КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ГЕОЛОГИИ И НЕФТЕГАЗОВЫХ ТЕХНОЛОГИЙ

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра системного анализа и информационных технологий

Т.С. ТАГИРОВ

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ НА MS QBASIC 4.5 И QB64

Учебно-методическое пособие

Казань – 2018

ДК 004.42
ББК 32.973-018

Принято на заседании учебно-методической комиссии Института геологии и нефтегазовых технологий
Протокол № 4 от «_23_» _марта__ 2018 года

Введение в программирование на MS QBASIC 4.5 и QB64: учеб.-метод. пособие / Т.С. Тагиров – Казань: Казан. ун-т, – 2018. – 74 с.

Учебно-методическое пособие предназначено для студентов, обучающихся на первых курсах ИГиНГТ, в частности, также и по направлению 21.03.01 «Нефтегазовое дело» по дисциплине «Информационные технологии».

В пособии рассматриваются основы самостоятельного овладения навыками программирования. Приводится ряд программ, требующих самостоятельного осмысления и развивающих умение находить ошибки. Для освоения пособия требуется начальное знакомство с IT, знания из курса математики (функции, матрицы, интегралы, вектора).

Издание содержит инструкции по установке и запуску языка программирования, краткий английский словарь, а также вопросы для самостоятельной работы.

Введение

В последние три-четыре года почти все компьютеры «работают» в ставшей привычной для глаза среде Windows. Уже появилось поколение молодых людей и девушек просто теряющихся, завидев пустой черный экран и мерцающий на нем командный «ДОСовский» промптер. Даже простая опция выхода в «DOS-сессию» вызывает у них весьма смутные представления о последствиях сделанного выбора. Нет никаких сомнений в том, что программное обеспечение будет развиваться и далее в сторону все большего «сокрытия» от «пользователей» того, что творится на самом деле с файлами, директориями и т.д., как это уже давно сделано на «Макинтошах». Пользователь такого компьютера понятия не имеет о том, что за драмы происходят на его «винтах», в файловой системе, какие символы стоят в начале, внутри и в конце файла, как конкретно выглядит машинная программа, что она делает, и т.д. Добраться же до командного промптера в «Маках» – дело слабо-осуществимое по определению.

Однако, а это события «новейшей» истории, совсем недавно всякий студент ничего более сильного, чем ДОСовский Нортон-Коммандер на мониторе учебного компьютера не видел, и видеть не мог. Все сказанное выше объясняется резким скачком в развитии вычислительной техники в сторону качественного увеличения мощности машин, качественных сдвигов в сторону улучшения пользовательских свойств как программных продуктов, так и системных оболочек. Именно поэтому у всякого преподавателя возникает проблема выбора: преподавать ли в курсе с довольно старым, но весьма обязывающим названием «Программирование на ЭВМ» новейшие офисные приложения (как правило, разработанные компанией Майкрософт г-на Билла Гейтса, совсем недавно ушедшего в новые разработки), работающие исключительно на мощных Пентиумах с качественным монитором и исключительно под руководством «мыша», или же вместо оглушения студенческой головы всяческими «Икселами» и «Уордами» вначале изложить достаточно ясные части ДОС и научить ее (голову) работать хоть на каком-то языке программирования? На вопрос этот ответить сразу достаточно трудно хотя бы потому, что:

1. Сложно отрицать необходимость «пользовательского» вла… двумя-тремя офисными приложениями, но и не менее сложно отрицать силу традиционного подхода к изучению любого предмета по правилам «от простого – к сложному», «от арифметики – к математическому анализу».
2. Нужно быть твердо убежденным в том, что без осознания ДОСовской «компьютерной арифметики» и изучения какого-либо языка программирования на уровне написания программ нельзя научить студента информатике и открыть ему доступ к действительному овладению компьютером как инструментом научного познания.

Именно фундаментальность получаемых знаний во многом определяет качество образования, а в этом отношении отечественной высшей школе за качество краснеть не приходится (чего не скажешь о существующем учебном оборудовании и лицензионном программном обеспечении…). Итак, примем за исходную точку тезис о том, что без овладения элементарными навыками программирования нельзя научиться «понимать» компьютер, да и всю информатику.

Сразу же возникает вопрос второй: какой язык программирования наиболее удобен для начального изучения? Язык «Си», «Паскаль», или же «Бэйсик»? Ответ на этот вопрос не может быть однозначным. Учтем, что далеко не у всех студентов дома есть мощный (или вообще какой-нибудь) компьютер. Далеко не у всех информатика в школе преподавалась на уровне выдающегося лицея. Про язык «Си» говорить не будем, так как выгоднее изучать сразу «С++». Но даже если на вашем факультете исторически сложилось так, что на кафедральных ПК кроме «Паскаля» ничего нет, да и все научные сотрудники понимают только этот язык, то все еще рано сбрасывать QuickBASIC (в дальнейшем, QB) со счетов. Причин этому несколько, важнейшая из которых – совместимость QB с самой операционной системой MS-DOS. Еще один плюс — QB обычно «встраивался» в каталог поставляемых операционных систем MS-DOS 3.1 и выше. Кроме того, нельзя не отметить достаточную мощность QB для решения многих задач по сбору и обработке информации, полученной в

сперименте. Если учесть также и богатую справочную систему QB, в ко… приводится не только формат (т.е. способ представления) команд, но и множество деталей и даже примеры (с единственным небольшим недостатком — все справки выдаются на английском языке), то становятся ясными преимущества данного языка. Кроме того, QB качественно отличается в лучшую сторону от иных существующих версий BASIC'ов своей универсальностью и интегрированностью в операционную систему.

Текст, предлагаемый вашему вниманию ниже, ни в коей степени не претендует на звание полного учебника по QB — для написания книги такого рода нужно намного больше и места, и терпения писателя, и терпения читателя. Цель данного пособия — помочь разобраться в самых началах программирования в QB, освоить полезные навыки в использовании некоторых популярных команд. Говоря иными словами, красивому и чистому батерфляю мы научиться вряд ли сможем; однако наша промежуточная и очень важная цель — не утонуть в бассейне — вполне достижима. Структура книги достаточно проста: вначале мы опишем некоторые особенности QB, затем займемся написанием нескольких конкретных программ на нем, а в конце приведем сведения, без которых читателю (a piori не англоязычному) будет нелегко установить контакт с интерфейсом QB. Мы предполагаем, что самые элементарные команды и понятия операционной системы MS-DOS читателю уже известны.

В качестве еще одного весьма важного обстоятельства отметим, что учащийся просто обязан:

1. вводить все символы в латинских символах, не употребляя русский шрифт вовсе и ни в коем случае не нажимая на клавиши «Shift» и «CapsLock» для придания правдоподобности набиваемому тексту программ;
2. уметь свободно набирать текст с клавиатуры и понимать разницу между указаниями {Alt + R, R} и {Alt, R} – в первом случае нужно 1) нажать клавишу {Alt} и затем, НЕ ОТПУСКАЯ КЛАВИШУ Alt, нажать (вряд ли это удастся сделать тем же пальцем!) клавишу R, после чего 3) ОТПУСТИТЬ ОБЕ КЛАВИШИ И 4) СНОВА нажать

на клавишу R. Второе указание предусматривает г[illegible]о последовательное нажатие клавиш Alt и R;

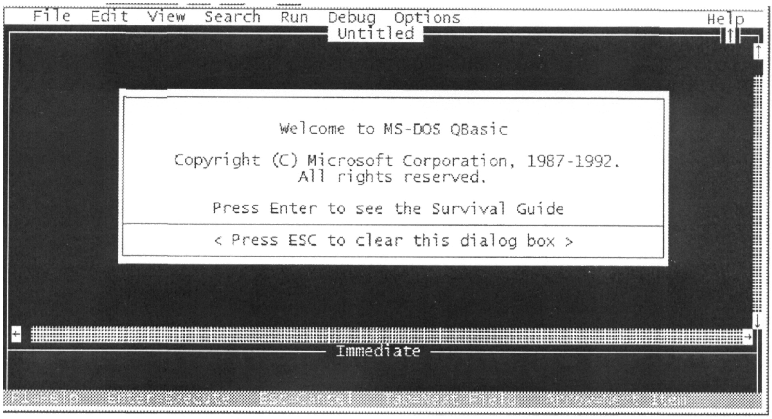
3. не забывать отделять операторы (то, что определяет действие) от операндов (то, над чем операторы работают, т.е. с чем, собственно, действия и производятся) ПРОБЕЛОМ;
4. помнить, что, например, механическое вызубривание иностранного языка не приводит к его освоению; в абсолютно той же степени эта истина верна и для языков программирования;
5. стараться менять условия задач, самому (или самой) придумывать новые задачи и затем самостоятельно писать программы для их решения — это лучшее средство научиться программировать;
6. ни в коем случае не заниматься простым (и тупым!) перенабором программ из книги на экран компьютера — вы тогда просто не научитесь писать программы! Лучше всего набирать программы, скрыв данный текст от ваших глаз;
7. здесь же отметим одну особенность именно женских характера и памяти: представительницы прекрасной половины человечества способны запоминать огромные количества текста, вовсе не переживая за его содержание. И этот факт даже можно подтвердить экспериментально. И именно это обстоятельство способствует тому, что многоуважаемые дамы иногда никак не могут понять чего же хочет от них тот, от кого они сами хотят получить зачет – они почти буквально (зрительно) запоминают текст программ (правда, путая при этом операторы и буквы), программы эти (и это тоже правда) почему-то не работают, а зачет почему-то не ставят… Утешением может послужить то обстоятельство, что первым в мире программистом считается Ада (Эйда) Лавлейс — дочь лорда Байрона;
8. опыт нескольких лет показывает также, что основной причиной неудач с получением зачетов у молодых людей является сочетание органически непреодолимой лени, большой порции небрежности и потрясающего неумения организовать себя и свою память.

Если Вы, добравшись до этих строк, улыбнулись, то к Вам они не [illegible] самое малое отношение.

В заключение мне очень бы хотелось поблагодарить сотрудников ЦИВТа за предоставленную возможность пользоваться на лекциях полиэкранным дисплеем в ауд. 209 (правда, в последнее время уже некоторые ячейки-трубки уже не работают). Слова искренней благодарности хотелось бы также высказать в адрес руководителей и сотрудников Центра дистанционного обучения КГУ (ауд. 109 ЦИВТа) за предоставленную возможность пользоваться ЖК-матрицей и профессиональным проектором. Специальное спасибо ЦИВТу КГУ и лаборатории компьютерных средств обучения за возможность пользования MS QuickBASIC'ом версии 4.5.

1. Запуск QuickBASICa и его пределы.

Если вы находитесь в командной строке DOS, а директория, в которой находятся файлы QuickBASICa имеет доступ, заранее открытый через команду MS-DOS "PATH" в файле autoexec.bat, то достаточно набрать символы qb" или "qbasic" и нажать на клавишу «Enter». Вы увидите голубое окно, в котором можно выделить несколько меню и подокон (см. рис.)



Заставка в центре экрана переводится так:

«Добро пожаловать в MS-QuickBASIC.

Авторские права Корпорации Майкрософт. Все права защищены.

Нажмите Enter, чтобы просмотреть Руководство по Выживан…
Нажмите Esc, чтобы очистить это диалоговое окно».

Верхняя часть пользовательского интерфейса (то-бишь, экранного представления той программы, с которой вы работаете) имеет строку меню, которое активизируется при нажатии клавиши Alt.

Основную часть экрана занимает окно редактора QB, в котором под вашим чутким руководством разворачиваются события по написанию программы. Это окно снизу и справа ограничено «полосами прокрутки», т.е. областями, в которых 1) темные квадратики указывают положение видимого куска текста в целом тексте. Если вы работаете в QB с помощью мыши (занятие очень вредное с образовательной точки зрения), то по всему тексту можно передвигаться, щелкая мышкой по стрелкам в полосах прокрутки.

Нижняя часть экрана занята так называемым окном «Сразу». В этом окошке вы можете выяснить некоторые вопросы, не запуская всю программу (например, вычислить $\sqrt{7}$ или выяснить, чему будет равно число 3,141592654/2). «Прыгать» по окнам нужно клавишей F6.

Самая нижняя строка зеленоватого цвета содержит некоторые подсказки.

Зайдем в верхнее меню. Оно содержит восемь пунктов:

File Edit View Search Run Debug Options и **Help**

(В конце этой книги имеется маленький англо-русский словарик, в который уже можно заглянуть!)

Всякий конкретный язык программирования, за исключением самых абстрактных версий, работает с числами или оперирует числовыми величинами. Такие величины именуют иногда ***константами***, но они могут и изменяться в процессе выполнения программ. Для использования таких меняющихся цифровых величин служат объекты языка, называемые ***переменными***. Переменные могут содержать не только числовую, но и символьную информацию, поэтому их и различают как ***числовые*** и ***символьные*** (***string***). Отметим, что всякая переменная

личина должна иметь ***имя***, которое является, по определению, вели… **символьной**. Сразу отметим, что всякая переменная может быть простой (без дополнительных параметров) или, наоборот, параметризованной, причем параметров может быть не один или два, а больше.

Для большей ясности приведем такую интерпретацию понятия переменной некоторого языка. Если в программе следует работать с какими-то величинами, изменяющимися или изменяемыми в ходе выполнения программы, то машина, натолкнувшись на такой объект в теле программы, который она интерпретирует как переменную, открывает в машинной памяти некий ящик (выделяет частичку памяти на цели работы с числами или символами, которые будут фигурировать под флагом данной переменной). Такой ящик непременно должен иметь имя. Если в программе попадаются простые переменные, то их можно считать разбросанными по программе ящиками, в которые программа что-нибудь складывает, записывает, вычитает, умножает, делится с другими программами и т.д.

