Министерство образования и науки Российской Федерации
ФГБОУ ВО «Удмуртский государственный университет»
Институт математики, информационных технологий и физики
Кафедра теоретических основ информатики

А.Е. Анисимов

ВВЕДЕНИЕ В АЛГОРИТМИЗАЦИЮ ЗАДАЧ

Учебно-методическое пособие

Ижевск

2017

УДК 004.424
ББК 32.973.26-018
А67

Рекомендовано к изданию Учебно-методическим советом УдГУ

Рецензент:
М. А. Клочков, кандидат физико-математических наук, доцент кафедры высокопроизводительных вычислений и параллельного программирования

Анисимов А.Е.
А67 Ведение в алгоритмизацию задач/А.Е. Анисимов.– Ижевск: Изд-во «Удмуртский университет», 2017. – 44 с.

ISBN 978-5-4312-0512-5

Предлагаемое учебно-методическое пособие предназначено для использования в рамках дисциплин начального обучения программированию программ среднего профессионального образования по специальности «Информационные системы» и направлений подготовки высшего образования по программам бакалавриата «Прикладная информатика», «Информационные системы», «Фундаментальная информатика и информационные технологии» и ряда других.

Изучение теоретической части пособия и выполнение предлагаемых заданий направлено на формирование компетенций специалиста, связанных с программированием и проектированием приложений, осуществлением и обоснованием выбора проектных решений по видам обеспечений информационных систем.

УДК 004.424
ББК 32.973.26-018

ISBN 978-5-4312-0512-5

Содержание

Введение

Данное учебно-методическое пособие предназначено для использования в рамках таких курсов программ среднего профессионального образования по специальности "Информационные системы" как «Основы алгоритмизации и программирования», и дисциплин высшего образования по программам бакалавриата направлений "Прикладная информатика", "Информационные системы", "Фундаментальная информатика и информационные технологии", таких как «Информатика и программирование», «Практикум по программированию», «Технологии программирования» и ряда других.

Алгоритмизация понимается как систематическая деятельность по построению алгоритмов решения прикладных задач. Алгоритмизация является одним из этапов проектирования и разработки программных приложений.

Результатом изучения темы алгоритмизации предполагается формирование у обучающихся ряда профессиональных компетенций, в том числе способность применять в системный подход и математические методы в формализации решения прикладных задач, осуществлять и обосновывать выбор проектных решений по видам обеспечения информационных систем. Теоретическая и практическая части пособия предназначены для закрепления студентами начальных сведений и получения первых анализа поставленной задачи на программирование и формирование эффективного алгоритма её решения.

Каждая тема снабжена контрольными материалами для проверки и самопроверки уровня полученных знаний, умений и навыков. Основной формой освоения материала является самостоятельная деятельность студента при наличии необходимого контроля со стороны преподавателя и дифференцированного подхода к распределению задач.

Тема 1: Определение и свойства алгоритма

Теоретический материал

Алгоритмом называется точное и понятное исполнителю предписание (инструкция) совершить определенную последовательность действий для достижения указанной цели или решений поставленной задачи.

Каждый алгоритм рассчитан на определенного исполнителя. Под ***исполнителем алгоритма*** понимается некий объект или субъект, который способен понимать и исполнять действия, предписываемые ему алгоритмом. Примерами исполнителя могут служить дрессированное животное, автоматы по продаже напитков, лифт в доме, человек, если он действует строго по некоторой инструкции или под руководством командира. Для каждого исполнителя существует ***система команд***, каждую из которых он в состоянии понять (распознать) и исполнить соответствующее действие.

Пример 1. Исполнители и их системы команд.

Исполнитель	Система команд
Служебная собака	«Сидеть», «Лежать», «Вперед», «Фас», …
Солдат	«Стройся», «Смирно», «Шагом марш», «Налево», «Разойдись», …
Процессор ЭВМ	«Сложить содержимое регистра АХ с содержимым регистра ВХ, результат записать в АХ», «Записать в память по адресу 1000 значение 0», …
Лифт в доме	«Открыть дверь», «Закрыть дверь», «Двигаться вверх», «Двигаться вниз», «Остановиться»

И, совершенно очевидно, что если исполнитель получит команду, не входящую в его систему команд, то понять, а уж тем более исполнить команду он вряд ли сможет. Например, служебная собака не поймет команду «Сложить содержимое регистра АХ…», так как эта команда не входит в собачью систему команд.

При рассмотрении алгоритмов мы будем считать, что они рассчитаны на формальных исполнителей, то есть таких, которые не способны самостоятельно принимать решения или почему-либо отказываться от исполнения алгоритма. Например, собака может заболеть, а лифт сломаться, что делает

невозможным исполнение алгоритма этим исполнителем, а, значит, эти исполнители не являются формальными.

Таким образом, соотношение понятий алгоритм, исполнитель алгоритма и процесс исполнения алгоритма можно изобразить следующей схемой:

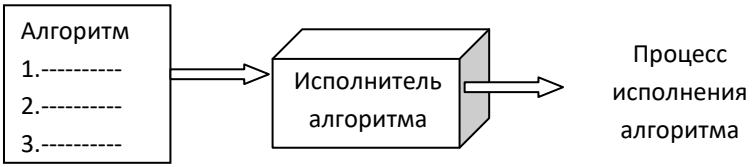


Рисунок 1. Алгоритм и исполнитель алгоритма

Свойства алгоритма

Алгоритм, как инструкция к действию (исполнения), предписание для исполнения должен обладать следующими свойствами:

Дискретность. Алгоритм состоит из конечной последовательности отдельных шагов-команд. Следовательно, команды можно пронумеровать. Исполняются команды последовательно, если иное не указано в самом алгоритме. Исполнение очередного шага-команды начинается только после завершения исполнения предыдущего шага.

Пример 2. **Пример 2.** Составим алгоритм для заваривания чая.

```
Алгоритм ЗавариваниеЧая
Начало алгоритма
      1. Ополоснуть заварочный чайник кипятком.
      2. Засыпать в чайник заварку из расчета одна
         чайная ложка на одну чашку чая.
      3. Залить в чайник кипяток на 2/3 объема.
      4. Подождать 5 минут.
      5. Заварка готова.
Конец алгоритма
```

Выполнимость. Исполнение алгоритма исполнителем заканчивается за конечное число действий. Не может быть выполнен алгоритм, то есть успешно получен результат, если количество шагов бесконечно.

Пример 3. Требуется составить алгоритм для решения следующей задачи. Найти сумму всех членов бесконечной геометрической прогрессии $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, … с первым членом $\frac{1}{2}$ и знаменателем $\frac{1}{2}$.

Решение 1. Составим для некоторого исполнителя следующий алгоритм решений указанной задачи.

```
Алгоритм СуммаПрогрессииНеВыполнимая
Начало алгоритма
      1. Переменой S присвоить значение 0.
      2. Переменной N присвоить значение 2.
      3. Увеличить значение S на величину, равную 1/N.
      4. Домножить N на 2.
      5. Перейти к исполнению шага № 3.
Конец алгоритма
```

Попробуйте его исполнить. Ясно, что такой алгоритм никогда не сможет быть выполненным, так как после исполнения шагов 3, 4 и 5 вновь придется исполнить шаги 3, 4 и 5 и так бесконечно.

Приведем другое решение этой же задачи.

Решение 2.

```
Алгоритм СуммаПрогрессии
Начало алгоритма
      1. Переменой S присвоить значение, равное 1/2 / (1-1/2).
Конец алгоритма.
```

В этом алгоритме использована формула суммы бесконечной геометрической прогрессии. Ясно, что, в отличие от предыдущего алгоритма, в этом

решении сумма будет найдена за исполнение всего одного шага. Это выполнимый алгоритм.

Определенность (детерминированность). При исполнении алгоритма исполнителем не должно возникать ситуаций неопределенности, когда исполнитель «не знает», что делать дальше или не может правильно «понять» очередной шаг. То есть алгоритм рассчитан на чисто механическое исполнение, так как исполнитель не имеет права на самостоятельное принятие решений. Все действия исполнителя полностью определяются в самом алгоритме.

