MSX BASIC для начинающих

Fix Me!

Здесь представлена обработка этого материала.

Введение

Наверное, не стоит читать эту книжку тем, кто что–то уже умеет делать на Бейсике. Эта книжка для начинающих, и к тому же, учеников не очень старших классов. Её главное достоинство — простота (автор надеется, что это не самообман).

Чтобы привыкнуть к Бейсику, знать его как свой собственный язык, недостаточно только читать эту книжку. Нужно каждый приведённый в ней пример в рамочке тотчас же опробовать на компьютере MSX. Обязательно выполняйте упражнения (выделены серым фоном) ! Если не выполнять чего-то своего после ознакомления с чужой программой, то и не станешь самостоятельным программистом!

Урок 1

Начинают знакомство с Бейсиком с простой команды: ПЕЧАТАЙ.

В Бейсике для этого есть служебное слово `PRINT` (или print, поскольку MSX BASIC, имеющийся в Ямахе, понимает одинаково заглавные и строчные буквы. Главное, чтобы служебные слова были написаны не русским, а латинским шрифтом!).

После слова `PRINT` нужно указать, что именно Бейсик должен напечатать. Бейсик умеет печатать ТЕКСТЫ и ЧИСЛА.

ТЕКСТ — это любые символы (буквы, цифры, специальные знаки), которые есть на клавиатуре. В начале текста стоит кавычка «"».

Кончается текст такой же кавычкой. Все, что стоит между кавычками — это и есть текст. Бейсик печатает текст без изменений. Например, если мы напишем

```
PRINT "привет"
```

и нажмём клавишу ⏎ (чёрную с изогнутой стрелкой. Её нужно нажимать после любой команды!), то на следующей строчке появится:

```
привет
Ok
```

Ok — это подсказка Бейсика, которая обозначает: Бейсик готов к приёму следующей команды. Если же вы забыли написать кавычки, то результат выполнения команды `PRINT` предсказать несложно: он ничего не напечатает, а просто пропустит строчку и напишет Ok. Кстати, если вы вместо слова «привет» напишете английскими буквами «hallo» и забудете кавычки, то Бейсик вместо пустой строки напечатает 0. Пока текст в кавычках, для `PRINT` безразлично, русский он или английский. Если же кавычки пропущены, Бейсик считает английский текст своими служебными словами, и его поведение зависит от того, встретятся ли в нем на самом деле служебные слова Бейсика (в том случае будет сообщение об ошибке, потому что служебные слова не должны стоять после `PRINT`) или не встретятся (тут–то и будет напечатан 0).

Кроме текстов PRINT умеет печатать ЧИСЛА. Можно напечатать готовое число:

```
PRINT 25
```

Если же напечатать после `PRINT` какое–нибудь арифметическое выражение, например:

```
PRINT (11+6.12)*3
```

то на экран выведется результат вычислений. Вы видите из примера, что в десятичной дроби вместо запятой используется точка. Это так не только в Бейсике, но и в других языках программирования. Скобки в арифметических выражениях используются как и в математике, для того, чтобы выделенное в скобках выражение было самым приоритетным (то–есть, выполнялось в первую очередь).

В арифметических выражениях используются такие действия:

1. [^] — возведение в степень. Например, 2^3 это 2 в 3–й степени. Это — самая приоритетная операция (то–есть, в выражениях она выполняется в первую очередь)
2. [*] — умножение
3. [/] — деление. Умножение и деление — следующие по приоритетности действия, то–есть, они выполняются после возведения в степень. Дальше по старшинству идут такие действия:
4. [+], [-] — сложение и вычитание.

Напечатайте результат таких действий:

```
 2+11•(6-3)
──────────── +52
8-216:(1+71)
```

Надеюсь, вы не забыли, что и дробная черта, и [:] в Бейсике заменяются знаком [/]? Надеюсь также, что вы догадались и числитель и знаменатель взять в скобки? Если все правильно сделано, получилось 59.

Вы поработали, ваш экран полон мусора. Нужно его очистить. Для этого напишите команду `CLS` и нажмите ⏎. Вместо этого можно было бы просто нажать вместе кнопки SHIFT и CLS. Но все равно программисту нужно знать команду `CLS`, которая чистит экран.

После чистки экрана, если вам нужно снова выполнить те же расчёты, придётся по новому написать эти команды. В памяти компьютера они не сохранились. Дело в том, что вы работали в командном режиме, когда команда сразу выполняется, но в памяти не остаётся. Возможно работать и в программном режиме, когда команды не выполняются сразу, а только заносятся в память. Программный режим удобен тем, что после запуска программы все команды будут выполняться по очереди до самой последней. Напишите, нажимая ⏎ в конце каждой строки:

```
10 PRINT "Рассчеты показали,"
20 PRINT "что 2+2*2 будет"
30 PRINT 2+2*2
```

Вы увидели, что команды после нажатия ⏎ не выполняются. Но зато они занесены в память. Чтобы убедиться в этом, сотрём все с экрана с помощью `CLS` и напишем команду `LIST` (просмотр программы). Вы увидите, что программа сохранилась. Теперь нужно её выполнить. Для этого служит команда `RUN`. Выполните её и все строки выполнятся по очереди. Ещё раз хотите запустить программу,— ещё раз выполните `RUN`. (Команды `RUN` и `LIST` закреплены за специальными функциональными клавишами F5 и F4. Вместо того, чтобы писать `RUN`, можно просто нажать F5.) Написанную программу можно изменить. Для этого есть несколько способов:

1. Чтобы добавить новую строку в программу, нужно просто написать её на свободном месте экрана и нажать ⏎. Номер новой строки не должен совпадать с уже имеющимися, иначе она запишется в память вместо строки с тем же номером. Напишите

   ```
   15 print "(Это было вчера)"
   ```

   и нажмите ⏎

   Выполнив команду `list`, вы увидите:

   ```
   list
   10 PRINT "Рассчеты показали,"
   15 PRINT "(Это было вчера)"
   20 PRINT "что 2+2*2 будет"
   30 PRINT 2+2*2
   ```

   Строка, которую мы написали отдельно, разместилась в программе между другими строками так, что номера строк возрастают.

2. Чтобы изменить уже имеющуюся строку, нужно вывести её на экран с помощью команды LIST и внести нужные исправления, а затем не забыть нажать ⏎ (Если вы этого не сделаете, исправления не будут приняты). Fix Me!

Урок 2

Вы знаете, что компьютер может запоминать числа и тексты.

Посмотрим, как это сделать. Введите команды:

```
NEMNOGO=23
PRINT NEMNOGO+2
```

Последняя команда напечатает 25. Это говорит о том, что первая команда занесла в память под именем NEMNOGO число 23, а вторая команда в арифметическом выражении вместо имени NEMNOGO использовала его значение 23.

Чем такой способ удобнее, чем просто использование числа 23 ? Дело в том, что такие имена, которые хранят в себе числа, можно использовать, как переменные в математике. А ведь уже второклассники хорошо знают, что переменные куда удобнее для решения задач, чем просто числа.

Для примера разберём, как работает программа, отгадывающая задуманное число. Пусть человек задумает число. Умножит его на 2. Прибавит 10 и то, что получилось снова умножит на 2. От полученного отнимет 8 и разделит все на 4. Теперь пусть скажет нам ответ, а мы угадаем задуманное. Чтобы это сделать, мы с его числом проделаем все действия в обратном порядке.

