорошо сочетаться с применяемыми алгоритмами кэширования и подкачки.
- **Внешняя сортировка** оперирует с запоминающими устройствами большого объёма, но с доступом не произвольным, а последовательным (упорядочение файлов), т. е. в данный момент мы 'видим' только один элемент, а затраты на перемотку по сравнению с памятью неоправданно велики. Это накладывает некоторые дополнительные ограничения на алгоритм и приводит к специальным методам упорядочения, обычно использующим дополнительное дисковое пространство. Кроме того, доступ к данным на носителе производится намного медленнее, чем операции с оперативной памятью.
  - Доступ к носителю осуществляется последовательным образом: в каждый момент времени можно считать или записать только элемент, следующий за текущим.
  - Объём данных не позволяет им разместиться в ОЗУ.

Также алгоритмы классифицируются по:

- потребности в дополнительной памяти или её отсутствии
- потребности в знаниях о структуре данных, выходящих за рамки операции сравнения, или отсутствии таковой

Список алгоритмов сортировки

В этой таблице n — это количество записей, которые необходимо упорядочить, а k — это количество уникальных ключей.

Алгоритмы устойчивой сортировки

- Сортировка пузырьком (англ.*Bubblesort*) — сложность алгоритма: $O(n^2)$; для каждой пары индексов производится обмен, если элементы расположены не по порядку.
- Сортировка перемешиванием (Шейкерная, Cocktail sort, bidirectional bubble sort) — Сложность алгоритма: $O(n^2)$

- Гномья сортировка — имеет общее с сортировкой пузырьком и сортировкой вставками. Сложность алгоритма — O(n^2).
- Сортировка вставками (Insertion sort) — Сложность алгоритма: O(n^2); определяем где текущий элемент должен находиться в упорядоченном списке и вставляем его туда
- Блочная сортировка (Корзинная сортировка, Bucket sort) — Сложность алгоритма: O(n); требуется O(k) дополнительной памяти и знание о природе сортируемых данных, выходящее за рамки функций "переставить" и "сравнить".
- Сортировка подсчётом (Counting sort) — Сложность алгоритма: O(n+k); требуется O(n+k) дополнительной памяти (рассмотрено 3 варианта)
- Сортировка слиянием (Merge sort) — Сложность алгоритма: O(n log n); требуется O(n) дополнительной памяти; выстраиваем первую и вторую половину списка отдельно, а затем — сливаем упорядоченные списки
- Сортировка с помощью двоичного дерева (англ.*Treesort*) — Сложность алгоритма: O(n log n); требуется O(n) дополнительной памяти

Алгоритмы неустойчивой сортировки

- Сортировка выбором (Selection sort) — Сложность алгоритма: O(n^2); поиск наименьшего или наибольшего элемента и помещения его в начало или конец упорядоченного списка
- Сортировка Шелла (Shell sort) — Сложность алгоритма: O(n $\log^2 n$); попытка улучшить сортировку вставками
- Сортировка расчёской (Comb sort) — Сложность алгоритма: O(n log n)
- Пирамидальная сортировка (Сортировка кучи, Heapsort) — Сложность алгоритма: O(n log n); превращаем список в кучу, берём наибольший элемент и добавляем его в конец списка
- Плавная сортировка (Smoothsort) — Сложность алгоритма: O(n log n)
- Быстрая сортировка (Quicksort) — Сложность алгоритма: O(n log n) — среднее время, O(n^2) — худший случай; широко известен как быстрейший из известных для упорядочения больших случайных списков; с разбиением исходного набора данных на две половины так, что любой элемент первой половины упорядочен относительно любого элемента второй половины; затем алгоритм применяется рекурсивно к каждой половине
- Introsort — Сложность алгоритма: O(n log n), сочетание быстрой и пирамидальной сортировки. Пирамидальная сортировка применяется в случае, если глубина рекурсии превышает log(n).
- Patience sorting — Сложность алгоритма: O(n log n + k) — наихудший случай, требует дополнительно O(n + k) памяти, также находит самую длинную увеличивающуюся подпоследовательность
- Stooge sort — рекурсивный алгоритм сортировки с временной сложностью $O(n^{\log_{1,5} 3}) \approx O(n^{2.71})$.
- Поразрядная сортировка — Сложность алгоритма: O(n·k); требуется O(k) дополнительной памяти.
- Цифровая сортировка — то же, что и Поразрядная сортировка.

Непрактичные алгоритмы сортировки

- Bogosort — O(n·n!) всреднем. Произвольно перемешать массив, проверить порядок.

- Сортировка перестановкой — O($n \cdot n!$) — худшее время. Для каждой пары осуществляется проверка верного порядка и генерируются всевозможные перестановки исходного массива.
- Глупая сортировка (Stupid sort) — O(n^3); рекурсивная версия требует дополнительно O(n^2) памяти
- Bead Sort — O(n) or O($\sqrt{n}$), требуется специализированное аппаратное обеспечение
- Блинная сортировка (Pancake sorting) — O(n), требуется специализированное аппаратное обеспечение

Алгоритмы, не основанные на сравнениях

- Блочная сортировка (Корзинная сортировка, Bucket sort)
- Лексикографическая или поразрядная сортировка (Radix sort)
- Сортировка подсчётом (Counting sort)