Массовость. Алгоритм, как правило, рассчитан на решение не только какой-то конкретной задачи, а на решение, вообще говоря, любой задачи из некоторого класса однотипных задач. Например, в начальной школе учеников знакомят с алгоритмом сложения натуральных чисел столбиком. При использовании этого алгоритма школьник сможет сложить *любые* натуральные числа, а не только какие-то два конкретных числа. То есть такой алгоритм применим для сложения любых натуральных чисел, что и определяет его как массовый.

В примере 3 приведен алгоритм СуммаПрогрессии, который не является массовым, так как предназначен для нахождения суммы только *одной конкретной* геометрической прогрессии. Сделать его массовым легко, это выполнено ниже.

Наличие входа и выхода. ***Вход*** – это данные, задаваемые исполнителю для выполнения алгоритма. Считаем, что значения входных данных поступают *извне* процесса исполнения, из внешнего мира по отношению к исполнителю. Кстати, именно значения входных величин и определяет, какую именно конкретную задачу из множества однотипных задач решает исполнитель (см. Массовость). Следует отметить, что значения входных данных должны принадлежать заранее определенному *множеству допустимых входных значений*. Это *гарантирует*, что выполнение алгоритма исполнителем приведет к успешному решению задачи. В противном случае алгоритм может и *не быть успешно исполненным*.

Выход – полученный результат. Так как мы будем иметь дело с обработкой данных, то под выходом обычно будем понимать одно или несколько значений, которые являются результатом вычислений.

Понятия алгоритма и исполнителя, процесса выполнения, входа и выхода взаимосвязаны, это показано на следующей схеме (Рисунок 2).

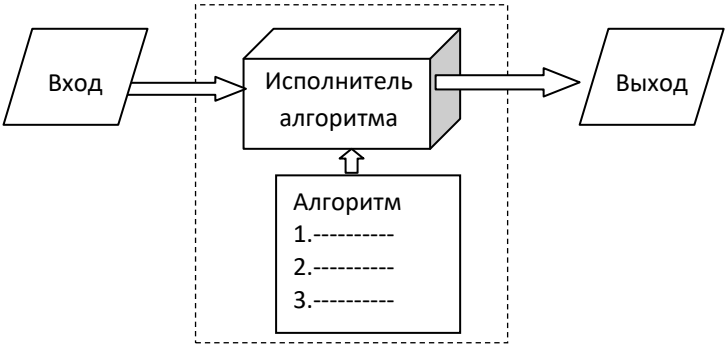


Рисунок 2. Вход и выход алгоритма

Таким образом, можно считать, что алгоритм, выполняемый исполнителем, преобразует входные данные в выходные. Часто в постановке задачи входные данные описываются предложением «Дано ...»

Пример 4 .Составить алгоритм для решения следующей задачи. Даны действительные числа a и q, отличные от нуля, причем $-1 < q < 1$. Найти сумму геометрической прогрессии, первый член которой равен a, а знаменатель равен q.

Решение. Определим, что необходимо задать исполнителю, чтобы он смог найти требуемую сумму. Ясно, что необходимо задать два значения – первый член прогрессии a и знаменатель q. Однако, исходя из постановки задачи, эти значения не могут быть произвольными. Каждое из них должно принадлежать соответствующему множеству допустимых значений. Именно принадлежность этому множеству гарантирует, что алгоритм будет успешно исполнен. В этой задаче множеством допустимых значений величины a яв-

ляются все действительные числа, кроме 0 ($a \in (-\infty;0) \cup (0;+\infty)$), а для q – это множество всех действительных чисел из интервала от –1 до 1, исключая 0 ($q \in (-1;0) \cup (0;1)$).

Таким образом, искомый алгоритм можно сформулировать так:

```
Алгоритм СуммаПрогрессии2
Вход: a, q (где a ∈ (−∞; 0) ∪ (0; +∞), q ∈ (−1; 0) ∪ (0; 1)
Выход: S - сумма геометрической прогрессии с первым
       членом, равным a и знаменателем q
Начало алгоритма
      1. Переменой S присвоить значение a/(1−q).
Конец алгоритма.
```

Обратите внимание, что если бы q не принадлежало множеству $(-1;0) \cup (0;1)$ (например, $q = 1$), то исполнение алгоритма привело бы к ошибке.

Введем еще некоторые понятия.

Тестом называется набор значений входных данных алгоритма, для которого заранее известен набор выходных значений. Например, в предыдущем примере тестом могут служить значения входных данных $a = \frac{1}{2}$ и $q = \frac{1}{2}$, для которых известен выход $S = 1$. Тест может быть использован для проверки правильности алгоритма; задавая входные данные теста на вход мы ожидаем получить корректный выход, и в случае, если выход не соответствует ожидаемому, то предполагается, что в алгоритме допущена ошибка, которую необходимо обнаружить и исправить.

Тестированием алгоритма будем называть формальное исполнение алгоритма на заранее подготовленной ***системе тестов*** с целью обнаружения ошибок, допущенных при составлении алгоритма. Для удобства систему тестов будем размещать в таблице, как это сделано в следующем примере.

Пример 5. Приведем систему тестов для примера 4 (нахождение суммы геометрической прогрессии):

Тесты для алгоритма СуммаПрогрессии2			
№ теста	Вход		Выход
	a	q	
1.	0,5	0,5	1
2.	–2	0,75	–8
3.	1	–0,25	0,8

Составление системы тестов (разумный подбор входных значений) для алгоритма – отдельная тема.

Формы представления алгоритма

Существует несколько различных форм представления алгоритмов, отличающиеся друг от друга языком описания алгоритма. Представим здесь некоторые из них.

а) Словесная форма. В этом случае алгоритм описывается средствами естественного языка, например, русского или английского. Этот способ хорош тем, что его может понять практически любой человек, знакомый с предметной областью решаемой задачи. Недостатками (в общем случае) являются отсутствие строгой формализации, многословность, неоднозначность толкования – в общем все, что свойственно естественному языку. Рассмотрим пример использования словесного представления алгоритма Евклида для поиска *наибольшего общего делителя* (НОД) двух натуральных чисел.

Пример 6.

Алгоритм Евклида
Вход: m, n, *где* $m \in N, n \in N$ (N – *множество натуральных чисел*);
Выход: целое число, равное НОД(m, n)
Начало алгоритма
1. Если $m \neq n$, то перейти к шагу 2, иначе – перейти к шагу 5.
2. Найти большее из m и n.

```
      3. Заменить большее на разность большего и
         меньшего.
      4. Перейти к шагу 1.
      5. Выход: m.
Конец алгоритма.
```

Составим систему тестов для приведенного алгоритма.

Тесты для алгоритма Евклида			
№ теста	Вход		Выход
	m	n	
1.	21	35	7
2.	17	16	1
3.	100	100	100

Исполним этот алгоритм на примере теста 1. Начальные значения переменных m=21 n=35. Процесс исполнения по шагам зафиксируем в следующей таблице, в которой для каждой переменной алгоритма назначен столбец, а исполнение каждого шага алгоритма (действие), изменяющее значение переменной – размещено в отдельной строке.

Процесс исполнения тестов алгоритма Евклида				
№ действия	m	n	Выход	Пояснения
1.	21	35		Так как 21≠35, то переходим к шагу 2. Большее из 21 и 35 равно 35 (переменная n)
2.		14		Заменяем значение n на разность 35-21=14 и вновь возвращаемся к шагу 1
3.	7			Так как 21≠14, то переходим к шагу 2. Большее из 21 и 14 равно 21 (переменная m). Заменяем m на 21-14=7
4.		7		Так как 7≠14, то переходим к шагу 2. Большее из 7 и 14 равно 14 (переменная n). Заменяем n на 14-7=7
5.			7	Так как 7=7, то переходим к шагу 5. На выход подается значение переменной m, то

				есть 7.
				Исполнение алгоритма закончено.

Остальные тесты исполните самостоятельно, пояснения можно не писать.

б) Графическая форма. При описании структуры алгоритма используются графические обозначения. Чаще всего языком графического представления алгоритма является «язык» блок-схем. Достоинством такого обозначения является легкость восприятия структуры алгоритма человеком.

