И. Сафронов

Бейсик
в задачах и примерах

2-е издание

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.06
ББК 32.973.26-018.1Basic
С12

Сафронов И. К.

С12 Бейсик в задачах и примерах. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2006. — 320 с.: ил.

ISBN 5-94157-527-0

В сборнике содержатся разработанные автором задачи и примеры для освоения ключевых понятий программирования с использованием языка Бейсик. В занимательной и доступной форме осваиваются виды алгоритмов, переменные, операторы, массивы, подпрограммы. Большое внимание уделяется наиболее популярной среди школьников теме графики. Второе издание книги обусловлено неспадающим читательским интересом, дополнено 150-ю новыми задачами, примерами удивительных и увлекательных игр и программ. Также книга содержит справочник по языку программирования QBasic.

Может использоваться в качестве задачника для учащихся старшей школы.

Для начинающих программистов

УДК 681.3.06
ББК 32.973.26-018.1Basic

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Людмила Еремеевская*
Зав. редакцией	*Григорий Добин*
Редактор	*Руслан Абдрахманов*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.12.05.
Формат 60×90$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 20.
Тираж 5000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-527-0

Оглавление

Предисловие

О предлагаемой книге

Прошло пять лет с момента выхода в свет первого издания этой книги. Она выдержала несколько допечаток тиража, что свидетельствовало об интересе читателей к излагаемому предмету.

За это время многое изменилось в аппаратной части компьютерной техники, появились новые мощные языки программирования, но, на мой взгляд, не угасает интерес к старому доброму Бейсику.

В новом, исправленном и дополненном издании, я добавил более 220 новых задач и примеров их решения, а также, для фанатов Бейсика, привел много примеров заставок, игр и полезных программ, которые можно либо использовать готовыми, либо (что гораздо ценнее ☺), дополнить, развить и улучшить собственными силами.

Задачи из этой книги могут быть применены для решения на любых других языках программирования, послужат хорошим тренингом для воспитания алгоритмического мышления у начинающих программистов.

Представляемая книга содержит опыт, накопленный автором за время работы преподавателем информатики в школе. Пятнадцать лет назад мы учили школьников работать на программируемых калькуляторах, сегодня — на самой современной вычислительной технике. Но, в любом случае, убеждение, что преподавание основ программирования в школе необходимо, осталось до сих пор, хотя и претерпело какие-то изменения.

Алгоритмизация мышления позволяет человеку выживать в бушующем море информации, формирует системный подход к любым жизненным ситуациям.

Кроме того, за этот пятнадцатилетний период работы в школе автору очень редко приходилось встречать хорошие задачники по программированию, которые были бы насыщены разнообразными примерами. А еще, полагая, что Бейсик сейчас преподается в большинстве своем школьникам и, по опыту зная, насколько им интереснее "живые", не сухие задания, автор попытался вести разговор с читателем на понятном им языке, включая иногда и какие-то сленговые компьютерные словечки.

В данной книге в предисловии вашему вниманию предлагается ряд сведений о языках программирования, в том числе и о языке Бейсик. В первой части вы найдете большой набор авторских и творчески обработанных задач, охватывающих все основные разделы программирования на языке начинающих программистов — Бейсик. Задачи сгруппированы по темам и, в основном, расположены по степени возрастания сложности.

Во второй части вы сможете найти решения некоторых задач с необходимыми пояснениями, а в третьей части и в приложении самые любознательные отыщут много интересных дополнительных сведений об изучаемом языке и представлении информации в компьютере.

Немного истории

Давайте для начала договоримся об определениях, чтобы в дальнейшем говорить на одном языке, поскольку язык алгоритмов должен быть понимаемым ясно и однозначно.

