



**Современный
Гуманитарный
Университет**

Дистанционное образование

Рабочий учебник

Фамилия, имя, отчество _____

Факультет _____

Номер контракта _____

ИНФОРМАТИКА. УГЛУБЛЕННЫЙ КУРС

ЮНИТА 2

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

МОСКВА 1999

Рекомендовано Министерством
общего и профессионального
образования Российской
Федерации в качестве учебного
пособия для студентов высших
учебных заведений

Курс: ИНФОРМАТИКА. УГЛУБЛЕННЫЙ КУРС

Юнита 1. Основы информационной культуры. Информация.

Юнита 2. Алгоритмизация и программирование.

Юнита 3. Технические и программные средства реализации информационных процессов.

Юнита 4. Информационные технологии.

ЮНИТА 2

Даны понятия алгоритма, алгоритмизации и программирования на языке BASIC.

Для студентов Современного Гуманитарного Университета

Юнита соответствует образовательной профессиональной программе № 2

ОГЛАВЛЕНИЕ

	стр.
ТЕМАТИЧЕСКИЙ ПЛАН	4
ЛИТЕРАТУРА	5
ПЕРЕЧЕНЬ УМЕНИЙ	6
ТЕМАТИЧЕСКИЙ ОБЗОР	7
1. Методика подготовки и решения задач на ЭВМ	7
2. Способы записи алгоритма	10
2.1. Алгоритм и его свойства	10
2.2. Изобразительные средства для описания алгоритмов	10
2.3. Схемы алгоритмов	13
3. Программирование	23
3.1. Принцип программного управления	23
3.2. Языки программирования	26
4. Язык программирования BASIC	30
4.1. BASIC - немного истории	30
4.2. Алфавит языка	32
4.3. Переменные и константы	32
4.4. Выражения и операции	34
4.5. Первая программа на QBASIC	37
4.6. Основные операторы языка	38
5. Программирование основных алгоритмических структур на языке QBASIC	46
5.1. Программы линейной структуры	46
5.2. Программирование алгоритмов разветвляющейся структуры	47
5.3. Программирование алгоритмов циклической структуры	51
5.4. Программирование вложенных циклов	64
5.5. Вывод результатов в виде графиков и таблиц	70
5.6. Работа в среде программирования QBASIC	73
6. Модульное программирование	75
6.1. Подпрограммы	75
6.2. Функции	78
6.3. Метод пошаговой детализации	79
ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ	83
ТРЕНИНГ УМЕНИЙ	84
ФАЙЛ МАТЕРИАЛОВ	104
ГЛОССАРИЙ*	

* Глоссарий расположен в середине учебного пособия и предназначен для самостоятельного заучивания новых понятий.

ТЕМАТИЧЕСКИЙ ПЛАН

Этапы решения задач на ЭВМ. Алгоритм. Свойства алгоритмов. Способы записи алгоритмов: словесный, структурно–стилизированный, графический, программный. Правила записи схем алгоритмов. Основные структуры алгоритмов: следование, повторение, ветвление. Структурный подход к разработке алгоритмов.

Система программирования. Языки программирования. Транслятор. Компилятор. Ассемблер. Интерпретатор. Уровень языка.

Программа на языке Бейсик. Алфавит языка QBASIC. Данные. Типы данных на QBASIC: целый, вещественный, символьный .

Выражения и операции. Основные операторы языка: присваивания, ввода и вывода данных, условный, цикла, передачи управления, выбора, перехода, повторений.

Представление основных структур программирования. Программирование линейной, разветвляющейся, циклической структур на языке QBASIC. Работа в среде программирования QBASIC.

Модульное программирование. Подпрограмма. Функции, определяемые пользователем. Метод пошаговой детализации.

ЛИТЕРАТУРА

Базовая

*1. Петров А.В., Алексеев В. Е., Ваулин А. С. и др., Вычислительная техника и программирование. М., 1990.

*2. Алексеев В.Е., Ваулин А.С., Петрова Г.Б. Вычислительная техника и программирование. М., 1991.

Дополнительная

*3. Зельднер Г.А. Quick BASIC для носорога. М., 1994.

*4. Светозарова Г.И., Мельников А.А., Козловский А.В. Практикум по программированию на языке Бейсик. М., 1988.

*5. Уолш Б. Программирование на Бейсике. М., 1988

6. ЭВМ. В 8 т / Под ред. А.Я. Савельева М., 1987. Т 7.

7. ЭВМ. В 8 т / Под ред. А.Я.Савельева М., 1987. Т 3.

8. ЭВМ. В 8 т / Под ред. А.Я. Савельева М., 1991. Т 2.

9. Богумирский Б.С. Руководство пользователя ПЭВМ. Ч. 1. Спб., 1994.

10. Ван Тассел Д. Стил, разработка, эффективность, отладка и испытание программ. М., 1985

Примечание. Знаком (*) отмечены работы, выдержками из которых сформирован тематический обзор.

ПЕРЕЧЕНЬ УМЕНИЙ

№	Умение	Алгоритмы
1	2	3
1	Разрабатывать алгоритмы линейной, разветвляющейся и циклической структуры.	1. Поставить задачу. 2. Выбрать метод решения. 3. Определить последовательность действий, ведущих к получению результатов. 4. Дать точное предписание вычислительного процесса в виде последовательно размещенных блоков в соответствии с ГОСТом.
2	Программировать алгоритмы линейной структуры.	1. Разработать алгоритм линейной структуры. 2. Определить наименования переменных в программе. 3. Записать алгоритм с помощью операторов языка Бейсик (ввод-вывод, присваивание).
3	Программировать алгоритмы разветвляющейся структуры.	1. Разработать алгоритм разветвляющейся структуры. 2. Определить наименования переменных в программе. 3. Записать алгоритм с помощью операторов языка Бейсик, используя операторы IF...THEN, IF...THEN...ELSE, SELECT...END SELECT.
4	Программировать алгоритмы циклической структуры.	1. Разработать алгоритм циклической структуры, используя структуры цикла типа «Пока» или типа «До». 2. Определить наименования переменных в программе. 3. Записать алгоритм с помощью операторов языка Бейсик, используя операторы цикла WHILE...WEND, FOR...NEXT, DO...LOOP.
5	Разрабатывать программы обработки массивов.	1. Разработать алгоритм решения задачи. 2. Определить наименования переменных в программе. 3. Составить описание массива. 4. Записать алгоритм с помощью операторов языка Бейсик.
6	Использовать подпрограммы при разработке программ.	1. Разработать общую структуру программы. 2. Разработать алгоритм подпрограммы. 3. Записать алгоритм подпрограммы с помощью операторов языка Бейсик. 4. Определить формальные и фактические параметры. 5. Записать алгоритм общей программы на языке Бейсик.

1. МЕТОДИКА ПОДГОТОВКИ И РЕШЕНИЯ ЗАДАЧИ НА ЭВМ

Решение задачи на ЭВМ – сложный и трудоемкий процесс. Любая задача начинается с постановки задачи. На основе словесной формулировки задачи выбираются переменные, подлежащие определению, записываются ограничения, связи между переменными, в совокупности образующие математическую модель решаемой проблемы. Анализируется метод решения. На этом этапе необходимо принять очень важное решение – использовать ли имеющееся готовое программное обеспечение или разрабатывать собственную программу. Дешевле и быстрее использовать имеющиеся в наличии готовые разработки. Обновление программного обеспечения – задача программистов. В этом случае традиционно выделяются следующие основные *этапы решения* задачи на ЭВМ [1] :

- 1) постановка задачи, разработка математической модели;
- 2) выбор метода численного решения;
- 3) разработка алгоритма и структуры данных;
- 4) проектирование программы;
- 5) производство окончательного программного продукта;
- 6) решение задачи на ЭВМ.

Постановка задачи – точное описание исходных данных, условий задачи и целей ее решения. На этом этапе многие из условий задачи, заданные в форме различных словесных описаний, необходимо выразить на точном (формальном) языке математики. Часто задача программирования задается в математической формулировке, поэтому необходимость в выполнении этапов 1 и 2 отпадает. Для решения достаточно сложных задач этап формализации может потребовать значительных усилий и времени. Среди опытных программистов распространено мнение, что выполнить этап формализации – это значит сделать половину всей работы по созданию программы.

Выбор метода решения тесно связан с постановкой задачи. На первом этапе задача сводится к математической модели, для которой известен метод решения. Метод численного решения сводит решение задачи к последовательности арифметических и логических операций. Однако возможно, что для полученной модели известны несколько методов решения и тогда предстоит выбрать лучший. Можно усовершенствовать существующий или разработать новый метод решения формализованной задачи. Эта работа по своему характеру является научно-исследовательской и может потребовать значительных усилий. Разработкой и изучением таких методов занимается раздел математики, называемый численным анализом.

При выборе метода надо учитывать требования, предъявляемые постановкой задачи, и возможности его реализации на конкретной ЭВМ: точность решения, быстроту получения результата, требуемые затраты оперативной памяти для хранения исходных и промежуточных данных и результатов.

Алгоритм устанавливает последовательность точно определенных действий, приводящих к решению задачи. При этом последовательность действий может задаваться посредством словесного или графического описаний. Если выбранный для решения задачи численный метод реализован в виде стандартной библиотечной подпрограммы, то алгоритм обычно сводится к описанию и вводу исходных данных, вызову стандартной подпрограммы и выводу результатов на

* Жирным шрифтом выделены новые понятия, которые необходимо усвоить. Знание этих понятий будет проверяться при тестировании.

экран или на печать. Более характерен случай, когда стандартные подпрограммы решают лишь какую-то часть задачи. Здесь эффективным подходом является разделение сложной исходной задачи на некоторые подзадачи, реализующиеся отдельными модулями. Определяется общая структура алгоритма, взаимодействие между отдельными модулями, детализируется логика. Этот этап тесно связан со следующим этапом проектирования программы.

Проектирование программы включает в себя несколько подзадач. Во-первых, необходимо выбрать язык программирования. Во-вторых, определить, кто будет использовать разработанное программное обеспечение и каким должен быть интерфейс (средство общения с пользователем). В-третьих, решить все вопросы по организации данных. В-четвертых, кодирование, т. е. описание алгоритмов с помощью инструкций выбранного языка программирования. Если задача, для которой разрабатывается алгоритм, сложная, то не следует сразу пытаться разрешить все проблемы. Сложившийся в настоящее время подход к разработке сложных программ состоит в последовательном использовании принципов проектирования сверху вниз, модульного и структурного программирования.

Окончательный программный продукт получается после отладки и испытания программы. При программировании и вводе данных с клавиатуры могут быть допущены ошибки. Их обнаружение, локализацию и устранение выполняют на этапе отладки и испытания (тестирования) программы. Причем могут быть допущены логические ошибки и на этапе постановки задачи, и на этапе алгоритмизации. В этом случае необходимо вернуться к предыдущим этапам. Дорабатывать и улучшать программу можно в течение всего жизненного цикла программного продукта.

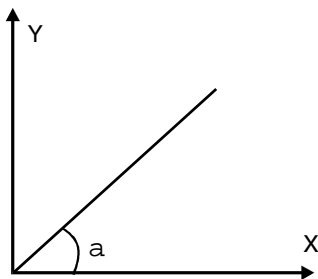
Решение задачи на ЭВМ – выполнение всех предусмотренных программой вычислений и вывод результатов расчета на экран дисплея или на печать.

Пример. Построить траекторию движения тела, брошенного под углом к горизонту, и определить дальность бросания.

Постановка задачи, разработка математической модели. Для упрощения модели примем следующие допущения:

- а) пренебрегаем кривизной Земли, считая ее поверхность плоскостью;
- б) ускорение свободного падения считаем константой;
- в) пренебрегаем сопротивлением воздуха;
- г) пренебрегаем движением Земли.

Математическое описание объекта. Уравнения движения тела, брошенного под углом к горизонту, в Декартовых координатах записывается следующим образом:



$$\begin{aligned} Y &= V_0 t \sin \alpha - \frac{gt^2}{2}; \\ X &= V_0 t \cos \alpha. \end{aligned} \quad (1)$$

Рис. 1.

Y, X – вертикальная и горизонтальная составляющие движения;
g – ускорение свободного падения;
t – время.

Создание математической модели позволяет поставить задачу математически.

Пусть движение тела, брошенного под углом α к горизонту с начальной скоростью V_0 (рис. 1), описывается системой уравнений (1). α и V_0 заданы. Требуется определить дальность полета и положение тела в каждый момент времени от $t = 0$ с шагом h .

Вторым этапом решения задачи можно пренебречь, так как никакого особенного численного метода решения в данном случае применять не нужно. Точка падения определяется из системы уравнений при условии $Y = 0$.

$$x = \frac{V_0^2}{g} \sin 2\alpha.$$

Изменяя значение t с шагом h , рассчитываем координаты X , Y .
Алгоритм решения лучше всего представить в виде схемы (рис. 2).

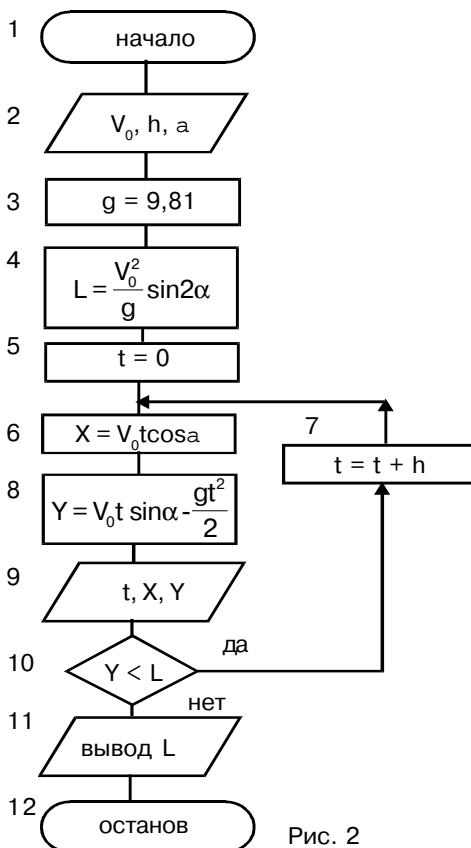


Рис. 2

На следующем этапе алгоритм записывается на языке программирования и текст программы переносится на носитель, с которого она вводится в ЭВМ. Часто программа вводится с клавиатуры дисплея в оперативную память компьютера, а затем записывается на диск.

По готовой программе производятся расчеты и результаты сравниваются с контрольным примером. Контрольным примером могут служить экспериментальные данные. Если в процессе отладки программы возникают ошибки, то их необходимо исправить.

2. СПОСОБЫ ЗАПИСИ АЛГОРИТМА

2.1. Алгоритм и его свойства

В основе решения любой задачи лежит понятие алгоритма. Под **алгоритмом** принято понимать “точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к искомому результату” (ГОСТ 19.781–74).

При составлении алгоритмов следует учитывать ряд требований, выполнение которых приводит к формированию необходимых свойств.

Алгоритм должен быть однозначным, исключая произвольность толкования любого из предписаний и заданного порядка исполнения. Это свойство алгоритма называется **определенностью**.

Реализация вычислительного процесса должна через определенное число шагов привести к выдаче результатов или сообщения о невозможности решения задачи. Это свойство алгоритма называется **результативностью**.

Решение однотипных задач с различными исходными данными можно осуществлять по одному и тому же алгоритму, что дает возможность создавать типовые программы для решения задач при различных вариантах задания значений исходных данных. Это свойство алгоритма называется **массовостью**.

Предопределенный алгоритмом вычислительный процесс можно расчленить на отдельные этапы, элементарные операции. Это свойство алгоритма называется **дискретностью**.

Алгоритмизация – техника составления алгоритмов и программ для решения задач на ЭВМ.

2.2. Изобразительные средства для описания алгоритмов

К изобразительным средствам описания алгоритмов относятся следующие основные способы их представления:

- а) словесный (записи на естественном языке);
- б) структурно–стилизированный (записи на языке псевдокода);
- в) программный (тексты на языках программирования);
- г) графический (схемы графических символов).

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных и задается в произвольном изложении на естественном языке. Способ основан на использовании общепринятых средств общения между людьми и, с точки зрения написания,

трудностей для авторов алгоритмов не представляет. Однако для “исполнителей” такие описания алгоритмов часто неприемлемы. Они строго не формализуемы, страдают многословностью записей, допускают неоднозначность толкования отдельных предписаний. Поэтому такой способ описания алгоритмов не имеет широкого распространения.

Пример. *Записать алгоритм нахождения наибольшего общего делителя двух натуральных чисел (m и n) на естественном языке.*

При таком словесном способе содержание алгоритма может быть следующим:

- 1) если числа равны, то необходимо взять любое из них в качестве ответа, в противном случае – продолжить выполнение алгоритма;
- 2) определить большее из чисел;
- 3) заменить большее число разностью большего и меньшего чисел;
- 4) повторить алгоритм с начала.

Структурно–стилизированный способ записи алгоритмов основан на формализованном представлении предписаний, задаваемых путем использования ограниченного набора типовых синтаксических конструкций. Такие средства описания алгоритмов часто называются **псевдокодами**. Разновидностью структурно–стилизированного способа описания алгоритмов является известный школьникам алгоритмический язык в русской нотации (АЯРН).

Пример. *Приведем описанный на АЯРН алгоритм решения задачи об определении принадлежности точки D треугольнику ABC .*

алг Определение принадлежности точки треугольнику (действ $x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D$ целое z лит a);
арг $x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D$;
рез z, a ;

нач

действ S_1, S_2, S_3, S_4

вычислить значение S_1 , равное площади тр–ка ABC ;

вычислить значение S_2 , равное площади тр–ка ABD ;

вычислить значение S_3 , равное площади тр–ка ACD ;

вычислить значение S_4 , равное площади тр–ка CDB ;

если $S_1 = S_2 + S_3 + S_4$

то $z := 1$,

$a :=$ “точка внутри треугольника”,

иначе $z := 0$,

$a :=$ “точка вне треугольника”,

все

напечатать значение a ;

кон

Программный способ записи алгоритмов – это алгоритм, записанный на языке программирования, позволяющем на основе строго определенных правил формировать последовательность предписаний, однозначно отражающих смысл и содержание частей алгоритма с целью их последующего исполнения на ЭВМ.

Пример. Программа на языке Бейсик перевода температуры из градусов Цельсия в градусы Фаренгейта.

```

PRINT "Перевод температуры из град. Цельсия в град. Фаренгейта"
6 PRINT "Укажите температуру в град. Цельсия"
INPUT C
IF C = 9999 THEN 7
F = C*1.8 + 32
PRINT C, F
GOTO 6
7 END

```

Для **графического изображения алгоритмов** используются *графические символы*. Наиболее распространенными являются блочные символы (блоки), соединяемые линиями передач управления. Существует государственный стандарт на выполнение графической записи алгоритма. Графическая запись алгоритма является наиболее наглядной. Перечень условных графических символов, их наименования, форма, размеры и отображаемые функции устанавливаются ГОСТ 19.003–80. Основные графические символы, используемые для описания алгоритмов, приведены в [1, 2].

Схемы могут быть представлены также в виде структограмм или по имени их авторов, диаграммами Нэсси – Шнейдермана [7].

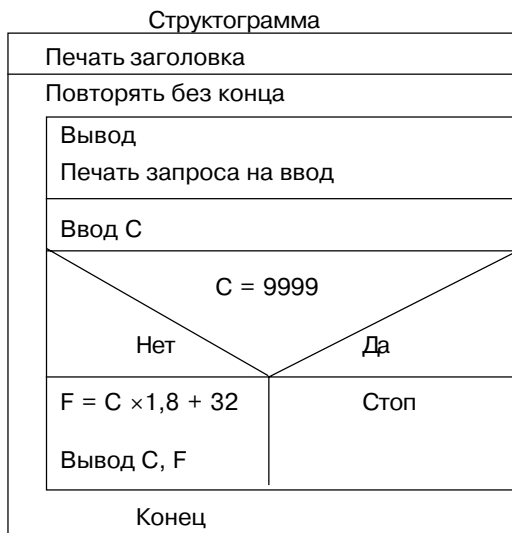
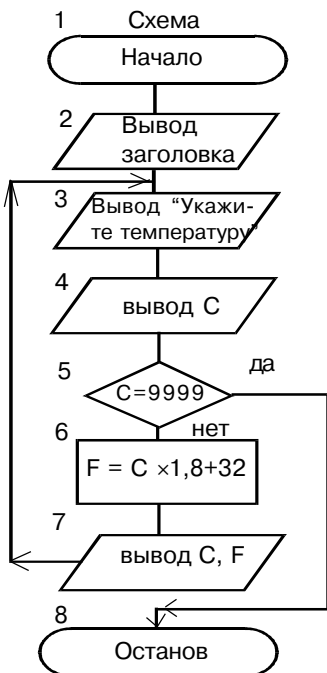


Рис. 3. Схема и структограмма перевода температуры из шкалы Цельсия в шкалу Фаренгейта

На рис. 3 приведена схема алгоритма перевода температуры из шкалы Цельсия в шкалу Фаренгейта. Перевод осуществляется по формуле:

Температура по Фаренгейту = (температура по Цельсию) $\times 180/100 + 32$.

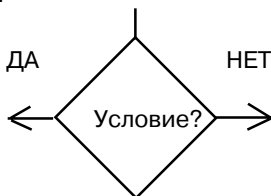
2.3. Схемы алгоритмов

Использование схем позволяет представить алгоритм в наглядной форме, поэтому они наиболее часто используются.

Вычислительный блок представляет собой прямоугольник, в котором записываются расчетные формулы. Причем формула должна быть записана таким образом, что вычисляемая переменная записывается слева, далее идет знак равенства (в данном случае этот знак называется присваиванием), далее – расчетная формула.

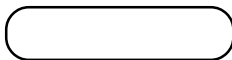


Проверка условия изображается ромбом, внутри которого записывается это условие. В результате проверки выбирается один из двух возможных путей вычислительного процесса.

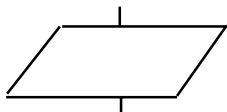


Если условие выполняется, то есть имеет значение ДА, то следующим выполняется этап по стрелке ДА. Если условие не выполняется, то осуществляется переход по стрелке НЕТ.

Начало и конец вычислительного процесса изображаются овалом, в котором записываются слова “Начало” или “Останов”.

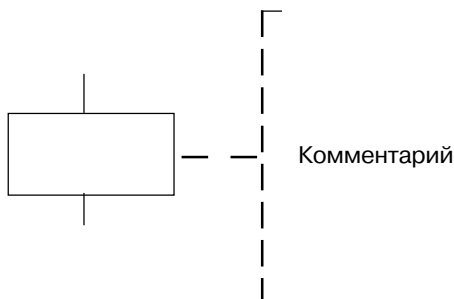


При решении задач на ЭВМ исходные данные задаются при вводе в машину. *Ввод данных* может осуществляться разными способами, например, с клавиатуры, с перфоленты, с диска и т. д. Задание численных значений исходных данных мы будем называть вводом, а фиксацию результатов расчета – выводом. Ввод исходных данных и вывод результатов изображаются параллелограммом. Внутри него пишется слово “Ввод” или “Вывод” и перечисляются переменные, подлежащие вводу или выводу.



Параллелограммом обозначается ввод – вывод, не привязанный к какому-либо конкретному устройству. Если ввод или вывод осуществляется с использованием конкретных устройств, то блоки ввода – вывода изображаются с помощью специальных фигур [2].

Комментарий используется в тех случаях, когда пояснение не помещается внутри блока.



По стандарту высота блока равна a , ширина $2a$, где размер a выбирается из ряда 10, 15, 20 мм. Блоки “начало” и “конец” имеют высоту $0.5a$.

Линии потока проводят параллельно внешним краям рамки листа. Направление линий потока сверху вниз и слева направо принимают за основное; если линии потока основного направления не имеют изломов, то их направление стрелками можно не обозначать. В остальных случаях направление линий потока обозначать стрелкой обязательно. Записи внутри символа или рядом с ним должны выполняться машинописью или чертежным шрифтом и должны быть краткими. В левом верхнем углу в разрыве линий ставится номер блока.

В настоящее время основная тенденция в использовании схем алгоритмов состоит не столько в указании последовательности операций, сколько в группировании блочных символов в виде базовых управляющих конструкций. К ним относятся *следование, ветвление и повторение*. **Основные структуры алгоритмов** – это ограниченный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий. Такие структуры рекомендуются при использовании так называемого структурного подхода к разработке алгоритмов и программ. Структурный подход предполагает использование только нескольких основных структур, комбинация которых дает все многообразие алгоритмов и программ.

2.3.1. Алгоритмы линейной структуры

Алгоритм линейной структуры (следование) – алгоритм, в котором все действия выполняются последовательно друг за другом. Такой порядок выполнения действий называется естественным.

Рассмотрим несколько примеров.

Задача 1. Определить площадь треугольника по формуле Герона

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

где a, b, c – длины сторон;

$p = (a + b + c)/2$ – полупериметр треугольника.

Для того чтобы рассчитать S , необходимо иметь численные значения p, a, b, c . Мы можем рассчитать p по формуле, а вот значения a, b, c должны быть заданы заранее, иначе задачу решить невозможно.

Запишем словесный алгоритм.

1. Задать численные значения a, b, c .

2. Вычислить p по формуле:

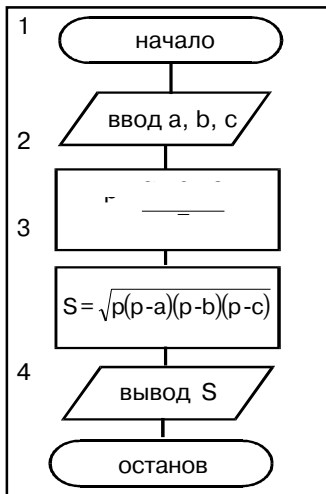
$$p = (a + b + c)/2.$$

3. Вычислить S по формуле:

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

4. Зафиксировать результат.

Схема представляет собой последовательность блоков, соединенных линиями потоков. Направление потока задается стрелкой, но стрелка не ставится, если направление потока сверху вниз и слева направо. В левом верхнем углу в разрыве линий ставится номер блока.



Внутри блока ввода записывается слово “Ввод” и перечисляются исходные данные (имена переменных), которые задаются извне. Внутри блока вывода записывается слово “Вывод” и перечисляются переменные, которые являются результатом расчета.

Рис. 4. Схема алгоритма линейной структуры

2.3.2. Алгоритмы разветвляющейся структуры

На практике редко удастся представить схему алгоритма решения задачи в виде линейной структуры. Часто в зависимости от каких-либо значений промежуточных результатов необходимо организовать вычисление либо по одним, либо по другим формулам. **Ветвление** – такая схема, в которой предусмотрено разветвление указанной последовательности действий на два направления в зависимости от итога проверки заданного условия. В схемах такой структуры используется логический блок (рис. 5).

Задача 2.

Рассчитать Y .

$$Y = \begin{cases} -X, & X < 0, \\ X^2, & X \geq 0. \end{cases}$$

Разработка алгоритма. В этой задаче должно быть задано X . Далее анализируется X . Если $X < 0$, то вычисления производятся по первой формуле, если это условие не выполняется, то выполняется второе условие $X \geq 0$, так как условия $X < 0$ и $X \geq 0$ взаимоисключающие, и Y вычисляется по второй формуле.

Словесный алгоритм решения этой задачи будет выглядеть следующим образом.

1. Задать численное значение для X .
 2. Проверить условие $X < 0$:
если условие выполняется перейти к п. 5;
если условие не выполняется перейти к п. 3.
 3. Вычислить Y по формуле $Y = X^2$.
 4. Перейти к пункту 6.
 5. Вычислить Y по формуле $Y = -X$.
 6. Зафиксировать вычисленное Y .
- Схема данного алгоритма представлена на рис. 5.

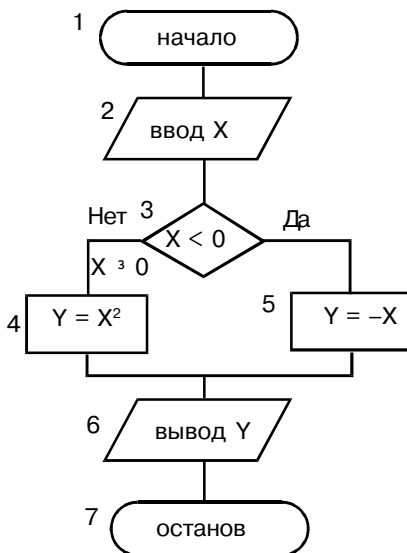


Рис. 5. Схема алгоритма разветвляющейся структуры

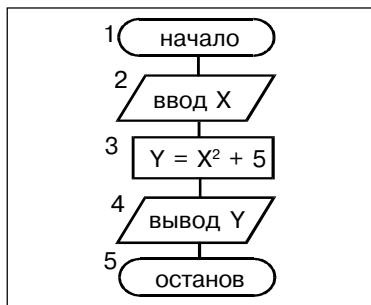
Рекомендуется под словом “нет” записывать условие, противоположное проверяемому.

2.3.3. Алгоритмы циклической структуры

Алгоритмы, отдельные действия в которых многократно повторяются, называются **алгоритмами циклической структуры (повторение)**. Совокупность действий алгоритма, связанную с повторением, называют **циклом**.

Приведем пример схемы алгоритма циклической структуры.

Задача 3. Вычислить множество значений функции $Y = X^2 + b$ для всех значений X от -10 до 10 с шагом 2 , при $b = 5$.



Разработка алгоритма. Значения Y необходимо вычислить 11 раз, то есть необходимо 11 раз выполнить алгоритм линейной структуры (рис. 6).

Рис. 6

Задание X можно автоматизировать, организовав цикл. Для этого необходимо задать начальное значение X , т. е. $X = -10$. Далее рассчитать Y по формуле, вывести численное значение Y , изменить X и вернуться к расчету Y .

Тогда схема будет выглядеть следующим образом (рис. 7).

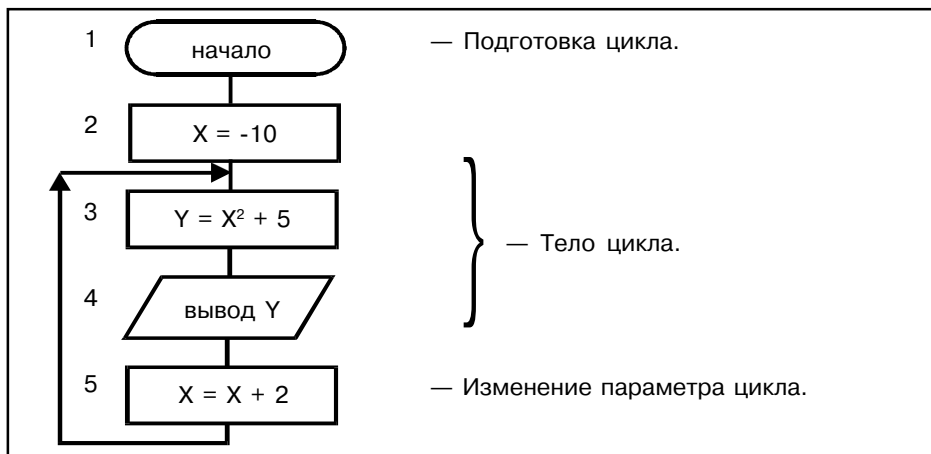


Рис. 7.

На рис. 7 представлена схема алгоритма циклической структуры. Блоки 3, 4, образующие тело цикла, повторяются многократно. Сколько раз? Бесконечное количество. При каждом расчете к предыдущему значению X прибавляется 2, далее следует возврат к расчету Y , вывод Y и опять X изменяется на 2. По условию задачи расчетом Y при $X = 10$ нужно ограничиться. Следовательно, необходимо

включить условие окончания расчетов. До тех пор, пока $X \leq 10$, расчеты производить; как только X станет больше 10 – вычисления закончить. В схему включим логический блок (рис. 7).

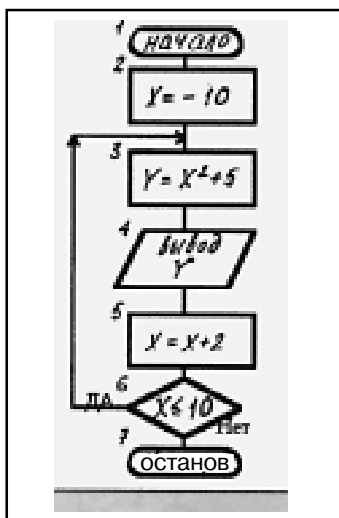


Рис. 8. Схема алгоритма циклической структуры

Чтобы алгоритм стал более универсальным, начальное значение X_n , конечное значение X_k и шаг изменения ΔX зададим в блоке ввода (рис. 9).

Величина, с изменением которой связано многократное выполнение цикла, называется **параметром цикла**. В нашем примере это X . Блоки 4, 5 – тело цикла. Блок 3 представляет собой подготовку цикла. Блок 6 – изменение параметра цикла (подготовка очередного шага), а блок 7 – условие продолжения цикла.

В блоке 2 осуществляется задание начального значения для X . В блоке 3 рассчитываются значения Y . В блоке 5 фиксируется текущее значение X с заданным шагом. В блоке 6 анализируется величина X . Если X еще не превысил своего конечного значения, то необходимо вернуться к блоку 3 и повторить вычисления. Если X стал больше предельного значения, расчеты нужно закончить.

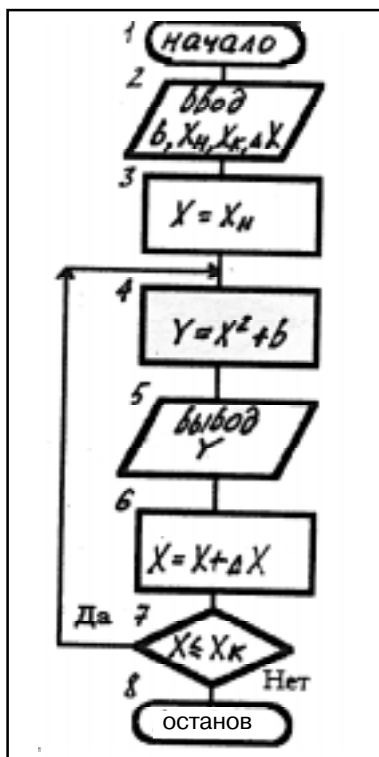


Рис. 9.

Представим схемы циклов в общем виде.

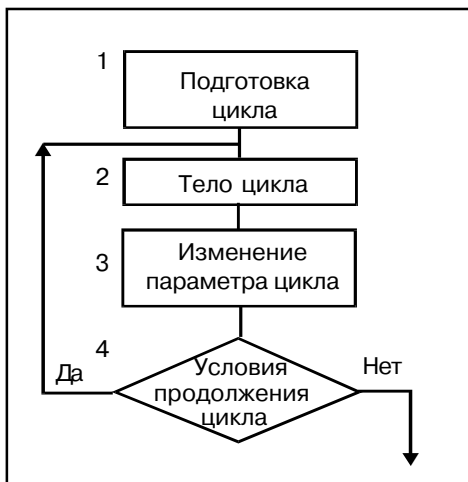


Рис. 10

Такая циклическая структура называется циклом “До” (рис. 10). Особенность этого цикла состоит в том, что он выполняется хотя бы один раз, так как первая проверка условия выхода из цикла происходит после того, как тело цикла выполнено.

Существует еще цикл “Пока” (рис. 11).

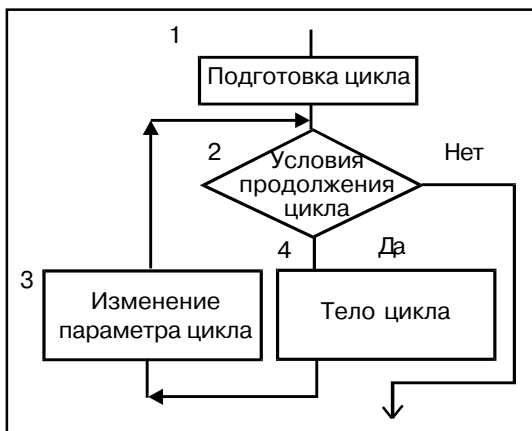


Рис. 11.

Цикл “Пока” отличается от цикла “До” тем, что здесь проверка условия проводится до выполнения тела цикла. Если при первой проверке условие выхода из цикла выполняется, то тело цикла не выполняется ни разу.

Используем цикл типа “Пока” для нашего примера (рис. 12).

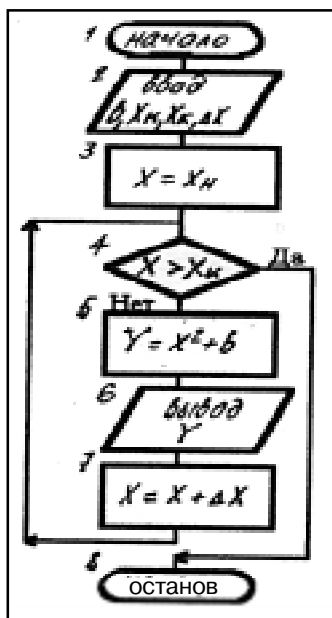


Рис. 12.

Для изображения алгоритмов циклической структуры используется также блок “**модификация**”. В блоке “модификация” объединяются несколько блоков: подготовка цикла, проверка окончания, изменение параметра цикла (подготовка очередного шага). В блоке модификации записывается параметр цикла, знак равенства (присваивание), начальное значение параметра цикла, конечное значение параметра цикла и шаг изменения параметра цикла. Для нашего примера схема алгоритма с использованием блока модификации выглядит следующим образом (рис. 13):

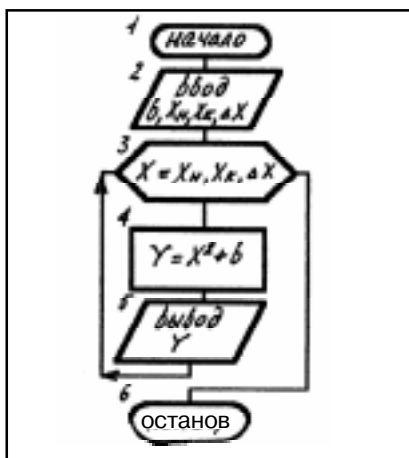


Рис 13.

Блок 3 можно прочесть таким образом: выполнить для всех X от X_n до X_k с шагом ΔX . Тело цикла выделено линией потока, замкнутой на блок модификации.

2.3.4. Алгоритмы со структурой вложенных циклов

Внутри одного цикла могут находиться один или несколько других циклов. В этом случае охватывающий цикл называется **внешним**, а вложенные в него циклы - **внутренними**. Правила организации как внешнего, так и внутренних циклов аналогичны правилам организации простого цикла. Параметры внешнего и внутреннего циклов изменяются не одновременно, т.е. при одном значении параметра внешнего цикла параметр внутреннего последовательно принимает все возможные значения. При организации вложенных циклов необходимо следить за тем, чтобы область действия внутреннего цикла не выходила за область действия внешнего цикла.

Задача 4. Составить схему алгоритма для вычисления и печати значения выражения

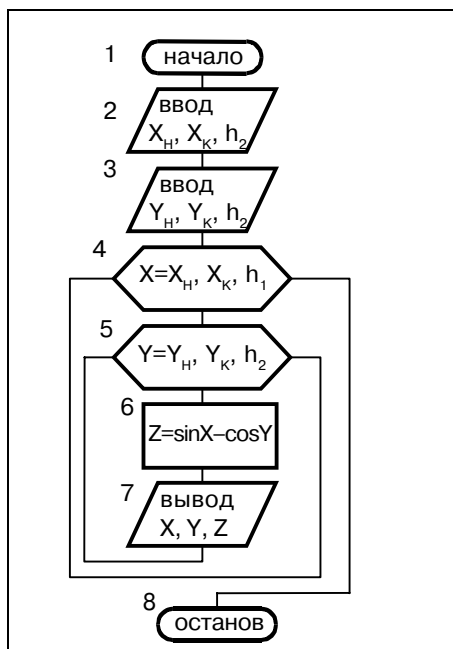
$$Z = \sin X - \cos Y$$

для всех значений X от $X_{\text{нач}}$ до $X_{\text{кон}}$ с шагом h_1 и для всех значений Y от $Y_{\text{н}}$ до $Y_{\text{к}}$ с шагом h_2 .

Такие вычисления значений заданной функции называются табулированием.

Выполняя табулирование, необходимо последовательно перебрать все значения X в указанном диапазоне и для каждого X перебрать все значения Y в диапазоне изменения Y .

Схема алгоритма

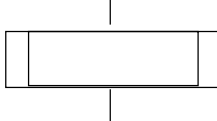


Блок модификации 4 (рис. 14) организует внешний цикл по X . X меняет значения от $X_{\text{н}}$ до $X_{\text{к}}$ с шагом h_1 . Телом внешнего цикла являются блоки 5, 6, 7. При каждом значении X Y пробегает все значения от $Y_{\text{н}}$ до $Y_{\text{к}}$ с шагом h_2 . Блок модификации 5 организует внутренний цикл по Y . Телом цикла по Y являются блоки 6 и 7.

Рис. 14

2.3.5. Подчиненные алгоритмы

При записи алгоритмов могут использоваться алгоритмы, составленные раньше. Алгоритмы, целиком используемые в составе других алгоритмов, называются **подчиненными алгоритмами или подалгоритмами**. Не исключено, что алгоритм, содержащий в своем описании подчиненные алгоритмы, сам может выступать в роли подалгоритма. В схемах подалгоритм изображается символом



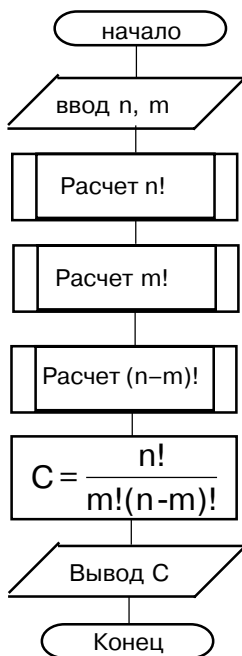
Задача 5. Составить алгоритм вычисления числа сочетаний. Число сочетаний

рассчитывается по формуле: $C_n^m = \frac{n!}{m!(n-m)!}$

Вычисление факториала оформить в виде подалгоритма.

В блоках “начало” и “останов” в подчиненном алгоритме записываются слова “вход” и “выход” (рис. 15 В).

А. Схема алгоритма



В. Схема подалгоритма

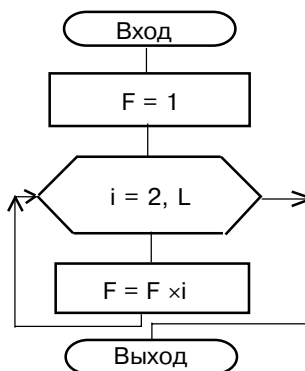


Рис. 15

В данной задаче необходимо трижды вычислить факториал: $n!$, $m!$ и $(n - m)!$. Для расчета факториала разработан подалгоритм. Вместо полной записи

последовательности блоков подалгоритма, которая должна повторяться трижды в основной схеме, введены блоки обращения к подалгоритму (рис. 15).

Использование подалгоритмов находит широкое применение в практике алгоритмизации и является одним из наиболее значительных и интересных приемов.

Одним из приемов разработки алгоритма решения более сложных задач является **метод пошаговой детализации**, когда первоначально продумывается и фиксируется общая структура алгоритма без детальной проработки отдельных его частей, но при этом также используются лишь основные структуры алгоритмов. Блоки, требующие дальнейшей детализации, обозначаются пунктирной линией. Далее прорабатываются (детализируются) отдельные блоки, не детализированные на предыдущем шаге. Полностью закончив детализацию всех блоков, получим решение всей задачи в целом.

3. ПРОГРАММИРОВАНИЕ

3. 1. Принцип программного управления

Основой функционирования ЭВМ является **принцип программного управления**, суть которого заключается в следующем. Программа, представляющая собой последовательность команд, реализующих алгоритм решения задачи, вводится в память ЭВМ, после чего начинается ее автоматическое выполнение с первой команды. После каждой выполненной команды машина автоматически переходит к выполнению следующей команды, и так до тех пор, пока не встретится команда, которая обычно предписывает закончить вычисления.

Структура команды ЭВМ в простейшем случае включает в себя две части : операционную и адресную. *Операционная часть* содержит код операции КОП (сложить, вычесть, ...). *Адресная часть* содержит адреса ячеек памяти (A1, A2, ...), в них хранятся значения операндов, с которыми надо выполнить заданную операцию. В зависимости от числа адресов, указанных в команде, различают одно-, двух-, трехадресные команды.

КОП	A		
КОП	A1	A2	
КОП	A1	A2	A3

Операционная часть Адресная часть

Команда может храниться в памяти ЭВМ, как обычное слово длиной 2, 3, 4 байт и. т. д. Длина команды в байтах обычно зависит от ее структуры и числа разрядов, отведенных под адресную и операционную части. Восемь двоичных разрядов, отведенных под код операции, позволяют закодировать 256 различных команд, что обычно является достаточным. Длина адресной части команды зависит от числа адресов и числа разрядов, отведенных для одного адреса. Так, чтобы

иметь возможность указать в команде адрес любой ячейки оперативной памяти емкостью 64 Кбайт, необходимо под адрес отвести 16 двоичных разрядов ($2^{16} = 64 \text{ Кб}$).

Рассмотрим пример составления простейшей программы для ЭВМ.

Пусть требуется вычислить выражение

$$Y = C \cdot X^2 + B.$$

Алгоритм решения задач:

1. Ввести в оперативную память значения исходных переменных C , X , B .
2. Вычислить X^2 . Присвоить переменной Y значение полученного результата.
3. Вычислить $Y \cdot C$. Присвоить переменной Y значение полученного результата.
4. Вычислить $Y + B$. Присвоить переменной Y значение результата.
5. Напечатать значение Y .

Для составления программы решения рассмотренной задачи необходимо, чтобы система команд ЭВМ включала следующие команды: ввести, умножить, запомнить, сложить, напечатать.

Будем считать, что команды используемой ЭВМ двухадресные и в общем виде записываются так:

$K A_1 A_2$,

где K - код операции; A_1, A_2 - значения первого и второго адреса команды (в некоторых командах A_1 и A_2 могут быть не только адресами операндов, но и операндами).

Примем, что коды операций представляются двухзначными числами и имеют следующие значения:

- | | |
|---------------------------|------------------|
| 01 - ввести с клавиатуры, | 04 - запомнить, |
| 02 - сложить, | 05 - напечатать, |
| 03 - умножить, | 06 - остановить. |

Уточним особенности выполнения каждой команды и использования их адресной части.

ВВЕСТИ - вводятся с клавиатуры N чисел и запоминаются в поле оперативной памяти, начинающемся с ячейки A_2 .

СЛОЖИТЬ - складываются два операнда, один из которых хранится в ячейке с адресом A_1 , второй - в ячейке с адресом A_2 .

УМНОЖИТЬ - умножаются два операнда, один из которых хранится в ячейке оперативной памяти с адресом A_1 , второй - в ячейке с адресом A_2 .

ЗАПОМНИТЬ - записывается значение, хранящееся в сумматоре АЛУ (арифметическо-логического устройства), в ячейку с адресом A_1 . Адрес A_2 в данной команде не используется.

НАПЕЧАТАТЬ - из поля оперативной памяти, начинающегося с адреса A_2 , на печать выводятся значения хранимых в памяти N чисел.

ОСТАНОВИТЬ - прекращается выполнение программы. Значение адресов A_1, A_2 не используются.

Будем считать, что в перечисленных командах адреса A_1 и A_2 могут изменяться в пределах 00 - 99, а адресуемая ячейка оперативной памяти может хранить значение одной переменной или команду программы.

Составление программы. Сначала распределяется память. В рассматриваемом примере вычисляемое выражение содержит четыре

переменных С, В, Х, Y, для хранения значений которых отводится четыре ячейки памяти с номерами 01, 02, 03, 04 соответственно.

Запишем команды программы в соответствии с приведенным выше алгоритмом:

- 1) 01 03 01 - команда предписывает ввести с клавиатуры ($K = 01$) три значения ($A_1=01$) в поле оперативной памяти, начинающейся с ячейки 01 ($A_2 = 01$). Предполагается, что переменные в соответствии с принятым распределением памяти вводятся в порядке: С, В, Х;
- 2) 03 03 03 - команда предписывает умножить ($K = 03$) число, хранящееся в ячейке оперативной памяти с адресом 03 ($A_1=03$), на число, хранящееся в ячейке оперативной памяти с адресом 03 ($A_2 = 03$). Результат, равный значению X^2 , сохраняется в АЛУ в сумматоре;
- 3) 04 04 00 - команда предписывает запомнить ($K=04$) в ячейке оперативной памяти с адресом 04 ($A_1=04$) значение результата предыдущей операции. В результате переменная Y получает текущее значение, равное X^2 ;
- 4) 03 01 04 - команда предписывает умножить ($K=03$) два числа, хранящиеся в ячейках оперативной памяти с адресами 01 и 04 ($A_1 = 01, A_2 = 04$). В результате находится произведение $Y * C$ или $(X^2) * C$, так как $Y = X^2$;
- 5) 04 04 00 - команда предписывает запомнить ($K=04$) в ячейке оперативной памяти с адресом 04 ($A_1=04$) значение результата предыдущей операции. В результате в ячейку 04, хранящую текущее значение переменной Y, будет записано значение, равное $(X^2) * C$;
- 6) 02 02 04 - команда предписывает сложить ($K=02$) два числа, хранящиеся в ячейках оперативной памяти с адресами 02 и 04. В результате вычисляется значение, равное $(X^2) * C + B$;
- 7) 04 04 00 - команда предписывает запомнить ($K=04$) в ячейке 04 ($A_1=04$) значение результата предыдущей операции. В итоге в ячейке оперативной памяти с адресом 04 будет храниться искомое значение переменной Y, равное $(X^2) * C + B$;
- 8) 05 01 04 - команда предписывает вывести на печать ($K=05$) одно число ($A_1 = 01$), хранящееся в ячейке оперативной памяти с адресом 04. В результате на печать будет выведено значение переменной Y;
- 06 00 00 - команда предписывает прекратить выполнение программы ($K=06$).

Составленная программа включает в себя девять команд. Для хранения программы необходимо выделить место в оперативной памяти.

На практике часто встречаются случаи, когда вычислительный процесс разветвляется и дальнейший ход вычислений зависит от результатов проверки соблюдения каких-либо условий. Поэтому в состав ЭВМ включают специальные команды, которые обычно относят к группе **команд управления**

вычислительным процессом . Эти команды позволяют нарушить естественный порядок выполнения программы и указывают адрес команды, к которой необходимо перейти при соблюдении определенного условия. В качестве такого условия чаще всего используется сравнение результата предыдущей операции с нулем. Так, например, может выполняться двухадресная команда “иди по условию”:

$K A_1 A_2$.

Здесь, как и выше, K - код операции (пусть для этой операции он равен 07), A_1 - адрес команды программы, к которой следует перейти, если результат предыдущей операции меньше 0; A_2 - адрес команды, к которой следует перейти, если результат предыдущей операции положительный.

В случае, если результат предыдущей операции равен нулю, сохраняется естественный порядок выполнения программы.

Рассмотренная команда является примером команды *условного перехода*. Для организации вычислительных процессов используются также команды *безусловного перехода*. В результате действия такой команды, вне зависимости от каких-либо условий, прерывается естественный порядок выполнения программы и *управление передается любой другой команде*.

Команда безусловного перехода в принятой системе обозначений может быть записана обычным образом

$K A_1 A_2$,

где K - код операции безусловного перехода (например, 08);

A_1 - адрес команды, которой передается управление.

Адрес A_2 в этой команде не используется.

3.2. Языки программирования

Рассмотренный пример составления программы дает представление о *программировании в машинных кодах*, характерном для начального этапа развития вычислительной техники. В настоящее время подобные программы составляются крайне редко и лишь для узкоспециализированных управляющих ЭВМ.

С начала развития вычислительной техники стала очевидной возможность автоматизации процесса получения программ. Основная идея *автоматизации программирования* состояла в разработке специальных языков, использование которых для написания программ было бы более удобным для человека, чем программирование в машинных кодах. Но в отличие от естественных языков (русского, английского и т.д.), допускающих возможность неоднозначного толкования составленных на них предложений, язык программирования должен быть строгим, позволяющим автоматически однозначно преобразовать написанную на нем программу в программу на языке машины, т.е. в машинные коды.

Первым шагом в этом направлении явилось создание так называемых **автокодов**. Автокоды являются *машинно-ориентированными языками* программирования. Это означает, что программы, написанные на автокоде, могут выполняться только на тех ЭВМ, для которых разработан соответствующий автокод. Состав и структура команд автокода полностью соответствуют составу и структуре команд ЭВМ. Для записи кодов операций в автокодах используются их

мнемонические обозначения, облегчающие программисту запоминание команд.

Например, для использовавшихся в предыдущем примере команд можно ввести следующие мнемонические обозначения:

ВВКЛ - ввести с клавиатуры,

СЛ - сложить,

УМ - умножить,

ЗР - запомнить результат,

ВЫПЧ - вывести на печать,

ОСТ - остановить.

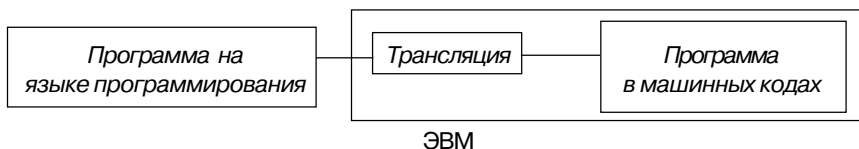
При программировании в автокоде упрощается процесс распределения памяти. В адресной части команд вместо конкретных значений можно записывать наименование переменной решаемой задачи. Ниже для сравнения приведены тексты программы вычисления выражения $Y = C \cdot X^2 + B$ в машинных кодах (слева) и на соответствующем автокоде (справа):

01 01 01	ВВКЛ	1, С
01 01 02	ВВКЛ	1, В
01 01 03	ВВКЛ	1, X
03 03 03	УМХ,Х	
04 04 00	ЗР	Y
03 01 04	УМ	С,Y
04 04 00	ЗР	Y
02 02 04	СЛ	В,Y
04 04 00	ЗР	Y
05 01 04	ВЫПЧ	Y
06 00 00	ОСТ	

В приведенном примере для ввода переменных С, В, X использованы три отдельные команды.

Рассмотренный пример позволяет убедиться в том, что программа, написанная на автокоде, более наглядна, легко читается и полностью соответствует текстовому описанию алгоритма решения задачи.

Перевод программы из автокода в машинные коды осуществляется автоматически специальной программой-**транслятором**, для которой команды программы на автокоде являются исходными данными, а программа в машинных



кодах - результатом работы.

Упрощенно процесс трансляции может быть описан следующим алгоритмом:

1. Составить список переменных, используемых в адресной части программы на автокоде. В результате получим список: С, В, X, Y.

2. Поставить в соответствие каждой переменной из списка значения адресов оперативной памяти, начиная с первой свободной ячейки (например, с ячейки 01).

Получим: С ® 01; В ® 02; X ® 03; Y ® 04.

3. Переписать команды программы на автокоде, заменяя мнемонические наименования кодов операций их числовыми эквивалентами (для этого программа-транслятор обращается к специальной таблице, устанавливающей соответствие между мнемоническими обозначениями кодов и их числовыми эквивалентами). Вместо переменных в адресной части команд записываются поставленные им в соответствие (шаг 2 алгоритма) значения адресов ячеек оперативной памяти.

В результате первая команда запишется в виде

01 01 01, ...;

шестая – в виде

03 01 04

и т.д.

Применительно к процедурно-ориентированным языкам трансляторы называют также **компиляторами**, если процессы трансляции и выполнения программы в ЭВМ разделены во времени, или **интерпретаторами**, если эти два процесса проходят одновременно.

Программа, подающаяся на вход транслятора, называется *исходной*, а результат трансляции – *объектной программой*.

Для того чтобы понять, насколько упростился процесс программирования с появлением автокодов, надо принять во внимание, что программы даже первых ЭВМ включали до нескольких тысяч команд, а современные ЭВМ работают под управлением программных комплексов, число команд которых измеряется десятками и сотнями тысяч. Поэтому вручную распределять память ЭВМ, отыскивать ошибки для программ такого размера чрезвычайно трудоемко.

Автокоды современных ЭВМ образуют группу языков программирования, известных под общим названием *ассемблеры*. **Ассемблеры** позволяют составлять эффективные программы, реализующие все возможности, предусмотренные системой команд и конструкцией ЭВМ. Однако программирование на ассемблерах остается весьма трудоемким и к использованию ассемблеров прибегают лишь в тех случаях, когда нельзя использовать другие языки программирования. К недостаткам ассемблеров можно отнести их *машинную ориентированность*, высокие требования, предъявляемые к уровню подготовки программистов, от которых требуется хорошее знание конструктивных особенностей используемых ЭВМ. Поэтому дальнейшее развитие языковых средств шло по пути создания **машинно-независимых языков**, позволяющих выполнять написанные на них программы на любых ЭВМ. При разработке языков программирования ставилась цель в возможно большей степени приблизить их к профессиональному языку пользователей ЭВМ различных областей науки и техники, не являющихся специалистами в области программирования. За короткое время появилось большое число так называемых **процедурно-ориентированных языков программирования**, предназначенных для решения инженерных и научно-технических задач, задач обработки экономической информации, обработки списков, моделирования и т.д. Среди первых процедурно-ориентированных языков, получивших наибольшее распространение, выделяются ФОРТРАН, АЛГОЛ, КОБОЛ.

Язык программирования **ФОРТРАН**, разработанный в 1956 г. до настоящего времени широко используется для решения научных и инженерно-технических

задач. За это время на языке ФОРТРАН накоплено очень большое число программ, относящихся к различным областям науки и техники. Совокупность таких программ называют **библиотекой программ**. Язык ФОРТРАН очень прост в освоении. Реализации языка для ЭВМ различных типов отличаются незначительно, что в большинстве случаев позволяет переносить программы, написанные для одной ЭВМ, на ЭВМ других типов практически без каких бы то ни было переделок. Все эти качества обеспечили языку ФОРТРАН высокую популярность среди программистов и сделали его наиболее широко используемым языком программирования в мире.

Практика разработки программ на различных языках программирования показала, что производительность труда программистов измеряется числом строк программы, написанной и отлаженной программистом в единицу времени (в день, в неделю,...). Показатель этот практически не зависит от используемого языка программирования. Поэтому утверждают, что программирование на языке ФОРТРАН приблизительно в 10 раз эффективнее, чем на Ассемблере, так как каждое предложение (оператор) программы на языке ФОРТРАН эквивалентно в среднем десяти командам на автокоде (ассемблере). Поэтому такие языки как ФОРТРАН, АЛГОЛ, КОБОЛ и другие в отличие от ассемблеров относят к **языкам высокого уровня**, имея в виду, что уровень языка определяется средним отношением числа операторов в программе к числу команд программы в машинных кодах, получаемой после трансляции. Чем выше это отношение, тем выше уровень языка, а следовательно, и обеспечиваемый им уровень автоматизации программирования.

Язык **АЛГОЛ-60**, разработанный для решения сложных вычислительных задач, был одним из первых языков, унифицированных на международном уровне. Это позволило использовать его не только как эффективный язык программирования, но и как язык для публикации алгоритмов.

В настоящее время используются и многие другие языки, сочетающие большую универсальность с простотой освоения и использования: ПАСКАЛЬ, СИ, БЕЙСИК и другие. Толчком к появлению многих из этих языков явилось массовое внедрение во все сферы жизни и деятельности человека персональных ЭВМ.

Язык программирования **СИ** первоначально разработан в 70-х годах. В настоящее время в СИ сочетаются достоинства современных высокоуровневых языков и возможность доступа к аппаратным средствам машины на уровне, который обычно ассоциируется с языком Ассемблера. СИ имеет синтаксис, обеспечивающий чрезвычайную краткость программ, а компиляторы, вследствие особенностей языка, способны генерировать быстрые и эффективные программы на машинном коде.

Язык программирования **APL** создан в 1969 году. К числу его основных преимуществ относятся богатый набор мощных операторов, позволяющих работать с многомерными массивами как с единым целым, а также предоставление пользователю возможности определять собственные операторы. Основное его назначение - обработка массивов.

FORTH - гибкий и достаточно простой язык, разработанный в 1971 году. Важная его особенность - открытость (расширяемость). Программист может добавлять новые операции, типы данных и операторы. Последнее достигается путем связывания любой строки программы с заданным программистом словом,

которое затем может использоваться наравне со стандартными операторами. Однако расширение языка ведет к снижению эффективности.

Одним из наиболее популярных языков программирования является язык **ПАСКАЛЬ**. Первая версия языка программирования Паскаль была разработана в 1968 году. Прошло много времени с момента появления Паскаля на рынке программных продуктов, прежде чем он получил всеобщее признание вследствие разработки языка программирования **ТУРБО ПАСКАЛЬ (ТП)** - диалекта языка, созданного американской фирмой Борланд. Эта фирма объединила очень быстрый компилятор с редактором текста и добавила к стандартному Паскалю мощное расширение, что способствовало успеху первой версии этого языка. С тех пор Турбо Паскаль значительно расширился. Появились новые графические процедуры, возможность использования при написании программ языка программирования низкого уровня – Ассемблера, возможность создавать объектно-ориентированные программы и многое другое. В лингвистической концепции Паскаля пропагандируется системный подход, выражающийся, в частности, в расчленении крупных проблем на меньшие по сложности и размеру задачи, легче поддающиеся решению. Набор операторов стандартного Паскаля относительно мал и легко изучаем. Но это порождает проблему расширения языка в приложениях. В Турбо Паскале эта проблема решается за счет поставок большого количества библиотек разнообразных процедур, готовых к употреблению в прикладных программах.

Наиболее известным и используемым языком является язык БЕЙСИК, с освоения которого большинство непрофессиональных пользователей ЭВМ начинает свое знакомство с информатикой и вычислительной техникой.

4. ЯЗЫК ПРОГРАММИРОВАНИЯ BASIC

4.1. BASIC - немного истории

БЕЙСИК представляет собой простой язык программирования, разработанный в 1964 году для использования новичками. Работа в среде Бейсика первоначально велась только в режиме интерактивной (диалоговой) интерпретации. В настоящее время имеются и компиляторы с этого языка. В этом языке широко используются разного рода умолчания, что считается плохим тоном в большинстве современных языков. Несмотря на это, Бейсик очень популярен, в особенности на ПЭВМ. Существует множество его диалектов. Бейсик является одним из наиболее динамичных языков. Не без оснований этот язык иногда сравнивают с питоном, заглатывающим и переваривающим все новое, что появляется в других языках программирования. Уровень Бейсика нельзя определить однозначно. Современные диалекты весьма развиты и мало чем напоминают своего предка. По словам президента фирмы Microsoft Билла Гейтса, BASIC переживает все другие языки программирования.

Сегодня BASIC - постоянно развивающийся язык, снабженный всем необходимым для профессиональной разработки программ: поддержка модульного и структурного программирования, эффективная среда разработки, создание исполняемых модулей, мощные средства отладки, дополнительные библиотеки процедур различного назначения.

При этом BASIC сохранил свою привлекательность для начинающих и непрофессиональных программистов прежде всего за счет более простой технологии работы в среде интерпретатора и наличия большого числа операторов высокого уровня (графика, обработка строковых переменных, работа с аппаратурой и прочее).

Первым поколением языка BASIC можно считать GW BASIC, который входил в состав MS - DOS до версии 4.01 включительно. Сегодня GW BASIC - это уже давно морально устаревшая система для персональных компьютеров. В нем отсутствует современная среда разработки. Редактирование текста осуществляется с помощью нумерации строк, а работа только в среде интерпретатора, что крайне замедляло выполнение готовой программы. GW BASIC использовал только 64 Кб оперативной памяти.

Создание QuickBASIC (сокращенное обозначение - QB) в середине 80-х годов произвело настоящую революцию в мире BASIC, результатом которой было то, что впервые этот язык занял достаточно прочные позиции среди средств разработки серьезных прикладных систем. В QuickBASIC в достаточно полной мере реализованы идеи структурного и модульного программирования, возможности использования процедур и функций. Создан механизм применения двоичных библиотек, а следовательно, и готовых программных модулей, в том числе, написанных на других языках.

Специфика технологии программирования в среде QB определяется наличием в ней двух трансляторов - интерпретатора и компилятора. Используется новый вид интерпретатора - интерпретатор компилирующего типа (ИКТ), который производит предварительные “компиляцию и компоновку” программы в специальный псевдокод, а затем уже ее выполнение. При завершении отладки программы пользователь может создать исполняемый EXE-модуль с помощью настоящего компилятора и компоновщика программ.

Начиная с версии MS-DOS 5.0, вместо устаревшего GW BASIC фирма Microsoft стала поставлять систему QBASIC, которая представляет собой усеченный вариант QuickBASIC без компилятора и некоторых возможностей модульного программирования. QBASIC может использоваться для обучения и написания небольших программ, но не для серьезной работы. В версии QBASIC программа может состоять только из одного модуля, нельзя загружать и использовать ранее созданные модули.

С 1989 года фирмой Microsoft ведется разработка новой системы для профессиональной разработки программ - Microsoft Basic Professional Development System. BASIC PDS позволяет создавать более мощные программные комплексы. В нем расширяется круг решаемых задач за счет использования дополнительных возможностей процессора и оперативной памяти. Имеется встроенная система управления большими базами данных.

Следующим поколением языка является Visual Basic. VB является реализацией идеи событийно-управляемого и визуального программирования в среде Windows. Суть визуального программирования отражена в рекламе - “Теперь Вы можете создать программу, не написав ни строки кода” (используя термины - “нарисовать”, “передвинуть” и пр.). Новые идеи программирования, заложенные в VB, касаются в основном проблемы диалогового интерфейса. Что касается самого языка, VB является развитием QuickBASIC.

BASIC часто используется в качестве первого языка программирования. Это объясняется простотой его синтаксиса, ориентацией на диалоговый характер программирования.

Дальнейшее описание языка ориентировано на язык QBASIC.

4.2. Алфавит языка

Набор символов языка QBASIC:

- все прописные и строчные буквы латинского алфавита;
- цифры от 0 до 9;
- математические знаки +, -, *, /, =, <, >, ^ (возведение в степень);
- разделители , . : ; () ' (апостроф) _ (подчеркивание);
- символы для объявления типа данных %, &, !, #, \$;
- русские буквы и другие символы используются только в качестве комментариев и символьных констант.

4.3. Переменные и константы

Одна из характерных особенностей современных ЭВМ состоит в возможности получать, накапливать и хранить большие объемы информации, содержащие формализованные сведения об окружающем человека мире. Поступившая в ЭВМ информация может быть обработана с целью извлечения из нее требуемых пользователю сведений в зависимости от его интересов и потребностей практики.

Информация, хранящаяся в памяти ЭВМ и доступная для обработки, состоит из данных. **Данные** в определенной степени являются приближенным отражением действительности, поскольку в них игнорируются некоторые свойства и характеристики реальных объектов, несущественные для решаемой задачи.

Применение ЭВМ для хранения и обработки данных приводит к разделению данных и их смыслового содержания. ЭВМ, как правило, имеют дело с данными как таковыми, а большая часть интерпретирующей информации в явной форме не фиксируется. Ее должен знать пользователь и применять при анализе данных, полученных от ЭВМ.

Каждый из параметров для задания свойства должен быть представлен либо отдельным значением, либо совокупностью значений. Значения могут характеризовать как количественную, так и качественную сторону объекта. Так, параметр масса может задаваться количественно (например, 1275), параметр цвет - качественно (например, красный); одни параметры - масса, длина, ширина, высота, стоимость, цвет - могут относиться к объекту в целом, другие - характеризовать разные свойства деталей, из которых состоит изделие (массу, стоимость, материал и т. п.). Параметр может иметь и более сложную структуру, отражая одновременно и название детали, и ее серийный номер, и массу, и объем, и положение в пространстве (координаты). Это уже агрегированные параметры.

Следует иметь в виду, что при решении задач ЭВМ оперирует не со значениями, а с их обозначениями, которыми являются конфигурации битов, байтов или слов. Чтобы ЭВМ могла при выполнении операций распознавать

принадлежность этих конфигураций к тому или иному типу данных, необходимо при разработке алгоритмов, и особенно программ, прямо указывать эту принадлежность. Достигается это путем явного описания типов используемых данных. Каждый тип данных характеризуется так называемым кардинальным числом - количеством различных значений, принадлежащих типу. Для каждого типа данных должен быть строго определен набор операций, которые можно применять при обработке данных этого типа.

Изначально определенные типы данных называются простыми, причем те из них, которые непосредственно “встроены” в ЭВМ, носят название стандартных типов. Каждый алгоритмический язык программирования предоставляет пользователю набор различных типов элементарных данных, средства их описания и операторы обработки, обеспечивающие выполнение над данными тех или иных действий. Компилятор связывает имя элемента данных с определенным адресом памяти ЭВМ, по которому в процессе выполнения программы хранится значение именованного элемента данных, что освобождает программиста от необходимости знать этот адрес.

В языке программирования, как и в математике, используется понятие величины. Величины могут быть переменными и константами. **Переменная** - это величина, которая может меняться при выполнении программы. **Константы** - это заранее объявленные величины, которые не меняются в процессе выполнения программы.

В QBASIC константы бывают символьные и числовые.

Символьные константы - последовательность алфавитно-числовых символов, заключенных в кавычки. Например, “Москва”, “VECTOR”, “222”, “Иванов Иван Иванович”.

Числовые константы бывают двух видов: вещественные и целые.

Целая константа представляет собой запись числа без десятичной точки в диапазоне от -32768 до +32767. Например, -289 или 1.

Вещественная константа - это число с десятичной точкой. Вещественные константы могут быть представлены с различной точностью: обычной (6 знаков) и двойной (16 знаков).

Вещественные константы записываются в естественной или в экспоненциальной форме. По-другому говорят – с фиксированной точкой или с плавающей точкой.

123.45 или -10000.0 – форма записи с фиксированной точкой.

1.2345E2 или -1.0E4 – форма записи с плавающей точкой.

E - это основание степени 10, далее число, указывающее степень 10.

Примеры записи одного и того же числа:

567.8 0.05678E4 56.78E1 5678.0E-1 56780.0E-2

Целые числа, которые выходят за допустимый диапазон, а также числа, завершающиеся знаком !, независимо от количества цифр рассматриваются как вещественные. Например, -56! или 812345.

Константы с удвоенной точностью имеют те же способы записи, что и константы с обычной точностью, только вместо буквы E для указания порядка употребляется D, а для завершающего знака вместо ! используется знак #.

Для обозначения переменных в программе используются имена, называемые идентификаторами. **Идентификатор переменной** в QBASIC состоит из букв, цифр и точек. Первым символом должна быть буква. Имена переменных могут

содержать произвольное количество символов, однако различаются только первыми 40 символами. Кроме того, имя не должно совпадать с каким-либо из зарезервированных слов языка. Заглавные и прописные буквы не различаются. Примеры идентификаторов:

X, X12, xx, alfa, risunok.1

Также как и константы, переменные могут быть вещественные, целые и строковые.

Целые (INTEGER) - занимают в памяти 2 байта и используются для значений в диапазоне от -32768 до +32768. Вещественные обычной точности (SINGLE) - 4 байта. Используются для значений в диапазоне от -3.402823E+38 до -1.40129E-45 для отрицательных значений и от +1.40129E-45 до +3.402823E+38 для положительных значений. Переменные вещественные двойной точности занимают в памяти 8 байт. Это числа порядка E-324 и E308.

Тип переменной может быть задан либо явно, либо с помощью оператора DEF, либо по умолчанию. При явном задании для указания типа переменной к имени добавляется символ типа: % - для переменных целого типа; ! - для переменных вещественного типа обычной точности; #- для переменных с двойной точностью; \$ - для переменных символьного типа. Например, a% - переменная целого типа; a\$ - переменная символьного типа; a! - вещественного типа с обычной точностью; a# - вещественного типа с удвоенной точностью.

Оператор DEF ... указывает, что тип переменной будет определяться по первой букве имени переменной. Явное задание типа имени переменной оператор DEF... не отменяет. Общий вид оператора:

DEFINT буква или диапазон букв [,буква или диапазон букв]

DEFSNG буква или диапазон букв [,буква или диапазон букв]

DEFDBL буква или диапазон букв [,буква или диапазон букв]

DEFSTR буква или диапазон букв [,буква или диапазон букв]

Буква - любая буква латинского алфавита, диапазон букв - диапазон букв, на которые распространяется действие оператора. Тип переменной определяется следующим образом: DEFINT - целые; DEFSNG - с обычной точностью; DEFDBL - с удвоенной точностью; DEFSTR - строковые. Например, по оператору

DEFINT A, P – S, Z

все переменные, начинающиеся с A, P, Q, R, S, Z, относятся к целочисленным переменным.

По умолчанию переменная относится к вещественному типу с обычной точностью.

4.4. Выражения и операции

Последовательность операций, которые необходимо произвести над данными, чтобы получить требуемое значение, *называется выражением*. Результатом выражения может быть символьная или числовая константа, переменная или значение.

В QBASIC существует пять категорий операций:

1. Арифметические операции.
2. Операции отношений.
3. Логические операции.

4. Функциональные операции.

5. Строковые операции.

Арифметические операции. Арифметические операции в порядке убывания приоритета выполнения:

^ возведение в степень;

+, – присвоение знака числу;

*, / умножение, деление;

\ целочисленное деление;

MOD остаток после целочисленного деления;

+, – сложение, вычитание.

В выражении сначала выполняются операции более высокого приоритета, затем операторы одного уровня слева направо. Например: $A = 3 + 6 / 12 * 3 - 2$. Значение A в этом выражении = 2.5. Скобки нарушают естественный порядок вычисления арифметического выражения. Сначала выполняются вычисления в скобках.

Примеры: a) ab/c

$a * b / c$

b) $\frac{x+2}{c+d}$

$(x + 2) / (c + d)$

c) $3a^2 + \frac{b}{4} + \frac{7}{1-a}$

$3 * a ^ 2 + b / 4 + 7 / (1 - a)$

Функциональные операции. При программировании различных задач часто бывает необходимо вычислить корень квадратный или логарифм числа, синус угла и т. д. Вычисление этих и других величин осуществляется посредством подпрограмм, называемых функциями, которые заранее запрограммированы и встроены в интерпретатор языка. Такие функции называются **встроенными функциями или стандартными функциями**.

Функциональные операции определяют правила работы с функциями. Функции используются в выражениях для осуществления заранее определенных операций. Список математических функций приведен на стр. 87 (Приложение 1).

Например,

$A = \text{SQR}(20.25) + \text{SQR}(37)$.

В QBASIC существуют два вида функций: встроенные и определенные пользователем. Функция задается с помощью указателя функции, который записывается в виде имени и аргументов, заключенных в круглые скобки. В качестве аргументов могут использоваться константы, переменные, функции, арифметические выражения. Например:

$\text{SIN}(0.87)$, $\text{SIN}(X)$, $\text{SIN}(X * 2.53)$, $\text{SIN}(\text{ABS}(X))$

Операции отношения. Операции отношения используются для сравнения арифметических выражений. В QBASIC существует шесть операций отношения, перечисленных в таблице:

= равно;

> больше;

< меньше;

<> не равно;
<= меньше или равно;
>= больше или равно.

Результатом операции отношения является либо “true” (истина - ненулевое значение), либо “false” (ложь - ноль). Управляющие операторы в программе могут использовать результат операции отношения. Если в одном выражении встречаются и арифметические операторы и операторы отношения, то первыми выполняются арифметические операторы. Например: $X + Y < (T - 1) / Z$. Сначала вычисляется выражения $X + Y$, затем $(T - 1) / Z$, затем отношение между вычисленными значениями.

Логические операции. В QBASIC существует шесть логических операций.

NOT - отрицание. NOT A истинно тогда и только тогда, когда A ложно.

AND - логическое умножение. A AND B истинно тогда и только тогда, когда истинно A и истинно B.

OR - логическое сложение. A OR B истинно тогда, когда истинно A или истинно B.

XOR - исключающее или. A XOR B истинно тогда и только тогда, когда значения A и B не совпадают.

EQV - эквивалентность. A EQV B истинно тогда и только тогда, когда A и B одновременно истинны или одновременно ложны.

IMP - импликация. A IMP B принимает значение “ложь”, если A истинно, а B ложно, и значение “истина” в других случаях.

Результат действия логических операций сведем в таблицу.

A	B	NOT	AND	OR	XOR	EQV	IMP
1	1	0	1	1	0	1	1
1	0	0	0	1	1	0	0
0	1	1	0	1	1	0	1
0	0	1	0	0	0	1	1

1 - истина, 0 - ложь.

Порядок выполнения логических операций задается круглыми скобками и приоритетом логических операций. Приоритет операций установлен следующий: сначала выполняются операции отношений, затем логические операции в порядке старшинства: NOT, AND, OR, XOR, EQV, IMP.

Операции отношений и логические операции используются в логических выражениях. Примеры записи логических выражений:

1. $(-4 \leq X) \text{ AND } (X \leq 4)$
2. $X > 0 \text{ AND } Y > 0$
3. $X = 0 \text{ OR } X = 1$

Строковые операции. Над строками можно осуществлять следующие действия - конкатенация (сложение) и сравнение строк.

Для сложения используется символ +. Если $c\$ = a\$ + b\$$, то при $a\$ = \text{“file”}$ и $b\$ = \text{“name”}$ $c\$ = \text{“filename”}$.

Сравнение строк производится при помощи операций сравнения: <>, =, <, >, <=, >=. Сравнение строк производится в соответствии с ASCII кодами каждого

символа в сравниваемых строках. Если ASCII коды равны, то строки считаются равными. Строка, символы которой имеют большие ASCII коды, считается большей. Если одна из строк короче другой, то она считается меньшей (если до этого места строки были равными). При сравнении учитываются начальные и конечные пробелы.

4.5. Первая программа на QBASIC

ЭВМ отводится роль исполнителя приказов человека, записанных по определенным правилам языка программирования. Нарушение этих правил может привести к тому, что ЭВМ не воспримет (или воспримет неправильно) приказы, в результате чего требуемые действия не будут выполнены. Следовательно, до решения задачи человек (программист) должен продумать и правильно записать последовательность приказов, выполнение которых он хочет поручить ЭВМ. Приказ - это побуждение к некоторому действию. В языке программирования приказы принято называть **операторами**. **Программа** - задание для ЭВМ, написанное на понятном для нее языке в установленной форме.

Напишем программу сложения двух чисел.

‘Программа сложения двух чисел

input a

input b

c = a + b

print c

end

В программе на языке QBASIC нет специального оператора начала программы. Записанный первым оператор является началом всей программы. Правилom хорошего тона считается начинать программу с комментария. Программы должны быть понятны не только компьютеру, но и человеку, не знакомому с данной программой. Поэтому желательно в самой программе коротко комментировать ее назначение, алгоритм решения. **Комментарий на BASIC** - это набор любых символов, начинающийся с ‘(апострофа). В самом первом комментарии, как правило, записывается наименование программы и ее основные функции. ЭВМ не читает текст комментария. Начиная от апострофа и до конца строки BASIC не анализирует и пропускает как информацию, к нему не относящуюся. Вместо апострофа может также использоваться ключевое слово REM.

Операторы языка программирования содержат ключевые слова. **Ключевое (служебное) слово** - слово, смысл, написание и способ употребления которого постоянен для языка программирования.

Оператор INPUT используется для ввода информации. При выполнении программы в этом месте ЭВМ приостановит свои действия и будет находиться в состоянии ожидания до тех пор, пока пользователь не введет численные значения переменных, указанных в списке. Необходимо ввести численное значение для a и затем для b.

Следующий оператор $c = a + b$ называется **оператором присваивания**. У него тоже есть ключевое слово. Это слово LET. Только у оператора присваивания ключевое слово можно опускать, так как транслятор идентифицирует его по совокупности идентификатора переменной в левой части и знака присваивания.

Формат оператора присваивания:

[LET] v = a,

где v - имя переменной, принимающей новое значение; a – выражение. При выполнении оператора присваивания вычисляется значение выражения в правой части, затем оно присваивается имени переменной в левой части оператора. Если для a было введено значение 77, а для b – значение 23, то с примет значение 100.

Оператор PRINT предназначен для вывода численного значения переменных, перечисленных в списке, на экран дисплея. Следовательно, на экране дисплея появится число 100.

Оператор END обозначает конец программы.

В QBASICе можно записывать несколько операторов в строке. В этом случае между операторами ставится двоеточие. Приведенную выше программу можно записать в две строки:

‘Программа сложения двух чисел :

input a : input b: c = a + b : print c: end

Следует иметь в виду, что чтение такой программы затруднено. На длину строки в QBASIC наложено ограничение. Максимальное число символов в строке 255.

4.6. Основные операторы языка

Различают выполняемые и невыполняемые операторы. **Невыполняемые операторы** описывают свойства данных, например, размерности массивов (оператор DIM), комментируют текст программы. **Выполняемые операторы** реализуют действия, предусмотренные алгоритмом решения задачи, например, присвоить значение, выдать на печать, передать управление, остановить процесс вычислений. Рассмотрим пока только некоторые операторы языка QBASIC, которые позволяют программировать основные алгоритмические структуры.

4.6.1. Оператор присваивания

Оператор присваивания служит для вычисления значения арифметического или строкового выражения и присваивания этого значения переменной. Формат оператора присваивания:

[LET] v = a,

где v - имя переменной, принимающей новое значение; a - выражение. Переменная и выражение должны быть или оба числовыми, или оба строковыми.

Примеры записи операторов присваивания:

a = 21.223

d = b * b – 4 * a * c

i = i + 1

A\$ = “ Гуманитарный”

A\$ = A\$ + “ Университет”

4.6.2. Операторы ввода данных

Операторы ввода служат для задания исходных данных при выполнении программы. Задать исходные значения можно с помощью оператора присваивания:

a = 22 ;

S\$ = "Внимание!".

Можно использовать оператор DATA/READ:

DATA "Лена", 1976, "Катя", 1977, "Оля", 1980

READ Girl1\$, year1, Girl2\$, year2, Girl3\$, year3

Оператор READ считывает данные, перечисленные в операторе DATA. Данные, подлежащие вводу, могут располагаться в одном операторе DATA или в нескольких операторах DATA, но в том порядке, в котором эти данные должны использоваться оператором (или операторами) READ. Операторы DATA могут располагаться в любом месте программы.

Для ввода данных с клавиатуры служит оператор INPUT. С помощью оператора INPUT можно вводить данные и с других внешних устройств. Формат оператора:

INPUT [;] ["Символьная константа" {;|,}] список переменных,

где ; - точка с запятой сразу после INPUT оставляет курсор на той же линии экрана после нажатия клавиши Enter; "Символьная константа" - это текст-приглашение к вводу данных; {;|,} - точка с запятой или запятая после символьной константы означает присутствие или отсутствие знака вопроса после текста-приглашения; список переменных - идентификаторы переменных через запятую.

При записи формата операторов в *квадратных скобках* записываются те данные, которые могут отсутствовать.

Примеры записи оператора ввода:

INPUT N\$, K

INPUT "Введите 2 числа X, Y"; X, Y

INPUT "Начальное значение", X1

При выполнении оператора INPUT программа ожидает ввода данных. Пользователь с клавиатуры вводит значение каждой переменной, указанной в списке, по порядку, разделяя их запятыми.

INKEY\$ - функция ввода/вывода, читающая символы с клавиатуры. Эта функция возвращает одно- или двухбайтную строку, содержащую символ, считанный с клавиатуры. Если символ не был считан, то возвращается нулевая строка. То есть данная функция не ожидает ввода в отличие от оператора INPUT.

4.6.3. Оператор вывода данных

Для вывода данных на экран дисплея в процессе выполнения программы используется оператор **PRINT**. Формат оператора:

PRINT список вывода

Список вывода может содержать константы, переменные, арифметические выражения. Элементы списка разделяются запятыми или точкой с запятой. Например:

PRINT "Сумма элементов вектора = " ; S

PRINT a, b, SIN(c)

PRINT "Корень уравнения = " ; (X0 - X1) / 2

PRINT "Адрес"; a\$; "Телефон"; T\$

Если переменные разделены точкой с запятой, то числовые значения разделяются одним (если перед числом стоит знак минус) или двумя пробелами. Если переменные разделены запятой, то осуществляется зонный формат вывода.

При зонном формате строка разделяется на 5 зон по 14 позиций в каждой. Каждое выводимое значение размещается в своей зоне. Если последняя зона в строке заполнена, то вывод данных продолжается с первой зоны следующей строки.

Отсутствие запятой или точки с запятой в конце списка вывода вызывает переход на новую строку. Например, при выполнении операторов

```
PRINT x, y,  
PRINT w
```

все три значения выводятся в одной строке и каждое из них в своей зоне.

При выполнении операторов

```
PRINT x, y  
PRINT w
```

выводятся два значения (x и y) в одной строке, а значение w - в следующей. Оператор PRINT без списка аргументов выводит строку пробелов.

В операторе PRINT используется функция TAB (n), где n - целое число в диапазоне от 1 до 255. Функция TAB позиционирует курсор на экране дисплея или печатающую головку принтера в положение n по горизонтали. Оператор PRINT TAB (30) "*" выведет символ * в 31-й позиции текущей строки.

Существует оператор форматного вывода данных PRINT USING, о котором речь будет идти в следующей главе.

Для позиционирования курсора на экране дисплея предназначен **оператор LOCATE** x, y, где x и y - координаты положения курсора по вертикали и по горизонтали. Например, операторы LOCATE 10, 40 : PRINT "HELLO!" позволяют вывести сообщение "HELLO!" в центре экрана. (На экране дисплея 24 строки и 80 столбцов).

4.6.4. Условные операторы

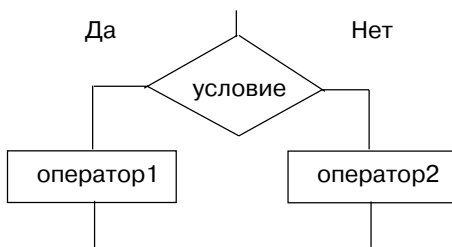
Условный оператор позволяет проверить некоторое условие и в зависимости от результата выполнить то или иное действие. С помощью этого оператора программируются алгоритмы разветвляющейся структуры. Формат оператора:

```
IF <условие> THEN <оператор1> ELSE <оператор2> [ENDIF]
```

где IF, THEN, ELSE, ENDIF - зарезервированные слова (если, то, иначе);

<условие> - произвольное выражение логического типа;

<оператор1>, <оператор2> - любые операторы языка;



ENDIF - конец условного оператора при блочной форме записи.

Условный оператор работает по следующему алгоритму. Вначале вычисляется условное выражение <условие>. Если результат есть TRUE (истина), то выполняется <оператор1>, а <оператор2> пропускается; если результат есть

FALSE (ложь), наоборот, <оператор1> пропускается, а выполняется <оператор2>.

Например:

```
IF printer% = 1 THEN
    PRINT "Принтер готов для работы"
ELSE
    PRINT "У принтера сегодня выходной"
ENDIF
```

Этот условный оператор читается: если условие $\text{printer\%} = 1$ выполняется, то печатается сообщение "Принтер готов к печати", иначе "У принтера сегодня выходной".

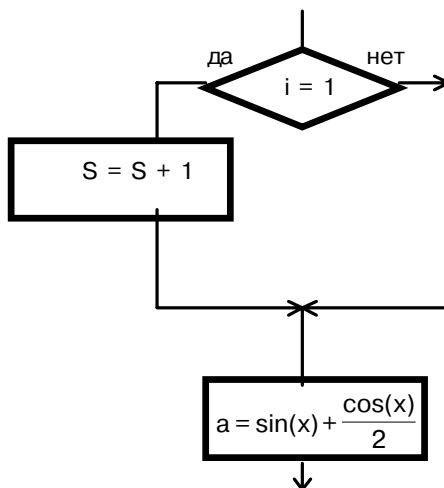
Оператор IF...THEN...ELSE можно записать в блочной или линейной форме. При блочной структуре возможна запись оператора на нескольких строках. При этом важно, чтобы ключевые слова IF, ELSE и ENDIF были первыми операторами в строке, а после ключевого слова THEN не должно быть операторов. При блочной записи оператора каждый IF должен иметь свой ENDIF.

При линейной форме записи условный оператор должен уместиться на одной строке. Например,

```
IF x < 10 THEN y = 1 ELSE y = 2
```

Часть ELSE <оператор2> условного оператора может быть опущена. Тогда при значении TRUE условного выражения выполняется <оператор1>, в противном случае этот оператор пропускается.

```
IF i = 1 THEN s = s + 1
```



$a = \sin(x) + \cos(x) / 2$

4.6.5. Оператор выбора

Для организации структуры множественного выбора служит оператор **SELECT ... END SELECT**. Это управляющий оператор, выполняющий один из нескольких блоков операторов в зависимости от значения выражения.

Формат оператора:

SELECT CASE выражение выбора

[CASE список выражений 1]

```

[операторы цикла 1]
[CASE список выражений 2]
[операторы цикла 2]
.
.
.
[CASE ELSE
[операторы цикла n]]
END SELECT

```

Выражение выбора - любое числовое или символьное выражение; список выражений - одно или более выражений такого же типа, как и выражение выбора; операторы цикла - любое количество операторов. *Ключевое слово CASE должно предшествовать блоку операторов.* Элементы списка выражений должны иметь одну из следующих трех форм:

- выражение [, выражение...];
- выражение TO выражение;
- IS выражение с операцией отношения.

Выражение - любое числовое или символьное выражение того же типа, что и выражение выбора.

```

Пример:
INPUT TestValue
SELECT CASE TestValue
CASE 1, 3, 5, 7, 9
  PRINT "Нечетное"
CASE 2, 4, 6, 8
  PRINT "Четное"
CASE IS < 1
  PRINT "Очень маленькое"
CASE IS > 9
  PRINT "Очень большое"
CASE ELSE
  PRINT "Не целое значение"
END SELECT

```

Если выражение выбора отвечает условиям списка выражений данного блока CASE, выполняются операторы из этого блока. После этого управление передается оператору, следующему за END SELECT.

Можно использовать несколько выражений или пределов в каждом условии CASE. Например:

```

CASE 1 TO 4, 7 TO 9, 11, 13, IS > MaxNumber%

```

Блок операторов CASE ELSE выполняется только в том случае, если выражение выбора не удовлетворяет ни одному из условий CASE. Обычно используется для обработки нежелательных значений.

4.6.6. Операторы цикла

В QBASIC существует три вида циклов:

1. FOR ... NEXT.
2. WHILE ... WEND.
3. DO ... LOOP.

Формат оператора **FOR ... NEXT**:

FOR счетчик = начало TO конец [STEP шаг]

[тело цикла]

[EXIT FOR]

NEXT [счетчик [, счетчик ...]]

Счетчик - числовая переменная, используемая как счетчик цикла; начало - начальное значение счетчика; конец - конечное значение счетчика; шаг - шаг изменения значения счетчика, по умолчанию 1.

Пример: FOR i% = 1 TO 10
PRINT i%, 1/i%
NEXT

Схема для данного примера (рис.16):

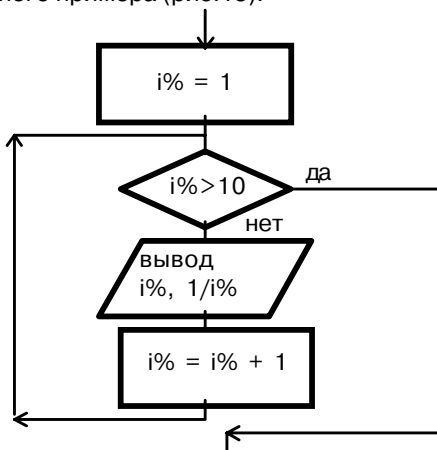


Рис. 16

FOR ... NEXT - управляющий оператор, повторяющий блок операторов (тело цикла) указанное число раз.

Цикл FOR ... NEXT выполняется только в том случае, если начало плюс шаг меньше или равно конечному значению счетчика. Если конец меньше начала, то шаг должен быть отрицательным. Цикл выполняется до тех пор, пока текущее значение счетчика не выйдет за рамки его конечного значения. При завершении текущего цикла к значению счетчика прибавляется значение шага. Если начало и конец имеют одно и то же значение, цикл выполняется один раз, вне зависимости от значения шага. Если шаг равен нулю, цикл продолжается неопределенное время.

Формат оператора **WHILE ... WEND**:

WHILE условие

[операторы]

WEND

Условие - логическое выражение; операторы - любое количество операторов.

WHILE ... WEND - управляющий оператор, выполняющий блок операторов до тех пор, пока указанное условие истинно. Если условие ложно, выполняется оператор, следующий за WEND.

Пример:

```
INPUT R$  
WHILE R$ <> "Д" AND R$ <> "Н"  
  PRINT R$ : INPUT R$  
WEND
```

Схема для примера (рис. 17):

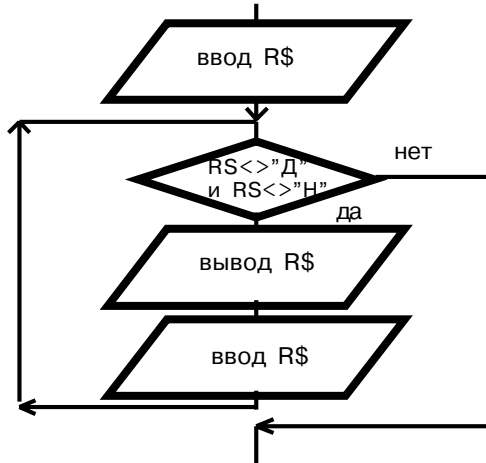


Рис. 17

DO ... LOOP - управляющий оператор, который повторяет блок операторов, пока условие истинно, или до тех пор, когда оно станет ложным. Существует два варианта цикла DO ... LOOP (рис. 18, 19).

Цикл DO ... LOOP с проверкой выражения в начале:

DO WHILE ... LOOP;

DO UNTIL ... LOOP.

Формат оператора DO ... LOOP с проверкой выражения в начале (рис. 18):

DO [{WHILE | UNTIL}]

логическое выражение]

[операторы цикла]

[EXIT DO]

LOOP

Схема DO WHILE ... LOOP

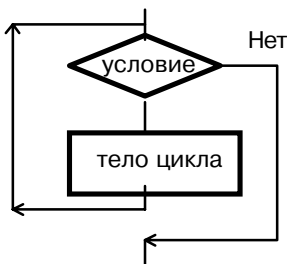


Схема DO UNTIL ... LOOP

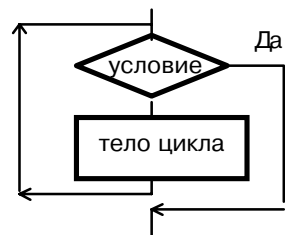


Рис. 18.

Пример:

```
DO UNTIL INKEY$ <>""
LOOP
```

Это пустой цикл, в котором ожидается нажатие любой клавиши.

Цикл DO ... LOOP с проверкой выражения в конце (рис. 19):

```
DO ... LOOP WHILE;
```

```
DO ... LOOP UNTIL.
```

Формат оператора DO ... LOOP с проверкой выражения в конце:

```
DO
```

```
[операторы цикла]
```

```
[EXIT DO]
```

```
LOOP [WHILE | UNTIL логическое выражение]
```

Схема DO ... LOOP WHILE

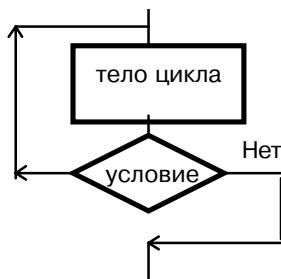


Схема DO ... LOOP UNTIL

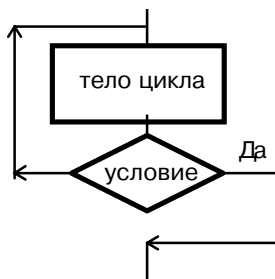


Рис. 19

Пример:

```
DO
INPUT "Число: "; Ch
Total = Total + Ch
LOOP WHILE Ch <> 0
```

Вводимые числа складываются до тех пор, пока не будет введен ноль.

4.6.7. Оператор безусловной передачи управления GOTO

Формат оператора GOTO:

```
GOTO m,
```

где m - метка строки, к которой осуществляется переход.

Например, оператор GOTO 100 передает управление оператору с меткой 100.

Любой оператор программы может быть помечен. **Метка в QBASIC** - это либо идентификатор с двоеточием, либо целое число без знака (как в предыдущих версиях BASIC).

Оператор GOTO должен быть либо единственным оператором строки, либо последним оператором многооператорной строки. Например,

```
s = 0 : x = 0.01 : n = n + 1 : GOTO 120
```

Программу с обилием GOTO трудно читать, отлаживать, модифицировать, так как она не имеет четкой структуры. Использование в программе оператора GOTO свидетельствует о том, что программист не полностью овладел всем богатством управляющих структур языка QBASIC.

4.6.8. Оператор множественного выбора ON GOTO

ON GOTO служит для передачи управления разным строкам программы в зависимости от значения указателя перехода.

Формат оператора:

ON указатель перехода GOTO список меток,
где указатель перехода - любое допустимое арифметическое выражение или переменная.

При выполнении оператора вычисляется прежде всего значение числового выражения. Целая часть его используется в качестве указателя перехода на одну из перечисляемых меток в списке: так, при значении 1 переход осуществляется к строке с первой меткой из списка; при значении 2 переход осуществляется к строке со второй меткой из списка и т. д.

Пример.

```
20 PRINT "Введите номер элемента от 1 до 4";  
  INPUT n  
  ON n GOTO 100, 200, 300, 400  
  GOTO 999  
100 PRINT "Литий" : GOTO 20  
200 PRINT "Водород" : GOTO 20  
300 PRINT "Гелий" : GOTO 20  
400 PRINT "Барий" : GOTO 20  
999 PRINT "Конец" : END
```

При $n = 1$ оператор ON GOTO передаст сообщение строке 100, при этом выводится сообщение: "Литий". Далее управление с помощью оператора GOTO передается строке 20. При отсутствии строки, соответствующей указателю перехода, например $n = 7$, управление передается оператору, следующему за ON GOTO, то есть выполнится GOTO 999.

Оператор ON GOTO также является устаревшей конструкцией. Аналогичный оператор SELECT ... END SELECT обладает явными преимуществами.

Таблицу операторов QBASIC см. в конце файла (Приложение 2).

5. ПРОГРАММИРОВАНИЕ ОСНОВНЫХ АЛГОРИТМИЧЕСКИХ СТРУКТУР НА ЯЗЫКЕ QBASIC

Любая сколь угодно большая программа состоит из набора основных алгоритмических структур. Научившись программировать основные алгоритмические структуры, можно переходить к созданию больших содержательных программ.

5.1. Программы линейной структуры

Программа линейного вычислительного процесса представляет собой последовательность операторов присваивания, ввода и вывода, которые теперь нам известны. Приведем пример программы линейной структуры на QBASIC (рис. 20).

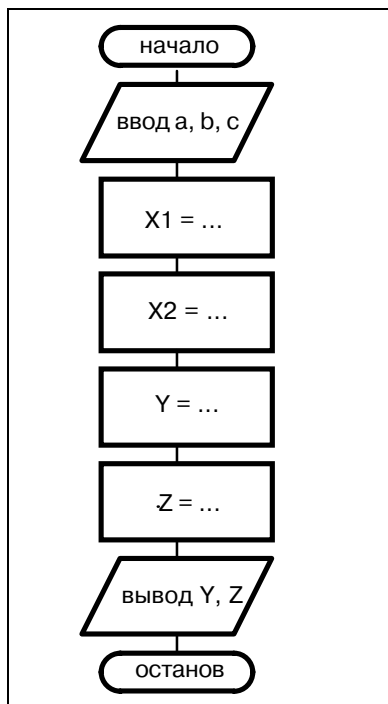
Задача 1. Вычислить значения функций Y и Z .

$$y = (e^{-x_1} + e^{x_2}) / 2; \quad Z = (a\sqrt{x_1} - b\sqrt{x_2}) / c$$

$$X1 = (b + \sqrt{b^2 - 4ac}) / 2a \quad X2 = (b - \sqrt{b^2 - 4ac}) / 2a$$

Схема алгоритма

Программа



REM Задача N1

INPUT "a"; a

INPUT "b"; b

INPUT "c"; c

X1 = (b + SQR(ABS(b^2 - 4 * a * c)))/(2 * a)

X2 = (b - SQR(ABS(b^2 - 4 * a * c)))/(2 * a)

Y = (EXP(-x1) + EXP(x2)) / 2

Z = (a * SQR(x1) - b * SQR(x2)) / c

PRINT "Y = " ; Y ; "Z = " ; Z

END

Рис. 20

Разработка алгоритма. Исходные данные: a, b, c. X1 и X2 должны быть определены раньше, чем Y и Z, так как входят в расчетную формулу для Y и для Z.

После того, как программа написана и для нее выбраны исходные данные, нужно, чтобы эта программа была выполнена. Программа вводится в оперативную память ЭВМ (например, с клавиатуры), дается команда на выполнение. Процессор выполняет программу. Оператор INPUT потребует ввода трех численных значений. Оператор PRINT выведет

Y = . . . Z = . . . ,

где вместо многоточия численные значения переменных Y и Z.

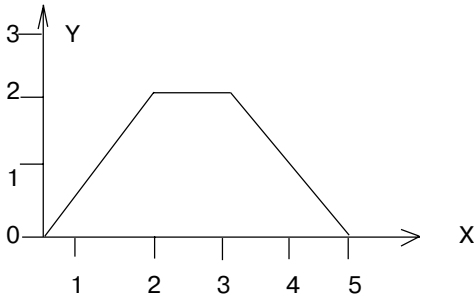
5.2. Программирование алгоритмов разветвляющейся структуры

На практике редко удастся представить схему алгоритма решения задачи в виде линейной структуры. Часто, в зависимости от каких-либо значений промежуточных результатов, необходимо организовать вычисление либо по одним, либо по другим формулам. В QBASIC существует несколько операторов

для программирования разветвляющихся структур. Вот несколько практических примеров.

Задача 2.

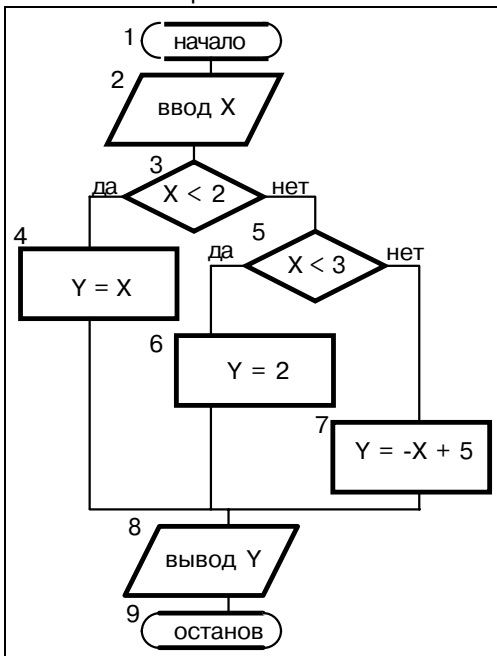
Пусть значение Y зависит от значения X .



Составить программу вычисления значения Y по значению X .

Разработка алгоритма. На графике представлена сложная функция. Зависимость Y от X имеет разный характер на различных участках оси X . При $X < 2$ $Y = X$. В интервале от 2 до 3 $Y = 2$. При $X > 3$ $Y = -X + 5$. При построении схемы алгоритма (рис. 21) достаточно двух логических блоков для организации расчетов по трем ветвям. В программе каждому логическому блоку соответствует условный оператор.

Схема алгоритма



Программа

```

REM Задача N 2
INPUT "X";X
IF X < 2 THEN
    Y = X
ELSE IF X < 3 THEN
    Y = 2
ELSE
    Y = -X + 5
ENDIF
ENDIF
PRINT " Y= " ; Y
END
  
```

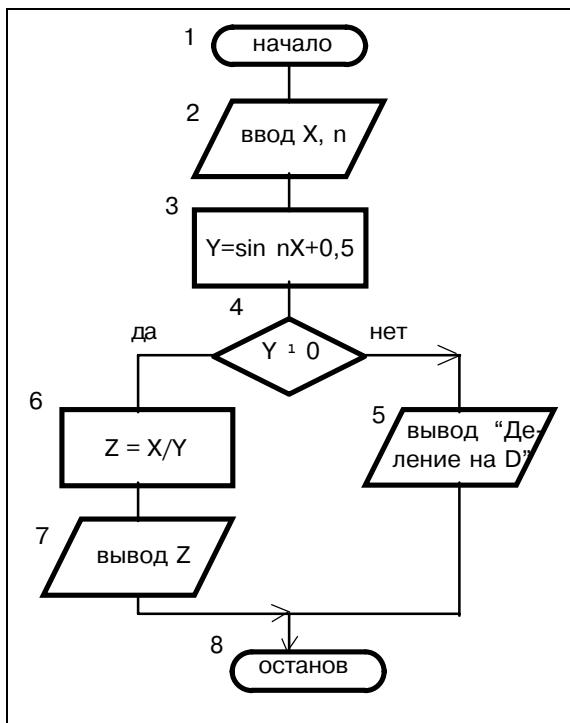
Рис. 21

Если удовлетворено условие $X < 2$, то у получит значение, равное значению X, это значение затем будет выведено. Если условие $X < 2$ не удовлетворяется, то значение Y будет определено выполнением условного оператора

```
IF X < 3 THEN
    Y = 2
ELSE
    Y = - X + 5
ENDIF
```

Задача 3. Вычислить значение функции $Z = X/Y$, где $Y = \sin(nX) + 0.5$.

Разработка алгоритма. Казалось бы, решение этой задачи можно описать алгоритмом линейной структуры. Однако для удовлетворения свойств массовости и результативности алгоритма необходимо, чтобы при любых исходных данных был получен результат или сообщение о том, что задача не может быть решена при заданных исходных данных. Действительно, если $Y = 0$, задача не может быть решена. Поэтому в алгоритме необходимо предусмотреть этот случай и выдать в качестве результата информацию о том, что $Y = 0$. Рассматриваемый вычислительный процесс должен иметь две ветви: в одной, если $Y \neq 0$, необходимо вычислить значение переменной Z, а в другой - дать информацию $Y = 0$ (рис. 22).



Решение задачи N3

```
INPUT "X "; X
INPUT "n "; n
Y = SIN(n * X) + 0.5
IF Y <> 0 THEN
    Z = X / Y
    PRINT "Z = ", Z
ELSE
    PRINT "Деление на ноль"
ENDIF
END
```

Рис. 22

Задача 4. В зависимости от вводимого признака q рассчитать значение функции Z .

$$Z = \begin{cases} 0, & \text{если } q = 0; \\ \sin X, & \text{если } q = 1; \\ e^x, & \text{если } q = 2; \\ \log X, & \text{если } q = 3; \\ x^x & \text{во всех остальных случаях.} \end{cases}$$

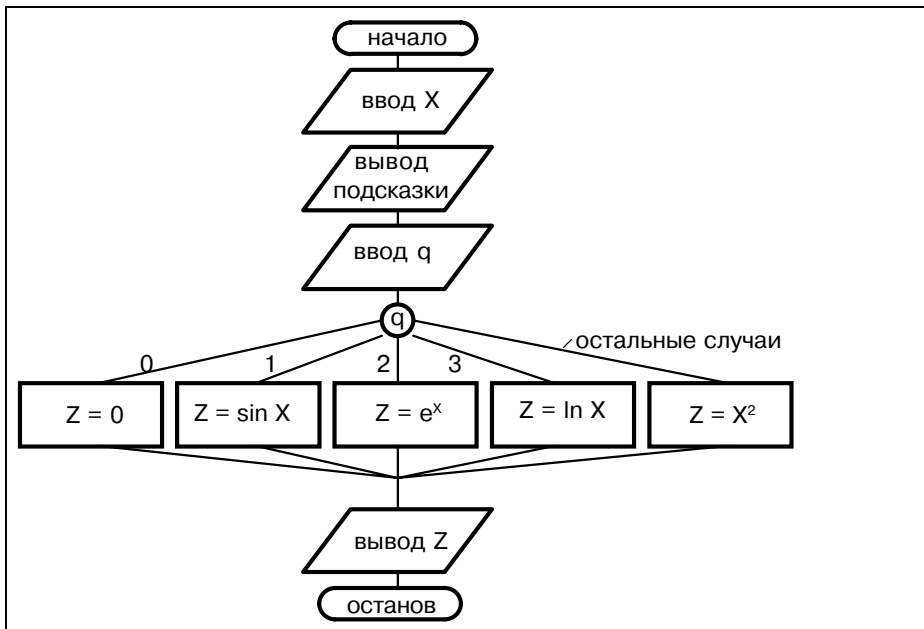


Рис. 23. Схема алгоритма

На рис. 23 приведена структура, которая называется “множественный выбор”. В зависимости от вводимого значения q вычисления осуществляются по одной ветви из пяти.

На QBASIC такая структура может быть запрограммирована с помощью операторов:

- IF ... THEN ... ELSE;
- ON GOTO;
- SELECT ... END SELECT.

Используем оператор SELECT как наиболее структурный.

Программа:

‘ Решение задачи N4

INPUT “X “; X

PRINT “Введите признак расчетной формулы: 0 - Z=0, 1 - Z = SIN(X)”

PRINT “ 2 - Z = EXP(X), 3 - Z = LOG(X), в остальных случаях - Z = X * X”

INPUT “q “; q

```

SELECT CASE q
CASE 0 : Z = 0
CASE 1 : Z = SIN(X)
CASE 2 : Z = EXP(X)
CASE 3 : Z = LOG(X)
CASE ELSE : Z = X * X
END SELECT
PRINT "Z = ";Z
END

```

При использовании оператора ON GOTO программа имела бы следующий вид:

```

' Решение задачи N4 ( Вариант с использованием ON GOTO)
INPUT "X "; X
PRINT "Введите признак расчетной формулы: 0 – Z =0, 1 – Z = SIN(X)"
PRINT " 2 – Z = EXP(X), 3 – Z = LOG(X), в остальных случаях – Z = X * X"
INPUT "q "; q%
ON q% + 1 GOTO 10, 20, 30, 40
Z = X * X : GOTO 50
10 Z = 0 : GOTO 50
20 Z = SIN(X) : GOTO 50
30 Z = EXP(X) : GOTO 50
40 Z = LOG(X)
50 PRINT "Z = "; Z
END

```

При q% = 0 переход осуществляется на оператор с меткой 10, при q% =1 - на оператор с меткой 20 и т. д.

5.3. Программирование алгоритмов циклической структуры

Различают **регулярные** (с известным числом повторений) **циклы** с управляющим параметром, условием окончания повторения которого является достижение параметром цикла своего конечного значения; **циклы с неизвестным числом повторений**, в которых условие повторения или окончания цикла задается по некоторому вычисляемому промежуточному или окончательному результату, например, пока не будет достигнута требуемая точность вычисления. Типичным примером таких циклов являются так называемые **итерационные циклы**.

Цикл FOR ... NEXT - это цикл с заранее заданным количеством повторений. Можно также выйти из цикла, не дожидаясь выполнения всех повторений, воспользовавшись альтернативным выходом из цикла при помощи оператора EXIT FOR. Управление будет передано на оператор, стоящий после NEXT.

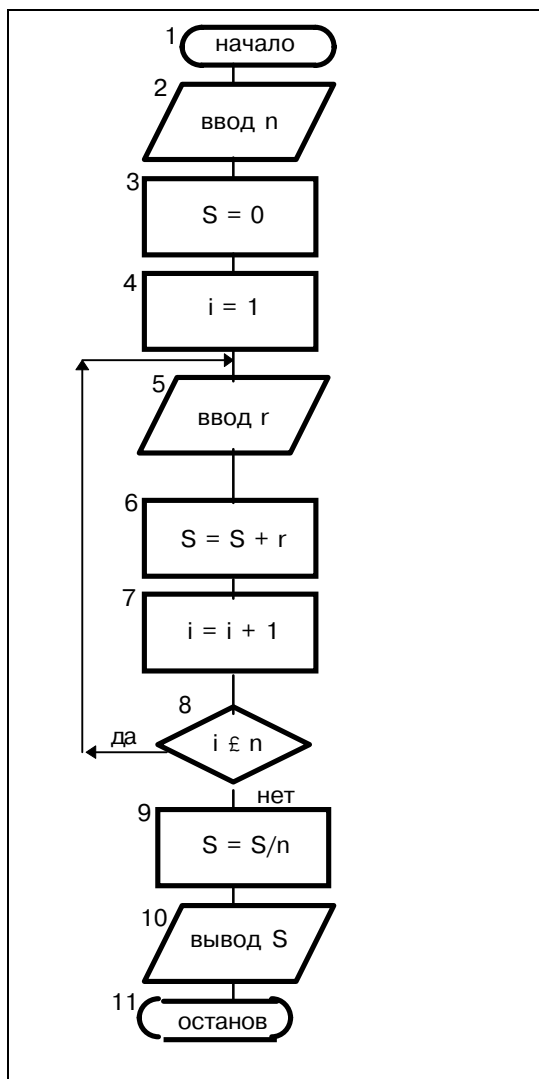
В QBASIC структуру с предусловием реализуют операторы: WHILE ... WEND, DO WHILE ... LOOP, DO UNTIL ... LOOP и FOR ... NEXT. Структуру с постусловием реализуют операторы: DO ... LOOP WHILE и DO ... LOOP UNTIL.

Для организации циклических структур можно использовать операторы IF ... THEN и GOTO.

Задача 5. Определить средний рост студентов в группе.

Разработка алгоритма. Исходной информацией для решения этой задачи являются число студентов в группе и рост каждого студента. В этой задаче мы

встречаемся с распространенной задачей расчета суммы. Сумма получается путем накопления слагаемых в какой-либо переменной. Накопление осуществляется в цикле. Начальному значению суммы присваивается значение ноль. В цикле к текущему значению суммы прибавляется значение очередного слагаемого $S = S + r$.



Число студентов в группе обозначим n . Это исходная величина (рис. 24). В переменной S будем накапливать сумму. Зададим S значение ноль. Подсчет номера студента будем осуществлять в переменной i . Начнем с первого студента $i = 1$. Вводим рост первого студента. К предыдущему значению суммы, т. е. к нулю, прибавим рост первого студента и результат присвоим переменной S . Перейдем к следующему шагу $i = i + 1 = 1 + 1 = 2$. У переменной i теперь значение 2. Выполним проверку выхода из цикла. Если i не превысило еще значения n , то мы возвращаемся к блоку 5 и вводим рост следующего студента. К предыдущему значению суммы, а это рост первого студента, прибавляем рост второго студента и результат записываем в переменную S . В переменной S теперь будет храниться сумма двух значений: рост первого студента и рост второго студента. Далее переходим к следующему шагу. Цикл повторится N раз и в переменной S накопится сумма всех значений ростов студентов. Средний рост определяем по формуле $S = S/n$.

Рис. 24. Схема алгоритма

Запишем программу, используя операторы IF ... THEN и GOTO.

‘ Задача N 5, вариант 1

INPUT “Введите n “;n

s = 0 : i = 1 ‘Начальные присвоения

100 PRINT ”Введите рост “ ; i ; “- го ученика”; : INPUT r

s = s + r : i = i + 1

IF i <= n THEN 100

s = s / n

PRINT “Средний рост учеников в классе - “ ; s

END

Операторы тела цикла

100 PRINT ”Введите рост “ ; i ; “- го ученика”;

INPUT r

s = s + r : i = i + 1

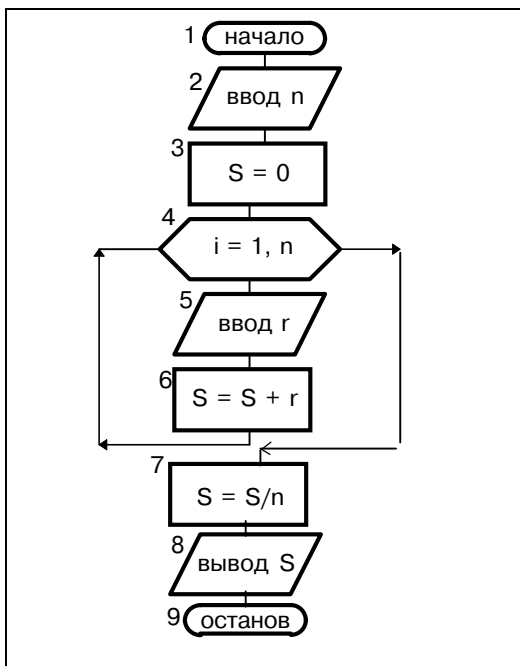
Проверка окончания цикла производится оператором IF ... THEN без ELSE.

Полная запись оператора IF i <= n THEN GOTO 100. В случаях, когда два служебных слова THEN и GOTO стоят рядом, одно из них можно опустить.

Здесь организован цикл с постусловием. Тело цикла выполнится хотя бы один раз, независимо от того, чему равно n.

В этой программе: n - количество студентов в группе; i - номер текущего студента; r - рост текущего студента; s - переменная, в которой накапливается сумма, а затем в эту же переменную записывается средний рост студентов, рассчитанный как среднее арифметическое.

Запишем схему, используя блок модификации (рис. 25).



Еще раз обратите внимание на то, что блок 4 - модификация - включил в себя три блока предыдущей схемы - блоки 4, 7, 8.

Рис. 25

Для реализации этой схемы (рис. 25) на QBASIC больше всего подходит оператор цикла FOR ... NEXT.

‘Задача N 5, вариант 2

INPUT “Введите n “; n

s = 0

FOR i = 1 TO n

PRINT “Введите рост “ ; i ; “- го ученика”; : INPUT r

s = s + r

NEXT i

s = s / n

PRINT “Средний рост учеников в классе - “ ; s

END

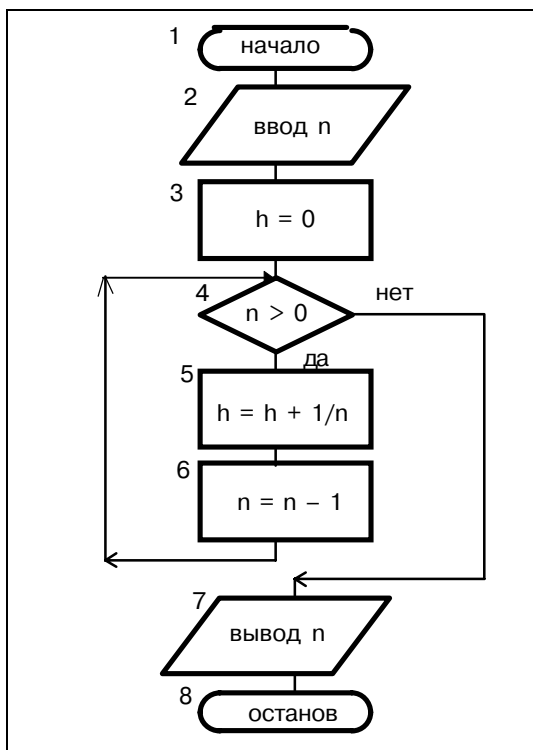
Оператор FOR i = 1 TO n можно было бы записать

FOR i = 1 TO n STEP 1

Напомним, что шаг равный 1 принимается по умолчанию.

Задача 6. Вычислить сумму гармонического ряда $h = 1 + 1/2 + 1/3 + \dots + 1/n$.

Разработка алгоритма. Суммирование начинаем с n - о го члена. Первое слагаемое $1/n$. В цикле n уменьшается на единицу: $n = n - 1$. Суммирование $h = h + 1/n$



производится в цикле до тех пор, пока $n > 0$.

‘Задача N 6 (рис. 26).

INPUT “Введите n “; n

h = 0

WHILE n > 0

h = h + 1/n

n = n - 1

WEND

PRINT “ Сумма = “; h

END

Рис. 26. Схема алгоритма

Напишем программу, используя оператор *FOR ... NEXT*.

Для изображения схемы алгоритма воспользуемся блоком “модификация” (рис. 27).

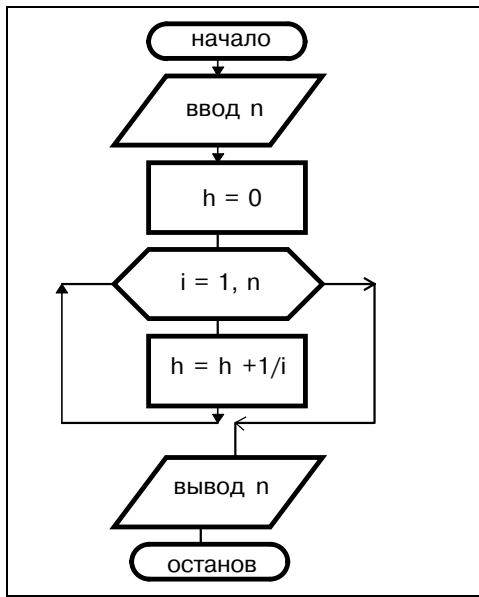


Рис. 27. Схема алгоритма

‘Задача 6, но с использованием оператора цикла *FOR*.

INPUT ” Введите n “; n

h = 0

FOR i = 1 *TO* n

h = h + 1/i

NEXT i

PRINT “Сумма = “ ; h

END

Напишем программу для задачи 6 с использованием оператора цикла с постусловием (рис. 28).

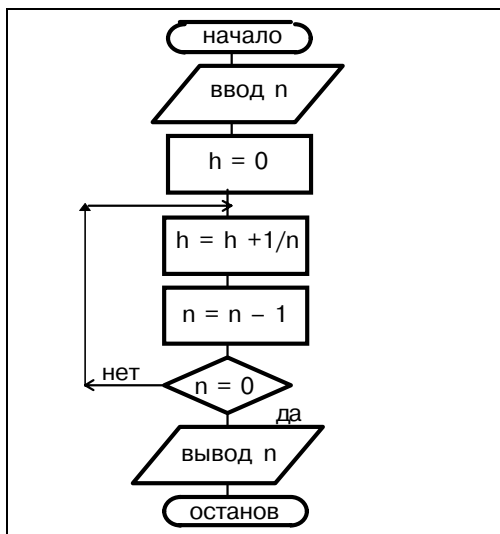


Рис. 28. Схема алгоритма

‘Задача N 6. Цикл с постусловием

INPUT “Введите n ”; n

h = 0

DO

h = h + 1/n : n = n - 1

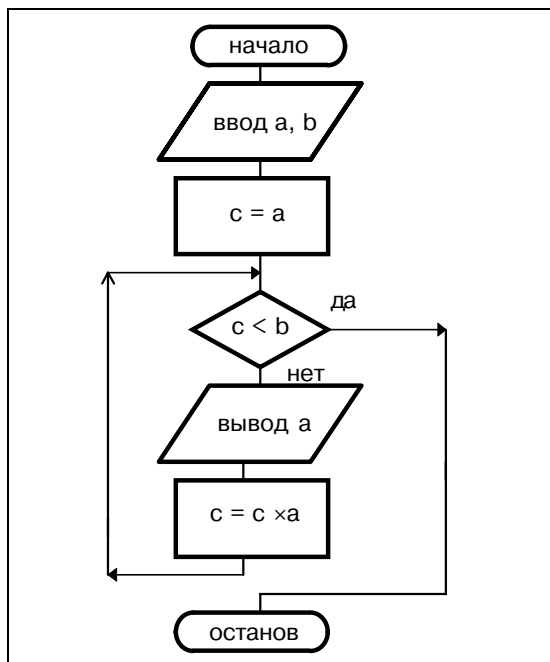
LOOP UNTIL n = 0

PRINT “h = “,h

END

Задача 7. Пусть даны числа a , b ($a > 1$). Получить все члены бесконечной последовательности a, a^2, a^3, \dots меньшие числа b .

Разработка алгоритма. Каждый член последовательности может быть рассчитан из предыдущего по формуле $c = c * a$. Такой расчет будет производиться многократно, следовательно можно организовать цикл. Число повторений цикла заранее неизвестно. В программе используем конструкцию



```

WHILE ... WEND
'Задача N 7 (рис. 29).
INPUT "Введите a "; a
INPUT "Введите b "; b
c = a
WHILE c < b
  PRINT a;
  c = c*a
WEND
END
  
```

Рис. 29. Схема алгоритма

При выполнении этой программы переменная **c** последовательно принимает значения a, a^2, a^3, \dots . Изменение значения **c** происходит до тех пор, пока оно не станет больше или равно значению b . Если $a > b$, то не будет выведено ни одного члена последовательности.

Воспользуемся конструкцией **DO ... LOOP** с принудительным выходом из цикла **EXIT DO**.

'Задача N 7. Вариант 2.

```
INPUT " Введите a "; a : INPUT "b "; b
```

```
c = a
```

```
DO
```

```
  PRINT a; : c = c*a
```

```
  IF c < b THEN EXIT DO
```

```
LOOP
```

```
END
```


Задача 8. Вычислить сумму бесконечного ряда.

$$1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \dots$$

с точностью ϵ .

Разработка алгоритма. Исходными данными являются X и ϵ . Необходимо

подсчитать сумму членов последовательности, удовлетворяющих условию $\frac{X^i}{i!} < \epsilon$.

Каждый член последовательности должен быть получен из предыдущего члена этой последовательности.

$$P = P \cdot \frac{X}{i}; \quad P_1 = \frac{X}{1}; \quad P_2 = \frac{X \cdot X}{1 \cdot 2}; \quad P_3 = \frac{X \cdot X \cdot X}{1 \cdot 2 \cdot 3} \text{ и т.д.}$$

Такая форма записи, в которой каждый следующий элемент расчета может быть получен из предыдущего, называется **рекуррентной формой**.

Для данной задачи $P = P \times X/i$. Для расчета величины текущего члена последовательности необходимо знать его порядковый номер и значение предыдущего члена.

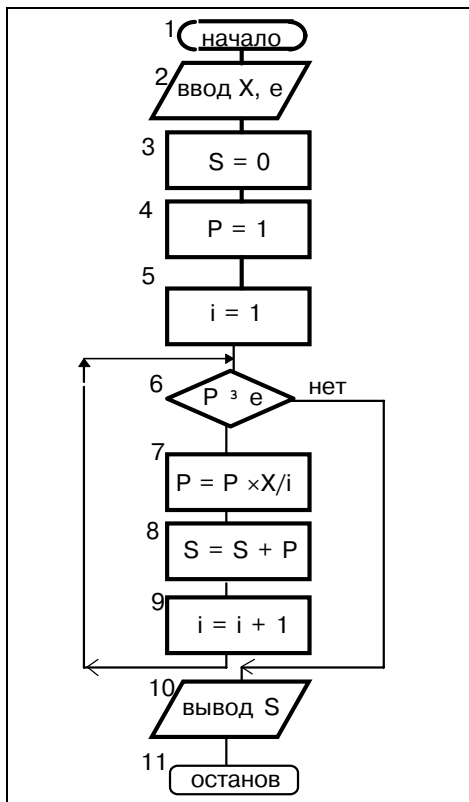


Рис. 30

Блок 3 - обнуляем начальное значение суммы. Блок 4 -

вспомогательное действие. Задание начального значения для P . Блок 5 - задание порядкового номера члена последовательности. Блок 7 - расчет величины текущего члена последовательности. Блок 6 - сравнение текущего члена последовательности с заданной точностью. Если $P \geq \epsilon$, то такой элемент нужно суммировать, если нет, то заданная точность достигнута и расчет можно закончить. Блок 8 - накопление суммы. Блок 9 - переход к следующему текущему номеру и затем к блоку 6.

В данном примере организована циклическая структура типа “Пока” (рис. 30). Повторение последовательности операторов с проверкой условия в начале каждого прохода цикла называется **итерацией**.

Программа:

```
‘Задача N 8
INPUT “Введите X” ; X
INPUT “Введите точность расчета ” ; e
S = 0 : P = 1 : i = 1
DO WHILE p >= e
    P = P * X / i : S = S + P : i = i + 1
LOOP
PRINT “Сумма ряда = ” ; S
END
```

Задача 9. Определить количество студентов в группе, имеющих рост выше среднего.

Разработка алгоритма. Исходные данные: количество студентов и рост каждого студента. Первая половина алгоритма разработана в предыдущей задаче - определен средний рост. Далее рост каждого сравнивается со средним.

Если вводимое значение роста студента превышает среднее значение, то в счетчик числа таких студентов необходимо добавить единицу $K = K + 1$. Счетчик образуется из переменной K, начальное значение которой полагаем равным нулю (блок 8), а суммирование происходит в блоке 12, если условие (блок 11) выполняется. В схеме А (рис. 31) имеется один недостаток: рост студентов в группе вводится дважды. При решении задачи на ЭВМ, придется вводить одну и ту же информацию дважды. Этот недостаток можно устранить введением переменной с индексом. В схеме В (рис. 31) рост студента обозначен переменной r_i , где i - текущий номер студента. r - одномерный массив из n элементов. Блок 10 из алгоритма исключен.

Программа для схемы А.

```
‘Задача N 9
INPUT “n” ; n
S = 0
FOR i = 1 TO n
    PRINT “Введите рост “ ; i ; “- го ученика” ; : INPUT r
    S = S + r
NEXT i
S = S / n
K = 0
FOR i = 1 TO n
    PRINT “Введите рост “ ; i ; “- го ученика” ; : INPUT r
    IF r > S THEN K = K + 1
NEXT i
PRINT “Учеников выше среднего “ ; K
END
```

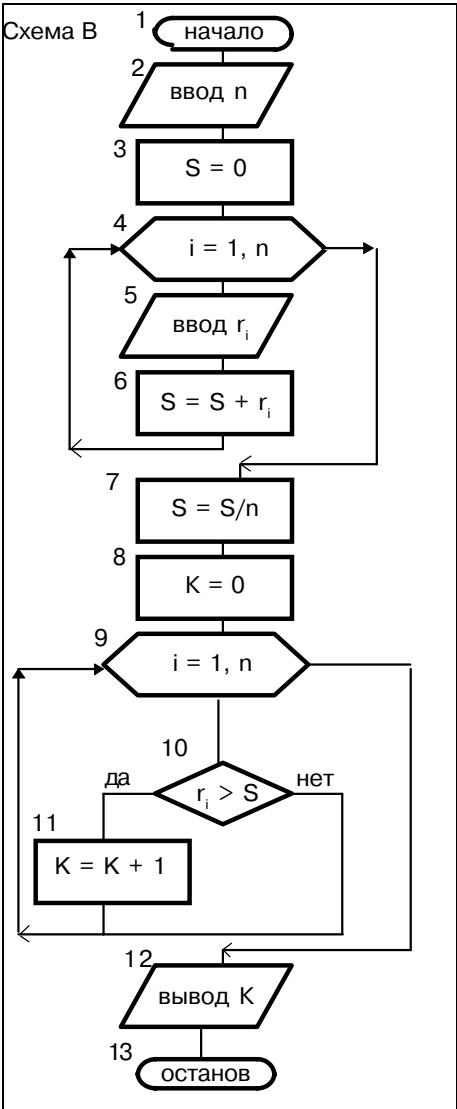
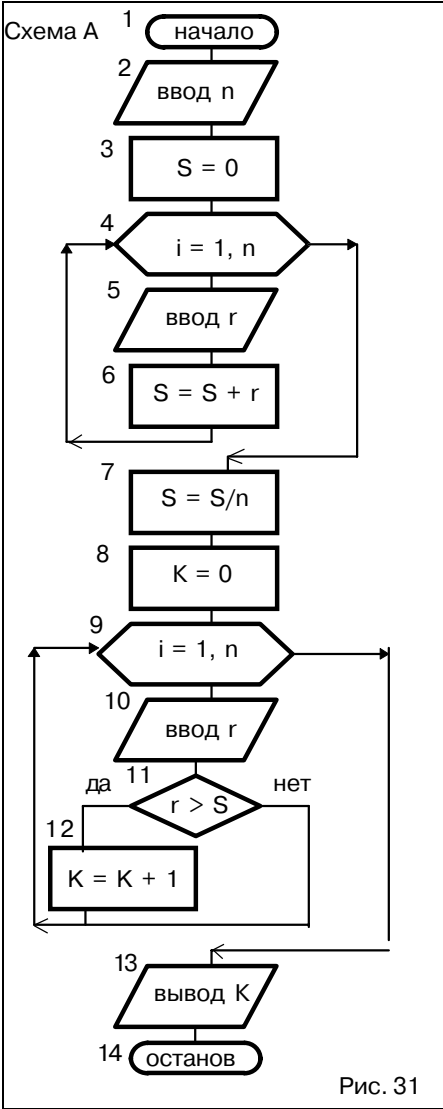
При программировании данной задачи по схеме В воспользуемся одномерным массивом. **Массив** - упорядоченный набор переменных одного типа, для хранения которых отводятся последовательно расположенные поля памяти. Массив характеризуется идентификатором, числом измерений (индексов) и числом переменных в каждом измерении (верхней границей каждого индекса).

Для образования имен массивов используются те же правила, что и для образования имен простых переменных. В одной и той же программе допустимы

одинаковые имена массивов и простых переменных. При работе с массивами в программе необходимо дать команду для резервирования места в памяти ЭВМ. Информация о размерности массива содержится в операторе DIM, имеющем формат:

DIM список массивов,
где список массивов содержит имена массивов и в круглых скобках разделенные запятыми верхние границы каждого из индексов.

Так, оператор DIM A(9) резервирует место в памяти для одномерного массива A, состоящего из 10 элементов - от A(0) до A(9).



Программа по схеме В:

‘Задача N 9 Вариант 2

INPUT “n “; n

DIM r(n) ‘Выделяется n ячеек памяти

S = 0

FOR i = 1 TO n

PRINT “Введите рост “ ; i ; “- го ученика”; : INPUT r(i)

S = S + r(i)

NEXT i

S = S / n

K = 0

FOR i = 1 TO n

IF r(i) > S THEN K = K + 1

NEXT i

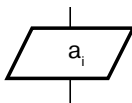
PRINT “Учеников выше среднего “; K

END

В программе используется переменная с индексом $r(i)$. Индекс записывается в круглых скобках и может быть константой или выражением. В данном случае используется имя переменной, что является частным случаем выражения.

Задача 10. Задана последовательность из n чисел. Определить количество положительных, отрицательных и нулевых элементов.

Разработка алгоритма. Исходные данные - одномерный массив из n элементов. Назовем его $[A]$. Элементы массива - $a_1, a_2, a_3, \dots, a_n$. Ввести последовательность - это значит ввести все элементы одномерного массива. В схеме (рис. 32) будет присутствовать элемент



Этот элемент должен повториться для всех i от 1 до n . Следовательно, необходимо организовать цикл.

Число элементов последовательности должно быть задано раньше, так как это конечное значение параметра в цикле ввода элементов. Схема ввода одномерного массива представлена на рис. 32.

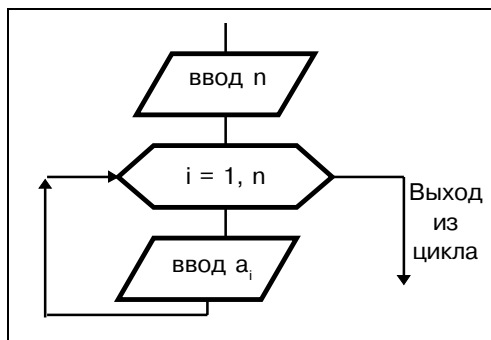


Рис. 32

Если шаг параметра цикла равен единице (+1), то в блоке “модификация” (рис. 33) шаг можно не указывать. Шаг 1 принят принимать по умолчанию.

Чтобы определить количество отрицательных элементов, нужно перебрать все элементы и проверить условие $a_i < 0$. Если условие выполняется, то такой элемент нужно сосчитать, т. е., как в предыдущей задаче, отложить 1 в отведенную для подсчета переменную. K - количество отрицательных элементов, L -

количество положительных элементов, М - количество нулевых элементов. Для подсчета L и М не нужно организовывать новые циклы. Подсчет количества отрицательных, положительных и нулевых элементов организован в одном цикле.

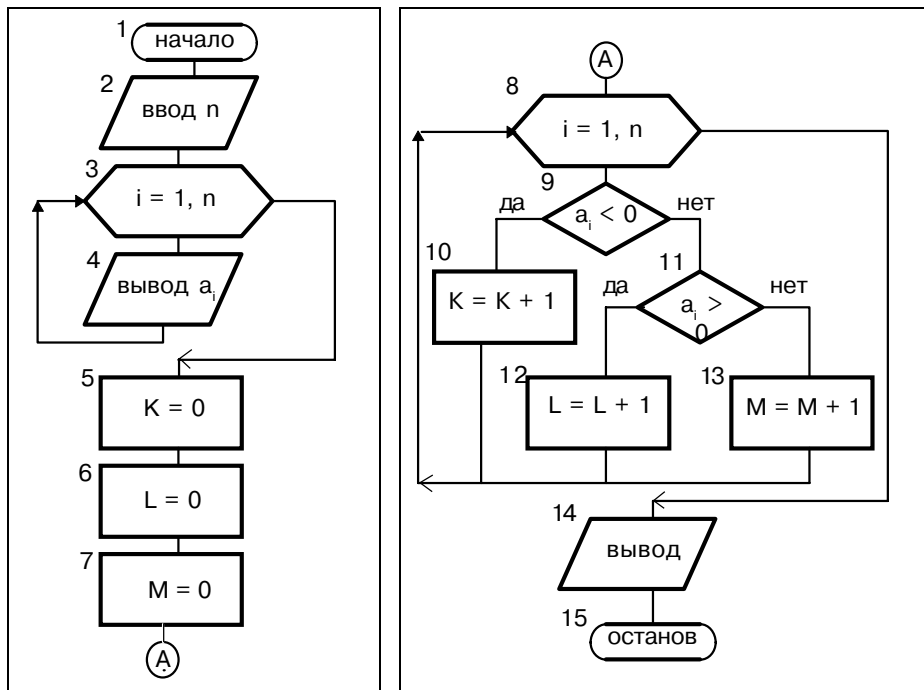


Рис. 33

Ⓐ - соединитель. При большой насыщенности схемы символами отдельные линии потока допускается обрывать (рис. 33). При этом в конце (начале) обрыва должен быть помещен символ Ⓐ.

Программа:

‘Задача N 10

INPUT “Количество элементов в массиве”; n

DIM a(n)

FOR i = 1 TO n

INPUT a(i)

NEXT i

K = 0 : L = 0 : M = 0

FOR i = 1 TO n

IF a(i) < 0 THEN

K = K + 1

ELSE IF a(i) > 0 THEN

L = L + 1

ELSE

M = M + 1

ENDIF

```

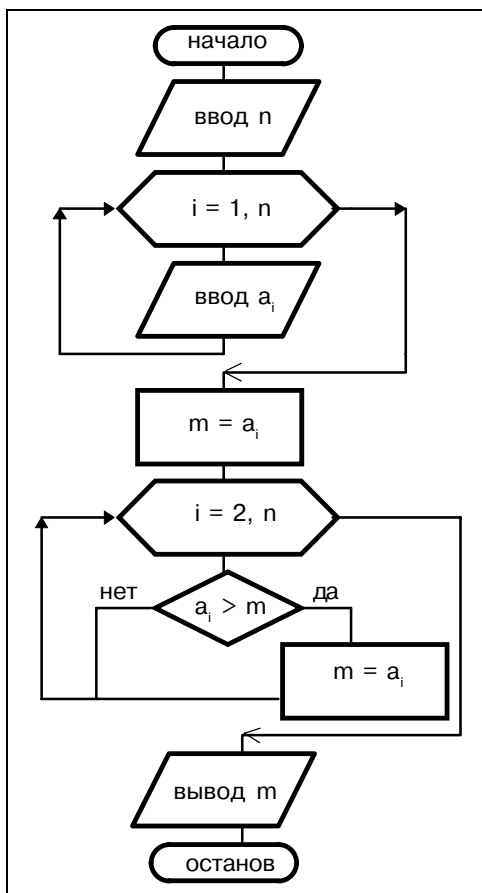
ENDIF
NEXT i
PRINT "Количество отрицательных элементов = "; K
PRINT "Количество положительных элементов = "; L
PRINT "Количество нулевых элементов = "; M
END

```

Задача 11. Найти наибольший элемент последовательности из n элементов.

Разработка алгоритма. Схема ввода последовательности чисел (одномерного массива) разобрана в предыдущей задаче. Для поиска максимального элемента введем вспомогательную переменную m .

Присвоим m значение первого элемента и далее будем перебирать все элементы последовательности от 2 до n , сравнивая a_i и m . Если a_i окажется больше, чем m , то m нужно присвоить значение a_i . В противоположном случае осуществляется переход к следующему элементу последовательности. По окончании цикла m примет значение наибольшего элемента (рис. 34).



```

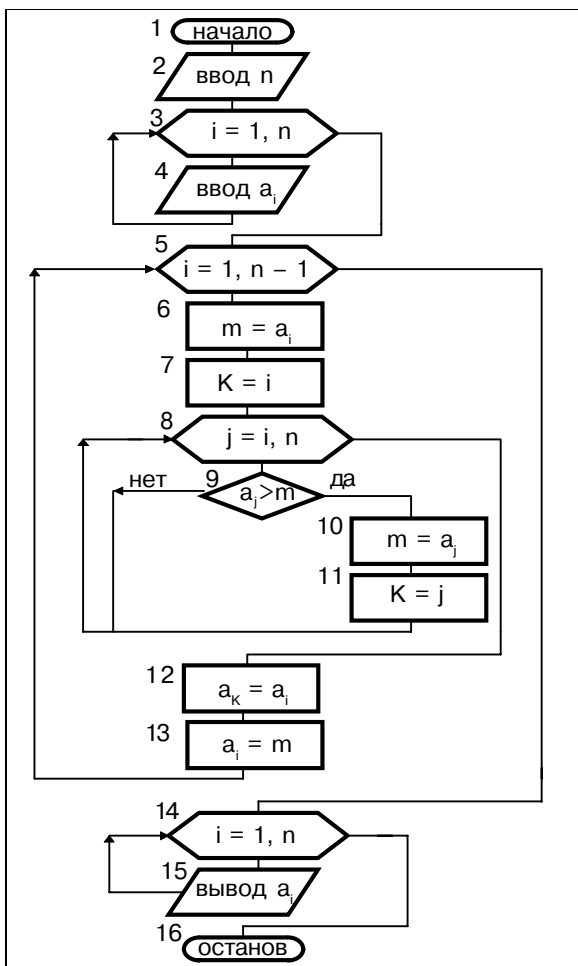
'Задача N 11
INPUT " n "; n
DIM a(n)
FOR i = 1 TO n
INPUT a(i)
NEXT i
m = a(1)
FOR i = 2 TO n
IF a(i) > m THEN m = a(i)
NEXT i
PRINT "Max = "; m
END

```

Рис. 34

Задача 12. Задана последовательность a_1, a_2, \dots, a_n . Расположить элементы в убывающей последовательности.

Разработка алгоритма. Воспользуемся алгоритмом, разработанным в предыдущей задаче. В исходной последовательности найдем наибольший элемент и запомним его порядковый номер K (рис. 35). Пусть это будет a_K . Поменяем местами a_1 с элементом a_K . Наибольший элемент оказался на первом месте. Такую процедуру нужно проделать с оставшимися элементами от 2 до n . В этом случае наибольший элемент a_K нужно поменять местами со вторым элементом. На втором месте окажется второй по величине элемент. Далее ищем наибольший элемент последовательности от 3 до n . Получается, что процедуру поиска наибольшего элемента нужно повторить $n-1$ раз. Последний элемент сам с собой можно не сравнивать. Для запоминания порядкового номера максимального элемента используем переменную K .



Пояснения к схеме.

Блоки 2, 3, 4 - ввод одномерного массива. Блок 5 - организуется цикл для определения порядкового номера элемента, с которого начинается поиск максимального. Блоки 6, 7 - задаются начальные значения для m и K . В m записывается значение наибольшего элемента, в K - порядковый номер этого элемента. Блоки 8, 9, 10 - алгоритм поиска максимального элемента последовательности. Блок 11 - запоминаем порядковый номер элемента, записанного в m . Блок 12 - на место максимального значения элемента записываем i -ый элемент (с i -го элемента начинается поиск максимального). Блоки 14, 15 - вывод последовательности.

Следует обратить внимание на то, что блоки 12 и 13 нельзя поменять местами. Если это сделать, то значение a будет потеряно.

Рис. 35

```

Программа
'Задача N 12
INPUT " n "; n
DIM a(n)
FOR i = 1 TO n
INPUT a(i)
NEXT i
FOR i = 1 TO n - 1
m = a(i) : k = i
FOR j = i TO n
IF a(j) > m THEN m = a(j) : k = j
NEXT j
a(k) = a(i) : a(i) = m
FOR i = 1 TO n
PRINT a(i)
NEXT i
END

```

5.4. Программирование вложенных циклов

Матрицы - удобный пример для демонстрации алгоритмов со структурой вложенных циклов.

В матрице $[A]$ m столбцов и n строк. Каждый элемент матрицы определен двумя индексами, которые указывают положение этого элемента. На первом месте записывается номер строки, на втором - номер столбца. Например, a_{23} - элемент матрицы $[A]$, расположенный во второй строке третьего столбца.

Если необходимо задать значение матрицы, значит нужно задать значения всех ее элементов. Назовем элемент матрицы a_{ij} . Для всей матрицы в целом i пробегает значения от 1 до n , j от 1 до m .

Каждую строку матрицы можно рассматривать как одномерный массив. С вводом одномерного массива мы знакомы. Ввод первой строки матрицы определяет следующую совокупность элементов схемы (рис. 36).

Схема без блока модификации

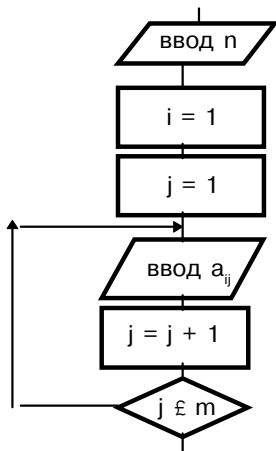


Схема с использованием блока модификации

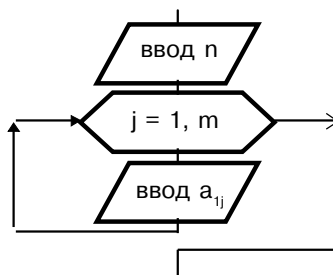
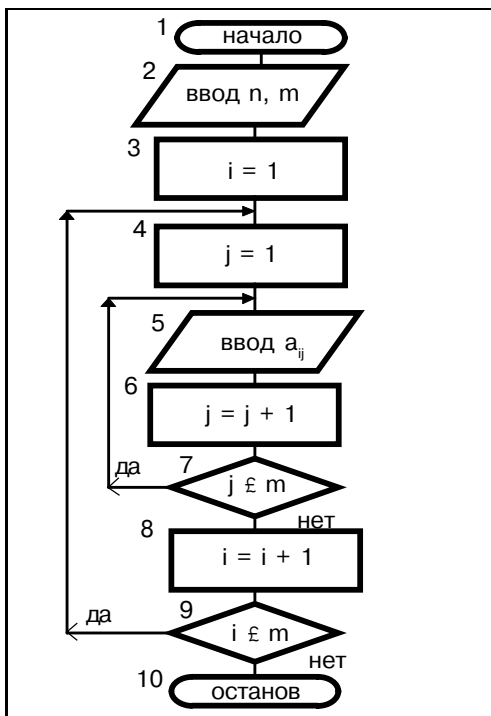


Рис. 36

Такая же схема должна быть использована для второй строки матрицы, для третьей и так далее до m -ой строки. То есть номер строки может быть задан циклом и этот цикл будет внешним по отношению к циклу по столбцу.



Внешний цикл по i - блоки 3, 4, 5, 6, 7, 8, 9.

Блок 3 - подготовка цикла по i .

Блок 8 - изменение параметра цикла.
Блок 9 - проверка условия окончания цикла.

Блоки 4, 5, 6, 7, 8 - тело цикла.

Внутренний цикл по j - блоки 4, 5, 6, 7.

Блок 4 - подготовка цикла по j .

Блок 6 - изменение параметра цикла.

Блок 7 - проверка условия окончания цикла.

Блоки 5, 6 - тело цикла.

Рис. 37. Схема ввода матрицы $[A]$ без блока модификации

При переходе к следующему значению i (к следующей строке) j восстанавливает первоначальное значение равное 1.

Задана матрица

$$[A] = \begin{bmatrix} 1 & 0 & 2 & 3 \\ 0 & 1 & 1 & 1 \\ 5 & 4 & 6 & 0 \end{bmatrix}$$

При использовании разработанной схемы алгоритма элементы матрицы должны быть введены в следующем порядке: 1, 0, 2, 3, 0, 1, 1, 1, 5, 4, 6, 0.

Для того, чтобы ввести элементы матрицы по столбцам (1, 0, 5, 0, 1, 4, 2, 1, 6, 3, 1, 0), нужно цикл по j сделать внешним, а цикл по i внутренним. В этом случае второй индекс меняется медленнее первого.

$$a_{11}, a_{21}, a_{31}, \dots, a_{n1}, a_{12}, a_{22}, a_{32}, \dots, a_{n2}, \dots, a_{1m}, a_{2m}, a_{3m}, \dots, a_{nm}$$

Встречаются задачи, в которых не имеет значения, по какому параметру организовать внешний и внутренний цикл. Но бывают случаи, когда это принципиально.

При использовании блока модификации схема выглядит следующим образом (рис. 38).

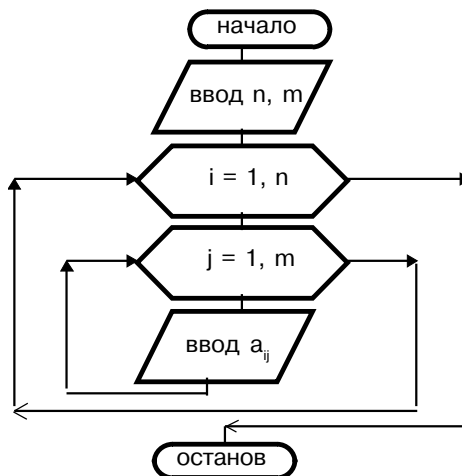


Рис. 38

Матрицы в программах называются двумерными массивами. Двумерные массивы описываются с помощью оператора DIM. Например, DIM c(5,6) резервирует место под двумерный массив c, содержащий шесть строк и семь столбцов (включая нулевые).

Задача 13 (рис. 39). Задана квадратная матрица [A], состоящая из n строк и n столбцов. Найти сумму диагональных элементов.

Разработка алгоритма. Исходные данные: количество строк n, количество столбцов n, элементы матрицы.

Диагональные элементы матрицы $a_{11}, a_{22}, a_{33}, \dots, a_{nn}$ удовлетворяют условию $i = j$. Необходимо осуществить перебор элементов матрицы и просуммировать те, что отвечают условию $i = j$.

Блоки 2, 3, 4, 5 - ввод матрицы [A].

Блок 6 - задание начального значения для суммы.

Блоки 7, 8 - циклы, осуществляющие перебор элементов матрицы.

Блок 9 - проверка условия для суммирования.

Блок 10 - накопление суммы.

Блок 11 - вывод суммы.

Программа:

' Задача 13

INPUT "Размер массива "; n

DIM a (n, n)

FOR i = 1 TO n

FOR j = 1 TO n

PRINT "Введите a(" ; i ; " ; " ; j ; ")"; : INPUT a(i, j)

NEXT j

NEXT i

S = 1

```

FOR i = 1 TO n
FOR j = 1 TO n
  IF j = i THEN S = S + a(i , j)
NEXT j
NEXT i
PRINT "Сумма диагональных элементов = " ; S
END

```

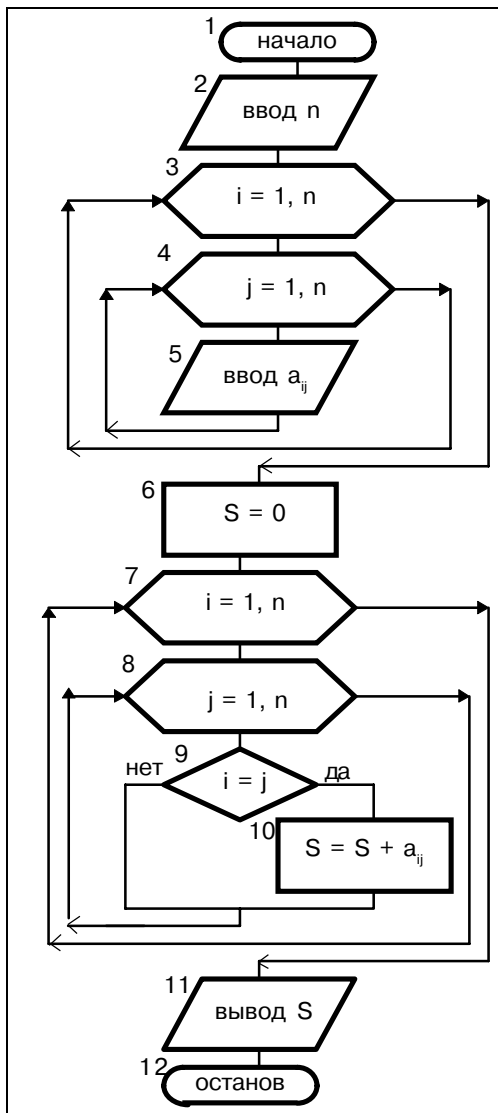


Рис. 39

Схема значительно упрощается, если для обозначения индекса строки и индекса столбца использовать одну и ту же переменную. В этом случае можно ограничиться одним циклом при суммировании. Фрагмент схемы представлен на рис. 40:

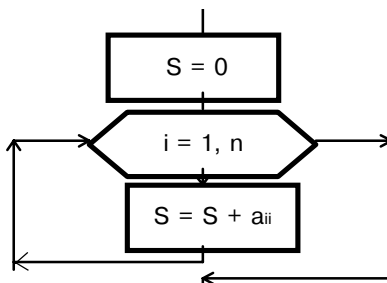


Рис. 40

В программе этот фрагмент будет выглядеть следующим образом:

```

FOR i = 1 TO n
    S = S + a(i, i)
NEXT i
    
```

Задача 14 (рис. 41). Группа из n человек сдавала в зимнюю сессию m экзаменов. Известны результаты экзаменов. Подсчитать средний балл каждого студента в зимнюю сессию.

Разработка алгоритма. Результаты экзаменов представляют собой матрицу, состоящую из n строк и m столбцов. Строка представляет собой оценки каждого студента, полученные им по предметам сессии. Номер столбца - это номер предмета. Задача заключается в том, чтобы найти сумму элементов каждой строки.

$$S_i = \sum_{j=1}^m a_{ij}.$$

Тогда среднее:

$$S_i = S_i / m.$$

Результатом расчета будет одномерный массив S , состоящий из n элементов.

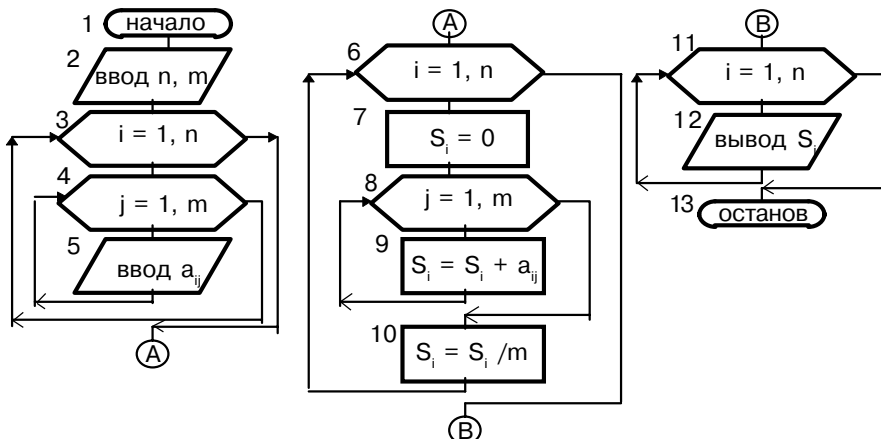


Рис. 41. Схема алгоритма

Блок 6 - цикл по строкам. В задаче i - это номер студента в ведомости. Блок 7 - обнуление суммы. $S(i)$ - переменная, в которой накапливается сумма оценок i -го студента. Блок 8 - перебор предметов от 1 до m . Блок 9 - суммирование оценок. Блок 10 - определение среднего. После того, как цикл по j отработал, полученная сумма делится на количество предметов. Блоки 11, 12 - вывод одномерного массива.

Программа:

‘ Задача 14

```
INPUT " Количество студентов в группе " ; n
INPUT " Количество экзаменов в сессию " ; m
DIM a( n , m)
FOR i = 1 TO n
PRINT "Оценки " ; i ; "- го студента"
FOR j = 1 TO m
PRINT "По " ; j ; "- му экзамену"; : INPUT a( i , j)
NEXT j
NEXT i
FOR i = 1 TO n
S ( i ) = 0
FOR j = 1 TO m
S ( i ) = S ( i ) + a( i , j)
NEXT j
S ( i ) = S ( i ) / m
NEXT i
FOR i = 1 TO n
PRINT "Средний балл " ; i ; "- го студента" ; S (i)
NEXT i
END
```

В этой задаче желательно было бы ввести фамилии студентов, сдававших сессию. Тогда программа приобретает следующий вид.

‘ Задача 14

```
INPUT " Количество студентов в группе " ; n
INPUT " Количество экзаменов в сессию " ; m
DIM a( n , m ) , f$( n )
FOR i = 1 TO n
PRINT "Фамилия " ; i ; "- го студента"; : INPUT f$( i )
FOR j = 1 TO m
PRINT "Оценка " ; f$( i ) ; "по " ; j ; "- му экзамену"; : INPUT a( i , j)
NEXT j
NEXT i
FOR i = 1 TO n
s ( i ) = 0
FOR j = 1 TO m
s ( i ) = s ( i ) + a( i , j)
NEXT j
s ( i ) = s ( i ) / m
```

```

NEXT i
FOR i = 1 TO n
PRINT "Средний балл " ; f $ ( i ) ; s (i)
NEXT i
END

```

В программе появился одномерный символьный массив, состоящий из фамилий студентов.

5.5. Вывод результатов в виде графиков и таблиц

При решении задач на ЭВМ большое значение имеет наглядность и удобство дальнейшего использования выводимых результатов. **Оператор форматного вывода данных PRINT USING** позволяет более гибко управлять формой представления выводимых данных. Значения выводятся по форматам, задаваемым программистом в списке форматов. Форма оператора

PRINT USING список форматов; список вывода

Список вывода оператора PRINT USING аналогичен списку вывода оператора PRINT. Элементы списка перечисляются через запятые.

Список форматов заключается в кавычки и состоит из строковых констант, числовых и строковых форматов, записанных без пробелов и разделителей.

Числовой формат указывается символом #. Каждый такой символ соответствует одному разряду десятичного числа. Положение десятичной точки определяется символом "точка". Например, операторы

```

PRINT USING "###.##" ; 19.849
PRINT USING "##.###" ; -2.54
PRINT USING "S = ##.#" ; 19.849
PRINT USING "X = ####" ; -13987

```

вызовут появление следующих сообщений на терминале

19.85

-2.540

S = 19.8

X = %-13987

Знак % в последнем выводе указывает, что заданное в шаблоне числовое поле мало для вывода числа.

Если список форматов содержит несколько форматов и список элементов - несколько элементов, то первый элемент списка выводится по первому формату, второй - по второму и т. д. Когда список форматов будет исчерпан, форматы начнут выбираться с начала списка, а вывод продолжится с новой строки.

Четыре символа ^ ^ ^ ^, которыми заканчивается формат, означают вывод числового значения в экспоненциальной форме.

Строковый формат для вывода строк символов задается одним из следующих трех способов:

! - задает вывод только первого символа строки;

& - задает вывод всей строки;

n пробелов\ - задает вывод 2 + n первых символов строки.

Задача 15. Вывести на экран ведомость для оплаты коммунальных услуг в следующем виде:

Стоимость услуг	Руб
Вода	0.96
Электричество	4.52
Телефон	2.50
Отопление	1.86

Разработка алгоритма. Воспользуемся числовым массивом x из четырех элементов для стоимости услуг, для наименования услуг - символьным массивом $c\$$ из четырех элементов. Горизонтальную строку сформируем из символов "-". Для ввода исходных данных используем оператор DATA/READ.

Программа:

'Задача N15

DATA Вода, Электричество, Телефон, Отопление

DATA 0.96, 4.52, 2.50, 1.86

FOR $i = 1$ TO 4 'Ввод символьного массива из блока DATA

READ $c\$(i)$

NEXT i

FOR $i = 1$ TO 4 'Ввод числового массива из блока DATA

READ $x(i)$

NEXT i

FOR $i = 1$ TO 26 'Формирование горизонтальной линии

$a\$ = a\$ + "-"$

NEXT i

PRINT TAB(10) "ВЕДОМОСТЬ" ' Вывод шапки таблицы

PRINT $a\$$

PRINT TAB(4) "| Стоимость услуг | Руб |

PRINT $a\$$

$f\$ = "| \quad \quad \quad \backslash | \# \# . \# \# | "$ 'Присвоение симв. перем. списка форматов

FOR $i = 1$ TO 4 'Вывод содержимого таблицы

PRINT USING $f\$$; $c\$(i)$, $x(i)$

NEXT i

PRINT $a\$$ ' Вывод горизонтальной линии

END

Вывод данных на устройство печати выполняется операторами LPRINT и LPRINT USING. Формат операторов полностью идентичен формату операторов вывода данных на экран.

Задача 16. Построить график функции $\cos x$ в интервале от 0 до 2π с шагом $\pi/8$.

Разработка алгоритма. Точки графика изобразим с помощью символов "*", располагаемых в последовательных строках на расстоянии от начала строки, пропорциональном значению функции (рис. 42). При этом вертикальная ось экрана является осью абсцисс x , а горизонтальная - осью ординат y , т. е. график оказывается повернутым относительно привычного представления на 90° . Вывод звездочки в нужной позиции можно было бы осуществить с помощью оператора PRINT TAB (COS (x)), но значения функции $\cos x$ изменяются от -1 до 1. Необходимо выбрать масштаб. Если располагать график на экране от 5-й до 65-й позиции, то цена деления (одной позиции) по горизонтали $s = 2 / 60$. Вывод звездочки следует делать в позиции $35 + 30 * \text{COS}(x)$.

При вычерчивании оси абсцисс следует иметь в виду, что при $\cos x > 0$ символ, изображающий ось x (|), должен печататься до символа, изображающего функцию, при $\cos x < 0$ - наоборот. Если же функция равна 0, то в одну позицию должен выводиться только один символ “*”. Поэтому в программе должно быть три различных оператора PRINT.

Программа.

```
PRINT TAB (4) ; “-1” ; TAB( 35 ) ; “0” ; TAB ( 65 ) ; “1”
a$ = “” : FOR i = 1 TO 65 : a$ = a$ + “-” : NEXT i
a$ = a$ + “> Y” : PRINT a$
x = 0
WHILE x < 6.28
a = 30 * COS ( x ) + 35
IF COS(X) THEN
PRINT TAB (35) ; “*”
ELSE
IF COS(X) > 0 THEN
PRINT TAB (35) ; “|” ; TAB (a) ; “*”
ELSE
PRINT TAB (a ) ; “*” ; TAB (35) ; “|”
END IF
END IF
x = x + 3.14 / 8
WEND
PRINT TAB (35 ) ; “v” : PRINT TAB (35 ) ; “X”
END
```

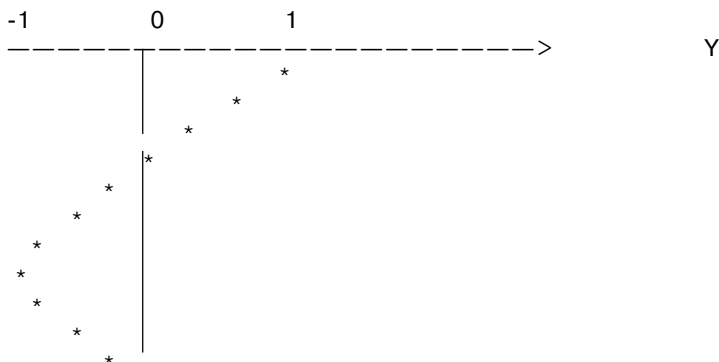


Рис. 42. Вывод части графика

В QBASIC для рисования одинарных и двойных линий, рамок и закрашки изображений шаблонами среди элементов таблицы ASCII имеются символы псевдографики. Эти символы вводятся на дополнительной клавиатуре при одновременном нажатии клавиши Alt. Коды символов псевдографики приводятся в литературе, например, [2, стр. 459].

5.6. Работа в среде программирования QBASIC

При работе в среде QBASIC экран монитора выглядит следующим образом:

строка меню
экран программы
экран команд непосредственного исполнения
строка контекстной подсказки

В экране программы записываются операторы программы один за другим. В строке может быть записан один оператор или несколько операторов. Если в строке записаны несколько операторов, они разделяются двоеточием (:). Ввод набранной строки завершается нажатием клавиши {enter}. Встроенный редактор среды QBASIC обладает всеми основными свойствами редактора текстов. Интеллектуальный редактор среды QBASIC осуществляет синтаксический контроль вводимого текста. Если строка написана правильно, то после того, как вы перевели курсор на следующую строку, все ключевые слова QBASIC будут записаны заглавными буквами, операторы отделены друг от друга. Это позволяет сразу понять, есть ли в строке ошибки. Если же допущена ошибка с точки зрения синтаксиса языка, будет высвечено диалоговое окно с описанием возникшей ошибки, а курсор установится на место предполагаемой ошибки.

Редактор QBASIC поддерживает работу с блоками текста. Это бывает полезно при копировании участков программы. Для выделения блоков текста используются клавиши: {Shift} + стрелки перемещения курсора. Выделенный фрагмент можно: {del} - стереть навсегда; {Shift} + {del} - стереть, но запомнить в буфере; {Ctrl} + {ins} - запомнить в буфере, не стирая; {Shift} + {ins} - вставить текст из буфера. Текст, запомненный в буфере, можно вставить в другое место программы неограниченное число раз.

К числу несомненных достоинств среды QBASIC относится система оперативной подсказки. Чтобы вы ни делали, нажав клавишу {F1}, всегда можно получить объяснение.

Переход в экран непосредственного исполнения операторов осуществляется нажатием клавиши {F6}. В экране непосредственного исполнения каждая строка, заканчивающаяся нажатием клавиши конца ввода <enter>, передается на выполнение интерпретатору QBASIC. Это очень удобно использовать при отладке: оборвав выполнение программы, можно изменить значение какой-либо переменной и продолжить выполнение программы; можно вывести промежуточные значения переменных.

Для выхода в меню используется клавиша {alt}. При нажатии этой клавиши курсор перемещается в строку меню и высвечивает один из пунктов. Каждый пункт меню содержит подпункты. Чтобы подпункты высветились на экране, необходимо нажать клавишу {enter}. В QBASIC достаточно разветвленная система меню. Кратко перечислим назначение пунктов меню:

Пункт *“Файл”* - используется для создания новой программы, загрузки и сохранения программ или частей программ, печати файлов или частей файлов, выхода из среды QBASIC.

Пункт “Редактирование” - используется для стирания, копирования или перемещения текста программы, создания новой процедуры (SUB) и функции (FUNCTION).

Пункт “Просмотр” - используется для просмотра процедур (SUB) и функций (FUNCTION), экрана программы.

Пункт “Поиск” - предназначен для поиска или замены названий переменных, меток или фрагментов исходного текста в активном окне, в текущем модуле или во всех загруженных модулях.

Пункт “Запуск” - используется для исполнения загруженной программы, продолжения выполнения прерванной программы, очистки переменных в памяти перед выполнением.

Пункт “Отладка” - используется для отладки программы путем открытия окон наблюдения, которые показывают, как переменные изменяются при работе программы; установки точек прерывания, которые прерывают выполнение программы для того, чтобы вы могли просмотреть значения переменных.

Пункт “Опции” - используется для настройки цветов экрана, установки путей для поиска служебных файлов, переопределения правой кнопки мыши и т. п.

Пункт “Помощь” - используется для получения справки по ключевым словам QBASIC, информации по языку программирования, контекстно-зависимой помощи, определяемой месторасположением курсора.

На первых порах необходимо знание содержимого пункта “Файл”. Пункт “Файл” состоит из следующих подпунктов:

“Новый” - очистка экрана.

“Открыть” - вызов в оперативную память программы, записанной на диске.

“Сохранить” - запись файла из оперативной памяти на диск.

“Сохранить как” - запись файла с новым именем.

“Печать” - печать программы или предварительно выделенной части программы.

“Выход” - выход из среды QBASIC.

Выполнение любого пункта меню не требует дополнительной подготовки, так как происходит в диалоговом режиме. Пользователю предоставляется возможность отвечать на запросы системы.

Для наиболее часто используемых операций предусмотрены клавиатурные сокращения; {F1} - получение помощи; {F2} - вызов списка процедур и функций; {F4} - просмотр результатов работы программы; {Shift+F5} - запуск программы на выполнение; {alt+F, S} - сохранение файла программы; и т. д.

Загрузив инструментальную среду QBASIC в оперативную память, необходимо ввести новую программу в окно программы, запустить программу на выполнение с помощью набора клавиш {Shift + F5}. Язык QBASIC относится к языкам интерпретирующего типа - при исполнении программы интерпретатор переводит в машинные коды одну строку программы и сразу же ее выполняет. Результаты работы программы (списки операторов вывода) отображаются в отдельном окне. Это окно называется окном результатов. При этом окно программы становится невидимым. Когда программа отработала, на экране появляется надпись: “Нажмите какую-либо клавишу для продолжения”. При нажатии любой клавиши вы опять попадаете в окно программы.

Если в программе допущены ошибки, которые не могут быть обнаружены редактором, появляется заставка с сообщением об ошибке, курсор

устанавливается в позицию ошибочного оператора. В этом случае необходимо проверить правильность разработанного алгоритма или введенных исходных данных. Например, часто встречается ошибка “Неверный аргумент функции”. Начинающий программист не продумал вероятность появления отрицательного числа в качестве аргумента функции LOG(X).

Далее ошибку необходимо исправить и выполнить программу с начала {Shift+F5} или с того места, где произошла остановка {F5}.

Для записи программы на диск используется пункт меню “Файл”, “Сохранить”. На экране появится заставка с предложением ввести имя файла. Необходимо ввести имя файла и нажать {enter}. При следующем сеансе этот файл можно будет считать, используя подпункт “Открыть” пункта “Файл”.

6. МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

QBASIC позволяет разрабатывать программы по модульному принципу. **Модуль** - это отдельная, полностью независимая от других часть программы. Каждая программа на QBASIC может состоять из одного или нескольких модулей.

Один из модулей называется главным. Он содержит так называемую точку входа, с которой начинается выполнение программы. В каждой программе может быть только один главный модуль. К главному модулю можно подключить один или несколько вспомогательных модулей. **Вспомогательные модули** - это отдельные файлы программ, в которых содержатся процедуры и функции, использующиеся одновременно в нескольких программах. У каждого программиста через некоторое время появляется большой набор заготовок, неординарных решений, которые он может оформить в виде процедур и функций. В составе модуля можно выделить следующие самостоятельные блоки: функции DEF FN, подпрограммы GOSUB, процедуры SUB и FUNCTION.

Большинство версий языка BASIC располагает обширными библиотеками стандартных подпрограмм, которые существенно облегчают программирование реальных задач. **Стандартные подпрограммы** - организованные и оформленные стандартным образом готовые подпрограммы, которые при задании соответствующих параметров могут использоваться для решения типовых задач. Имеющийся набор стандартных подпрограмм, организованный соответствующим образом, называется **библиотекой стандартных подпрограмм**.

6.1. Подпрограммы

Когда некоторая совокупность действий должна выполняться в нескольких различных местах программы, то нежелательно каждый раз повторять группу операторов, реализующих эти действия. Чтобы избежать повторений, указанную группу операторов можно записать в программе один раз и обращаться к ней, когда в этом возникает необходимость. Обособленную группу операторов, которую можно выполнять многократно, обращаясь к ней из различных мест программы, называют **подпрограммой**.

Чтобы подпрограмма при обращении к ней выполнялась каждый раз с новыми данными, ее нужно составить в общем виде, а исходные данные для работы передавать в переменные подпрограммы перед обращением к ней. Если,

например, в программе требуется решить три квадратных уравнения с различными коэффициентами, то алгоритм нахождения корней квадратного уравнения целесообразно оформить в виде подпрограммы, используя для обозначения коэффициентов переменные. Перед каждым обращением к подпрограмме нужно задать этим переменным числовые значения, соответствующие коэффициентам решаемых уравнений.

Использование подпрограмм улучшает структуру программы. Облегчается отладка программ, так как работа каждой подпрограммы может быть проверена по отдельности.

Подпрограмма может оформляться:

- 1) как группа операторов, составляющих с программой единое целое;
- 2) как отдельная процедура SUB, текст которой не входит в главную часть модуля.

Обращение к подпрограмме осуществляется оператором

GOSUB N,

где N - метка строки, с которой начинается подпрограмма.

Подпрограмма размещается в последовательных строках, начиная с N-й.

Последним оператором, выполняемым в подпрограмме, должен быть оператор RETURN. По оператору RETURN осуществляется возврат в то место программы, из которого произошло обращение к подпрограмме, а именно к оператору, следующему за GOSUB N. Подпрограмма может содержать обращения к другим подпрограммам. При использовании подпрограмм следует следить за тем, чтобы перед обращением к подпрограмме ее переменные получили нужные значения.

Во втором разделе разработан алгоритм вычисления числа сочетаний по

формуле
$$C_n^m = \frac{n!}{m!(n-m)!}$$

Программа на языке QBASIC:

‘Расчет числа сочетаний

PRINT “Введите n, m”;

INPUT n,m

L = n : GOSUB 1000 : c1 = p

L = m : GOSUB 1000 : c2 = p

L = n - m : GOSUB 1000

c = c1 / (c2 * p)

PRINT “ Число сочетаний из “ ; n ; “ по “ ; m ; “ = “ ; c

STOP

1000 ‘ Подпрограмма вычисления факториала

p = 1

FOR i = 2 TO L

p = p * i

NEXT i

RETURN

Программа может заканчиваться оператором STOP. Оператор END должен быть обязательно последним оператором в программе.

GOSUB ... RETURN - устаревшая программная конструкция. Вместо нее лучше использовать процедуры SUB или FUNCTION, функцию DEF FN. Использовать GOSUB ... RETURN имеет смысл только внутри процедур SUB или FUNCTION, при

обращении к одному фрагменту текста, когда выделение фрагмента в отдельную процедуру нецелесообразно.

SUB ... END SUB - операторы, указывающие начало и конец процедуры SUB.

Процедура SUB в языке QuickBasic может использоваться вне того модуля, где она определена. Процедура SUB, которая находится во вспомогательном модуле, доступна любому модулю программы. Процедура оформляется следующим образом:

```
SUB имя [список параметров]
[операторы]
[EXIT SUB]
[операторы]
END SUB
```

Имя - идентификатор, до 40 знаков, имя процедуры. Это имя не должно использоваться другими операторами FUNCTION или SUB или в подключаемой библиотеке. Список параметров - содержит имена простых переменных и массивов, передаваемых в процедуру. Имена в списке разделяются запятыми. Любое изменение аргумента в процедуре приведет к изменению его значения в вызывающем модуле.

Список параметров имеет следующую форму:

переменная [()] [AS тип] [, переменная [()] [AS тип]] ...

Переменная - имя переменной или массива. Массив задается с пустыми скобками.

SUB и END SUB отмечают начало и конец процедуры. Для выхода из процедуры можно использовать также EXIT SUB.

Процедуры вызываются оператором CALL или путем указания имени со списком аргументов. Ключевое слово CALL можно опускать в том случае, если процедура SUB объявлена оператором DECLARE.

Пусть существует процедура печати сообщения в указанных строке и столбце.

```
SUB Message (r, c, Mess$)
curr = CSRLIN
curc = POS(0)
LOCATE r,c : PRINT Mess$;
LOCATE curr,curc
END SUB
```

В процедуре встретились незнакомые стандартные функции CSRLIN и POS (0). CSRLIN возвращает номер строки, в которой в данный момент находится курсор. POS (0) возвращает номер столбца, в котором в данный момент находится курсор. Процедура запоминает позицию курсора, выводит сообщение Mess\$ в позицию с координатами r и c и восстанавливает текущую позицию курсора curr и curc. Если в головном модуле возникает необходимость вывода сообщения "Hello!" в центре экрана, то обращение к процедуре будет выглядеть:

```
CALL Message (12, 40, "Hello!")
```

Если процедура SUB объявлена оператором DECLARE

```
DECLARE SUB Message (r, c, Mess$),
```

то при вызове процедуры ключевое слово CALL можно опустить
Message 12, 40, "Hello!"

Процедура SUB может возвращать несколько значений в вызывающую

подпрограмму. Она не может быть использована как часть выражения. Процедуры могут вызывать сами себя, то есть могут быть **рекурсивными**.

6.2. Функции

В QBASIC существует два вида функций: DEF FN и FUNCTION.

Функция DEF FN - входит в главную часть модуля. С ее помощью можно определить и далее использовать нестандартные функции. Общий вид оператора:

DEF FNv (x1 [, x2, ...]) = арифметическое выражение,

где v - имя функции - латинская буква; x1, x2, ... - простые переменные - формальные аргументы функции; арифметическое выражение - формула, по которой вычисляется функция. Арифметическое выражение в правой части должно обязательно содержать хотя бы один из формальных аргументов x1, x2, ..., но может так же содержать и другие переменные, общие для всей программы.

Оператор DEF FN должен располагаться в программе до первого использования определяемой им функции. Правила хорошего тона требуют помещать такие операторы в начале программы.

Вычисление функции, описанной оператором DEF FN, осуществляется при обращении к ней при помощи записи указателя функции:

FNv (a1 [, a2, ...]),

где a1, a2, ... - арифметические выражения, заменяющие формальные аргументы x1, x2, ... в арифметическом выражении правой части оператора DEF FN перед вычислением.

Использование функции, заданной оператором DEF FN (нестандартной функции), аналогично использованию стандартных функций. Обращение к нестандартной функции можно записать в арифметическом выражении, в списке вывода оператора PRINT и т. д., вообще везде, где требуется значение этой функции. Например,

```
DEF FNE(x, y) = x * x * x + y * y + a * a
```

```
...
```

```
a = 2 : t = 3.1 : z = 8.6
```

```
...
```

```
p = 0.5 * SQR ( FNE ( SIN( t / 22 ), 2 * z ) )
```

В правой части последнего оператора присваивания имеется обращение к функции FNE, используемой в качестве аргумента стандартной функции SQR. При вычислении функции FNE сначала будут вычислены арифметические выражения SIN(t / 22) и 2 * z, значения которых заменят формальные аргументы x и y в арифметическом выражении, определяющем функцию FNE (x, y), а затем вычислено значение этого арифметического выражения.

Другой пример. В QBASIC нет стандартной функции вычисления десятичного логарифма; ее можно определить с помощью DEF:

```
DEF FNL ( x ) = LOG ( x ) / LOG( 10.0 )
```

Функция DEF FN - это устаревшая программная конструкция, доставшаяся "по наследству" от GW - BASIC. Практически во всех случаях лучше использовать более современную конструкцию FUNCTION ... END FUNCTION. Существуют редкие случаи, когда использование функции DEF FN предпочтительнее. В отличие от FUNCTION, функция DEF FN не проверяет тип числового аргумента.

Процедура **FUNCTION** является более мощной альтернативой DEF FN. В языке QuickBasic процедура FUNCTION может использоваться вне того модуля, где она определена. Это означает, что она становится доступна любому модулю программы, если определена во вспомогательном модуле. Необходимо добавить оператор DECLARE в те модули, где она используется.

FUNCTION ... END FUNCTION - операторы, указывающие начало и конец процедуры FUNCTION. Формат:

```
FUNCTION имя [ ( параметры ) ]  
[ операторы ]  
имя = выражение  
[ операторы ]  
END FUNCTION
```

Имя задается как обычная переменная. *Параметры* - список переменных, значения которых передаются функции при ее вызове. Изменение значения параметра внутри функции изменит его значение в вызывающем модуле. *Выражение* - возвращаемое значение функции. Если имени функции ничего не присваивается, возвращается значение по умолчанию: для числовых функций - ноль, для символьных - пустая строка. Чтобы передать в процедуру в качестве аргумента массив, достаточно указать пустые скобки.

Процедуры FUNCTION могут быть рекурсивными, то есть вызывать сами себя. Типичным случаем применения рекурсии является функция вычисления факториала:

```
n! = n * ( n - 1 ) * ( n - 2 ) * ... * 2 * 1  
FUNCTION Fact ( n% )  
IF n% > 1 THEN  
    Fact = n% * Fact ( n% - 1 )  
ELSE  
    Fact = 1  
END IF  
END FUNCTION
```

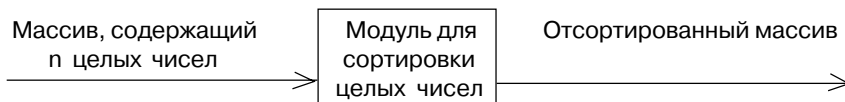
В основной программе вычисление факториала будет осуществляться каждый раз, когда в арифметическом выражении встретится имя функции Fact, например, a% = Fact(c%).

6. 3. Метод пошаговой детализации

Программирование можно было бы определить как “проектирование, кодирование и тестирование программы”. Для того, чтобы писать грамотные программы с наименьшими затратами сил, необходимо программировать в соответствии с заранее определенной дисциплиной. Программирование с заранее определенной дисциплиной называется **нисходящим структурным программированием**. Нисходящее структурное программирование включает в себя метод пошаговой детализации и использование базовых структур при составлении алгоритмов и программ.

На этапе проектирования необходимо построить структуру программы, выбрать или разработать все алгоритмы, которые она будет реализовывать, решить все вопросы по организации данных. Здесь эффективным подходом

является разложение сложной задачи на некоторые подзадачи. Каждая отдельная подзадача должна быть относительно независимой и представлять собой некоторый законченный модуль программы. Таким образом, модульность (способность разделения) программы является ее важным свойством. В целом внешняя спецификация модуля выражается описанием того, что он делает с некоторыми допущениями; того, что ему можно подать на вход, и того, что можно получить на его выходах. Спецификация модуля, используемого для сортировки



Предположения: $0 \leq n \leq 1000$

целых чисел:

После разработки общей структуры программы необходимо установить, какие библиотечные (ранее разработанные) средства можно использовать и какие новые процедуры необходимо разрабатывать. Далее переходят к разработке новых структур данных и алгоритмов выполнения новых процедур.

К разработке отдельных модулей применяется тот же подход, что и ко всей программе. Такой метод называется **методом пошаговой детализации**. В методе пошаговой детализации можно выделить следующие существенные этапы:

На уровне 1 создается общее описание модуля в целом. Определяются основные логические шаги, требуемые для решения задачи, даже если пока не известно, как их выполнить. Это могут быть групповые имена для тех действий, выполнение которых представляется довольно смутно.

На уровне 2 в общих терминах детализируется описание шагов, введенных на этапе 1. В детализированное описание может входить обозначение циклических структур, в то время, как действия внутри циклов могут по-прежнему оставаться неясными. Таким образом, выполняются только общие эскизы сложных действий.

На уровне 3 и в последующих уровнях в виде последовательных итераций производятся те же действия, что описаны на этапе 2. При каждой новой итерации уточняются детали, оставшиеся неясными после предыдущих итераций, и создаются более определенные описания. Если в процессе пошаговой детализации возникли детали и переменные, не зависящие от остальных частей программы, или произошло дублирование модулей, то подобные модули являются очень вероятными кандидатами в подпрограммы или функции. В подобной ситуации модуль описания используется для создания тела процедуры или функции и заменяется на вызов процедуры или функции.

Пример. Пусть требуется написать программу, которая получает в качестве исходных данных значение n , равное числу учеников в классе, а также рост, возраст и вес каждого ученика. Программа должна выдать средние значения всех трех показателей.

Уровень 1.

Ввести число учеников.

Вычислить средний рост.

Вычислить средний возраст.

Вычислить средний вес.

Применим этап 2 к каждому из описанных выше шагов.

Детализация 1.1. Вычислить средний рост.

Ввести рост каждого ученика.

Вычислить, используя циклические вычисления, средний рост.

Отобразить результаты.

Детализации 1.2 и 1.3 аналогичны приведенной выше, но применяются к возрасту и весу вместо роста. В результате получен *уровень 2*.

Продолжим этот процесс и, поскольку все три детализации практически одинаковы, рассмотрим детализацию 1.1. Раскроем суть трех действий, составляющих ее описание.

Детализация 2.1. Ввести рост каждого ученика.

Печать приглашения к вводу значений.

Ввод значений в массив, используя цикл.

Детализация 2.2. Вычислить с помощью цикла средний рост.

Организовать цикл, пробегающий n значений и накапливая сумму в s .

Поделить s на n для получения среднего роста.

Детализация 2.3. Изобразить результаты.

Печать названия показателя и его среднего значения

Так как производится по сути одно и то же расширение всех трех основных частей уровня 2, то представляется естественным реализовывать указанные выше действия в виде подпрограмм общего назначения, передавая им в строковой переменной название показателя - рост, возраст, вес.

Уровень 3.

Ввод "Число учеников"; n

Присвоить переменной "показатель" значение "Рост".

Вызвать детализации 2.1, 2.2 и 2.3.

Присвоить переменной "показатель" значение "Возраст".

Вызвать детализации 2.1, 2.2 и 2.3.

Присвоить переменной "показатель" значение "Вес".

Вызвать детализации 2.1, 2.2 и 2.3.

Конец

Полученная структура является модульной. Неявным образом было принято решение вводить значения всех показателей в один и тот же массив, скажем $a()$, и передавать название показателя через одну строковую переменную, скажем $a\$$.

Детализация 2.1.

1000 ' Подпрограмма ввода значений

FOR $i = 1$ TO n

PRINT "Введите " ; $a\$$; i ; " - го ученика";

INPUT $a(i)$

NEXT i

RETURN

Детализация 2.2.

1001 ' Подпрограмма вычисления среднего значения

$s = 0$

FOR $i = 1$ TO n

$s = s + a(i)$

NEXT i

$s = s / n$

RETURN

Детализация 2.3.

1002 ' Подпрограмма вывода значений

PRINT "Среднее значение " ; $a\$$; " равно " ; s

RETURN

Так как все три действия всегда выполняются последовательно, то можно сэкономить на повторяющихся вызовах, поместив их также в подпрограмму.

```
200 ' Подпрограмма процесса  
GOSUB 1000 : GOSUB 1001 : GOSUB 1002  
RETURN
```

Основная программа:

```
INPUT "Число учеников" ; n  
DIM a( n )  
a$ = "Рост" : GOSUB 200  
a$ = "Возраст" : GOSUB 200  
a$ = "Вес" : GOSUB 200  
STOP
```

Для более сложных задач желательно использовать графические изображения алгоритмов: схемы или структограммы [5].

Программы на QBASIC могут быть сколь угодно большими. Оперативная память компьютера не накладывает ограничений на величину программы. Отдельные модули программы могут вызываться с помощью оператора CHAIN. Оператор CHAIN вызывает в оперативную память программный модуль с указанным именем, при этом все строки старой программы уничтожаются, открытые файлы закрываются, в оперативную память помещается новая программа и процесс вычислений продолжается. Общий вид оператора

CHAIN имя

Значения переменных при взаимодействии таких программ могут передаваться через файлы данных [1, стр.128; 2 стр. 107].

Подробнее см. [1, 2, 4, 5].

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

- 1. Составьте логическую схему базы знаний по теме юниты.*

ТРЕНИНГ УМЕНИЙ

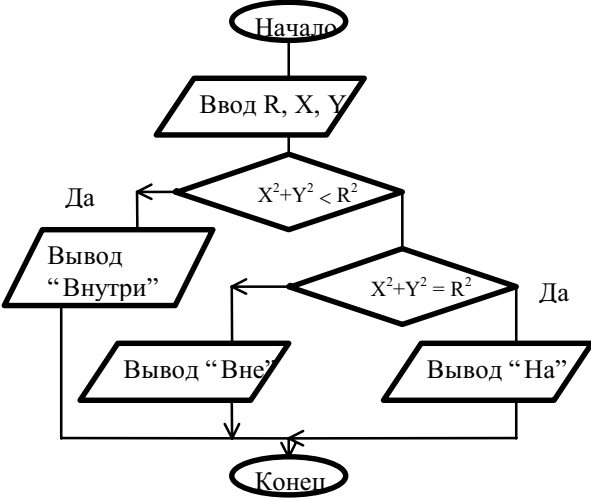
Примеры выполнения упражнений тренинга на умение 1

Задание 1

Даны координаты точки X и Y. Определить, находится ли точка внутри окружности радиусом R, лежит ли она на окружности или находится вне ее.

Решение

№	Алгоритм	Конкретное действие
1	Поставить задачу.	Уравнение окружности $R^2 = X^2 + Y^2$. Точка находится внутри окружности при условии $R^2 > X^2 + Y^2$. Точка находится вне окружности при условии $R^2 < X^2 + Y^2$. Точка находится на окружности при условии $R^2 = X^2 + Y^2$.
2	Выбрать метод решения	Метод решения очевиден.
3	Определить последовательность действий, ведущих к получению результатов	Задать значения переменных R, X, Y. Проверить условие $R^2 > X^2 + Y^2$ и, если условие выполняется, выдать сообщение “Точка находится внутри окружности”. Проверить условие $R^2 = X^2 + Y^2$. Если условие выполняется, выдать сообщение “Точка находится на окружности”, иначе выдать сообщение “Точка находится вне окружности”.

№	Алгоритм	Конкретное действие
4	Дать точное предписание вычислительного процесса в виде последовательно размещенных блоков в соответствии с ГОСТом.	 <pre> graph TD Start([Начало]) --> Input[/Ввод R, X, Y/] Input --> Dec1{X²+Y² < R²} Dec1 -- Да --> Out1[/Вывод "Внутри"/] Dec1 -- Нет --> Dec2{X²+Y² = R²} Dec2 -- Да --> Out2[/Вывод "На"/] Dec2 -- Нет --> Out3[/Вывод "Вне"/] Out1 --> End([Конеч]) Out2 --> End Out3 --> End </pre>

Задание 2

В соревнованиях по бегу принимают участие N спортсменов. Вводя по очереди результаты участников в ЭВМ, определить, сколько из них выполнило норму ГТО.

Решение

№	Алгоритм	Конкретное действие
1	Поставить задачу.	r — результат участника соревнования; gto — норма ГТО; $r < gto$ — норма выполнена.
2	Выбрать метод решения	Используем прием подсчета количества $S=S+1$.
3	Определить последовательность действий, ведущих к получению результатов	Задать количество участников соревнования N . Организовать цикл по i . Для этого переменной i присвоить значение 1; в цикле ввести значение результата забега i -го спортсмена; сравнить результат с нормой ГТО; если норма выполнена, включить счетчик $S=S+1$; увеличить i на единицу;

№	Алгоритм	Конкретное действие
3		организовать проверку окончания цикла $i > N$. Если условие выполняется, то выдать значение S , иначе продолжить ввод и подсчет количества.
4	Дать точное предписание вычислительного процесса в виде последовательно размещенных блоков в соответствии с ГОСТом.	<pre> graph TD Start([Начало]) --> InputN[/Ввод N/] InputN --> InputGto[/Ввод gto/] InputGto --> Init[S=0; i=1] Init --> OutputR[/Вывод r/] OutputR --> Cond1{gto > r} Cond1 -- Да --> Sum[S=S+1] Sum --> IncI[i=i+1] IncI --> Cond2{i > N} Cond2 -- Да --> OutputS[/Вывод S/] OutputS --> End([Конец]) Cond1 -- Нет --> LoopBack(()) Cond2 -- Нет --> LoopBack LoopBack --> InputGto </pre>

Задание

Вычислить длину вектора X размером 4.

Решение

№	Алгоритм	Конкретное действие
1	Поставить задачу.	Длина вектора вычисляется по формуле $L = \sqrt{(X_1^2 + \dots + X_4^2)}.$
2	Выбрать метод решения	Используем прием накопления суммы.
3	Определить последовательность действий, ведущих к получению результатов	Обнулить начальное значение суммы. Организовать цикл по i , где i меняется от 1 до 4. Ввести элементы массива X_i . В цикле организовать накопление суммы $L = L + X_i^2$. Вывести результат \sqrt{L} .
4	Дать точное предписание вычислительного процесса в виде последовательно размещенных блоков в соответствии с ГОСТом.	<pre> graph TD Start([Начало]) --> L0[L=0] L0 --> i14{i:=1, 4} i14 --> Input[/Ввод Xi/] Input --> Sum[L=L + Xi] Sum --> Output[/Вывод sqrt(L)/] Output --> i14 Output --> End([Конец]) </pre>

Решите самостоятельно следующие задания:

Задание 1

Заданы площади круга R и квадрата S . Определить, поместится ли квадрат в круге.

Задание 2
 Вычислить $\sum_{i=2}^N (i + 1) \cdot (i + 2)$ для всех i от 2 до N .

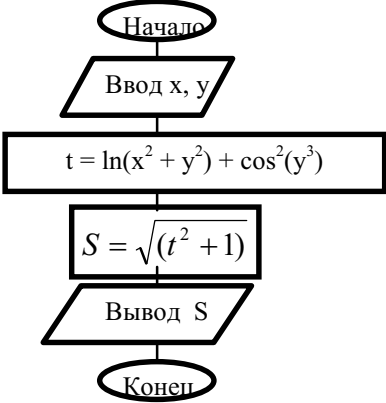
Задание 3
 Задан массив размером 10. Сформировать два массива размером 5, включая в первый элементы исходного массива с четными индексами, а во второй — с нечетными.

Пример выполнения упражнения тренинга на умение 2

Задание

Рассчитать значение S по формуле $S = \sqrt{(t^2 + 1)}$, где $t = \ln(x^2 + y^2) + \cos^2(y^3)$.

Решение

№	Алгоритм	Конкретное действие
1	Разработать алгоритм линейной структуры	 <pre> graph TD Start([Начало]) --> Input[/Ввод x, y/] Input --> Process1[t = ln(x² + y²) + cos²(y³)] Process1 --> Process2[S = √(t² + 1)] Process2 --> Output[/Вывод S/] Output --> End([Конеч]) </pre>
2	Определить наименования переменных в программе.	Имена переменных в программе соответствуют наименованиям в условии задачи.

3	Записать алгоритм с помощью операторов языка Бейсик (ввод-вывод, присваивание).	<pre> INPUT "Введите x"; x INPUT "Введите y"; y t = LOG(x*x + y*y) + COS(y^3)^2 S := SQR(t*t + 1) PRINT "Результат: S = ";S </pre>
---	---	--

Решите самостоятельно следующие задания:

Задание 1

Рассчитать t по формуле $t = \ln(u^2 + v^2)$, где $u = x + 5\cos(2y)$,

$$v = \frac{e^x + 2y + 3}{3x + \sqrt{x^2 + y^2}}.$$

Задание 2

Рассчитать U по формуле $U = V^2 + W$, где $W = \sqrt{e^x} + 2y + 3\operatorname{tg}x$,

$$V = \frac{3x + \ln^2 2y + 3}{e^{\frac{x}{2}} + \cos y^2}.$$

Примеры выполнения упражнений тренинга на умение 3

Задание

Даны действительные числа $a_1, b_1, c_1, a_2, b_2, c_2$. Выяснить, верно ли, что

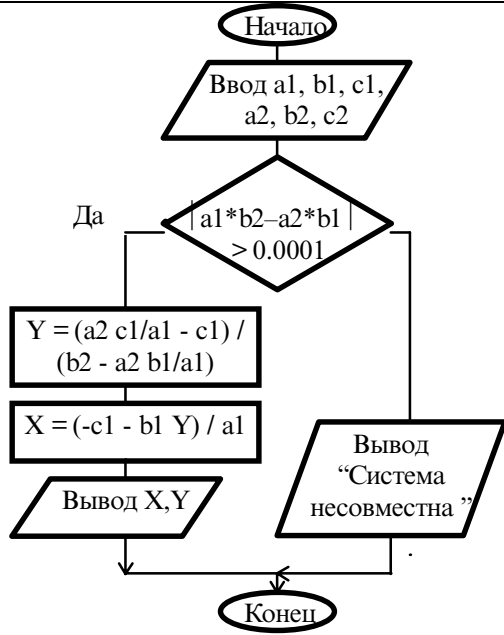
$$\frac{1}{2} a_1 b_2 - a_2 b_1 \frac{1}{2} > 0.0001,$$

и если верно, то найти решение системы линейных уравнений.

$$a_1 x + b_1 y + c_1 = 0$$

$$a_2 x + b_2 y + c_2 = 0$$

Решение

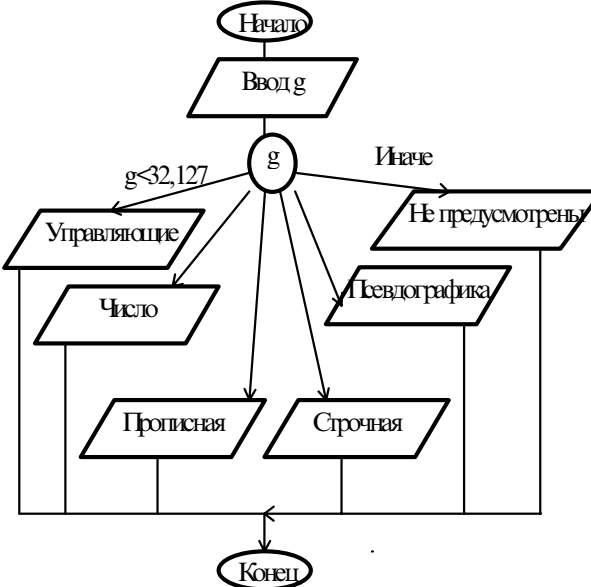
№	Алгоритм	Конкретное действие
1	Разработать алгоритм разветвляющейся структуры.	 <pre> graph TD Start([Начало]) --> Input[/Ввод a1, b1, c1, a2, b2, c2/] Input --> Decision{ a1*b2 - a2*b1 > 0.0001 } Decision -- Да --> YCalc[Y = (a2 c1/a1 - c1) / (b2 - a2 b1/a1)] YCalc --> XCalc[X = (-c1 - b1 Y) / a1] XCalc --> XYOut[/Вывод X, Y/] Decision -- Нет --> Inconsistent[/Вывод "Система несовместна"/] XYOut --> End([Конец]) Inconsistent --> End </pre>
2	Определить наименования переменных в программе.	Наименования переменных в программе соответствуют названиям этих переменных в условии задачи.
3	Записать алгоритм с помощью операторов языка Бейсик, используя операторы IF...THEN, IF...THEN...ELSE, SELECT ... END SELECT.	<pre> INPUT "Введите a1, b1, c1"; a1, b1, c1 INPUT "Введите a2, b2, c2"; a2, b2, c2 IF ABS(a1*b2-a2*b1) > 0.0001 THEN Y = (a2*c1/a1 - c1) / (b2 - a2*b1/a1) X = (-c1 - b1*Y) / a1 PRINT "Результат: X = ", X, "Y = ", Y ELSE PRINT "Система несовместна" ENDIF END </pre>

Задание 2

Написать программу, которая выдает сообщения о кодах ASCII.

Если код меньше 32 или равен 127, то выдается сообщение “Управляющие коды”. Если значение кода находится в интервале от 48 до 57 — “Число”; от 65 до 90 и от 128 до 159 — “Прописная буква”; от 97 до 122, от 160 до 175 и от 224 до 241 — “Строчная буква”; от 176 до 223 — “Символ псевдографики”.

Решение

№	Алгоритм	Конкретное действие
1	Разработать алгоритм разветвляющейся структуры.	 <pre> graph TD Start([Начало]) --> Input[/Ввод g/] Input --> G((g)) G -- "g < 32, 127" --> Control[/Управляющие/] G --> Number[/Число/] G --> Uppercase[/Прописная/] G --> Lowercase[/Строчная/] G -- Иначе --> Pseudo[/Псевдографика/] G -- Иначе --> NotProvided[/Не предусмотрено/] Control --> End([Конеч]) Number --> End Uppercase --> End Lowercase --> End Pseudo --> End NotProvided --> End </pre>
2	Определить наименования переменных в программе.	g — идентификатор вводимого кода.

№	Алгоритм	Конкретное действие
3	Записать алгоритм с помощью операторов языка Бейсик, используя операторы IF...THEN, IF...THEN...ELSE, SELECT ... END SELECT.	<pre> INPUT "Введите код"; g SELECT CASE g CASE IS < 32, 127 PRINT "Управляющие коды" CASE 48 TO 57 PRINT "Число" CASE 65 TO 90, 128 TO 159 PRINT "Прописная буква" CASE 97 TO 122, 160 TO 175, 224 TO 241 PRINT "Строчная буква" CASE 176 TO 223 PRINT "Символ псевдографики" CASE ELSE PRINT "Код в программе не указывается" END SELECT END </pre>

Решите самостоятельно следующие задания:

Задание 1

Даны положительные числа a , b , c , d . Выяснить, можно ли прямоугольник со сторонами a , b , уместить внутри прямоугольника со сторонами c , d так, чтобы каждая из сторон одного прямоугольника была параллельна или перпендикулярна каждой стороне второго прямоугольника.

Задание 2

Даны действительные числа X и Y . Меньшее из этих двух чисел заменить их полусуммой, а большее — их удвоенным произведением.

Задание 3

Составить программу определения названия животного по китайскому гороскопу, соответствующего введенному пользователем году. Указание: Проанализировать остаток от деления G на 12, где G — год. В зависимости от остатка вывести символ года (0 — Обезьяна, 1 — Петух, 2 — Собака, 3 — Кабан, 4 — Крыса, 5 — Бык, 6 — Тигр, 7 — Кролик, 8 — Дракон, 9 — Змея, 10 — Лошадь, 11 — Овца).

Примеры выполнения упражнений тренинга на умение 4

Задание 1

Ученикам 1– го класса назначается дополнительно стакан молока (200 мл), если их вес составляет меньше 30 кг. Определить, сколько литров молока потребуется ежедневно для одного класса, состоящего из N учеников. После взвешивания вес каждого ученика вводится в ЭВМ.

Решение

№	Алгоритм	Конкретное действие
1	Разработать алгоритм циклической структуры, используя структуры цикла типа "Пока" или типа "До".	<pre> graph TD Start([Начало]) --> InputN[/Ввод N/] InputN --> S0[S=0] S0 --> i1[i=1] i1 --> InputV[/Ввод V/] InputV --> V30{V < 30} V30 -- Да --> Splus[S=S+200] Splus --> iplus[i=i+1] V30 -- Нет --> iplus iplus --> iN{i > N} iN -- Да --> Output[/Вывод S/1000/] Output --> End([Конец]) iN -- Нет --> InputV </pre>

№	Алгоритм	Конкретное действие
2	Определить наименования переменных в программе.	<p>Vmin — минимальный вес школьника, которому не требуется дополнительное молоко;</p> <p>Milk — количество дополнительной порции молока;</p> <p>S — суммарное количество молока;</p> <p>N — количество учеников в классе;</p> <p>i — текущий номер ученика;</p> <p>V — вес i – го ученика.</p>
3	Записать алгоритм с помощью операторов языка Бейсик, используя операторы цикла WHILE ...WEND, FOR ... NEXT, DO ... LOOP.	<pre> Vmin = 30 : Milk = 200 INPUT "Количество учеников? "; N S = 0 : i = 1 DO PRINT "Вес ", i, "–го ученика" INPUT V IF V < Vmin THEN S = S + Milk i = i + 1 LOOP UNTIL i > N PRINT "Для класса необходимо "; S/1000; PRINT " литров молока" END </pre>

Задание 2

Составить программу, которая поможет Вам убедиться в том, что датчик случайных чисел в Бейсике работает по принципам теории вероятности. При большом количестве генерирования чисел в интервале от 0 до 100, количество четных и нечетных чисел будет приблизительно одинаковым.

Решение

№	Алгоритм	Конкретное действие
1	Разработать алгоритм циклической структуры, используя структуры цикла типа "Пока" или типа "До".	<pre> graph TD Start([Начало]) --> Input[/Ввод N/] Input --> Init[i=1; k1=0; k2=0] Init --> Cond1{i ≤ N} Cond1 -- Нет --> Output[/Вывод k1, k2/] Output --> End([Конеч]) Cond1 -- Да --> S[Random(100)] S --> Cond2{S mod 2 = 0} Cond2 -- Да --> K1[k1 = k1 + 1] Cond2 -- Нет --> K2[k2 = k2 + 1] K1 --> Inc[i = i + 1] K2 --> Inc Inc --> Cond1 </pre>
2	Определить наименования переменных в программе.	<p>N — общее количество сгенерированных чисел; S — случайное число в интервале от 0 до 99; k1 — количество четных чисел; k2 — количество нечетных чисел; i — текущий номер генерирования чисел.</p>
3	Записать алгоритм с помощью операторов языка Бейсик, используя операторы цикла WHILE ...WEND, FOR ...NEXT, DO ...LOOP.	<pre> INPUT "Введите N "; N i = 1 : k1 = 0 : k2 = 0 WHILE i <= N S = INT(RND*100) IF S MOD 2 = 0 THEN k1 = k1 + 1 ELSE k2 = k2 + 1 ENDIF i = i + 1 WEND PRINT "Четных ", k1, " Нечетных ", k2 END </pre>

Решите самостоятельно следующие задания:

Задание 1

Группа, состоящая из N студентов, сдает нормы ГТО по метанию гранаты. Вводя в цикле результат каждого студента, определить, сколько студентов выполнило норму ГТО.

Задание 2

Составить программу, подсчитывающую число удалений в каждой команде при игре в хоккей. После каждого удаления выводить на экран фамилию хоккеиста, время, на которое он удаляется с поля, и суммарное число удалений в каждой команде. После окончания игры выдать итоговое сообщение.

Задание 3

Составить программу помощника кассира в универсальном магазине. ЭВМ должна запрашивать цену товара и его количество, подсчитывать суммарную стоимость купленных товаров, запрашивать сумму денег, внесенных покупателем, и определять причитающуюся ему сдачу.

Примеры выполнения упражнений тренинга на умение 5

Задание 1

В области 10 районов. Заданы площади, засеваемые пшеницей, и средняя урожайность (ц/га) в каждом районе по области. Определить количество пшеницы, собранное в области, и среднюю урожайность по области.

Решение

№	Алгоритм	Конкретное действие
1	Разработать алгоритм решения задачи.	<pre> graph TD Start([Начало]) --> Init[S=0, P=0] Init --> LoopStart{i = 1, 10} LoopStart --> ReadPLi[/Ввод P_{Li}/] ReadPLi --> ReadURi[/Ввод U_{Ri}/] ReadURi --> CalcS[S = S + P_{Li}] CalcS --> CalcP[P = P + U_{Ri}] CalcP --> LoopEnd(()) LoopEnd --> LoopStart LoopEnd --> Output[/Вывод S, S/P/] Output --> End([Конеч]) </pre>

№	Алгоритм	Конкретное действие
2	Определить наименования переменных в программе.	S — количество пшеницы, собранной в области; P — общая площадь в области, засеваемая пшеницей; PLi — площадь, засеваемая пшеницей, в районе; URi — средняя урожайность в районе.
3	Составить описание массива.	DIM PL(10), UR(10)
4	Записать алгоритм с помощью операторов языка Бейсик.	DIM PL(10), UR(10) S = 0 : P = 0 FOR i = 1 TO 10 PRINT "Введите площадь "; i ; "–го района "; INPUT PL(i) PRINT "Введите урожайность "; i ; "–го района "; INPUT UR(i) S = S + PL(i) * UR(i) : P = P + PL(i) NEXT i PRINT "Кол–во пшеницы, собранной в области "; S PRINT "Средняя урожайность в области "; S / P END

Задание 2

Для заданной квадратной матрицы сформировать одномерный массив из ее диагональных элементов. Найти след матрицы, суммируя элементы одномерного массива.

Решение

Решите самостоятельно следующие задания:

№	Алгоритм	Конкретное действие
1	Разработать алгоритм решения задачи.	<pre> graph TD Start([Начало]) --> S0[S=0] S0 --> InputN[/Ввод N/] InputN --> i1N{i = 1, N} i1N --> j1N{j = 1, N} j1N --> InputAij[/Ввод Ai j/] InputAij --> Xi[Ai i] Xi --> Ssum[S = S + Xi] Ssum --> i1N j1N --> i1N i1N --> OutputS[/Вывод S/] OutputS --> End([Конец]) </pre>
2	Определить наименования переменных в программе.	<p>N — размер матрицы (количество строк и столбцов); S — результат расчета (след матрицы); $A_{i,j}$ — элемент матрицы A; X_i — одномерный массив из диагональных элементов матрицы A; i, j — индексы элементов массива.</p>

№	Алгоритм	Конкретное действие
3	Составить описание массива.	DIM A(N, N), X(N)
4	Записать алгоритм с помощью операторов языка Бейсик.	<pre> INPUT "Размер матрицы", N DIM A(N, N), X(N) S = 0 FOR i = 1 TO N FOR j = 1 TO N PRINT "A("; i ; ", "; j ; ")"; INPUT A(i , j) NEXT j NEXT i FOR i = 1 TO N X(i) = A(i , i) S = S + X(i) NEXT i PRINT "След матрицы "; S END </pre>

Задание 1

В соревнованиях по прыжкам в длину принимают участие 10 спортсменов. Считая заданным список фамилий спортсменов и их результаты в порядке стартовых номеров, получить итоговую таблицу, в которой содержатся фамилии и результаты в порядке занятых мест.

Задание 2

Задан массив размером 10. Сформировать два массива размером 5, включая в первый элементы массива с четными индексами, а во второй — с нечетными.

Задание 3

Таблица футбольного чемпионата задана квадратной матрицей порядка N, в которой все элементы, принадлежащие главной диагонали, равны нулю, а каждый элемент, не принадлежащий главной диагонали, равен 2, 1 или 0 (числу очков, набранных в игре: 2 — выигрыш, 1 — ничья, 0 — проигрыш). Найти число команд, имеющих больше побед, чем поражений.

Пример выполнения упражнения тренинга на умение 6

Задание

Три группы студентов в каждой не более 25 человек, в сессию сдавали по 5 экзаменов. Определить лучшую по среднему баллу группу.

Решение

№	Алгоритм	Конкретное действие
1	Разработать общую структуру программы.	<pre> graph TD Start([Начало]) --> ZInit{{z = 1, 3}} ZInit --> Read[/Ввод результатов сессии z-ой группы/] Read --> Calc[/Расчет среднего балла z-ой группы/] Calc --> ZNext ZNext --> ZInit ZNext --> Init[Max = S1, K = 1] Init --> IInit{{i = 2, 3}} IInit --> Compare{Si > Max} Compare -- Да --> Update[Max = Si K = i] Update --> INext Compare -- Нет --> INext INext --> IInit INext --> Output[/Вывод S/] Output --> End([Конеч]) </pre>

№	Алгоритм	Конкретное действие
2	Разработать алгоритм подпрограммы.	<p>Подпрограмма ввода результатов сессии</p> <pre> graph TD Start([Вход]) --> Input[/Ввод N/] Input --> LoopStart{{i = 1, N}} LoopStart --> LoopStartBody{{j = 1, 5}} LoopStartBody --> InputData[/Ввод Ai j/] InputData --> LoopStartBody LoopStartBody --> LoopStart LoopStart --> End([Возврат]) </pre> <p>Подпрограмма расчета среднего балла</p> <pre> graph TD Start([Вход]) --> Init[S=0] Init --> LoopStart{{i = 1, N}} LoopStart --> LoopStartBody{{j = 1, 5}} LoopStartBody --> Calc[S = S + Ai,j] Calc --> LoopStartBody LoopStartBody --> LoopStart LoopStart --> Avg[Sz = S / (5N)] Avg --> End([Возврат]) </pre>

№	Алгоритм	Конкретное действие
3	Записать алгоритм подпрограммы с помощью операторов языка Бейсик.	<p>Подпрограмма ввода результатов сессии</p> <pre> 1000 INPUT "Количество студентов", N FOR i = 1 TO N FOR j = 1 TO 5 PRINT "Оценка "; i ; " студента по "; j ; " экз."; INPUT A(i , j) NEXT j NEXT i RETURN </pre> <p>Подпрограмма расчета среднего балла</p> <pre> 2000 S = 0 FOR i = 1 TO N FOR j = 1 TO 5 S = S + A(i , j) NEXT j NEXT i S(z) = S / (N * 5) RETURN </pre>
4	Определить формальные и фактические параметры	В подпрограммах используются глобальные переменные и, следовательно, нет необходимости определять фактические и формальные параметры.
5	Записать алгоритм общей программы на языке Бейсик.	<pre> REM Программа определения лучшей группы DIM A (25 , 5), S(3) FOR z = 1 TO 3 GOSUB 1000 : GOSUB 2000 NEXT z Max = S(1) : K = 1 FOR i = 2 TO 3 IF S (i) > Max THEN Max = S (i) : K = i NEXT i PRINT "Лучшая " ; K ; " - я. Балл - " ; Max END </pre>

Решите самостоятельно следующее задание:

Задание

Стрелок производит по мишени 5 выстрелов. Вероятность попадания в мишень при каждом выстреле 0.6. Вычислить вероятности того, что стрелок не попадет в мишень ни разу; попадет один раз; 2 раза; 3 раза. Вычисление вероятности m попаданий при n выстрелах оформить в виде подпрограммы. Указания: Вероятность того, что стрелок при n выстрелах попадет m раз в мишень,

равна $C_n^m P^m q^{n-m}$, где $C = \frac{n!}{m!(n-m)!}$; $p=0.6$; $q = 1 - p = 0.4$.

ФАЙЛ МАТЕРИАЛОВ

Приложение 1

Название функции	Математическое обозначение	Имя	Примечание
Синус	$\sin x$	SIN (X)	В радианах
Косинус	$\cos x$	COS (X)	В радианах
Тангенс	$\operatorname{tg} x$	TAN (X)	В радианах
Арктангенс	$\operatorname{arctg} x$	ATN (X)	Вычисляемое знач. угла в интервале $(-\pi/2; +\pi/2)$
Показательная	e^x	EXP (X)	$x \geq 87$
Логарифм натуральный	$\ln x$	LOG (X)	$x > 0$
Корень квадратный	\sqrt{x}	SQR (X)	$x \geq 0$
Присвоение знака	знак x	SGN (X)	+1 для $x > 0$, 0 для $x = 0$, -1 для $x < 0$
Абсолютное значение	$ x $	ABS (X)	
Целочисленная функция		INT (X)	Опр-ся наибольшее целое, не превыш-ее знач. аргумента
Отбрасывание дробной части числа	–	FIX (X)	FIX (34.67) есть 34.0
Генерирование случ. числа в интервале [0,1]	–	RND [(X)]	Аргумент может быть опущен

Таблица операторов QBASIC

Оператор присваивания	[LET] v = a, v - имя переменной, принимающей новое значение; a - выражение.
Оператор ввода	DATA список констант READ список переменных через запятую
Оператор ввода	INPUT [;] [" Символьная константа" {; ,}] список переменных LINPUT - для ввода строковых констант с сохранением нач. и кон. пробелов.
Функция ввода/вывода, читающая символы с клавиатуры.	INKEY\$ Данная функция не ожидает ввода в отличие от оператора INPUT.
Оператор вывода	PRINT список вывода
Оператор вывода на печать	LPRINT список вывода
Оператор форматного вывода	PRINT USING список форматов; список вывода
Оператор позиционирования курсора на экране дисплея.	LOCATE x, y, где x и y - координаты положения курсора по вертикали и по горизонтали.
Оператор условной передачи управления	IF <условие> THEN <оператор1> ELSE <оператор2> [ENDIF]
Оператор множественного выбора	SELECT CASE выражение выбора [CASE список выражений 1] [операторы 1] [CASE список выражений 2] [операторы 2] ... [CASE ELSE [операторы n]] END SELECT
Оператор цикла с заранее заданным числом повторений	FOR счетчик = начало TO конец [STEP шаг] [тело цикла] [EXIT FOR] NEXT [счетчик [, счетчик ...]]
Оператор цикла с предусловием	WHILE условие [операторы] WEND
Оператор цикла с предусловием	DO [{WHILE UNTIL} логическое выражение] [операторы цикла] [EXIT DO] LOOP

Оператор цикла с постусловием	DO [операторы цикла] [EXIT DO] LOOP [WHILE UNTIL логическое выражение]
Оператор перехода	GOTO m, где m - метка строки, к которой осуществляется переход
Оператор множественного выбора	ON указатель перехода GOTO список меток, где указатель перехода - любое допустимое арифметическое выражение или переменная.
Оператор обращения к подпрограмме	GOSUB N, где N - метка строки, с которой начинается подпрограмма.
Последний оператор подпрограммы	RETURN
Операторы начала и конца процедуры SUB.	SUB имя [список параметров] [операторы] [EXIT SUB] [операторы] END SUB
Оператор вызова процедуры	CALL имя со списком аргументов
Оператор описания функции пользователя	DEF FNV (x1 [, x2, ...]) = арифметическое выражение
Операторы начала и конца процедуры FUNCTION	FUNCTION имя [(параметры)] [операторы] имя = выражение [операторы] END FUNCTION
Оператор описания массива	DIM список массивов

** В таблице перечислены операторы, которые разобраны в тематическом обзоре. Все остальные операторы можно посмотреть в пункте “Помощь” инструментальной оболочки QBASIC.*

ИНФОРМАТИКА
Углубленный курс
ЮНИТА 2

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Редактор Л.С. Лебедева
Оператор компьютерной верстки И.Ю. Маслова

Изд. лиц. ЛР № 071765 от 07.12.1998

Сдано в печать

НОУ “Современный Гуманитарный Институт”

Тираж

Заказ
