Тамбовское областное государственное
бюджетное профессиональное образовательное учреждение
«Котовский индустриальный техникум»

Основы алгоритмизация и программирование на языке Basic

Учебно-методическое пособие

по дисциплине «Информатика и ИКТ»

Котовск 2016 г.

Рассмотрено и одобрено
на заседании ПЦК
протокол № _____
от «___»_____20__ г.

УТВЕРЖДАЮ
зам. директора по УР
_____Улуханова И.В.
«___»_____20__ г

Председатель ПЦК: _____ (Н.В. Мартынова) преподаватель спец.дисциплин

Разработал: _____ (И.В. Пятибратова) преподаватель Информатики

Аннотация

Учебно-методическое пособие по дисциплине «Информатика и ИКТ» Основы алгоритмизация и программирование на языке Basic разработано в соответствии с действующей программой по курсу дисциплины «Информатика и ИКТ».

Содержит изложение основ алгоритмизации задач, а также их компьютерную реализацию на языке программирования высокого уровня Basic. Изложены основы технологии работы в среде программирования QBasic. Приведены методические материалы по освоению технологии написания, отладки и тестирования программ на языке программирования Basic для задач с различными видами алгоритмических структур.

Учебное пособие предназначено для студентов «Котовского индустриального техникума», изучающих дисциплину «Информатика и ИКТ».

СОДЕРЖАНИЕ

Введение

Образованным культурным человеком, грамотным специалистом в XXI веке может быть тот, кто хорошо владеет информационными технологиями. Ведь деятельность людей все в большей степени зависит от информативности, способности эффективно использовать информацию. Для свободной ориентации в информационных потоках современный специалист любого профиля должен уметь получать, обрабатывать и использовать информацию с помощью компьютеров, телекоммуникаций и других средств связи. Заложить фундамент информационной культуры призвана дисциплина «Информатика и ИКТ», изучение которой начинается со школьной скамьи, а затем, приобретая более целенаправленный характер, продолжается в среднем профессиональном учебном заведении. Эта дисциплина достаточно новая и своим появлением обязана развитию индустрии информатики, бурному процессу информатизации, начавшемуся в нашей стране. Об информации начинают говорить как о стратегическом ресурсе общества, как о ресурсе, определяющем уровень развития государства.

Информатика, как никакая другая область знаний, характеризуется чрезвычайно высокой степенью динамики изменений предмета ее изучения. Сегодня количество компьютеров в мире превышает 500 миллионов единиц и продолжает удваиваться в среднем каждые три года. При этом в среднем один раз в полтора года удваиваются основные технические параметры аппаратных средств, один раз в два-три года меняются поколения программного обеспечения, и один раз в пять-семь лет меняется база стандартов, интерфейсов и протоколов. Следует также отметить, что каждая вычислительная система по-своему уникальна. Найти две системы с одинаковыми аппаратными и программными конфигурациями весьма сложно, поэтому для эффективной эксплуатации вычислительной техники от специалиста требуется достаточно широкий уровень знаний и практических навыков.

К сожалению, до сих пор нет единой точки зрения на область знаний под названием «Информатика» и соответствующую ей дисциплину. Одни говорят – это веление времени, дань моде, явление преходящее или считают информатику сложной премудростью, где к тому же требуется познать характер некоего современного монстра – персонального компьютера и научиться управлять им. Другие

воспринимают информатику лишь как практическую дисциплину, где учат «правильно нажимать кнопки» персонального компьютера. Однако не вызывает сомнения, что современному специалисту, чтобы на должном уровне выполнять свои обязанности, необходимы инструментарий и методология его применения для обработки информации. Это сравнимо с использованием средств передвижения: теоретически человек может пешком преодолеть любое расстояние, но современный темп жизни просто немыслим без применения автомобиля, поезда, самолета и т. д. То же самое происходит и в области, связанной с обработкой информации: теоретически человек сам может переработать без компьютера любую информацию, но сделает это эффективнее, если он овладеет знаниями и умениями, которыми располагает информатика.

Информатика в широком смысле представляет собой единство разнообразных отраслей науки, техники и производства, связанных с переработкой информации главным образом с помощью компьютеров и телекоммуникационных средств связи во всех сферах человеческой деятельности.

Информатику в узком смысле можно представить как состоящую из трех взаимосвязанных частей — технических средств (hardware), программных средств (software), алгоритмических средств (brainware).

Возможности компьютера как технической основы обработки данных связаны с используемым программным обеспечением (программами).

Программа (*program*, *routine*)– упорядоченная последовательность команд (инструкций) компьютеру для решения задачи.

Программы предназначены для машинной реализации задач. Термин *задача и приложение* имеют очень широкое употребление в контексте информатики и программного обеспечения.

Задача (*problem*, *task*)– проблема, подлежащая решению.

Приложение (*application*)– программная реализация на компьютере решения задачи.

Таким образом, задача означает проблему, подлежащую реализации с использованием средств информационных технологий, а приложение – реализованное на компьютере решение по задаче. Приложение, являясь синонимом слова «программа», считается более удачным определением.

Процесс создания программ можно представить в укрупненном виде как последовательность, представленную на рис. 1.

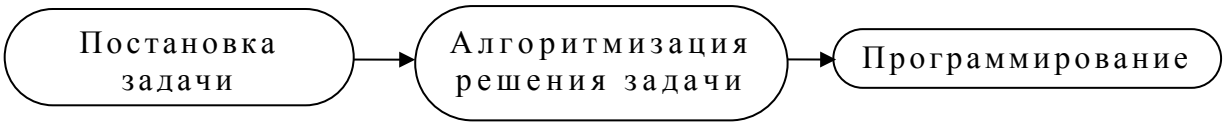


Рис. 1. Схема процесса создания программ

Традиционно наибольшее количество задач решаемых с помощью компьютера составляют инженерно-технические задачи. В процессе подготовки и решения их на компьютере можно выделить следующие этапы:

1. постановка задачи;
2. моделирование задачи;
3. алгоритмизация задачи;
4. разработка, подготовка и отладка программы;
5. эксплуатация программы.

Следует также отметить, что вышеперечисленные этапы тесно связаны друг с другом. Например, отладка программы может показать необходимость внесения изменений в программу, алгоритм или даже в постановку задачи.

Данное учебно-методическое пособие посвящено рассмотрению вопросов алгоритмизации и программированию на языке Бейсик, что позволяет освоить как логику решения различных задач, так и использование возможностей компьютера для реализации полученной логической схемы решения конкретной задачи.

1. Алгоритмизация задач

1.1. Понятие алгоритма

Для составления программы, предназначенной для решения на компьютере какой-либо задачи, требуется составление алгоритма ее решения.

Понятие алгоритма – одно из фундаментальных понятий информатики. Алгоритмизация наряду с моделированием выступает в качестве общего метода информатики. К реализации определенных алгоритмов сводятся процессы управления в различных системах, что делает понятие алгоритма близким к кибернетике.

Алгоритмы являются объектом систематического исследования,

пограничной между математикой и информатикой научной дисциплины, примыкающей к математической логике – теории алгоритмов.

Особенность положения состоит в том, что при решении практических задач, предполагающих разработку алгоритмов для реализации на компьютере, и тем более при использовании на практике информационных технологий, можно, как привило, не опираться на высокую формализацию данного понятия. Поэтому представляется целесообразным нижеприведенное определение алгоритма.

Алгоритм — это система точно сформулированных правил, определяющая процесс преобразования исходных данных (входной информации) в желаемый результат (выходную информацию).

Алгоритмами, например, являются правила сложения, умножения, решения алгебраических уравнений, умножения матриц и т.п. Слово алгоритм происходит от *algorithmi*, являющегося латинской транслитерацией арабского имени великого математика IX века Аль Хорезми. Благодаря латинскому переводу трактата Аль Хорезми европейцы в XII веке познакомились с позиционной системой счисления, и в средневековой Европе алгоритмом называлась десятичная позиционная система счисления и правила счета в ней.

Применительно к компьютерам алгоритм определяет вычислительный процесс, начинающийся с обработки некоторой совокупности возможных исходных данных и направленный на получение определенных этими исходными данными результатов. Термин *вычислительный процесс* распространяется и на обработку других видов информации, например, символьной, графической или звуковой.

