Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА
Институт Информационных Технологий
Кафедра Промышленной Информатики

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ

Тема лекции «Понятие алгоритма»

Лектор Каширская Елизавета Натановна (к.т.н., доцент, ФГБОУ ВО "МИРЭА - Российский технологический университет") e-mail: liza.kashirskaya@gmail.com

Лекция № 1

АННОТАЦИЯ ДИСЦИПЛИНЫ

Процедурное программирование

Для студентов 1 курса бакалавриата

всех направлений подготовки

Института информационных технологий
и
Института кибернетики

Примерное содержание курса (будет изменяться в процессе)

1. Алгоритм и его особенности
2. Блок-схемы алгоритмов
3. Алгоритмические языки
4. Трансляторы
5. Команда машинной программы и ее характеристики
6. Режимы работы компьютера
7. Программное обеспечение
8. Язык С++
9. Синтаксис языка С++
10. Структура программы для Microsoft Visual Studio
11. Операции в С++
12. Комбинированные операторы
13. Простая программа на С++
14. Переменные и типы данных в С++
15. Ветвления в С++
16. Циклы в С++
17. Массивы в С++
18. Функции в С++
19. Указатели в С++
20. Динамические массивы в С++
21. Параметры командной строки.

Немного истории

Само слово «алгоритм» происходит от имени хорезмского учёного аль-Хорезми (из древнего государства Хорезм). Около 825 года он написал сочинение *Китаб аль-джебр валь-мукабала* («Книга о сложении и вычитании»), из оригинального названия которого происходит слово «алгебра» (аль-джебр — восполнение). В этой книге впервые дал описание придуманной в Индии позиционной десятичной системы счисления. Персидский оригинал книги не сохранился. Аль-Хорезми сформулировал правила вычислений в новой системе и, вероятно, впервые использовал цифру 0 для обозначения пропущенной позиции в записи числа (её индийское название арабы перевели как *as-sifr* или просто *sifr*, отсюда такие слова, как «цифра» и «шифр»). Приблизительно в это же время индийские цифры начали применять и другие арабские учёные.

Кстати, ранее в русском языке писали «алгорифм».

Немного истории

В первой половине XII века книга аль-Хорезми в латинском переводе проникла в Европу. Переводчик, имя которого до нас не дошло, дал ей название *Algoritmi de numero Indorum* («Алгоритмы о счёте индийском») — таким образом, латинизированное имя среднеазиатского учёного было вынесено в заглавие книги. Сегодня считается, что слово «алгоритм» попало в европейские языки именно благодаря этому переводу. В течение нескольких следующих столетий появилось множество других трудов, посвящённых всё тому же вопросу — обучению искусству счёта с помощью цифр, и все они имели в названии слово *algoritmi* или *algorismi*.

Про аль-Хорезми позднейшие авторы ничего не знали, но поскольку первый перевод книги начинается словами: «Dixit algorizmi: …» («Аль-Хорезми говорил: …»), всё ещё связывали это слово с именем конкретного человека.

Как видите первоначально алгоритм означал искусство счёта с помощью цифр.

Определение алгоритма

Существует несколько определений понятия алгоритма. Приведем два самых распространенных.

Алгоритм – последовательность чётко определенных действий, выполнение которых ведёт к решению задачи.

Алгоритм — это точно определенная последовательность действий, которые необходимо выполнить над исходной информацией, чтобы получить решение задачи.

Алгоритм – это совокупность действий, приводящих к достижению результата за конечное число шагов.

Вообще говоря, первое определение не передает полноты смысла понятия алгоритм. Используемое слово «последовательность» сужает данное понятие, т.к. действия не обязательно должны следовать друг за другом – они могут повторяться или содержать условие.

Свойства алгоритма

1. **Дискретность** (от лат. *discretus* — разделенный, прерывистый) – это разбиение алгоритма на ряд отдельных законченных действий (шагов).
2. **Детерминированность** (от лат. *determinate* — определенность, точность) - алгоритм должен содержать набор точных и понятных указаний, не допускающих неоднозначного толкования.
3. **Конечность** – каждое действие в отдельности и алгоритм в целом должны иметь возможность завершения.
4. **Массовость** – алгоритм должен быть воспроизводимым, пригодным для решения всех задач определенного класса на всем множестве допустимых значений исходных данных.
5. **Результативность** – алгоритм должен давать конкретное конструктивное решение, а не указывать на возможность решения вообще.
6. **Достоверность** – алгоритм должен соответствовать сущности задачи и формировать верные, не допускающие неоднозначного толкования решения. .
7. **Реалистичность** – возможность реализации алгоритма при заданных ограничениях: временных, программных, аппаратных, финансовых.

Основная цель алгоритмизации – составление алгоритмов для ЭВМ с дальнейшим решением задачи на ЭВМ.

Примеры алгоритмов

1. Любой прибор, купленный в магазине, снабжается инструкцией по его использованию. Данная инструкция и является алгоритмом для правильной эксплуатации прибора.
2. Каждый шофер должен знать правила дорожного движения. Правила дорожного движения однозначно регламентируют поведение каждого участника движения. Зная эти правила, шофер должен действовать по определенному алгоритму.
3. Массовый выпуск автомобилей стал возможен только тогда, когда был придуман порядок сборки машины на конвейере. Определенный порядок сборки автомобилей – это набор действий, в результате которых получается автомобиль, то есть алгоритм сборки.
4. Любой рецепт приготовления кулинарного блюда является алгоритмом.
5. Выкройка платья является алгоритмом.

Пример последовательности действий, не являющейся алгоритмом

Задание. Получить наибольшее положительное число, не превосходящее единицу.

Решение. 1) Напишем число 0,9.

2) Припишем справа 9, получим 0,99.

3) Припишем справа 9, получим 0,999.

4) Припишем справа 9, получим 0,9999.

……

Вы можете продолжать это интересное занятие до потери пульса, но описанная процедура алгоритмом не является в силу некорректно поставленной задачи: максимального числа, не превосходящего единицу, просто не существует! Описанная последовательность действий не приведет к получению результата, а значит она не обладает одним из свойств алгоритма – результативностью.

Существует несколько способов задания и записи алгоритмов. На практике наиболее распространены следующие **формы представления алгоритмов**:

- словесная (содержит последовательные этапы алгоритма и описывается в произвольной форме на естественном языке);
- формульная (основана на строго формализованном аналитическом задании необходимых для исполнения действий);
- табличная (подразумевает отображение алгоритма в виде таблиц, использующих аппарат реляционного исчисления и алгебру логики для задания подлежащих исполнению взаимных связей);

- 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.);
- графическая (отображение алгоритмов в виде блок-схем — самый распространенный способ).
- операторная (базируется на использовании для отображения алгоритма условного набора специальных операторов: арифметических, логических, печати и т.д.);
- программная (тексты на языках программирования – код программы); алгоритм, записанный на языке машины, есть программа решения задачи.

Словесный способ записи алгоритмов

Рассмотрим подробно каждый вариант записи алгоритмов на примере следующей задачи. **Требуется найти частное двух чисел.**

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке. Ответ при этом получает человек, который выполняет команды согласно словесной записи.

Пример словесной записи.

Задать два числа, являющиеся делимым и делителем.

Проверить, равняется ли делитель нулю.

Если делитель не равен нулю, то найти частное, записать его в ответ.

Если делитель равен нулю, то в ответ записать «нет решения».

Словесный способ не имеет широкого распространения, так как такие описания: строго не формализуемы; страдают многословностью записей; допускают неоднозначность толкования отдельных предписаний.

Графическая реализация алгоритма представляет собой блок-схему (схему алгоритма). Блок-схема состоит из блоков определенной формы, соединенных стрелками. Ответ при этом получает человек, который выполняет команды согласно блок-схеме. Более подробно о блок-схемах будет рассказано сегодня же, но чуть позже.

Программный способ записи алгоритма

Программная реализация алгоритма – это компьютерная программа, написанная на каком-либо алгоритмическом языке программирования, например, C++, Pascal, Basic и т.д.

Программа состоит из команд определенного языка программирования. Отметим, что одна и та же блок-схема может быть реализована на разных языках программирования. Ответ при этом получает ЭВМ, а не человек. Более подробно о составлении программ на языке программирования C++ мы поговорим в следующем разделе.