Скажем, команда

```
LET X = 1
```

означает, что машина создает ящик с именем X и закладывает в этот ящик единицу. Если где-то далее встретится команда

```
LET Y = X + 1
```

то машина организует новый ящик с биркой «Y», затем заглянет в ящик X, «увидит» там единицу, эту единицу занесет в Y, а затем добавит туда же еще одну единицу. Таким образом, далее на «поле военных действий» нашей программы возникает два ящика с табличками, в одном лежит 1, в другом же — число 2. При использовании так называемых **параметрических** (сложных) переменных, их можно обозначать или некоторой общей частью, или вводить **параметр переменной**. Например, возможно использование имен X1, X2, X3, …, Xn-1, Xn — это пример написания в QB **простых переменных**, о взаимной связи которых мы в явном виде ничего QB не говорим. А вот если мы запишем то же самое в виде X(1), X(2), X(3), …, X(n-1), X(n), то QB поймет это как приказ открыть целую «улицу» «ящиков» с общим именем X, причем на этой улице существует «нумерация домов» от 1 до n. «Номер дома» каждого ящика служит в

нном случае **параметром** этой **однопараметрической переменной**. … м надеяться, что теперь запись X(1,.1), X(1,2,), …, X(1,m), X(2,1), X(2,2), …, X(2,m), … X(n, m) станет понятной нам записью некоторого «палаточного городка» с этими самыми ящиками переменных, в котором есть n переулков и m улиц (а может быть и наоборот). Имя этому городку — **двухпараметрическая переменная** X(I,J), где I и J изменяются в определенных пределах. Если нам понадобиться записать трехпараметрическую (причем символьную, а не числовую) переменную, то можно воспользоваться записью Pogoda$(I,J,K), где в каждый ящик теперь уже многоэтажного муравейника со своими улицами (это где J и K фиксированы, а I меняется), переулками (на них фиксируются I и K, а свободно меняется J) и этажами (каждому K-ому этажу соответствует фиксация I,J) можно вложить уже не число, а символьную строчку (например, «клюет» или «не клюет»). Если вы проводите наблюдения за успехами вашей рыбалки и записываете под I —номер рыбалки, J — температура при этой рыбалке, а K — давление, придавая переменной значение «клюет» или «не клюет», то в результате вы сможете организовать некоторое множество данных. Отсортировав их по признаку «клюет» - «не клюет», вы сможете получить метеоусловия, при которых рыба клюет или, наоборот, параметры погоды, при которой удочки надо хранить дома.

Имя переменной в QB **не может иметь разрывов или пробелов** (представьте себе имя «Ив ан» или «Ел ена»!) и должно **не** содержать некоторых специальных символов (например, ?,*, / и т.д.).

Очевидно, что никакая машина не может работать со сколь угодно большими или сколь угодно (бесконечно) малыми величинами, будь они числовыми или символьными, т.е. у машины существуют определенные пороги «чувствительности» чисел, или ***пределы*** числовых и символьных выражений. Говорят, что на островах Полинезии существуют племена, у которых счет идет по принципу «один, два, три, много». У столь выдающегося создания, коим является компьютер, также существуют рамки, за которыми величины (числовые или символьные) становятся ему, скажем, непонятными. Отметим сразу, что эти пределы зависят также и от описания (или класса) этих величин. QB различает величины:

ЧИСЛОВЫЕ

Целые или INTEGERS,

Длинные целые или LONG INTEGERS,

Числа одинарной точности или SINGLE-PRECISION NUMBERS,

Числа удвоенной точности или DOUBLE-PRECISION NUMBERS;

СИМВОЛЬНЫЕ

(в случае обозначения переменных величин используются простые и параметрические символьные записи, последние отличаются присутствием скобок (), внутри которых через запятую вписываются параметры).

(**NB** Nota bene или «запомни хорошенько»): *Как только в потоке вводимого вами числа появится буква или иной нецифровой символ, QB проинтерпретирует такое выражение как стринг; если же вы вводите вместо стринга довольно длинное число, то QB проинтерпретирует его как странноватый, но стринг, в котором нет букв, а есть только символы-цифры.*

Собственно пределы величин в QB можно описать следующей таблицей:

	МАКСИМУМ	МИНИМУМ
Длина **имени** переменной	40 символов	1 символ
Длина символьной величины (стринг'а)	32 767 символов	0 символов
Целые	32 767	-32 768
Длинные целые	2 147 483 647	-2 147 483 648
Числа одинарной точности:		
Положительные	3 .402823E+38	2.802597E-45

Отрицательные

-2.802597E-45 -3.402823E+38

Числа двойной точности:

Положительные

1.79769313486231D+308 4.940656458412465D-324

Отрицательные

-4.940656458412465D-324 -1.79769313486231D+308

Это значит, что число, описанное в вашей программе на QB как величина двойной точности, хранится и обрабатывается QB со 1) знаком, 2) шестнадцатью цифрами, 3) указателем точности («D») и 4) порядком. Отметим, что представление чисел в QB использует знак # для обозначения целых величин, значок «E» используется для чисел одинарной точности, тогда как «D» означает двойную точность числа, внутри которого такой знак Вам встретился.

В QB существуют понятия **утверждений**, которые мы обычно будем традиционно называть **операторами**, и **функций.** Для отделения первых (операторов) от того, к чему они относятся (операндов) используется обязательный **пробел.** Есть, правда, операторы столь популярные, что имя их даже не приводится (оператор присвоения LET). Что же касается функций, то на их присутствие указывает наличие аргумента в скобках.

В QB все операторы и функции имеют одну отличительную особенность — они пишутся исключительно заглавными буквами. НО (!) ни в коем случае НЕЛЬЗЯ их вводить с клавиатуры заглавными буквами. Напротив, используйте только маленькие буквы! Редактор QB настолько умен, что распознает ПРАВИЛЬНО введенные операторы и при переходе на новую строку, САМ ПРОПИСЫВАЕТ ИХ ЗАГЛАВНЫМИ БУКВАМИ. Поэтому ввод с клавиатуры заглавных букв при написании программы есть не просто обман программы, а очень вредный САМООБМАН.

Отметим, что в QB существуют некоторые «запрещенные» имена, которые не стоит использовать для обозначения переменных. В Приложениях можно увидеть список операторов и функций QB.

Меню «File» в минимальном виде содержит опции **N**ew **O**pen… Save **A**s … **P**rint и Exit. В QB версии 4.5 это меню может быть расширено выбором {Alt + O} и соответствующего пункта.

Буквы, выделяемые белым цветом (у нас они просто полужирные), означают, что при нажатии соответствующей буквы на клавиатуре вы получите желаемый выбор опции без всяких передвижений курсора по пунктам меню. Так, например, сочетание клавиш {Alt + F, x} просто выведет вас из QB, а весьма важное сочетание {Alt + F, S} позволяет достаточно быстро и часто сохранять (спасать) сотворенные вами **коды** программы или ее **листинг** (так называют исходный текст программы).

2. Немного о программах, программировании и программистах

Известно, что **программой** называют реализацию какого-то алгоритма на том или ином языке программирования. **Алгоритм**, в свою очередь, это строго определенная **последовательность** элементарных **шагов** и/или **переходов** для достижения определенной **цели**.

Та часть алгоритма, которая предусматривает строго последовательное выполнение шагов по порядку, называется **линейной частью** алгоритма. Если в какой-то точке алгоритма появляется возможность перейти далее на более чем один определенный шаг, то такую точку называют точкой **ветвления** или точкой **перехода.**

Если инструкция «перейти на такой-то шаг» не имеет никаких условий, то переход называют **безусловным**. Если же переход осуществляется в зависимости от значения какой-то величины (параметра), которая к данному шагу становится определенной, то говорят об **условном переходе**, величину, влияющую на переход, называют **параметром перехода**, а пороговые или определяющие значения этого параметра — **условием перехода**. Выполнение каждого шага или некоторого набора шагов порядком их расположения в тексте программы (мы уже говорили о том, что его иногда называют **кодами программы** или ее **листингом**).

Если одна и та же последовательность шагов программы может повториться при ее выполнении более, чем один раз, то такую часть программы

зывают **циклом**. Так как количество прохождений цикла (да и в[се] «впадение» программы в цикл) определяется некоторым условием на величину, называемую по-иному **параметром цикла**, то мы обнаруживаем еще одно использование слова параметр. Ясно, что для попадания программы в некоторый цикл требуется, чтобы это позволили сделать пороговые значения параметра. Если цикл прописан неверно, то может произойти так называемое «зацикливание» программы (например, при слишком больших значениях параметра цикла, или при неизменности параметра цикла, несмотря на прохождение по циклу, или «колебательные», «маятниковые» изменения параметра цикла, из-за чего программа не может увести управление из тела цикла). В качестве примера можно рассмотреть такую ситуацию: вы приходите к бюрократу Ивану Ивановичу, а он посылает вас к Петру Петровичу, который вас отсылает к Сидору Сидоровичу, а оный — снова к Ивану Ивановичу и т.д. Если за параметр принять ваше терпение, то такой цикл прервется только тогда, когда ваше терпение (или здоровье) закончится. Если же обязательным условием Вашей задачи является фраза «непременно получить такую-то бумагу», то из постановки задачи видно, что цикл не закончится полезным результатом. Циклы могут входить один в другой (говорят, «**вложенные циклы**»), или выполняться по частям каждый период.

Искусство программиста, т.е. человека, пишущего или создающего программы, заключается не в простом буквоедском написании самого текста при безусловно широком и полном знании «оборотов» каждого конкретного языка программирования, но в создании красивой и целенаправленной структуры выполнения программы. Поэтому часто программисты не только пишут саму программу по набору правил, но и находят остроумные и нестандартные решения для выполнения шагов «макро-алгоритма» написания программы и достижения поставленной цели. Именно накапливающийся мировой опыт в создании программ и определяет основные направления развития всего программирования. Некоторые части программ становятся (узко-признанными – потому что среди профессионалов) стандартами. Так, например, уже стал стандартом вид графического пользовательского интерфейса (GUI – Graphic User Interface) — обычно вверху располагается меню опций, каждый пункт которого

разворачивается вниз (pull-down) при выборе. Именно этот принцип постр... GUI и использован в QB.

Остановимся подробнее на верхнем меню QB и зайдем в некоторые его пункты.

В пункте **File** раскрываются подпункты

New — начать создание новой программы

Open — открыть ранее созданную программу

Save — сохранить (спасти) созданную или отредактированную программу под старым именем

Save As— сохранить (спасти) созданную или отредактированную программу под новым именем или в другом месте

Print — распечатать программу на принтере

Load file — загрузить какой-нибудь файл в окно редактора QB

Unload file — выгрузить файл из окна редактора QB

DOS Shell — выйти в DOS (обратное возвращение происходит после набора EXIT с клавиатуры)

Exit — выйти из активного сеанса работы с QB.

Достаточно важным является также и пункт **Edit**. В нем можно увидеть опции (подпункты)

Cut — вырезать выделенный фрагмент текста

Copy — скопировать (не вырезая или не удаляя) выделенный фрагмент текста в буфер

Paste — вставить фрагмент текста из буфера в текст в окне редактора на текущей позиции курсора

Clear — удалить весь выделенный фрагмент текста (без запоминания в буфере).

Следует обратить особое внимание на указанные справа от этих опций клавиатурные сокращения (ShotKeys). Например, обычно сочетание клавиш {Ctrl + Insert} забирает фрагмент в буфер без удаления с экрана, {Shift + Delete} — забирает фрагмент с удалением с экрана, {Shift + Insert}— вставляет фрагмент из буфера на место после курсора.

Для выделения текста нужно 1) встать курсором на начало нужной строки, 2) нажав и удерживая клавишу {Shift}, *стрелками «забелить» нужный кусок текста.*

Нельзя «тараканить», т.е. по шажочкам ползать курсором по всему тексту. Это утомительно, некрасиво, непрофессионально и т.д. Для перескакивания через слово служит комбинация {Ctrl + [←] или [→]}. *В начало строки можно сразу попасть клавишей* {Home}, *в конец строки — клавишей* {End}. *В сочетании с* {PgDn} *и* {PgUp} *можно быстро добраться до начала текста или его конца. Разумно пользоваться обеими режимами курсора в редакторе QB: при нажатии клавиши* {Insert} *вы заметите, что курсор стал мигать во всю букву (а не подчеркивать ее). В этом режиме вы не вставляете новый текст, а замещаете новыми буквами старый (пишите поверх старых знаков). Еще раз нажмите ту же клавишу, и курсор снова уменьшится, а новые буквы будут «вставляться», сдвигая остаток строчки вправо от курсора.*

Отметим, что важными (в принципе, все пункты очень важны!) пунктами меню являются **Run** и **Help**. Первый пункт позволяет запускать, продолжать выполнять запущенную или перезапускать программу. Во втором пункте вы найдете расширенную помощь по всем вопросам, правда, только на английском языке. История последней половины XX века сложилась так, что программирование более всего развивалось в США. Именно поэтому мы имеем дело с операторами, написание которых часто имеет прямую английскую трактовку. Научиться программировать — значит научиться хотя бы азам английского языка. При этом полезно вспомнить, что 1) весь Интернет понимает только по-английски; 2) человек, знающий два языка становится в два раза богаче; 3) если вы знаете только один язык, то вы и его-то не знаете толком.

Кроме того, QB активно работает с вами, выставляя вам окошко с комментариями или предупреждениями или сразу после ошибки в набираемой программе, или после ее запуска. Нужно уметь понимать то, что 1) вам пытаются помочь, 2) что именно вам говорят.