Под *языком программирования* мы будем понимать совокупность средств и правил представления алгоритма в виде, приемлемом для компьютера. Отсюда неискушенный читатель может решить, что компьютер, оказывается, при всей его кажущейся могущественности, не поймет задачи, поставленной ему на простом человеческом языке, будь то русский, английский или даже китайский. Таким образом, существует разделение всех языков программирования на две большие группы — языки высокого и

низкого уровней. Человек считает себя венцом творения (с этим можно согласиться, но можно и поспорить, если внимательней присмотреться к некоторым таким "венцам"), поэтому языком самого высокого уровня считается человеческий язык, и когда компьютер станет его легко понимать, то он вплотную приблизится к человеку. Языком самого низкого уровня считается язык так называемых машинных кодов. Все остальные алгоритмические языки лежат где-то посередине. Например, к языкам низкого уровня принадлежат так называемые языки семейства ассемблеров. Их достоинство в том, что они почти не требуют перевода для компьютера, и он практически сразу выполняет алгоритм. Есть, однако, существенный недостаток — писать программы на таких языках может только очень опытный программист, и получаются они слишком громоздкими. Напротив, языки высокого уровня в достаточно сильной степени приближены к человеческому (чаще к английскому) языку — это и Фортран, и Паскаль, и Си, но выполнение алгоритма компьютером в данном случае несколько тормозится предварительным переводом на язык машинных кодов.

К языкам высокого уровня принадлежит и тот, который рассматривается в данной книге. Он служит своего рода переводчиком между человеком и компьютером, помогая им понять друг друга и совместно добиваться решения поставленных задач. Разработан первый Бейсик в 1964 г. сотрудниками Дартмутского колледжа Дж. Кемени и Т. Курцем. Интересно происхождение названия языка. В прошлом веке один английский миссионер выделил из английского языка триста наиболее употребительных слов, назвал их Basic English и стал обучать туземцев. Опыт оказался весьма успешным, и контакты с аборигенами значительно упростились. Создатели языка Бейсик стремились достигнуть того же эффекта — облегчить понимание между "туземцами" — начинающими программистами, и компьютерами. Аббревиатура BASIC так и расшифровывается — "Beginner's All purpose Symbolic Instruction Code", что в переводе значит "многоцелевой язык символических команд для начинающих".

Идея оказалась удачной, и на десятилетия язык Бейсик стал основным в деле вовлечения в программирование новых и новых

адептов. У автора этой книги есть большое количество выпускников, давно превзошедших своего учителя в деле программирования, успешно работающих у нас и за рубежом, но начинавших с того же Бейсика — тогда еще для ZX-Spectrum. Даже в те времена на несовершенной, зачастую собранной своими руками, технике они писали очень приличные прикладные программы — и для дела, и для игры. Большое достоинство Бейсика, из-за которого его изучение продолжается в школах и поныне, — это возможность создавать диалоговые программы.

Ныне Бейсик вышел за рамки языка для начинающих, и его могучий потомок — Visual Basic позволяет творить на компьютере просто чудеса.

Но вернемся к нашему старому доброму Бейсику. За эти годы было создано несколько его версий — GW-Basic, MSX-Basic, TurboBasic, QuickBasic. Автор за время своей работы в школе программировал на всех этих версиях и практически на всех видах техники, начиная от БК и "Корвета" и заканчивая (на момент написания книги) Pentium 4. И все же в качестве основного для этой книги выбрал Russian QuickBasic. Но я уверен, что искушенный читатель знает, а неискушенный пусть успокоится, все эти версии очень похожи друг на друга базовым набором операторов и конструкций, и, изучив какую-либо одну версию, очень легко перейти на другую. Это примерно как различия русского языка в Санкт-Петербурге и Москве: мы говорим "карточка", а москвичи "проездной", мы говорим "ларек", а они — "палатка", мы произносим "дожди", а они — "дожжи". Можно бесконечно подтрунивать над этими небольшими различиями, но мы прекрасно друг друга понимаем. Итак, давайте говорить на великом и могучем языке высокого уровня — Бейсике.

О языках: интерпретаторы и компиляторы

Как уже было сказано выше, языки высокого уровня — это своего рода посредники в общении между человеком и компьютером. Непосредственно переводом задуманного человеком алго-

ритма с языка программирования на язык машинных кодов занимаются программы-трансляторы.