Различают три основных вида алгоритмов:

1) линейный алгоритм,
2) разветвляющийся алгоритм,
3) циклический алгоритм.

Линейный алгоритм – это алгоритм, в котором действия выполняются однократно и строго последовательно.

Опишем простой *пример* реализации линейного алгоритма – путь из университета домой.

Словесный способ записи данного алгоритма:

1) выйти из университета на остановку;
2) подождать нужный автобус;
3) сесть на нужный автобус;
4) оплатить проезд;
5) выйти на требуемой остановке;
6) дойти до дома.

Очевидно, что данный пример относится к линейному алгоритму, т.к. все действия следуют одно за другим, без условий и повторений.

Разветвляющийся алгоритм – это алгоритм, в котором в зависимости от условия выполняется либо одна, либо другая последовательность действий.

Опишем простой *пример* реализации разветвляющегося алгоритма – если на улице идет дождь, то необходимо взять зонт, иначе не брать зонт с собой.

Приведенный ранее пример нахождени частного двух чисел также относится к разветвляющемуся алгоритму.

Циклический алгоритм – это алгоритм, команды которого повторяются некое количество раз подряд.

Самый простой пример реализации циклического алгоритма – при чтении книги будут повторяться одни и те же действия: прочитать страницу, перелистнуть и т.д.

Более подробно о линейном, разветвляющемся и циклическом алгоритмах поговорим чуть позже.

Любая задача может быть разбита на элементарные действия. Для любой математической задачи или ситуации из жизни можно составить алгоритм решения. Алгоритм может быть описан словесно, псевдокодом, графически или программно. Задача всегда решается с помощью базовых типов алгоритма – линейного, разветвляющегося или циклического.

1. Что такое алгоритм?
2. В чем состоит задача алгоритмизации?
3. Какими свойствами обладает алгоритм?
4. Какие виды алгоритма бывают?

1. Составьте алгоритмы по походу в магазин за яблоками. Используйте линейный и разветвляющийся алгоритмы. Реализуйте их словесно.

2. Составьте алгоритм приготовления яичницы. Учтите, что в холодильнике может и не оказаться ингредиентов.

3. Составьте алгоритм по нахождению корней квадратного уравнения через дискриминант. Используйте разветвляющийся алгоритм. Реализуйте его псевдокодом.

Блок-схема – это графическая реализация алгоритма.

Блок-схема (схема алгоритма) представляет собой удобный и наглядный способ записи алгоритма.

Блок-схема состоит из функциональных блоков разной формы, связанных между собой стрелками. В каждом блоке описывается одно или несколько действий. Основные виды блоков представлены в таблицах через один слайд.

Любая команда алгоритма записывается в блок-схеме в виде графического элемента – блока, и дополняется словесным описанием. Блоки в блок-схемах соединяются линиями потока информации. Направление потока информации указывается стрелкой. В случае потока информации сверху вниз и слева направо стрелку ставить не обязательно. Блоки в блок-схеме имеют только один вход и один выход (за исключением логического блока – блока с условием).

Стандарты на изображение блок-схем

Далее приводятся разные маркировки одного и того же стандарта.

ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения

Международный стандарт **ISO 5807:1985**.

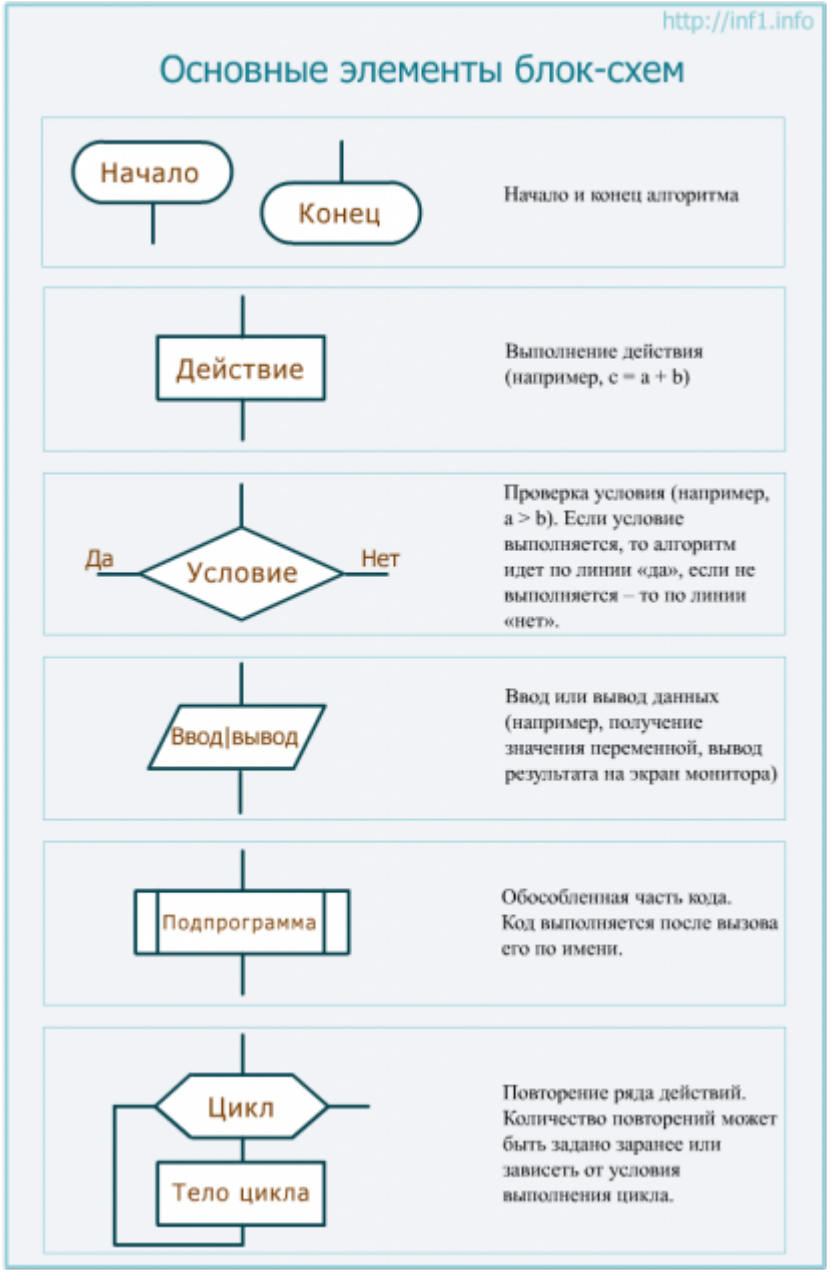
ГОСТ 19.701–90 (ИСО 5807–85) «Единая система программной документации».

Еще материалы:
http://cert.obninsk.ru/gost/282/282.html
https://prog-cpp.ru/block-schema/
https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema8/tema8_2

Элементы блок-схем

Обозначение блока	Название блока
	процесс
	альтернативный процесс
	решение
	данные
	типовой процесс
	внутренняя память
	документ
	несколько документов
	пред. завершение
	подготовка
	ручной ввод
	ручное управление
	узел
	ссылка на другую страницу
	карточка
	перфолента
	узел суммирования
	ИЛИ
	сопоставление
	сортировка
	извлечение
	объединение
	сохраненные данные
	задержка
	память с посл. доступом
	магнитный диск
	память с прямым доступом
	дисплей

Основные элементы блок-схем



Блок начала блок-схемы имеет один выход и не имеет входов, блок конца блок-схемы имеет один вход и не имеет выходов. Блок условия – единственный блок, имеющий два выхода, т.к. соответствует разветвляющемуся алгоритму. На одном выходе указывается «да», на другом – «нет». Все остальные блоки имеют один вход и один выход. Блок выполнения действия может содержать присвоение значения переменной (например, "x = 5") или вычисление (например, "$y = x$-4").

Математические выражения и логические высказывания должны быть описаны математическим языком, т.к. блок-схема не должна иметь привязки к какому-то определенному языку программирования. Одна и также блок-схема может быть реализована в программах на разных языках программирования. К примеру, функция в блок-схеме будет выглядеть таким образом: $y = x^2$, а не таким образом: $y = x$^2.