3. Операторы (Statements) QuickBASIC'a

Все операторы (Statements) в QB, вообще говоря, пишутся прописными буквами (мы уже условились этого никогда не делать, он сам их сделает потом большими!). Операторы предъявляются всегда в определенном **формате**, т.е. по некоторым правилам, в которых имеется достаточно много свободы, но порядок, в котором указываются опции формата, должен соблюдаться строго. Цель каждого оператора — реализовать один шаг требуемого алгоритма или некоторую его часть (например, цикл). Оператор всегда занимает начало строки (или строки начала и конца оператора). Единственное, что может стоять **перед** оператором, это **метка**. После оператора обязан присутствовать **пробел**, после которого записывают то, с чем оператор должен оперировать, или **операнды**.

В тексте (листинге) программы QB автоматически различает **строки** (каждая строка заканчивается определенным не печатающимся символом EOL (End of Line)), а затем порядок текста в строке (такой порядок называется лексикографическим).

Именно лексикографический порядок и определяет очередность исполнения (или передачи управления) операторов в ходе выполнения той или иной программы. Единственное, что может нарушить такой порядок, это явное указание (опять же при помощи оператора) на «принудительное» исполнение не следующей строки а той, метка которой стоит в **операторе перехода**. Кстати, в различных старых версиях BASICa обычно текст каждой строки начинался с определенного номера. Фактически, это был способ пометить каждую строку. Старые BASICи определяли для себя порядок выполнения именно по метке в начале строки (номеру, ибо такая метка обычно была числом). Большим неудобством при этом было то, что при желании «втиснуть» какой-то кусок программы между двумя уже написанными строками приходилось перенумеровывать все строки. Способ избежать связанных с этим трудностей заключался в том, что строки нумеровались не подряд, а с «зазором» (например, 100, 110, 120 и т.д.). Как мы видим, теперь QB сам определяет порядок по их нахождению в тексте.

Еще одно замечание: QB может допускать присутствие не одного оператора в одной строке; в этом случае операторы разделяют двоеточием «:».

Иногда возможность написать несколько операторов в одной строке не только компактифицирует текст программы, но и позволяет избежать неуклюжести. Не следует, однако, перегружать строки и пользоваться двоеточием без особой нужды.

4. Операторы LET, PRINT, CLS и END

4.1 Оператор LET

Слово «Let» в переводе с английского означает «пусть», «допустим». Оператор LET неизбежен при желании: «зарядить» какую-либо переменную конкретным числом; выполнить операцию присвоения, т.е. присуждения тому, что стоит в левой части выражения того значения (возможно и алгебраического), которое стоит в его правой части. Самое интересное заключается в том, что QB разрешает и вовсе **не писать сам оператор**.

Формат предъявления оператора LET:

```
[LET] [new]variable = variable, expression, number
```

здесь запись выражения внутри квадратных скобок [ ] означает, что его можно опустить, сразу после оператора стоит [новая] переменная, знак равенства всегда и обязательно присутствует, а справа от равенства стоит или старая переменная, или выражение, содержащее определенные (старые) переменные, или вообще число. Довольно общим местом является предложение о том, что знак равенства в программе не имеет ничего общего со знаком равенства в математике. В математике выражения

```
A = A+1
D= 3*D
B=SQR((B^3))
```

не имеют достаточно ясного смысла (не может же нормальное число равняться себе да еще плюс единица). Данные выражения читаются так

`A = A+1` (теперь переменная A есть старое значение A (имевшееся до этой строки) и еще плюс один),

`D = 3*D` (а теперь переменная D равна своему старому значени... этой строки), умноженному на три),

`B = SQR((B^3))` (переменная B теперь равна корню квадратному из своего старого значения, возведенного в куб).

Мы уже использовали некоторые «грамматические» правила QB в вышеуказанных записях. Отметим, что в QB арифметические (над числами) или алгебраические (над переменными) действия записывают так:

Сложение	$(a+b)$	=>	`a + b`
Вычитание	$(a-b)$	=>	`a - b`
Умножение	$(a \cdot b)$	=>	`a * b`
Деление	$(a:b)$	=>	`a / b`

Возведение в натуральную степень

(a^n) => `a ^ b`.

QB «понимает», что такое скобки и вложение скобок, а порядок выполнения операций такой же, как и в обычной алгебре (сначала возведение в степень, потом умножение или деление, и лишь затем сложение или вычитание). Рекомендую использовать скобки, если вы не совсем уверены в том, поймет ли QB ваши намерения по очередности выполнения операций. Также рекомендую не путать 2A и 2*A, ибо первое означает просто новую переменную, а второе — переменную A увеличенную вдвое. Итак, написав

```
A = 2
B = 2
C = A*B
```

мы получим, что в переменной C у нас окажется число 4 (= 2 x 2). Однако мы пока не умеем увидеть этот столь крупный результат. Именно для этого существует

4.2 Оператор PRINT

PRINT в переводе с английского означает «печатать». Таким образом, именно этот оператор позволяет возвращать на экран значения переменных, а также тексты.

Формат предъявления оператора PRINT:

PRINT [“Tekst”] [,][;] [variable, expression, symbol, number, etc.] [,][;] [“Tekst2”] де Tekst2 и Tekst2 — любой текст; variable, expression, symbol, number, etc. — все те числовые (или символьные) значения, которые вы хотите вывести на экран монитора. Стоящие между операндами знаки — запятая или точка с запятой — используются для того, чтобы на выводе между операндами или оставался табулированный промежуток (при использовании запятой), или же печать происходила с минимальным зазором (точка с запятой).

Отметим, что QB абсолютно равнодушен к тому, что заключено **между кавычками** — поэтому вы можете там печатать по-русски, по-китайски, по-амхарски (на амхарском языке говорят в Эфиопии) — QB все это в полной сохранности выведет на монитор как только до этого дойдет дело (управление), совершенно не вникая в смысл написанного вами.

Одна из наиболее популярных студенческих ошибок — не закрывать кавычки. Для того, чтобы такое неприличие не случилось именно с вами, послушайтесь моего совета. Все профессиональные кодировщики (те, кто пишет программные листинги) обычно сразу ставят обе кавычки, затем уходят курсором налево, вписывают нужный текст, потом нажимают клавишу {End} (это переводит курсор в конец строки) и пишут себе свою программку далее.

Вторая частая ошибка – это перепутанные буквы переменных. В процессе **отладки** программы (т.е. доведения программы до правильно работающего состояния) найти или вспомнить то место, где вы могли ошибиться, довольно кропотливо. Поэтому советую сразу смотреть на то, что вы пишите вместо переменных. В противном случае вы будете получать или нули вместо результата, или неприличные комментарии со стороны QB, вроде «попытка деления на нуль» (считается очень большим оскорблением среди программистов).

Еще одно дополнение: в QB вместо ввода строки

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

```
print "skorost' = ";v
```

достаточно написать

```
?"skorost' = "; v
```

и QB сам вместо «?» поставит оператор PRINT, наладит требуемые по формату пробелы и т.д. К сожалению, другие операторы он в таком виде не распознает.

4.3 Оператор CLS

Происходит от сокращения английских слов «clear screen» (очистить экран). Его формат предъявления очень прост — напечатайте CLS и … и все! Его действие сводится к очистке экрана, что бывает необходимо при начале очередного интерактивного (или диалогового) блока программы.

Этот оператор обычно ставится первым номером в самом начале программ.

4.4 Оператор END

В переводе с английского означает «конец». Формат предъявления таков же, что и у предыдущего оператора, но ставят его в конце **исполнения** (в листинге это может стоять и в середине, и даже весьма близко к началу листинга!) программы или используют для того, чтобы прервать программу в каком-то среднем месте в целях отладки.

Если в явном ли неявном виде нет ссылки на строки ниже той, где стоит END, то при попадании управления на строку с END этот оператор прерывает передачу управления на нижележащие строки. Отметим также, что END может попадать несколько раз, завершая отдельные ветви алгоритма программы, в которые она может, скажем, «попасть насовсем».

Еще два коротких оператора, которым не стоит посвящать слишком много места, это

REM (все, что стоит после него в строке, программой игнорируется) и

‘ — то же, что и REM, его можно ставить для написания комментариев.

Рассмотрим некоторые примеры программ.

Задача. Расстояние от города А до города В 820 км. Мы летим на самолете со скоростью 532 км/час. Написать программу, которая бы вычисляла время, затраченное на путь из А в В (без учета ветра, дождя, иных метеоусловий, приема пилотом таблеток «Саридон» и отсутствия керосина).

Решение:

```
CLS ' polet samoleta
s=820
v=532
t=s/v
print «vremya v puti »; s
end
```

Другое решение:

```
print «vremya v puti», 820/532
end
```

В первом решении есть ма-а-аленькая ошибочка. Отладьте!

Второе решение короче, но и … тупее. Потому, что мы далее его развить не сможем. В принципе, программка сама по себе не очень хороша тем, что довольно неинтеллигентно находит одно число как результат обработки двух заданных наперед. Единственно приятное заключается в том, что всякий может убедиться воочию в том, что компьютер не хуже чем калькулятор. Более того, если вы введете строку

```
print ((3.141502654/2)*(sqr(2))/(256 -512))^2
```

то[1)] в результате выполнения этой строки компьютер выдаст на мониторе

```
262135.1143094216
```

[1)] Очень надеюсь, что Вы понимаете причину отсутствия запятой перед словом "то", точки после числа, кончающегося на 216 чуть ниже и иных знаков препинания далее в тексте☺

Таким образом, в отличие от калькулятора, QB может выполнять го[illegible]ло более сложные цепочки действий. Между прочим, указанную строчку можно выполнить, не выгружаясь из программы которую вы набираете. Достаточно воспользоваться клавишей {F6}, и вы из основного окна текста программы в редакторе QB перейдете в нижнее окошко {Immediate}. Оно служит для непосредственного выполнения отдельных строк (вдруг вам надо вычислить что-то по ходу написания программы!). Именно это окошко вполне заменяет и калькулятор.

4.5 Оператор ввода данных INPUT

Input в переводе с английского означает «ввести». Оператор служит для ввода (в таком виде — с клавиатуры) данных по запросу программы. Формат его

```
INPUT ["Tekst"] [,][;] variable
```

где Tekst между кавычками это тот же объект, что использовался в операторе PRINT. Однако оператор PRINT может печатать тексты несколько раз, а у INPUTа такая возможность одна и она всегда завершается именем переменной, к которой будет приписано введенное число или символьная строка. Потому текст просто обязан прокомментировать, что же вы просите ввести. Vartiable, как мы уже видели, это переменная.

Примеры:

```
input "vvedi argument x"; x
```

или

```
input "vvedi familiyu x"; x$
```

В первом случае будет запрошено число, во втором — символьная величина.

Еще один пример:

```
input "vvedi svoyo imya "; x$
print "Privet,";x$;"!"
```

4.6 Операторы перехода GOTO и IF … THEN

Оператор GOTO происходит, очевидно, от «go to» (идти к). Формат его предъявления таков:

```
GOTO Label
```

где Label означает метку той строки, на которую осуществляется переход. Метка может быть просто натуральным числом или символьным выражением, завершенным двоеточием, например, «nachalo:», «faktorial:» или «korni:».

Управление выполнением программы по этому оператору (или, говорят, по предъявлению этого оператора, т.е. когда оператор появится в начале строки или, после двоеточия, даже в середине ее или конце) **безусловно** переходит на строку помеченную меткой. Таким образом, если оператор перехода посылает на метку строки **выше себя,** и между такой меткой и им самим нет ничего, что может помешать управлению снова добраться до этого GOTO, то мы будем иметь классический пример **зацикливания**.

Если же этот оператор посылает управление куда-то ниже себя, то далее управление может подняться выше метки (или ссылки) только если в программе используется еще какой-то переход на такую (выше метки) строку.

Теперь обратимся к более активному оператору, смысл которого заключен в передаче управления или выполнении действий в зависимости от состояния условия перехода или параметра перехода.

Слово IF переводится как «если», а THEN — как «то» или «тогда». Краткий формат оператора:

```
IF condition THEN action
```

где condition означает условие, например, raznost=1, i<10, или k и т.д., а action — действие (например переход на строку с определенной меткой, или печать, или что-то еще).

**** ный формат этого оператора предусматривает множественность и условий и действий:

```
IF condition1 THEN
        [statementblock-1]
[ELSEIF condition2 THEN
        [statementblock-2]]
… … … …
        [ELSE
                statementblock-n]]
END IF
```

или

```
IF condition THEN statements [ELSE statements]
```

Здесь под

```
condition1
```

подразумевается любое выражение, которое можно оценить как истинное (ненулевое) или ложное (нулевое)

```
statementblock-1
        statementblock-2
statementblock-n
```

подразумевается использование одного или более оператора на одной или более строках, а под

```
statements
```

подразумеваются один или более операторов, разделенных двоеточием.

Пример:

```
INPUT "1 or 2? ", i%
      IF i% = 1 OR i% = 2 THEN
      PRINT "OK"
            ELSE
      PRINT "Out of range"
      END IF
```

Эта программка просит ввести 1 или 2. Если вы слушаетесь и вводите 1 или 2, она печатает «ОК», в противном случае сообщает вам «Вне пределов».

Возвращаясь к путешествию из Казани в Москву на аэроплане, напишем программу, которая не только запрашивает расстояние и скорость полета, но и выдает некоторые сообщения при неправдоподобии вводимых величин.

```
20 CLS ' polet samoleta
print "     polet samoleta"
30 input "vvedi rasstoyaniye ";s
if s>10000 then print "a viza est'?": goto 30
if s<30 then print "peshkom doidesh'!": goto 30
40 input "vvedi skorost' ";v
if v>1200 then print "Gde brali MiG-24?": goto 40
if v<70 then print "voz'mite velosiped": goto 40
t=s/v
min=(t-int(t))*60
sec=(min-int(min))*60
print «vremya v puti»
print     int(t);"chasov",int(min);"minut";     sec;
"sekund"
input «Eshcho poletaem? Da - vvedi odin. Net -
drugoye chislo», ask
if ask = 1 then goto 20
```

```
cls
print
print
print
print «Schastlivogo puti!»
end
```

Написав и отладив программу, вы освоите азы использования оператора IF…THEN.