Трансляторы, в свою очередь, тоже делятся на две большие группы — интерпретаторы и компиляторы.

Компиляторы сначала переводят всю программу, написанную на алгоритмическом языке, в машинные коды, и после этого очень быстро исполняют ее. Быстрота выполнения — это плюс компиляторов. Но они требуют довольно большой предварительной работы, поскольку мы сможем увидеть результат выполнения программы только после успешной компиляции — перевода, а на этом этапе программа-компилятор обычно требует устранить все синтаксические ошибки. Поэтому невозможность видеть промежуточные результаты — это небольшой минус компиляторов.

К компиляторам принадлежат, например, языки Паскаль, Си, Турбо Бейсик.

Интерпретаторы покомандно переводят алгоритм с языка программирования на язык машинных кодов и тут же исполняют переведенную команду. В случае допущенной ошибки программа-интерпретатор прекращает работу и просит исправить неверную конструкцию. К интерпретаторам относятся как раз в основном языки семейства Бейсик. В том числе и рассматриваемый в этой книге Russian QuickBasic.

Достоинство интерпретаторов — в возможности видеть промежуточные результаты выполнения алгоритма и по ходу дела вносить в исполняемый алгоритм изменения. Недостаток — гораздо более медленная работа по сравнению с компиляторами.

Для большего понимания я бы сравнил процесс трансляции с процессом перевода текста с иностранного языка на русский. Компиляция — это письменный перевод, когда я получаю текст и целиком его перевожу, чтобы затем, уже на иностранном, его довольно быстро изложить. Интерпретация — это синхронный перевод, когда я после каждой произнесенной фразы ее перевожу.

Ну а теперь, если вышеизложенное не вызвало затруднений в понимании, вперед, к сверкающим вершинам Бейсика.

Дружественный интерфейс

Правда, у подножия сияющих вершин ваших будущих программных достижений я вас немножко приторможу с тем, чтобы вы отчетливо понимали, что плодами вашего творчества будут пользоваться и другие люди, зачастую ничего не сведущие в программировании. А поэтому я коротко изложу свои взгляды на мои представления об оформлении программ или о так называемом дружественном интерфейсе. Ну, кто хоть немного знает английский, тот уже может догадаться, что "интер" — это "между", а "фэйс" — это то, что бывает об "тэйбл", т. е. "лицо". Соответственно, интерфейс — "между лицами", в данном случае имеет смысл в значении оформления диалога между пользователем и компьютером в процессе исполнения написанной программистом (вами, уважаемый читатель!) программы.

Кстати, короткое лирическое отступление об английском языке в деле программирования. Это не роскошь, а суровая необходимость. Опять же вспоминаю с улыбкой своих выпускников, особенно тех, кто учил в школе немецкий. Мой программистский сленг обогатился такими терминами, как "филе наме" (file name), "гамовер" (game over), "лине" (line), "цирцле" (circle). Но произношение — это как раз не самое страшное, а вот писать и понимать ограниченный набор (Basic English) англоязычных терминов необходимо.

Итак, несколько моих принципов по оформлению программ:

- "Кашу маслом не испортишь" — как можно больше поясняющих комментариев к вашему алгоритму, и пользователям вашей программы хорошо, и (что немаловажно для учащихся) преподаватель всегда очень радуется. Но самое главное, лично вам будет очень просто редактировать и отлаживать программу. К сожалению, этот принцип зачастую игнорируется начинающими программистами, что снижает как текущие оценки, так и олимпиадные, а кроме того, очень затрудняет понимание сути алгоритма и нахождение возможностей для его улучшения.
- "Fool Proof" — защита от дурака. Ваша программа должна выдерживать натиск неграмотного пользователя (от хакера-то

все равно защиты не найти!) и сопротивляться его попыткам при исполнении программы ввести цифры, где надо было имя, или нажать клавишу <Пробел>, когда была указана клавиша <Enter>.