Обозначения, используемые в блок-схемах, будут введены в последующих разделах. Здесь лишь приведем пример блок схемы алгоритма Евклида, рассмотренного выше.

Пример 7. Составим блок-схема алгоритма Евклида (Рисунок 3). Сравните представление алгоритма с предыдущим примером.

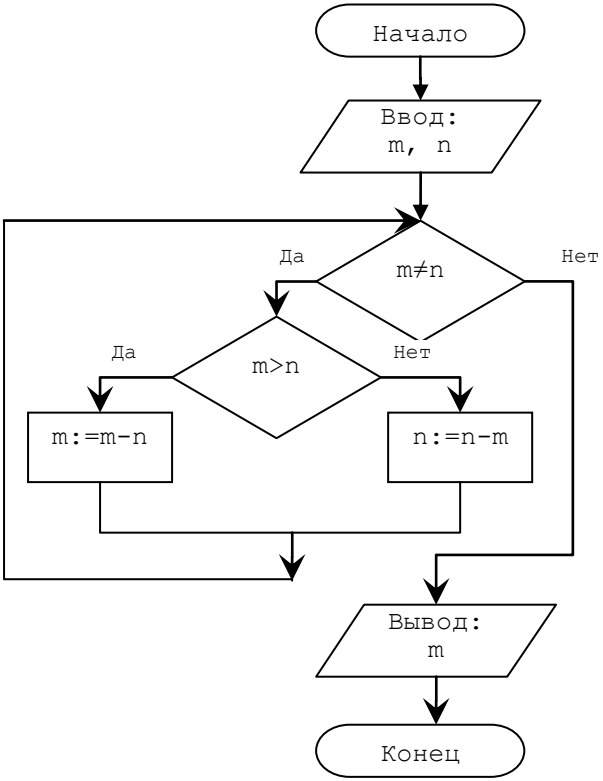


Рисунок 3. Блок-схема алгоритма Евклида

в) Язык программирования. В этом случае алгоритм описывается средствами некоторого алгоритмического языка – языка программирования. Подробнее о программировании будет сказано во второй части пособия.

Пример 8. Программа на языке Паскаль, реализующая алгоритм Евклида. Сравните программу с представлениями того же алгоритма в двух последних примерах.

```
program Euklid;
var
  m, n : integer;
begin
  readln(m,n);
```

```
  while m<>n do
    if m>n then
      m:=m-n
    else
      n:=n-m;
  writeln(m);
end.
```

Материал для проверки усвоения темы 1

Контрольные вопросы и задания темы 1

1. Что такое алгоритм? Приведите примеры алгоритмов, которые встречаются в обыденной жизни.
2. Кто или что такое исполнитель алгоритма?
3. Как соотносятся понятия «алгоритм» и «исполнение алгоритма»?
4. Почему у каждого исполнителя своя система команд? Приведите свои примеры исполнителей и их системы команд.
5. Сформулируйте алгоритмы «Уборка квартиры», «Получение денег в банкомате», «Решение линейного уравнения».
6. Опишите свойства алгоритма на примере одного из алгоритмов, построенных в предыдущем упражнении.
7. Чем отличается формальный исполнитель от неформального?
8. Какие свойства алгоритмов вы знаете?
9. Определите каждое свойство алгоритма.
10. Что такое «вход» и «выход» алгоритма?
11. Чем может закончиться исполнение алгоритма, если на вход были заданы данные, не входящие в множество допустимых значений?
12. Что такое тест?
13. Что такое тестирование?
14. Перечислите формы представления алгоритма. В чем особенность каждой из форм представления алгоритма?

15. Всегда ли можно алгоритм, представленный в одной из форм записи, преобразовать в другую форму?

Упражнения по теме 1

1. Составьте алгоритм сложения натуральных чисел столбиком в словесной форме (устно или письменно).
2. Составьте в виде блок-схемы алгоритм решения уравнения $bx + c = 0$ (b, c даны).
3. Составьте в виде блок-схемы алгоритм решения следующей задачи: дано натуральное число N. Выяснить, является ли оно простым или составным?

Задания для самостоятельной работы по теме 1

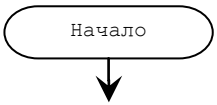
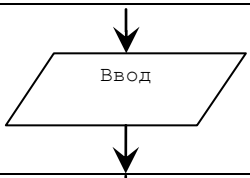
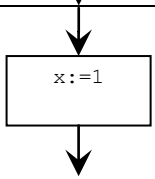
1. Протестируйте алгоритм алгоритма Евклида на всех тестах из примера 6.
2. Составьте таблицу тестов для тестирования алгоритма решения квадратного уравнения, предусмотрев, по возможности, все различные по смыслу случаи входных данных.
3. Напишите в словесной форме алгоритм сложения двух обыкновенных дробей. Какие дополнительные алгоритмы для этого могут потребоваться?

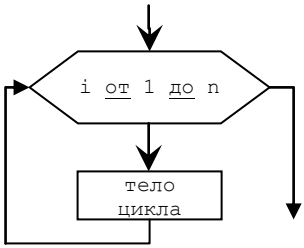
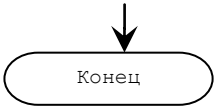
Тема 2: Линейный алгоритм, алгоритм с ветвлением

Теоретический материал

Вначале введем условные обозначения графического языка блок-схем (Таблица 1).

Таблица 1. Условные обозначения блок-схем

Условное графическое обозначение	Название	Комментарий
	Стрелка	Означает, к исполнению какого действия следует перейти. С помощью стрелки обозначают последовательность исполнения команд
	Начало	Точка блок-схемы, с которой начинается исполнение алгоритма
	Ввод или Вывод	Блок, означающий, что в этом месте алгоритма необходимо произвести ввод или вывод данных
	Простое действие (в данном случае – присваивание)	Один элементарный шаг алгоритма. Присваивание значения выражения переменной (в данном случае переменной x присвоено значение 1)
	Условие	Исполнение предполагает вычисление условия – логического выражения. Если условие истинно (Истина, True), то необходимо перейти к действию по стрелке

Условное графическое обозначение	Название	Комментарий
		помеченной T, если условие ложно – то по стрелке F (Ложь, False)
	Модификация (цикл со счетчиком)	Повторение исполнения тела цикла для каждого из последовательных значений целочисленного счетчика i от 1 до n.
	Конец	Точка блок-схемы, дойдя до которой прекращается процесс исполнения алгоритма.

Выделяют три вида алгоритмических схем, с помощью которых можно представить (сконструировать) любой алгоритм.

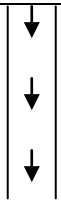
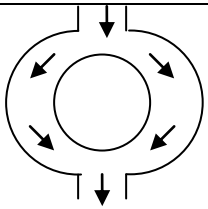
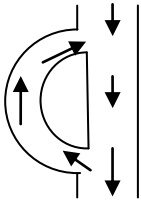
Линейный алгоритм представляет собой последовательность шагов (действий) без ветвлений и возвратов. Последовательность исполнения шагов такого алгоритма полностью совпадает с его структурой.

Алгоритм с ветвлением. В таком алгоритме исполнение тех либо иных действий зависит от истинности некоторого условия (логического выражения).

Циклический алгоритм. Алгоритм, в котором некоторая последовательность шагов (команд), исполняется многократно. Исполняемые многократно шаги называются ***телом цикла***. Количество повторений может оказаться равным 0 (то есть тело цикла ни разу не исполнилось), 1 (однократное исполнение) или быть больше (многократное исполнение).

Указанные виды алгоритмических схем можно проиллюстрировать следующими рисунками[1] (Таблица 2). Стрелки на рисунках обозначают направление *возможного* движения по «коридору».

Таблица 2. Пояснение смысла алгоритмических схем

Линейный	Ветвление	Цикл
		

Подробнее о применении указанных алгоритмических схем будет рассмотрено в следующих разделах.

Линейные алгоритмы

Линейный алгоритм представляет собой простую последовательность шагов, которые исполняются в том порядке, в котором они перечислены в алгоритме (Рисунок 4). В линейном алгоритме не может быть ветвлений и возвратов.