Все три вида алгоритмов реализуются в блок-схеме названными выше типами блоков. К примеру, в линейном алгоритме могут присутствовать все блоки, кроме блока условия. В разветвляющемся и циклическом алгоритмах могут быть использованы все названные виды блоков, но обязательным является блок условия. Внутри блока условия записывается условие, про которое можно однозначно ответить, истинно оно или ложно. Если условие истинно, то выполняются действия, соответствующие стрелке «да», иначе - стрелке «нет».

Нужны ли блок-схемы?

Частные конторы никакие блок-схемы не используют, в книжках по алгоритмам вместо них применяют словесное описание. Возможно блок-схемы применяют на государственных предприятиях, которые должны оформлять документацию согласно требованиям *ЕСПД*, но есть сомнения — даже для регистрации программы в Государственном реестре программ для ЭВМ никаких блок-схем не требуется.

Тем не менее, рисовать блок-схемы заставляют школьников (примеры из учебников ГОСТ не соответствуют) — выносят вопросы на государственные экзамены (ГИА и ЕГЭ), студентов — перед защитой диплом сдается на нормоконтроль, где проверяется соответствие схем стандартам.

Нужны ли блок-схемы?

Разработка блок-схем выполняется на этапах проектирования и документирования, согласно каскадной модели разработки ПО, которая сейчас почти не применяется, т.к. сопровождается большими рисками, связанными с ошибками на этапах проектирования.

Появляются подозрения, что система образования прогнила и отстала лет на 20, однако аналогичная проблема наблюдается и за рубежом. Международный стандарт *ISO 5807:1985* мало чем отличается от *ГОСТ 19.701-90*, более нового стандарта за рубежом нет. Производится множество программ для выполнения этих самых схем — Dia, MS Visio, yEd, **Diagram Designer 1.23.6** (очень хороший построитель, автоматически генерирующий программный код по блок-схеме), а значит списывать их в утиль не собираются. Вместо блок-схем иногда применяют диаграммы деятельности *UML*, однако удобнее они оказываются разве что при изображении параллельных алгоритмов.

Нужны ли блок-схемы?

Периодически поднимается вопрос о том, что ни *блок-схемы*, ни *UML* не нужны, да и документация тоже не нужна. Об этом твердят программисты, придерживающиеся методологии *экстремального программирования (XP)*, ходя даже в их кругу нет единого мнения.

В ряде случаев, программирование невозможно без рисования блок-схем, т.к. это один процесс — существуют *визуальные языки программирования,* такие как *ДРАКОН*, кроме того, блок-схемы используются для верификации алгоритмов (формального доказательства их корректности).

В общем, единого мнения нет. Очевидно, есть области, в которых без чего-то типа блок-схем обойтись нельзя, но более гибкой альтернативы нет. Для формальной верификации необходимо рисовать подробные блок-схемы, но для проектирования и документирования такие схемы не нужны — я считаю разумным утверждение экстремальных программистов о том, что *нужно рисовать лишь те схемы, которые помогают в работе*.

Приведем простейшие примеры, соответствующие линейному алгоритму.

Пример. Вася хочет позвонить Пете по городскому телефону. Необходимо составить блок-схему, описывающую порядок действий Васи.

Решение. Чтобы позвонить по городскому телефону, нужно знать номер Пети. Значит, сначала надо найти номер телефона Пети, набрать его и поговорить с Петей. На этом цель Васи (поговорить с Петей по телефону) будет достигнута. Результат блок схемы представлен на рисунке.

Графическая реализация линейного алгоритма

Пример. Ученику требуется купить учебник. Составить блок-схему, описывающую порядок действий ученика.

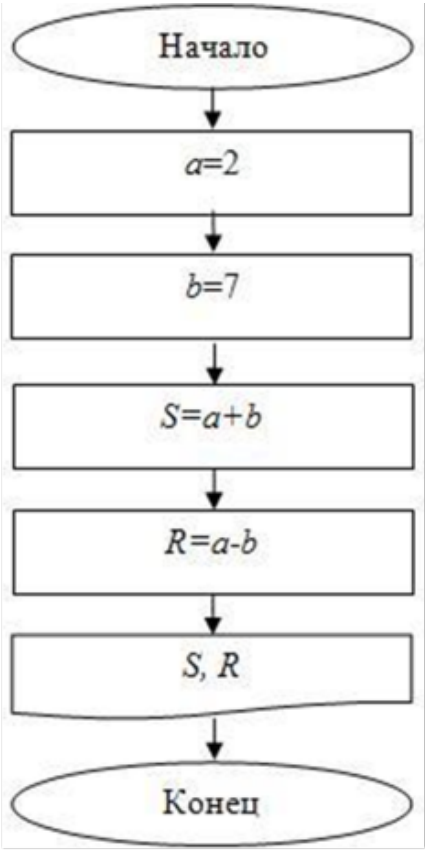
Решение. Сначала ученику нужно взять деньги, потом прийти в книжный магазин и заплатить за учебник. На этом цель (покупка учебника) будет достигнута. Результат блок схемы представлен на рисунке.

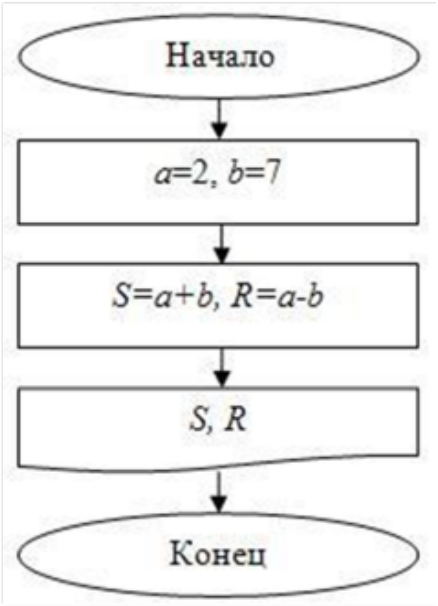
Графическая реализация линейного алгоритма

Пример. Даны числа $a = 2$, $b = 7$. Вычислить сумму S и разность R чисел a и b.

Решение. Сначала следует задать значения для чисел a и b, согласно условиям задачи. После этого их уже можно будет использовать в расчетах для получения суммы и разности по формулам: $S = a + b$, $R = a - b$. Полученные значения суммы и разности нужно будет напечатать, и мы используем блок вывода данных. Если не выводить данные на бумагу (или на экран), то пользователь нашего алгоритма не узнает, какие получились значения суммы и разности. Результат построения блок-схемы представлен на рисунке.

Графическая реализация линейного алгоритма



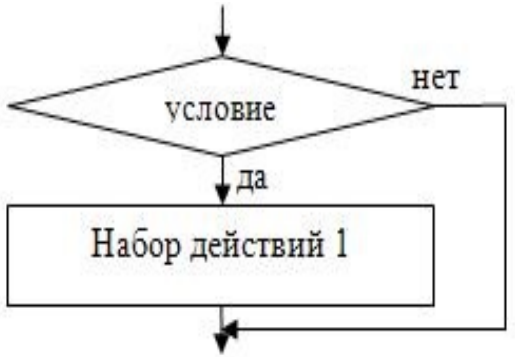


Здесь слева: в каждом блоке по одному действию, справа: действия объединены по смыслу операции. В дальнейшем мы будем объединять некоторые действия в один блок. Это очень удобно и визуально упрощает чтение блок-схемы.

В разветвляющемся алгоритме обязательным блоком является блок условия, который представлен на рисунке.

Внутри блока условия записывается условие. Если данное условие верно, то выполняются блоки, идущие по стрелке «да», т.е. «Набор действий 1». Если условие оказывается неверным, т.е. ложным, то выполняются блоки, идущие по стрелке «нет», а именно «Набор действий 2». Разветвление заканчивается, когда обе стрелки («да» и «нет») соединяются.

На следующем рисунке представлен еще один вариант использования блока условия. Бывают задачи, в которых, исходя из условия, необходимо либо выполнить действие, либо пропустить его.