- Доходчивость. При написании программы не стоит рассчитывать, что работать с ней будут умные люди (хотя и такие встречаются среди пользователей), а потому при исполнении вашего алгоритма программа должна максимально подробно объяснять человеку, что она от него хочет, и какую "пимпочку" в данный момент надо нажать.
- Ну и конечно, "красота спасет мир". Пусть вы делаете даже просто программу проверки знания таблицы умножения, но сделайте это красиво — примените цветовое и шрифтовое оформление, а можно ведь и звуковые эффекты. Не скупитесь на экранные похвалы в адрес пользователя вашей программы — ласковое слово и кошке приятно!

В общем, я попробую в тех случаях, где буду приводить решения каких-то, на мой взгляд, ключевых обобщающих заданий, соблюсти эти принципы. А вам рекомендую брать с них пример.

Основные этапы решения алгоритмических задач на компьютере

Подход многих моих учеников к решению поставленных мной задач по программированию достаточно однообразен и в народе носит название "метода тыка". Получив задание, такие, с позволения сказать, "программисты" садятся к компьютеру и начинают комбинировать операторы языка Бейсик в различных вариантах, пытаясь достичь требуемого результата. Да, иногда получается. Но чаще — нет. Почему? Потому что отсутствует системный подход к задаче, что, во-первых, разбазаривает самый драгоценный человеческий ресурс — время, а во-вторых, не приносит плодов ни в программировании, ни в жизни.

Предлагаю метод, как, на мой взгляд, надо подходить к решению программистских задач. А вы уж решайте сами. "Это вам не ме-

лочь по карманам тырить", — как говорил незабвенный Остап Бендер, — "тут думать надо!".

Итак, этапы решения алгоритмических задач на компьютере.

1. *Постановка задачи*. Один из самых главных этапов. Вы должны добиться от того, кто дает вам задачу (это можете быть и вы сами), ясной и четкой ее постановки. Вы однозначно и вполне определенно должны понять, что будет результатом решения задачи. Каковы исходные данные? Существуют ли ограничения для этих данных? Можно сказать, что точность и четкость в постановке задачи — это половина дела. Напротив, в случае недопонимания каких-то моментов вероятность непроизводительной траты времени и отрицательного результата резко возрастает.
2. Следующий этап — решение вопроса "Как будет реализовываться поставленная задача?". Как достичь требуемых результатов? Каковы способы и методы достижения уясненных на первом этапе целей?
3. После первых двух этапов наступает пора еще одного очень важного момента — этапа *разработки алгоритма решения* поставленной задачи, т. е. структуризация, разбиение задачи на последовательность простых модулей, каждый из которых легко может быть реализован на языке программирования.
4. Очередной этап — непосредственный перевод словесного алгоритма или его блок-схемы на выбранный язык программирования и ввод полученной программы в компьютер.
5. После ввода программы обычно выясняется, что где-то мы допустили просто синтаксические ошибки, где-то недоработали алгоритм, где-то не хватает исходных данных и т. д. Поэтому теперь начинается *отладка программы*, иными словами, устранение ошибок и неточностей, допущенных на предыдущих этапах.
6. После того как программа заработала, необходимо проверить ее на правильность работы, используя набор контрольных данных (в тех случаях, где это возможно). Так, например, если мы написали программу для расчета корней квадратного уравнения по заданным коэффициентам, то можем проверить

работу программы, вводя такие коэффициенты, для которых предварительно были рассчитаны значения корней или их отсутствие. Это так называемый *тестовый этап*.

7. После тестового этапа (если программа его выдержала!) можно применять программу по назначению. Ну и последнее. Страна должна знать своих героев, а потому завершающим этапом работы по решению алгоритмической задачи следует считать *документирование*, т. е. распечатку листинга программы, снабженную необходимыми комментариями автора. С этого момента разработанная программа становится интеллектуальной собственностью программиста, и им начинает гордиться семья и школа.