Например, у него получилось 17. Запомним 17 под именем X:

```
10 x=17
20 y=x*4
30 x=y+8
40 y=x/2
50 x=y-10
60 y=x/2
70 PRINT y
```

Мы получим 14, что и задумывал человек. Кстати, все эти действия можно было разместить в одном PRINT, не используя промежуточной переменной Y:

```
10 x=17
20 PRINT ((x*4+8)/2-10)/2
```

Результат получится тот же. Скажу вам по секрету, что если упростить выражение после PRINT, раскрыв скобки по правилам математики, то получится вообще коротко:

```
10 x=17
20 PRINT x-3
```

Тут уж никакого компьютера не надо: можно устно угадать задуманное число!

Как бы сделать, чтобы не в программу вписывать то, что получилось у человека, а чтобы он сам ввёл своё число?

Для этого есть специальная команда запроса INPUT. В нашем случае:

```
10 INPUT x
20 PRINT x-3
```

При выполнении команды INPUT на экране появляется [?], после которого стоит курсор. Когда человек напишет число и нажмёт ⏎, это число будет занесено в переменную X, которая указана после INPUT, и программа продолжит выполняться.

Ещё лучше, если перед тем, как ждать от пользователя (так называют человека, который пользуется готовой

программой) числа, программа сообщит ему, что от него нужно:

```
10 PRINT "Задумайте число"
20 PRINT "Умножьте его на 2"
30 PRINT "Прибавьте 10"
40 PRINT "Снова умножьте на 2"
50 PRINT "Отнимите от полученного 8"
60 PRINT "и разделите на 4"
70 PRINT "Напишите, сколько у Вас получилось"
80 INPUT X
90 PRINT "Вы задумали";X-3
```

Такая программа, которая что–то спрашивает у пользователя и использует в дальнейшей работе его ответы, называется диалоговой программой. Напишем ещё пример диалога:
Пусть программа спросит у человека его возраст и посчитает, сколько ему осталось до 100 лет:

```
10 PRINT "Сколько Вам лет?"
20 INPUT le
30 PRINT "До 100 лет Вам осталось еще";100-le
```

Кстати, команда `INPUT` и сама без `PRINT` может напечатать текст вопроса:

```
10 INPUT "Сколько Вам лет?";le
20 PRINT "До 100 лет Вам осталось еще";100-le
```

Запомните только, что текст вопроса — в кавычках, и после него перед именем переменной, куда будет запоминаться ответ, обязательно стоит [;]

Запомните! Только один текст вопроса в кавычках перед переменными! Больше ничего `INPUT` печатать не может. Все остальное (значения переменных, расчёты,...) печатайте с помощью `PRINT`.

Выполните упражнения:

> Программа спрашивает радиус окружности у пользователя, а затем печатает длину окружности и площадь круга.

Со следующим упражнением справятся 8/9–классники. Можно воспользоваться синусом или теоремой Пифагора по вашему выбору. Если вы моложе и таких вещей не знаете, пропустите его. Бейсик понимает `SIN`, `COS`, `TAN` (так в Бейсике записывается тангенс), но нужно не забывать, что это — функции, а значит аргументы должны помещаться в скобках. Например, `SIN(5)`...

> Программа спрашивает длину одной стороны равностороннего треугольника и печатает его периметр и площадь.

Кроме известных вам арифметических операций могут понадобиться ещё две:

1. Деление нацело (остаток отбрасывается). Например, 25 делённое на 7 даёт 3 и 4 в остатке. При делении нацело получается 3. Для деления нацело Бейсик использует не [/], а [\]
   `print 24\5` даст 4.

2. Остаток от деления одного числа на другое может быть получен операцией, которая обозначается в Бейсике не значком, а служебным словом `MOD`. Например,
   `print 27 mod 5` даст 2 (остаток от деления 27 на 5).

Пользуясь новыми командами, решим следующий пример: Программа должна запросить, сколько секунд длилось

какое–нибудь событие, а затем перевести эти секунды в сутки, часы, минуты и секунды. Вы помните, что сутки=24 часа, час=60 минут, минута=60 секунд... Решать будем по такому плану:

- 1) Узнаем, сколько всего секунд (с помощью INPUT)
- 2) Узнаем, сколько всего минут (поделив нацело на 60)
- 3) Узнаем, сколько всего часов (поделим количество минут на 60)
- 4) Сколько суток (Часы делим на 24)

Теперь нужно идти в обратном порядке: Зная сколько полных суток, нужно рассчитать, сколько осталось часов, не вошедших в полные сутки:

- 5) От общего числа часов отнимаем число суток, умноженное на 24, получаем количество часов кроме полных суток.
- 6) Чтобы узнать количество минут кроме тех, которые составили полные часы, нужно от всех минут отнять те, которые вошли в полные часы
- 7) Отняв от общего количества секунд те, которые вошли в полные минуты, мы получим количество оставшихся секунд.

Если не все в этом плане понятно, лучше не смотрите программу, а внимательно разберитесь.

```
10 input"Введите время в секундах";sec
20 min=sec\60
30 sec=sec-min*60   '  Сколько секунд кроме вошедших в минуты?
40 hour=min\60
50 min=min-hour*60  '  Сколько минут кроме вошедших в часы?
60 days=hour\24
70 hour=hour-days*24    '  Сколько часов кроме целых суток?
80 print "Суток-";days;
90 print "Часов-";hour;
100 print"Минут-";min;
110 print"Секунд-";sec
```

Используя операцию остаток от деления, можно упростить следующие строки:

```
30 sec=sec mod 60   Находим остаток секунд
…
50 min=min mod 60   … минут
…
70 hour=hour mod 24 … часов
…
```

Разберём напоследок задачу, которая часто встречается в задачниках по математике: Первый землекоп выкопает яму за 6 дней. Второй — за 8 дней. За сколько дней выкопают вместе?

Решить эту задачу уравнением несложно, но сначала объясню по действиям:

- 1) Какую часть ямы выроет за 1 день 1–й землекоп? 1/6 ямы.
- 2) Какую часть выроет за день 2–й землекоп? 1/8 ямы.

- 3) Какую часть ямы выроют за день оба землекопа вместе?
  1/6 + 1/8 = 7/24 (привели к общему знаменателю)

- 4) За сколько дней выроют вместе?
  1 : 7/24 = 1 x 24/7 = 24/7 = 3 и 3/7 дня

Теперь те же действия, но с применением переменных: t1 — время, за которое выроет яму 1–й землекоп, t2 — время для второго.

1. v1 = 1/t1
2. v2 = 1/t2
3. v3 = v1 + v2
4. t3 = 1/v3 — это и есть ответ.

```
10 INPUT "Введите время для 1-го землекопа";t1
20 INPUT "Введите время для 2-го землекопа";t2
30 v1=1/t1 : v2=1/t2 : v3=v1+v2 : t3=1/v3
40 PRINT "Вместе выроют за";t3;"дней"
```

Вы заметили, что в 30 строке выполняется не одна а несколько команд. Бейсик позволяет, разделяя двоеточием, разместить в одной строке много команд. Можно даже продлить строку программы больше, чем длина строки на экране (Бейсик сам перенесёт продолжение на следующую строчку). Всего строка программы может содержать (вместе с номером) до 255 букв, цифр и других символов. Таким образом, наш пример мог бы быть записан одной строкой программы:

```
10 INPUT "Введите время для 1-го землекопа";t1:INPUT "Введите время для 2-го землекопа";t2:v1=1/t1:v2=1/t2:v3=v1+v2:t3=1/v3:PRINT "Вместе выроют за";t3;"дней"
```

Не подумайте только, что я призываю вас писать программы именно так. Программа в таком виде трудночитаема, ошибки в ней искать сложнее, а допустить легче. К такому уплотнению прибегают в том случае, если длина программы во много раз больше, чем высота экрана, а вы хотите, чтобы какой–то фрагмент программы уместился целиком в одном экране.