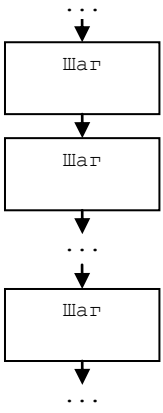


Рисунок 4. Линейный алгоритм

Рассмотрим несколько примеров задач на составление линейных алгоритмов.

Пример 8. Составить блок-схему алгоритма, решающего следующую задачу. Даны три вещественных положительных числа a, b и c. Найти площадь треугольни-

[1] Рисунки весьма условны и поясняют смысл алгоритмических схем, но не являются их точным определением.

ка, стороны которого равны a, b и c.

Решение. Для нахождения площади треугольника по трем его известным сторонам воспользуемся формулой Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)} \quad ,$$

где p – полупериметр треугольника, равный

$$p = \frac{a+b+c}{2} .$$

Зная последовательность вычислений из школьного курса математики, легко составить алгоритм (Рисунок 5).

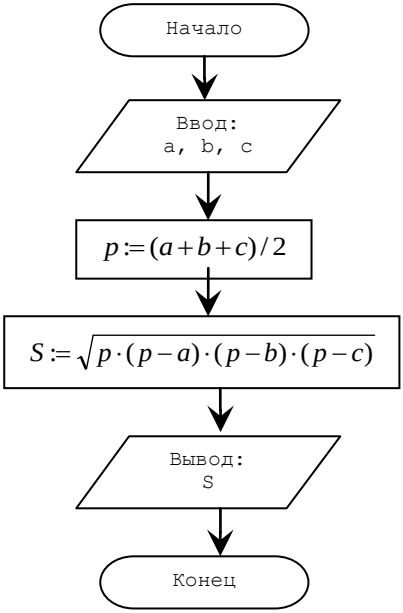


Рисунок 5. Блок-схема алгоритма нахождения площади треугольника

Задача решена. Сделаем два замечания. Первое: в отличие от математики, где приводимые формулы декларируют некоторые факты, соотношения и порядок указания которых не так важен (или важен только с точки зрения логичности изложения), в алгоритмических схемах важна в первую очередь правильная последовательность действий (формул), которая и определяет порядок выполнения шагов. Например, если в приведенной блок-схеме переставить местами шаги, вычисляющие S и p, то алгоритм не будет правильным, так как до вычисления S необходимо *предварительно* вычислить p.

Второе замечание. В решении этой задачи никак не рассматривается вопрос существования треугольника, площадь которого вычисляется. То есть мы предполагаем, что входные данные должны быть корректны. В данном случае должны выполняться ус-

ловия существования треугольника: $a<b+c, b<a+c, c<a+b$. Алгоритм не может быть успешно исполненным, если эти неравенства не выполняются. Кстати, почему?

Пример 9. Составить блок-схему решения следующей задачи. Даны значения двух действительных переменных a и b. Обменять местами их значения, то есть добиться, чтобы a получила значение, которое изначально имела переменная b, а b – получила бы значение a.

Если первым же присваиванием алгоритма мы переменной *a* присвоим *b*, то сразу же потеряем исходное значение *a.* Поэтому воспользуемся для временного хранения исходного значения переменной *a* дополнительной переменной *d*. Блок-схема алгоритма приведена ниже (Рисунок 6).

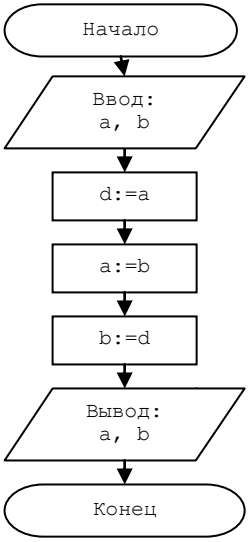


Рисунок 6. Блок-схема алгоритма обмена значений

Пример 10. Составить блок-схему решения следующей задачи. Даны значения двух действительных переменных a и b. Обменять местами их значения без использования дополнительных переменных.

В предыдущем примере решалась та же задача, но сейчас запрещается использовать какие-либо переменные, кроме самих a и b. Казалось бы, это невозможно, однако, можно найти и такое решение, причем не одно! Блок-схема решения – на рисунке (Рисунок 7).

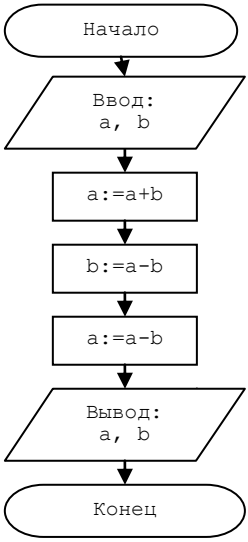


Рисунок 7. Решение задачи без дополнительных переменных

Упражнение 1. Составить блок-схему решения следующей задачи. Даны значения двух действительных переменных a и b. Обменять местами их значения, при этом нельзя использовать никаких дополнительных переменных.

Упражнение 2. Составить блок-схему решения следующей задачи. Даны значения трех действительных переменных a, b и c. Обменять местами их значения так, чтобы a получила бы значение b, b получила значение c, а переменная c получила исходное значение a.

Алгоритмы с ветвлением

При создании алгоритмов часто возникает необходимость исполнение тех или иных действий поставить в зависимость от некоторого условия. Такая возможность называется **ветвлением**. Блок-схема ветвления приведена на рисунке 8.

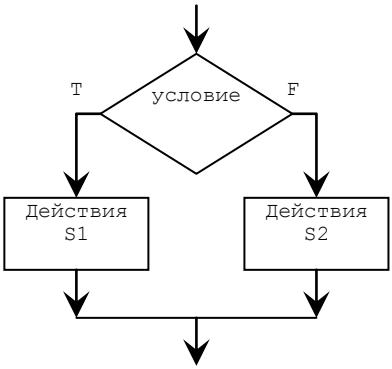


Рисунок 8. Алгоритм с ветвлением

Последовательность исполнения ветвления такова: вначале вычисляется значение условия – логического выражения. Затем, если значение условия истинно, то исполняются действия S1 (мы будем их всегда размещать на левой ветви ветвления и помечать эту ветвь буквой T); если же значение условия ложно, то исполняются действия S2 (такие действия будем располагать на правой ветви и помечать её буквой F).

Заметим, что при исполнении ветвления будет исполнена либо ветвь T (действия S1, слева), либо ветвь F (действия S2, справа). Одновременно пройти по обеим ветвям или не пройти ни по одной *нельзя*.

Иногда одна из ветвей алгоритма с ветвлением отсутствует, то есть нет либо действий S1, либо действий S2. Этот случай называют *неполным ветвлением***.**

Пример 11. Составить блок-схему решения следующей задачи. Даны чения двух действительных переменных a и b. Найти наибольшее значение из a и b.

Составим блок-схему алгоритма по следующим соображениям. Мы должны сравнить значения переменных a и b, и если из них a имеет большее значение, то присвоить это значение переменной max. Если же a не больше b, но присвоить переменной max значение b. После этого в переменной max будет храниться искомое наибольшее значение из a и b. Получаем блок-схему (Рисунок 9).

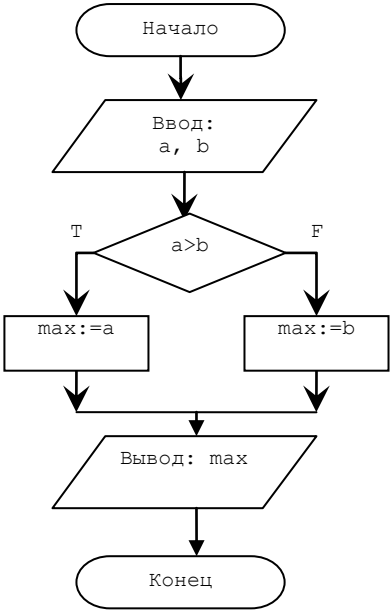


Рисунок 9. Нахождение наибольшего из a и b.

Пример 12. Составить блок-схему решения следующей задачи. Даны значения трех действительных переменных a, b и c. Найти наибольшее значение из a, b и c.