Если условие выполняется, то следуют блоки, соответствующие стрелке «да», т.е. «Набор действий 1». Если же условие оказывается ложным, то следует перейти по стрелке «нет». Т.к. стрелке «нет» не соответствует ни одного блока с действием, то ни одного действия не будет выполнено. Т.е. получается, что мы пропустили и не выполнили «Набор действий 1».

В разветвляющемся алгоритме возможна запись сразу нескольких условий, которые могут объединяться союзом «ИЛИ» или пересекаться союзом «И». Рассмотрим случай двух условий: «условие 1» и «условие 2».

Если необходимо, чтобы оба условия были верными одновременно, то следует использовать логическое пересечение «И»:

«условие 1 И условие 2».

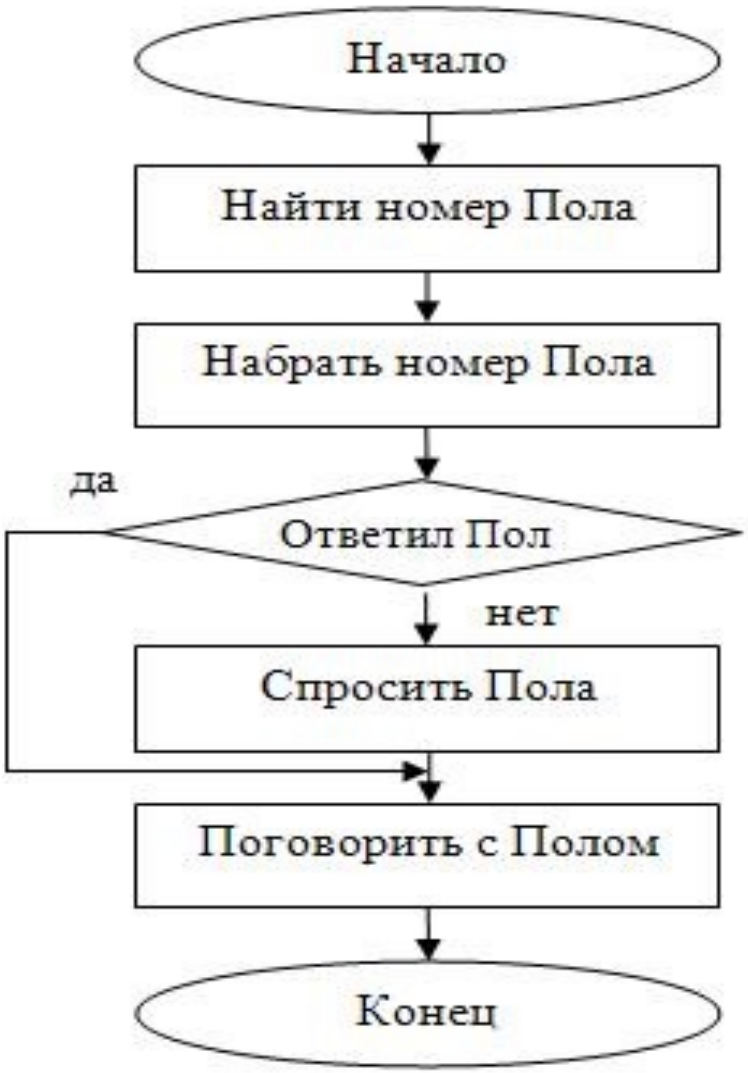
Если достаточно, чтобы только одно условие выполнялось – или первое, или второе, но можно и оба вместе, то следует использовать логическое объединение «ИЛИ»:

«условие 1 ИЛИ условие 2».

Пример. Джон звонит Полу по городскому телефону, но трубку может взять не только Пол. Составить блок-схему, описывающую действия Джона в этом случае.

Решение. Здесь, в отличие от примера Васи с Петей, присутствует условие – Пол ли взял трубку телефона. На данное условие можно однозначно ответить: «да», Пол, или «нет», кто-то другой. Если трубку взял Пол, то Джону нужно с ним поговорить, и цель будет достигнута. Если трубку взял кто-то другой, то необходимо позвать Пола к телефону, поговорить с ним, и цель также будет достигнута. Третьего варианта, например, «не туда попали» или «его нет дома» мы не рассматриваем. Результат блок-схемы представлен на рисунке.

Примеры разветвляющихся алгоритмов



Пример разветвляющегося алгоритма

Пример. Ученику требуется купить учебник. В магазине в наличие оказался нужный учебник в жесткой и мягкой обложке. Составить блок-схему, описывающую действия ученика, которому нужен учебник в жесткой обложке.

Решение. В данном примере присутствует условие: «Нужна жесткая обложка».

Ученик может согласиться с этим высказыванием, тогда он выполнит действие, соответствующее стрелке «да» и купит учебник в жесткой обложке.

Если ученик не соглашается с данным условием, то будет выполняться действие, соответствующее стрелке «нет», и в этом случае ученик купит учебник в мягкой обложке.

И в том, и в другом случае, цель будет достигнута и задача будет выполнена, т.к. ученик купит учебник. Результат блок схемы представлен на рисунке.

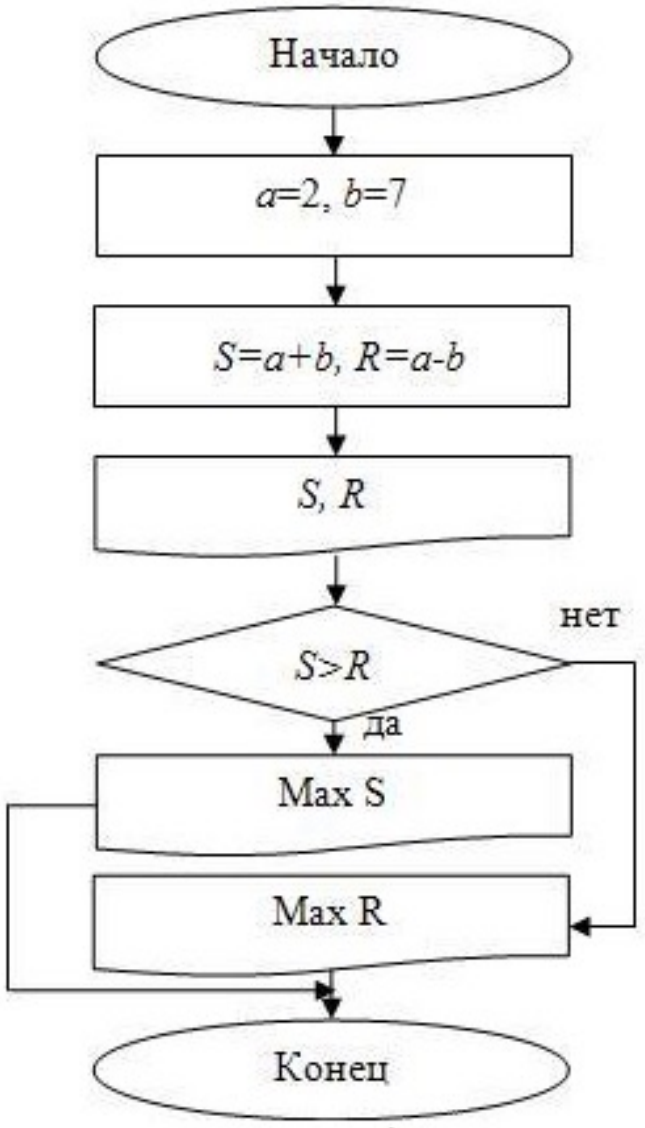
Графическая реализация разветвляющегося алгоритма

Пример разветвляющегося алгоритма

Пример. Даны числа $a = 2$, $b = 7$. Вычислить сумму S и разность R чисел a и b. Сравнить полученные значения S и R и указать большее из них.

Решение. Как и в предыдущем примере, сначала необходимо задать значения a и b. Затем рассчитать сумму и разность по формулам: $S = a + b$, $R = a - b$ и вывести полученные числа (блок вывода данных). Когда значения S и R будут получены, следует сравнить их между собой. Условие запишется в виде: $S>R$. Если полученная сумма S будет больше разности R, то мы пойдем по стрелке «да» и выведем фразу «max S». Если же условие окажется ложным (т.е. $S<R$), то пойдем по стрелке «нет» и выведем фразу «max R». Результат блок схемы представлен на рисунке.