Урок 3

Как запоминается число в переменной, вы уже усвоили. В переменной может храниться не только число, но и текст. Только переменная для этого должна быть со специальным знаком [$] в конце. Например,

```
10 t$="День рождения кота Леопольда"
20 PRINT t$
```

Обратите внимание: текст справа заключён в кавычки. Собственно, текстом является не только набор букв, но и цифр и других символов, лишь бы он был заключён в кавычки. В программе:

```
10 k$="2+2="
20 PRINT k$
```

выражение "2+2=" является текстом, и PRINT печатает не 4, а 2+2=. То–есть, на содержимое текста в кавычках Бейсик не обращает внимание, а просто сохраняет и печатает все без изменений.

Как и с числами, с текстами возможны некоторые операции.

Например, тексты можно склеивать. Для этого между текстами ставится знак [+]. Например,

```
10 t$="День "
20 p$="рождения кота Леопольда"
0 k$=t$+p$
40 PRINT k$
```

Кстати, в отличие от арифметического сложения, операция склейки не обладает переместительным свойством, то–есть, t$+p$ не равно p$+t$ (Можете попробовать в 30–й строке изменить порядок переменных (p$+t$) и посмотрите результат.)

Текстовые величины можно не только печатать, но и запрашивать у пользователя. Благодаря этому возможны диалоги:

```
10 PRINT "Я - ЯМАХА."
20 input "А ты кто";a$
30 PRINT "Я понял. Ты - ";a$
```

Вообще, работа с текстами очень важна. Благодаря текстовым возможностям компьютера он становится пригодным не только для счета, но и для обработки текстов, что повышает «разум» компьютера. Для обработки текстов в Бейсике есть специальные функции: mid$(), left$(), right$(), len(), val(), str$(),...

Функцией называется служебное слово со скобками. В скобках обычно пишут те величины, над которыми производится операция. Если в конце служебного слова (имени функции) стоит [$], то результатом работы функции является текст. Если [$] отсутствует, значит функция — числовая. Рассмотрим примеры:

Нужно из введённого слова напечатать только 6 первых букв. Если в слове меньше 6 букв, нужно напечатать их все.

Функция `LEFT$(A$,n)` как раз вырезает из текста A$ первые n символов. Если их там меньше, чем n, берутся все, что имеются.

```
10 input"Введите текст ";a$
20 p$=left$(a$,6)
30 PRINT "Получится ";p$
```

Ещё пример: Пусть введут 2 слова, а программа склеит из первых 3–х букв первого и первых 3–х букв 2–го слова новое слово:

```
10 input"Введите 1-е слово ";a$
20 input"Введите 2-е слово ";b$
30 p$=left$(a$,3)+left$(b$,3)
40 PRINT "Получится ";p$
```

Следующий пример: Нужно ввести слово, а напечатать первую его половину. (Если число букв нечётное, то нужно взять без одной буквы, например, из слова «соковыжималка», где 13 букв и в середине слова буква «ж», нужно вырезать до «ж»:«соковы».)

В этом примере нам понадобится функция `LEN(A$)`, которая находит длину текста A$. Например, команда `PRINT len("Соковыжималка")` напечатает 13 (длину текста)

Нашу задачу решит такая программа:

```
10 input"Введите слово";e$
20 dl=len(e$)\2:'Нашли длину половины слова
30 r$=left$(e$,dl):'Вырезали
40 print r$
```

Обратили внимание, что в 20 строке использовано деление нацело? Это как раз для случая с нечётным числом букв.

Может быть, вы заметили, что в программе после знака ['] стоят пояснения? Такие пояснения называются комментарием, и его Бейсик вообще не читает. Мой совет: в больших программах не жалейте место для комментариев. При повторном чтении программы вам будет легче вспомнить, для чего нужна та или иная часть программы. В дальнейшем и мы в примерах часть объяснений введём в комментарий. При проверке примеров вы можете комментарии пропускать.

Теперь такая задача: Из первого введённого слова взять первую половину, а из второго слова — последнюю половину. Для получения правой части слова используем функцию `RIGHT$(a$,n)`.

```
10 input"Введите 1-е слово ";a$
20 d1=len(a$)\2
30 input"Введите 2-е слово ";b$
40 d2=len(b$)\2
50 p$=left$(a$,d1)+right$(b$,d2)
60 PRINT "Получится ";p$
```

Составьте программу, которая запрашивает слово и печатает его первую половину, приклеивает к ней «ынык» и доклеивает вторую половину слова. Например, «коробка» — «корыныкбка».

Функция `MID$(A$,k,n)` из текста A$ вырезает n букв, но не первых или последних, а начиная с k–того номера. Например,

```
10 input"Введите слово ";a$
```

```
20 p$=mid$(a$,3,5)
30 PRINT "Получится ";p$
```

напечатает 5 букв из введённого текста, начиная с 3–й.

Пример простого диалога:

```
10 input "Напиши твое любимое слово";sl$
20 input "Каким цветом его напечатать";c
30 color c
40 print sl$,sl$,sl$,sl$,sl$,sl$,sl$,sl$
```

В этом фрагменте использована новая команда `COLOR`. Она назначает цвет букв. Например, `COLOR 10` делает цвет букв жёлтым. Номера цветов 1 … 15 — от чёрного до белого. Если в команде `COLOR` после запятой стоит второе число, оно задаёт цвет экрана. Например, `COLOR 15,4` определяет белые (15) буквы на синем (4) фоне.

> Измените последний пример, чтобы он запрашивал не только цвет букв, но и цвет фона.

> Напишите программу, которая запрашивает у пользователя 4 слова, вырезает из каждого первую букву и склеивает их в одно 4–буквенное слово, которое печатает.

Урок 4

Все примеры, которые мы делали до сих пор, представляли собой линейные программы, то–есть, программы, в которых все команды выполняются подряд, одна за другой. Но часто встречается необходимость изменить порядок следования программы.

Например, нам нужно, чтобы после 40 строки программа перешла сразу в 60–ю, не выполняя 50–й строки. Для этого используется команда перехода `GOTO`. Например, в программе:

```
10 print "Посмотрите: я считаю"
20 goto 40
30 print"Лучше вашего калькулятора"
40 print "2+2=";2+2
```

строка 30 не выполнится: программа обошла её с помощью `GOTO`.

`GOTO` может переводить не только вперёд по тексту программы, но и назад, к уже выполнявшимся строкам:

```
10 print "Посмотрите: я считаю"
20 goto 10
30 print "2+2=";2+2
```

В этом случае до строки 30 программа никогда не доберётся: дойдя до 20 строки, она возвращается в 10, затем снова в 20 и 10… Такой участок программы, который повторяется с помощью `GOTO`, называется бесконечным циклом. Его прервать можно аварийным способом: нажатием одновременно клавиш CTRL и STOP.

Ещё пример бесконечного цикла:

```
10 print "У попа была собака. Он ее любил."
20 print "Она съела кусок мяса. Он ее убил."
30 print "В яму закопал. На камне написал:"
40 goto 10
```

При всей на первый взгляд бессмысленности бесконечные циклы используются очень часто. Они имеют смысл в тех случаях, когда при повторении действия выполняются не одинаковые.