Выбирать наибольшее значение из двух уже умеем (см. пример выше). В этой задаче надо найти большее из трех. Можно ли свести эту задачу к предыдущей? Можно, а именно – вначале найти наибольшее значение из *a* и *b,* а потом сравнив его и *c,* снова найти наибольшее из двух. Это можно представить такой блок-схемой (рис. ниже)

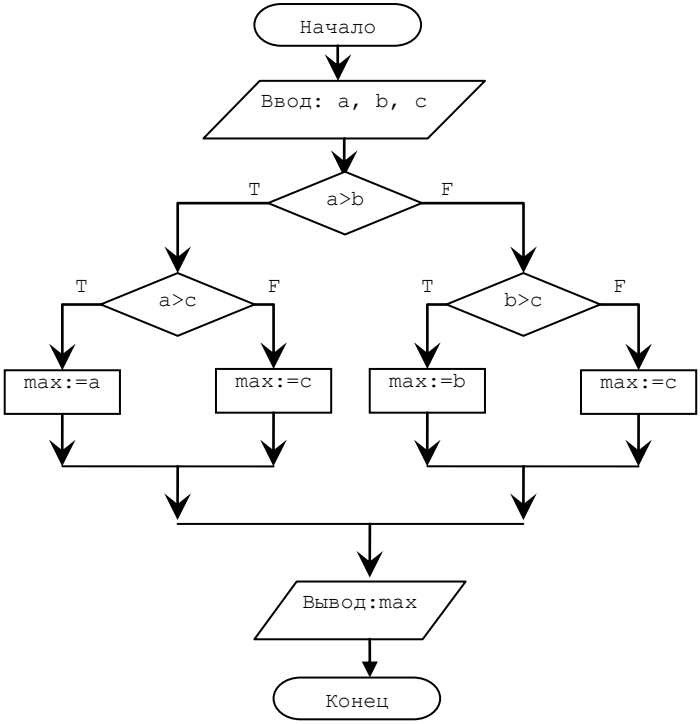


Рисунок 10. Алгоритм нахождения максимума из трех

Тесты для Примера 12				
№ теста	Вход			Выход
	a	b	c	
1.	2	1	3	3
2.	0	-2	-1	0
3.	11	56	29	56
4.	4	2	4	4
5.	3	3	3	3

Пример 13. Составить блок-схему решения следующей задачи. Даны значения действительных переменных b и c. Решить линейное уравнение bx+c=0.

Напомним, что решить уравнение – это найти множество всех его корней. Решение «в лоб» в виде «корень равен –c/b» неверно, так как не учитывает случай, когда b=0, что вполне допустимо условием задачи. Действительно, если b≠0, то существует единственный корень, равный –c/b. Если же b=0, то возможны два случая. Первый: если c≠0, то корней уравнение не имеет (то есть множество корней пусто ∅). Второй: если же c=0, то уравнение не определено, то есть любое действительное число является его нем. Иначе говоря, множество корней уравнения в этом случае – все действительные числа $(-\infty; +\infty)$.

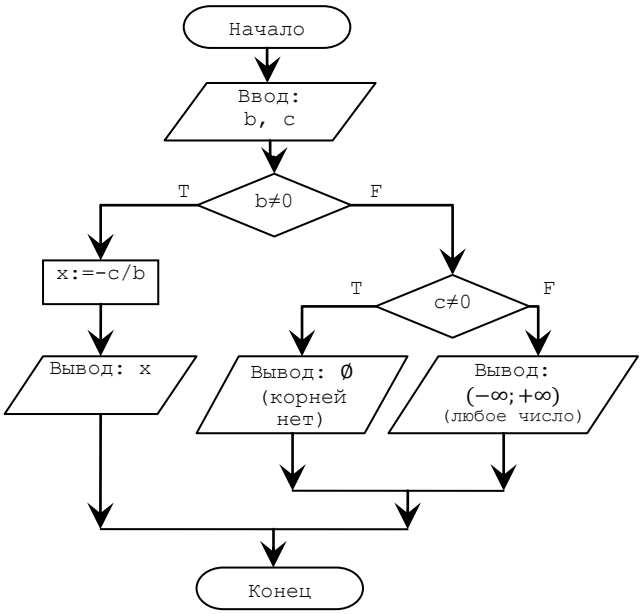


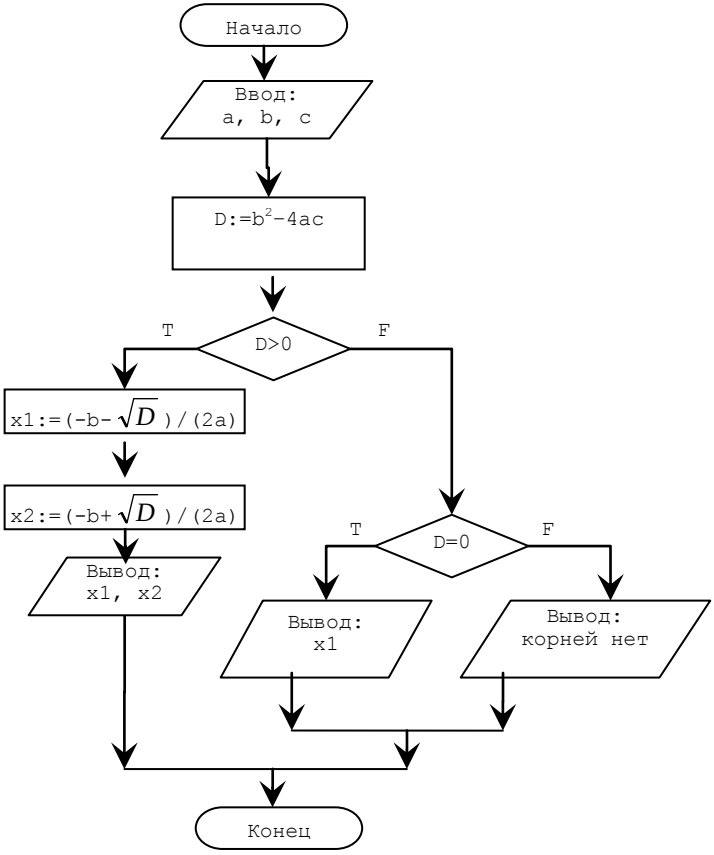
Рисунок 11. Алгоритм полного решения линейного уравнения

Все вышесказанное можно представить в виде блок-схемы алгоритма (рис. 11) и протестировать его на таблице тестов.

Тесты для Примера 13			
№ теста	Вход		Выход
	b	c	
1.	1	-2	2
2.	2	1	-0,5
3.	10	0	0
4.	0	1	∅
5.	0	0	$(-\infty;+\infty)$

Пример 14. Составить блок-схему решения следующей задачи. Даны значения действительных переменных a, b и c, причем a≠0. Решить уравнение $ax^2+bx+c=0$.

Это квадратное уравнение, алгоритм его решения (через дискриминант) известен любому школьнику. Приведем его блок-схему и таблицу тестов. Обращаем внимание, что по условию этой задачи a не может равняться нулю.



Тесты для Примера 14				
№ теста	Вход			Выход
	a	b	c	
1.	1	2	-3	-3; 1
2.	2	8	8	-2
3.	2	-1	4	∅

Пример 15. Составить блок-схему решения следующей задачи. Даны значения действительных переменных a, b и c. Решить уравнение $ax^2+bx+c=0$.

В отличие от предыдущего примера, здесь нет условия *a≠0,* то есть *a* может равняться нулю. Поэтому надо отдельно рассмотреть два случая. В первом случае, если *a≠0*; уравнение является квадратным, алгоритм его решения изложен в примере 14. Во втором случае, если *a=0*; уравнение является линейным, как в примере 13. Таким образом, блок-схема решения этой задачи будет содержать алгоритмы решения квадратного и линейного уравнений. Предлагается составить эту блок-схему самостоятельно. Тесты даны.