Остальные алгоритмы сортировки

- Топологическая сортировка
- Внешняя сортировка

4. Пример

Сортировка простым включением. Предположим, что на некотором этапе работы алгоритма левая часть массива с 1-го по (i — 1)-й элемент включительно
является отсортированной, а правая часть с i-го по n-й элемент остается такой, какой она была в первоначальном, неотсортированном массиве. Очередной шаг алгоритма заключается в расширении левой части на один элемент и, соответственно, сокращении правой части. Для этого берется первый элемент правой части (с индексом i) и вставляется на подходящее ему место в левую часть так, чтобы упорядоченность левой части сохранилась.
Процесс начинается с левой части, состоящей из одного элемента A[1], а заканчивается, когда правая часть становится пустой.

```
Const n=1000;
Type tovar=Record
              name: String;
              key: Integer
            End;
Var A: array[1..n] Of tovar;

Procedure StraightInsertion;
Var i,j: Integer; x: tovar;
Begin  For i:=2 To n Do
Begin x:=A[i]; {В переменной x запоминается
значение, которое нужно поставить на свое место в
левой части}
      j:=i-1; {Правый край левой части}
      While (x.key<A[j].key) and (j>=1) Do
      Begin A[j+1]:=A[j]; j:=j-1; {Продвижение
"дырки" в левой части массива справа налево до той
позиции, в которую должен быть включен элемент
      A[i]}
      End;
      A[j+1]:=x {включение A[i] в "дырку" в левой
части}

End
End;
```

Оценим сложность алгоритма сортировки простым включением. Очевидно, что временная сложность зависит как от размера сортируемого массива, так и от его исходного состояния в смысле упорядоченности элементов. Временная сложность будет минимальной, если исходный массив уже отсортирован в нужном порядке значений ключа (в данном случае — по возрастанию). Максимальное значение сложности будет соответствовать противоположной упорядоченности исходного массива, т.е. упорядоченности исходного массива по убыванию значений ключа. Обычно для алгоритмов сортировки временная сложность оценивается количеством пересылок элементов.

Оценим величину минимальной временной сложности алгоритма. Если массив уже отсортирован, то тело цикла while не будет выполняться ни разу. Выполнение процедуры сведется к работе следующего цикла:

```
For i:=2 To n do
Begin x:=A[i];
      j:=i-1;
      A[j+1]:=x
End;
```

Поскольку тело цикла for исполняется n — 1 раз, то число пересылок элементов массива

Mmin = 2(n - 1),

а число сравнений ключей равно

Cmin = n - 1.

Сложность алгоритма будет максимальной, если исходный массив упорядочен по убыванию. Тогда каждый элемент A[i] будет «прогоняться» к началу массива, т.е. устанавливаться в первую позицию. Цикл while выполнится 1 раз при i = 2, 2 раза при i = 3 и т. д., п — 1 раз при i = п. Таким образом, общее число пересылок записей равно:

$$M_{max} = 2(n-1) + \sum_{i=2}^{n}(i-1) = 2(n-1) + n(n-1)/2 = \frac{1}{2}(n^2 + 3n - 4).$$

Более подходящей для реальной ситуации является средняя оценка сложности. Для ее вычисления надо предположить, что все элементы исходного массива — случайные числа и их значения никак не связаны с их номерами. В таком случае результат очередной проверки условия x. key<A[j].key в цикле While также является случайным. Разумно допустить, что среднее число выполнений цикла While для каждого конкретного значения i равно i/2, т. е. в среднем каждый раз приходится просматривать половину последовательности до тех пор, пока не найдется подходящее место для очередного элемента

Тогда формула для среднего числа пересылок (средняя оценка сложности) будет следующей:

$$M_{ср} = 2(n-1) + \sum_{i=2}^{n} i/2 = 2(n-1) + (n-2)(n-1)/4 = \frac{1}{4}(n^2 + 9n - 10).$$

Как максимальная, так и средняя оценка сложности алгоритма квадратична (является полиномом второй степени) по параметру n — размеру сортируемого массива.

5. Задание

Определить сложность алгоритма.

Упорядочить массив n целых чисел таким образом, чтобы элементы с четными и

нечетными значениями чередовались (пока имеются элементы разной четности).

Дополнительные задания

Разработать алгоритм быстрой сортировки. Оценить сложность алгоритма.
Этот алгоритм был разработан Э. Хоаром.
В алгоритме быстрой сортировки используются три идеи:
• разделение сортируемого массива на 2 части, левую и правую;
• взаимное упорядочение двух частей (подмассивов) так, чтобы все элементы левой части не превосходили элементов правой части;
• рекурсия, при которой подмассив упорядочивается точно таким же способом, как и весь массив.
Для разделения массива на две части нужно выбрать некоторое «барьерное» значение ключа. Это значение должно удовлетворять единственному условию: лежать в диапазоне значений для данного массива (т.е. между минимальной и максимальной величиной). За «барьер» можно выбрать значение ключа любого элемента массива, например первого, или последнего, или находящегося в середине.
Далее нужно сделать так, чтобы в левом подмассиве оказались все элементы с ключом, меньшим барьера, а в правом — с большим: Затем, просматривая массив слева направо, необходимо найти позицию первого элемента с ключом, большим барьера, а просматривая справа налево — найти первый элемент с ключом, меньшим барьера. Следует поменять эти значения, затем продолжить встречное движение до следующей пары элементов, предназначенных для обмена. Необходимо повторять эту процедуру, пока индексы левого и правого просмотров не совпадут. Место совпадения станет границей между двумя взаимно упорядоченными подмассивами. Далее алгоритм рекурсивно применяется к каждому из подмассивов (левому и правому). В конечном счете приходим к совокупности из п взаимно упорядоченных одноэлементных массивов, которые делить дальше невозможно. Эта совокупность образует один полностью упорядоченный массив. Сортировка завершена!