Графическая реализация разветвляющегося алгоритма



В рассмотрении циклического алгоритма следует выделить несколько понятий.

Тело цикла – это набор инструкций, предназначенный для многократного выполнения.

Итерация – это единичное выполнение тела цикла.

Переменная цикла – это величина, изменяющаяся на каждой итерации цикла.

Каждый цикл должен содержать следующие необходимые элементы:

1) первоначальное задание переменной цикла,

2) проверку условия,

3) выполнение тела цикла,

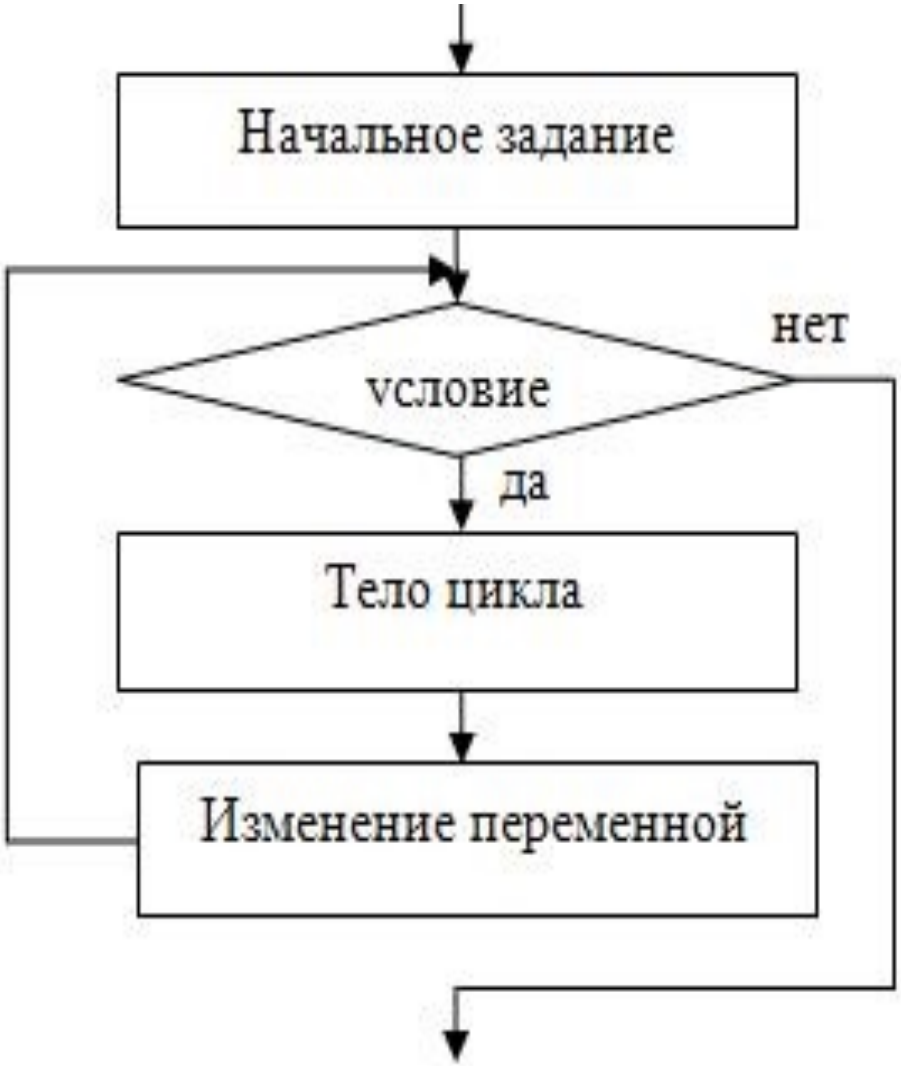
4) изменение переменной цикла.

Циклы бывают нескольких видов – с параметром, с предусловием, с постусловием.

Цикл с параметром предписывает выполнять действия, которые содержит функциональный блок «тело цикла», для всех значений некоторой переменной «параметра цикла» в заданном диапазоне от начального значения до конечного значения с заданным шагом.

В **цикле с предусловием** сначала проверяется условие входа в цикл, а затем выполняется тело цикла, если условие верно. Цикл с предусловием представлен на рисунке. Цикл с предусловием также может быть задан с помощью счетчика. Это удобно в тех случаях, когда точно известно количество итераций. В общем виде блок-схема, реализующая цикл с предусловием, представлена ниже. Сначала задается начальное значение переменной цикла, затем условие входа в цикл, тело цикла и изменение переменной цикла. Выход из цикла осуществляется в момент проверки условия входа в цикл, когда оно не выполняется, т.е. условие ложно. Цикл с предусловием может ни разу не выполниться, если при первой проверке условия входа в цикл оно оказывается ложным.

Графическая реализация цикла с предусловием



В цикле с постусловием сначала выполняется тело цикла, а потом проверяется условие.

Цикл с постусловием

Если условие верно, то итерация повторяется, если же неверно, то осуществляется выход из цикла. В отличие от цикла с предусловием, любой цикл с постусловием всегда выполнится хотя бы один раз.

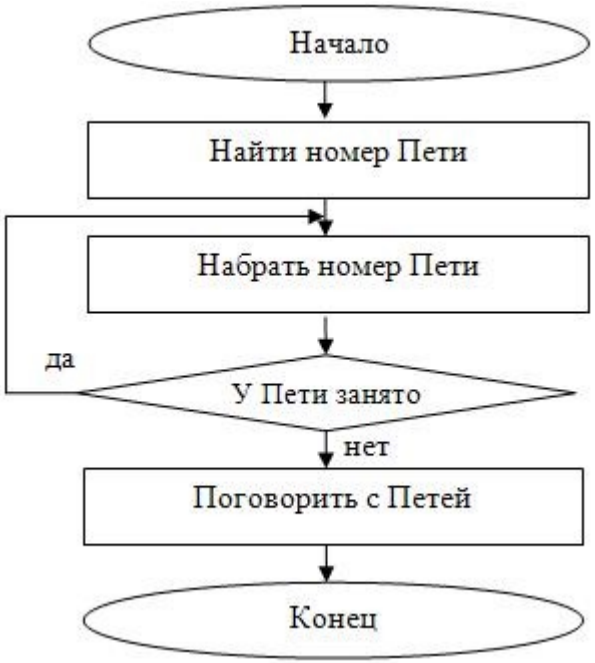
Как видно из представленных блок-схем для циклов с предусловием и постусловием, условие записывается внутри блока условия (формы ромба), как и в разветвляющемся алгоритме. Принципиальная разница между разветвляющимся и циклическим алгоритмами при графической реализации состоит в том, что в циклическом алгоритме в обязательном порядке присутствует стрелка, идущая наверх. Именно эта стрелка обеспечивает многократный повтор тела цикла.

Приведем простейшие примеры, соответствующие циклическому алгоритму.

Пример цикла с постусловием

Пример. Вася звонит Пете, но у Пети может быть занята линия. Составить блок-схему действий Васи в этом случае.

Решение. Когда телефонная линия занята, то необходимо снова и снова набирать номер, пока Петя не закончит предыдущий разговор, и телефонная линия не окажется вновь свободной. Блок-схема представлена на рисунке.