Задача. Попробуйте использовать иной формат оператора IF…THEN.

Вопросы. 1) Что означает переменная ask в последней части программы? **2)** Не запуская программу, объясните как она будет работать. **3)** Почему в самом начале стоит метка 20 и для чего она нужна? **4)** Почему мы пишем min=(t-int(t))*60 и что это нам дает? **5)** Зачем нужна функция int(t) и как она действует на переменную t?

Задача. Написать программу, которая бы решала квадратные уравнения, т.е. запрашивала значения коэффициентов уравнения и печатала его корни во всех известных случаях.

Решение.

```
nachalo: CLS 'programma dlya resheniya kv. ur-ya
print " Reshaem kvadrantoya uravneniye"
print "  2"
print "ax  + bx + c = 0"
input "vvedi a=";a
input "vvedi b=";b
input "vvedi c=";c
dskr=b^2-4*a*c
ch1=-b/(2*a):ch2=(sqr(abs(dskr))/(2*a)
if dskr>0 then gosub real
if dskr=0 then gosub odin
if dskr<0 then gosub compl
input "eshcho reshem? DA=1. Net=2"; vybor
```

```
if vybor=1 then goto nachalo
cls
print "spasibo za ispolzovaniye porgammy"
end
real: print "Imeem 2 veshch. kornya,
print "x1=";ch1-ch2,"x2=";ch1+ch2
return
odin: print "Imeem 1 veshch. koren',
print "x1=x2=";ch1
return
compl: print "Imeem 2 compleksnykh kornya,
print "x1=";ch1;"-i*";ch2,"x2=";ch1;"+i*";ch2
```

Внимательный читатель увидит здесь два новых оператора — `GOSUB` вместо `GOTO` и tot какой-то оператор `RETURN`. Оператор `GOSUB Label` не просто отсылает управление на строку с меткой `Label`, но и ждет затем появления своей второй половины — строки `RETURN` (от английского «вернуться»).

Такой переход похож на безусловный, но обладает дополнительным свойством: управление не просто передается на строку `Label:` , но и спускается все ниже до тех пор, пока не появится строка `RETURN`. После этого управление передается на строку, сразу следующую за той, на которой стоял `GOSUB` и даже на тот оператор, который стоит на той же строке, что и `GOSUB`, но сразу после (`GOSUB:`) — двоеточия! Вообще говоря, `GOSUB` происходит от `GO` to `SUBroutine` (routine = программа, subroutine = подпрограмма).

На возможный вопрос «А чего это мы `END` в середине программы поставили?» ответ будет таким. Оператор `GOSUB...RETURN`, вообще говоря, является оператором перехода на подпрограмму (т.е. относительно самостоятельный блок программы). Для того, чтобы выполнение и адресация на подпрограммы была разумной, следует выносить их НИЖЕ ОСНОВНОГО БЛОКА программы. Они (подпрограммы) становятся такими самостоятельными программками, у которых «голова» находится в теле большой программы, а

«концом» является как раз оператор RETURN, который, фактически, завершает выполнение подпрограммы и возвращает управление на то место, с которого его передали вниз GOSUBом. Ниже мы приведем пример более слабого решения, в котором GOSUB не используется.

```
nachalo: CLS 'resheniye kv. ur-ya
print " Reshaem kv. ur-e
print "  2"
print "ax  + bx + c = 0"
input "vvedi a=";a
input "vvedi b=";b
input "vvedi c=";c
dskr=b^2-4*a*c
ch1=-b/(2*a):ch2=(sqr(abs(dskr))/(2*a))
if dskr>0 then
print "2 veshch. kornya"
print "x1=";ch1-ch2,"x2=";ch1+ch2
elseif dskr=0 then
print "Imeem 1 veshch. koren',
print "x1=x2=";ch1
elseif dskr<0 then
print "Imeem 2 compleksnykh kornya,
print "x1=";ch1;"i*";ch2,"x2=";ch1;"+i*";ch2
input "eshcho reshem? DA=1. Net=2"; vybor
if vybor=1 then goto nachalo
cls
print "spasibo za ispolzovaniye porgammy"
end
```

Ура, мы поместили подпрограммы внутри программы! Увы, это лишь потому, что подпрограммы очень простенькие… Что выбирать? Ответ такой же, как и со знанием иностранных языков — нужно знать и то, и другое!

5. Организация циклов

Для того, чтобы пояснить, что такое цикл, рассмотрим такой пример.

Говорят, что когда-то великий математик Эйлер был маленьким и учился в школе. В один прекрасный день преподаватель математики, имея плохое состояние или настроение, начал урок с задания: «А сегодня, господа ученики, вы должны просуммировать все числа от 1 до 100» и уселся в креслах. Маленький Эйлер через несколько минут поднял руку: «Господин учитель, а результат 5050?». Учитель посмотрел в свои записи и спросил: «А как ты это сосчитал?». Эйлер ответил: «Если записать все числа от одного до ста, а потом снизу приписать их же, но только наоборот, то в каждом столбике будет 1+100=2+99=3+98=…=100+1=101. Наверху мы всякий раз добавляем единицу, а внизу отнимаем, поэтому сумма столбика остается той же. Таких столбиков 100, значит, все вместе будет 101*100. Но так как мы брали две одинаковые строчки, то сумма чисел от 1 до 100 будет половина от этого, т.е. ((1+100)*100)/2=5050».

Говорят, правда, что числа были не от 1 до 100, а еще похуже. Однако, разобрав мысли Эйлера, мы можем заметить, как и он, что всякое новое число увеличивается на единицу. Мы сейчас смогли бы написать программу, которая считает числа от 1 до 100, например:

```
SUM=1+2+3+4+5+6+7+8+9+10+…
print SUM
```

Н-да… длинновато получается. Проще на счетах. Хорошо, давайте установим некоторое накопительное повторяющееся выражение

```
SUM =SUM+1
```

Тогда в ящик `SUM` при каждом исполнении такой строки будет добавляться единица. Но нужно уметь … остановиться, то есть сделать эту операцию ровно сто раз. Здесь и возникает так называемый «счетчик». Его можно строить так: как только операция добавления единицы в переменную `SUM`

вершается, так счетчик растет ровно на единицу. При достижении им счетчиком значения 100 действия прекращаются.

```
i=0
20 i=i+1
sum=sum+1
if i<100 then goto 20
print sum, i
```

Или иначе

```
i=101
20 i=i-1
sum=sum+1
if i>0 then goto 20
print sum, i
```

Стоит ли говорить, что программки не работают правильно? Отладить!

В последнем случае возникает вопрос: а сто ли раз выполняется действие суммирования? Попытайтесь ответить на него, просто **подумав**, и не загружая компьютер. Затем придумайте способ выяснить ваши подозрения каким-то образом на PC.

Еще один интересный объект для нахождения — это факториал. Факториал натурального числа n определяется как произведение всех натуральных чисел от одного до этого заданного числа, т.е.

$$n! \equiv 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-2) \cdot (n-1) \cdot n \, .$$

Роль факториала очень велика (вы, вероятно, помните, что без него не обходились ни формула Тейлора, ни коэффициенты бинома Ньютона, ни определители).

Докажем, что число различных перестановок в наборе из n элементов равно $n!$. Применим метод математической индукции, суть

которого в том, что доказательство справедливости какого-то предложения, скажем $\pi(n)$, для любого n можно свести к трем действиям:

1) Проверить, справедливо ли $\pi(1)$ или другом начальном n?
2) Если да, то, предположив, что предложение верно при некотором $n = k$, ДОКАЗАТЬ, что тогда оно будет выполняться и при $n = k + 1$.
3) Рассуждения в последнем, третьем пункте довольно просты: по пункту 1) имеем, что предложение справедливо при $n = 1$; но по 2) оно верно и при $n = 2$, следовательно и при $n = 3$ и так далее до бесконечности.

В нашем случае, обозначив за $P(n)$ число перестановок, имеем с очевидностью $P(1) = 1$ (в наборе из одного элемента и вариантов-то всего один). Допустив, что при $n = k$ мы имеем $P(k) = (k+1)!$, получаем, что с добавлением одного нового элемента в набор этот элемент будет на первом месте участвовать в $k!$ перестановках остальных членов, на втором также в $k!$ перестановках и т.д., всего в $(k+1)k! = (k+1)!$ перестановках. По третьему пункту индукции, получаем то, что требовалось доказать.

Число разных комбинаций выхода 6 волейболистов на площадку равно 6! = 1·2·3·4·5·6 = 720 (не правда ли, впечатляет!). Напомню, что число различных вариантов рассадки 25 студентов на 25 местах в одной и той же аудитории как раз равно 25!, что представляет величину 15512100433309859840000000 $\approx 1.55 \cdot 10^{25}$. Это больше, чем число Авогадро $N_A = 6.0221367 \cdot 10^{23}$ mol^{-1}.

Напишем программу для вычисления факториалов. Это то же самое, что и суммирование ряда натуральных чисел, только нужно не добавлять единицу, а домножать на «счетчик».

```
i=0
fkt=1
20 i=i+1
```

```
fkt=fkt*i
if i<25 then goto 20
print fkt, i
```

Программа, как ни странно, работает. Однако дефект программы проявится, если вы в условии перехода возьмете большое число. Так, например, при числе 34 проблем нет, а уже при 35 вы получите сообщение о переполнении (overflow). Для того, чтобы устранить данный недостаток нужно прибегнуть к дополнительным определениям переменной. Хорошие программисты всегда определяют тип переменной и иные ее параметры. Об этом ниже.

5.1 Оператор DIM

Английское слово DIMENSION, от которого и берется сокращение DIM, означает «размерность», «размер». Этот оператор нужен для уточнения и определения переменных, с которыми работает программа. Дело в том, что по умолчанию (так называемый default, который нехорошо ассоциируется с датой 19 августа 1998 года) QB пользуется обычными переменными одинарной точности. Если, скажем, нам понадобятся однопараметрические переменные A(I), то он их нам даст штук 10-12, а затем объявит об overflow (переполнении). Вообще говоря, оператором DIM объявляют массивы (об этом чуть позже) или определяют тип данных для немассивной переменной. Второй оператор REDIM, который можно считать дополнительным видом оператора DIM, заявляет параметры или преобразовывает параметры динамических массивов, снимая предыдущие ограничения. Формат операторов таков:

```
DIM [SHARED] variable[(subscripts)] [AS type]
      [,variable[(subscripts)] [AS type]]...
REDIM [SHARED] variable(subscripts) [AS type]
      [,variable(subscripts) [AS type]]...
```

Дадим некоторые комментарии. Здесь

SHARED указывает на то, что записанные за ним перем... разделяются также и со всеми подпрограммами или процедурами функций в создаваемом и исполняемом модуле.

variable это имя переменной или массива.

subscripts размеры массива, указываемые в формате:

[нижний TO] верхний [,[нижний TO] верхний]...

Параметр нижний ставят для указания нижнего предела индекса массива. По умолчанию это нуль. Верхний указывает на верхний предел для данного индекса массива.

AS type объявляет тип данных массива или переменной (INTEGER, LONG, SINGLE, DOUBLE, STRING, или же указывается тип данных, определяемых самим пользователем).

В операторе DIM объявляют также и характер (статический или динамический) массива. Если способ хранения массива не указывается явно как $STATIC, $DYNAMIC, или COMMON, массивы с цифрами вместо индексов являются статическими, а массивы в DIM с символьными переменными являются динамическими. Оператор REDIM всегда объявляет динамические массивы. Режим статических массивов автоматически заявлен при запуске программы и таким и остается. Хранение динамических массивов происходит в памяти машины во время выполнения программы.

Вернемся к программе, считающей факториал. Добавим в ней такие строки

```
dim fkt as double
i=0
       fkt=1
       20 i=i+1
       fkt=fkt*i
       if i<25 then goto 20
print i;"! = "fkt
end
```

Вот теперь для того, чтобы заставить программу «заткнуться» от переполнения, нужно взять весьма большое число, например 171.

Еще одно дополнение. Для экономии памяти машины, часто используют такую возможность, заложенную в QB, как приписывание знака % (процент) к имени **числовой** переменной для того, чтобы она была заявлена в программе, как целое число. Таким образом, наша последняя программа с учетом такого способа описания целых переменных будет выглядеть так:

```
dim fkt as double
i%=0
       fkt=1
       20 i%=i%+1
       fkt=fkt*i%
       if i%<25 then goto 20
print i%;"! = "fkt
end
```

Отметим, что циклы весьма важны в написании программ, без них невозможно вводить и считывать массивы данных, вычислять многие функции, вводить списки и т.д. Именно важностью циклов при написании программ можно объяснить тот факт, что почти во всех языках существуют специальные операторы для организации циклов. В QB такой оператор выглядит как

5.2 Оператор FOR … NEXT

Слова FOR … NEXT переводятся как «для» … «следующий». Формат оператора таков

```
FOR counter = start TO end [STEP increment]
                    [statementblock]
NEXT [counter [,counter]...]
```

Здесь

`counter`	числовой счетчик, указывающий на количество необходимых операций (повторений),
`start и end`	начальное и конечное значения счетчика,
`increment`	шаг, с которым счетчик должен изменяться за один проход цикла,
`[statementblock]`	блок операторов (в которых переменная счетчика должна использоваться).

Отметим, что начальное и конечное значения, как и собственно шаг, могут быть и (целыми) отрицательными величинами. Для вычисления нашего факториала этот оператор придает программе более простой и прозрачный вид:

```
' vychislenie faktoriala, n!=1*2*3*…*n
dim fkt as double
input "vychilyaem n!; vvedi n (<171)=",n%
fkt=1
for i%=1 to n%
fkt=fkt*i%
next i%
print i%;"! = "fkt
end
```

Программа вычисляет факториалы весьма больших чисел (до 170!). Такую организацию циклов можно использовать для очень многих целей. Ниже следуют примеры того, как можно использовать уже известные нам операторы для решения разных задач и нахождения различных величин.