Практическая работа №22. Построение машины Тьюринга, вычисляющей простейшие арифметические функции

1. Цель работы:
Формирование представления учащихся об универсальном исполнителе алгоритмов.
Получение навыков построения программ для машины Тьюринга.

2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Способы записи алгоритмов.

3. Теоретический материал
В 1936 г. Аланом Тьюрингом для уточнения понятия алгоритма был предложен **абстрактный универсальный исполнитель**. Его абстрактность заключается в том, что он представляет собой логическую вычислительную конструкцию, а не реальную вычислительную машину. Термин «универсальный исполнитель» говорит о том, что данный исполнитель может имитировать любой другой исполнитель. Например, операции, которые выполняют реальные вычислительные машины можно

имитировать на универсальном исполнителе. В последствие, придуманная Тьюрингом вычислительная конструкция была названа **машиной Тьюринга**. Кроме того, предполагается, что универсальный исполнитель должен уметь доказывать существование или отсутствие алгоритма для той или иной задачи.

Что собой представляет машина Тьюринга?

Машина Тьюринга состоит из бесконечной в обе стороны ленты, разделенной на ячейки, и автомата (головки), которая управляется программой. Программы для машин Тьюринга записываются в виде таблицы, где первые столбец и строка содержат буквы внешнего алфавита и возможные внутренние состояния автомата (внутренний алфавит). Содержимое таблицы представляет собой команды для машины Тьюринга. Буква, которую считывает головка в ячейке (над которой она находится в данный момент), и внутренне состояние головки определяют, какую команду нужно выполнить. Команда определяется пересечением символов внешнего и внутреннего алфавитов в таблице.

Чтобы задать конкретную машину Тьюринга, требуется описать для нее следующие составляющие:

- **Внешний алфавит.** Конечное множество (например, A), элементы которого называются буквами (символами). Одна из букв этого алфавита (например, a_0) должна представлять собой пустой символ.
- **Внутренний алфавит.** Конечное множество состояний головки (автомата). Одно из состояний (например, q_1) должно быть начальным (запускающим программу). Еще одно из состояний (q_0) должно быть конечным (завершающим программу) – состояние останова.
- **Таблица переходов.** Описание поведения автомата (головки) в зависимости от состояния и считанного символа.

Автомат машины Тьюринга в процессе своей работы может выполнять следующие действия:

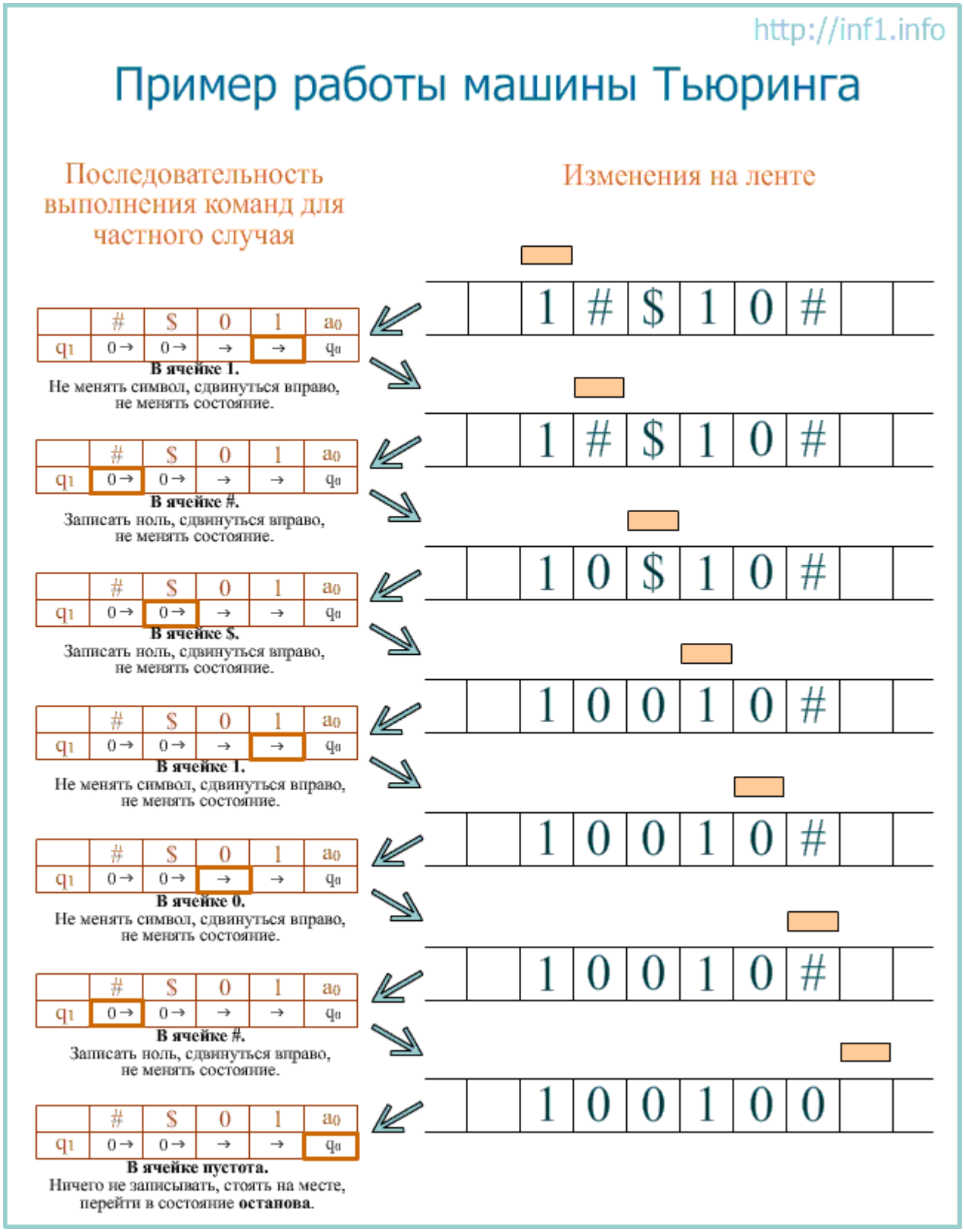
- Записывать символ внешнего алфавита в ячейку (в том числе и пустой), заменяя находившийся в ней (в том числе и пустой).
- Передвигаться на одну ячейку влево или вправо.
- Менять свое внутреннее состояние.