ЧАСТЬ 1
Язык Бейсик

Прежде всего, очень важное замечание! Под операционными системами Windows 2000 и Windows XP изначально не поддерживается кириллический шрифт при работе с графикой, поэтому вы рискуете увидеть вместо нормальных пунктов меню и комментариев в своей программе следующие "иероглифы":

·tŽŹ ąžŻŘ ŷŰĽ źŹžŵąŲŷŨũ Ũ ŽąĽŨŻąŵŘŷńũ űŹũ`·

Выходы есть:

1. Найти в Интернете русификатор под именно эти операционные системы.
2. Работать не в Russian QuickBasic, а просто в QBasic.
3. Писать русские слова транслитом (т. е. заменять русские буквы латинскими аналогами, например "привет!" — "privet!".
4. Параллельно с Windows 2000 или XP поставить Windows'98 и наслаждаться Russian QuickBasic!

Главное, понимать, что это никак не отражается на работе программ.

Оболочка Russian QuickBasic

Прежде чем непосредственно приступить к программированию, надо научиться пользоваться средой предлагаемого к изучению языка. А поскольку она на русском языке, то это не составит большого труда.

Итак, нам предстоит узнать:

- где взять русский Бейсик;
- как его запустить и начать работу;
- как вводить текст программы в окне редактора;
- как запускать программы на выполнение;
- как сохранить программу на диске в виде файла и открыть уже существующую для просмотра и редактирования.

Где взять русский Бейсик

Собственно, проблем, я думаю, с этим возникнуть не должно — надо поспрашивать у учителей информатики, у друзей на компакт-дисках может быть, в конце концов обращайтесь к автору книги — не откажу. Рабочая версия занимает около 400 Кбайт.

Запуск русского Бейсика и начало работы

Существует три основных варианта.

- Если у вас на компьютере ничего, кроме MS-DOS, нет, то придется открыть каталог с Бейсиком, а затем в командной строке набрать имя запускающего файла `qbasic` и нажать клавишу <Enter>.
- Если у вас есть операционная оболочка Norton (или Volkov) Commander, то задача упрощается — переходите на панель, где содержится каталог с Бейсиком, открываете ее клавишей <Enter> или двойным щелчком левой кнопкой мыши, затем при помощи стрелок управления курсором находите файл qbasic.exe и запускаете его нажатием клавиши <Enter> или двойным щелчком левой кнопкой мыши.
- Самый модный вариант, если у вас MS Windows: найдите на Рабочем столе ярлык **QuickBasic** и дважды щелкните по нему левой кнопкой мыши.

Во всех трех случаях экран очищается и появляется среда русского Бейсика (рис. 1.1).

Рис. 1.1. Первое окно после запуска русского Бейсика

Далее следуйте инструкции, появившейся на экране. При нажатии клавиши <Enter> вы попадаете в Руководство для начинающих, а, нажав клавишу <Esc> — в окно редактора (рис. 1.2).

Компьютер теперь готов к вводу и редактированию ваших программ.

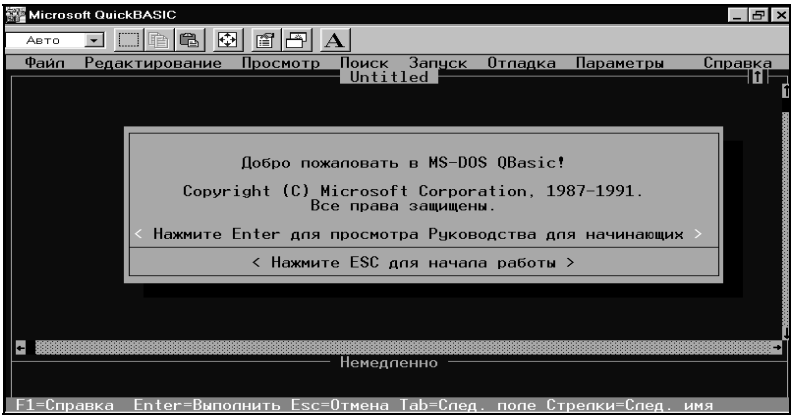

Рис. 1.2. Окно редактора

Вам для работы предоставляется два окна:

- верхнее — окно редактирования;
- нижнее — окно немедленного выполнения.