Если вычислительный процесс заканчивается получением результатов, то говорят, что соответствующий алгоритм применим к рассматриваемой совокупности исходных данных. В противном случае говорят, что алгоритм неприменим к совокупности исходных данных. Любой применимый алгоритм имеет ряд обязательных свойств**:**

- *дискретность* – разбиение процесса обработки информации на более простые этапы (шаги выполнения), выполнение которых компьютером или человеком не вызывает затруднений;

- *результативность (выполнимость)* – конечность действий алгоритма решения задачи, позволяющая получить желаемый результат при допустимых исходных данных за конечное число шагов;
- *определенность (детерминированность)* – однозначность выполнения каждого отдельного шага преобразования информации, т.е. совпадении получаемых результатов независимо от пользователя и применяемых технических средств;
- *формальность* – применение правил выполнения алгоритма механически без выяснения вопроса: почему надо делать так, а не иначе;
- *массовость* – пригодность алгоритма для решения целого класса однотипных задач, различающихся конкретными значениями исходных данных.

Для задания алгоритма необходимо описать следующие его элементы:

- набор объектов, составляющих совокупность возможных исходных данных, промежуточных и конечных результатов;
- правило начала;
- правило непосредственной переработки информации (описание последовательности действий);
- правило окончания;
- правило извлечения результатов.

Алгоритм всегда рассчитан на конкретного исполнителя. В нашем случае таким исполнителем является компьютер. Для обеспечения возможности реализации на компьютере алгоритм должен быть описан на языке, понятном компьютеру, то есть на языке программирования.

Таким образом, можно дать следующее определение программы. *Программа для компьютера* представляет собой описание алгоритма и данных на некотором языке программирования, предназначенное для последующего автоматического выполнения.

1.2. Способы описания алгоритмов

К основным способам описания алгоритмов можно отнести следующие:

- словесно-формульный;
- структурный или в виде блок–схем;
- с помощью граф–схем;
- с помощью сетей Петри.

Перед составлением программ чаще всего используются словесно-формульный и блок–схемный способы. Иногда перед составлением программ на низкоуровневых языках программирования типа языка Ассемблера алгоритм программы записывают *на алгоритмическом языке*, представляющем в общем случае систему обозначений и правил для единообразной и точной записи алгоритмов и исполнения их. Алгоритмический язык удобно использовать для описания алгоритмов функционирования сложных программных систем, например, для описания принципов функционирования операционной системы. Отметим, что между понятием «алгоритмический язык» и «язык программирования» есть различие. Прежде всего, под исполнителем в алгоритмическом языке может подразумеваться не только компьютер, но и устройство для работы «в обстановке». Программа, записанная на алгоритмическом языке, не обязательно предназначена компьютеру. Практическая реализация алгоритмического языка – отдельный вопрос в каждом конкретном случае.

Как и каждый язык, алгоритмический язык имеет свой словарь. Основу этого словаря составляют слова, употребляемые для записи команд, входящих в систему команд исполнителя того или иного алгоритма. Такие команды называются простыми командами. В алгоритмическом языке используются слова, смысл и способ употребления которых задан раз и навсегда. Эти слова называются служебными. Использование служебных слов делает запись алгоритма более наглядной, а форму представления различных алгоритмов – единообразной.

Например, алгоритм, определяющий движение робота–исполнителя по некоторому маршруту, может иметь вид:

АЛГ в_склад
НАЧ
вперед
поворот на 90□ направо
вперед
КОН

При словесно-формульном способе алгоритм записывается в виде текста с формулами по пунктам, определяющим последовательность действий.

Пусть, например, необходимо найти значение следующего выражения: $y = 2\tilde{a}\ (x\ 6)$. Словесно-формульным способом алгоритм решения этой задачи может быть записан в следующем виде:

1. Ввести значения *a* и *x*.

2. Сложить *x* и 6.

3. Умножить *a* на 2.

4. Вычесть из 2*a* сумму (*x*+6).

5. Вывести *y* как результат вычисления выражения.

При блок–схемном описании алгоритм изображается геометрическими фигурами (блоками), связанными по управлению линиями со стрелками (направлениями потока данных). В блоках записывается последовательность действии.

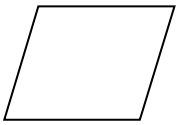
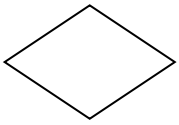
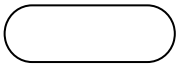
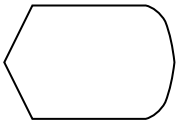
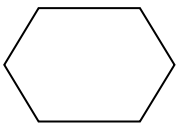
Данный способ по сравнению с другими способами записи алгоритма имеет ряд преимуществ. Он наиболее нагляден: каждая операция вычислительного процесса изображается отдельной геометрической фигурой. Кроме того, графическое изображение алгоритма наглядно показывает разветвления путей решения задачи в зависимости от различных условии, повторение отдельных этапов вычислительного процесса и другие детали.

Оформление программ должно соответствовать определенным требованиям. В настоящее время действует единая система программной документации (ЕСПД), которая устанавливает правила разработки, оформления программ и программной документации. В ЕСПД определены и правила оформления блок-схем алгоритмов (ГОСТ 19.701-90 ЕСПД, который действует взамен ГОСТа 19.002-80 ЕСПД, на который до сих пор ссылаются некоторые авторы учебных и справочных пособий по информатике).

Операции обработки данных и носители информации изображаются на схеме соответствующими *блоками*. В пределах одной схемы рекомендуется изображать блоки одинаковых размеров. Большая часть блоков по построению условно вписана в прямоугольник со сторонами *a* (высота) и *b* (длина). ГОСТ 19.701-90 ЕСПД жестких требований на размеры *a* и *b* не устанавливает. Но для качественного выполнения блок-схемы, целесообразно придерживаться рекомендации ГОСТа 19.002-80 ЕСПД: минимальное значение *a* равно 10 мм, увеличение *a* производится на число, кратное 5 мм; размер *b*=1,5*a*; для отдельных блоков допускается соотношение между *a* и *b*, равное 1:2. Нумерация блоков сейчас необязательна (по ГОСТу 19.002-80 ЕСПД порядковый номер блока записывался в разрыве верхней линии в ее левой части). Виды и назначение основных блоков в соответствии с действующим ГОСТом

19.701-90 ЕСПД приведены в таблице 1.

Таблица 1. Условные обозначения блоков схемы алгоритмов

Наименование	Обозначение	Функция
Процесс		Выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположение данных.
Ввод–вывод		Преобразование данных в форму, пригодную для обработки (ввод), или отображения результатов обработки (вывод), а также описания данных, участвующих в обработке.
Решение		Выбор направления выполнения алгоритма в зависимости от некоторых переменных условий
Предопределенный процесс		Использование ранее созданных и отдельно написанных программ (подпрограмм)
Терминатор (*Пуск–останов*)		Начало, конец, прерывание процесса обработки данных или выполнения программы
Соединитель		Указание связи между прерванными линиями, соединяющими блоки.
Дисплей		Отображение данных, представленных в человекочитаемой форме на носителе в виде отображающего устройства (экран для визуального наблюдения, индикаторы ввода информации).
Ручной ввод		Отображение данных, вводимых вручную во время обработки с устройства любого типа (клавиатура, переключатели, кнопки, световое перо, полоски со штриховым кодом).
Подготовка (*Модификация*)		Выполнение операций, меняющих команды или группу команд, изменяющих соответственно программу.
Комментарий		Связь между элементом схемы и пояснением.

Линии, соединяющие блоки и указывающие последовательность связей между ними, должны проводиться параллельно линиям рамки. Стрелка в конце линии может не ставиться, если линия направлена слева направо или сверху вниз. В блок может

входить несколько линий, то есть блок может являться преемником любого числа блоков. Из блока (кроме блоков «Решение» и «Подготовка») может выходить только одна линия. Блок «Решение» может иметь в качестве продолжения один из двух блоков, и из него выходят две линии. Если на схеме имеет место слияние линий, то место пересечения выделяется точкой. В случае, когда одна линия подходит к другой и слияние их явно выражено, точку можно не ставить.

Схему алгоритма следует выполнять как единое целое, однако в случае необходимости допускается обрывать линии, соединяющие блоки.

Если при обрыве линии продолжение схемы находится на этом же листе, то на одном и другом конце линии изображается специальный символ *соединитель* — окружность диаметром 0,5*a*. Внутри парных окружностей указывается один и тот же идентификатор. В качестве идентификатора, как правило, используется порядковый номер блока, к которому направлена соединительная линия.

В ГОСТ 19.701-90 ЕСПД понятие *межстраничного соединителя отсутствует, поэтому е*сли схема занимает более одного листа, то используется описанный выше *соединитель* в виде окружности.