Например, когда действия зависят от реакции человека. Так большинство игр представляют собой бесконечный цикл: окончившись, начинаются снова. Попробуйте такой пример:

```
10 print "купи слона!"
20 input otw$
30 print "Все говорят ";otw$;", а ты "
40 goto 10
```

Более сложный пример бесконечного цикла:

```
10 print "Предлагаю научиться считать время."
20 print "Я буду считать 50 раз в секунду,"
30 print "А ты нажмешь клавишу <ВВОД>, когда"
40 print "по-твоему я насчитаю 6000. Понял?"
50 input otw$60%
60 print "Начинаю считать до 6000. Если го-"
70 print "тов, жми <ВВОД>."
80 input otw$
90 time=0
100 input"Для остановки жми <Ввод>";otw$
110 print "Насчитал";time
120 input "Для продолжения нажми <ВВОД>";otw$
130 goto 60
```

В этом примере есть кое–что новое. Вомпервых, мы воспользовались тем, что Бейсик имеет специальную переменную TIME, в которую он (независимо от наших действий) 50 раз в секунду добавляет единицу. Поэтому после нажатия ⏎ мы печатаем в строке 110 новое значение TIME, накопленное за время ожидания команды INPUT из 100 строки. При повторении мы снова обнуляем в строке 90 переменную TIME и накопление начинается сначала. Ещё заметим, что в строках 50, 80 и 100 мы в команде INPUT ожидаем ответа, но никак этот ответ не используем. INPUT в этом примере служат только для задержки до нажатия клавиши ⏎.

Конечно, хорошо бы, если бы ответы человека можно было использовать. Например, в 120 строке спросить:«Продолжим или нет?», и если пользователь ответит «да», то GOTO 60, а в противном случае закончим программу. Измените в своей программе такие строки:

```
120 INPUT "Продолжим (да/нет)";OTW$
130 IF OTW$="да" THEN GOTO 60 ELSE PRINT "Конец!"
```

Новая команда (проверки условия) состоит из 3–х служебных слов: IF (если), THEN (то), ELSE (иначе).

После IF стоит проверяемое условие. Примеры возможных условий:

- IF 2.5=X+Y
- IF X>5+2*Y
- IF A+B⇐14 ([⇐] значит меньше или равно)
- IF A$="привет"
- IF B$<>A$ ([<>] значит не равно)

После THEN стоят одно или несколько (через [:]) действий, которые будут выполнены, если условие после IF истинно. Все они должны разместиться в одной строке программы. Если таких действий много, и в этой строке они не помещаются, то можно поставить GOTO и переслать на строку с нужными действиями. (Кстати, вместо THEN GOTO 60 можно написать просто THEN 60)

После ELSE находятся те команды, которые должны быть выполнены в том случае, когда условие после IF не истинно. Если все команды в строку не входят, можно тоже употребить ELSE GOTO 100 или просто ELSE 100.

ЗАПОМНИТЕ: слева или справа от слов THEN и ELSE не ставят двоеточия — это не отдельные команды, а части команды IF.

Примеры программ с проверкой условия:

```
10 input "Введите Ваш любимый цвет";c
20 if c<0 or c>15 then 10 else color 15,c
```

Мы проверили здесь, чтобы введённый номер цвета был в допустимых пределах.

```
10 print "Я работаю только со своим программистом!"
20 input "Введите Ваше имя";n$
30 if n$="Виктор" then print "Привет, хозяин!" else print "Вы не мой хозяин":goto 10
```

Эта программа сработает лишь при точном совпадении ответа с ожидаемым текстом. Программу можно усовершенствовать, если предусмотреть несколько имён. Для этого в строке 30 нужно поменять условие:

```
30 if n$="Виктор Алексеевич" or n$="Дима" or n$="Шурик" then print "Привет, хозяин!" else print "Вы не мой хозяин":goto 10
```

Служебное слово OR в условии обозначает ИЛИ: условие будет в целом истинным, если выполняется хотя бы одно из условий, соединённых словом OR.

Обратили ли вы внимание, что в этом примере тоже используется цикл, но он уже не является бесконечным: имеется условие его окончания: когда будет названо правильное имя.

Переделайте программу «Купи слона» так, чтобы при каком–то определённом ответе цикл заканчивался.

```
10 'Непедагогичное воспитание
20 input"Шлеп! Будешь слушаться (да/нет)";a$
30 if a$="да" then 60 else if a$="нет" then 40 else 50
40 print "Ах, так!": goto 20
50 print "Я не понял!":goto 20
60 print "Вот то-то же!"
```

В этом примере вы увидели, что проверка условия может быть и не простая: если первое условие не выполнилось, мы новым IF продолжаем проверку другого условия. Следующий пример развивает этот способ:

```
10 'Электронный продавец
20 S=0:'Начинаем с нулевой суммы
30 cls:print"СПИСОК ТОВАРОВ:"
40 print"1. Шашлык свежий    - 200 р."
50 print"2. Хлеб    - 5  р."
60 print"3. Сок гранатовый - 40  р."
70 print"4. Шоколад импортный - 240 р."
80 print:print:input"Введите выбранный номер";N
90 input"Сколько штук";K
100 if N=1 then P=200 else if N=2 then P=5 else if N=3 then P=40 else if N=4 then P=240 else
30:'ошибочный N 110 S=S+P*K:'Увеличили общую сумму на стоимость покупки
120 input"Продолжим (да/нет)";OTW$
130 if OTW$="да" or OTW$="ДА" or OTW$="Да" then 30
140 print "Заплатите";S;"рублей в кассу"
```

Здесь нужно остановиться на нескольких моментах:

Во–первых, обратите внимание, как накапливается сумма: в 0 строчке ей даётся начальное значение 0, а в 110 она увеличивается на стоимость текущей покупки. Поступайте таким способом всегда, когда нужно какую–то величину наращивать от заданного начального значения.

Во–вторых, такую проверку, как в 130 строке делайте тогда, когда нет уверенности, ответит ли пользователь строчными или заглавными буквами.

В–третьих, вы заметили, что в 130 строке при проверке отсутствует ELSE. Дело в том, что при проверке IF можно

Fix Me!

Урок 5

Вы уже познакомились с циклом, который организуется с помощью `GOTO`. Иногда такой цикл бывает бесконечным, но чаще всего в цикле имеется условие, которое проверяется при каждом повторении, и если оно выполнилось, цикл заканчивается, и программа продолжается после цикла. Условия окончания цикла могут быть разными. Если это условие определяется ответом пользователя, или случайным числом, или другим труднопредсказуемым событием, то `GOTO` и `IF` — лучший способ для организации такого цикла. Но часто встречаются такие циклы, окончание которых определяется значением переменной, которая при каждом повторении меняется на одну и ту же величину. Поясню на примерах:

Программа печатает квадраты всех чётных чисел от 40 до 76:

```
10 x=40
20 if x > 76 then end else print x^2
30 x=x+2:goto 20
```

Fix Me!

```
└──────────────────────────────────────────────────────────────────────┘
```

Единица в скобках после `input$` обозначает, что функция закончит работу после первой же нажатой клавиши. Если бы стояло `input$(2),` то функция ждала бы нажатия второй клавиши, а затем в переменную `a$` записала бы оба нажатых символа.

Fix Me!