Переход из одного окна в другое осуществляется клавишей <F6> или щелчком в соответствующем окне левой кнопкой мыши.

В окне немедленного выполнения вы можете сразу видеть результаты работы набранных команд после нажатия клавиши <Enter>.

Как вводить текст программы в окне редактора

В окне редактора мигает курсор — горизонтальная светящаяся черточка, указывающая, где будет вводиться текст программы при наборе его с клавиатуры.

Если вы заметили, что что-то набрали неправильно, есть несколько путей исправить допущенные ошибки.

- *Удаление лишних символов*. Слева от курсора — клавиша <Backspace>. Символ, под которым мигает курсор и справа от курсора — клавиша <Delete>.
- *Вставка недостающих символов*. Курсор установить под тот символ, слева от которого необходимо произвести вставку и набрать нужные символы. Текст при этом раздвинется автоматически.
- *Переход из режима вставки в режим замены*. Режим вставки включается автоматически. В режим замены и обратно мы переходим нажатием клавиши <Insert>. Курсор при этом принимает вид светящегося прямоугольника. Теперь, при вводе какого-либо символа, будет стираться тот символ, который находился прежде на этом месте. Автоматическое раздвижение строки в этом случае не происходит.
- *Переход на следующую строку* осуществляется нажатием клавиши <Enter>. Если вы случайно нажали клавишу <Enter> в середине строки, то вторая половина ее перейдет вниз. Пугаться этого не надо. Нажимайте клавишу <Backspace>.

- *Копирование и перемещение фрагментов* текста программы. Сначала фрагмент нужно выделить. Выделение производится либо мышью при удерживаемой левой кнопке, либо клавишами управления курсором <←> и <→> при нажатой клавише <Shift>. После выделения возможны варианты:
  - удаление фрагмента — клавиша <Delete>;
  - копирование фрагмента в буфер — сочетание клавиш <Ctrl>+<Insert>;
  - вырезка фрагмента в буфер — сочетание клавиш <Shift>+ +<Delete>;
  - вставка фрагмента из буфера в новое место (сколько угодно раз) — сначала курсор нужно поместить в новое место, а затем — сочетание клавиш <Shift>+<Insert>.

Впрочем, все то же самое можно сделать через меню **Редактирование**.

Как Бейсик сообщает об ошибках

Если в программе была допущена ошибка, то после запуска программы на экране появится окно с сообщением об ошибке. У вас есть два варианта действий — выбрать кнопку **Справка**, чтобы получить информацию о допущенной ошибке, либо нажать клавишу <Enter> или <Esc>. Тогда вы вернетесь в окно редактирования, где курсор будет находиться в том месте, где компьютер нашел ошибку.

Запуск программы на выполнение

Как же запустить программу? Когда вы написали программу и хотите посмотреть, а что, собственно, из этого получилось, то надо нажать клавишу <F5>. Программа будет исполнена в случае отсутствия синтаксических ошибок, и тогда вы увидите результаты ее работы и сообщение внизу экрана: "Чтобы продолжить, нажмите любую клавишу...". Эта надпись вызывает иногда смятение в душах неопытных программистов, и они начинают судорожно искать на клавиатуре надпись "Любая клавиша". Надеюсь,

у вас до этого не дойдет. Если же в программе есть ошибки, то вы их исправляете, и у вас вновь два варианта:

- запустить программу с места, где она прервалась — клавиша <F5>;
- запустить программу сначала — сочетание клавиш <Shift>+ +<F5>.

Если вы хотите видеть окно, в котором видны результаты выполнения программы, то нажмите клавишу <F4>.

Основы есть? Если возникают вопросы, то, во-первых, при выборе тех или иных пунктов меню внизу появляется краткая информация о нем на русском языке, а по нажатии клавиши <F1> вы получите более подробные сведения. Во-вторых, в самом меню есть справка и по командам языка Бейсик, и по содержанию. Учитесь пользоваться справочным материалом!