1.3. Структурные схемы алгоритмов

Как уже было отмечено ранее, одним из свойств алгоритма является дискретность, т. е. возможность расчленения процесса вычислений, предписанных алгоритмом, на отдельные этапы, возможность выделения участков программы с определенной структурой. Можно выделить и наглядно представить графически три простейшие структуры:

- последовательность двух или более операций;
- выбор направления;
- повторение.

Любой вычислительный процесс может быть представлен как комбинация этих элементарных алгоритмических структур (это содержание теоремы Бема–Якопини). Соответственно, вычислительные процессы, выполняемые на компьютере по заданной программе, можно разделить на три основных вида:

- линейные;
- разветвляющиеся;

- циклические.

Линейным принято называть вычислительный процесс, в котором операции выполняются последовательно, в порядке их записи. Соответственно, линейный алгоритм – это алгоритм, в котором действия выполняются одно за другим.

Каждая операция является самостоятельной, независимой от каких-либо условий. На схеме блоки, отображающие эти операции, располагаются в линейной последовательности.

Линейные вычислительные процессы имеют место, например, при вычислении арифметических выражений, когда имеются конкретные числовые данные и над ними выполняются соответствующие условию задачи действия.

Рассмотрим *пример линейного вычислительного процесса с соответствующим ему линейным алгоритмом*.

Задача: Вычислить периметр произвольного треугольника по его трем сторонам (вопрос существования треугольника не рассматривается).

Решение:

1 этап: Постановка задачи.

Исходные данные: A, B, C – стороны произвольного треугольника

Выходные данные: P – периметр треугольника.

2 этап: Математическая модель.

P=A+B+C

3 этап: Составление алгоритма

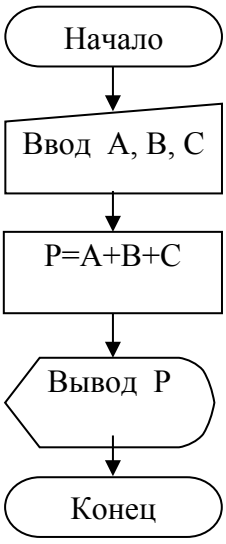


Рис. 2. Пример линейного алгоритма

Вычислительный процесс называется разветвляющимся, если для его реализации предусмотрено несколько направлений (ветвей). Соответственно, **разветвляющийся алгоритм –** это алгоритм, в котором в зависимости от условия выполняется либо одна, либо другая последовательность действий.

Каждое отдельное направление процесса обработки данных является отдельной ветвью вычислений. Разветвление в программе — это выбор одной из нескольких последовательностей команд при выполнении программы. Выбор направления зависит от заранее определенного признака, который может относиться к исходным данным, к промежуточным или конечным результатам. Признак характеризует свойство данных и имеет два или более значений.

Разветвляющийся процесс, включающий в себя две ветви, называется простым, более двух ветвей — сложным. Сложный разветвляющийся процесс можно представить с помощью простых разветвляющихся процессов.

Направление ветвления выбирается логической проверкой, в результате которой возможны два ответа: «да» — условие выполнено и «нет» — условие не выполнено.

Следует иметь в виду, что, хотя на схеме алгоритма должны быть показаны все возможные направления вычислений в зависимости от выполнения определенного условия (пли условий), при однократном прохождении программы процесс реализуется только по одной ветви, а остальные исключаются. Любая ветвь, по которой осуществляются вычисления, должна приводить к завершению вычислительного процесса.

Разветвляющиеся алгоритмы могут иметь различную структуру (Рис.3).

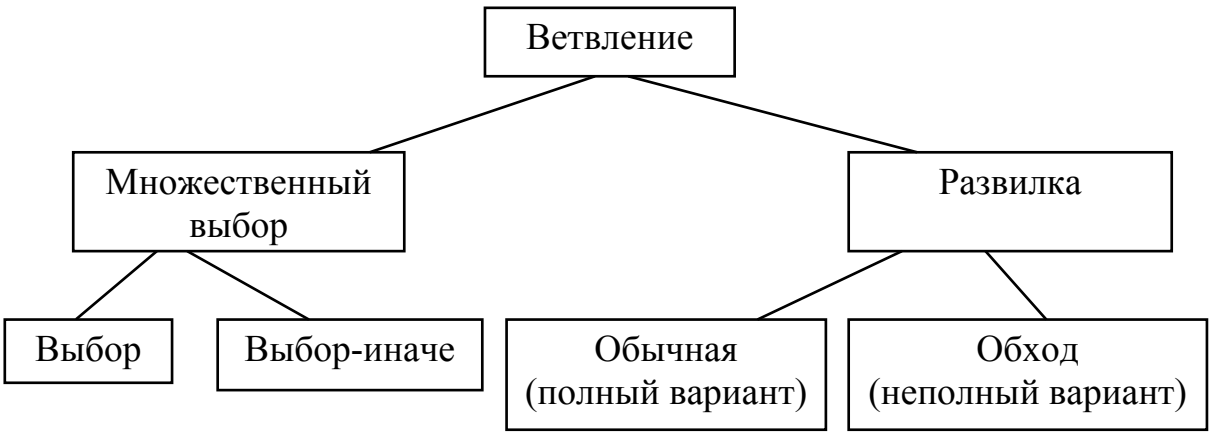


Рис. 3. Структуры разветвляющегося алгоритма

Запись разветвляющегося алгоритма типа «Развилка» в виде блок-схемы

представлена на рисунке 4.

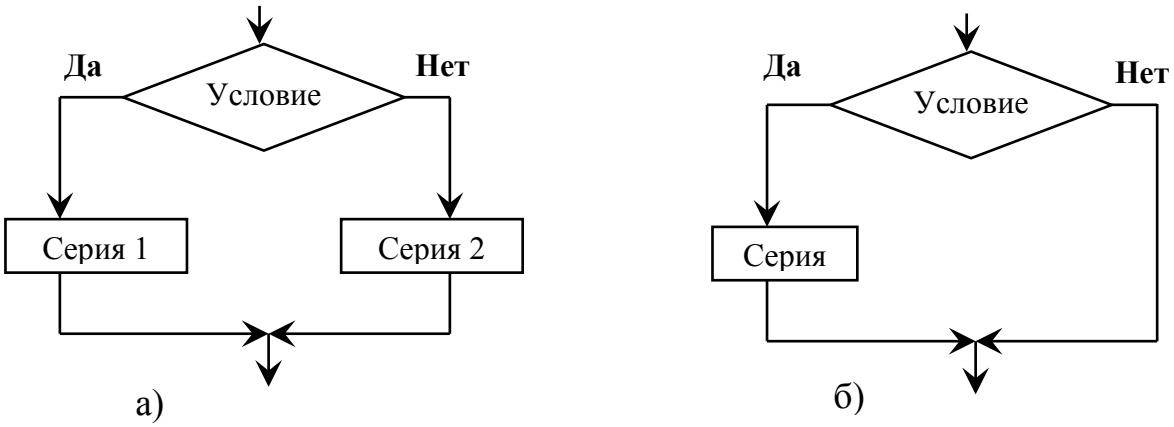


Рис. 4. Блок-схемы разветвляющихся алгоритмов типа «Развилка»: а) обычная, б) обход

Рассмотрим *пример разветвляющегося вычислительного процесса с соответствующим ему разветвляющимся алгоритмом*.

Задача: Вычислить значение функции $y(x) = \begin{cases} -5, & при\ x \le 10 \\ x^3, & при\ x > 10 \end{cases}$ для заданного x.

Решение:

1 этап: Постановка задачи.

Исходные данные: x – значение аргумента функции.

Выходные данные: y – искомое значение функции при заданном x.

2 этап: Математическая модель.

$$y(x) = \begin{cases} -5, & при\ x \le 10 \\ x^3, & при\ x > 10 \end{cases}$$

3 этап: Составление алгоритма

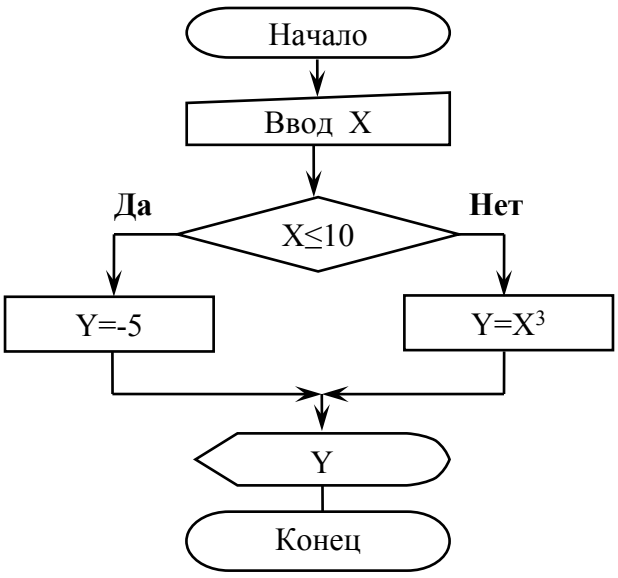


Рис. 5. Пример разветвляющегося алгоритма

Циклическими называются программы, содержащие циклы. Цикл — это многократно повторяемый участок программы. **Циклический алгоритм –** это алгоритм, в котором имеется последовательность действий, выполняющихся многократно. Последовательность действий, выполняющихся многократно, называется **телом цикла**.