Пример цикла с постусловием

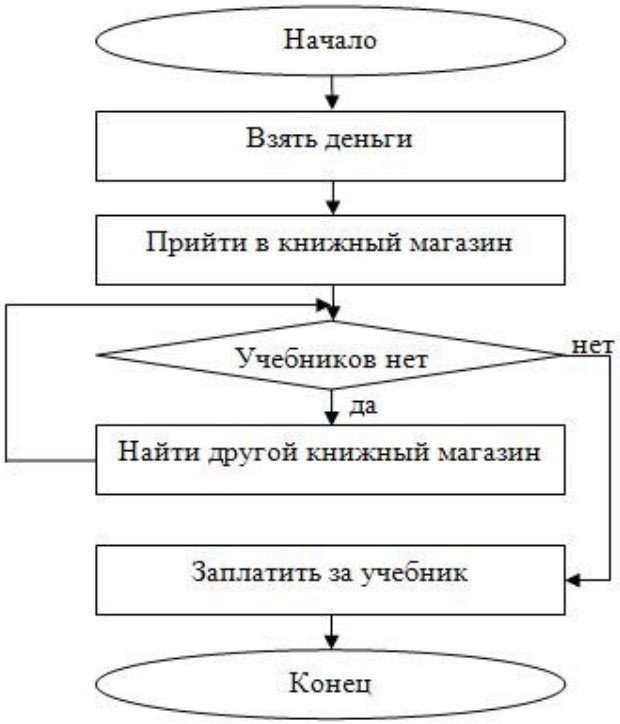
Здесь тело цикла состоит из одного действия «Набрать номер Пети», т.к. именно это действие следует повторять, пока линия будет занята. Под итерацией цикла понимается очередная попытка дозвониться до Пети. Как таковой переменной цикла здесь нет, т.к. ситуация взята из жизни. Выход из цикла происходит в тот момент, когда условие «У Пети занято» стало неверным, т.е. телефонная линия свободна – действительно нет необходимости больше набирать номер Пети. В данном примере применен цикл с постусловием, т.к. сначала необходимо набрать номер Пети, ведь иначе мы не можем ответить на вопрос – занята ли линия у Пети.

Пример цикла с предусловием

Пример. Ученику требуется купить учебник. Составить блок-схему, описывающую действия ученика в случае, если учебника нет в ряде магазинов.

Решение. Действия ученика в данном примере очевидны: когда он приходит в первый и любой последующий магазины, то возможны два варианта – учебник имеется в наличии или учебника нет в продаже. Если учебника нет в продаже, то ученику следует пойти в другой книжный магазин и спросить данный учебник, и т.д. пока учебник не будет куплен, т.к. перед учеником стоит конечная цель – купить учебник. Мы будем использовать цикл с предусловием, т.к. сначала требуется найти магазин, имеющий в наличии данный учебник. Цикл будет выполняться, пока условие «В данном магазине нет учебника» будет верным, а выход из цикла осуществится, когда условие станет ложным, т.е. когда ученик придет в магазин, в котором есть данный учебник. Действительно, в этом случае ученик купит нужный ему учебник и не будет больше искать книжные магазины. Результат в виде блок-схемы представлен на рисунке.

Пример цикла с предусловием



Здесь тело цикла состоит из одного действия «Найти другой книжный магазин». Переменной цикла в явном виде нет, но можно подразумевать номер магазина, в который пришел ученик в очередной раз. Как любой другой цикл с предусловием, данный цикл может ни разу не выполниться (не иметь итераций), если в первом же магазине окажется нужный учебник.

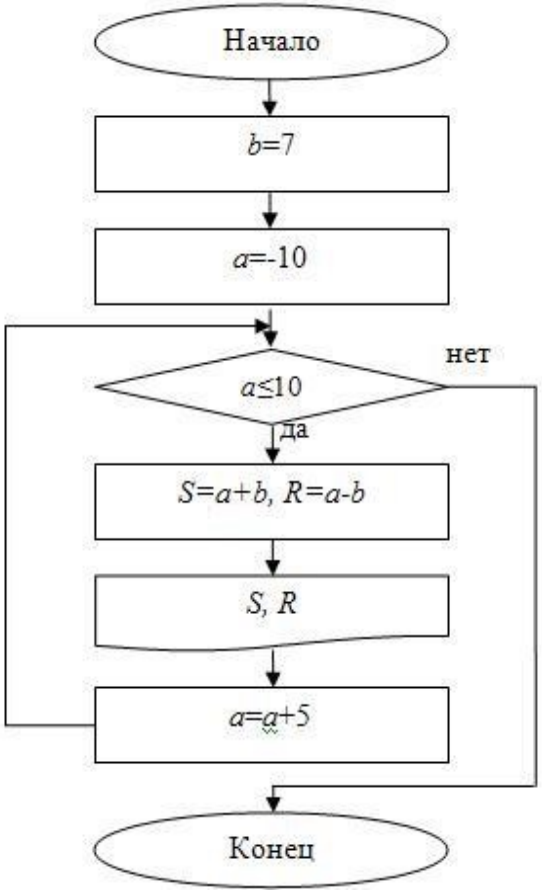
Примечание. Если в данную задачу добавить условие выбора учебника в жесткой или мягкой обложке, то оно появится после выхода из цикла. На реализацию циклического алгоритма данное условие не повлияет.

Пример цикла с предусловием

Пример. Даны числа *a, b*. Известно, что число *a* меняется от -10 до 10 с шагом 5, *b* = *7* и не изменяется. Вычислить сумму *S* и разность *R* чисел *a* и *b* для всех значений *a* и *b*.

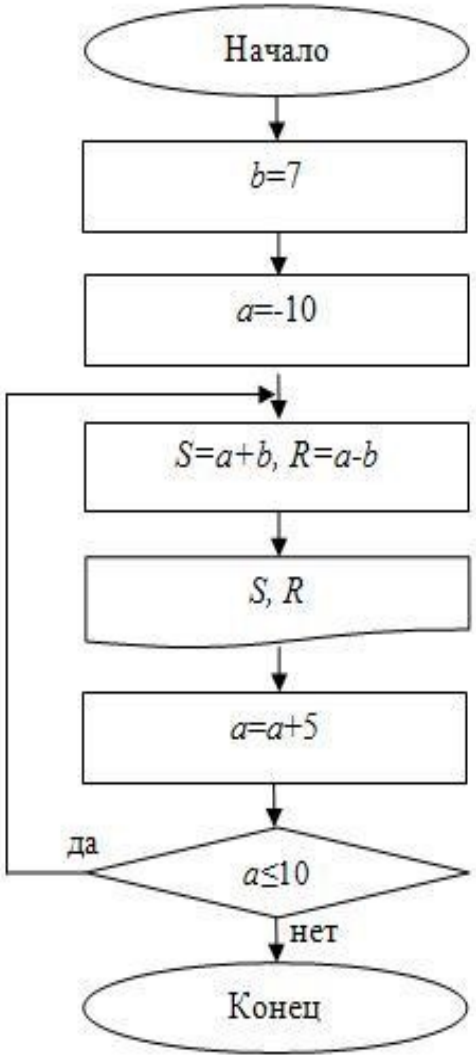
Решение. Здесь число *a* меняется от -10 до 10 с шагом 5. Это означает, что число *a*является переменной цикла. Сначала *a* равно -10 – это первоначальное задание переменной цикла. Далее *a* будет изменяться с шагом 5, пока не будет достигнуто значение 10 – это соответствует изменению переменной цикла. Итерации надо повторять, пока выполняется условие $a \leq 10$. Итак, *a* будет принимать следующие значения: -10, -5, 0, 5, 10. Результат в виде блок-схемы цикла с предусловием представлен на рисунке.