Предупреждение

Если у вас отсутствует мышь, то попадание в меню осуществляется клавишей <Alt>, а работа в нем — клавишами управления курсором <←> и <→> и клавишей <Enter>.

Сохранение и открытие файлов в Бейсике

Написав программу и желая ее сохранить на диске, выберите в меню **Файл** команду **Сохранить** или **Сохранить как**. Откроется диалоговое окно, в котором вы должны выбрать диск и каталог для сохранения, дать имя файлу и нажать кнопку **ОК**. После этого в заголовке окна редактирования появится имя, назначенное файлу.

Предупреждение

Будьте внимательны к тому, *куда* вы сохраняете файл, и *под каким именем*. Иначе могут возникнуть проблемы с поиском вашей работы!

Чтобы открыть уже существующий файл, в меню **Файл** выберите команду **Открыть**. Появится диалоговое окно, в котором надо найти диск и каталог, где записан файл, указать его имя и нажать кнопку **ОК**.

Чтобы начать работу с новым файлом, в меню **Файл** выберите команду **Новый**.

Для выхода из Бейсика в меню **Файл** существует команда **Выход**. Если вы забыли сохранить свою программу, то Бейсик напомнит вам об этом до выхода и предложит сохранить ваш труд.

И только после этого вы выйдете в ту среду, из которой Бейсик был загружен.

Алфавит языка

В любом учебнике иностранного языка вначале дается его алфавит, т. е. набор символов для записи слов, предложений и всевозможных понятий этого языка. У языка Бейсик тоже есть алфавит, который содержит в себе следующие символы:

- Заглавные (или прописные) буквы латинского алфавита: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z. При наборе программы, впрочем, нет нужды следить за тем, чтобы буквы были заглавными. Интерпретатор сам изменит строчные буквы на заглавные.
- Арабские цифры: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0.
- Разделители: , (запятая), ; (точка с запятой), . (точка), : (двоеточие), ‘ (апостроф), " (кавычки), ((открывающая скобка),) (закрывающая скобка), символ <Пробел>.
- Знаки арифметических операций: + (сложение), – (вычитание), * (умножение), / (деление), ^ (возведение в степень).
- Знаки операций отношений: > (больше), < (меньше), = (равно), <> (не равно), >= (больше либо равно), <= (меньше либо равно).

Переменная и что в ней меняется

Компьютер, как и вы, уважаемые читатели, обладает памятью. Она бывает разная. В процессе отладки вашей программы компьютер напрягает эту самую память, размещая в ней исходные данные, обрабатывая их, используя ваш алгоритм, получая результаты и доводя их до вашего сведения — чаще на экран мони-

тора. Я думаю, для вас не секрет, что память эта называется ОЗУ (Оперативное Запоминающее Устройство), или по-английски RAM (Random Access Memory). Это, собственно, одно из основных устройств компьютера, имеющее, правда, ограниченный объем, измеряющийся в мегабайтах.

Если вы пишете достаточно сложную и уже нелинейную программу, то наверняка потребуются *переменные*, т. е. такие области этой самой оперативной памяти, которые имеют имя, данное нами, и значения, которые могут меняться. Имя переменной в ходе выполнения программы постоянно, а значение может меняться многократно. Этот процесс можно сравнить со сдаваемой на лето дачей. Дача — это участок (область памяти компьютера), имеющий уникальный и неповторимый адрес, по которому его можно найти (имя переменной) и который не меняется, и каждое лето на дачу приезжают новые жильцы (значения переменной).

Каковы правила на этот счет в Бейсик? Так как Russian QuickBasic — это язык, рассчитанный на использование из-под DOS (Disk Operation System, дисковая операционная система), то существуют ограничения на имена переменных:

- имя переменной должно состоять не более чем из сорока символов;
- в качестве символов можно использовать только латинские буквы, цифры;
- имя переменной не может начинаться с цифры;
- категорически запрещены в именах файлов символы точки, запятой, звездочки, вопросительного знака, пробела.