Задача. Найти сумму квадратов первых N членов натурального ряда чисел.

Решение.

```
cls
dim sum as double
input "vychilyaem summu kvadratov N chisel, N=? ",n%
for i%=1 to n%
       sum=sum+i%^2
next i%
```

```
print "summa kvadratov ravna"; sum
end
```

Задача. Распечатать таблицу синусов углов от 0^O до 45^O с шагом 5^O.
Решение.

```
cls
print "      Tablitsa sinusov "
print "ugol x (grad), sinx "
for i%=0 to 45 step 5
      print "-----------------------------------"
print i%,,sin(i%*3.141592654/180)
next i%
print "-----------------------------------"
end
```

Давайте попробуем решить одну задачу, настолько же важную, насколько важен для вас ответ на вопрос «Как вычисляют различные функции, например, синусы и косинусы в таблицах Брадиса?». Честно говоря, мне кажется, что понять каким образом вычисляется, например, экспонента (т.е. функция $y = e^x$ или $y = \exp x$, как иногда ее записывают) очень важно для понимания всей математики. Если вы вспомните формулу Эйлера

$$z = |z| \cdot e^{\cos\varphi + i\sin\varphi},$$

где $\varphi = \arg z$, то станет ясной та глубокая внутренняя связь, которой связаны второй замечательный предел и вся тригонометрия! Таким образом, ответив на вопрос об экспоненте, мы сможем вычислять и тригонометрические функции.

Формула Тейлора, а вернее ее частный случай — формула Маклорена, дает нам представление экспоненты степенным рядом

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots,$$

причем ряд этот сходится для всех вещественных x. В качестве приближенного значения можно взять некоторую частную сумму S_n этого ряда

$$S_n = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}.$$

Эта частная сумма приближает истинное значение экспоненты тем лучше, чем большее число n мы берем.

Задача. Написать программу, которая бы вычисляла $y = e^x$ или $y = \exp x$ с желаемой точностью и сверяла результат с тем, который выдает QB.

Решение.

```
' vechislenie eksponenty
input "Ishchem znachenie y=exp(x). Vvedi x=";x
input "vvedi stepen', do kotoroi schitetsya ryad,
n=";n%
dim fkt(n%+1) as double, res as double, a as double
for i%=0 to n%
gosub fct
res=res+(x^i%)/fkt(i%)
next i%
a=exp(x)
print "resul'tat =";res, "machina daet - ";a
end
fct:
fkt(i%)=1
for j%=1 to i%
fkt(i%)=fkt(i%)*j%
next j%
return
```

Задание. Напишите программы, которые бы вычисляли sin$x$ и cos$x$ с требуемой точностью и выводили результат, который дают встроенные в QB подпрограммы вычисления этих функций.

Десятичная запись чисел, как это нетрудно заметить, является наследием биологического присутствия пяти пальцев на наших конечностях. Поэтому те числа, с которыми мы имеем дело каждый день, абсолютно неестественны с точки зрения математики. С точки зрения информатики именно бинарный способ записи числа есть самый натуральный способ записи. Кроме того, еще более неестественно делить полный круг именно на 360 *частей (градусов). Поэтому весь математический анализ функций подразумевает, что их аргументы берутся в радианах. Радианная мера угла столь естественна, что об этом косвенно говорит сам факт отсутствия обозначения этой единицы — с математической точки зрения радианы есть самая натуральная числовая мера*

ла. Именно поэтому все аргументы всех тригонометрических функций в [illegible] по умолчанию берутся в радианах.

Обратимся к достаточно сложной (но лишь на первый взгляд!) задаче численного интегрирования функций.

Задача. Вычислить интеграл

$$\int_a^b \sin x dx .$$

Решение. Напомним, что определенным интегралом от функции $f(x)$ по промежутку называют предел всевозможных интегральных сумм $S_N = \sum_{i=1}^{N} f(\xi_i) \cdot \Delta x_i$ по всем λ -разбиениям при стремлении длины наибольшего отрезка разбиения к нулю (и, следовательно, при *N* стремящемся к бесконечности. Частная интегральная сумма для некоторого λ -разбиения сегмента $[a;b]$:

$$a = x_0 \le x_1 \le x_2 \le \ldots \le x_{N-1} \le x_N = b ,$$

$$\Delta_i \quad x_i - x_{i-1}, = \lambda \quad \max_{1 \le i \le N} \Delta_i ,$$

строится так: на каждом элементарном сегментике $[x_{i-1}; x_i;]$, $i = \overline{1, N}$, выбирается точка ξ_i и вычисляется значение функции $f(\xi_i)$ в ней. Это значение умножается на ширину сегментика Δx_i, и затем все такие произведения складываются.

Для целей практического численного интегрирования удобнее разбивать весь сегмент интегрирования не произвольно, а равномерно, т.е. на *N* равных частей. Все Δx_i окажутся равными некоторой величине $h = (b - a)/N$ и поэтому это число, уже не зависящее от индекса суммирования, можно вынести за знак суммы как постоянный сомножитель каждого слагаемого.

Далее, в качестве точки ξ_i также удобно брать, например, самую левую точку x_{i-1} или самую правую точку x_i элементарного сегментика. Интегральные суммы, получающиеся при таком выборе значений функции, назовем также левой или правой. Отметим, что при условии возрастания функции на сегменте интегрирования левая сумма становится минимальной для данного разбиения, т.е. нижней суммой Дарбу, в то время как правая сумма есть верхняя сумма Дарбу.

При убывании функции все будет с точностью до наоборот. Как изв... критерием существования определенного интеграла является равенство пределов нижних и верхних интегральных сумм Дарбу при неограниченном увеличении числа точек в λ-разбиении и потому стремлении λ к нулю. Как известно, при увеличении числа точек, участвующих в разбиении, нижняя сумма Дарбу не убывает, а верхняя не возрастает. Более того, для вычисления определенного интеграла можно воспользоваться формулой трапеций, в которой каждое элементарное произведение в интегральной сумме заменяется на площадь трапеции (вертикально поставленной на одну боковую сторону), основаниями которой будут служить значения функции соответственно в левой и правой точках элементарного сегментика. Помня, что ее площадь равна произведению полусуммы оснований на высоту (а высота — это ширина *h* нашей узкой трапеции), мы замечаем, что в качестве результата формулы трапеций при равномерном разбиении сегмента интегрирования можно взять полусумму левой и правой интегральных сумм. Таким образом, для вычисления определенного интеграла от синуса по произвольному промежутку по формуле трапеций мы можем составить следующую программу:

```
cls
print "      Vychislenie integrala"
print "            b"
print "            ";chr$(218) '217? 179
print "            ";chr$(179); " sinxdx"
print "            ";chr$(217)
print "            a"
```

Эта часть программы, которую иногда называют «шапкой», создает вводную часть программы. Мы используем функцию `chr$(x)`, которая возвращает на экран символ, отвечающий его десятичному коду `x`.

Можно было бы написать интеграл и получше, но проблема в том, что при наличии руссификации в системных файлах autoexec.bat и config.sys вторая половина таблицы из 256 выводимых на экран ASCII кодов (Anerican Standard Codes for Information Interchange, произносится как «аски») содержит руссифицированные символы взамен дополнительных символов (которые по

старой памяти иногда называют «псевдографикой»), которые бы позволил... м создать более точное изображение знака интеграла.

Далее, введем значения трех параметров, которые и определят результат работы программы:

```
input "vvedi a";a
input "vvedi b";b
input "vvedi kolichestvo razbienii, n="; n%
```

Определим с двойной точностью числовые переменные для того, чтобы результат был более надежным:

```
dim x(n%+1) as double, y(n%+1) as double, h as double
dim sum as double, lsum as double, rsum as double,
dim lint as double, rint as double, int as double
```

Затем приступим к вычислениям. Помня, что нам нужны и левая, и правая интегральные суммы, мы сначала найдем сумму всех значений функции на всех $n + 1$ точках разбиения.

```
for i%=0 to n%
x(i%)=a+i%*h
y(i%)=sin(x%) 'zdes stoit sama funnktsiya!
sum=sum+y(i%)
next i%
```

Теперь можно вычислить левую и правую суммы значений:

```
lsum=sum-y(n%)
rsum=sum-y(0)
```

и затем собственно левую и правую интегральные суммы и их среднее

```
lint=lsum*h
rint-rsum*h
int=(lint+rint)/2
```

Теперь можно печатать результат:

```
print "zpnachenie integrala ravno", int
end
```

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

В качестве проверки считается интеграл

$$\int_0^{\pi/2} \sin x dx = 1,$$

в котором величина π берется как 3.1415926536 (Можно взять и большее количество знаков в выражении $\pi \approx 3.14159265358979323846264 3$). Нам нужно подставить половину этого числа вместо верхнего предела *b*. Как это сделать, не выходя из QB для вызова, например, стандартного калькулятора в Windows? Ответ нужно искать на нижней строке интерфейса QB, где имеется подсказка:

```
<Shift+F1=Help> <F6=Window> <Esc=Cancel>
   <Ctrl+F1=Next> <Alt+F1=Back>
```

Именно клавиша F6, выделенная выше полужирным шрифтом, позволяет перейти из окна редактора в нижнее окошко “Immediate” (сразу, непосредственные вычисления). Однако следует помнить, в этом нижнем окошке можно вводить только одну строку, например:

```
print 3.1415926536/2 {Enter}
```

Появляющийся при этом экран может показать все предыдущие выведенные значения, но самым последним будет именно требуемое число. Запишите его на листочек, перейдите в окно редактора QB, нажав еще раз F6, и затем смело вводите полученный результат.

Задание. Написать программу, в которой интегральные суммы вычислялись бы со значениями функции в серединах элементарных сегментиков равномерного разбиения.

6. Работа с массивами. Одномерные (однопараметрические) массивы

Массивом («array» по-английски) называют упорядоченное множество однородных данных. Тот параметр (или несколько параметров), который упорядочивает массив, называют его *параметром* (или *параметрами*). Количество данных, входящий в массив, называют его *размером*, а количество параметров, определяющих характер массива, называют его *размерностью*. Например, данные метеорологических ежедневных четырехкратных наблюдений

мпературы воздуха за пять лет представляют собой *одномерный м...* определяющим *параметром* которого является время (или номер) наблюдения. Нетрудно подсчитать, что данных в таком массиве будет чуть больше 7300 (нужно правильно учесть високосные годы!), т.е. именно таков его *размер*.

Вычисление среднего значения температуры за год или же, скажем, за все время наблюдений займет у вас достаточно большое количество времени. При этом, если потребуется найти среднесуточную температуру июля или января какого-то года, вам придется отыскивать, складывать и делить такие данные отдельно взятым образом. А если к тому же записываются еще и давление, влажность, оценка облачности, количество осадков, скорость ветра и его направление, да еще и фамилия наблюдателя? А если эти данные нужно обработать по 80 точкам наблюдений? А если нужно заодно и провести научный статистический (факторный, групповой, латентно-контекстный) анализ результатов? А если обрабатывать данные нужно для территории и быстро получать результаты для точного суточного прогноза?

Для облегчения обработки результатов и достижения большей точности и достоверности вычислений нужно прибегнуть к естественному и совершенно незаменимому помощнику — компьютеру. Именно компьютеры позволяют не только записать введенные массивы и точно обработать их, но и сохранить данные или их наращивания и последующей обработки, или же для более глубокого их анализа, построения новых моделей и т.д.

Например, чем плоха идея о том, что характер погоды за год определяется тем, на какое время выпадает празднование Пасхи или Науруза (другими словами, характер лунного обращения вокруг Земли и поэтому распределения по временам года лунного притяжения и приливов вполне может сказаться на характере погоды!)? Говорят же, что год был засушливым или дождливым, а зима снежной и/или теплой, или, наоборот, суровой. Для проверки такой гипотезы нужно «перелопатить» огромное количество данных метеорологических наблюдений, распределить или перераспределить выборки тем или иным образом, провести строгий статистический анализ и т.д. и т.п.

Помимо числовых, существуют также и символьные, и смешанные (символьно–числовые) массивы (например, списки избирателей, цены на товары,

казатели курсов акций на разных биржах и т.д.). Ниже рассмотрим то, … м образом можно организовать работу с массивами в QB.

Одномерный массив, то есть некоторое конечное множество $\{x_1, x_2, x_3 \ldots x_{N-2}, x_{N-1}, x_N\}$ числовых данных общего размера N можно записать в оперативную память компьютера как значения некоторой однопараметрической переменной (отметим, что сразу же возникает естественная задача о том, как эти данные сохранить для того, чтобы не вводить их всякий раз, как стартует программа). Для этого просто используется цикл с оператором input, который всякий раз запрашивает у вас ввод числового (или символьного) значения и присваивает введенное значение параметрической переменной при пробегаемом в данный момент цикла значении параметра.

Задача. Ввести числовой одномерный массив размера N и найти его среднее значение.

Решение.

```
'vvod massiva i nakhozhdenie srednego
input "vvedi razmer odnoparametricheskogo massiva, N"; n%
dim a(n%) as double
for i%=1 to n%
print "a(";i%;")=":input a(i%)
next i%
for i%=1 to n%
sum=sum+a(i%)
next i%
srd=sum/n%
print "srednee massiva ravno"; srd
end
```

Более элегантное решение с одновременной проверкой вводимых величин можно записать в виде:

```
cls  'vvod massiva i nakhozhdenie srednego
input "vvedi razmer N odnomernogo massiva "; n%
dim a(n%) as double
for i% = 1 to n%
20 print " a("; i%; ")="; : input a(i%)
print "Vy vveli a("; i%; ")="; a(i%);
print "Pravil'no? Enter (y/n)";
40 input resp$
if len(resp$)> 1 then ' ogrnichitel dliny vvoda
print "po-vimatelnee! <<y>> ili <<n>>?": goto 40
if lcase$(resp$) = "n" then
print "vvedem zanovo...": GOTO 20
elseif lcase$(resp$)<> "y" then
print "otvechaite po-vnimatel'nee!...": goto 40
end if
sum = sum + a(i%)
next i%
print "vvod zavershen, spasibo!"
srd = sum / n%
print "srednee massiva ravno"; srd
end
```

Две новые функции, использованные в данной программе достаточно просты и понятны. Функция `len(variable$)` возвращает длину символьной строки (стринга) `variable$`, поэтому в строчке с комментарием срабатывает ограничение «не больше одного символа», а функция `lcase$(variable$)` указывает на нижний регистр введенного символа.