Одна команда для машины Тьюринга как раз и представляет собой конкретную комбинацию этих трех составляющих: указаний, какой символ записать в ячейку (над которой стоит автомат), куда передвинуться и в какое состояние перейти. Хотя команда может содержать и не все составляющие (например, не менять символ, не передвигаться или не менять внутреннего состояния).

Пример работы машины Тьюринга

Допустим, на ленте есть слово, состоящее из символов #, $, 1 и 0. Требуется заменить все символы # и $ на нули. В момент запуска головка находится над первой буквой слова слева. Завершается программа тогда, когда головка оказывается над пустым символом после самой правой буквы слова.
Примечание: длина слова и последовательность символов значения не имеют. На рисунке приводится пример последовательности выполнения команд для конкретного

случая. Если на ленте будет другое слово, то и последовательность выполнения команд будет другой. Несмотря на это, данная программа для машины Тьюринга (на рисунке – таблица слева) применима к любым словам описанного внешнего алфавита (соблюдается свойство применимости алгоритма ко всем однотипным задачам – массовость).



Можно усложнить программу. Допустим, головка располагается не обязательно над первым, а над любым символом слова. Тогда программа для данной машины Тьюринга может быть такой (а могла бы быть и другой):

Здесь происходит сдвиг головки влево до тех пор, пока она не окажется над пустым символом. После этого машина переходит в состояние q_2 (команды которого совпадают с командами q_1 предыдущей программы).

4. Пример

Пример. Требуется построить машину Тьюринга, которая прибавляет единицу к числу на ленте. Входное слово состоит из цифр целого десятичного числа, записанных в последовательные ячейки на ленте. В начальный момент машина находится против самой правой цифры числа.

Решение. Машина должна прибавить единицу к последней цифре числа. Если последняя цифра равна 9, то ее заменить на 0 и прибавить единицу к предыдущей цифре. Программа для данной машины Тьюринга может выглядеть так:

	a_0	0	1	2	3	4	...	7	8	9
q_1	1 Н q_0	1 Н q_0	2 Н q_0	3 Н q_0	4 Н q_0	5 Н q_0	...	8 Н q_0	9 Н q_0	0 Л q_1

В этой машине Тьюринга q_1 — состояние изменения цифры, q_0 — состояние останова. Если в состоянии q_1 автомат видит цифру 0..8, то он заменяет ее на 1..9 соответственно и переходит в состояние q_0, т.е. машина останавливается. Если же он видит цифру 9, то заменяет ее на 0, сдвигается влево, оставаясь в состоянии q_1. Так продолжается до тех пор, пока автомат не встретит цифру меньше 9. Если же все цифры были равны 9, то он заменит их нулями, запишет 0 на месте старшей цифры, сдвинется влево и в пустой клетке запишет 1. Затем перейдет в состояние q_0, т.е. остановится.

5. Задание

1. На ленте машины Тьюринга содержится последовательность символов “+”. Напишите программу для машины Тьюринга, которая каждый второй символ “+” заменит на “–”. Замена начинается с правого конца последовательности. Автомат в состоянии q_1 обозревает один из символов указанной последовательности. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

2. Дана десятичная запись натурального числа $n > 1$. Разработать машину Тьюринга, которая уменьшала бы заданное число n на 1. Автомат в состоянии q_1 обозревает

правую цифру числа. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.
3. На ленте машины Тьюринга находится число, записанное в десятичной системе счисления. Умножить это число на 2. Автомат в состоянии q_1 обозревает крайнюю левую цифру числа. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

Дополнительные задания
1. На ленте машины Тьюринга находится десятичное число. Определить, делится ли это число на 5 без остатка. Если делится, то записать справа от числа слово “да”, иначе — “нет”. Автомат обозревает некую цифру входного числа. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

2. Дано число n в восьмеричной системе счисления. Разработать машину Тьюринга, которая увеличивала бы заданное число n на 1. Автомат в состоянии q_1 обозревает некую цифру входного слова. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

3. Дана строка из букв “a” и “b”. Разработать машину Тьюринга, которая переместит все буквы “a” в левую, а буквы “b” — в правую части строки. Автомат в состоянии $q1$ обозревает крайний левый символ строки. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

Практическая работа №23. Построение машины Поста, вычисляющей простейшие арифметические функции.