Пример цикла с предусловием



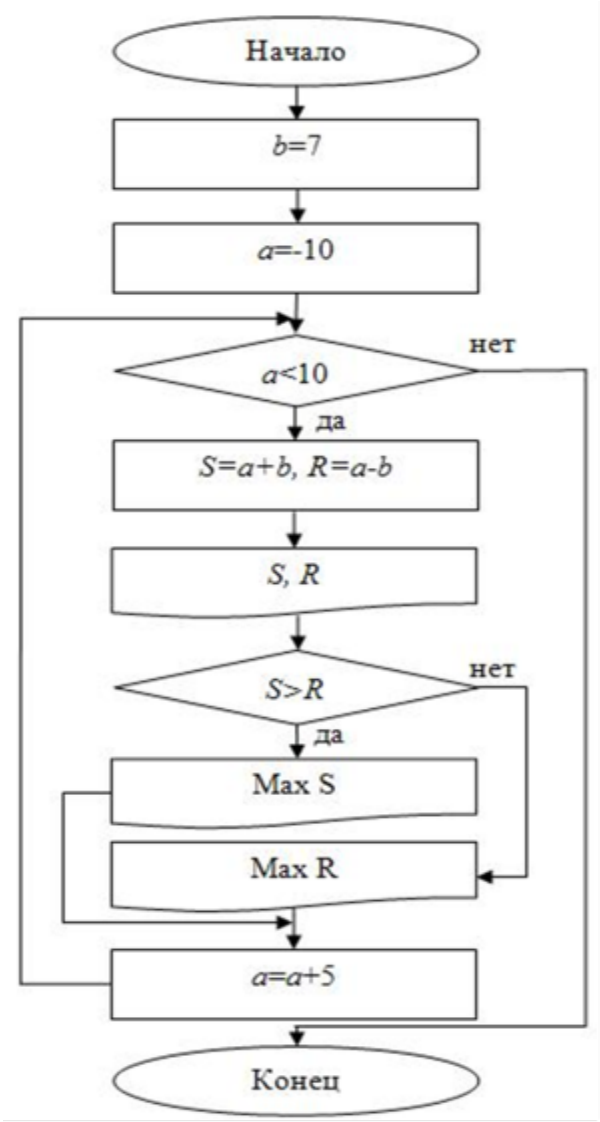
Тело цикла состоит из нескольких действий: вычисление суммы, вычисление разности и вывод полученных данных. Таким образом, у нас получится несколько значений сумм и разностей, т.к. *a* изменяется. Количество сумм и количество разностей совпадет с количеством различных значений *a*, т.е. пять.

Данная задача может быть сделана с циклом с предусловием и с циклом с постусловием. В этом случае тело цикла, условие и изменение переменной цикла будут такими же, как и в цикле с предусловием, но сначала необходимо выполнить тело цикла, а потом проверить условие для выполнения следующей итерации. Приведем блок-схему, использующую цикл с постусловием.

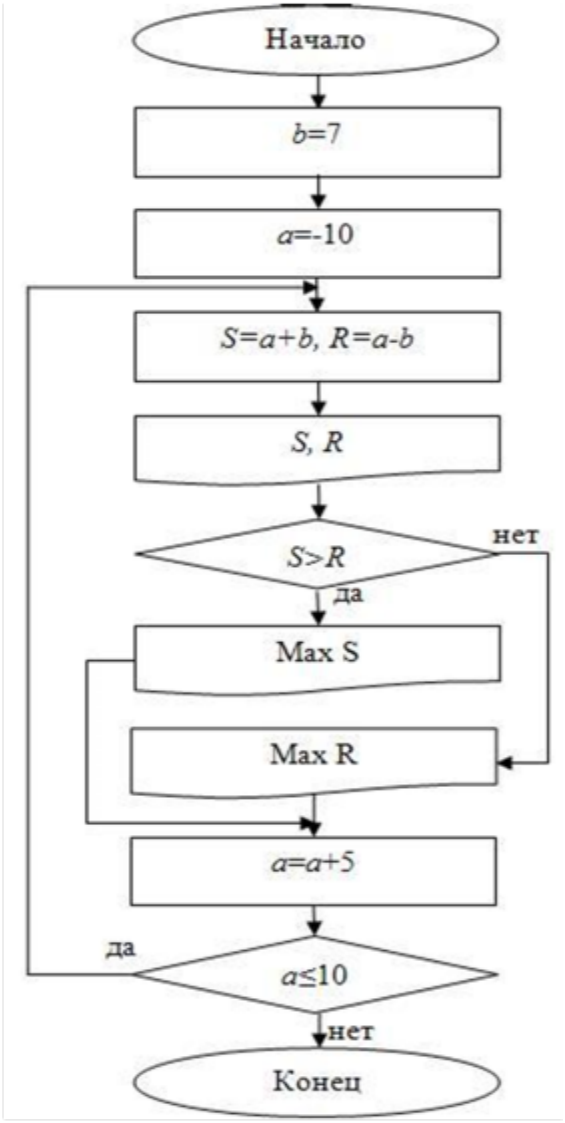


В данной задаче также могут быть соединены циклический и разветвляющийся алгоритмы, если по условию задачи требуется сравнить полученные значения суммы и разности. В этом случае цикл можно реализовать как с предусловием, так и с постусловием, а сравнение суммы и разности добавится внутрь тела цикла, т.к. следует сравнить между собой все полученные суммы и разности. Организация самого цикла останется прежней. Приведем на рисунках блок-схемы циклов с предусловием и с постусловием.

Решение в виде циклов с условием



а)



б)

Программирование – это написание текста на алгоритмическом языке.

Цель программирования – описание процесса обработки данных.

Система программирования – это язык и средства разработки программ.

В состав интегрированной среды разработки программ входят:

- текстовый редактор;
- транслятор;
- редактор связей;
- библиотеки подпрограмм;
- программа-отладчик;
- система помощи и подсказок.

Язык программирования (ЯП) — формальная знаковая система, предназначенная для описания алгоритмов в форме, которая удобна для исполнителя (например, компьютера).

ЯП определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы.
Он позволяет программисту точно определить то, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими данными при различных обстоятельствах.

Общие особенности языков программирования

- ЯП предназначен для написания компьютерных программ, которые применяются для передачи компьютеру инструкций по выполнению вычислительного процесса и организации управления отдельными устройствами.
- ЯП отличается от естественных языков, тем что предназначен для передачи команд и данных от человека компьютеру, в то время, как естественные языки используются лишь для общения людей между собой.
- ЯП может использовать специальные конструкции для определения и манипулирования структурами данных и управления процессом вычислений.

Алгоритмические языки

Виды алгоритмических языков: машинно-ориентированные, процедурно-ориентированные, проблемно-ориентированные.

Машинно-ориентированные языки программирования низкого уровня — программирование на них наиболее трудоемко, но позволяет создавать оптимальные программы, максимально учитывающие функционально-структурные особенности конкретного компьютера.

Процедурно-ориентированные и проблемно-ориентированные языки относятся к языкам высокого уровня, использующим макрокоманды. Макрокоманда при трансляции генерирует много машинных команд.

Алгоритмические языки

Языки программирования делятся на два класса — компилируемые и интерпретируемые.

- Программа на компилируемом языке при помощи специальной программы-компилятора преобразуется (компилируется) в набор инструкций для данного типа процессора (машинный код) и далее записывается в исполняемый файл, который может быть запущен на выполнение как отдельная программа (компилятор переводит программу с языка высокого уровня на низкоуровневый язык, понятный процессору).

- Программа на интерпретируемом языке: интерпретатор непосредственно выполняет (интерпретирует) ее текст без предварительного перевода. При этом программа остается на исходном языке и не может быть запущена без интерпретатора.

Трансляторы – это программы, которые переводят текст с языка программирования на машинный язык.

Трансляторы бывают двух видов: трансляторы-компиляторы и трансляторы-интерпретаторы

КОМПИЛЯТОРЫ при трансляции переводят на машинный язык сразу всю программу и затем хранят ее в памяти машины в двоичных кодах.

ИНТЕРПРЕТАТОРЫ каждый раз при исполнении программы заново преобразуют в машинные коды каждую макрокоманду и передают ее для непосредственного выполнения компьютеру.

Компилятор переводит программу на машинный язык сразу и целиком, создавая при этом отдельную программу, а интерпретатор переводит команды программы на машинный язык прямо во время исполнения программы.

Разделение на компилируемые и интерпретируемые языки является условным:

- для любого интерпретируемого языка можно создать компилятор;
- создаваемый во время исполнения программы код может так же динамически компилироваться во время исполнения.

Особенности скомпилированных программ

- выполняются быстрее и не требуют для выполнения дополнительных программ, так как уже переведены на машинный язык;
- при каждом изменении текста программы требуется ее перекомпиляция;
- скомпилированная программа может выполняться только на том же типе компьютеров и, как правило, под той же операционной системой, на которую был рассчитан компилятор.

Особенности интерпретируемых программ

- интерпретируемые программы выполняются заметно медленнее, чем компилируемые;
- программы можно запускать сразу же после изменения, что облегчает разработку;
- программа на интерпретируемом языке может быть зачастую запущена на разных типах машин и операционных систем без дополнительных усилий;
- не могут выполняться без дополнительной программы-интерпретатора.