7. Запись массива в файл и чтение из файла

Введенный по ходу программы массив обычно нуждается в сохранении для последующей обработки или хранения, переноса на другой компьютер, отсылки, отчетности и т.д. Произвести сохранение введенных данных можно при помощи специальных операторов открытия и закрытия файлов для записи или чтения . Их форматы таковы

```
OPEN file$ [FOR mode][ACCESS access][lock]AS
[#]fileno% [LEN=reclen%]
```

Здесь опции операндов таковы:

`file$` Имя файла или устройства. Имя файла может включать также и полный путь к нему.

`mode` Один из режимов: `APPEND` (присоединить), `BINARY` (бинарный ввод/вывод), `INPUT` (ввод или считывание из файла), `OUTPUT` (вывод или запись в файл), или `RANDOM` (открытый или произвольный доступ).

`access` В сетевом окружении указывает на доступ к файлу для операций `READ` (прочитать), `WRITE` (записать), или `READ WRITE` (и то, и другое).

`lock` В сетевом окружении указывает на тип блокировки (доступность/недоступность) файла: `SHARED` (разделенный ресурс), `LOCK READ` (закрыт для чтения), `LOCK WRITE` (закрыт для записи), `LOCK READ WRITE` (закрыт и для того, и для другого).

`fileno%` Натуральное число в пределах от 1 до 255 идентифицирующее открытый файл в программе.

`reclen%` Для файлов оперативного (случайного) доступа означает максимально возможную длину одновременной записи (по умолчанию –128 байт). Для последовательно записываемых и считываемых файлов — число символов в буфере (по умолчанию – 512 байт).

Однако одного оператора `open` еще не достаточно для работы с файлом. С одной стороны, всякий открытый для чтения и/или записи файл нужно рано или поздно закрыть. Для этих целей в конце того блока операторов программы, в котором вы совершили с файлом все, чего вы хотели, ставится оператор `close`. `close` без каких либо операндов закрывает все открытые файлы. Если необходимо закрыть именно те файлы, которые идентифицированы в программе как `#1` и `#12`, то нужно указать их идентификаторы как операнды через запятую после оператора `close`.

Предположим, что мы ввели некоторый массив. Как записать его в файл с именем `mass.dat`? Для этого применяется уже знакомый нам оператор `print` в формате

```
print#n%, "text", const, variable, variable$
```

Таким образом, указание «#n%,» распознается этим оператором как «почтовый адрес» записи данных или текста. Отметим, что при записи данных в файл разделители «запятая» и «точка с запятой» работают аналогично тому, как это происходит с выводом информации на экран монитора оператором `print`, т.е. разделитель операндов «запятая» организует запись с табуляцией (отделение пробелами фиксированной длины).

Чтение данных из файла с массивом в оперативную память машины осуществляется при помощи тех же операторов открытия-закрытия файлов, но с другим блоком действий между ними и с выбором опции `for input` вместо `for output` в операторе `open`.

Рассмотрим еще один оператор или команду, которая позволяет осуществлять временный выход в операционную систему DOS. Формат этой команды

```
shell [commandstring$]
```

в котором

```
commandstring$
```

означает имя DOS-команды или bat-файла. Исполнение программы возобновляется сразу же после того как выполнена команда или bat-файл. Если операнд `commandstring$` опущен, то `shell` выводит вас в DOS сессию и отображает на экране DOS промптер. Завершить работу в DOS нужно вводом командной строки `exit`. Отметим, что данная команда весьма полезна, например, при сортировке файлов данных с помощью известной DOSовской команды `sort`. Но, на самом деле, считать, что вы вышли именно в ту DOS-сессию, из которой запускали QB неверно. На самом деле осуществляется новая эмуляция DOS с запуском нового командного интерпретатора command.com в новом разделе оперативной памяти. Если вы входили в QB из оболочки Norton Commander, то при правильном распределении ресурсов оперативной памяти вы даже сможете запустить еще один Norton Commander (не тот, из которого вы, скажем, запускали первый Norton Commander!). Более

того, можно запустить даже еще один QB (!) и попытаться из него снова [illegible] в еще один DOS.

Доходчивее всего, на мой взгляд, ситуация с выходом в DOS и возвратом из него интерпретируется одной старинной гравюрой, на которой изображена воображаемая небесная сфера и человек, просунувший голову через нее и выглядывающий наружу (а что это вы там делаете?); правда, композицию нужно слегка изменить — человек должен смотреть вовнутрь сферы извне… Так вот, выйдя по указанной команде (оператору) в новый DOS, вы сможете посмотреть, присутствует ли записанный вами файл mass.dat в каталогах (например, DOS-команда с маской `dir m*.dat`), можно даже просмотреть этот DOSовский файл командой `type m*.dat`. А затем вернуться в исполняемую программу уже упомянутой командой `exit`.

9. Двумерные и многомерные массивы различной природы. Запись, чтение

Перейдем к практической реализации того, о чем мы писали выше.

Задача. Написать программу, которая бы записывала в файл введенный массив данных {фамилия, оценка1, оценка2, оценка3, средн.оценка} на группу из 12 человек, печатала в конец файла среднюю оценку по всей группе и создавала упорядоченную копию такого файла[2)].

Решение.

```
cls    'vvod spiska dannykh (i ego sortirovka)
n%=12 'zameniv etu stroku, mozhno poluchit vozmozhnist'
'vvodit' spisok proizvil'nogo razmera…
dim a$( n%), o1(n%), o2(n%), o3(n%), osr(n%)
for i% = 1 to n%
print i%; " chelovek. Familiya?": input a$(i%)
print "otsenka1 ("; i%; ")="; : input o1(i%)
print "otsenka2 ("; i%; ")="; : input o2(i%)
print "otsenka3 ("; i%; ")="; : input o3(i%)
osr(i%)=(o1(i%)+o2(i%)+o3(i%))/3
```

[2] К сожалению, требовать безусловного выполнения этой части задания невозможно (см. след. сноску).

```
ext i%
print "vvod zavershen, spasibo!"
'obshchaya srednyaya otsenka po gruppe
for i%=1 to n%
ComOts=ComOts+o1(i%)+o2(i%)+o3(i%)
next i%
print "vash fail sokhranyaetsya kak <<spisok.dat>>
open spisok.dat for output as #1
print #1,"Familiya","ots1","ots2","ots3","sred."
print #1,
for i%=1 to n%
print #1,a$(i%),o1(i%),o2(i%),o3(i%),osr(i%)
next i%
print #1,"srednii ball po gruppe ="; ComOts/(n%*3)
close
shell "sort name$.dat" 3)
      exit
```

Очевидно, что для практических исследовательских целей обычно необходима гораздо более сложная обработка данных, чем та, что выполнена в вышеприведенной программе. Но наша «тактическая» задача — это научиться не только вводить, но и записывать, и считывать массивы данных, а эта цель вполне достигается этой программой.

9. Работа с векторами и матрицами

Как известно, операции с векторами и/или матрицами составляют основу самого большого числа современных математических действий и расчетов. Ясно, что всякий n-мерный вектор представляет собой однопараметрический числовой массив, а всякая вещественная $n \times m$–матрица есть двумерный (двухпараметрический) массив размера $n \cdot m$. Таким образом, ввести два вектора — это просто ввести два массива (одной и той же длины или размера). Ввод матрицы представляет собой ввод двухпараметрического массива, у которого размеры

[3] При эмуляции QB из среды Windows не все DOSовские команды проходят корректно. Поэтому не расстраивайтесь при получении сообщения «Incorrect DOS version». Можно обратиться к системному администратору или попытаться запускать QB в сеансе DOS, вызванном через «Пуск» и перезагрузку компьютера в DOS режим. Вообще говоря, рано или поздно нужно изучить Linux — эта операционная система, наследница UNIX'а, работает гораздо стабильнее.

трицы служат ограничителями соответствующих двухпараметрич[illegible] переменных в операторе dim.

Ниже мы приведем решение двух задач алгебраического характера.

Скалярное произведение $<\bar{a},\bar{b}> \equiv (\bar{a},\bar{b}) \equiv \bar{a}\cdot\bar{b}$ двух векторов $\bar{a}$ и $\bar{b}$ равно следующему выражению

$$<\bar{a},\bar{b}> \equiv \sum_{i=1}^{n} a_i \cdot b_i = a_1 \cdot b_1 + a_2 \cdot b_2 + ... + a_n \cdot b_n ,$$

где $a_i, b_i \in$ □ , $i \equiv \overline{1,n}$.

Задача. Написать программу, вычисляющую скалярное произведение двух векторов произвольной размерности *n*.

Решение.

```
cls
print "skalyarnoye proizvedenie <a,b> vektorov a i
b"
input "vvedi razmernost' vektorov n"; n%
dim a(n%), b(n%)
for i% = 1 to n%
       skl=skl+a(i%)*b(i%)
next i%
print "<a,b> = "; skl
   end
```

При желании можно вставить блок сверки/проверки вводимых данных.

Перейдем теперь к умножению таких важных алгебраических структур, какими являются матрицы. Если даны две (квадратные и одинаковой размерности $n\times n$)[4)] матрицы *A* и *B*, тогда под их *матричным произведением* понимают третью матрицу *C* той же размерности[5)], элемент c_{ij} которой есть обычное скалярное произведение *i*-ой строки первой матрицы *A* на *j*-ый столбец второй матрицы *B*. Иными словами,

[4] Можно посчитать и произведение $n\times m$–матрицы на $m\times p$ –матрицу, слегка подправив программу.

[5] Точнее, размерности $n\times p$, см. предыдущую сноску.

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{1n} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nj} & \cdots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2j} & \cdots & b_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{i1} & b_{i2} & \cdots & b_{ij} & \cdots & b_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{n1} & b_{n2} & \cdots & b_{nj} & \cdots & b_{nn} \end{pmatrix} \equiv$$

$$\equiv \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{1n} & \cdots & c_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{i1} & c_{i2} & \cdots & \boxed{c_{ij}} & \cdots & c_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{n1} & c_{n2} & \cdots & c_{nj} & \cdots & c_{nn} \end{pmatrix}.$$

Здесь помечены соответствующие строка, столбец и элемент произведения.

Задача. Создать программу для перемножения двух матриц и вывода на экран результирующей матрицы.

Решение. Вначале напишем блок ввода матрицы *A*

```
cls ' umnozhenie matrits
input "Vvedi razmernost' matrits A i B "; n%
print "Vvod matritsy A"
for i%=1 to n%
       for j%=1 to n%
       print "a(";i,j;")=": input a(i.j)
       next j%
next i%
print "Matritsa A vvedena"
```

Отметим, что блок ввода реализуется в форме двух вложенных циклов. В самом деле, открыв первый цикл по `i`, мы на каждом его шаге уже

по циклу в отношении `j (j=1 to n%)` вводим все элементы первой с[illegible], затем (когда `i=2`) всю вторую строку и т.д.

Блок ввода второй матрицы выглядит абсолютно также с той лишь разницей, что все вхождения букв `A` и `a` в операнды должны быть заменены соответственно на `B` и `b`. Поэтому

стоит воспользоваться таким преимуществом компьютера перед простой пишущей машинкой, как возможность выделения части текста ({Shift}+{←} *или* {→} *или* {End}), *удаления* ({Shift}+ {Delete} *или копирования* ({Ctrl}+{Insert}) *такой выделенной части в буфер* (*иногда говорят "забрать в карман"*) *и вывода* ({Shift}+{Insert}) *из буфера в текст. Подсказка по сочетанию клавиш дается в верхнем меню* Edit.

Не забывайте также, что грамотный человек будет перемещаться по тексту с использованием клавиш ({Home}, {End}) *и сочетания клавиши* {Ctrl} *со стрелками, перепрыгивая, таким образом, через целые слова, а не тащиться по текстовому полю «по-символьно».*

```
print "Vvod matritsy B"
for i%=1 to n%
       for j%=1 to n%
              print "b(";i,j;")=": input b(i,j)
next j%
next i%
print "Matritsa B vvedena"
```

Теперь можно устроить блок вычисления элементов произведения. Его также легче скопировать из блока ввода матрицы A с дополнительной редакцией и добавлением строк.

```
for i%=1 to n%
       for j%=1 to n%
              for k%=1 to n%
              c(i%,j%)= c(i%,j%)+a(i%,k%)*b(k%,j%)
              next k%
       next j%
next i%
```

Здесь мы видим уже три вложенных цикла (внутренний по внешний по `i%`, между ними — по `j%`). Их работа достаточно наглядно видна из самого построения.

Далее следует блок вывода результата, который также весьма близок к первому блоку ввода:

```
for i%=1 to n%
       for j%=1 to n%
       print c(i%,j%), ', ili ; vliayet na pechat!
       next j%
       'print
next i%
end
```

Хочется заранее предупредить от увлечения большими числами при вводе размерности матриц и других величин. Так, при вводе числа 5 вам придется вводить 25 чисел в каждую матрицу. Если же вы все-таки ввели что-нибудь большое, то прекратить выполнение программы можно двумя путями: 1) *нажатие* {Ctrl}+{C} *приводит к остановке передачи управления*; 2) *если машина* (*вашей собственной персоной, больше-то некому...*) *зациклена или программа «зависла», то нажмите* {Ctrl}+{Pause}.

Не забудьте также перед запуском уже исправленной версии программы нажать {Alt}+{R}, {R}, *иначе программа продолжит выполнение шагов старой* (*неисправленной*) *версии вашей программы.*

Если по запросу программы вам нужно ввести нуль, то достаточно просто нажать {Enter}.

Проверить программу для ее отладки проще всего при помощи следующего нехитрого приема: нужно ввести первую матрицу в форме последовательных чисел, начиная с единицы, а вторую матрицу ввести единичной (у такой матрицы все члены, кроме диагональных единиц, равны нулю). Результат должен быть равен первой матрице.

10. Решение уравнений методами последовательных приближени

Из курса математики известно, что достаточно большое число алгебраических уравнений может быть решено при помощи достаточно простого приема последовательных приближений. Если уравнение $F(x) = 0$ можно преобразовать к виду $x = \varphi(x)$, то, взяв в качестве начального значения какое-нибудь число x_0, можно получить $x_1 = \varphi(x_0)$, а затем перейти к любому новому шагу по правилу $x_i = \varphi(x_{i-1})$. Такие соотношения называют *рекурсивными* (или *рекурсией*, от recourse – возвращение), а входящие в него аргументы — приближенными решениями. Очевидно, что нам наиболее интересен процесс: $|x_i - x_{i-1}| \xrightarrow[i\to\infty]{} 0$, при котором ясно, что может существовать некоторое число x^*, $x^* = \lim\limits_{i\to\infty} x_i$, превращающее соотношение $x^* = \varphi(x^*)$ в тождество, т.е. являющееся точным решением нашего уравнения. Для выяснения того, сходится или нет последовательность приближенных решений, можно взять соотношение $r_i = \left| \frac{x_{i+1} - x_i}{x_i - x_{i-1}} \right|$. Если рано или поздно оно становится перманентно меньшим единицы, то это означает, что шаги уменьшаются. В тех случаях, когда мы можем подозревать, что решением может быть нуль или речь идет о величинах, близких к нулю, можно нахально заменить его на соотношение $R_i = \frac{1+|x_{i+1} - x_i|}{1+|x_i - x_{i-1}|}$, которое также указывает на хорошее поведение приближений при его значениях, меньших единицы.

Задача. Напишем программу для нахождения методом итераций решения уравнения, преобразованный вид которого таков

$$x = 3 + \mathrm{tg}(0,2x) - 0.9\sqrt{x}\ .$$

Решение

```
cls
'metod posled. priblizh.
print "Reshaem uravneniye, presobrazovannyi vid kotorogo"
print "x = tg(0.2x) - 0.9sqr(x)"
input "vvedi kolichestvo iteratsii"; n%
dim x(n% + 1) as double, r(n%) as double
INPUT "vvedi nachalnoye znachenie"; x(0)
FOR i% = 1 TO n% STEP 1
x(i%) = 3 + TAN(.2 * x(i% - 1)) - .9 * SQR(ABS(x(i% - 1)))
print "reshenie na"; i%; "iteratsii = "; x(i%);
print "shag ="; x(i%) - x(i% - 1)
print
next i%
print
print n%; "-oe pribizh.= "; x(n%);
print "s poslednim shagom="; x(n%) - x(n% - 1)
end
```

Рассмотрим еще один простой, но мощный метод решения уравнений, основанный на том, что при необходимости решить уравнение $f(x)=0$ для функции $y=f(x)$ можно сначала поискать такой сегмент $[a;b]$, что $f(a)\cdot f(b)<0$ (т.е. сегмент, на котором функция меняет знак). Дополнительным предположением для того, чтобы метод наверняка давал хотя бы одно решение, служит требование *непрерывности* функции $y=f(x)$ на сегменте $[a;b]$. Надеюсь, что вы сразу же узнали условия теоремы о нуле непрерывной функции? Помните, как она доказывалась? Пусть $f(a)>0$. Тогда, очевидно, $f(b)<0$. Сегмент $[a;b]$ делится пополам и вычисляется значение функции в его средней точке $c_1=(b-a)/2$. Если $f(c_1)=0,$ то результат уже получен. Если $f(c_1)>0,$ то $a_1=c_1$ и $b_1=b$, а если $f(c_1)<0,$ то $a_1=a$ и $b_1=c_1$. В новом сегменте $[a_1;b_1]$ процедура повторяется и получается или готовое решение (посередине), или новый сегмент $[a_2;b_2]$. Последний также может быть снова подвержен той же самой

процедуре, и т.д. На каждом шаге с номером i получается сегмент длины $(b-a)/2^i$. С ростом i он весьма быстро стремится к нулю. Поэтому (точнее, по лемме о стягивающихся отрезках), существует единственная точка $c = \lim_{i\to\infty} a_i = \lim_{i\to\infty} b_i$. И в этой точке, по нашему построению, мы имеем $[f(c) \ge 0] \bigcap [f(c) \le 0] \Rightarrow f(C) = 0$.

Вы уже должны были догадаться по всему вышеизложенному, каким именно способом будет действовать нужный алгоритм. Фактически, именно его мы и прописали в доказательстве теоремы.

Задача. Написать программу для решения уравнения

$$x^3 - 2x^2 + 5x - 4.1234567 = 0$$

методом «вилки» (деления отрезка пополам).

Решение. [6)]

```
CLS
?"Ishchem resheniye uravneniya x^3-2x^2+5x-4.1234567=0"
DIM x AS DOUBLE, y AS DOUBLE, z AS DOUBLE
'opredeleniye funktsii, izmenit' dlya drogoi zadachi!!!
DEF FNUser (x) = x * x * x - 2 * x * x + 5 * x - 4.1234567#
DEF FNNew (y, z) = y * z
INPUT "predely simmetrichnogo intervala [-A;A], A=?"; Dia
m% = INT(ABS(Dia))
DIM x(m%) AS DOUBLE, y(m%) AS DOUBLE
FOR i% = -m% TO m%
PRINT i%, FNUser(i%)
NEXT i%
```

[6)] Мы напечатали операторы “как положено”, т.е. прописными буквами. Вводить же их с клавиатуры следует в обычном режиме, т.е. без нажатия клавиши “**CapsLock**”. Кроме того, здесь использован новый оператор **DEF FN**, который позволяет определять функции пользователя. И даже для **нескольких переменных**!

```
) INPUT "dlya [a;b] vvedi a="; Left: INPUT "i b="; Rig
INPUT "opredeli kolichestvo iterastii N"; n%
REDIM a(n% + 1) AS DOUBLE, b(n% + 1) AS DOUBLE, c(n% + 1)
AS DOUBLE
a(0) = Left: b(0) = Right
CLS
PRINT "proverka godnosti intervala..."
IF FNNew(Left, Right) < 0 THEN CLS : PRINT "OK": GOTO 40
PRINT "Interval ne goditsya!!!"
GOTO 20
40 PRINT "interval goditsya!"
PRINT "Nazhmi <<PROBEL>> dlya prodolzheniya..."
DO
LOOP UNTIL INKEY$ = CHR$(32)   '032 is the ASCII code for
"Space"
GOTO 60
50 INPUT "opredeli NOVOYE kolichestvo iterastii N"; n%
REDIM a(n% + 1) AS DOUBLE, b(n% + 1) AS DOUBLE, c(n% + 1)
AS DOUBLE
a(0) = Left: b(0) = Right
60 FOR i% = 1 TO n%
c(i%) = (a(i% - 1) + b(i% - 1)) * .5
PRINT i%, c(i%), FNUser(c(i%))
GOSUB decision
GOSUB decision2
NEXT i%
PRINT "poluchili interval [an;bn]="; "["; a(n%); ";";
b(n%); "]"
PRINT "Znachenie funktsii v an ="; FNUser(a(n%))
PRINT "Znachenie funktsii v bn ="; FNUser(b(n%))
PRINT "Znachenie funktsii v seredine ="; FNUser((a(n%) +
b(n%)) * .5)
PRINT "Resheniye c="; (a(n%) + b(n%)) * .5
INPUT "Uvelichit' kol-vo iteratsii? Da=1/Net-drugoe
chislo", ask
IF ask = 1 THEN GOTO 50
PRINT "Zadacha zavershena!"
END
decision:
k = i%
IF FNUser(c(k)) = 0 THEN
PRINT "Resheniye polycheno! c="; c(k)
ELSE PRINT "Srazu ne popali!"
END IF
RETURN
decision2:
k = i%
IF (FNUser(c(k)) * FNUser(b(k - 1))) < 0 THEN
```

```
        b(i%) = b(i% - 1): a(i%) = c(i%)
ELSEIF (FNUser(c(k)) * FNUser(a(k - 1))) < 0 THEN
        a(i%) = a(i% - 1): b(i%) = c(i%)
END IF
RETURN
```

Как это видно из предложенного решения, две подпрограммы после оператора `End` обрабатывают оба возможных случая: 1) когда решение получается посередине нового (`decision`) и 2) когда в середине текущего интервала решения нет (`decision2`).

11. Графика в QB

Одним из основных преимуществ компьютера перед иными вычислительными устройствами является возможность вывода данных в графической форме. Как известно, электронный луч (или несколько лучей, как это происходит при передаче цветных изображений) разогнанный катодной пушкой лучевой трубки кинескопа и отклоняемый затем магнитами для получения не просто яркой точки на экране, а целого пучка параллельных горизонтальных линий, которые и передают изображение, движется в лексикографическом порядке. Так же, например, читаются элементы матриц в алгебре — сначала строка слева направо, а затем переход на нижележащую строку. При этом экран монитора становится куском координатной плоскости, у которой: 1) началом координат является левый верхний угол; 2) ось X горизонтальна и растет вправо, 3) а ось Y, в отличие от привычного расположения на графиках функций из курса математики, растет не вверх, а вниз.

Основными характеристиками графического режима, в который можно перейти при помощи QB, являются количество точек по горизонтали и по вертикали, а также передача цветов. Кроме того, принято говорить о графическом режиме при помощи терминов, отражающих развитие графических адаптеров мониторов компьютеров (CGA, EGA, VGA, SVGA). Существуют также и иные типы адаптеров (Hercules, MCGA и т.д.). Ниже приведем таблицу, в которой дается резюме основных графических экранных режимов, обычно поддерживаемых QB.

Для графических адаптеров MDPA, CGA, Hercules, Olivetti, EGA, VGA, или MCGA

Режим — SCREEN 0: Режим только текста
Формат текста — 40 x 25, 40 x 43, 40 x 50, 80 x 25, 80 x 43, или 80 x 50,
Ячейки букв — 8 x 8 (8 x 14, 9 x 14, или 9 x 16 с адаптерами EGA или VGA)
Цвет — 16 цветов приписываются любому из 16 атрибутов (при CGA или EGA)
64 цвета при каждом из 16 атрибутов (при EGA или VGA)
Видеопамять — до 8 страниц видеопамяти, в зависимости от разрешения текста и адаптера
(0-7), 4 стр. (0-3), 2 стр. (0-1) или 1 страница (0)

Для графических адаптеров CGA, EGA, VGA или MCGA
Режим — SCREEN 1: графика 320x200 точек
Формат текста — 40 x 25 при размере буквенной ячейки 8 x 8
Цвет —16 фоновых цветов и один или два набора из трех 3 наложенных цветов приписываемых при помощи оператора COLOR при адаптере CGA, 16 цветов, приписываемых 4 атрибутам при EGA или VGA
1 страница видеопамяти (0)

Режим — SCREEN 2: графика 640 x 200 точек
Формат текста — 80 x 25 при ячейке 8 x 8
Цвет — 16 цветов приписаны 2 атрибутам при EGA или VGA
1 видеостраница памяти (0)

Адаптеры Hercules, Olivetti или AT&T
Режим — SCREEN 3: требуется адаптер Hercules, только монохромный
Графика —720 x 348 точек
Формат текста — 80 x 25 при ячейках 9 x 14
Обычно 2 видеостраницы памяти (0-1); 1 страница (0), если установлен второй дисплей
Оператор PALETTE не поддерживается
Следует вызвать драйвер для Hercules MSHERC.COM перед использованием оператора **screen mode** 3

Режим — SCREEN 4:
Поддерживается ПК Olivetti Personal Computers моделей M24, M240, M28, M280,
M380, M380/C, а также M380/T и ПК AT&T серии 6300
Графика — 640 x 400 точек
Формат текста —80 x 25 при ячейке 8 x 16
1 из 16 цветов приписывается как наложенный цвет (выбирается при помощи оператора COLOR); фиксированный фоновый цвет — черный

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

видеостраница памяти (0)
Оператор PALETTE не поддерживается

Адаптеры EGA или VGA
Режим — SCREEN 7: графика 320 x 200 точек
Формат текста — 40 x 25 при ячейке 8 x 8
Приписываются 16 цветов любому из 16 атрибутов
Если память адаптера EGA равна 64K, то 2 видеостраницы памяти (0-1); в противном случае, до 8 страниц (0-7)

Режим — SCREEN 8: графика 640 x 200 точек
Формат текста —80 x 25 при ячейке 8 x 8
Приписываются 16 цветов любому из 16 атрибутов
Если адаптер EGA имеет память объемом 64K, то 1 видеостраница в памяти (0); иначе, до 4 стр. (0-3)

Режим — SCREEN 9: графика 640 x 350 точек
Формат текста — 80 x 25 или 80 x 43 при ячейке 8 x 14 или 8 x 8
Цвет —16 цветов приписаны 4 атрибутам (память адаптера 64K) или же 64 цвета приписываются 16 атрибутам (если память видеоадаптера больше 64K)
При 64K памяти у EGA адаптера 1 страница видеопамяти (0); иначе 2 страницы (0-1)

Адаптеры EGA или VGA, только для монохромных мониторов:
Режим — SCREEN 10: графика 640 x 350 точек только для монохромных мониторов
Формат текста — 80 x 25 или 80 x 43 при ячейках 8 x 14 или 8 x 8
До 9 псевдоцветов приписываются 4 атрибутам
2 страницы видеопамяти (0-1), требуется 256K памяти адаперта

Адаптеры VGA или MCGA
Режим — Screen 11 (VGA или MCGA)
Графика — 640 x 480 точек
Формат текста 80 x 30 или 80 x 60 при ячейках 8 x 16 или 8 x 8
Приписываются до 256K цветов по 2 атрибутам
1 страница видеопамяти (0)

Режим — Screen 12 (VGA)
Графика —640 x 480 точек

Формат текста 80 x 30 или 80 x 60 при ячейках 8 x 16 или 8 x 8
Приписаны до 256K цветов по 16 атрибутам
1 страница видеопамяти (0)

Режим —13 (VGA или MCGA)
Графика — 320 x 200 точек
Формат текста — 40 x 25 при ячейке 8 x 8
Приписываются до 256K цветов по 256 атрибутам
1 страница видеопамяти (0)

Таким образом, оператор SCREEN определяет в каком именно режиме будет работать монитор, сколько точек составит ось X, а сколько — ось Y. Так, при режиме 9 (640x350 точек), по горизонтали можно разместить 640 точек, а по вертикали — 350.

Существует еще один важный оператор для работы с цветом (он уже упоминался в вышеприведенной таблице) — оператор COLOR. Формат предъявления этого оператора таков:

```
COLOR [foreground%] [,[background%] [,border%] Screen mode
0 (text only)
COLOR [background%] [,palette%] Screen mode 1
COLOR [foreground%]      Screen modes 4, 12, 13
COLOR [foreground%] [,background&] Screen modes 7-10
```

Здесь

`foreground%` число, устанавливающее на мониторе цвет переднего
`foreground&` плана. При режиме `screen mode 0`, `foreground%` это цветовой атрибут цвета текста. При других режимах `screen`, `foreground%` является цветовым атрибутом или 4-битным значением цвета(только для `screen mode 4`), устанавливающим цвет текста и прорисовываемых линий.

`background%` число, устанавливающее фоновый цвет `background&` дисплея. При `screen mode 0`, `background%` является цветовым атрибутом. При `screen mode 1`, `background%` есть 4-битное значение цвета. При режимах `screen mode` от 7 до 10, `background&` есть значение цвета.

`border%` цветовой атрибут для каймы экрана.

`palette%` число (0 или 1) указывающее который из двух атрибутов используется:

palette%	Атрибут 1	Атрибут 2	Атрибут 3

```
ННННННН      НННННННННННН  НННННННННННН  НННННННННННН
   0            Зеленый       Красный       Коричневый
   1             Циан        Пурпурный      Ярко-белый
```

Доступные цветовые атрибуты и значения зависят от графического адаптера и последнего значения (режима) оператора SCREEN.

Если система снабжена адаптером EGA, VGA или MCGA, то используйте оператор PALETTE для изменения цветовых приписываний цветовым атрибутам.

Пример:

```
'Пример требует наличия цветного адаптера.
   SCREEN 7
   FOR i% = 0 TO 15
          COLOR i%
          PRINT i%
   NEXT i%
   end
```

Для работы с точками следует использовать операторы, которые могут рисовать на экране. Основных операторов воспроизведения графики в QB два, это операторы LINE и CIRLE.

Оператор LINE рисует на экране линию, пунктирную линию, прямоугольник или заполненный прямоугольник. Формат предъявления этого оператора в программах таков:

```
LINE [[STEP](x1!,y1!)]-[STEP](x2!,y2!) [,[color%] [,[B |
BF] [,style%]]]
```

где

STEP	указывает на то, что координаты берутся относительно текущей позиции графического курсора.
(x1!,y1!)	экранные координаты начала линии
(x2!,y2!)	экранные координаты конца линии.
color%	атрибут цвета, которые определяет цвет линии или прямоугольника. Доступные цвета зависят от характеристик вашего графического адаптера и режима работы экрана, определяемого оператором screen, встреченным управлением программы последний раз в программе перед оператором line.
B	прорисовка прямоугольника вместо линии.
BF	прорисовка заполненного прямоугольника.
style%	16-битовое значение, биты указывают но то, прорисовываются или нет пиксели. Используется для прорисовки точечный или пунктирных линий.

Пример:

```
'Пример требует наличия цветного адаптера.
  SCREEN 1
  LINE (110, 70)-(190, 120), , B
  LINE (0, 0)-(320, 200), 3, , &HFF00
  End
```

Наконец, рассмотрим оператор CIRCLE, используемый для прорисовки на экране заполненных или нет кругов и эллипсов. Его формат предъявления таков:

```
CIRCLE [STEP] (x!,y!),radius![,[color%] [,[start!] [,[end!]
[,aspect!]]]]
```

Здесь

STEP	указывает на то, что координаты берутся относительно текущего положения графического курсора.
(x!,y!)	координаты центра окружности или эллипса.
radius!	радиус окружности ил эллипса в единицах текущей координатной системы, определяемой, фактически, последним предъявлением операторов SCREEN, VIEW и WINDOW.
color%	цветовой атрибут цвета линии. Доступность атрибутов зависит от типа графического адаптера и последнего предъявления оператора SCREEN.
start!	начальный угол дуги окружности в радианах
end!	конечный угол дуги в радианах.
aspect!	отношение длины оси Y к длине оси X у рорисовываемого эллипса.

Для перевода градусов в радианы градусы умножаются на (PI / 180).

Пример:

```
'Пример требует наличия цветного графическ. адаптера.
      SCREEN 2
      CIRCLE (320, 100), 200
      CIRCLE STEP (0,0), 100
      end
```

При помощи этих операторов можно добиться получения различных рисунков на экране монитора, например, нарисовать карту определенного региона. Для этой цели можно использовать такой прием: через поднятие карты на светостоле на миллиметровую бумагу получают возможность отследить на какое количество пикселей и в каком направлении нужно переместить графический курсор при переходе по контуру региона от одной точки до следующей. Для упрощения задачи, определив в какую именно сторону должен быть направлен градиент перемещения графического курсора, можно использовать относительные

ординаты, значения которых будут колебаться в небольших пределах. Если же вам захочется использовать везде абсолютные, а не относительные координаты точек контура на карте, то придется создавать файл массива, в котором будет присутствовать двух- трехзначные числа. Раскрасить карту, т.е. залить контуры (открытые, незамкнутые линии заливать сложнее, о приемах чуть ниже) можно оператором PAINT (раскрась) — он заливает области с полностью закрытым контуром границы. Если же понадобится закрасить область с незамкнутой границей, то можно под нее нарисовать копию с замкнутым контуром, но без цвета (используя нулевую краску), а затем уже видимым цветом поверх закрашенной области нарисовать незамкнутую границу.

Еще одним способом создания изображений является использование оператора DRAW, дополнительные сведения о котором можно получить в меню «Помощь».

Ниже мы приведем пример использования графических возможностей QB для создания «движущихся» изображений («секундомер»).

Задача. Нарисовать движущийся объект, напоминающий секундомер.

Решение.

```
CLS
c = 3.141502654# / 180
SCREEN 9
CIRCLE (320, 175), 150, 7
FOR i = 1 TO 360 STEP 30
LINE (320 + 145 * SIN(c * i), 175 - 105 * COS(c * i))-(320
+ 150 * SIN(c * i), 175 - 110 * COS(c * i)), 4
NEXT i
FOR i = 1 TO 360
LINE (320, 175)-(320 + 120 * SIN(c * i), 175 - 96 * COS(c *
i)), 6
FOR j = 1 TO 2000
s = s ^ 3
NEXT j
LINE (320 + 120 * SIN(c * i), 175 - 96 * COS(c * i))-(320,
175), 0
NEXT i
END
```

Не все строчки поместились полностью, но нужно помнить, что длина строки может быть гораздо больше, чем на экране редактора (до 256 символов).

Здесь вложенный цикл по `j` нужен для того, чтобы «заторомозить» прорисовку движущейся стрелки секундомера. Ее движение можно довольно точно прокалибровать, меняя конечное значение параметра цикла `j`.

ПРИЛОЖЕНИЯ

Словарик

Alt — альтернатива (название клавиши)

Array — массив

Arrow — стрелка

Backspace — шаг назад (название клавиши)

Cancel — отказаться

CapsLock — переход в режим заглавных букв (название клавиши)

Clear — очистить

Close — закрыть

Control — управление (название клавиши)

Copy — копировать

Delete —удалить

Division by zero — попытка деления на нуль!

DOS — Дисковая Операционная Система

Edit — редактировать

End — на конец строки (название клавиши)

Enter — ввести, клавиша «Ввод» (название клавиши)

Escape — выйти, убежать (название клавиши)

Execute — выполнить, исполнить

Exit — выйти, выход

File — файл

Find — найти

Help — помощь

Home — на начало строки (название клавиши)

Insert — переход в режим замещения (или вставки) (название клавиши)

Label — метка

Line — срока

Name — имя

Number — число

NumLock — включена правая цифровая подклавиатура (клавиша , надпись под фотодиодом)

One or more — один или несколько

Open — открыть

Overflow — переполнение

Past — вывести из буфера

Path — путь (к файлу по структуре каталогов, начиная от корневого)

Press — нажать

Prompter — значок командной строки операционной системы

Quit — выйти

Replace — заменить

Restart — перезапуск

Resume — принять, вернуться и продолжить

Return — возвратиться

Run — запуск

Save — сохранить

Save as — сохранить, изменив имя или место

Search — искать

Shift — сдвиг регистра (название клавиши)

Specify — указать

Start — начать, начало

String — символьная переменная или строка

Tab — клавиша табуляции (обычно, позволяет переходить из окна в окно в диалоговом режиме)

Upper/Lower case —прописные/маленькие буквы

View — просмотреть

Window — окно

Основной символьный набор в QB

Набор символов в QB включает в себя все буквы латинского алфавита (A-Z, a-z), цифры (от 0 до 9, а также A-F и a-f, различаемые как дополнительные для шестнадцатеричной записи чисел). Помимо этого существуют специальные символы, которые имеют в QB определенный смысл.

Суффиксы, определяющие тип данных

! —— одинарная точность
—— двойная точность
% —— целые
& —— длинные целые
$ —— символьные (стринги)

Математические операторы

* — знак умножения
- — знак вычитания
/ — знак деления
= — оператор сравнения или присваивания
> — больше, чем
+ — знак сложения
. — десятичная точка
< — меньше, чем
\ — целочисленное деление
^ — возведение в натуральную степень

Спецсимволы

‘— символ комментирования
; — управление действием операторов PRINT и INPUT
, — управление действием операторов PRINT и INPUT
: — разделяет операторы на одной строке
? — промптер оператора INPUT
_ — подчеркивание для обозначения продолжающейся строки (не используется в QB, хотя и зарезервирован для других версий BASIC)

Код ошибки	Сообщение	Перевод
	NEXT without FOR	Оператор NEXT без части FOR
	Syntax error	Синтаксическая ошибка
	RETURN without GOSUB	Оператор RETURN без части GOSUB
	Out of DATA	Не подходит тип данных
	Illegal function call	Незаконный вызов функции
	Overflow	Переполнение
	Out of memory	Не хватает объема памяти
	Label not defined	Метка не определена
	Subscript out of range	Индекс выходит за пределы
0	Duplicate definition	Повторное определение
1	Division by zero	Деление на нуль
2	Illegal in direct mode	Нелегальное действие в прямом режиме
	Type mismatch	Неправильно напечатано

3		
4	Out of string space	Выход за пределы симв. перем. (стринга)
6	String formula too complex	Формула стринга слишком сложна
7	Cannot continue	Не могу продолжить
8	Function not defined	Функция не определена
9	No RESUME	Нет оператора RESUME
0	RESUME without error	Оператор RESUME без ошибки
4	Device timeout	Вышло время ожидания сигнала устройства
5	Device fault	Ошибка устройства
6	FOR without NEXT	Оператор FOR без NEXT
7	Out of paper	Кончилась бумага на принтере
9	WHILE without WEND	Оператор WHILE без WEND
0	WEND without WHILE	Оператор WEND без WHILE

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

3	Duplicate label	Двойная метка
5	Subprogram not defined	Подпрограмма не определена
7	Argument-count mismatch	Несовпадение счетчика аргумента
8	Array not defined	Массив не определен
0	Variable required	Требуется переменная
0	FIELD overflow	Переполнение FIELD
1	Internal error	Внутренний сбой
2	Bad file name or number	Плохие имя файла или его номер
3	File not found	Файл не найден
4	Bad file mode	Плохой тип файла
5	File already open	Файл уже открыт
6	FIELD statement active	Оператор FIELD уже активен
7	Device I/O error	Ошибка ввода/вывода на устройстве

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

8	File already exists	Файл уже существует
9	Bad record length	Плохая длина записи
1	Disk full	Диск переполнен
2	Input past end of file	Ввод перешел за метку конца файла
3	Bad record number	Плохой номер записи
4	Bad file name	Плохое имя файла
7	Too many files	Слишком много файлов
8	Device unavailable	Устройство недоступно
9	Communication-buffer overflow	Переполнение буфера связи
0	Permission denied	Доступ запрещен
1	Disk not ready	Диск не готов
2	Disk-media error	Ошибка разметки носителя на диске
3	Feature unavailable	Свойство недоступно

4	Rename across disks	Переименование по символу диска
5	Path/File access error	Ошибка доступа пути/файла
6	Path not found	Путь не найден

ЛИТЕРАТУРА

1. Microsoft QuickBASIC. Programming in BASIC. Microsoft Document. 00888 Part No. 04376. 1988. P. 457.

Содержание