1. Цель работы:
Формирование представления учащихся об универсальном исполнителе алгоритмов. Получение навыков построения программ для машины Поста.

2. Темы для предварительной проработки
- Общее понятие об алгоритме.
- Способы записи алгоритмов.

3. Теоретический материал
Машина Поста – это абстрактная (несуществующая реально) вычислительная машина, созданная для уточнения (формализации) понятия алгоритма. Представляет собой универсальный исполнитель, позволяющий вводить начальные данные и читать результат выполнения программы.
В 1936 г. американский математик Эмиль Пост в статье описал систему, обладающую алгоритмической простотой и способную определять, является ли та или иная задача алгоритмически разрешимой. Если задача имеет алгоритмическое решение, то она представима в форме команд для машины Поста.
Машина Поста состоит из …
1. **бесконечной ленты**, поделенной на одинаковые ячейки (секции). Ячейка может быть пустой (0 или пустота) или содержать метку (1 или любой другой знак),

2. **головки (каретки)**, способной передвигаться по ленте на одну ячейку в ту или иную сторону, а также способной проверять наличие метки, стирать и записывать метку.

Текущее **состояние машины Поста** описывается состоянием ленты и положением каретки. **Состояние ленты** – информация о том, какие секции пусты, а какие отмечены. **Шаг** – это движение каретки на одну ячейку влево или вправо. Состояние ленты может изменяться в процессе выполнения программы.

Кареткой управляет программа, состоящая из строк команд. Каждая команда имеет следующий синтаксис:

i K j,

где **i** - номер команды, **K** – действие каретки, **j** - номер следующей команды (отсылка).

Всего для машины Поста существует шесть типов команд:

- **V j** - поставить метку, перейти к j-й строке программы.
- **X j** - стереть метку, перейти к j-й строке программы.
- **<- j** - сдвинуться влево, перейти к j-й строке программы.
- **-> j** - сдвинуться вправо, перейти к j-й строке программы.
- **? j_1; j_2** - если в ячейке нет метки, то перейти к j_1-й строке программы, иначе перейти к j_2-й строке программы.
- **!** – конец программы (стоп).

У команды «стоп» отсылки нет.

Варианты окончания выполнения программы на машине Поста:

1. **Команда "стоп"** - корректная остановка. Возникает в результате выполнения правильно написанного алгоритма.
2. **Выполнение недопустимой команды** – нерезультативная остановка. Случаи, когда головка должна записать метку там, где она уже есть, или стереть метку там, где ее нет, являются аварийными (недопустимыми).
3. **Уход в бесконечность, зацикливание**. Машина Поста в результате работы алгоритма может вообще не остановиться (никогда не дойти до команды «стоп» и никогда не завершиться аварийной ситуацией).

Элементарные действия (команды) машина Поста проще команд машины Тьюринга. Поэтому программы для машины Поста имеют большее число команд, чем аналогичные программы для машины Тьюринга.

Почему достаточно лишь два различных символа (есть метка, нет метки)? Дело в том, что любой алфавит может быть закодирован двумя знаками; в зависимости от алфавита возрастать может только количество двоичных символов в букве алфавита.

Порядок работы

Запустить программу Algo2000.exe и при помощи команды ИНТЕРПРЕТАТОР перевести ее в режим работы Машина Поста. Используя файл add.pst, изучить принцип программирования на машине Algo2000.exe.

Порядок работы машины Поста

Работа машины Поста состоит в том, что каретка передвигается вдоль ленты и печатает или стирает метки. Чтобы машина Поста работала, надо задать некоторую программу и некоторое состояние машины (т.е. нужно как-то расставить метки по

секциям ленты, в частности, можно все секции оставить пустыми, и поставить каретку против одной из секций).
Работа машины на основании заданной программы происходит следующим образом. Машина приводится в начальное состояние и приступает к выполнению первой команды программы. Эта команда выполняется за один шаг, после чего машины приступает к выполнению той команды, номер которой равен отсылке первой команды. Эта команда также выполняется за один шаг, после чего начинается выполнение той команды, номер команды которой равен отсылке предыдущей команды.
Вообще каждая команда выполняется за один шаг, а переход от выполнения одной команды к выполнению другой происходит по следующему правилу: пусть на k-ом шаге выполнялась команда с номером a, тогда:
1) если эта команда имеет единственную отсылку b, то на (k + 1)-ом шаге выполняется команда с номером b;
2) если эта команда имеет две отсылки b1 и b2, то на (k + 1)-ом шаге выполняется одна из двух команд - с номером b1 или с номером b2;
3) если же выполняющаяся на k-ом шаге команда вовсе не имеет отсылки, то на (k +1) - ом шаге и на всех последующих шагах не выполняется никакая команда – машина останавливается.
Возможные случаи останова машины Поста:
1) в ходе выполнения программы машина дойдет до выполнения невыполнимой команды; выполнение программы прекращается, машина останавливается - происходит безрезультатная остановка.
2) в ходе выполнения программы машина дойдет до выполнения команды остановки; программа в этом случае считается выполненной, машина останавливается - происходит результативная остановка.
3) в ходе выполнения программы машина не дойдет до выполнения ни одной из команд, указанных в первых двух вариантах; выполнение программы при этом никогда не прекращается, машина никогда не останавливается - процесс работы машины происходит бесконечно.

3.2. Информационная лента машины Поста

Лента состоит из 1999 ячеек, нумерация от -999 до 999. Ячейка с толстой рамкой, находящаяся в центре ленты - каретка. Всплывающее меню ленты вызывается при щелчке правой кнопкой мыши на ленте. При получении фокуса лентой на ней появляется курсор (синий прямоугольник). Его можно сдвигать по ленте вправо и влево клавишами управления курсором. Удерживая клавишу Shift и нажимая клавиши "Влево"/ "Вправо" можно выделить несколько ячеек. Также ячейки можно выделить мышью: удерживая левую кнопку и выделив нужные. Метки ставятся/ удаляются в ячейках, которые выделены. Это можно сделать несколькими способами (лента должна быть сфокусирована):

- клавишей "Пробел"
- щелкнуть мышью на кнопку "V" на панели инструментов
- выбрать пункт главного или всплывающего меню "поставить/ удалить метки"
- двойным щелчком мыши (в этом случае изменение происходит только в одной ячейке).

Клавиши управления курсором "Вверх" или "Вниз" ставят курсор в каретку.

"Home" - курсор в начало видимой части ленты.
"End" - курсор в конец видимой части ленты.
"Ctrl" + "Влево" - сдвиг каретки влево
"Ctrl" + "Вправо" - сдвиг каретки вправо
Кнопки со стрелкой слева и справа от ленты - прокрутить каретку соответственно влево и вправо. Кнопки со стрелкой и вертикальной палочкой слева и справа от ленты - прокрутить каретку соответственно до крайней левой отмеченной ячейки и до крайней правой отмеченной ячейке. Номера этих ячеек указываются в строке состояния как мин и макс соответственно. Можно непосредственно поставить каретку в нужную ячейку, указав ее номер в редакторе над кареткой. Ленту можно запомнить (пункт меню Команда/ Запомнить ленту) и затем восстановить (Команда/ Восстановить ленту).

3.3. Исполнение алгоритма машины Поста

Запуск алгоритма на исполнение осуществляется:
- пунктом главного меню Пуск/ Запустить
- кнопкой Запустить на панели инструментов

При этом выполнение программы будет идти до тех пор, пока не встретиться команда стоп или не возникнет какая-нибудь ошибка.
Приостановить выполнение можно:
- пунктом главного меню Пуск/ Пауза;
- кнопкой Пауза на панели инструментов.

Если выполнение не было приостановлено, то оно всегда начинается с первой команды алгоритма. Если была нажата пауза, то можно продолжить выполнение с той команды, на которой машина остановилась. Программу можно выполнить по шагам:
- пунктом главного меню Пуск/ Пошагово;
- кнопкой Пошагово на панели инструментов.

При этом выполнится очередная команда и выполнение приостановится. Полностью прервать выполнение программы можно:
- пунктом главного меню Пуск/ Прервать;
- кнопкой Прервать на панели инструментов.

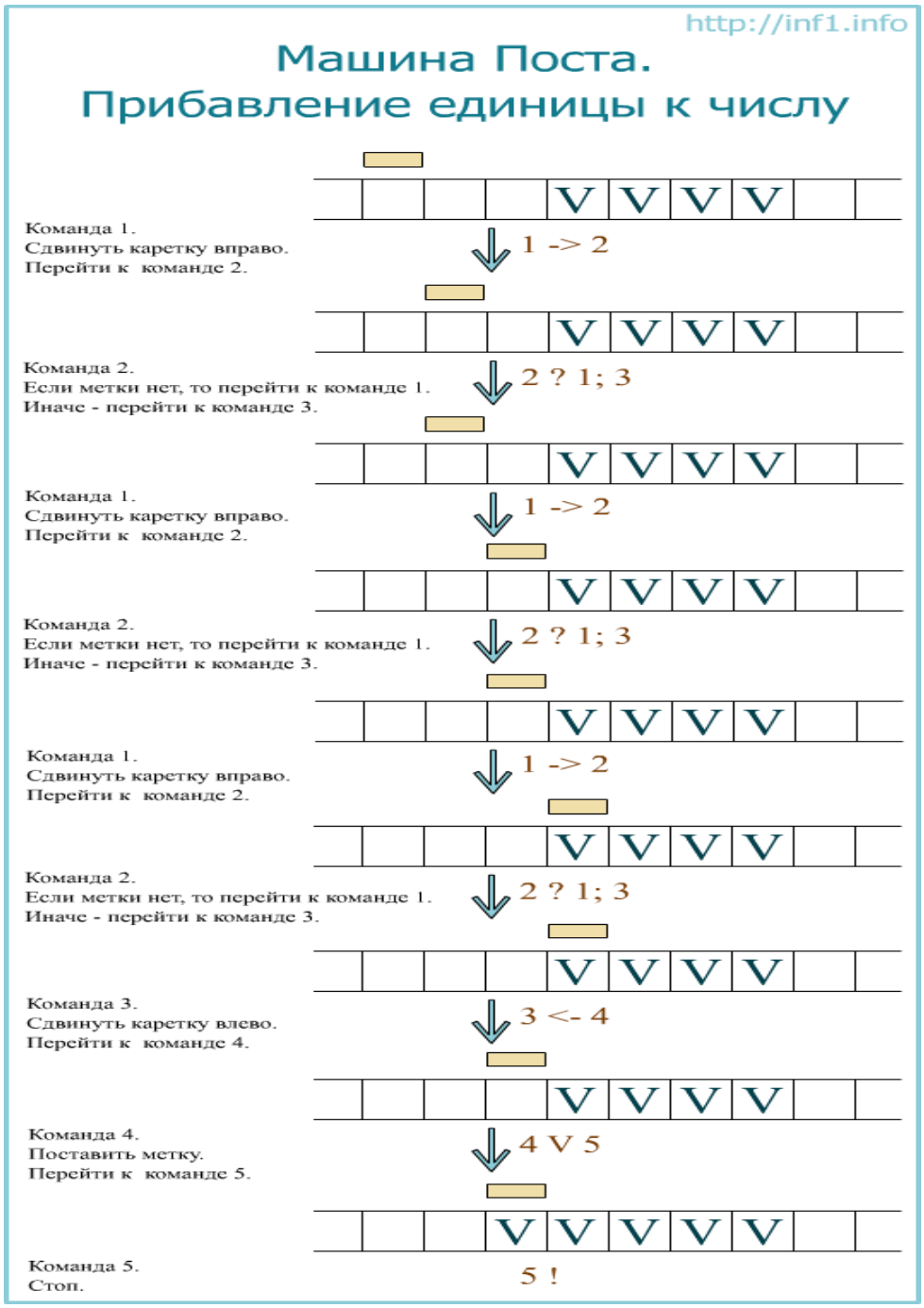
Регулировать скорость выполнения можно:
- пунктом главного меню Скорость