Следующая программа печатает подряд все числа, пропуская кратные 5–ти. Чтобы это проверить, достаточно найти остаток от деления числа на 5. Если он =0, то число кратно 5:

```
10 for m=0 to 200
20 if m mod 5>0 then print m :'Если не кратно 5
30 next
```

Если положить сумму в банк с процентной ставкой 20% годовых, то по истечение года сумма возрастёт на 20%. К концу следующего года УЖЕ НОВАЯ сумма возрастёт на 20% и т.д. Составим программу, которая запрашивает начальную сумму, процентную ставку и количество лет, а вычисляет конечную сумму.

```
10 input"Начальная сумма";s
20 input"Процентная ставка  (%)";n
30 n=1+n/100:'Во столько раз возрастает за год
40 input"На сколько лет";y
50 for l=1 to y:'Начинаем считать годы
60 s=s*n:next:print"Сумма=";s
```

Поясню, откуда взялась 30 строка: Если процентная ставка 53%, то за год сумма возрастёт на 0.53 от исходной суммы, или, что то же самое — в 1.53 раза = в 1+53/100 раза или в общем случае — в 1+n/100 раз.

Кстати, в этом примере можно обойтись вовсе без цикла, если увидеть, как возрастает каждый год начальная сумма:
За 1 год — в 1.53 раза
За 2 года — в 1.53 * 1.53
За 3 года — в 1.53 * 1.53 * 1.53 или 1.53^3
И т. д. ...
За Y лет — в 1.53^Y раза или, для общего случая, в (1+n/100)^Y раз — и не нужно цикла:

```
10 input"Начальная сумма";s
20 input"Процентная ставка  (%)";n
30 input"На сколько лет";y
40 s=s*(1+n/100)^y
50 print"Сумма=";s
```

Циклами очень удобно обрабатывать тексты. Рассмотрим такую программу: Пользователь вводит текст, например, «Победа», а программа печатает сначала все слово, затем без первой буквы, без второй и т. д.:

```
Победа
обеда
беда
еда
да
а
```

```
10 input "Введите слово "; A$
20 L=len(A$):' Нашли длину текста
30 for I=0 to L-1:print right$(A$,L-I):next
```

В 30–й строчке с помощью `right$()` вырезаем правую часть введённого слова, длиной L-I. Например, когда в цикле переменная I дойдёт до 5, будет вырезано L-5 букв, то–есть, от 5–й до L–й (последней в тексте).

Следующая программка отрезает от слова последние буквы:

```
10 input "Введите слово "; A$
20 L=len(A$)
30 for I=L to 1 step -1:print left$(A$,I):next
```

Нетрудно слово и вовсе разобрать по отдельным буквам:

```
10 input "Введите слово "; A$
30 for I=1 to len(a$):print mid$(A$,I,1):next
```

Внимательно разберитесь с последним примером и не двигайтесь дальше, пока не выполните задание:

> Составить программу, получающую слово и печатающую отдельные буквы этого слова в обратном порядке.

Следующий пример такой: программа должна получить слово и напечатать его задом наперёд. Не отдельные буквы, как в предыдущем примере, а слово!

```
10 input "Введите слово "; A$
20 B$="":'Заводим другую текстовую переменную, в которую будем накапливать новое слово
30 for I=L to 1 step -1:B$=B$+mid$(A$,I,1):next
40 print B$
```

Две кавычки, которые использованы в 20–й строке подряд (без пробела) называются пустым текстом. Мы берём пустой текст в переменную B$ для того, чтобы приклеивать к нему букву за буквой, вырезая их из текста A$, начиная с конца.

Пример немного сложнее: Программа должна ввести предложение пользователя и разделить его на отдельные слова.

В качестве разделителей между словами могут быть пробел, запятая, двоеточие и другие знаки препинания.

Программа накапливает новое слово, просматривая текст буква за буквой. Когда встретится разделитель, слово напечатается, обнулится и будет снова накапливаться.

```
10 input "Введите текст ";A$: L=len(A$)
20 B$="":for I=1 to L: K$=mid$(A$,I,1)
30 if K$=" " or K$="," or K$="." or K$=":" then print B$:B$="" else B$=B$+K$
40 next
```

Хотя явных ошибок в этой программе нет, не если её запустите, вы увидите, что она работает только тогда, когда в предложении нет знаков препинания. Если встречается запятая, все, что в тексте после запятой, отбрасывается. При этом Бейсик сообщает:

?EXTRA IGNORED (лишнее пропускаю).

Причина этого — особенности INPUT. Если после INPUT имеется не одна, а несколько переменных, INPUT может принять сразу несколько чисел или текстов, разделённых запятой. Например,

```
input "Введите два слова ";A$,B$
```

вводит два текста.

В предыдущей программе, когда текст включал запятые, они воспринимались INPUT не как часть текста, а как разделитель между двумя текстами для двух переменных. Этого недостатка лишена другая команда:LINE INPUT (ввод строки). Она беднее INPUT, так как не принимает чисел, а только текст (причём, только один!). Зато ей безразличны запятые в этом тексте. Измените 10–ю строку в программе:

```
10 line input "Введите текст ";A$: L=len(A$)
```

Но и изменённая программа не лишена недостатка: Если в конце предложения не стоит точка, последнее слово не напечатается (ведь программа печатает слово в момент, когда найден разделитель). Эту ошибку устранить и вовсе легко: если последняя буква не «.», «!» или «?», добавим к тексту пробел.

```
10 line input "Введите текст ";A$:K$=right$(A$,1):if K$<>"." and K$<>"!" and K$<>"?" then A$=A$+"
"
20 B$="":for I=1 to len(A$): K$=mid$(A$,I,1)
30 if K$=" " or K$="," or K$="." or K$=":" then print B$:B$="" else B$=B$+K$
40 next
```

У этой программы есть ещё недостатки. Всех их устранять мы не будем, но ещё один исправим. Обратите внимание: два раза встречается проверка, является ли K$ разделителем. Из–за громоздкости проверки мы не все возможные знаки проверяем.

Воспользуемся функцией INSTR(T1$,T2$). Она проверяет, является ли текст T2$ частью текста T1$. Если T2$ не входит в T1$, то функция даёт 0, а если входит, то значение функции равно номеру позиции, начиная с которой T2$ входит в T1$. Например,

```
print INSTR("Барабанщик","ба")
```

напечатает 5 (а не 1, так в начале слова — «Б» а не «б»), а

```
print INSTR("Барабанщик","ва")
```

напечатает 0, так как «ва» в слово не входит.

В нашей программе мы с помощью INSTR можем проверить, входит ли K$ (вырезанная буква) в список разделителей:

```
10 line input "Введите текст ";A$:K$=right$(A$,1):if instr(" ,.!?-:",K$)=0 then A$=A$+" "
20 B$="":for I=1 to len(A$): K$=mid$(A$,I,1)
30 if instr(" ,.!?-:",K$) then print B$:B$="" else B$=B$+K$
40 next
```

Проверка получилась меньше, а знаков проверили больше. 10–ю строчку можно ещё сократить, если внести RIGHT$ прямо в аргумент INSTR: получится функция от функции. Так можно!

```
10 line input "Введите текст ";A$:if instr(" ,.!?-:",righ t$(a$,1) )=0 then A$=A$+" "
```

В 20 и 30 строчках вносить функцию MID$() в функцию INSTR() не нужно, так как K$ используется не только в INSTR(), но и дальше.

Торопливый читатель упрекнёт меня, зачем столько времени доделывать и переделывать одну и ту же программу? Не проще ли сразу дать готовую в окончательном виде, а потом объяснить?
Ответ: я показываю, как работает программист: большую часть времени у программиста занимает не написание, а совершенствование программы.

Ещё раз напомню: сия книжка — не художественная литература, которую проглатывают за один раз! Читайте, перечитывайте помногу раз объяснения, разбирайтесь в примерах, и тогда от вашей работы будет толк!

Урок 6

Много хороших программ можно сделать, если научиться размещать печатаемый текст в нужном месте экрана. Например, можно заставить текст ползти по экрану, можно печатать не только слева направо, но и справа налево, сверху вниз, под углом...

Можно организовать на экране табличку и аккуратно впечатывать числа или тексты в клетки этой таблицы... Можно, наконец, делать простенькие игры, в которых движутся не рисунки, слова. Чтобы этому научиться, нужно разобраться, как организован экран. На экране помещаются 24 строки текста. Они пронумерованы: первая имеет номер 0, 24–я — номер 23. Обычно в 23–й строке находятся подсказки, и ваша программа ею пользоваться не может. Чтобы убрать подсказки, используют команду `KEYOFF`, тогда 23–ю строку можно использовать. Если нужно восстановить строку с подсказками — команда `KEYON`. Сколько помещается в каждой строке символов, зависит от того, какие установлены экранный режим и ширина экрана. Экранный режим устанавливается командой `SCREEN`:

- `SCREEN 0` — до 80 символов в строке.
- `SCREEN 1` — до 32 символов в строке.

При включении обычно автоматически устанавливается `SCREEN 0`.

После установки режима, командой `WIDTH` можно установить ширину экрана. Например, `WIDTH 80` устанавливает 80 символов в строке (мелкий шрифт). Такая команда не сработает в 1–м экранном режиме (Максимально `WIDTH 32`). Команда `WIDTH 40` в 0–м режиме установит 40 символов в строке (крупный шрифт).

Все дальнейшие примеры (если не оговаривается особо) приведены из расчёта на 0–й режим, 80–символьный экран и пустую служебную строку.

Команда `LOCATE` позволяет указать, где будет печатать следующий после неё `PRINT`. Например,

```
LOCATE 30,12:PRINT "Привет"
```

напечатает «Привет», начиная с 30–й позиции в 12–й строке. (Вы не забыли, что нумерация идёт от 0?)

Если в `LOCATE` использовать не числа, а переменные, то можно в цикле печатать красивые вещи. Например, напечатаем уголком слово «привет»:

```
10 for i=0 to 10:locate i,i:print"Привет!":next
20 for i=0 to 10:locate 10-i,11+i:print"Привет!":next
```

Посмотрите, как программка работает. В 10–й строке `LOCATE` в качестве номера строки и номера позиции использует одно и то же число (i). В 20–строке мы хотим, чтобы позиция уменьшалась, а номер строки увеличивался. Поэтому рассчитываем их по–разному.

Если хотим растянуть по X в 2 раза, нужно номер позиции умножить на 2 (а номер строки не умножать):

```
10 for i=0 to 10:locate i*2,i:print"Привет!":next
20 for i=0 to 10:locate 20-i*2,11+i:print"Привет!":next
```

Напечатаем из левого верхнего в правый нижний угол:

```
10 for i=0 to 23:locate i*3.5,i:print"ШШШШШШШ":next
```

Выполнив пример, вы заметите, что когда программа допечатает до низа, весь экран дёрнется вверх. Одна из причин этого — та, что Бейсик, окончив программу, печатает «Ok» и при этом переводит строку, поднимая экран. Чтобы этого не было, зациклим программу, то–есть, добавим к ней строчку:

```
20 goto 20
```

Но даже и в этом случае экран дёрнулся вверх, хотя и меньше, чем раньше. Виноват `print`, который после последней строчки переводит строку. Если вы добавите после ...ШШШ" знак <;>, то все будет в порядке.

Если вы разобрались в этом примере, то попробуйте напечатать так же под углом, но снизу вверх.

Ещё труднее, но вы постараетесь, напечатать текст справа налево сверху вниз.

Сделаем программу, которая пишет текст под каким–нибудь углом, и, дойдя до границы экрана, отражается от не1, как на рисунке:

```
┌────────────────────лл──────────лл┐
│          ллллл      ллллл
│  лл       лл лл     лл л
│   лл      лл  лл    лл лл
│    лл     лл   лл     лл│
│     лл    лл    лл    лл│
│      лл   лл     лл   лл│
│       лл лл       лл лл │
│        ллллл       ллллл │
└──────────лл────────────────лл──────┘
```

Поскольку количество шагов не задано, лучше использовать не цикл `FOR`, а `GOTO`, и координаты X и Y для `LOCATE` рассчитывать независимо друг от друга. Рассмотрим последовательность действий:

1. Заведём переменные X и Y для `LOCATE`.
2. Заведём переменные DX и DY, которые показывают, на сколько изменяются X и Y в каждом шаге.
3. Напечатаем слово в позиции X,Y
4. Изменим X на DX и проверим, не зашли ли на левый или правый край экрана. Если зашли, нужно поменять DX на -DX, чтобы двигаться в противоположном направлении.
5. Изменим Y на DY и проверим так же верхнюю и нижнюю границы экрана.
6. Для продолжения вернёмся к п.3 и все повторим.

```
10 screen 0:keyoff:width 80
20 x=10:y=10:dx=1:dy=1
30 locate x,y:print"*";
40 x=x+dx:if x>78 or x<1 then dx=-dx
50 y=y+dy:if y>22 or y<1 then dy=-dy
60 goto 30
```

Всем бы хороша эта программа, но через некоторое время в Fix Me!

Урок 7

Часто бывает так, что какой–либо цикл нужно повторить несколько раз. Для этого его помещают в другой цикл. Получается цикл в цикле или вложенные циклы.

Например, нужно сделать на экране 5 строк, в каждой из которых имеется десять чисел (1,3,5,... каждое на 2 больше предыдущего).

```
10 screen0:width80:keyoff
20 for st=1 to 5 :'Изменяется номер строки
30 for po=1 to 10:'Изменяем номер позиции
40 locate po*4,st:print po*2-1;
50 next:next:'Закрыли оба цикла
```

В 40–й строке мы умножили PO на 4 в `LOCATE`, чтобы цифры располагались не подряд, а через 4 позиции (иначе слепятся).

При изменении PO на 1 мы хотим печатать числа, которые отличаются на 2. Поэтому в `PRINT` увеличиваем PO в 2 раза.

Но тогда будут числа 2,4,6,... Чтобы были 1,3,5... отнимаем 1.

Почти так же делается таблица Пифагора (таблица умножения), в которой каждое число равно произведению номера строки, на номер столбца, в котором оно находится.

```
10 screen0:width80:keyoff
20 for st=1 to 9 :'Изменяется номер строки
30 for po=1 to 9 :'Изменяем номер столбца
40 locate po*4,st:print po*st;:next:next
```

Пора, читатель, делать задания немного посложнее. Сделайте программу, которая печатает на экране ту же таблицу Пифагора но с рамкой и заголовками сверху и слева:

```
   1  2  3  4  и т.д.
  ┌──────────────────
 1│ 1   2   3   4
 2│ 2   4   6   8
 3│ 3   6   9   12
 4│ 4   8   12  16
```

Если вы справились с этим заданием, то разберётесь в программе, которая двигает звёздочку сверху вниз в первой колонке, затем сверху вниз во второй, ...

```
10 screen0:width80:keyoff:xs=0:ys=0
20 for x=0 to 78:for y=0 to 21
30 locate x,y:print"*";:'нарисовали
40 locate xs,ys:print" ";:'стерли
50 xs=x:ys=y:next:next
```

В 50–й строке переназначаются xs,ys — координаты места для стирания.

Вы поняли приведённый пример, если вы сумеете его переделать так, чтобы звёздочки двигались горизонтально, сначала вдоль самой верхней строки, потом ниже, ниже...