Тесты для Примера 15				
№ теста	Вход			Выход
	a	b	c	
1.	1	2	-3	-3; 1
2.	2	8	8	-2
3.	2	-1	4	∅
4.	0	1	-2	2
5.	0	0	1	∅
6.	0	0	0	$(-\infty;+\infty)$

Материал для проверки усвоения темы 2

Контрольные вопросы и задания темы 2

1. Из каких блоков может состоять блок-схема алгоритма?
2. В чем заключается действие алгоритма «присваивание»?
3. Что такое «переменная» алгоритма?
4. Перечислите виды алгоритмических схем.
5. Определите алгоритмическую схему «линейный алгоритм».
6. Определите алгоритмическую схему «алгоритм с ветвлением».
7. Определите алгоритмическую схему «циклический алгоритм». Что такое тело цикла?
8. Какие значения может принимать условие в алгоритме с ветвлением?
9. Могут ли быть исполнены обе ветви ветвления за один проход по алгоритму с ветвлением?

Упражнения по теме 2

1. Постройте блок-схему алгоритма решения следующей задачи: Дано число a. Найти
   а. a^{10} за четыре операции умножения;
   б. a^{36} за шесть операций умножения.
2. Составьте блок-схему алгоритма решения следующей задачи: Даны значения трех вещественных переменных a, b и c. Так обменять их значениями, чтобы переменная a получила бы значение переменной b, переменная b – значение c, а c – значение a.
3. Постройте блок-схему алгоритма решения следующей задачи: Даны три точки вещественной оси a, b и c. Указать две из них, наиболее удаленные друг от друга.
4. Составьте блок-схему алгоритма решения следующей задачи: Даны пять вещественных положительных чисел A, B, C, X, Y. Проверить, сможет ли пройти кирпич размерами A×B×C через прямоугольное отверстие в стене размером X×Y, если ребра кирпича будут параллельны или перпендикулярны сторонам отверстия?

5. *Постройте блок-схему алгоритма решения следующей задачи: Даны три вещественных числа a, b и c (a≠0). Решить биквадратное уравнение $ax^4+bx^2+c=0$.

Задания для самостоятельной работы по теме 2

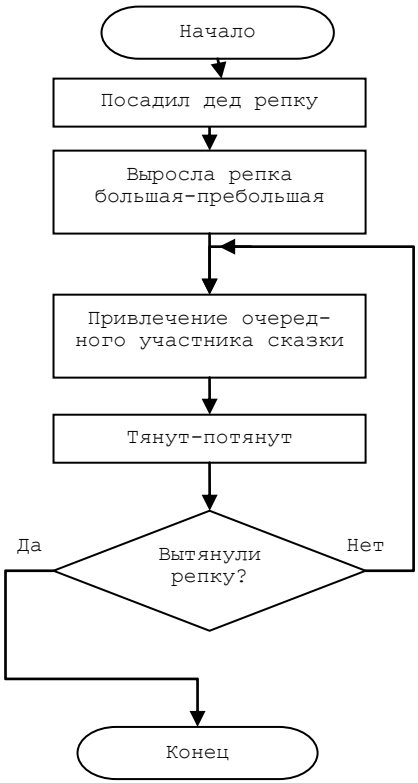
1. Составьте блок-схему алгоритма решения следующей задачи: Даны значения трех вещественных переменных a, b и c, все значения различны. Максимальную и минимальную переменные обменять значениями.
2. *Составьте блок-схему алгоритма: Даны вещественные x и y. Найти наибольшее из этих двух значений, не используя ветвления (то есть построить линейный алгоритм).

Тема 3: Циклические алгоритмы

Теоретический материал

Часто возникают ситуации, когда нужно совершить некоторую последовательность однообразных действий. Например, чтобы забить гвоздь молотком, пока шляпка не будет утоплена, нужно ударять молотком по гвоздю (первый раз, второй раз, третий раз и т.д.). Алгоритмы, основанные на понятии многократного повторения некоторых действий называются циклическими.

Итак, ***циклическим (циклом)*** называется такой алгоритм, в котором задана некоторая последовательность действий для многократного исполнения. Эта последовательность действий называется ***телом цикла***. Заметим, что тело цикла указано в алгоритме один раз, но исполняться оно может многократно. Однократное исполнение тела цикла называется ***итерацией***. Выражение, определяющее будет ли продолжаться исполнение очередной итерации или цикл завершиться, называется ***условием цикла***. В зависимости от того, как интерпретируется условие цикла, оно может быть ***условием продолжения*** либо ***условием окончания*** цикла.



Пример 16. Рассмотрим сказку о репке, «блок-схема» сюжета которой приведена на рисунке. Действия «привлечение очередного участника сказки» и «тянут-потянут» в сказке о репке – это тело цикла. Условием явля-

ется выражение «Вытянули репку». Тело цикла, кстати, в соответствии с первоисточником, было исполнено 6 раз (дед, бабка, внучка, жучка, кошка, мышка), то есть было совершено 6 итераций.

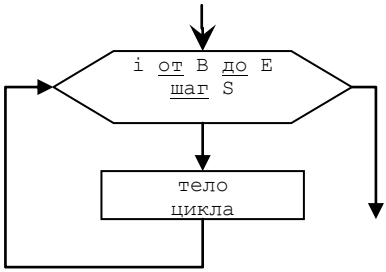
Существуют несколько разных видов циклических алгоритмов, отличающихся друг от друга расположением условия относительно тела, способом выхода из цикла, наличием счетчика итераций и др. Рассмотрим здесь три часто используемые разновидности циклов: цикл со счетчиком; цикл с предусловием; цикл с постусловием.

Цикл со счетчиком

Эта разновидность цикла содержит специальную переменную (***счетчик***, параметр цикла), которая последовательно изменяет свое значение от заданного ***начального*** до ***конечного значения*** с некоторым ***шагом,*** при этом для каждого значения счетчика тело цикла исполняется один раз (то есть происходит итерация). Обычно принято, что сам счетчик во время отдельной итерации не изменяется.

Так как начальное, конечное значения и шаг цикла явно заданы, то можно непосредственно *до исполнения* цикла *заранее* подсчитать количество итераций. Например, если известно, что студент вуза учится с 1 по 4 курс, причем на каждый курс он затрачивает один год, то всего он будет учиться 4-1+1=4 года. Так как количество повторов исполнения тела цикла вычислимо еще до самого процесса, то цикл со счетчиком часто называют ***циклом с заранее известным числом итераций***.

Введем условное обозначение цикла со счетчиком в виде блок-схемы (см. рис.). Здесь `i` – счетчик (параметр) цикла, `B` и `E` – начальное и конечное значения счетчика, `S` – шаг. Чаще всего шаг равен 1, в таком случае он не указывается.



Процесс исполнения такого алгоритма следующий. Вначале счетчик `i` принимает начальное значение `B` и исполняется тело цикла (первая итерация). Затем счетчик изменяется на шаг `S`, происходит вторая итерация и так далее. Когда счетчик достигнет значения `E`,

будет исполнена последняя итерация и цикл закончится. Таким образом, *количество итераций* для целочисленного счетчика с шагом S = 1 может быть вычислено по формуле

$C_1 = E - B + 1$, где $B \le E$,

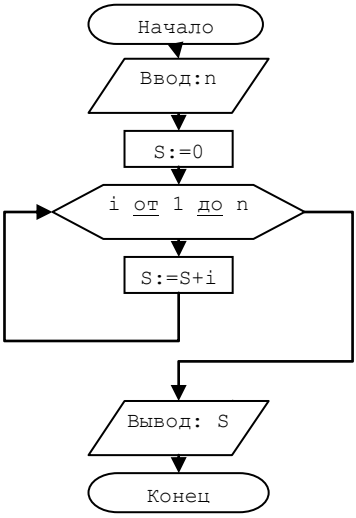
а с шагом S = –1

$C_{-1} = B - E + 1$, где $B \ge E$.

Пример 17. Составить блок-схему решения следующей задачи. Дано натуральное значение n. Найти сумму всех натуральных чисел от 1 до n.

В задаче требуется по заданному натуральному n найти значение 1+2+3+…+n. Составим таблицу тестов.

Тесты для примера 17		
№ теста	Вход	Выход
	n	
1.	1	1
2.	3	6
3.	5	15
4.	100	5050



Как бы мы поступили, если бы у нас в руках был калькулятор и нужно подсчитать, к примеру, сумму 1+2+3+4+5? Наверное, так: сбрасываем до 0 (на экране 0); добавляем 1 (+1) – получили на экране 1; добавляем 2 (+2, получили на экране 3); добавляем 3 (+3, получаем 6); потом +4 (10) и +5 (в итоге 15). А теперь эти же действия представим блок-схемой алгоритма.

Заметим, что в процессе вычисления у нас возникло две последовательности чисел. Первая – 1, 2, 3, 4, 5; это слагаемые искомой суммы. Вторая – 0, 1, 3, 6, 10, 15, это промежуточные суммы, они получаются последователь-

ным прибавлением слагаемых; последняя сумма 15 есть ответ. В алгоритме (см. рис. X) значения слагаемых последовательно принимает переменная i (счетчик цикла), значения сумм – переменная S.

Проделаем по шагам этот алгоритм, процесс исполнения запишем в таблицу решения.

Процесс исполнения теста № 3 для примера 17					
№ шага	n	i	S	Выход	Пояснения
1.	5				Вход: 5
2.			0		Инициализация S нулем
3.		1			Первая итерация при i=1
4.			1		S:=S+i, то есть 0+1
5.		2			Вторая итерация при i=2
6.			3		S:=1+2
7.		3			Третья итерация при i=3
8.			6		S:=3+3
9.		4			Четвертая итерация при i=4
10.			10		S:=6+4
11.		5			Пятая итерация при i=5
12.			15		S:=10+5
13.				15	Выход: 15

Тест алгоритмом пройден, так как для входа n=5 получили ожидавшийся выход 15.

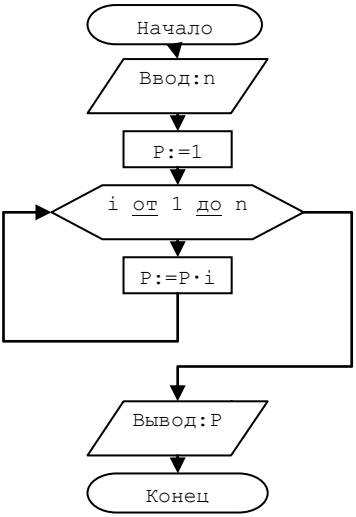
В этом алгоритме был использован прием ***накопления суммы***, который заключается в последовательном *прибавлении* к переменной суммирования S очередных слагаемых. Предварительно S инициализируется *нулем*.

Пример 18. Составить блок-схему решения следующей задачи. *Дано натуральное значение n. Найти n!* .

Значение факториала числа n равно $n!=1\cdot 2\cdot 3\cdot \ \cdot n$. Значит, здесь нужно найти произведение всех натуральных чисел от 1 до n. Поступим здесь аналогично предыдущему примеру – составим алгоритм с ***накоплением произведения***. Переменная, в которой будем накапливать произведение, инициализируется *единицей*, а затем в цикле она *домножается* на очередной множитель. Таблица тестов, алгоритм и протокол решения представлены.

Тесты для примера 18		
№ теста	Вход	Выход
	n	
1.	1	1
2.	3	6
3.	5	120

Процесс исполнения теста № 2 для примера 18				
№ шага	n	i	P	Выход
1.	3			
2.			1	
3.		1		
4.			1	
5.		2		
6.			2	
7.		3		
8.			6	
9.				6

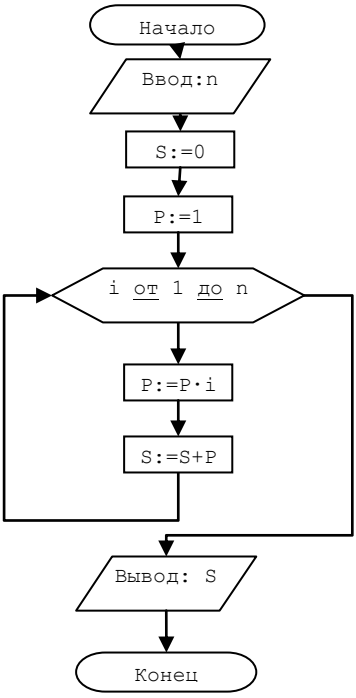


Пример 19. Составить блок-схему решения следующей задачи. Дано натуральное значение n. Найти $\sum_{i=1}^{n} i!$.

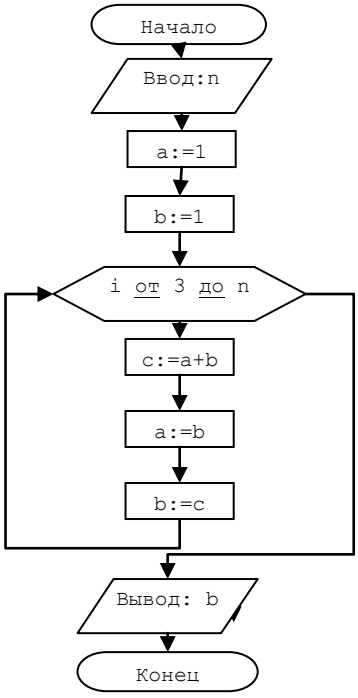
В данной задаче нужно найти значение $1!+2!+\ \ +n!$. Например, для n=3 это значение равно $1!+2!+3!=1+1\cdot 2+1\cdot 2\cdot 3=1+2+6=9$. Таблица тестов приведена ниже.

Тесты для примера 20		
№ теста	Вход	Выход
	n	
1.	1	1
2.	3	9
3.	5	153

С одной стороны, здесь необходимо найти сумму – значит, нужен алгоритм с накоплением суммы. С другой стороны каждое слагаемое – это факториал, то есть произведение. Вычислять каждый факториал накоплением слишком дорого, так как нужно начинать все время с единицы и повторять много уже совершенных действий. Воспользуемся следующим соотношением, а именно – факториал числа i равен факториалу $i-1$, домноженному на i : $i!=(i-1)!\cdot i$ (это верно для $i\geq 1$). Поэтому параллельно накоплению суммы будем накапливать факториал – произведение натуральных чисел. Алгоритм и таблица решения для n=3 приведены. Заметим, что в блок-схеме к переменной S добавляются слагаемые P, так как здесь суммируются факториалы.



Процесс исполнения теста № 2 для примера 19					
№ шага	n	i	P	S	Выход
1.	3				
2.				0	
3.			1		
4.		1			
5.			1		
6.				1	
7.		2			
8.			2		
9.				3	
10.		3			
11.			6		
12.				9	
13.					9



Пример 20. Составить блок-схему решения следующей задачи. Дано натуральное значение n. Найти a_n - n-ое число Фибоначчи.

Числовая последовательность a_i (i =1, 2, 3, …), задаваемая по правилам $a_1 = 1$, $a_2 = 1$, $a_i = a_{i-1} + a_{i-2}$ (для $i = 3,4,5,$) называется последовательностью ***чисел Фибоначчи***. Таким образом, начало последовательности Фибоначчи такое: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

Алгоритм дан на рисунке слева. Заметим, что если n<3, то тело цикла ни разу не будет исполнено, так как начальное значение счетчика будет больше конечного.

Убедитесь самостоятельно, что алгоритм успешно проходит тесты (например, для n=1, 2, 5, 10).

Цикл с предусловием

Цикл с предусловием – это цикл, исполнение которого продолжается, пока истинно условие цикла, проверяемое *до исполнения* очередной итерации. Блок-схема цикла с предусловием приведена на рисунке. Так как при истинности условия цикл продолжается, то условие называют «условием продолжения цикла».

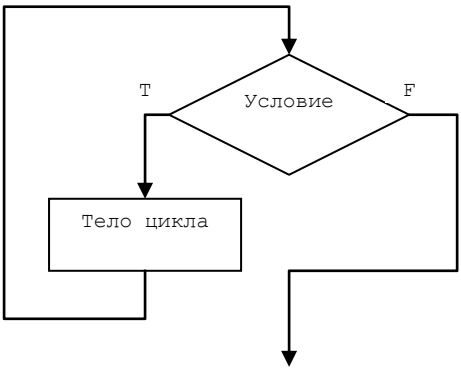


Рисунок 12. Цикл с предусловием

Если окажется, что с самого начала условие цикла ложно (false, F), то тело цикла ни разу не будет исполнено (0 итераций). Если в процессе исполнения цикла условие всегда принимает значение «истина» (true, T), то цикл начинает исполняться бесконечно – происходит ***зацикливание***. Эта ситуация ошибочна, поэтому необходимо, чтобы условие непременно в какой-то момент стало бы ложным. Тогда произойдет корректный выход из цикла.

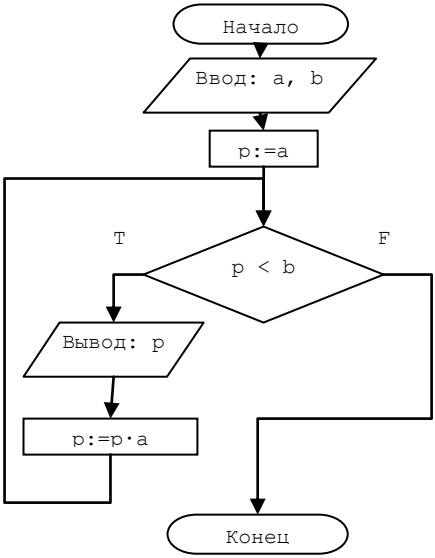
Заметим, что рассмотренный ранее алгоритм Евклида (см. Рисунок 3) содержит цикл с предусловием.

Пример 21. Составить блок-схему решения следующей задачи. Даны действительные числа a и b ($a>1$). Найти все члены последовательности a^1, a^2, a^3 …, меньшие b.

Составим таблицу тестов.

Тесты для примера 21			
№ теста	Вход		Выход
	a	b	

1.	2	10	2; 4; 8
2.	7	50	7; 49
3.	1,5	0	-

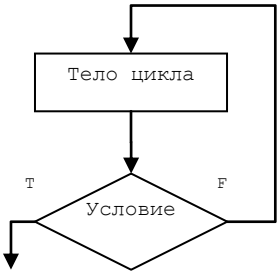


Заметим, что количество выходных значений в этой задаче может быть различным – это зависит от входных данных. В том числе возможна ситуация, когда на выходе нет ни одного числа.

Предлагается такое решение (см. блок-схему на рисунке). Протестируйте алгоритм на предложенных выше тестах.

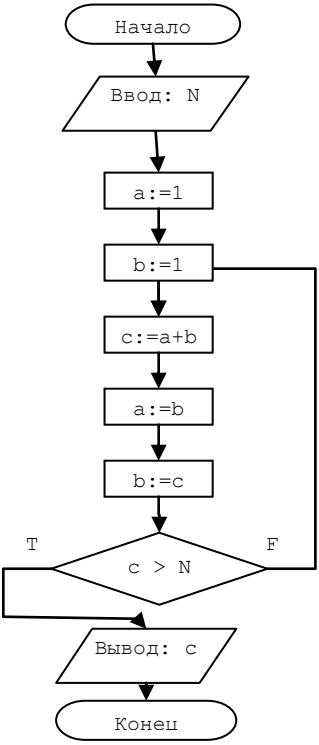
Цикл с постусловием

Цикл с постусловием – это цикл, исполнение которого продолжается, пока ложно условие цикла, проверяемое *после исполнения* тела цикла. Блок-схема цикла с постусловием приведена на рисунке.



Так как при истинности условия цикл прекращается, то постусловие называют «условием окончания цикла».

Из блок-схемы видно, что независимо от начального значения постусловия тело цикла будет исполнено по крайней мере один раз. Также здесь возможна ситуация



зацикливания, когда при исполнении постусловие всегда принимает значение «ложь» (F).

Заметим, что рассмотренный ранее алгоритм сказки о репке содержит цикл с постусловием.

Пример 22. Составить блок-схему решения следующей задачи. Дано натуральное число N. Найти первое из чисел Фибоначчи, большее N.

Определение последовательности чисел Фибоначчи дано в примере 21.

Тесты для примера 22		
№ теста	Вход	Выход
	N	
1.	*1*	2
2.	*10*	13
3.	*100*	144

Построим алгоритм, исходя из следующих соображений. Будем последовательно получать числа Фибоначчи. Каждое вновь полученное число будем сравнивать с N, и, если оно окажется больше N, процесс следует остановить. Так как в этом алгоритме *вначале* получаем число, а только *потом* проверяем, не является ли оно искомым, то разумно использовать цикл с постусловием, когда проверка осуществляется после действия. Блок-схема алгоритма приведена на рисунке.

Материал для проверки усвоения темы 3

Контрольные вопросы и задания темы 3

1. Что такое циклический алгоритм?
2. Укажите типы циклических алгоритмов?
3. В чем отличие цикла с параметром от алгоритмов с условиями?
4. В чем различие цикла с предусловием от цикла с постусловием?
5. Какой из циклических алгоритмов реализует «повторение ноль или более раз»?
6. Какой из циклических алгоритмов реализует «повторение один или более раз»?
7. В чем сходство и разница между ветвлением и циклом с предусловием?
8. Приведите пример из жизни, когда применяется цикл с параметром.
9. Приведите пример из жизни, когда применяется цикл с предусловием.
10. Приведите пример из жизни, когда применяется цикл с постусловием.
11. Допустим, дан некоторый алгоритм с предусловием. Как построить эквивалентный ему алгоритм с постусловием?
12. Допустим, дан некоторый алгоритм с постусловием. Как построить эквивалентный ему алгоритм с предусловием?
13. Допустим, дан некоторый алгоритм с параметром. Как построить эквивалентный ему алгоритм с постусловием?

Упражнения по теме 3

1. Постройте блок-схему алгоритма решения следующей задачи: Даны две точки вещественной оси a, b $(a \le b)$ и натуральное число N. Отрезок $[a, b]$ разбит равномерно точками x_i на N отрезков: $x_0 = a, \dots, x_N = b$. Выдать таблицу значений функции $f(x) = x^2 + 2x - \frac{1}{x}$ в точках x_i.
2. Дано натуральное число n. Найти $\sum_{i=1}^{n} \frac{1+2+\quad+i}{i!}$.

3. Дано натуральное число n. Найти $\prod_{i=1}^{n}\left(\frac{1+2+\quad+i}{3^i}+\frac{2^i}{i!}\right)$
4. Дано натуральное число n. Найти наименьший среди квадратов натуральных чисел, которые больше n.
5. Дано натуральное число n. Найти все пары натуральных чисел a и b, представимые в виде $a^2+b^2=n$.

Задания для самостоятельной работы по теме 3

1. Дано натуральное число n. Найти $\prod_{i=1}^{n}\left(\frac{i}{i+1}-\cos 2^i\right)$
2. Дано натуральное число n. Найти $\sum_{i=1}^{n}\frac{\frac{1}{2}+\frac{1}{3}+\quad+\frac{1}{i+1}}{(2i-1)!}$
3. Из всех чисел, меньших данного натурального числа N найти те, которые равны сумме своих натуральных делителей.
4. Из всех чисел Фибоначчи, не превосходящих данного числа A, найти те, которые являются простыми.
5. Дано натуральное число n. Найти $\prod_{i=1}^{n}\left(\frac{\frac{1}{1}+\frac{1}{2}+\quad+\frac{1}{i}}{1+2+\quad+i}\right)$

Список рекомендуемой литературы

1. Макарова Н.В., Волков В.Б. Информатика: Учеб. для вузов – М. : Финансы и статистика, 2011. – 576 с.
2. Стивенс Р. Алгоритмы. Теория и практическое применение / Род Стивенс. – Москва: Издательство «Э», 2016. – 544 с.

Учебное издание

Анисимов Андрей Евгеньевич

ВВЕДЕНИЕ В АЛГОРИТМИЗАЦИЮ ЗАДАЧ

Учебно-методическое пособие

Авторская редакция

Подписано в печать __.__.201_. Формат 60x84 1/16.
Усл. печ. л. 2,6. Уч.-изд. л. 1,1.
Тираж __ экз. Заказ № ____.

Издательство «Удмуртский университет»
426034, Ижевск, Университетская, д. 1, корп. 4, каб. 207.
Тел./факс: + 7 (3412) 500-295. E-mail: editorial@udsu.ru