3.4. Ошибки машины Поста

При возникновении этих ошибок происходит прерывание исполнения алгоритма.
- Не указана команда
- Не указана отсылка
- Неверно указан номер отсылки
- Команды с таким номером не существует
- Нельзя поставить метку, где она уже есть
- Нельзя стереть метку, где ее нет
- Ошибка чтения файла
- Ошибка записи файла

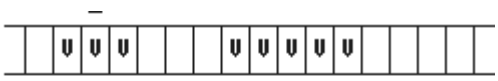
4. Пример

Увеличить число 3 на единицу (изменить значение в памяти с 3 на 4).
Целое положительное число на ленте машины Поста представимо идущими подряд метками, которых на одну больше, чем кодируемое число. Это связано с тем, что одна метка обозначает ноль, а уже две – единицу, и т.д.
Допустим, точно известно, что каретка стоит где-то слева от меток и обозревает пустую ячейку. Тогда программа увеличения числа на единицу может выглядеть так:
1 -> 2
2 ? 1;3
3 <- 4
4 V 5
5 !
А процесс выполнения может быть таким:



Приведем программу для сложения целых неотрицательных чисел a и b на машине Поста, когда головка находится над числом a, а число b находится правее числа a на некоторое количество клеток. Эта программа реализует следующий алгоритм: первое

число постепенно придвигается ко второму до их слияния, а потом стирается одна метка (иначе результат оказался бы на единицу больше правильного).



```
1 <- 2          7 -> 8          13 -> 14
   -> 3            -> 15
2 -|            8 -|            14 C 17
   -> 1            -> 9
3 -> 4          9 <- 10         15 <- 16
4  C 5          10 M 11         16 M 1
5 -> 6          11 <- 12        17 стоп 17
   -> 7            -> 7
6 -|            6 -|
   -> 5            -> 5
```

В случае более сложных начальных условий, когда неизвестно, справа или слева от головки (и на какое число клеток) находится число, можно применить такой принцип поиска числа: двигая головку поочередно вправо и влево и отмечая метками степень удаления головки от исходного положения, найти число, а потом уже применить известную программу сложения. При этом проверяется, находится ли головка над одной из меток числа и если да, то такая задача уже решена, иначе проверяется пуста ли секция справа от головки и следующая за ней; если обе пусты, то делается возврат головки на один шаг и ставится метка, а затем такая же операция выполняется слева и по отмеченной дорожке головка возвращается вправо и т.д. до тех пор, пока головка не натолкнется на число, после чего можно применить ранее рассмотренные выше программы:

```
   -> 2            6 <- 7          12 -> 13        18 -> 19
1 -|               7 M 8             -> 4           -> 20
   -> 16           8 <- 9          13 M 14         19 -|
2 -> 3                -> 10        14 -|              -> 18
   -> 4            9 -|               -> 15         20 M 21
3 -|                  -> 8          15 -> 14
   -> 16           10 <- 11                        21 стоп 12
4 -> 5                -> 12        16 <- 17
   -> 6            11 -|              -> 20
5 -|                  -> 16        17 -|
   -> 18                             -> 16
```

Машину Поста можно рассматривать как упрощенную модель ЭВМ. В самом деле, как ЭВМ, так и машина Поста имеют:

- неделимые носители информации (клетки - биты), которые могут быть заполненными или незаполненными;
- ограниченный набор элементарных действий - команд, каждая из которых выполняется за один такт (шаг).

Обе машины работают на основе программы. Однако, в машине Поста информация располагается линейно и читается подряд, а в ЭВМ можно читать информацию по адресу; набор команд ЭВМ значительно шире и выразительнее, чем команды машины Поста и т.д.