Несколько труднее сделать, чтобы звёздочка сначала двигалась вниз, а в следующем столбце — вверх и т.д.

```
10 screen0:width80:y=0:xs=0:ys=0:dy=1
20 for x=0 to 78:for k=0 to 21:y=y+dy
30 locate x,y:print"*";
40 locate xs,ys:print" ";
50 xs=x:ys=y:next:dy=-dy:next
```

Как видите, в 20 строке K — не координата Y, а просто счётчик. Y же равен вначале нулю, а в цикле меняется не на 1, как K, а на DY, который (в конце 50–й строки) принимает значения либо 1 (вниз), либо -1 (вверх).

Считайте, что вы поняли этот пример, а также пример со змейкой из 6–го урока, если вы сделаете в нем движение не одной звёздочки, а змейки из звёздочек.

Ещё пример на вложенные циклы: Пусть программа печатает в одной строке ряд чисел: 1,8,15,...,71, в следующей строке — ряд 2,9,16,...,72, дальше 3,10,17,...,73 и так до строки с рядом 9,16,...,79.

Первая переменная цикла (A) будет задавать номер строки, а тем самым — начальное значение ряда. Вторая (B) будет задавать числа в ряду и изменяться от A до A+70 с шагом 7:

```
10 for a=1 to 9
20 for b=a to A+70 step 7:print b;:next:print:next
```

В 20–й строке пустой PRINT — чтобы перейти на новую строку.

Сделайте подобную программу, чтобы она печатала такие ряды:

1,4,7,…,34
4,7,10,…,37
7,10,13,...,40 и т.д. ... до ряда 34,37,40,...,67

Урок 8

Представьте себе случай, когда одинаковые действия нужно выполнить для множества данных. Например, каждое из 20–ти чисел нужно увеличить в 2 раза. Ясно, что эту работу нужно делать в цикле. Для этого есть несколько способов:

Первый способ очень прост. Мы в цикле запрашиваем данные у пользователя и тут же их обрабатываем. Например, пусть программа сложит 10 чисел, введённых пользователем:

```
10 S=0:for n=1 to 10
20 input"Введите число";a
30 S=S+a:next
40 print "Сумма всех чисел =";S
```

Если данные заранее известны программисту, их удобнее поместить в программу, а не запрашивать командой INPUT. Делают это с помощью команд DATA (данные) и READ (читать): После слова DATA размещаются через запятую нужные данные. После слова READ стоит имя переменной (или несколько, через запятую), куда нужно занести очередное данное. Не обязательно все данные размещать в одной строке программы. DATA могут находиться в любом месте программы. Нужно только помнить, что READ считывают из DATA, начиная с самой первой встреченной в программе и до самой последней подряд. Если команд READ больше, чем DATA, то будет выдано сообщение об ошибке (OUT OF DATA — ЗА ПРЕДЕЛАМИ ДАННЫХ). Если данные считываются в цикле, то это значит, что число повторений больше, чем данных.

Посмотрите предыдущий пример с использованием READ и DATA:

```
10 S=0:for n=1 to 10:read a:S=S+a:next
20 print "Сумма всех чисел =";S
30 data 1,4,23,11,4,52,4,2,5,2
```

Нередко бывает, что данные нужно обработать вторично. Но ведь после первой обработки READ установлен на последнюю DATA!

Чтобы его переустановить на начало, используется команда RESTORE (переустановка). Если нужно повторить обработку данных не с самого начала программы, а с определённой строки, после RESTORE пишут номер строки, где начинаются нужные данные.

Например, чтобы звёздочка ходила по кругу зададим ей координаты в DATA, а когда опишем полную окружность, сделаем RESTORE:

```
10 color15,1,1:screen1:keyoff:xs=0:ys=0
20 for i=1 to 21: read xn,yn:locate xn,yn:print"*"
30 locate xs,ys:print" ";:xs=xn:ys=yn:next
40 restore:goto20
50 data 29,11, 29,14, 27,17, 25,19, 23,20, 20,21, 17,21
60 data 14,20, 12,18, 10,15, 9,12,  9,9,   10,7,  12,4
70 data 14,2,  17,1,  20,1,  23,2,  25,3,  27,5,  29,8
```

Здесь одним READ читаем 2 данных XN и YN. Старые координаты (для стирания) держим в переменных XS,YS. Кстати, SCREEN1 в этом фрагменте выбран из тех соображений, что в нем каждое знакоместо — квадрат. В screen 0 окружность превращается в эллипс, так как знакоместо заужено по X.

В качестве следующего примера, где READ читает не только числовые, но и текстовые величины, приведу программу медленной (буква за буквой) печати текстов, которые записаны в DATA:

```
10 screen0:width80:keyoff
20 read n:'Взяли количество строк
30 for i=1 to n:read a$:'Взяли текст строки
40 for k=1 to len(a$):b$=mid$(a$,k,1):'взяли букву
50 print b$;:fork=0to50:next:'поставили букву и задержались
60 next:d$=input$(1):'Написали строку. Ждем нажатия клавиши
70 print:next:'перевели строку и повторим со следующей
80 print:print"До свидания":end:'Закончив, попрощались
90 '--------------- сам текст ----------------------
100 data 4,"Однажды, в студеную зимнюю пору"
110 data "Я из лесу вышел, был сильный мороз."
120 data "Гляжу, поднимается медленно в гору"
130 data "Лошадка, везущая хворосту воз."
```

Вы поняли, для чего стоит 4 в 100–й строке? Правильно, для первого READ.

READ и DATA — очень удобны, но лишь в тех нечастых случаях, когда все данные нужно читать и обрабатывать подряд. А как быть, когда надо через одно? А если с конца в начало?

Во многих задачах вас выручат массивы. Массив — это табличка в памяти, где хранятся пронумерованные данные. У массива есть имя. Например, массив A. Но в отличие от переменной с именем A, в массиве под этим именем хранится не одно число, а несколько.

По имени массива и номеру можно получить любое данное. Например, если мы хотим напечатать 20–е число из массива WWW, то мы напишем: print www(20) . Для того, чтобы запомнить число в нужное место массива, нужно как и с переменной использовать присваивание: www(20)=62.11. Номер элемента массива может быть записан не только числом, но и переменной, в которой находится это число. Это позволяет использовать массив в цикле, где индекс (номер в скобках) меняет своё значение. Например, посмотрите, как выполняется такая задача: программа получает от пользователя 10 чисел, а затем печатает их в строку в обратном порядке:

```
10 dim a(9)
20 for i=0 to 9:input a(i):next
30 for i=9 to 0 step -1:print a(i):next
```

Как видите, программа несложная. Остаются неясными детали: В 10–й строке стоит команда DIM, которая объявляет номер последнего элемента массива. С этого момента Бейсик зарезервировал 10 (от 0 до 9) элементов и заполнил их нулями. В 20–строке вместо нулей мы INPUT заносим новые значения, а в 30–й строке печатаем в обратном порядке.

Если программа попытается использовать элемент массива с номером, большим, чем объявлено в DIM, то Бейсик сообщит об ошибке: «SUBSCRIPT OUT OF RANGE» или «ИНДЕКС ЗА ПРЕДЕЛАМИ РЯДА».

