Глава I. Основные объекты MSX BASIC

I.1. Алфавит

> Величайшее литературное произведение —
> в принципе не что иное, как разбросанный
> в беспорядке алфавит.
>
> *—Ж. Кокто*

> Multum in parvo.
>
> *—Латинское изречение*

Основой любого языка программирования является алфавит — набор допустимых литер, которые можно использовать для записи программ.

Литера — это печатный знак, или, для краткости, — знак. Литерами могут быть буквы, цифры и специальные знаки.

Алфавит MSX BASIC образуют:

- строчные и прописные буквы русского и латинского алфавита; имейте в виду тот факт, что буквы *латинского* алфавита A, B, C, E, H, K, M, O, P, T, X, Y не то же самое, что и графически похожие символы — *русские* буквы А, В, С , Е, Н, К, М, О, Р, Т, Х, У; для компьютера — это *различные* символы;
- арабские цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (нуль перечёркивается: Ø);
- дополнительные символы:

символ	действие
␣	пробел
=	равенство или присваивание
+	плюс или сцепление
-	минус
*	умножение (звездочка)
/	деление
^	возведение в степень
(	левая круглая скобка
)	правая круглая скобка
%	процент
#	номер
$	знак «доллар»
!	восклицательный знак
[	левая квадратная скобка
]	правая квадратная скобка
,	запятая
.	десятичная точка
"	кавычки
'	апостроф (для комментария)
;	точка с запятой

символ	действие
&	амперсенд (коммерческое «и»)
:	двоеточие
?	вопросительный знак
<	меньше
>	больше
\	обратная косая черта
@	цена (коммерческое «at»)
_	подчёркивание

По–видимому, символ @ изобрели коммерсанты, которым настолько некогда, что нет времени дописать как следует палочку буквы t.

Отметим, что пробел — такой же символ алфавита, как и все другие, хотя в позиции, занятой им, на экране дисплея и в распечатках никакой графический символ не изображается (просто она остаётся пустой!);

- служебные слова. Назначение служебных слов будет объяснено в дальнейшем по мере их введения.
  😎 В тексте книги служебные слова выделены моноширинным шрифтом.

I.2. Константы. Одинарная и двойная точность

> Представление данных — это сущность программирования.
>
> *—Ф. Брукс*

Величина в программировании — это объект, с которым связывается определённое множество значений.

Константы — это величины, значения которых не изменяются во время выполнения программы.

Константы делятся на два класса:

- *строковые* (или *символьные*) (вспомните литерные константы в школьном алгоритмическом языке!);
- *числовые*.

Строковая константа — это последовательность символов алфавита MSX BASIC, заключённая в кавычки. Отметим, что только в строковых константах наряду с латинскими буквами, цифрами и дополнительными символами допускается использование букв русского алфавита (прописных и строчных).

Например:

1. "MSX BASIC" ;
2. "Я M A X A" ;
3. "αβ7?* !&" .

Информация, которая заключена внутри кавычек, называется *значением* строковой константы, а количество символов внутри кавычек — её *длиной*.

Так в примере 1 значение строковой константы — MSX BASIC, её длина — 9, а в примере 2 значение строковой константы — Я M A X A, её длина — 9 (учтите наличие пробелов, а пробел " " — это символ алфавита!).

Заметим, что длина строковой константы должна быть не более 255 символов!

Числовые константы бывают шести типов:

1. *целые десятичные*; они находятся в диапазоне от -32768 до +32767.

Почему выбран именно такой диапазон? Дело в том, что целые числа кодируются последовательностями из 16 нулей и единиц. Поэтому можно закодировать лишь 65536 = 2^16 (напомним, что символ «^» — символ возведения в степень) различных целых чисел. Способ кодирования выбран так, что он охватывает целые числа от -32768 до +32767 (разумеется, включая нуль!).

Целые десятичные константы не должны содержать десятичную точку;

2. *десятичные* с *фиксированной* точкой; это положительные или отрицательные вещественные числа, содержащие десятичную *точку*. Преимущество десятичной точки может оценить каждый, кому случалось писать перечень десятичных дробей, разделённых запятыми!

   Например:

   - 1.75
   - 0.00007 (допустима также форма записи .00007)
   - -556.1
   - +4.007

3. *десятичные* с *плавающей* точкой; десятичная константа с плавающей точкой состоит из целого десятичного числа или десятичного числа с фиксированной точкой, возможно со знаком (мантисса), буквы E («Exponent» — «показатель») или D («Double» — «удвоенный») и целого числа, состоящего не более, чем из двух десятичных цифр со знаком (порядок). Порядок указывает, на сколько разрядов вправо (при положительном знаке порядка) или влево (при отрицательном знаке порядка) нужно сместить десятичную точку в мантиссе, чтобы получить значение представляемой константы в виде целой десятичной константы или десятичной константы с фиксированной точкой.