В организации цикла можно выделить следующие этапы:

- подготовка (инициализация) цикла (И);
- выполнение вычислений цикла (тело цикла) (Т);
- модификация параметров (М);
- проверка условия окончания цикла (У).

Порядок выполнения этих этапов, например, Т и М, может изменяться. В зависимости от расположения проверки условия окончания цикла различают циклы с нижним и верхним окончаниями (соответственно: цикл «ДО» или цикл с постусловием и цикл «ПОКА» или цикл с предусловием) (рис. 6). Для цикла с нижним окончанием (рис. 6–а) тело цикла выполняется как минимум один раз, так как сначала производятся вычисления, а затем проверяется условие выхода из цикла. В случае цикла с верхним окончанием (рис. 6–б) тело цикла может не выполниться ни разу в случае, если сразу соблюдается условие выхода.

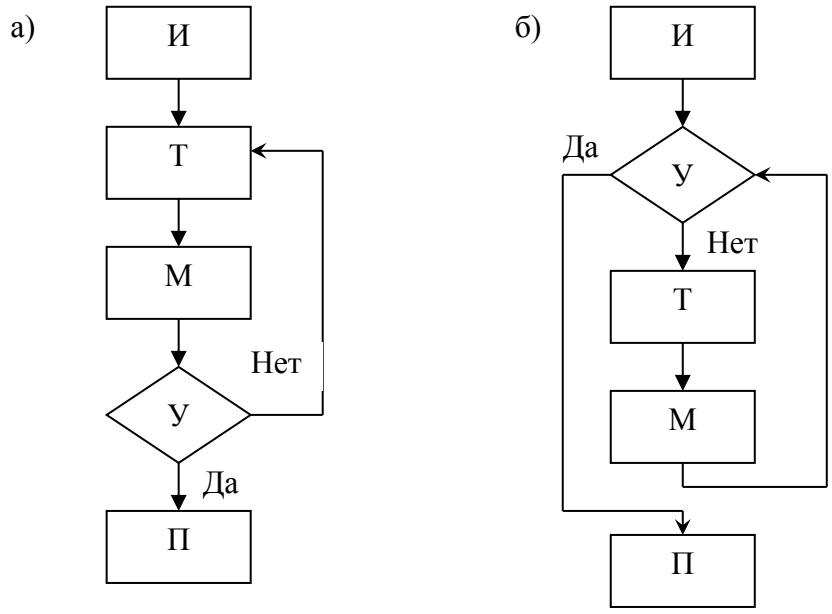
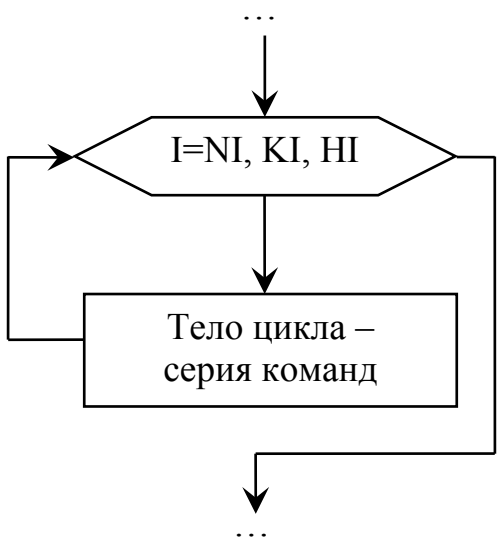


Рис. 6. Блок-схемы циклических алгоритмов: а) с постусловием, б) с предусловием

Цикл называется *детерминированным*, если число повторений тела цикла заранее известно или определено. Цикл называется итерационным, если число повторений тела цикла заранее неизвестно, а зависит от значений параметров

(некоторых переменных), участвующих в вычислениях. Для детерминированных циклов, которые называют также цикл с параметром или цикл со счетчиком, запись в виде блок-схемы представлена на рисунке 7.



I – имя параметра цикла

NI – начальное значение параметра цикла

KI – конечное значение параметра цикла

Рис. 7. Блок-схема циклического алгоритма с параметром

Работу цикла с параметром можно описать следующим образом (словесный алгоритм):

1. Вычисление значений выражений, определяющих NI, KI и HI;
2. I присваивается значение NI;
3. I сравнивается с KI: если I больше KI при положительном HI (или, наоборот, I меньше KI при отрицательном HI), то переход к пункту 7, иначе к следующему пункту;
4. Выполняется тело цикла;
5. I автоматически изменяется на HI;
6. Переход к пункту 3;
7. Конец цикла.

Рассмотрим *пример циклического вычислительного процесса с соответствующим ему циклическим алгоритмом*.

Задача: Требуется вычислить значение функции y=sin φ в диапазоне изменения аргумента от 10° до 90° с шагом 10°.

Решение:

1 этап: Постановка задачи.

Исходные данные: X – угол φ (аргумент функции)

NX – начальное значение угла φ (10°)

KX – конечное значение угла φ (90°)

HX – шаг изменения угла φ (10°)

Выходные данные: Y – значения функции при заданных φ.

2 этап: Математическая модель.

$$X = \frac{\pi}{180} \quad Y = \sin X$$

3 этап: Составление алгоритма

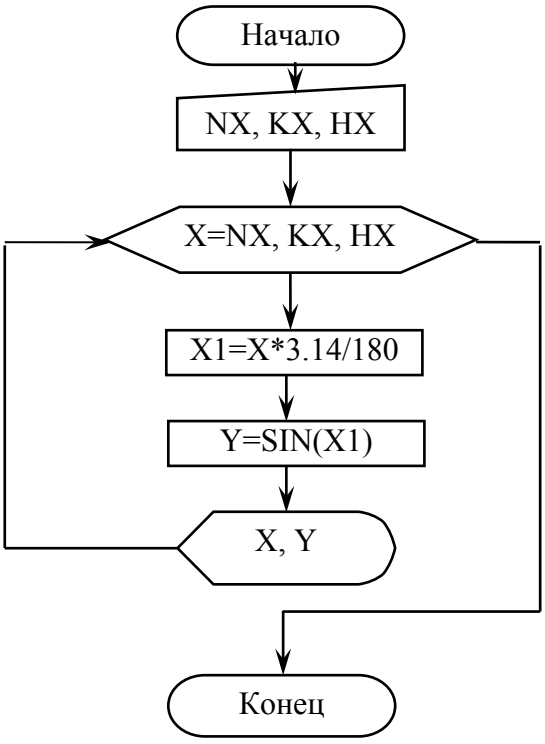


Рис. 8. Пример циклического алгоритма

Алгоритмы разветвляющихся и циклических структур можно комбинировать одну с другой – как путем организации их следования, так и путем создания суперпозиций: вложений одной структуры в другую, – сколь угодно разнообразно для выражения логики алгоритма решения любой задачи. Умение образовывать из базовых структур их суперпозиции в соответствии с условиями конкретной задачи – одно из важнейших в программировании.

2. Основы программирования на языке Basic

Язык программирования Basic (Beginner's All-purpose Instruction Code – Всецелевой язык программирования для начинающих) разработан Джоном

Кимини и Томасом Куртцем в Дартмутском университете, США, в 1964 г.

Basic занимает особое место среди всех языков высокого уровня. С самого начала он задумывался как универсальный язык для начинающих. Средства программирования на Basic долгое время включались в комплект поставки ПК как обязательный элемент программного обеспечения. В настоящее время существует множество версий этого языка, которые иногда очень сильно отличаются друг от друга. Для учебных заведений самыми распространенными являются версии MSX, впервые реализованная на японском компьютере «Ямаха», и версии фирмы Microsoft для компьютеров IBM. Кроме того, в нашей стране популярностью пользуется версия Turbo-Basic фирмы Borland. Без преувеличения можно сказать, что сегодня Бейсик до сих пор является самым распространенным языком программирования. После появления мощных компиляторов Visual Basic этот язык стал популярен и у профессиональных программистов. Бейсик относится к языкам программирования высокого уровня. Как и другие языки, этот язык имеет алфавит, синтаксис и семантику.