Составим такую программу: программа заносит из DATA в массив 10 чисел и печатает в одну строку на экран, а затем все чётные делит на 2 и снова весь массив печатает на экран одной строкой.

```
10 dim a(9)
20 for n=0 to 9:read a(n):print a(n);:next
30 for n=0 to 9:if a(n) mod 2=0 then a(n)=a(n)/2
40 print a(n);:next
50 data 1,4,23,11,4,52,4,2,5,2
```

В 30–й строке проверяется, чётное ли число (для чётных остаток от деления на 2 равен 0). Если чётное, то делим на 2. В 40–й строчке печатается a(n), независимо от того, делили его на 2 или нет. У вас может возникнуть вопрос: почему в 40–й строке после окончания печати не стоит команда END? Дело в том, что когда окончится цикл, программа перейдёт на последнюю, 50–ю строку, и, ничего не сделав, закончится. (Команда DATA не выполняет никаких действий. Она просто хранит данные для READ. Поэтому команды DATA можно размещать в любом месте программы.)

Попробуем с помощью массивов улучшить программу, в которой звёздочка движется по кругу. Занесём координаты в массив. Обращение к массиву происходит быстрее, чем READ, поэтому движение ускорится. Кроме того, можно пустить звёздочку в обратном направлении:

```
10 color15,1,1:screen1:keyoff:xs=0:ys=0
20 for i=1 to 21: read x(i),y(i):next
30 for i=21 to 1 step -1:locate x(i),y(i):print"*"
40 locate xs,ys:print" ";:xs=x(i):ys=y(i):next
50 goto20
60 data 29,11, 29,14, 27,17, 25,19, 23,20, 20,21, 17,21
70 data 14,20, 12,18, 10,15, 9,12,  9,9,   10,7,  12,4
80 data 14,2,  17,1,  20,1,  23,2,  25,3,  27,5,  29,8
```

Если обойтись без FOR в 30 строке, сделать цикл по старинке, с помощью GOTO, то можно организовать одновременное движение двух звёздочек по– и против часовой стрелки:

```
10 color15,1,1:screen1:keyoff:x1=0:y1=0:x2=0:y2=0
20 dim x(21),y(21):for t=1 to 21: read x(t),y(t):next
30 t=1:k=21
40 locate x(t),y(t):print"*";:locate x1,y1:print" ";
50 x1=x(t):y1=y(t):t=t+1:if t=22 then t=1
60 locate x(k),y(k):print"Ш";:locate x2,y2:print" ";
70 x2=x(k):y2=y(k):k=k-1:if k=1 then k=21
80 goto40
90 data 29,11, 29,14, 27,17, 25,19, 23,20, 20,21, 17,21
100 data 14,20, 12,18, 10,15, 9,12,  9,9,   10,7,  12,4
110 data 14,2,  17,1,  20,1,  23,2,  25,3,  27,5,  29,8
```

Надеюсь, без объяснений понятно, как 50–й строке делают, чтобы дойдя до 21 переменная t снова стала =1, а в 70–й строке k, дойдя до 1 снова становится =21. В строке 60 использована «Ш», а не «*», чтобы в разные стороны двигались разные фигуры. Переменные x1,y1 и x2,y2 хранят координаты места для стирания.

> Довольно простым заданием будет внести в ту программу такие изменения, чтобы звёздочки двигались не по окружности, а по другой траектории. Для этого нужно изменить координаты положений звёздочки в строках 90–110. Если при этом точек станет больше, то изменится количество пар координат.
>
> Не забудьте внести изменения в строки 20,30,50,70, где это количество учитывается.

Массивы можно, конечно, использовать не только для координат, но и для хранения любых других данных.

Попробуем с помощью массивов составить личный телефонный справочник. В массиве FA$(9) будем держать фамилии 10–ти друзей. В массиве NO(9) будем держать их телефоны. Программа попросит ввести фамилию, разыщет её в массиве FA$ и по тому же индексу из массива NO возьмёт телефонный номер. Вы, конечно же, будете использовать не 10, а нужное вам число друзей и настоящие их фамилии с телефонами:

```
10 dim fa$(9),no(9)
20 for k=0 to 9:read fa$(k),no(k):next
30 input "Введите фамилию";f$
40 for k=0 to 9:if f$=fa$(k) then 60
50 next:print"Нет в справочнике!":goto 30
60 print"Телефон ";no(k):goto 30
70 data Петров,29045,Коваль,23433,Павлов,23321
80 data Сурков,21412,Моргунов,29981,Мышкин,24337
90 data Ауль,22221,Монахов,27725,Робов,21111,Пак,23324
```

В 20–й строке заполнили массивы из DATA. Обратите внимание, в DATA фамилии не стоят в кавычках. Это допускается с текстами только в случае, если текст состоит из одного слова без запятых или других знаков. В иных случаях текст в DATA должен заключаться в кавычки.

В 40–й строке в цикле просматриваем все фамилии из массива, и если найдём совпадающую с введённой фамилией,

то переходим на строку 50, где печатаем телефон из массива no с таким же индексом K, как и найденная фамилия. Если же фамилию в массиве не нашли, то после окончания цикла сработает `print` из 50–й строки.

```
Сделайте программу, которая спрашивает название школьного предмета и печатает имя-отчество преподавателя.
```

Сделаем программу «Электронный продавец», используя массивы. Это поможет упростить нахождение цены купленного продукта, уменьшит объем программы при большем количестве предлагаемых товаров:

```
10 color15,1:screen0:keyoff:width80:s=0
20 read kol:dim towar$(kol),cena(kol)
30 for k=1 to kol:read towar$(k),cena(k):next
40 '----- прочли данные ------ начинаем цикл -----
50 cls:print "М Е Н Ю:"
60 for k=1 to kol:locate 0,k+1:print k;towar$(k);
70 locate 30,k+1:print cena(k);:next
80 locate30,21:input"Введите выбранный номер";n
90 locate30,21:print"                          ";
100 locate30,21:input"Сколько штук берете";t
110 s=s+t*cena(n)
120 locate30,21:print"
130 locate30,21:input"Будем продолжать (да/нет)";yes$
140 if yes$="да" or yes$="ДА" or yes$="Да" then 50
150 print "Общая сумма";s;"руб. Заплатите в кассу!"
160 data 5
170 data "журнал <Мурзилка>",40,"аквариум",400
180 data "булочка с изюмом",45,"джинсы",17000
190 data "шоколад <milki way>",250
```

В работе программиста с массивами часто необходимо найти максимальный (минимальный) элемент массива, упорядочить (отсортировать) массив по возрастанию (убыванию), по алфавиту (для текстового массива). Поучимся это делать:

Найдём максимальный элемент массива. (Заполнить массив из `DATA` или случайными числами я предоставляю самому читателю. Для этого отводятся строки 20–40)

Чтобы найти максимальный элемент, мы заведём переменную MA, куда вначале поместим 0–й элемент из массива. Просмотрим по очереди все остальные элементы, и если встретится больший MA, запомним в MA уже его. Таким образом, просмотрев весь массив, мы будем иметь в MA максимальный элемент:

```
10 dim a(19)
…
50 ma=a(0)
60 for k=1 to 19:if a(k)>ma then ma=a(k)
70 next:print"Максимум=";ma
```

Иногда нужно найти не только максимальное число в массиве, но и узнать его номер.(Прежде, чем смотреть решение, попытайтесь это сделать сами.)

Для этого заведём переменную N, куда вначале занесём номер 0–го элемента, а при обновлении MA будем запоминать новый номер (k):

```
10 dim a(19)
…
50 ma=a(0):n=0
60 for k=1 to 19:if a(k)>ma then ma=a(k):n=k
70 next:print"Максимум=";ma;"по номеру";n
```

Ссылки

http://mirrors.pdp-11.ru/_msx/books/basic_begin/

Исходный текст в UTF-8

Архив с исходными файлами

http://sysadminmosaic.ru/msx/basic_for_beginners/basic_for_beginners

2022-05-09 00:46