   Например:

   - 235.9888E-7 = .000023598888
   - 235D+6 = 235000000

   Допустимый интервал констант с плавающей точкой — от 10^(-64) до 10^63.

   Замечание. Строго говоря, понятие *мантиссы*, введённое выше, отличается от соответствующего понятия в вычислительной математике. Напомним, что в вычислительной математике запись десятичного числа в виде $A \times 10^p$, где p — целое число, называется записью десятичного числа в форме с *плавающей запятой*. Если $|A|<1$, то A называется *мантиссой*, а p — *порядком* числа. При $0.1 \leqslant |A|<1$ запись этого числа называется *нормализованной*, при $1 \leqslant A \leqslant 10$ запись называется *стандартной*;

4. *шестнадцатеричные*; шестнадцатеричные константы (по основанию 16) обозначаются *префиксом* (приставкой) &H (напомним, что знак «&» называется «амперсенд»). Для их записи используют цифры от 0 до 9 и латинские буквы A, B, C, D, E, F, обозначающие шестнадцатеричные цифры, соответствующие десятичным цифрам от 10 до 15. Например, &H76 , &h32F .

   Эти константы могут быть только *целыми*, изменяющимися в диапазоне от &H0000 до &HFFFF, причём константы в диапазоне от &H0000 до &H7FFF положительные (от 0 до 32767), а константы в диапазоне от &H8000 до &HFFFF отрицательные (от -32768 до -1);

5. *восьмеричные*; восьмеричные константы обозначаются префиксом &O (O — латинская буква). Они не должны содержать цифр 8 и 9.

   Например:

   - &O000000 (&O0)
   - &O347
   - &o17777

   Восьмеричные константы могут быть только *целыми*, изменяющимися в диапазоне от &O0 до &O177777, причём константы в диапазоне от &O0 до &O077777 — положительные (от 0 до 32767), а константы в диапазоне от &O100000 до &O177777 — отрицательные (от -32768 до -1);

6. *двоичные*; двоичные константы обозначаются префиксом &B; они содержат в записи только цифры 0 и 1 и могут быть только *целыми*, изменяющимися в диапазоне от &B0 до &B1111111111111111 (16 цифр), причём константы в диапазоне от &B0 до &B0111111111111111 — положительные (от 0 до 32767), а константы в диапазоне от &B1000000000000000 до &B1111111111111111 — отрицательные (от -32768 до -1).

   Приведём примеры записи двоичных чисел:

- &B01110110
- &b11100111

Так как целое число кодируется последовательностью из 16 нулей и единиц, то следует иметь в виду, что старший бит (шестнадцатый — самый левый) является знаковым битом (бит — от «binary digit» (двоичная цифра) — это цифра 0 или 1). Любое двоичное число с «1» в старшем бите считается отрицательным и представленным в дополнительном коде, поэтому шестнадцатый разряд называется *знаковым*.

Для представления отрицательного двоичного числа в дополнительном коде необходимо выполнить следующие преобразования: записать абсолютную величину числа, инвертировать каждый бит полученного числа (т.е. заменить все 1 на 0, а все 0 на 1), затем добавить 1 к самому младшему разряду (крайнему правому) (игнорируя возможный перенос 1 из старшего бита). Однако, вам эти преобразования самим делать не нужно, так как все подпрограммы ввода–вывода выполняют это автоматически. Такая кодировка позволяет экономить аппаратуру компьютера, так как при ней положительные и отрицательные числа обрабатываются абсолютно одинаково.

Например, покажем, что двоичное число -1 в дополнительном коде есть &B1111111111111111 .

Подробно распишем представление числа -1

0000000000000001	абсолютная величина числа
1111111111111110	вид числа после инверсии
1111111111111110 + 0000000000000001	добавляем 1 к самому младшему разряду (крайнему правому) (игнорируя возможный перенос 1 из старшего бита)
1111111111111111	получаем результат

При сложении, вычитании и умножении целых двоичных чисел не учитывается возможность переполнения; в случае его возникновения используются младшие 16 разрядов результата. Такая арифметика называется арифметикой по модулю 65536 (2 в степени 16); её основное достоинство состоит в том, что она даёт одинаковые в двоичном представлении результаты независимо от того, как понимаются операнды: как числа со знаком в диапазоне от -32768 до +32767 или как числа без знака в диапазоне от 0 до 65535. Однако, при выполнении деления возможно возникновение ошибочной ситуации, если делитель — нуль или частное не умещается в 16 разрядов (переполнение, возникающее, например, при делении целого числа -32768 на целое число -1).

Легко видеть, что чем меньше основание системы счисления, тем длиннее записи чисел в этой системе!

В свою очередь, десятичные числовые константы с фиксированной и плавающей точкой могут быть классифицированы на *два типа*: числовые константы *одинарной точности*, содержащие не более 6 значащих цифр, и числовые константы *двойной точности*, содержащие не более 14 значащих цифр.

По умолчании числовая константа имеет *двойную* точность !

Термином «умолчание» обозначается «сама собой разумеющаяся информация». В общении людей её очень много, например, вместо «Кто занял своё место в этой очереди позже всех ?» говорят просто: «Кто последний ?» Для компьютера же все операции должны быть определены не только однозначно, но и полностью.Это неудобство отчасти смягчается возможностью действовать по умолчании, то есть пользоваться тем, что иногда компьютер из нескольких вариантов действия способен сам выбрать один, предпочитаемый. Предпочтения закладываются в компьютер так, чтобы в большинстве случаев программисту было удобно пользоваться ими, а не оговаривать какой–то другой вариант.

Десятичная константа одинарной точности — это десятичная числовая константа, состоящая не более, чем из 6 значащих цифр и записанная в соответствии с одним из следующих правил:

1. в виде десятичной константы с плавающей точкой с использованием в своей записи латинской буквы E;
2. в виде десятичной константы с фиксированной точкой, за которой следует символ «!» (восклицательный знак);

Примеры:

- 6.23E3
- -1.09E-6
- 22.5!

- .4!

Десятичная константа двойной точности — это десятичная числовая константа, состоящая не более, чем из 14 значащих цифр, записанная в одной из следующих форм:

1. в виде десятичной константы с фиксированной точкой или без неё (принцип умолчания!);
2. в виде десятичной константы с фиксированной точкой, за которой следует символ «#»;.
3. в виде десятичной константы с плавающей точкой с использованием в записи буквы D вместо буквы E.

Примеры:

- 56
- -409345153489
- .345#
- -1.09433D-06

I.3. Переменные

> ...это, наверное, полезно тем, кто даёт им названия.
> Иначе зачем бы их вообще давали?
>
> *—Л.Кэролл. Алиса в Зазеркалье*

Возможности вычислительной машины были бы слишком ограничены, если бы при работе с ней можно было использовать только постоянные величины (константы). Поэтому в любом языке программирования есть средства, позволяющие выделить часть памяти компьютера для хранения изменяющихся значений. Каждое такое значение идентифицируется (определяется) уникальным *именем*, которое в совокупности со значением определённого *типа* называется *переменной*.

Таким образом, переменная связана с *тремя* объектами:

- *именем*, определяющим место в памяти,
- *значением* (хранимым объектом) и
- *типом*.

Поэтому, если мы говорим: «X имеет значение 3.45», то на самом деле имеется в виду, что «X — это имя места памяти компьютера, где в данный момент хранится число 3.45».

Подчеркнём, что в математической литературе с понятием *переменной* связывают некоторый именованный объект, который может принимать одно из допустимых значений.

Для программиста термин «переменная» ассоциируется с участком оперативной памяти, в котором хранится текущее значение объекта. Поэтому имя переменной можно рассматривать как синоним адреса такого участка. Знание этого адреса даёт пользователю возможность изменять значение переменной в процессе выполнения программы.

С переменными мы сталкиваемся не только в информатике, но и в обиходе. Хозяйка, у которой на кухне хранятся по сосудам и сусекам, баночкам и коробочкам крупы и чай, варенье и масло, постоянно оперирует переменными. Надпись «чай» на жестяной коробочке — это имя переменной; её содержимое — значение переменной. Когда коробочка пуста, то в наших терминах значение переменной «чай» равно нулю.

Переменную можно представить себе как своеобразную коробочку, на которую наклеен ярлык — *имя переменной*. Когда мы хотим взять значение переменной, мы смотрим на содержимое коробочки. Меняя значение, мы кладём в коробочку что-то новое.

Теперь, забегая вперёд, можно показать, как разрешается неоднозначность при использовании имён в операторах присваивания вида:

```
I = I + 1
```

В левой части оператора символ I обозначает *имя* (поскольку оно определяет место для размещения значения), в то время как в правой части оператора символ I обозначает *значение* (поскольку оно требуется для вычислений).

При составлении программы программист даёт имена используемым объектам. Имена выбираются им по своему усмотрению. Рекомендуется выбирать имена таким образом, чтобы они отражали смысл объекта и его роль в методе решения задачи (говорят, что «имена должны быть наглядными!»).

Имя (идентификатор) переменной есть последовательность, состоящая из не более чем из 252 латинских букв и арабских цифр и начинающаяся с *буквы*. Однако значащими являются лишь первые *два* символа имени.

Например, для программиста имена переменных MIL1 и MI различны, а для компьютера они одинаковы!

Пробелы в имени игнорируются (PI ≡ P I).

Вопрос читателю: сколько различных идентификаторов можно записать?