К достоинствам языка программирования QBASIC с точки зрения массового пользователя следует отнести:

- простота синтаксиса;
- простота организации данных и управляющих структур;
- большое число *встроенных* команд и функций, позволяющих без труда выполнять такие операции, как управление текстовым и графическим экраном, обработка символьных строк и т.п.

Особым достоинством QBASIC следует считать возможность работы в режиме *интерпретации*, который резко упрощает процесс отладки программ: исполнение почти каждой команды можно проверить сразу после написания.

2.1. Алфавит языка программирования Бейсик

Алфавит языка Basic базируется на кодировочной таблице символов ASCII, первая половина которой (символы с кодом 0 – 127) – стандартная, а вторая половина (символы с кодом 128 – 255) – специфична для каждой страны. В этой таблице каждый символ имеет 8-битовое обозначение. Таким образом, в алфавит языка Basic входят: 1) прописные и заглавные буквы английского алфавита, 2) прописные и заглавные буквы русского алфавита, 3) десятичные цифры, 4) знаки препинания (точка, запятая,

точка с запятой, кавычки, апостроф и др.), 5) знаки операций (сложение, деление, больше, равно и др.), 6) набор специальных символов, который имеется на клавиатуре компьютера (% & ! # $ и др.).

2.2. Константы и переменные языка Бейсик

Из алфавита языка программирования могут быть сформированы функционально законченные элементарные конструкции, которые называются синтаксическими единицами. Если провести аналогию с разговорным языком, например русским, то это отдельные слова: существительные, глаголы, междометия и т.д.

Синтаксические единицы условно можно разбить на следующие виды:

– константы;

– переменные;

– ключевые слова;

– метки.

Константы – это фиксированные величины, которые в процессе вычислений не изменяют своих значений. В Бейсике используются константы двух типов – числовые и символьные.

Числовые константы – это десятичные числа, которые состоят из десятичных цифр или десятичных цифр и дополнительных символов. В свою очередь, числовые константы можно разделить на целые и вещественные.

Целые константы – это целые числа, которые записываются в виде последовательности десятичных цифр, перед которой может стоять знак + или –. Числа без знака воспринимаются положительными. Бейсик допускает использование как целых чисел в диапазоне от –32768 до 32768, так и длинных целых чисел в диапазоне от –2147483648 до 2147483648.

Примеры записи целых чисел:

Математическая запись	Запись на языке Бейсик
41	41 или +41
-39	-39
-2147483648	-2147483648

Вещественные числа при переводе в двоичную систему счисления представляют собой, как правило, бесконечные дроби и округляются в пределах формы записи. Поэтому эти числа представляются в памяти приближенно.

Допускается запись вещественных чисел:

- в основной форме – числа с фиксированной точкой;
- в форме с порядком – числа с плавающей точкой.

Числа с фиксированной точкой – это дробные десятичные числа, у которых дробная часть отделяется от целой части десятичной точкой. Они могут быть записаны следующим образом:

□N. □N.M □.M ,

где N и M – последовательности десятичных цифр.

Десятичная точка в записи этого числа обязательна. Знак + или - записываются в начале константы. Число без знака воспринимается положительным.

Примеры записи чисел с фиксированной точкой:

Математическая запись	Запись на языке Бейсик	
	Верно	Неверно
0	0. или 0.0 или .0	0
4	4. или 4.0	4 или +.4
-1,0	-1.0 или -1.	.1 или -1,0
-1,5	-1.5	1.5 или -1,5

Числа с плавающей точкой – запись чисел представленных в математике в показательной форме. На языке бейсик они записываются:

- одинарной точности: □N.E□S □N.ME□S □.ME□S
- удвоенной точности: □N.D□S □N.MD□S □.MD□S

и имеют в записи отличия по основанию степени и где N и M - последовательность десятичных цифр.

Для чисел одинарной точности N + M = 6.

Для чисел удвоенной точности N + M = 14.

E – основание степени для чисел одинарной точности, т.е. число 10.

D – основание степени для чисел удвоенной точности, т.е. число 10.

S – порядок числа, выражается целой константой, состоящей из одной или двух десятичных цифр.

Положительный знак числа и показателя степени можно не указывать.

Примеры записи чисел с плавающей точкой одинарной точности:

Математическая запись	Запись на языке Бейсик	
	Верно	Неверно
$123{,}4\cdot10^{-2}=$	123.4E-2	123.4*10**(-2)
$=1{,}234=$	1.234E00	1.234*10
$=0{,}1234\cdot10^{1}$	0.1234E1	0.1234*E1

Примеры записи чисел с плавающей точкой удвоенной точности:

Математическая запись	Запись на языке Бейсик	
	Верно	Неверно
$1234,5678 \cdot 10^{-2} =$ $= 12345678 \cdot 10^{-6} =$ $= 0,12345678$	1234.5678D-2	1234.5678*10**(-2)
	12345678.D-6	12345678D-6
	0.12345678D2	.12345678E2

Символьная константа представляет собой последовательность любых символов алфавита. Значения символьных констант заключается в кавычки. *Примеры* записи символьных констант: ”Кстовский нефтяной техникум”, ”Array”, ”sin(x)= ”.

Переменная – это величина, которая может меняться при выполнении программы. При изучении алгебры дается понятие переменной величины. Например, в простом алгебраическом равенстве ***c = f + 2b – 5*** значение переменной ***c*** зависит от значения переменных ***f*** и ***b***, указанных в правой части равенства. Если ***f*=2** и ***b*=6,** тогда ***c*=9**. Такое же равенство можно записать в программе на Бейсике: ***c = f + 2*b – 5*** . В терминах языка Бейсик ***c***, ***f*** и ***b*** – это имена переменных. Такие имена также называют идентификаторами.

В языке Бейсик идентификатор (имя переменной) – это произвольный набор символов для обозначения переменной, который может содержать от 1 до 40 символов, причём первый символ должен быть латинской буквой, а остальные – латинские буквы или цифры или символы типа @, #, % и.т.д. *Примеры* имен переменных: A, Ds, SodRan, k1, n123, dlina!

Кроме имени переменная характеризуется типом. Переменные – это величины, которые имеют определенное значение в каждый момент выполнения программы, а оно может быть как числовым, так и символьным. Для хранения значения переменной выделяется поле памяти компьютера.

Тип переменной, определяемый ее значением	Символ, определяющий тип переменной	Описание типа переменной	Объём поля памяти	Пример значения переменной
Целые числа	**%**	integer	2 байта	17; -123
Вещественные числа	**!**	single	4 байта	314.15; 3.1415E+2
Символьный	**$**	string	min 1 байт	язык
Длинные целые числа	**&**	long	4 байт	-12543786
Вещественные числа двойной точности	#	double	8 байт	1.2543786

Тип переменной в Бейсике опознаётся по последнему символу в её имени: % – целая переменная; & – длинная целая переменная; ! – вещественная переменная обычной точности; # - вещественная переменная двойной точности; $ - символьная переменная, а также может определяться специальным оператором задания типа.

2.3. Арифметические выражения

Выражением называется последовательность операций, которые необходимо произвести над данными, чтобы получить требуемое значение.

В Бейсике существует четыре вида выражений:

- *арифметические;*
- *отношений;*
- *логические;*
- *строковые.*

Арифметическое выражение – это аналог обычной арифметической (алгебраической, математической) формулы. Результатом его вычисления является целое или вещественное число. Любое математическое выражение на Бейсике записывается в виде строки и может содержать: числовые константы, числовые переменные, математические функции, знаки математических операций, систему скобок. Понятие числовых констант и числовых переменных было уже дано выше, поэтому рассмотрим последующие три составляющие арифметических выражений.

Функция – это заранее определённая операция над данными, математическая функция – это функция, значением которой является число. Вот некоторые примеры математических функций встроенных в транслятор Бейсика:

Название функции	Запись в математике	Запись в Бейсике	Пример в Бейсике
Абсолютная величина	$\|x\|$	ABS(X)	ABS(-5)
Экспонента	e^x	EXP(X)	EXP(5)
Логарифм натуральный	ln x	LOG(X)	LOG(3)
Квадратный корень	$\sqrt{x}$	SQR(X)	SQR(4)
Синус	sin x	SIN(X)	SIN(3.14)
Косинус	cos x	COS(X)	COS(1)
Тангенс	tg x	TAN(X)	TAN(2)
Целая часть числа	[x]	INT(X)	INT(13.5)
Арктангенс	arctg x	ATN(X)	ATN(3)

Знак числа	$\text{sign } x = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$	SGN(X)	SGN(-5)

Таким образом, в Бейсике обращение к функции осуществляется по ее имени, а аргумент обязательно записывается в скобках. При вычислении арифметического выражения значения функций определяются в первую очередь еще до выполнения математических операций.