5. Задание

Пояснения к условиям задач

1) В задачах под *массивом* понимается последовательность подряд идущих меток, ограниченная пустыми ячейками.

2) Если в задаче говорится, что на ленте задано число в унарной системе, то имеется в виду, что натуральное число *n* закодировано с помощью массива длины *n*.

3) В задачах при описании начального состояния ленты будем указывать то, что записано начиная с самой левой непустой ячейки и заканчивая самой правой непустой

ячейкой. При этом будем использовать следующие обозначения: n подряд идущих меток будем обозначать $1n$, а m пустых ячеек — $0m$. При обозначении одной заполненной или пустой ячейки будем писать просто 1 или 0, соответственно.
К примеру, запись “12012” будет соответствовать записи “11011” на ленте.
4) Если не сказано ничего о местонахождении каретки в начальный момент времени, то будем считать, что каретка обозревает ячейку с самой левой меткой.

1. Составьте и проверьте программу для машины Поста, создающую на ленте копию заданной последовательности меток справа от нее.

2. На ленте проставлена метка в одной-единственной ячейке. Каретка стоит на некотором расстоянии левее этой ячейки. Необходимо подвести каретку к ячейке, стереть метку и остановить каретку слева от этой ячейки.

3. На ленте задан массив меток. Увеличить длину массива на 2 метки. Каретка находится либо слева от массива, либо над одной из ячеек самого массива.

4. Даны два массива меток, которые находятся на не котором расстоянии друг от друга. Требуется соединить их в один массив. Каретка находится над крайней левой меткой первого массива.

Дополнительные задания

1. Дан массив меток. Каретка располагается где-то над массивом, но не над крайними метками. Стереть все метки, кроме крайних, и поставить каретку в исходное положение.

2. На ленте машины Поста расположен массив из n меток (метки расположены через пробел). Нужно сжать массив так, чтобы все n меток занимали n расположенных подряд ячеек.

3. Дано несколько массивов меток. Удалить четные массивы. Каретка находится над первым массивом.

Литература

Основные источники:

1. Балашова Т.Е. Основы программирования на языке Pascal: учеб. пособие. – Брянск: БГТУ, 2017. – 155 с. – 15 экз.
2. Вайнштейн, Ю. В. Математическая логика и теория алгоритмов : учебное пособие / Ю. В. Вайнштейн, Т. Г. Пенькова, В. И. Вайнштейн. — Красноярск : Сибирский федеральный университет, 2019. — 110 с. — ISBN 978-5-7638-4076-6. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: https://www.iprbookshop.ru/100046.html
3. Гвоздева В.А. Информатика, автоматизированные информационные технологии и системы, - М.: Форум: Инфра-М, - 2017, - 544 с.
4. Голицына О.Л. Программирование на языках высокого уровня: учеб. пособие для сред. проф. образован., М.: Форум, 2017. – 495 с.
5. Голицына О.Л. Языки программирования: учеб. пособие для сред. проф. образован. – М.: Форум : Инфра-М, 2017 – 400 с.

6. Коростелев Д.А. Программирование в среде Microsoft. NET: учеб. пособие: лаб. практикум. – Брянск: БГТУ, 2017. – 145 с. (фонд БГТУ)
7. Мирзоев, М. С. Теория алгоритмов : учебное пособие / М. С. Мирзоев, В. Л. Матросов. — Москва : Прометей, 2019. — 200 с. — ISBN 978-5-907100-65-7. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: https://www.iprbookshop.ru/94547.html
8. Токманцев, Т. Б. Алгоритмические языки и программирование : учебное пособие для СПО / Т. Б. Токманцев ; под редакцией В. Б. Костоусова. — 2-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2019. — 102 с. — ISBN 978-5-4488-0510-3, 978-5-7996-2899-4. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/87785.html
9. Хлебников А.А. Информатика: учеб. для сред. проф. образован.. – Ростов н/Д: Феникс, 2017. – 426 с.

Дополнительные источники:

1. Воронцова, Н. В. Численные методы в программировании : учебное пособие для СПО / Н. В. Воронцова, Т. Н. Егорушкина, Д. И. Якушин. — Саратов : Профобразование, Ай Пи Эр Медиа, 2019. — 125 с. — ISBN 978-5-4486-0761-5, 978-5-4488-0278-2. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/86341.html
2. Пименов, В. Г. Численные методы. В 2 ч. Ч. 1 : учебное пособие для СПО / В. Г. Пименов ; под редакцией Ю. А. Меленцовой. — 2-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2019. — 111 с. — ISBN 978-5-4488-0398-7, 978-5-7996-2919-9. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/87906.html
3. Пименов, В. Г. Численные методы. В 2 ч. Ч. 2 : учебное пособие для СПО / В. Г. Пименов, А. Б. Ложников ; под редакцией Ю. А. Меленцовой. — 2-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2019. — 105 с. — ISBN 978-5-4488-0399-4, 978-5-7996-2894-9. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/87905.html

Интернет-ресурсы:

1. www.fcior.edu.ru (Федеральный центр информационно-образовательных ресурсов — ФЦИОР).
2. www. school-collection. edu. ru (Единая коллекция цифровых образовательных ресурсов).
3. www. intuit. ru/studies/courses (Открытые интернет-курсы «Интуит» по курсу «Информатика»).