Некоторые языки, например, Java и C#, находятся между компилируемыми и интерпретируемыми: программа компилируется не в машинный язык, а в машинно-независимый код низкого уровня, байт-код, который выполняется виртуальной машиной.

Для выполнения байт-кода обычно используется интерпретация, хотя отдельные его части для ускорения работы программы могут быть транслированы в машинный код непосредственно во время выполнения программы по технологии компиляции «на лету» (Just-in-time compilation, JIT).

Классификация языков программирования по близости к естественному языку

- Языки программирования высокого уровня
- Языки программирования низкого уровня

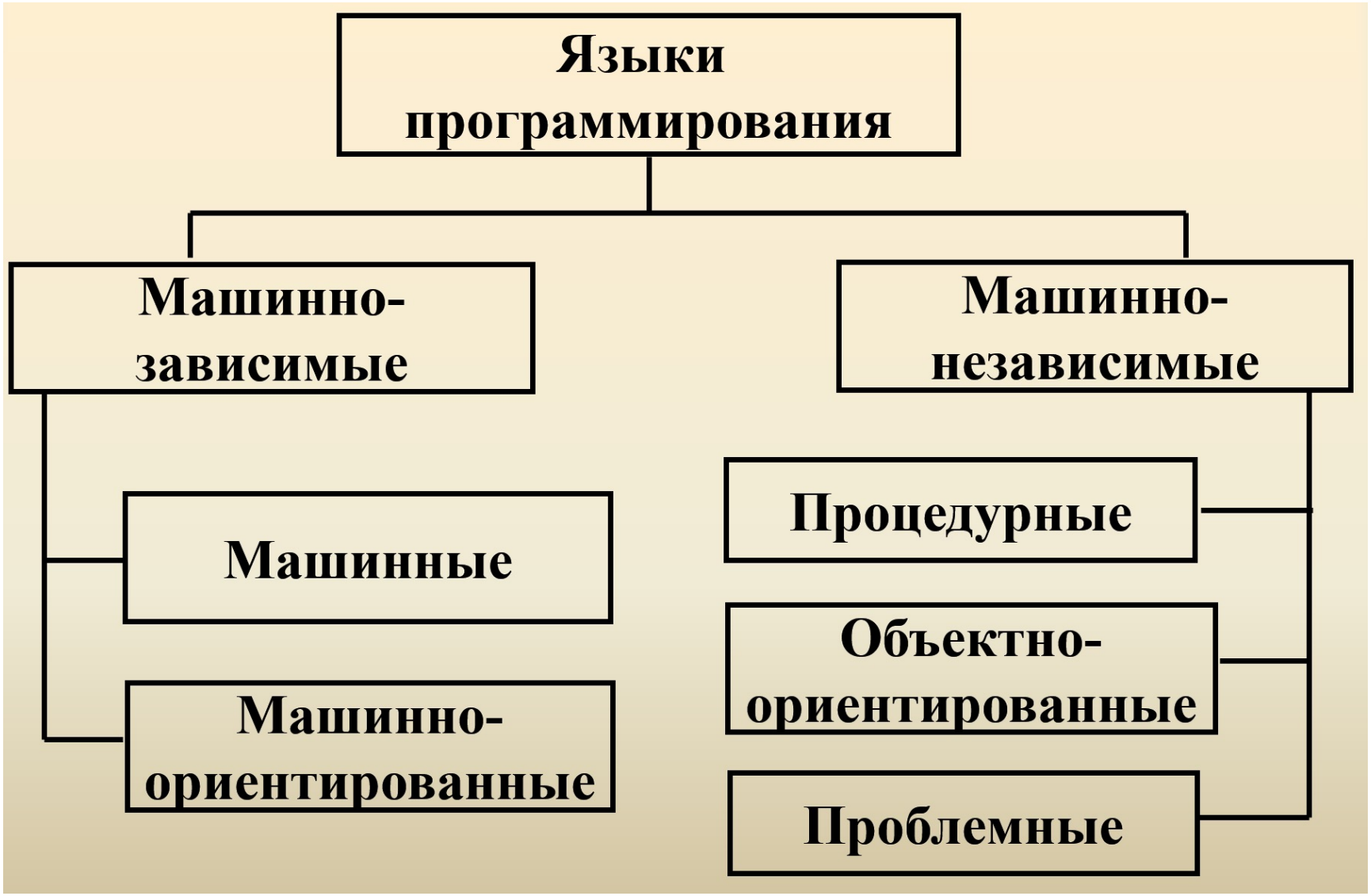
Высокоуровневый язык программирования — разработанный для быстроты и удобства использования программистом.

Слово «высокоуровневый» здесь означает, что язык предназначен для решения абстрактных задач и оперирует не инструкциями к оборудованию, а логическими понятиями и абстракцией данных (C++, Visual Basic, Java, Python, Ruby, Perl, Delphi, PHP).

Такие программы проще для понимания программистом, но гораздо менее эффективны, чем создаваемые при помощи низкоуровневых языков.

Низкоуровневый язык программирования — язык программирования, близкий к программированию непосредственно в машинных кодах.

Как правило, использует особенности конкретного семейства процессоров. Общеизвестный пример низкоуровнего языка — язык ассемблера.


Языки программирования
Машинно-зависимые
Машинно-независимые
Машинные
Машинно-ориентированные
Процедурные
Объектно-ориентированные
Проблемные

Языки программирования

- Машинный язык – это система команд компьютера.
- Машинно-ориентированные языки – это мнемокоды, автокоды, языки ассемблера.
- Проблемные языки направлены на решение узкого круга задач.
- Процедурные языки – это машинно-независимые языки для описания алгоритмов решения задачи.
- Объектно-ориентированные языки основаны на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного типа (класса).

Структурное программирование

Это программирование с разбиением на части сложных систем для последующей реализации в виде отдельных подпрограмм.

Примеры: PL/1, Pascal, Си

Модульное программирование

Оно предполагает выделение групп подпрограмм, использующих одни и те же глобальные переменные, в отдельные модули (библиотеки подпрограмм).

Примеры: Pascal, Си (C++), Ada, Modula

Процедурное программирование — основанное на концепции вызова процедуры (подпрограммы, метода или функции).

Процедуры просто содержат последовательность шагов для выполнения. В ходе выполнения программы любая процедура может быть вызвана из любой точки, включая саму данную процедуру.

Отличительные особенности:
возможность повторного использования одного и того же кода из нескольких мест программы без его копирования;
возможность написания программы без инструкций GOTO или JUMP (спагетти-код);
возможность поддержки модульности и структурности.

(Ада, Бейсик, Си, C++, C# (из Microsoft), ColdFusion, КОБОЛ, Delphi, JavaScript, JScript, Forth, Фортран, Java, Модула-2, Оберон, Глагол, Паскаль, Перл, ПЛ/1, Рапира, Visual Basic, REXX, PHP)

Любой алгоритм может быть реализован с помощью блок-схемы. Для каждого вида алгоритма предусмотрена своя конструкция из определенных блоков. Проверка блок-схемы и получение результата достигается при выполнении блок-схемы.

Допросы для самоконтроля

1. Что такое блок-схема?
2. Какие типы блоков бывают?
3. Какие блоки используются при реализации линейного, разветвляющегося, циклического алгоритмов?
4. Можно ли составить разные варианты блок-схем для одной и той же задачи?
5. Какие виды циклического алгоритма бывают?
6. Какие пункты должны присутствовать в любом цикле?
7. Что такое выполнение блок-схемы?
8. Для чего следует выполнять блок-схему?

1. Составьте блок-схемы для задачи по походу в магазин за яблоками. Используйте линейный, разветвляющийся и циклический алгоритмы.
2. Составьте блок-схему для нахождения корней квадратного уравнения через дискриминант. Используйте разветвляющийся алгоритм. Получите ответ, выполнив блок-схему.
3. Составьте блок-схемы для вывода на экран целых чисел от 1 до 10. Используйте цикл с предусловием, с постусловием. Выполните блок-схемы.

Литература

1. Материалы из открытого университета INTUIT.RU: Алгоритмизация. Введение в язык программирования C++ [Электронный ресурс]. Режим доступа: https://www.intuit.ru/studies/courses/16740/1301/info
2. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения