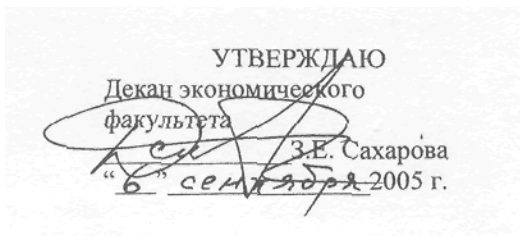


Министерство общего и профессионального образования РФ
Федеральное агентство по образованию

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ



Б.С. Лещинский

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ BASIC

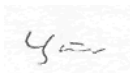
Учебное пособие

Томск 2005

РАССМОТРЕНО И УТВЕРЖДЕНО методической комиссией
экономического факультета

ПРОТОКОЛ № 1 от “ 6 ” _____ сентября 2004 г.

Председатель комиссии,
профессор



В.С. Цитленок

В учебном пособии приведены краткие сведения и даны рекомендации к программированию на языке Basic по курсу “Информатика”. Для некоторых типичных задач приведены соответствующие алгоритмы и программы.

Учебное пособие разработано для студентов экономического факультета дневной и вечерней формы обучения.

Составитель: Б.С. Лещинский

ВВЕДЕНИЕ

Функционирование компьютера, любых его компонент осуществляется только по определенным алгоритмам. **Алгоритм** – это *точно определенный* способ решения задачи в виде *конечной* (по времени) последовательности действий (операций). Для описания алгоритмов используют различные способы (графические, текстовые, формульные).

В силу технических особенностей компьютера алгоритм для него должен быть представлен вполне определенным образом – в виде *программы*, которая представляет собой последовательность команд для процессора. Каждая команда – это описание какой-либо инструкции, которую должен исполнить процессор. Эти команды, должны быть выражены *в машинном коде*, т.е. на языке, понятном процессору. Программист, конечно, может, затрачивая значительные усилия, понимать и составлять программы на машинном языке, но это очень сложно. Поэтому для их создания принят следующий подход: программист пишет текст алгоритма на языке, который ему понятен и удобен, а затем с помощью специальных программ (систем программирования) переводит его на машинный язык и превращает в удобный для процессора вид. Разговорный язык для этой цели не подходит в связи с нечеткостью и многозначностью, поэтому разрабатывают специальные языки, которые называют *алгоритмическими* или *языками программирования*.

Итак, **языки программирования** – это языки описания алгоритмов, которые нужны человеку, а не компьютеру. Они необходимы для облегчения создания программ, которые, в конечном итоге, должны быть представлены на одном и том же языке – языке команд для процессора. Поскольку человек имеет дело с текстом, написанном на языке программирования, то этот текст также называют *программой*. В настоящее время существует много алгоритмических языков, ориентированных на описание различных особенностей решаемых задач.

В целом, процесс разработки программы включает в себя три этапа:

- представление алгоритма в виде, удобном для уяснения логики решения задачи, для чего используют графические, текстовые, формульные средства;
- описание алгоритма средствами определенного алгоритмического языка;

– преобразование текста в готовую программу, которая будет исполняться процессором.

Первый этап выполняется вручную, а для других двух этапов используют специальные программные средства, которые называются *системами программирования*.

Язык программирования BASIC разработан в 1965 году. Название *BASIC* образовано из начальных букв английской фразы *Beginners All-purpose Symbolic Instruction Code* (многоцелевой язык символических инструкций для начинающих). Это наиболее простой язык программирования, предназначенный для решения вычислительных задач. Несмотря на свою простоту, он позволяет решать довольно большой круг экономических задач. На его примере можно достаточно быстро понять особенности и других языков программирования.

В настоящее время существует много разных вариантов, сохраняющих ядро BASIC и включающих различные дополнительные возможности. Мы рассмотрим один из таких вариантов под названием QBASIC.

1. ОСНОВЫ АЛГОРИТМИЗАЦИИ

1.1. Основные понятия

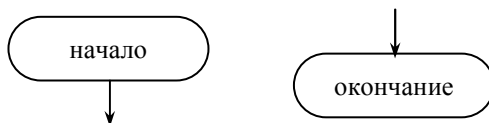
Для того, чтобы компьютер был способен решить какую-либо задачу, необходимо написать алгоритм.

Алгоритм – это *конечный* набор правил, позволяющий решать *любую* конкретную задачу из некоторого класса однотипных, при условии, что исходные данные для решения могут изменяться в заданных пределах. В алгоритме подробно описывается последовательность действий, необходимая для решения задачи. Составление такого пошагового описания процесса решения задачи называется **алгоритмизацией**. Существуют различные способы описания алгоритма, основными из которых являются словесно-формульный (пошаговый) и структурный (в виде блок-схемы). **Словесно-формульным** называется способ записи алгоритма на естественном языке, алгоритмическом (языке программирования) или псевдоязыке (сочетание элементов естественного и алгоритмического языка). **Структурный** способ – это компактная и наглядная форма записи в виде специальных графических знаков с указанием направленных связей между ними. Полученное таким способом описание называют блок-схемой (или структурной схемой).

Блок-схема (структурная схема) – это удобное для человека графическое изображение алгоритма в виде плоских геометрических фигур (их называют **блоками** или **вершинами**), соединенных направленными линиями (их называют **дугами**). Внутри каждого блока (вершины) записывается действие, которое следует выполнить, или условие, которое необходимо проверить. Направление дуг показывает последовательность выполнения действий.

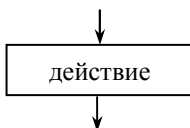
В блок-схеме присутствуют вершины разного типа, основными из которых являются:

– **вершины начала и окончания** (изображаются овалами). У вершины-начала нет входящих дуг, у нее есть лишь *одна* исходящая, направленная к вершине, с которой начинается алгоритм.



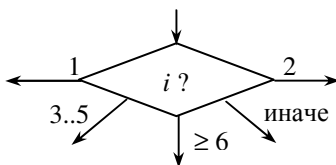
У вершины-окончания нет исходящих дуг и может быть несколько входящих;

– **вершины-действия** (изображаются прямоугольниками) соответствуют шагам, в которых выполняются действия по изменению значений, форм представления или расположения данных.

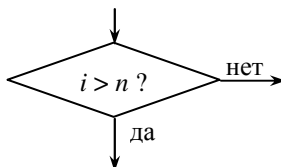


Каждая такая вершина может иметь несколько входящих дуг и только одну исходящую;

– **вершины-условия** (изображаются ромбами) соответствуют шагам, в которых проверяются условия. Такая вершина может иметь несколько входящих дуг и не менее двух исходящих, каждая из которых соответствует одному из результатов проверяемого условия и отмечается соответствующей меткой.

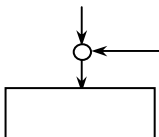


Если условием является выражение, предполагающее альтернативный выбор (*логическое выражение*),

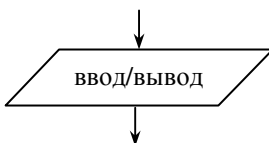


то одна из исходящих дуг соответствует отношению следования при условии, что проверяемое логическое выражение истинно. Такая дуга отмечается меткой «да». Другая исходящая дуга соответствует отношению следования при условии, что проверяемое логическое выражение ложно, и отмечается меткой «нет»;

– **вершины-узлы**, обозначающие объединение (*слияние*) нескольких входящих дуг (обозначаются точками или кругами небольшого размера).



Иногда вершины-действия делят на **операционные (вычислительные)** и **вершины ввода/вывода**. Тогда первые изображают прямоугольниками, а последние – параллелограммами



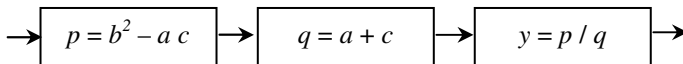
1.2. Основные алгоритмические структуры

Любой алгоритм представляет собой комбинацию трех элементарных алгоритмических структур: линейная, ветвящаяся, циклическая.

Линейная структура описывает процесс, в котором операции выполняются последовательно в порядке их описания. Вершины, отображающие эти действия, располагаются в линейной последовательности. Такие процессы имеют место, например, при вычислении арифметических выражений, когда имеются конкретные числовые данные и над ними выполняются соответствующие условию задачи действия. Например, вычисление

$$y = \frac{b^2 - ac}{a + c}$$

можно представить следующей линейной структурой



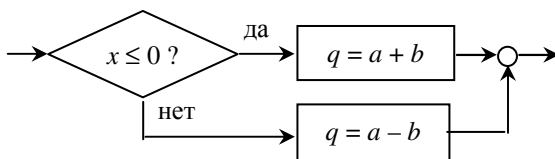
Обратите внимание на действия, указанные в блоках. Здесь использована очень важная операция **присваивания**. Запись
переменная = формула

означает, что необходимо провести вычисления, указанные справа от знака равенства, и полученное значение **присвоить** переменной, стоящей слева от этого знака. Здесь знак $=$ означает не проверку условия, а выполнение действия “обозначить именем значение” (“поместить значение в переменную”). Это действие называется **присваиванием** значения переменной. Например, операция увеличения i на 1 записывается $i = i + 1$ (читается “переменной i присвоить значение $i + 1$ ”).

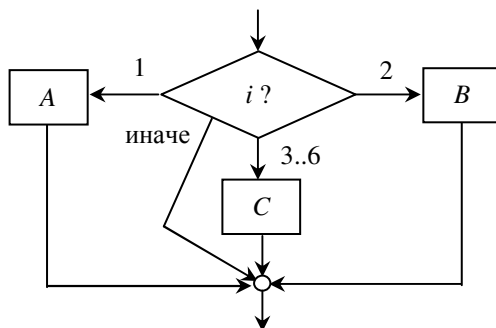
Ветвящаяся структура представляет процесс, для реализации которого предусмотрено несколько направлений (**ветвей**). Каждое отдельное направление является отдельной ветвью. Направление ветвления выбирается логической проверкой. Например, вычисление

$$q = \begin{cases} a + b, & \text{если } x \leq 0 \\ a - b, & \text{если } x > 0 \end{cases}$$

можем представить следующей ветвящейся структурой



Ветвящийся процесс, включающий в себя две ветви, называется **простым** (или **альтернативным**). Если процесс предполагает более двух ветвей, то он называется **сложным**. Например, структура



включает четыре ветви. Передвижение по ним осуществляется в зависимости от значения переменной i : если $i = 1$, производится пе-

переход к блоку A , при $i = 2$ – к блоку B , при значениях i от 3 до 6 – к блоку C , при других значениях – выход из процесса.

Заметим, что сложный ветвящийся процесс можно представить с помощью простых.

Циклической называется структура, описывающая процесс, содержащий цикл. *Цикл* – это последовательность многократно повторяющейся группы действий.

В описании цикла можно выделить следующие этапы:

- *подготовка (инициализация)* цикла включает действия по подготовке значений параметров, участвующих в действиях цикла;
- *выполнение (тело цикла)* включает действия, составляющие цикл;
- *модификация параметров* включает действия, изменяющие значения тех параметров, от которых зависит условие окончания цикла;
- *проверка условия* окончания цикла.

Цикл называется **детерминированным**, если число повторений тела цикла заранее (до начала цикла) известно или определено. Цикл называется **итерационным**, если число повторений тела цикла заранее не известно, а зависит от переменных, участвующих в вычислениях.

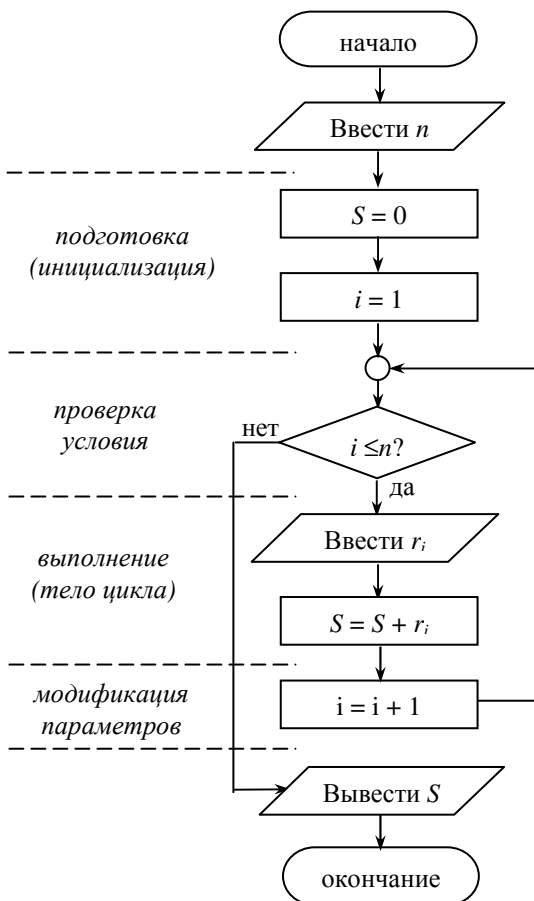
Проверка условия окончания может производиться как в конце цикла, так и в начале. В зависимости от его расположения различают *цикл с нижним окончанием* или *с постуловием* (условие проверяется после тела цикла) и *цикл с верхним окончанием* или *с предусловием* (условие проверяется перед телом цикла). Принципиальное отличие заключается в том, что в первом случае тело цикла обязательно выполняется по крайней мере один раз, а во втором – может не выполниться ни разу.

Для того, чтобы понять смысл циклического процесса, рассмотрим следующий пример. Пусть необходимо написать алгоритм вычисления суммы чисел, причем как количество чисел, так и сами числа заранее неизвестны и вводятся пользователем. Обозначим результат суммирования S , количество чисел – n , а каждое из них – r_i ($i = 1, \dots, n$), тогда

$$S = \sum_{i=1}^n r_i .$$

Если бы числа r_1, r_2, \dots, r_n были известны заранее, то для вычисления S достаточно было записать арифметическое выражение $S = r_1$

$+ r_2 + \dots + r_n$. В данном случае нам эти числа *заранее неизвестны*, поэтому поступим следующим образом. Сначала следует узнать у пользователя количество чисел и поместить в переменную n . После этого в S поместим нуль и n раз повторим одну и ту же последовательность действий: “ввести очередное число”, “прибавить число к значению S ”, “поместить результат сложения в переменную S ”. В результате этих действий в S будет накоплена сумма всех введенных пользователем чисел. Группа указанных действий составляет цикл, который выполняется n раз. Соответствующий алгоритм можно представить в виде следующей блок-схемы.



Здесь действия $S = 0$ и $i = 1$ составляют этап подготовки цикла. Тело цикла состоит из двух действий: ввод числа r_i и $S = S + r_i$. Модификация параметра выполняется действием $i = i + 1$. Такую переменную еще называют *переменной цикла*, т.к. ее значение показывает, сколько раз выполнено тело цикла. Кроме того, в данном случае это еще и номер очередного числа r_i . Цикл выполняется до тех пор, пока $i \leq n$. Заметим, что здесь имеем детерминированный цикл с верхним окончанием (с предусловием).

2. ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА BASIC

Общаясь между собой, мы формулируем мысли, пользуясь средствами того языка, который известен собеседникам. Знание языка означает, как минимум, знание алфавита (т.е. множества допустимых символов) и правил построения слов и предложений из элементов этого алфавита (т.е. синтаксиса). Но знание этих средств не гарантирует правильность понимания: нас могут неправильно понять, если мы недостаточно точно выразим свою мысль. Другими словами, от нас требуется еще и знание семантики, т.е. смысла элементов языка (элементов алфавита, слов и предложений). Таким образом, если мы хотим, чтобы собеседник нас правильно понял, мы должны синтаксически и семантически правильно изложить свою мысль на том языке, на котором происходит общение.

Язык программирования играет аналогичную роль, но предназначен не для общения между людьми, а для общения человека с ЭВМ. Как и другие языки, BASIC имеет алфавит, синтаксис и семантику. При этом, требования к написанию символов, слов и предложений в языке программирования значительно жестче, чем в любом разговорном языке. Это объясняется тем, что в отличие от человека, ЭВМ не обладает возможностями ассоциативного понимания смысла фразы. Другими словами, ЭВМ понимает только то, что написано в предложении, и не может догадаться о чем-либо, если это не выражено явно и однозначно. Например, если мы сначала укажем, что переменная h обозначает целое число, а затем этой переменной попытаемся присвоить текст “имя”, то эта попытка будет воспринята как ошибка (ведь перед этим мы объявили h целой, а не текстовой переменной). Как и любой язык программирования, BASIC построен таким образом, чтобы исключить многозначность понимания написанных на этом языке слов и предложений.

Предложение, написанное на языке программирования, называется **оператором**. Операторы играют роль инструкций по выполнению определенных действий. Выполнение программы сводится, по существу, к выполнению последовательности содержащихся в ней операторов. Они указывают действия присваивания значений переменным, изменения последовательности выполнения операторов, организации многократного выполнения какой-либо группы операторов и т.д.

Каждый оператор состоит из слов и специальных знаков. Какие-то из этих слов указывают, какое действие должно быть выполнено. Эти слова называются **ключевыми**, с них, как правило, начинается оператор. Другие указывают, с чем именно надо произвести действие и при каких условиях. Эти слова составляют так называемые **операнды**. Ключевые слова и операнды разделяются, как правило, пробелами.

Например, оператор

PRINT “Ответ”; 60

указывает, что необходимо вывести на экран в одной строке текст *Ответ* и число 60. Здесь PRINT является ключевым словом, означающим требование вывода на экран, текст *Ответ* и число 60 являются объектами вывода, т.е. операндами. Кавычки указывают, что *Ответ* – это текст, специальный знак точки с запятой говорит о том, что эти объекты при выводе должны быть расположены непосредственно друг за другом.

В операторе

$$d = a + b * s$$

записано, что необходимо сложить значение переменной *a* с произведением значений переменных *b* и *s*, после чего приравнять *d* этому результату (говорят: присвоить переменной *d* значение, равное полученному результату). Здесь знаки операций являются ключевыми словами, а имена переменных – операндами.

Программа, написанная на языке BASIC выполняется под управлением специальной программы, называемой интерпретатором (см. приложение). **Интерпретатор** последовательно читает операторы и обеспечивает их выполнение. Для этого он интерпретирует каждый оператор (определяет смысл данного оператора), переводит его на машинный язык и передает соответствующие инструкции (команды) процессору ЭВМ. Если интерпретатор не может выполнить

какой-либо оператор (например, в связи с синтаксической ошибкой), он сообщает пользователю об ошибке и прекращает исполнение программы.

Операторы могут быть указаны как в разных строках, так и в одной строке. В последнем случае между ними должен быть указан знак двоеточия. Строки в программе можно нумеровать. При этом, после номера должно быть не менее одного пробела. Номера строк могут быть любыми, но номер любой строки должен превышать номера предыдущих строк. При этом, можно нумеровать все строки, а можно – только некоторые из них.

Алфавит языка BASIC включает в себя буквы латинского и русского алфавитов, цифры от 0 до 9, знаки математических операций и специальные знаки (круглые скобки, знаки #, !, \$, %, @ и т.д.).

В данном пособии при описании операторов будем использовать следующие обозначения:

- в квадратных скобках ([...]) указываем ту часть оператора, которая является необязательной, т.е. при определенных условиях может не указываться в программе;

- в угловых скобках (<...>) указываем часть оператора, которая требует специального пояснения.

Эти скобки, разумеется, не указываются в программе, они нам нужны только для удобства записи в данном пособии.

3. ТИПЫ ДАННЫХ

Базовые данные, с которыми работает программа, относятся к одному из трех типов: числовые, строковые (текстовые) и логические.

Числовые данные могут быть целыми и вещественными (т.е. действительными), с ними можно производить математические операции. **Текстовые (строковые)** данные – это последовательности символов (тексты), для которых допустимы только текстовые преобразования (ввод, вывод, сравнение, перенос т.п.). **Логические** величины принимают только логические значения **True** (истина) и **False** (ложь). Они используются для проверки выполнения условий, при которых требуется выполнить различные действия.

Данные делятся на две группы: константы и переменные. **Константы** не изменяются на протяжении работы программы, а значения **переменных** могут изменяться (с сохранением типа). Константы

используются в программе в виде чисел (например, 35 и 659 – целые константы), последовательностей символов (например, “Фамилия” – текстовая константа), а переменные – в виде имен (идентификаторов).

Строковая константа в выражениях и операторах, как правило, должна быть ограничена слева и справа кавычками.

Вещественные константы могут быть указаны как в формате *с фиксированной точкой*, так и в формате *с плавающей точкой*. В первом случае целая часть отделяется от дробной точкой (например, 7.4 и 3290.75 – вещественные константы). Во втором случае число состоит из двух частей, мантиссы и порядка, разделенных латинской буквой *e* (для чисел с одинарной точностью) или *d* (для чисел с двойной точностью). **Мантисса** – это число в формате с фиксированной точкой, **порядок** – целое число, которое указывает степень, в которую надо возвести число 10, чтобы при умножении полученного числа на мантиссу получить вещественное число, которое имеется в виду. Например, число 234.47 в формате с плавающей точкой может быть записано как 2.3447e+2 (означает $2.3447 \cdot 10^2$). Здесь 2.3445 – мантисса, а 2 – порядок. Изменяя порядок (и, соответственно, мантиссу), можно указать то же самое число и другим образом: 23.447e+1, 0.23445e+3, 2344.7e–1. Формат с плавающей точкой используется для удобства ввода и вывода вещественных чисел большой длины, т.е. чисел, для записи которых требуется большое количество цифр.

Термины “*одинарная точность*” и “*двойная точность*” связаны с объемом памяти, который отводится для хранения вещественных чисел. Для вещественных чисел с двойной точностью этот объем больше, поэтому они обеспечивают большую точность вычислений.

Исключительно важным понятием языков программирования является понятие *переменной*. Оно позволяет использовать конкретные данные, обращаясь к ним с помощью **имен (идентификаторов)**. Такое имя представляет собой последовательность букв *латинского* алфавита и цифр, первый символ – обязательно буква (например, grand5). Оно *указывает на значение*. В процессе работы программы эти значения можно менять, обращаясь к ним по имени. Таким образом, переменная – это величина, которая может принимать различные значения (аналогично тому, как это понимается в

алгебре). Имена устанавливаются пользователем по своему усмотрению, но они не должны совпадать со служебными словами (ключевыми словами, именами функций и т.д.). Например, в выражении

$$f1 * 5.08 - f4 ^ (gray + 4)$$

$f1$, $f4$ и $gray$ – имена переменных,

5.08 и 4 – числовые константы.

Использование переменных в программе позволяет записать алгоритм решения задачи в общем виде и сделать программу пригодной для многократного использования с различными исходными данными.

Различают два вида переменных: простые и с индексами. **Простая переменная** указывает на одно значение, т.е. один элемент данных. В приведенном выше примере все переменные – простые.

Часто в программе необходимо описать выполнение действий с последовательностью однотипных данных (например, с целыми числами a_1, a_2, \dots, a_{100}). В этом случае удобнее использовать так называемый массив. **Массив** – это именованное упорядоченное множество однотипных данных (например, a – массив ста целых чисел). Элементы этого упорядоченного множества называются **элементами массива**. Элементы массива пронумерованы и обратиться к каждому из них можно по номеру (или по нескольким номерам – например, для элемента таблицы задается номер строки и столбца). Эти номера называются индексами. В языке QBASIC индексы указываются в круглых скобках после имени массива (например, $a(4)$). Подобное указание на элемент массива называется **переменной с индексами** или **индексированной переменной**.

Если элемент указывается одним индексом, то массив называется одномерным, если двумя индексами, то – двумерным, тремя – трехмерным и т.д. Индексы должны быть указаны в круглых скобках справа от имени массива через запятую (например, $g(4,7)$ указывает на элемент двумерного массива g).

В **одномерном** массиве индекс указывает на порядковый номер элемента. Например, переменная с индексом $d(5)$ указывает на пятый элемент среди всех значений, составляющих одномерный массив d . **Двумерный** массив, чаще всего, используется для представления таблиц (матриц).

Например, массивом w можно представить следующую таблицу

| | 1 | 2 | 3 |
|---|-----|-----|-----|
| 1 | 25 | 12 | 53 |
| 2 | 42 | 87 | 592 |
| 3 | 267 | 429 | 84 |
| 4 | 5 | 32 | 410 |

Смысл индексов определяет для себя сам программист, например, первый индекс может указывать на номер строки, а второй индекс – на номер столбца, на пересечении которых находится данный элемент. Тогда переменная с индексом $w(2,3)$ указывает на элемент двумерного массива w , соответствующий элементу таблицы, находящемуся на пересечении второй строки и третьего столбца, т.е. 592. Аналогично, число 429 находится на пересечении 3-ей строки и 2-го столбца и, следовательно, будет представлено элементом $w(3,2)$. Эту же таблицу можно представить и одномерным массивом, например, составляя его из строк последовательно друг за другом. Тогда это же число 429 будет восьмым элементом такого массива (например, $s(8)$, если s – имя массива). Очевидно, в подобных случаях удобнее пользоваться все-таки двумерным массивом.

Соответственно значениям, на которые указывают переменные, различают числовые (целые, вещественные) и текстовые переменные. Переменная целого типа указывает на целое число, вещественного типа – на вещественное число, а переменная текстового типа – на текстовое значение. Текстовая переменная в выражениях и операторах, как правило, обозначается знаком \$ после имени (например, $p\$ = \text{“текст”}$ означает присваивание текстовой переменной p текстового значения “текст”).

4. ВЫРАЖЕНИЯ

Самыми простыми являются операторы, которые определяют какие-либо вычисления. Выражения могут указывать на проведение арифметических действий (*арифметические выражения*), а также действий, направленных на проверку каких-либо условий (*логические выражения*). Для их записи используются знаки соответствующих операций: для арифметических – арифметические операции, для логических – операции сравнения и логические операции.

Например,

– *арифметическое* выражение $f + w$ указывает, что необходимо произвести действие сложения значений переменных f и w ;

– *логическое* выражение $h < 3 * p$ используется для проверки, является ли значение переменной h меньшим, чем произведение целого числа 3 и значения переменной p .

Арифметические операции:

- $+$, $-$ – знаки положительности и отрицательности числа;
- $+$ – сложение,
- $-$ – вычитание,
- $*$ – умножение,
- $/$ – деление,
- $^$ – возведение в степень.

Эти операции производятся над числами и результатом также является число.

Операции сравнения:

- $=$ – равно,
- $<>$ – не равно,
- $>$ – больше,
- $>=$ – больше либо равно,
- $<$ – меньше,
- $<=$ – меньше либо равно.

Эти операции предназначены для сравнения двух величин (например, в выражении $h >= k$ сравниваются значения двух переменных h и k). Результат принимает логическое значение **True** (истина), если условие выполняется, и значение **False** (ложь), если данное условие не выполняется. Подобные выражения называются *логическими*.

Например, результат логического выражения

$$15 < r * g ^ 2$$

равен **True** в том случае, если результат перемножения значения переменной r и квадрата значения переменной g больше 15. Если в процессе выполнения программы значения переменных r и g будут таковы, что результат этого произведения окажется меньшим или равным 15, то указанное условие не выполнится и результат выражения будет равен **False**.

Логические операции. Основными из них являются:

NOT – отрицание,

AND – логическое *И*,

OR – логическое *ИЛИ*.

Эти операции производятся над логическими величинами, т.е. величинами, принимающими значения **True** и **False**. Результатом является также логическая величина.

Операция **Not** выполняется над одной величиной и результат противоположен ее значению. В качестве такой величины может быть указано выражение с операциями сравнения и логическими операциями. Например,

$$\text{NOT } f > g(3)$$

принимает значение **False**, если значение переменной f больше третьего элемента массива g , и **True** в противном случае.

Операция **And** применяется к двум величинам. Результат равен **True** только в том случае, если оба значения этих величин равны **True**. Например, логическое выражение

$$i < 5 \text{ AND } j * k > m$$

принимает значение **True** только в том случае, если значение переменной i меньше целой числовой константы 5 и произведение значений переменных j и k больше значения переменной m .

Операция **Or** также, как и **And**, применяется к двум логическим величинам, но результат равен **False** только в том случае, если обе величины равны **False**. Другими словами, результат равен **True**, если хотя бы одна величина равна **True**. Например,

$$i < 5 \text{ OR } j * k > m$$

принимает значение **True**, если значение переменной i меньше 5 или результат произведения значений переменных j и k больше значения переменной m .

Приоритетность выполнения операций.

Операции в выражениях выполняются в соответствии с приоритетами. Среди арифметических операций самый высокий приоритет у операции возведения в степень (^), следующий приоритет – у операций умножения (*) и деления (/) и самый низкий приоритет у операций сложения (+) и вычитания (–). У операций сравнения равный приоритет. Логические операции выполняются в следующем порядке: сначала **Not**, затем **And** и последним **Or**.

В смешанных выражениях наиболее высокий приоритет у арифметических операций, затем выполняются операции сравнения и самый низкий приоритет у логических операций. Операции с одинаковым приоритетом выполняются слева направо. Порядок выполнения операций может быть изменен использованием круглых скобок, как это принято в математических выражениях.

Например, в выражении

$$\text{NOT } (b \geq 5.7 \text{ AND } (b > h - 1.4 \text{ OR } g / 4.5 < 7.3))$$

указан следующий порядок вычислений: сначала вычисляются значения выражений $h - 1.4$ и $g / 4.5$, затем выполняются операции сравнения в скобках, после этого – операция **Or**, затем – операция сравнения $b \geq 5.7$, операция **And** и последней – **Not**.

5. ОПЕРАТОРЫ ОЧИСТКИ ЭКРАНА, ПРИСВАИВАНИЯ, ОФОРМЛЕНИЯ КОММЕНТАРИЕВ

Очистка экрана. Оператор очистки экрана

CLS

приведет к тому, что во время выполнения программы интерпретатор (как только прочитает этот оператор) даст команду процессору удалить с экрана всю информацию и переместить курсор в левую верхнюю позицию.

Оператор присваивания.

<имя> = <выражение>

Здесь <имя> – это идентификатор переменной (простой или с индексами), <выражение> – арифметическое, текстовое или логиче-

ское выражение, которое, в частности, может быть переменной или константой.

Напомним (см. п. 1.2), что запись

$\langle \text{переменная} \rangle = \langle \text{формула} \rangle$

означает действие присваивания, т.е. указывает, что необходимо произвести вычисления, указанные справа от знака равенства, и полученное значение *присвоить* переменной, стоящей слева от этого знака. Например,

$$d = 34.5$$

означает, что в переменную d надо “поместить” константу 34.5 (переменной d присвоить значение 34.5);

$$f(2) = d + r(5) * y ^ 3$$

означает, что необходимо возвести значение переменной y в третью степень, умножить результат на значение пятого элемента массива r , сложить полученный результат со значением переменной d и результат этого действия поместить в массив f в качестве второго элемента. Другими словами, второму элементу массива f присвоить значение, полученное в результате выполнения арифметических действий, указанных справа от знака равенства.

При записи выражений следует иметь в виду, что запрещено присваивать числовым переменным символьные или логические значения и наоборот, а также логическим переменным символьные значения и наоборот. Кроме этого, если числовой переменной присваивается числовое значение другого типа, то это значение преобразовывается в тип, указанный для переменной, стоящей слева от знака равенства. Например, если f – переменная вещественного типа, то после выполнения оператора

$$f = 34 + 12$$

переменная f будет указывать на вещественное число 46.0, а не целое 46.

Оформление комментариев.

Комментарии используются для удобства просмотра текста программы и не выполняются в качестве действий. Они могут быть указаны как в отдельной строке, так и в одной строке с операторами *после них*. Перед комментариями должен быть указан знак апострофа (‘). Если комментарии указаны в отдельной строке, то вместо апострофа можно указать ключевое слово **Rem**.

6. ВВОД И ВЫВОД ДАННЫХ

6.1. Оператор позиционирования курсора на экране

LOCATE [<арг.1>] [,<арг.2>] [,<арг.3>]

Этот оператор устанавливает текстовый курсор в определенное место экрана и часто используется для удобства ввода или вывода данных. Величины <арг.1>, <арг.2> и <арг.3> должны быть целыми (константами, переменными или арифметическими выражениями, результатами которых являются целые числа). При этом, <арг.1> – порядковый номер строки (с отсчетом сверху вниз), <арг.2> указывает позицию в строке (слева направо), а <арг.3> определяет состояние курсора на экране (“0” – для погашения, “не 0” – для высвечивания).

6.2. Вывод данных на экран

PRINT [<список>]

Здесь ключевое слово *Print* означает “вывести на экран”. В списке указывается, что и в каком виде необходимо вывести. В нем можно указать одно или несколько выражений любого типа, разделенных запятыми или точкой с запятой. В частности, в списке могут быть указаны константы и переменные. Символьные константы должны быть ограничены с двух сторон кавычками. Символьные переменные должны быть отмечены знаком \$ после имени (без пробела).

Если элементы в списке разделены запятой, то при выводе элементу, указанному после запятой, отводится зона, равная 14 позициям (или более, если не хватает). Разделитель в виде точки с запятой означает, что данные выводятся на экран непосредственно один за другим. Следует иметь в виду, что если после списка указать символ точки с запятой, то после вывода текстовый курсор останется на месте и не будет автоматически перенесен в начало следующей строки.

Например, оператор

PRINT “Ответ: ”; 60

указывает, что необходимо вывести

Ответ: 60

Применение оператора

PRINT “Элемент массива г(“; 3; “) =”, 628
приведет к тому, что на экран будет выведено
Элемент массива г(3) = 628

6.3. Вывод данных на экран с позиционированием

PRINT TAB(<арг.1>); <выр.1> [;TAB(<арг.2>);<выр.2>]...

Этот оператор выводит данные, определяемые выражениями <выр.1>, <выр.2> и т.д., начиная с соответствующих позиций <арг.1>, <арг.2> и т.д., указанных в круглых скобках. Позиции могут быть заданы константами, переменными и арифметическими выражениями. Такой оператор удобен для вывода данных в табличной форме.

Например, если указать оператор

PRINT TAB(5); “Строка чисел”; TAB(20); 23.6; TAB(30); 67
то на экран будет выведено
Строка чисел 23.6 67

При этом, текст *Строка чисел* будет выведен, начиная с пятой позиции, число 23.6 – начиная с 20-ой, а число 67 – с 30-й позиции.

6.4. Диалоговый ввод данных

INPUT [“<текст>“;] <список>

Этот оператор используется для ввода данных с клавиатуры. Ключевое слово **Input** означает “ввести”. Действие оператора заключается в выводе на экран указанного текста, ожидании ввода данных и автоматическом присваивании введенных значений переменным, указанным в списке.

<текст> – это текстовая подсказка, которая должна быть выведена на экран. Эта подсказка полезна, если надо пояснить пользователю, что именно он должен ввести с клавиатуры. При выводе этого текста после него автоматически будет следовать знак вопроса. Если текстовую подсказку не указывать, на экран будет выведен только знак вопроса. *Вместо точки с запятой можно указать запятую.* В этом случае знак вопроса после текста выводиться не будет.

В списке указываются идентификаторы переменных (разделенные запятыми), которым должны быть автоматически присвоены значения, введенные с клавиатуры.

От пользователя требуется, увидев на экране текст (знак вопроса), ввести с клавиатуры константы, разделяя их запятыми. При этом, количество и типы констант должны строго соответствовать количеству и типам переменных, имена которых указаны в списке. Если такого соответствия при вводе не будет, интерпретатор зафиксирует ошибку и потребует повторить ввод. Если ввод произведен правильно, интерпретатор полученные значения присвоит переменным, указанным в списке.

Например, рассмотрим программу вычисления суммы любых четырех целых чисел, введенных пользователем с клавиатуры.

CLS

INPUT "Четыре целых числа"; a, b, c, d

PRINT "Сумма этих чисел равна "; a + b + c + d

В соответствии с этой программой сначала будет произведена очистка экрана и курсор помещен в верхнюю левую позицию. Затем на экране появится текст

Четыре целых числа?

От пользователя требуется ввести числа, т.е. набрать четыре целых числа, разделяя их запятыми (например, 5, 6, 8, 10), и нажать клавишу <Enter>. После этого на экран будет выведен результат, например,

Сумма этих чисел равна 29

6.5. Ввод данных из программы

Данные, необходимые для выполнения программы, можно указать непосредственно в тексте программы. Для этого используются три оператора, один из которых (*Data*) указывает эти данные, другой (*Read*) обеспечивает присваивание переменным указанных значений (говорят "считывает данные из программы"), а третий (*Restore*) применяется, если требуется присвоить эти же значения другим переменным.

Оператор, который указывает константы, имеет следующий вид:

DATA <список>

Здесь <список> – это константы (числовые или текстовые), разделенные запятыми (здесь текстовые константы необязательно окружать кавычками). Этих операторов может быть несколько, причем в любом месте программы.

Оператор

READ <список>

обеспечивает присвоение переменным, указанным в списке, значений, указанных в списках констант операторов **Data**. <список> – это имена переменных, разделенные запятыми. После имени переменной текстового типа обязательно следует указывать знак \$. Этот оператор еще называют оператором ввода данных (считывания данных) из программы. При повторном использовании он продолжает считывание данных из списков констант операторов **Data**. Например, последовательность операторов

```
DATA 4, 6, 8, 10
DATA 12, Сумма
READ a, b, c
...
READ d, e, f$
```

приведет к тому, что переменным *a*, *b* и *c* будут присвоены значения 4, 6, 8, переменным *d* и *e* – значения 10 и 12, а переменной *f* – текстовое значение *Сумма*.

Оператор

RESTORE [<номер строки>]

возвращает неотображаемый указатель чтения данных (для оператора **Read**) к первому элементу списка оператора **Data**.

<номер строки> – это номер строки, в которой указан оператор **Data**; он может быть опущен, если оператор **Data** – единственный в программе.

Например, последовательность операторов

DATA 4, 6, 8, 10, 12

READ a, b, c

...

RESTORE

READ d, e

приведет к тому, что переменным a , b и c будут присвоены значения 4, 6, 8, а переменным d и e – значения 4 и 6.

Рассмотрим пример, демонстрирующий использование операторов **Print**, **Input**, **Data** и **Read**. Пусть требуется решить задачу определения ежегодных платежей для погашения ссуды аннуитентными (равными) платежами.

Размер ежегодного платежа (P) определяется по формуле

$$P = \frac{m \cdot r^t \cdot (r - 1)}{r^t - 1}.$$

Здесь

m – размер ссуды;

t – количество лет, на которые взята ссуда;

r – величина, вычисляемая по формуле

$$r = 1 + \frac{d}{100}, \text{ где}$$

d – размер процента по ссуде.

Напишем программу, которая для любого человека вычисляет размер платежа при различных значениях размера ссуды, количества лет и размера процента.

Перед написанием любой программы необходимо четко определить входную информацию, способы ее обработки, выходную информацию и ту форму, в которой требуется ее выводить.

Прежде всего, отметим, что входную информацию для этой задачи можно разделить на постоянную и изменяющуюся. К постоянной отнесем информацию, необходимую для оформления вывода результатов. Эту информацию удобнее ввести в программу с помощью операторов **Data** и **Read**. К изменяющейся относятся данные о размере ссуды (m), количестве лет (t), размере процента (d) и фамилии клиента. Для ввода этих данных лучше использовать оператор **Input**.

В процессе решения задачи нам необходимо вычислить величину r и размер платежа P .

Результаты вычислений оформим в следующем виде.

Ф.И.О. _____

| Размер ссуды | Кол-во лет | Размер % | Размер платежа |
|--------------|------------|----------|----------------|
| ... | ... | ... | ... |

Таким образом, выходная информация включает данные, необходимые для формирования таблицы, данные о размере ссуды, количестве лет, размере процента и ежегодных платежах, а также фамилия клиента. Для удобства чтения программы используем комментарии.

```

REM    Программа вычисления ежегодных платежей
CLS                                         ' Очистка экрана
' Указание данных для оформления выходной таблицы
DATA _____
DATA Ф.И.О., Размер ссуды, Кол-во лет, Размер %, Размер платежа
' Ввод исходных данных
READ s$, a$, b$, c$, e$, g$
PRINT a$; : INPUT f$                      ' Ф.И.О. клиента
PRINT b$; : INPUT m                       ' размер ссуды
PRINT c$; : INPUT t                       ' количество лет
PRINT e$; : INPUT d                       ' размер % по ссуде
' Вычисление размера платежа
r = 1 + d / 100
P = m * r ^ t * ( r - 1 ) / ( r ^ t - 1 )
' Вывод результатов
CLS
PRINT TAB(20); a$; ";"; TAB(30); f$      ' Ф.И.О. клиента
PRINT s$ : PRINT b$, c$, e$, g$ : PRINT s$ ' названия столбцов
PRINT m, t, d, P                          ' данные
PRINT s$                                  ' окончание таблицы

```

Заметим, что в этой программе в некоторых строках указано несколько операторов, разделенных двоеточием. Также отметим, что некоторые операторы *Print* заканчиваются точкой с запятой. На-

помним, такое окончание указывает, что после вывода текстовый курсор не должен передвигаться на другую строку.

Результаты выполнения программы, например, могут быть следующими:

Ф.И.О. ? Иванов П.С.
Размер ссуды ? 300
Кол-во лет ? 7
Размер % ? 14

Ф.И.О.: Иванов П.С.

| Размер ссуды | Кол-во лет | Размер % | Размер платежа |
|--------------|------------|----------|----------------|
| 300 | 7 | 14 | 69.95771 |

7. ОПЕРАТОР ОПИСАНИЯ (ОБЪЯВЛЕНИЯ) ПЕРЕМЕННЫХ

Как мы уже знаем, переменные могут простыми или с индексами, а также разного типа (текстовые, целые, вещественные с одинарной или двойной точностью). Для того, чтобы интерпретатор правильно понимал имя используемой переменной (простая, с индексами, тип), необходимо с помощью специального оператора сообщить ему соответствующую информацию (говорят “описать переменную”). Например, можно указать, что под именем r следует понимать простую переменную целого типа, а под именем g – одномерный массив не более 26 вещественных значений с одинарной точностью.

Если таких явных указаний не сделать, интерпретатор берет на себя ответственность за понимание используемых имен переменных (говорят “по умолчанию”). При этом, тип каждой переменной устанавливается автоматически в момент использования в программе в зависимости от того, какого типа значение присваивается переменной.

Например, если переменная i использована в операторе присваивания $i = 1$, то с этого момента под i будет пониматься целая простая переменная, а если – в операторе $i = 1.0$, то – простая переменная вещественного типа. То есть тип переменной здесь устанавливается автоматически в зависимости от типа присваиваемой константы.

При этом, вещественный тип по умолчанию устанавливается с двойной точностью, а размерность массивов – 10 для каждого из индексов.

Часто при написании программ ориентируются именно на эти правила установки типов переменных по умолчанию. Однако, это не всегда удобно (например, если необходим массив большего размера, чем 10). Для явного указания типов переменных используется следующий оператор:

DIM <перем.1> [As <тип>], <перем.2> [As <тип>], ...

Здесь <перем.1>, <перем.2> и т.д. – имена переменных, *Dim* и *As* – ключевые слова. Если указывается массив, то после его имени обязательно необходимо указать в круглых скобках через запятую максимальные значения индексов, которые допустимо использовать в программе. Такие значения могут быть заданы целыми константами или целыми переменными (например, $d(k, 24)$). Если в качестве индекса указана переменная, то ее значение должно быть определено до этого (присваиванием или вводом).

Основные типы переменной задаются следующими ключевыми словами:

| | |
|----------------|--|
| Integer | – целое обычное (целое число от –32768 до +32767); |
| Long | – длинное целое (целое число от –2147483648 до +2147483647); |
| Single | – вещественное с одинарной точностью; |
| Double | – вещественное с двойной точностью; |
| String | – текстовый (строковый). |

Оператор *Dim* может быть указан в любом месте программы, но обязательно перед первым использованием описанных переменных.

Следует иметь в виду важную особенность использования *Dim*: типы переменных, объявленные этим оператором, остаются неизменными во всем последующем тексте программы. В частности, это делает ненужным указание знака \$ после имени текстовой переменной в операторах присваивания (см. п. 3) и *Read* (см. п. 6.5) .

Например, для объявления целой переменной *d*, вещественного с одинарной точностью двумерного массива *g* и текстовой переменной *t* можно написать следующий оператор:

DIM d As Integer, g(12,40) As Single, t As String

В результате, после него допустимо для двумерного массива g указывать переменную с индексами, превышающим 10 (например, $g(11,30)$), а в операторе присваивания $t = \text{“текст”}$ нет необходимости указывать знак $\$$ после t (см. п. 3), поскольку тип этой переменной уже объявлен.

8. ОПЕРАТОРЫ ОРГАНИЗАЦИИ ЦИКЛА

Напомним (см. п. 1.2), что циклом называется процесс многократного повторения одной и той же последовательности действий. Указание группы операторов для выполнения цикла производится с помощью разных операторов.

8.1. Цикл с предусловием (операторы *While* – *Wend*)

Операторы *While* – *Wend* выполняют указанную между ними последовательность операторов, пока условие, записанное после ключевого слова *While*, имеет значение *True*.

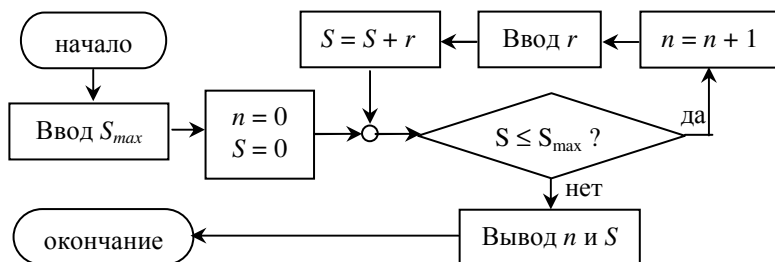
```

WHILE <условие>
  <оператор 1>
  <оператор 2>
  .....
  <оператор N>
WEND

```

Заметим, что эти операторы реализуют цикл с предусловием (см. п. 1.2), т.е. тело цикла может не выполниться ни разу.

Например, алгоритм вычисления суммы вводимых пользователем неотрицательных чисел до тех пор, пока она не превысит заранее заданного предела.



может быть реализован следующей программой

```
CLS
DIM n As Integer, r As Single, S As Single, Smax As Single
INPUT "Размер предела"; Smax
n = 0
S = 0
WHILE S <= Smax
    n = n + 1
    PRINT n; "-е число";
    INPUT r
    S = S + r
WEND
PRINT "Сумма введенных"; n; "чисел равна "; S
```

Возвращаясь к задаче вычисления суммы заранее неизвестных чисел (см. п. 1.2), соответствующую программу с помощью операторов **While** – **Wend** можно написать в следующем виде.

```
CLS
DIM n As Integer, S As Single
INPUT "Сколько чисел"; n
DIM r(n) As Single
S = 0
i = 1
WHILE i <= n
    PRINT i; "-е число";
    INPUT r(i)
    S = S + r(i)
    i = i + 1
WEND
PRINT "Сумма введенных"; n; "чисел равна "; S
```

Если требуется вычислить сумму трех чисел 31.8, 20 и 18.2, то в процессе выполнения этой программы на экране будет виден следующий диалог с пользователем.

```
Сколько чисел ? 3
1-е число ? 31.8
2-е число ? 20.
3-е число ? 18.2
```

Сумма введенных 3 чисел равна 70

8.2. Цикл со счетчиком (операторы *For* – *Next*)

Операторы *For* – *Next* повторяют выполнение указанной между ними последовательности операторов заданное число раз.

```
FOR <имя> = <арг.1> TO <арг.2> [STEP <арг.3>]  
  <оператор 1>  
  <оператор 2>  
  .....  
  <оператор N>  
NEXT [<список>]
```

Здесь *For*, *To*, *Step* – ключевые слова. *For* требуется для определения имени переменной цикла (<имя>) и ее начального значения (<арг.1>), т.е. значения, при котором цикл выполняется в первый раз. *To* определяет значение (<арг.2>) переменной цикла, при котором цикл выполняется в последний раз, а ключевое слово *Step* – шаг изменения значений переменной цикла (<арг.3>), т.е. то число, на которое должно изменяться значение переменной цикла после очередного выполнения группы операторов, составляющих цикл.

Величины <арг.1>, <арг.2> и <арг.3> могут быть заданы арифметическими выражениями (чаще всего, это константы или переменные).

Оператор *For* указывает, что за ним следует группа операторов, которая должна повторяться до тех пор, пока значение переменной цикла не превысит значение <арг.2>. Начинается цикл со значения переменной цикла, равного значению <арг.1>. После каждого повторения цикла значение этой переменной нарастается на величину, равную значению <арг.3>. Если *Step* (вместе с <арг.3>) не указан, то интерпретатор считает его равным единице.

Следует иметь в виду, что оператор *For*:

- проверяет достижение переменной цикла ее предельного значения и
- производит выход из цикла, как только значение этой переменной превысит значение, указанное после ключевого слова *To*.

Это означает, что цикл может ни разу не выполниться, если начальное значение переменной цикла (<арг.1>) превышает предель-

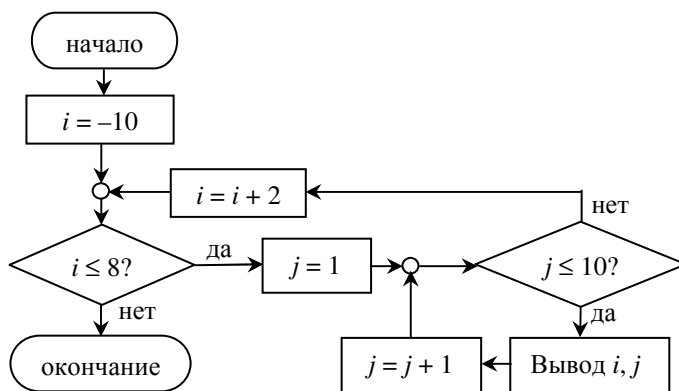
ное значение (<арг.2>). Другими словами, операторы **For – Next** так же, как и **While – Wend**, реализуют цикл с предусловием.

Оператор **Next** указывается последним в цикле. Он выполняет приращение значения переменной цикла на величину, указанную в <арг.3>. Допускается применение нескольких циклов, вложенных один в другой. В этом случае после **Next** указывается <список>, в котором перечисляются переменные циклов, вложенных друг в друга. Эти переменные должны быть разделены запятыми. Переменные в этом списке указываются в порядке от последнего к первому по вложенности циклов. Если <список> состоит из одного элемента, то его можно не указывать. Можно для каждого из вложенных циклов указывать свой оператор **Next**.

Например, программы

| | | |
|---|---|---|
| <pre>FOR i = -10 TO 8 STEP 2 FOR j = 1 TO 10 PRINT i, j NEXT j, i</pre> | и | <pre>FOR i = -10 TO 8 STEP 2 FOR j = 1 TO 10 PRINT i, j NEXT NEXT</pre> |
|---|---|---|

реализуют один и тот же алгоритм



Основное отличие между **While – Wend** и **For – Next** заключается в простоте описания действий, связанных с переменной цикла. Если применение **While – Wend** предполагает явную запись операторов

присваивания начального значения этой переменной (например, $i=1$), изменение ее значения (например, $i = i + 1$) и условия выхода из цикла (например, $i \leq n$), то применение **For – Next** этого не требует. Достаточно лишь указать <арг.1>, <арг.2> и <арг.3> в операторе **For**, что упрощает текст программы.

Например, программу вычисления суммы заранее неизвестных чисел (см. п. 1.2 и п. 8.1) с помощью операторов **For – Next** можно написать достаточно просто в следующем виде.

```
CLS
DIM n As Integer, S As Single
INPUT "Сколько чисел"; n
DIM r(n) As Single
S = 0
FOR i = 1 TO n
    PRINT i; "-е число";
    INPUT r(i)
    S = S + r(i)
NEXT
PRINT "Сумма введенных"; n; "чисел равна "; S
```

8.3. Цикл с постусловием (операторы **Do – Loop Until**)

Заметим, что в ранее рассмотренных примерах во время выполнения программы могли возникать ошибочные ситуации, связанные с неверными исходными данными. Например, в последней задаче п. 8.2 предполагается, что количество чисел должно быть, по крайней мере, неотрицательным. Однако, человеку свойственно ошибаться и желательно в программах предусматривать проверку корректности вводимых данных. Один из простых способов реализации такого контроля основан на использовании операторов **Do – Loop Until**.

Операторы **Do – Loop Until** выполняют указанную между ними последовательность операторов до тех пор, пока условие, записанное после ключевых слов **Loop Until**, не примет значение **True**.

DO

<оператор 1>

<оператор 2>

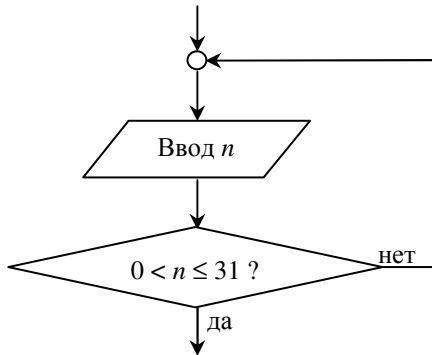
.....

<оператор N>

LOOP UNTIL <условие>

В отличие от *While – Wend* в данном случае условие проверяется *после* выполнения тела цикла, поэтому указанная последовательность обязательно выполнится по крайней мере один раз. Другими словами, эти операторы реализуют цикл с постусловием.

Например, если необходимо ввести число в пределах от единицы до 31, то добиться от пользователя правильного ввода можно следующим образом:



Как видно из блок-схемы, алгоритм не продолжится до тех пор, пока не будет введено число, удовлетворяющее проверяемому условию. Для реализации этого достаточно написать

...

DO

INPUT "Введите число: ", n

LOOP UNTIL 0 < n AND n <= 31

...

9. ОПЕРАТОРЫ ПЕРЕХОДА (ВЕТВЛЕНИЯ)

Операторы перехода служат для описания ветвлений, т.е. принудительного изменения последовательности выполнения операторов в программе.

9.1. Оператор безусловного перехода

GOTO <номер строки>

Этот оператор обеспечивает переход на строку с указанным номером.

9.2. Оператор условного перехода (условный оператор)

Этот оператор обеспечивает *простое (альтернативное)* ветвление, когда в качестве проверяемого условия записывается *логическое* выражение и в зависимости от истинности или ложности результата выбирается одна из указанных групп операторов.

```
IF <условие> THEN  
    <группа 1>  
    [ELSE  
        <группа 2>]  
END IF
```

Здесь

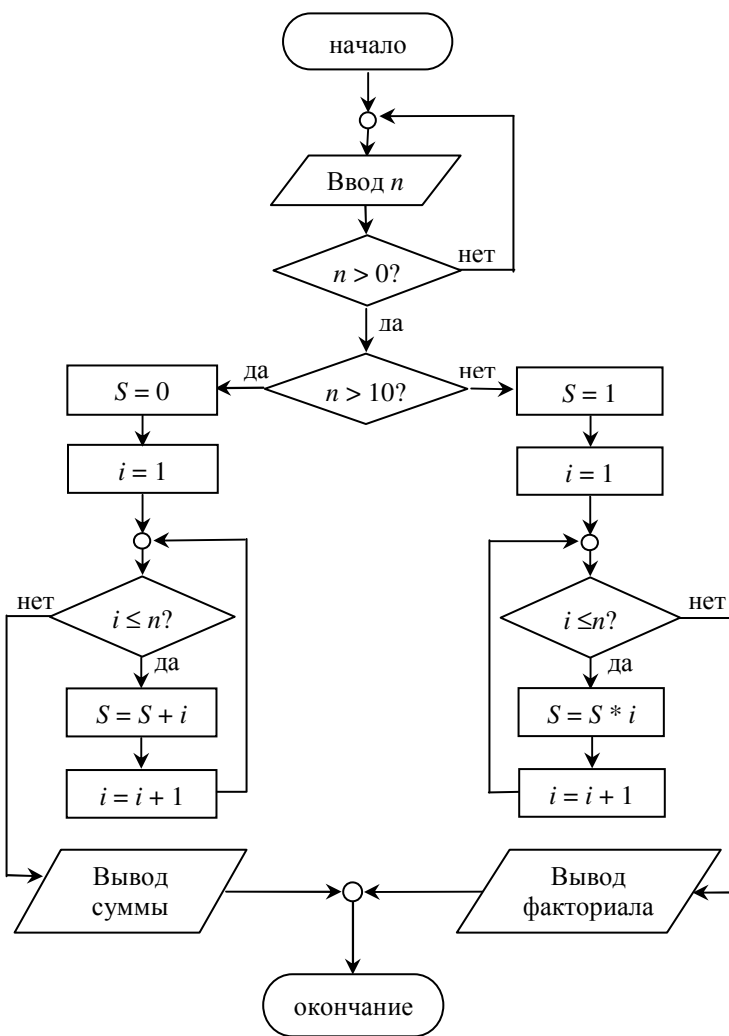
<условие> – логическое выражение;

<группа 1> – группа операторов, которые выполняются, если <условие> выполняется, т.е. результат выражения равен **True**;

<группа 2> – группа операторов, которые выполняются, если <условие> не выполняется, т.е. результат выражения равен **False**;

End If – оператор окончания условного оператора.

Например, рассмотрим задачу вычисления суммы от 1 до числа, указанного пользователем, если это число больше 10, и факториала этого числа в противном случае. Алгоритм решения можно представить следующей блок-схемой.



Для ее реализации можно использовать следующий программный текст:

```

DIM n As Integer, S As Double      ' типы переменных
DO
  CLS
  INPUT "Введите положительное число:", n    ' ввод n
LOOP UNTIL n > 0
DIM a(n) As Integer                ' тип массива
IF n > 10 THEN
  S = 0
  FOR i = 1 TO n                    ' вычисление суммы
    S = S + i
  NEXT
  PRINT "Сумма чисел от 1 до"; n; "равна"; S
ELSE
  S = 1
  FOR i = 1 TO n                    ' вычисление факториала
    S = S * i
  NEXT
  PRINT "Факториал числа"; n; "равен"; S
END IF

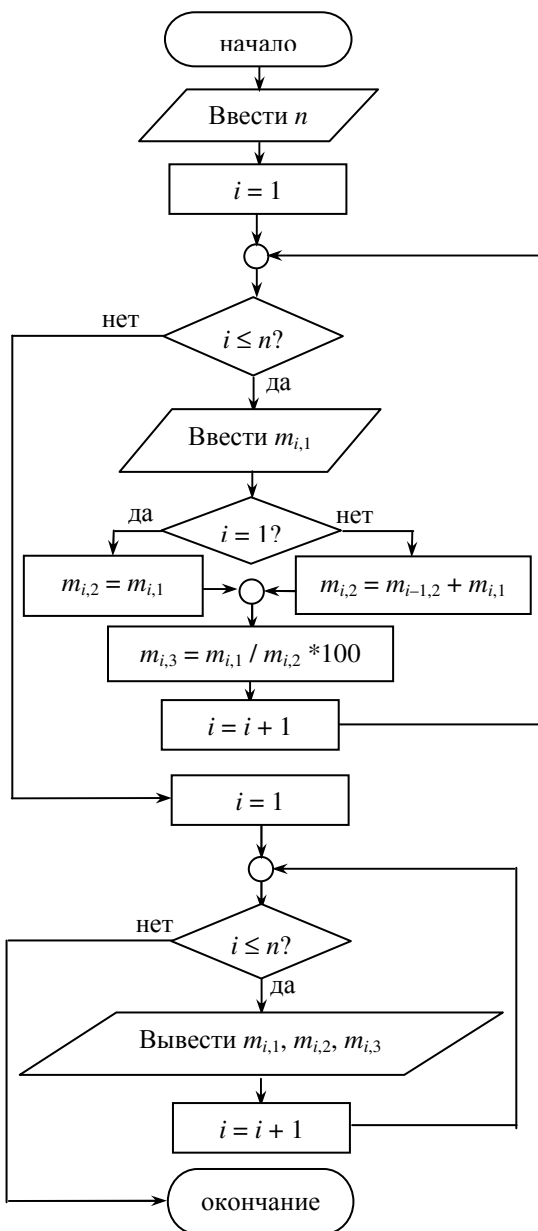
```

Если <группа 1> состоит из одного оператора и <группа 2> также состоит из одного оператора, то оператор **End If** можно не указывать, а условный оператор записывается в одной строке:

IF <условие> THEN <оператор 1> [ELSE <оператор 2>]

Рассмотрим следующий пример. Требуется написать программу, которая для введенных пользователем положительных чисел формирует таблицу, состоящую из трех столбцов: *Число*, *Сумма*, *Процент*. В первом столбце указываются введенные числа. Для каждого числа во втором столбце указывается сумма этого числа с предыдущей суммой (сумма чисел 1 столбца нарастающим итогом), а в третьем столбце – какую часть (в процентах) составляет данное число в этой сумме.

Блок-схема алгоритма решения этой задачи имеет следующий вид.



Этой блок-схеме соответствует следующая программа

```
CLS
DATA -----
DATA Число, Сумма, Процент
READ a$, b$, c$, d$
DIM n As Integer, i As Integer
DO
    INPUT "Сколько чисел"; n
LOOP UNTIL n > 0
DIM m(n,3) As Single
FOR i = 1 TO n
    PRINT i; "-е число";
    INPUT m(i,1)
    IF i = 1 THEN m(i,2) = m(i,1) ELSE m(i,2) = m(i-1,2) + m(i,1)
    m(i,3) = m(i,2) / m(i,1) * 100
NEXT
PRINT a$: PRINT b$, c$, d$: PRINT a$
FOR i = 1 TO n
    PRINT m(i,1), m(i,2), m(i,3)
NEXT
PRINT a$
```

Например, для чисел 10.2, 14.8 и 22.5 будем иметь следующий диалог

Сколько чисел? 3
1-е число? 10.2
2-е число? 14.8
3-е число? 22.5

| Число | Сумма | Процент |
|-------|-------|---------|
| 10.2 | 10.2 | 100 |
| 14.8 | 25 | 59.2 |
| 22.5 | 47.5 | 47.37 |

9.3. Оператор выбора

Оператор выбора обеспечивает описание *сложного* ветвящегося процесса, когда в качестве проверяемого условия записывается *арифметическое или строковое* выражение и в зависимости от результата выбирается одна из указанных групп операторов.

Этот оператор оформляется с помощью ключевых слов **Select Case** (описание проверяемого условия), **Case** (описание ветви) и **End Select** (окончание оператора выбора).

```
SELECT CASE <выражение>
  CASE <результат 1>
    <группа 1>
  CASE <результат 2>
    <группа 2>
    .....
  CASE <результат N>
    <группа N>
  [CASE ELSE
    <группа Else>]
END SELECT
```

Здесь

<выражение> – арифметическое или строковое выражение (в частности, это может быть число, текстовая константа, переменная числового или строкового типа);

<результат *i*> – *i*-й результат ($i=1, \dots, N$) вычисления выражения;

<группа *i*> – группа операторов, которые выполняются, если получен *i*-й результат ($i=1, \dots, N$);

<группа Else> – группа операторов, которые выполняются, если полученный результат не совпадает ни с одним из <результат *i*>.

В качестве результата после **Case** можно указывать

– константу, например, **Case 1**;

– список (через запятую), например, **Case 2, 3**;

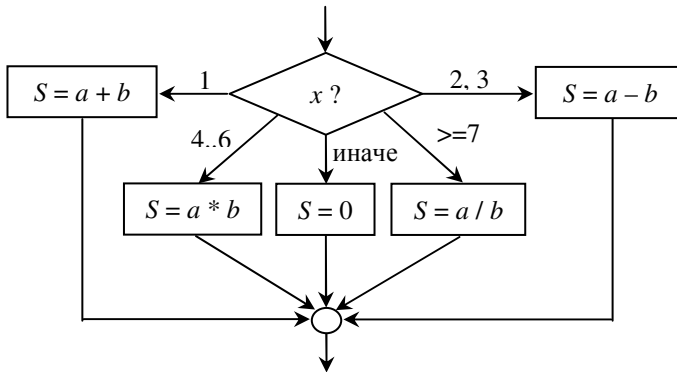
– диапазон (с помощью ключевого слова **To**), например, **Case 4 To 6**;

– логическое выражение (с помощью ключевого слова **Is** и оператора сравнения), например, **Case Is >= 7**.

Например, вычисление

$$S = \begin{cases} a + b, & \text{если } x = 1 \\ a - b, & \text{если } x = 2 \text{ или } x = 3 \\ a * b, & \text{если } 4 \leq x \leq 6 \\ a / b, & \text{если } x \geq 7 \\ \text{иначе } 0 \end{cases}$$

представляется сложной ветвящейся структурой



Для ее реализации можно использовать следующий программный текст:

```
...  
SELECT CASE X  
  CASE 1  
    S = a + b  
  CASE 2, 3  
    S = a - b  
  CASE 4 To 6  
    S = a * b  
  CASE Is >= 7  
    S = a / b  
  CASE ELSE  
    S = 0  
END SELECT  
...
```

10. ОПЕРАТОРЫ ПРЕРЫВАНИЯ

10.1. Выход из цикла

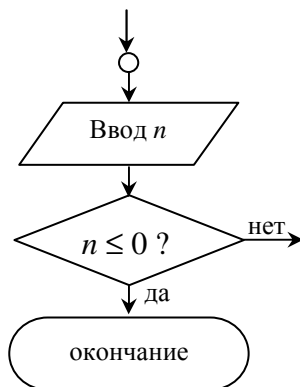
EXIT FOR и EXIT DO

Эти операторы позволяют прервать выполнение циклов соответственно *For – Next* и *Do – Loop Until*. Они применяются, если необходимо завершить цикл немедленно, не продолжая дальнейших действий, составляющих тело цикла.

10.2. Окончание выполнения программы

END – этот оператор указывает на прекращение выполнения программы. Он может располагаться в любом месте программы. Если он последний, его можно не указывать.

Например, следующий фрагмент обеспечивает прекращение выполнения программы, если пользователь введет нуль или отрицательное число.



```
...  
INPUT "Сколько чисел"; n  
IF n <= 0 THEN END  
...
```

11. ПОДПРОГРАММЫ

Подпрограммы используются для исключения дублирования в тексте программы одних и тех же групп операторов. Подпрограммы располагаются в программе после оператора **End**. В каждой из них первая строка должна быть пронумерована, а последним должен быть оператор **Return**.

Для использования подпрограмм применяются следующие два оператора:

GOSUB <номер строки> – переход на первую строку подпрограммы;

RETURN – окончание подпрограммы. Этот оператор фиксирует конец подпрограммы и обеспечивает автоматический возврат к следующему за **Gosub** оператору в программе.

Следует иметь в виду, что каждая подпрограмма работает с данными, которые имеются как в основной программе, так и во всех подпрограммах.

Рассмотрим задачу вычисления величины P в зависимости от максимальных элементов введенных пользователем четырех массивов по формуле

$$P = 3 \cdot M_1 \cdot (M_2 + M_3) - 2 \cdot M_4, \text{ где}$$

M_i – максимальный элемент i -го ($i = 1, 2, 3, 4$) массива.

Соответствующая программа имеет следующий вид:

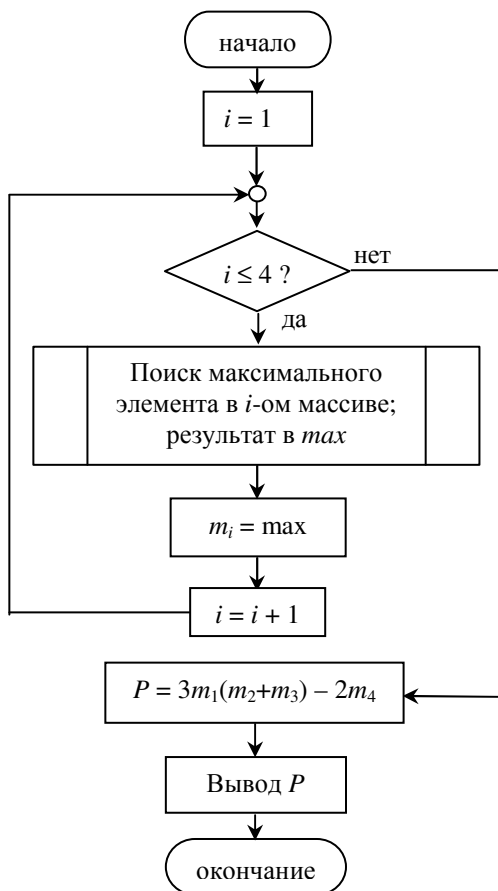
```
CLS
‘ Основная программа
DIM m(4) As Single
FOR i = 1 TO 4
    GOSUB 10          ‘ переход к подпрограмме
    m(i) = max
NEXT
PRINT “Результат:”; 3 * m(1) * (m(2)+m(3)) - 2 * m(4)
END ‘ окончание программы
‘ Подпрограмма поиска максимального элемента
10 DO
    PRINT “Кол-во элементов”; i; “-го массива”;
    INPUT n
```

```

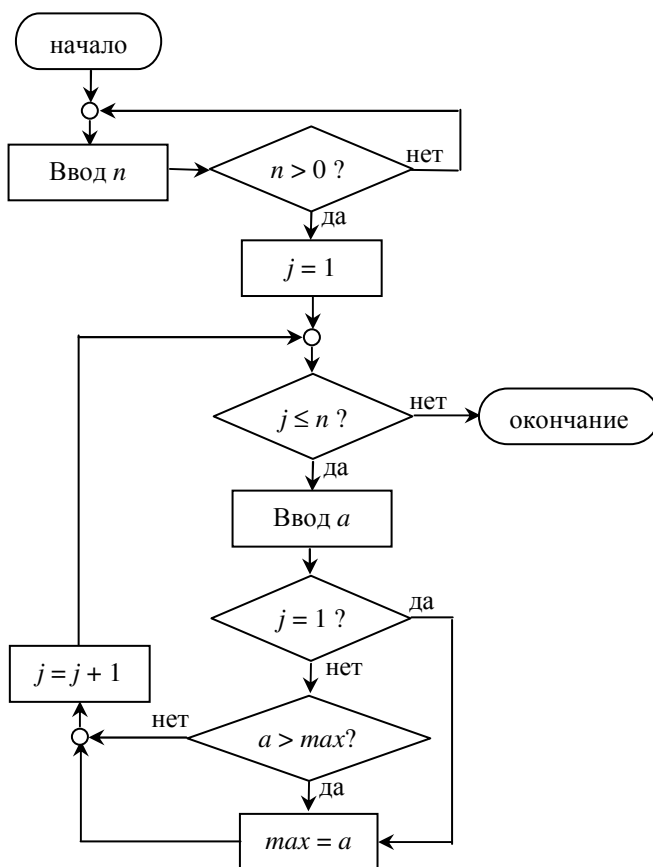
LOOP UNTIL n > 0
FOR j = 1 TO n
    PRINT j; "-й элемент";
    INPUT a
    IF j = 1 THEN max = a ELSE IF a > max THEN max = a
NEXT
RETURN          ' окончание подпрограммы

```

Эта программа реализует следующие алгоритмы:
 – алгоритм основной программы



– алгоритм поиска максимального элемента массива



ЛИТЕРАТУРА

1. Давидов П.Д., Марченко А.Л. Бейсик для начинающих. – М.: Наука, 1994. – 160с.
2. Зельднер Г.А. Программируем на языке QuickBasic 4.5. – М.: АБФ, 1997. – 230 с.
3. Федоренко Ю.П. Алгоритмы и программы на Qbasic. Учебный курс. – СПб: Питер, 2002. – 288 с.

Приложение

ИНТЕРПРЕТАТОР QBASIC

Запуск интерпретатора QBASIC осуществляется файлом qbasic.exe (версия 1.1) или qb.exe (версия 4.5). В результате, на экран выводится окно, в верхней строке которого находится основное меню групп команд, ниже расположено окно для ввода текста программы и внизу – строка состояния, в которой находятся комментарии к работе с интерпретатором. В частности, в ней расположена подсказка, как запустить готовый текст программы на выполнение (<F5=Run>). Справа в окне ввода программы расположена линейка вертикальной прокрутки, а внизу – линейка горизонтальной прокрутки.

Сразу после запуска интерпретатора в окне может появиться диалоговая панель, в которой предлагается закрыть эту панель и перейти к вводу текста программы (*Press Esc to clear this dialog box*) или перейти к работе со справочной информацией о работе с интерпретатором (*Press Enter to see to Survival Guide*). Для выбора одного из этих вариантов следует воспользоваться клавишей <Esc> (ввод текста) или <Enter> (справочная информация).

Управление интерпретатором осуществляется с помощью команд, находящихся в основном меню. Для поиска команды следует

- перейти в основное меню (<Alt>),
- установить указатель на меню команд (клавиши управления указателем),
- выбрать меню команд (<Enter>),
- установить указатель на команду (клавиши управления указателем),
- выбрать требуемую команду (<Enter>).

Работа с файлами осуществляется командами меню *Файл*. Основными из них являются:

- *New* создает новое окно для ввода текста программы;
- *Open...* обеспечивает считывание (*открытие*) готового файла и размещение его содержимого в окне;
- *Save* записывает (*сохраняет*) содержимое окна в файл с тем же названием, что и был ранее, когда считывался с помощью команды *Open*;

– **Save As...** записывает содержимое окна в файл с присваиванием ему нового названия;

– **Print...** производит вывод содержимого окна на печатающее устройство;

– **Exit** осуществляет прекращение работы с интерпретатором.

Троеточие означает, что для выполнения команды требуется дополнительная информация, которая должна быть указана в диалоговой панели, автоматически появляющейся на экране после активизации команды.

Команда **Open** применяется для продолжения работы с ранее созданным текстом программы, хранящемся в файле. После выбора этой команды на экран выводится диалоговая панель, в которой находятся

– поле ввода названия файла (**File Name**);

– окно списка файлов (**Files**);

– окно списка каталогов (**Dirs/Drives**);

– командные кнопки окончания диалога **Ok** (окончание с подтверждением всех установок, выполненных в диалоговой панели), **Cancel** (окончание с отменой всех установок, выполненных в диалоговой панели), **Help** (вывод на экран справочной информации).

Переход от одного элемента диалоговой панели к другому производится с помощью клавиши <Tab> (вперед) или <Shift>+<Tab> (назад).

Указание требуемого файла можно сделать вручную, набрав полную спецификацию файла в поле ввода **File Name**, а можно выбрать каталог в окне **Dirs/Drives** и название файла в окне **Files**, что приведет к автоматическому формированию маршрута (он будет показан под полем ввода) и появлению названия файла в поле ввода.

В окне списка файлов **Files** выводится список названий файлов в соответствии с шаблоном, указанным в поле ввода **File Name**. По умолчанию названия файлов имеют расширение **bas**, поэтому при активизации команды **Open** в поле ввода задан шаблон ***.bas**. Если указать другой шаблон, то в окне **Files** выводится список названий файлов, соответствующих новому шаблону. В частности, указание шаблона ***.*** приведет к выводу в окне списка всех файлов, имеющих в каталоге, выбранном в окне списка каталогов (**Dirs/Drives**).

Использование **Save** предполагает, что в окне программы находится текст, ранее считанный из файла (с помощью команды **Open**).

Активизация этой команды приведет к автоматическому уничтожению старого файла и созданию нового с тем же самым названием.

Команда **Save As...** позволяет записать в новый файл (с новым названием) текст, находящийся в окне текста программы. В частности, это может быть и текст, ранее считанный из файла. Заметим, что выбор команды **Save** после создания нового текста (текст не считывался командой **Open**) приведет к автоматической активизации **Save As...** (вместо **Save**).

В диалоговой панели **Save As...** требуется указать полную спецификацию файла. Для этого следует выбрать каталог в окне **Dirs/Drives** (автоматически будет сформирован маршрут) и указать название файла в поле ввода **File Name**. Кроме этого, можно, не используя окно списка каталогов, сразу ввести полную спецификацию файла в поле ввода.

Ввод и редактирование текста. При вводе и редактировании текста используется как клавиатура, так и команды меню **Edit**.

Для перемещения в тексте используются клавиши управления указателем, для удаления текстовых знаков – клавиши **<Delete>** (удаление знака в месте размещения текстового курсора) и **<Backspace>** (удаление предыдущего знака). В процессе редактирования возможна как вставка, так и замена знаков. В первом случае текстовый курсор изображается мигающей чертой под знаком, а во втором – мигающий курсор полностью накладывается на текстовый знак. Для изменения режима “вставка/замена” следует щелкнуть по клавише **<Insert>** (или **<Ins>**).

Копирование, перемещение и удаление текстовых фрагментов осуществляется командами меню **Edit**. При этом, следует иметь в виду две особенности:

- эти команды применяются к *выделенным* фрагментам; такое выделение осуществляется клавишами управления указателем при нажатой клавише **<Shift>**;

- для копирования и перемещения используется так называемый *буфер*, в который сначала надо скопировать или переместить выделенный фрагмент, а затем скопировать его содержимое в место расположения текстового курсора.

Поэтому в меню **Edit** содержатся команды:

- **Cut (<Shift>+)** перемещает фрагмент в буфер;
- **Copy (<Ctrl>+<Ins>)** копирует фрагмент в буфер;

– **Paste** (<Shift>+<Ins>) копирует содержимое буфера в то место, где находится текстовый курсор;

– **Clear** (или <Delete>) удаляет фрагмент.

Для вставки новой строки достаточно установить текстовый курсор в начало строки и щелкнуть по клавише <Enter>.

Ввод и редактирование осуществляется под контролем специального редактора, который помогает синтаксически правильно оформить программный текст. В частности, при переходе к следующей строке автоматически производится

– преобразование букв, составляющих ключевые слова, в прописные;

– установка отступа от начала строки, аналогичного предыдущей строке.

Исполнение программы. Как уже отмечалось ранее, исполнение программы обеспечивается интерпретатором. Если в процессе выполнения программы обнаружена ошибка, интерпретатор выводит на экран соответствующее сообщение и возвращает пользователя к работе с текстом. В случае, если ошибка не синтаксическая, а логическая (некорректные исходные данные, попытка деления на нуль и т.п.), интерпретатор отмечает строку, в которой произошла остановка. Такая отметка дает возможность пользователю запустить программу не с самого начала, а с этой строки.

Запуск программы на выполнение осуществляется командами меню **Run**, основными из которых являются:

– **Start** (<Shift>+<F5>) запускает программу на выполнение с самого начала;

– **Continue** (<F5>) запускает программу на выполнение с того места, где произошла остановка (или с самого начала, если остановки не было);

– **Restart** перемещает отметку (для команды **Continue**) на первую строку.

Пользователь может прервать исполнение программы, для чего достаточно выполнить <Ctrl>+<Break>.

СОДЕРЖАНИЕ

| | |
|--|----|
| Введение..... | 3 |
| 1. Основы алгоритмизации..... | 5 |
| 1.1. Основные понятия..... | 5 |
| 1.2. Основные алгоритмические структуры..... | 7 |
| 2. Основные понятия языка BASIC..... | 11 |
| 3. Типы данных..... | 13 |
| 4. Выражения..... | 17 |
| 5. Операторы очистки экрана, присваивания, оформления комментариев..... | 19 |
| 6. Ввод и вывод данных..... | 21 |
| 6.1. Оператор позиционирования курсора на экране..... | 21 |
| 6.2. Вывод данных на экран..... | 21 |
| 6.3. Вывод данных на экран с позиционированием..... | 22 |
| 6.4. Диалоговый ввод данных..... | 22 |
| 6.5. Ввод данных из программы..... | 23 |
| 7. Описание (объявление) переменных..... | 27 |
| 8. Организация цикла..... | 29 |
| 8.1. Цикл с предусловием (операторы <i>While – Wend</i>) | 29 |
| 8.2. Цикл со счетчиком (операторы <i>For – Next</i>)..... | 31 |
| 8.3. Цикл с постусловием (операторы <i>Do – Loop Until</i>) | 33 |
| 9. Операторы перехода (ветвления) | 35 |
| 9.1. Оператор безусловного перехода..... | 35 |
| 9.2. Оператор условного перехода (условный оператор).... | 35 |
| 9.3. Оператор выбора..... | 40 |
| 10. Операторы прерывания..... | 42 |
| 10.1. Выход из цикла..... | 42 |
| 10.2. Окончание выполнения программы..... | 42 |
| 11. Подпрограммы..... | 43 |
| Литература..... | 45 |
| Приложение. Интерпретатор QBASIC..... | 46 |

Борис Семенович Лещинский

ОСНОВЫ ПРОГРАММИРОВАНИЯ
НА ЯЗЫКЕ BASIC

Учебное пособие

Отпечатано на участке оперативной полиграфии
Редакционно-издательского отдела ТГУ.
Лицензия ИД № 00208 от 20.12.1999 г.

Заказ № 157 от «06» 09 2005 г. Тираж 200 экз.