В Бейсике допустимы следующие арифметические операции:

Название операции	Знак в математике	Знак в Бейсике	Пример в математике	Пример в Бейсике
Возведение в степень	нет	^	2^5	2^5
Умножение	·	*	2·5	2*5
Деление	:	/	10:2	10/2
Сложение	+	+	5+10	5+10
Вычитание	–	–	a–b	a–b
Остаток от деления	нет	mod	нет	17 mod 3 (результат 2)
Деление нацело	нет	\	нет	17\3 (результат 5)

Заметим, что операции указаны в вышеприведенной таблице в порядке уменьшения приоритета.

Действия в арифметических выражениях выполняют слева направо в зависимости от их приоритета. Для того чтобы изменить естественный порядок действий, используются круглые скобки. Выражения в круглых скобках выполняются в первую очередь. Заметим, что понятие система скобок означает то, что в выражении всегда должно совпадать количество открывающихся скобок с количеством закрывающихся скобок.

Примеры записи чисел математических выражений на языке Бейсик:

Математическое выражение $\sqrt[5]{\frac{\sin x^3 \cdot \cos^2 x}{\sqrt{x - 8{,}5} \cdot 1{,}75 \cdot 10^{-6}}} \cdot \frac{e^{x+1}}{5|x|}$ запишется на языке Бейсик в виде: ((sin(x^3)–cos(x)^2)/(sqr(x–8.5)+1.75E-6))^(1/5)+exp(x+1)/5/abs(x)

2.4. Операторы языка Бейсик для записи линейного алгоритма

2.4.1. *Оператор очистки экрана*

Оператор очистки экрана состоит из одного ключевого слова: **CLS**.

Название этого оператора произошло от английских слов Clear Screen, что в переводе означает очистить экран.

Например, результатом выполнения операторов:

CLS

? ”Привет”

будет черный экран монитора, а в левом верхнем углу будет выведено слово *Привет*.

2.4.2. *Оператор ввода*

Оператор ввода начинается с ключевого слова INPUT. INPUT в переводе с английского языка означает *вставлять, вводить*. Поэтому действие этого оператора состоит в ведении значения переменной или значений переменных с клавиатуры в память компьютера.

Синтаксис оператора ввода:

INPUT ”подсказка”; имя переменной

INPUT [”подсказка”;] имя1, имя2, …

Отметим, что конструкция в синтаксисе типа [...], означает, что данный параметр является необязательным, т.е. его можно пропустить. При встрече с оператором ввода (ключевое слово INPUT) программа приостанавливает своё действие; на экране появляется знак вопроса «?» или подсказка со знаком «?», после которого необходимо набрать на клавиатуре значение переменной, входящий в состав оператора, нажать клавишу Enter. Если в операторе указано несколько имен переменных, то их значения набираются на клавиатуре через запятую и потом нажимается клавиша Enter.

Оператор INPUT можно использовать для присваивания значения как числовым, так и строковым переменным. Пример его использования будет приведен ниже.

2.4.3. *Оператор присваивания*

Оператор присваивания начинается с ключевого слова LET. LET в переводе с английского языка означает *позволять, допускать*. Отметим, что ключевое слово LET можно опустить, что почти всегда и делается при написании программы, а отличительным признаком оператора является присутствие знака «=» в синтаксисе оператора.

Синтаксис оператора присваивания:

Имя переменной = арифметическое выражение

В результате выполнения оператора присваивания переменной, стоящей слева от знака равенства присваивается значение арифметического выражения, которое перед присваиванием вычисляется. В частном случае арифметическое выражение может состоять как из какого-либо числа, так и из одной переменной. Для корректного действия оператора присваивания необходимо, чтобы все переменные в выражении имели некоторые значения еще до его выполнения.

Например, результаты выполнения трех операторов присваивания:

Оператор присваивания	Действие оператора присваивания
X=15	присваивание переменной X значения 15
Y=2	присваивание переменной Y значения 2
Z=(X-3*Y^2)+7	1. вычисление значения правой части третьего оператора присваивания (X-3*Y^2)+7. Получается 10 2. присваивание переменной Z значения 10

Отметим, что с помощью оператора можно задавать значения переменных в самой программе, но это ограничивает возможность использования программы с различными исходными данными.

2.4.4. *Оператор вывода*

Оператор ввода начинается с ключевого слова PRINT. PRINT в переводе с английского языка означает *печать, оттиск*. Поэтому действие этого оператора состоит в выведении значения переменной или значений переменных, текста или значений выражений после их вычисления из памяти компьютера на экран монитора. Вместо слова PRINT можно набирать знак «**?**», который после набора оператора транслятор Бейсик заменяет ключевым словом.

Синтаксис оператора вывода очень разнообразен. Это связано тем, что визуализация результатов работы программы должна быть понятна как самому автору программы, так и сторонним пользователям. Поэтому продемонстрируем работу оператора вывода на конкретных примерах.

Оператор вывода на экран позволяет:

1. Выводить текстовую информацию, заключенную в кавычки, на экран монитора.

Например:

PRINT ”Привет”

2. Вычислять выражения арифметических выражений.

Например:

PRINT 5.5*4-5^2

3. Выводить значения переменных на экран монитора.

Например:

SQ1=12*25

DAY$=«понедельник»

PRINT SQ1, DAY$

Разделителем между выводимыми данными может быть:

запятая (**,**) – данные отделяются друг от друга шагом табуляции, равным 8 пробелам;

точка с запятой (**;**) – данные печатаются вплотную друг к другу.

2.4.5. *Программа линейного алгоритма*

Рассмотрим реализацию линейного алгоритма на примере задачи, приведенной в разделе 1.3, а именно: вычислить периметр произвольного треугольника по его трем сторонам.

```
CLS
INPUT " Введите значения:  A, B, C "; A, B, C
P = A + B + C
? " P= "; P
END
```

Если записать эти пять строк на русском языке, то получим:

1. Начало – очистить экран монитора
2. Ввод трех чисел A, B, C с клавиатуры
3. Вычислить сумму P = A + B + C и присвоить ее значение P
4. Вывод на экран дисплея значения P
5. Конец

В программе использован оператор конца программы. Его ключевое слово END. Действие этого оператора заключается в том, что транслятор определяет: программа завершена, далее операторов нет.

Для самостоятельной работы на построение блок-схемы и записи программы линейного алгоритма можно предложить следующие задачи:

Задача 1. Найти площадь (S) и длину окружности (L). R – радиус вводится с клавиатуры.

Задача 2. Найти площадь (S) и периметр прямоугольного треугольника (P). Значения катетов A и B вводятся с клавиатуры.

Задача 3. Найти общее сопротивление параллельно соединенных проводников (RO). Значения R1 и R2 вводятся с клавиатуры.

2.5. Операторы языка Бейсик для записи разветвляющегося алгоритма

Для записи программы, реализующей разветвляющийся алгоритм, используются операторы перехода: оператор условного перехода и оператор безусловного перехода. Как уже выше отмечалось, что при разветвляющемся алгоритме порядок действий зависит от выполнения или не выполнения некоторого условия. Поэтому в языке Бейсик предусмотрены специальные выражения либо отношения, либо логические, которые позволяют записать условия для ветвления.

2.5.1. *Выражения отношения*

Выражения отношения образуются из сравнения двух выражений либо арифметических, либо строковых. Ограничимся рассмотрением выражений отношения, в которых присутствуют арифметические выражения. По сравнению с арифметическими выражениями в составе выражения отношения **всегда** присутствует лишь одна операция отношения из возможных шести:

Операции отношения	Знак операции	Выражение в Бейсике
Равенство	=	***x = y***
Неравенство	< >	***x < > y***
Меньше	<	***x < y***
Больше	>	***x > y***
Меньше или равно	< =	***x < = y***
Больше или равно	> =	***x > = y***

Значением выражения отношения могут быть лишь TRUE или FALSE. TRUE с английского языка переводится как *истина*, что обычно ассоциируется с ответом «ДА», FALSE с английского языка переводится как *ложь*, что обычно ассоциируется с ответом «НЕТ». Значение выражение отношения может быть получено только после того, как получены значения арифметических выражений, входящих в выражение отношения.

Примеры выражений отношений, для записи условий на языке Бейсик:

A > B+10 SIN(C) < 30 A$ = ”cat X < >135 Y/17 > = Z S+T < = F

2.5.2. *Логические выражения*

Сложное условие записывается с помощью логических выражений, содержащих операции: **NOT** – логическое отрицание, **AND** – логическое умножение (в

переводе с английского языка означает И) – конъюнкция (название в математике), **OR** – логическое сложение (в переводе с английского языка означает ИЛИ) – дизъюнкция (название в математике). В логическом отношении операции осуществляются над операциями отношений. Поэтому действие логических операций можно полностью описать таблицей истинности над двумя выражениями отношений **A** и **B**, которые принимают значения либо TRUE, либо FALSE:

A	**B**	NOT **A**	NOT **B**	**A** AND **B**	**A** OR **B**
TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
FALSE	TRUE	TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	TRUE	TRUE	FALSE	FALSE

Логическое выражение в общем виде может содержать несколько логических операций, поэтому определен приоритет их выполнения: высший приоритет – логическое отрицание (**NOT**), затем – логическое умножение (**AND**), низший приоритет – логическое сложение (**OR**). Для изменения порядка выполнения логических операций используются круглые скобки (как в арифметических выражениях).

Примеры логических выражений, для записи условий на языке Бейсик:

1. Записать на языке Бейсик условие: $1 < C < 7$, т.е. С принадлежит интервалу (1, 7) или графически

Ответ: **C>1 AND C<7**

2. Записать на языке Бейсик условие: $C \leq 1$ или $C \geq 7$, т.е. С не принадлежит интервалу (1, 7) или графически

Ответ: **NOT (C>1 AND C<7)** или **C<=1 OR C>=7**

Следует отметить следующий порядок вычисления логического выражения: 1. вычисляются значения арифметических выражений, 2. вычисляются значения выражений отношений, 3. вычисляется значение логического выражения.

2.5.3. *Операторы условного перехода*

Программирование, т.е. составление программы на алгоритмическом языке, состоит в написании последовательности операторов алгоритмического языка, реализующих алгоритм решения задачи.

Обычно операторы в программе выполняются последовательно: один за другим. В языке Бейсик, как и в любом другом языке программирования, имеются такие

операторы, которые могут изменять порядок выполнения операторов в программе. В частности, те операторы, которые сообщают компьютеру о том, что управление должно быть передано некоторым другим частям программы, называются операторами передачи управления. Передача управления может быть условной или безусловной, т. е. управление будет передаваться в том случае, если некоторое условие истинно, или независимо от условия.

Условные операторы служат для изменения последовательности выполнения операторов программы в зависимости от некоторого условия. Они позволяют описывать разветвляющийся вычислительный алгоритм типа развилки, который был описан выше. По своей структуре условные операторы подразделяются на строчные и на блочные. Однако любой условный оператор начинается с ключевого слова IF, что в переводе с английского языка означает *если*. Затем записывается условие в виде логического выражения или выражения отношения и потом следует ключевое слове THEN, что в переводе с английского языка означает *тогда*. Далее следуют отличия в использовании блочного или строчного условных операторов, а также учитывается тип разветвляющегося алгоритма. В условных операторах могут использоваться еще два ключевых слова: ELSE, что в переводе с английского языка означает *иначе*, и ENDIF, что в переводе с английского языка означает *конец если*.

Синтаксис условного строчного оператора для полного варианта разветвления, т.е. алгоритм предусматривает одно действие (команду 1) при значении условия TRUE (истина, «Да») и одно действие (команду 2) при значении условия FALSE (ложь, «Нет»):

IF условие THEN команда 1 ELSE команда 2

Действие условного оператора состоит в том, что вычисляется значение условия. Если оно равно TRUE, то выполняется команда 1, следующая за ключевым словом THEN, и осуществляется переход к следующему оператору программы. Если оно равно FALSE, то выполняется команда 2, следующая за ключевым словом ELSE, и осуществляется переход к следующему оператору программы.

Синтаксис условного строчного оператора для неполного варианта разветвления («обход»), т.е. алгоритм предусматривает лишь одно действие (команду) при значении условия TRUE (истина, «Да»), а при значении условия FALSE (ложь, «Нет») действий нет:

IF условие THEN команда

Действие условного оператора состоит в том, что вычисляется значение условия. Если оно равно TRUE, то выполняется команда, следующая за ключевым словом THEN, и осуществляется переход к следующему оператору программы. Если оно равно FALSE, то при отсутствии ключевого слова ELSE сразу осуществляется переход к следующему оператору программы.

Синтаксис условного блочного оператора для полного варианта разветвления, т.е. алгоритм предусматривает одно или более действий (серию команд 1) при значении условия TRUE (истина, «Да») и одно или более действий (серию команд 2) при значении условия FALSE (ложь, «Нет»):

IF условие THEN

серия команд 1

ELSE

серия команд 2

ENDIF

Действие условного оператора состоит в том, что вычисляется значение условия. Если оно равно TRUE, то выполняется серия команд 1, которые записаны в следующих строках программы после заголовка условного блочного оператора, закачивающегося ключевым словом THEN. Сигналом для дальнейшего перехода является ключевое слово ELSE. Тогда программа анализирует появление ключевого слова ENDIF и осуществляется выполнение следующего оператора за этим ключевым словом. Если значение условия равно FALSE, то выполняется серия команд 2, которые записаны в следующих строках программы после ключевого слова ELSE до ключевого слова ENDIF, а осуществляется выполнение следующего оператора за этим ключевым словом.

Синтаксис условного блочного оператора для неполного варианта разветвления («обход»), т.е. алгоритм предусматривает лишь одно или более действий (серию команд) при значении условия TRUE (истина, «Да»), а при значении условия FALSE (ложь, «Нет») действий нет:

IF условие THEN

серия команд

ENDIF

Действие условного оператора состоит в том, что вычисляется значение условия. Если оно равно TRUE, то выполняется серия команд, которые записаны в следующих строках программы после заголовка условного блочного оператора, закачивающегося ключевым словом THEN. Не встретив ключевого слова ELSE, программа анализирует появление ключевого слова ENDIF и осуществляется выполнение следующего оператора за этим ключевым словом. Если значение условия равно FALSE, но, не встретив ключевого слова ELSE, а, получив для анализа ключевое слово ENDIF, осуществляется выполнение следующего оператора за этим ключевым словом.

2.5.4. *Оператор безусловного перехода*

Для осуществления переходов в программе используется оператор безусловного перехода. Ключевое слово этого оператора GOTO, что в переводе с английского языка означает: *иди к*.

Синтаксис оператора безусловного перехода:

GOTO N

где N – номер строки или метки оператора, на который происходит переход в программе.

Этот оператор служит для перехода из одной строки программы к другой, помеченной номером или меткой.

Примеры использования безусловного оператора перехода на языке Бейсик:

Пример 1.

```
10  X=3
20  GOTO 40
30  Y=2*X
40  Z=5*X
```

Пример 2.

```
 X=3
GOTO W1
Y=2*X
W1  Z=5*X
```

2.5.5. *Программа разветвляющегося алгоритма*

Рассмотрим реализацию разветвляющегося алгоритма на примере задачи, приведенной в разделе 1.3, а именно: вычислить значение функции $y(x) = \begin{cases} -5, & при\ x \le 10 \\ x^3, & при\ x > 10 \end{cases}$ для заданного x.

```
REM вычисление значения функции y(x)
CLS
INPUT " Введите значения:  X "; X
IF X<=10 THEN Y=-5 ELSE Y=X^3
? “Y= “; Y
```

END

В приведенной программе использовался условный строчный оператор, который можно всегда заменить на условный блочный, а именно:

```
IF X<=10 THEN
Y=-5
ELSE
Y=X^3
ENDIF
```

В программе использован оператор комментария Его ключевое слово REM (от английского слова remark – замечать, отмечать). Транслятор, встретив это ключевое слово, игнорирует эту строку программы для анализа синтаксиса и выполнения. Этот оператор служит для внесения пояснений в программу как для ее автора, так и для людей, которые будут ее читать.

Для самостоятельной работы на построение блок-схемы и записи программы разветвляющегося алгоритма можно предложить следующие задачи:

Задача 1. Даны катеты одного треугольника (A1, B1) и катеты другого треугольника (A2, B2). Определить, будут эти треугольники равновеликими, т.е. имеют они равные площади?

Задача 2. Найти общее сопротивление двух проводников R1 и R2, если они соединены параллельно (R3) или последовательно (R4). Значения R1 и R2 вводятся с клавиатуры.

2.6. Операторы языка Бейсик для записи циклического алгоритма

При рассмотрении вопросов алгоритмизации задач было сказано, что все вычислительные процессы, выполняемые на компьютере по заданной программе, можно разделить на три основных вида:

- линейные;
- разветвляющиеся;
- циклические.

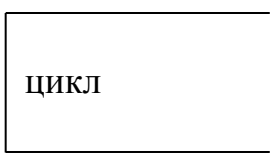
Напомним, что линейным принято называть вычислительный процесс, в котором операции выполняются последовательно, в порядке их записи в программе.

Вычислительный процесс называется разветвляющимся, если для его реализации предусмотрено несколько направлений (ветвей). В программе появляются

операторы передачи управления, которые могут изменять порядок выполнения операторов в программе. Это рассмотренные ранее оператор безусловного перехода и условные операторы.

Вычислительный процесс называется циклическим, если он содержит цикл. **Цикл** – это группа операторов, которые выполняются многократно путем возврата от последнего оператора этой группы к её первому оператору, т.е. это многократно повторяемый участок программы. Цикл может быть организован с использованием оператора безусловного перехода и условных операторов. Продемонстрируем это на примере.

Пример: Вычислить сумму $S = \sum_{1}^{1000} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{1000}$




```
CLS
S=0
N=0
1   N=N+1
    S=S+1/N
    IF N<=1000 THEN GOTO 1
PRINT "Сумма ="; S
END
```

Это так называемый цикл «ДО», т. е. проверка условия выхода из цикла осуществляется после выполнения тела цикла. Для его организации использован условный строчный оператор и оператор безусловного перехода. Не составит особого труда составить цикл, в котором проверка условия выхода из цикла осуществляется перед выполнением тела цикла, т. е. цикл «ПОКА».

Однако в языке программирования Бейсик, как и в других языках программирования высокого уровня, существует специальный оператор цикла, обеспечивающий более удобный способ организации циклов.

2.6.1. *Оператор цикла*

Оператор цикла является одним из наиболее важных операторов. Его эффективность особенно велика при использовании массивов, т. е. работе с элементами массивов.

Оператор цикла является структурным оператором, имеющим три части:

заголовок оператора цикла, тело цикла, завершающий оператор.

FOR I=NI **TO** KI **STEP** HI	*заголовок оператора цикла*
СЕРИЯ ОПЕРАТОРОВ	*тело цикла*
NEXT I	*завершающий оператор*

Оператор цикла содержит четыре ключевых слова: FOR, что в переводе с английского языка означает *для*, TO, что в переводе с английского языка означает *до*, STEP, что в переводе с английского языка означает *шаг* и NEXT, что в переводе с английского языка означает *следующий*.

Тело цикла состоит из последовательности исполняемых операторов, следующих за заголовком оператора цикла до завершающего оператора. В частном случае, тело цикла может состоять из одного оператора.

Напомним, I – имя параметра цикла (управляющая переменная цикла, такое определение более информативно: переменная, которая управляет циклом и входе его выполнения изменяет свое значение); NI – начальное значение параметра цикла (управляющей переменной цикла); KI – конечное значение параметра цикла (управляющей переменной цикла); HI – шаг изменения параметра цикла (управляющей переменной цикла). NI, KI и HI могут быть представлены любыми арифметическими выражениями, но значения переменных, которые входят в них, должны быть определены до выполнения цикла.

Работу оператора цикла можно описать следующим образом:

1. Вычисление значений выражений, определяющих NI, KI и HI;
2. I присваивается значение NI;
3. I сравнивается с KI: если I больше KI при положительном HI (или, наоборот, I меньше KI при отрицательном HI), то переход к выполнению оператора следующего за завершающим оператором цикла (NEXT I), иначе вход в цикл;
4. Выполняются операторы, входящие в тело цикла до завершающего оператора цикла (NEXT I);
5. I автоматически изменяется на HI;
6. Переход к пункту 3, т.е. как бы переход к заголовку цикла;

Иначе говоря, операторы FOR и NEXT обеспечивают изменение значения I от NI до KI с шагом HI и выполнение оператора (операторов), заключенных между FOR и NEXT, при каждом значении I.

Замечания:

1. Изменять значение величин NI, KI и HI в процессе выполнения оператора цикла не рекомендуется.

2. Вход в цикл, минуя заголовок оператора цикла, в Бейсике запрещен.

3. Значение HI не должно быть 0 (нуль), так как цикл будет бесконечным.

4. Если Значение HI равно 1, конструкцию STEP 1 можно опустить.

Пример: Вычислить сумму $S = \sum_{1}^{1000} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{1000}$


```
CLS
S=0
FOR  N=1  TO 1000  STEP  1
S=S+1/N
NEXT  N
PRINT  ”Сумма =”;  S
END
```

2.6.2. *Программа циклического алгоритма*

Рассмотрим реализацию разветвляющегося алгоритма на примере задачи, приведенной в разделе 1.3, а именно: вычислить значение y=sin φ в диапазоне изменения аргумента от 10° до 90° с шагом 10°.

```
CLS
INPUT " Введите значения:  NX"; NX
INPUT " Введите значения:  KX"; KX
INPUT " Введите значения:  HX"; HX
FOR  X=NX  TO  KX  STEP  HX
X1 = X*3.14/180
Y = SIN(X1)
? Y;"=sin(";X;")"
NEXT  X
END
```

В приведенной программе использовался оператор цикла, применение которого значительно проще, чем организация цикла с помощью условного оператора.

Для самостоятельной работы на построение блок-схемы и записи программы циклического алгоритма можно предложить следующие задачи:

Задача 1. Вычислить сумму (S) натуральных нечетных чисел от 1 до 99

(S=1+3+5+…+99).

Задача 2. Вычислить произведение (P) натуральных чисел от 1 до 10 (P=1*2*3*…*10=10! – это называется 10 факториал).

Список литературы

1. Гейн А.Г., Сенокосов А.И., Юнерман Н.А. Информатика: Учеб. пособие для 10– 11 кл.. общеобразоват. учреждений. – М.: Просвещение, 2000. – 255 с.
2. Голицына О.Л., Попов И.И. Основы алгоритмизации и программирование: Учеб. пособие (серия «Профессиональное образование»). – М.: ФОРУМ: ИНФРА-М, 2004. – 432 с.
3. Житкова В.Б., Кудрявцева Е.К. Алгоритмы и основы программирования. (Тематический контроль по информатике.). – М.: Интеллект-Центр, 1999. – 64 с.
4. Кириенко Д.П., Осипов П.О., Чернов А.В. ГИА-2013: Экзамен в новой форма: Информатика: 9-й класс: Тренировочные варианты экзаменационных работ для проведения государственной (итоговой) аттестации в новой форме. – М.: Астрель, 2013. – 78 с.
5. Колдаева Н.В. Основы информатики: Учебное пособие. – Кстово: НФВИУ, 2002. – 60 с.
6. Коханко В.В., Колдаева Н.В., Матвейчук. Основы программирования: Учебное пособие. – Кстово: НВВИКУ (ВИ), 2008. – 244 с.
7. Кузнецов А.А., Пугач В.И., Добудько Т.В., Матвеева Н.В. – 2-е изд., испр. – М.: БИНОМ, Лаборатория знаний, 2003. – 232 с.
8. Мамонтов Д.В. Quick Basic в задачах и примерах. – СПб.: Питер, 2006. – 256 с.
9. Острейковский В.А., Полякова И.В. Информатика. Теория и практика: Учеб. пособие. – М.: Издательство Оникс, 2008. – 608 с.
10. Островская Е.М., Самылкина Н.Н. ЕГЭ 2013. Информатика. Сдаем без проблем! – М.: Эксмо, 2012. – 160 с.
11. Самылкина Н.Н.. Островская Е.М., Кузнецова Е.Ю. ЕГЭ 2013. Информатика: тренировочные задания. – М.: Эксмо, 2012. – 200 с.
12. Светозарова Г.И., Мельников А.А., Козловский А.В. Практикум по программированию на языке бейсик: Учеб. пособие для вузов. – М.: Наука. Гл. ред. физ.-мат. лит., 1988. – 368 с.
13. Угренович Н.Д. Информатика и ИКТ. Базовый уровень: учебник для 11 класса. – 5-е изд. – М.: БИНОМ, Лаборатория знаний, 2012. – 187 с.
14. Хлебников А.А. Информатика: учебник (Среднее профессиональное

образование). – Изд. 3-е, стер. – Ростов н/Д.: Феникс, 2012. – 507 с.

15. Цветкова М.С. Информатика и ИКТ: учебник для нач. и сред. проф. образования. – 3-е изд. стер. – М.: Издательский центр «Академия», 2012. – 352 с.