Примеры правильных имен переменных:

- X, Y, Z, IVAN;
- IVAN3, S1, T234, LOVE7, R6N8F43;
- NM, MAX, GAVGAV.

Примеры неправильных имен переменных:

- Г56 (использована русская буква);
- ИВАН (использована кириллица);

- YOU+ME (использован недопустимый символ "+");
- 23DROVA (имя переменной начинается с цифры).

Переменные различаются по типу хранимой в них информации. Два наиболее крупных типа — числовой (для хранения различных чисел) и строковый (для хранения символов и строк). Во втором случае к имени переменной добавляется обязательный символ $ (на клавиатуре — там же, где цифра 4, при нажатой клавише <Shift>), например, X$ или QUIKE3$.

Арифметика в Бейсике

Прежде чем двигаться дальше *("Как трудно двигаться дальше..." — из песни Бориса Гребенщикова)*, необходимо напомнить, что в те далекие времена, когда только зарождались алгоритмические языки, а словосочетание "персональный компьютер" вызывало у тех, кто его слышал, сомнения в здравомыслии его произносившего, так вот, в те самые времена считалось, что компьютер (от англ. *compute* — вычислять), т. е. "вычислитель" только и предназначен для того, чтобы считать в тысячи, нет — в миллионы раз быстрее человека. По сути, это действительно так, и если вы немного представляете себе физику происходящего в компьютере, то все, что вы ни делаете за компьютером — сочиняете стихи или музыку, рисуете картинки, играете, общаетесь в Сети — все внутри этого "вычислителя" сводится к цифровой двоичной форме и к действиям, элементарным арифметическим действиям над этими самыми числами.

Отсюда вывод — если хочешь быть программистом, надо дружить с математикой. Начнем?

Итак, компьютер умеет вычислять элементарные арифметические выражения. Но для того, чтобы он смог это сделать, мы должны представить это самое выражение в понятном ему виде, а именно:

- в отличие от арифметики, выражение должно быть записано в *одну строку* безо всяких числителей и знаменателей;
- для записи арифметических действий допустимо использовать только перечисленные ниже знаки:

- + (сложение, слева от клавиши <Backspace> или на малой цифровой клавиатуре "серый плюс");
- – (вычитание, то же, что дефис, или на малой цифровой клавиатуре "серый минус");
- * (умножение, там же, где цифра 8 на основной клавиатуре при нажатой клавише <Shift> или на малой цифровой клавиатуре "серая звездочка");
- / (деление, на разных клавиатурах бывает в разных местах или на малой цифровой клавиатуре "серый слэш");
- ^ (возведение в степень, при выбранном латинском шрифте там же, где цифра 6 на основной клавиатуре при нажатой клавише <Shift>);
- () (скобки, там же, где цифры 9 и 0 на основной клавиатуре при нажатой клавише <Shift>);

- ❐ недопустим пропуск знака умножения между коэффициентом и переменной, как это возможно в алгебре (например, нельзя писать 2x, а надо 2*X, или нельзя 5d, а надо 5*D);
- ❐ дробная часть отделяется от целой *точкой*, а не *запятой* (нельзя писать 3,14, а надо 3.14);
- ❐ допустимо опускать в записи десятичной дроби *ноль*, стоящий перед точкой (вместо 0.123 можно .123).

Чтобы компьютер вычислил выражение правильно, необходимо помнить о приоритете выполнения действий. Тут все как в элементарной математике:

- ❐ сначала выполняются действия в скобках (в Бейсике скобки используются только круглые, в сложных выражениях они могут быть и двойные, и тройные, и т. д.);
- ❐ далее вычисляются функции, если они есть;
- ❐ затем выполняется возведение в степень;
- ❐ потом умножение и деление;
- ❐ в последнюю очередь — сложение и вычитание.

Действия одинаковой очередности выполняются слева направо.