Заметим, что служебные слова MSX BASIC, кроме слова ELSE, (например: END, PRINT, IF, THEN, MAX и т.д.) не могут служить именем или частью имени переменной!

Например, недопустимы имена SHIFR (содержит IF), TON (содержит TO), XOR (содержит и OR, и XOR).

Значение переменной может быть явно задано программистом или получено в результате вычислений в программе. Когда переменная используется впервые, то для её хранения выделяется область памяти компьютера, первоначально заполненная *нулями*. Объем этой области памяти зависит от *типа* переменной (или от типа по умолчании, если он явно не задан).

Тип переменной определяется типом принимаемых ею значений и может задаваться *определённым* символом (*указателем типа*), добавляемым к имени переменной, а именно:

- % — для *целой* переменной (переменной *целого* типа), например: limit%, A1% ;
- ! — для переменной *одинарной* точности, например: A!, mum! (но не maximum!);
- # или никакой символ не добавляется (по умолчании!) для переменной *двойной* точности, например: MIN, p1#, ABC (но не ABS !);
- $ — для *строковой* (*символьной*) переменной, например: N$, ST$, KAPPA$.

По умолчании числовая переменная имеет тип *двойная* точность!

Отметим, что переменные, имеющие одинаковое имя, но различные указатели типа, различаются!

Например, A#, A!, A$, A% — разные переменные, однако, A и A# — два имени одной и той же переменной.

В следующей важной таблице указано количество байт памяти ЭВМ YAMAHA (1 байт = 8 битам), занимаемых переменной в зависимости от типа.

Тип переменной	Количество байт
Целая	5
Одинарной точности	7
Двойной точности	11
Строковая	Длина области, занятой строкой + 6

Например, для *целой* числовой переменной компьютер запоминает «паспорт», указывающий количество байт, необходимых для хранения переменной («паспорт» имеет имя VALTYPE, «VALue TYPE» — «тип значения»), имя переменной и само её значение.

VALTYPE занимает один байт, имя переменной занимает два байта, само целое число — тоже два байта. Таким

образом, для хранения целой переменной требуется 5 байт.

Один *байт* памяти ЭВМ способен хранить только один символ, поэтому можно рассматривать байты как информационные символы, хотя в памяти компьютера помимо них хранится и другая информация, например, константы или команды. Более крупная единица памяти компьютера называется *килобайтом* (1 Кбайт), причём, 1 Кбайт = 1024 байт. Отметим, что в 8 Кбайт можно записать примерно *четыре* страницы машинописного текста. Далее,

- 1024 Кбайта равны 1 мегабайту (1 Мбайт) (от греч. «megas» — «большой»),
- 1024 Мбайта равны 1 гигабайту (1 Гбайт) (от греч. «gigas» — «гигантский»),
- 1024 Гбайта равны 1 терабайту (1 Тбайт) (от греч. «teras» — «чудовище»).

Заметим, что слово «byte» («байт») произошло от слова «bite» («кусок»), в котором, чтобы не путать при чтении, букву «i» заменили на букву «y»; произношение обоих слов осталось одинаковым.

Дополнительная информация: Приложение Д. Системы представления чисел. (добавлено в текст в 2022-05-09)

I.4. Понятие оператора. Оператор DEF

Оператор — это конструкция языка программирования, которая предписывает выполнять определённые действия над заданными объектами, либо устанавливает последовательность, в которой должны выполняться другие действия, либо описывает характеристики объектов программы.

Почти все операторы начинаются специфическим служебным словом; кроме того, ряд служебных слов может использоваться и в самой конструкции оператора. Каждое служебное слово в языке программирования имеет строго определённый смысл, установленный при разработке данного языка.

Смысл служебных слов будет поясняться далее по мере использования их в операторах языка. Поскольку язык программирования BASIC был разработан в США (математиками Дж.Кемени и Т.Курцем, Дартмутский колледж, 1964 г.), то естественно, что он максимально приближен к английскому языку. Поэтому вам необходимо запомнить *правильное* произношение и точный перевод каждого служебного слова. Отметим, что BASIC, как и любой другой язык программирования, не допускает вольного обращения с синтаксисом и пунктуацией. Каждый оператор языка должен быть записан по строго определённым синтаксическим и семантическим правилам. Применительно к языку программирования *синтаксис* — это совокупность правил, которым должна удовлетворять запись операторов программы, а *семантика* — это значение, смысл отдельных операторов или совокупности нескольких операторов (*блоков* операторов); семантика определяет, какие операции и в какой последовательности должен выполнять компьютер, реализующий программу.

Для начинающих программистов рекомендуется для удобства чтения программы всегда ставить пробелы до и после служебного слова, но *внутри* служебного слова пробелы *запрещены*!

Сейчас мы познакомимся с первым *оператором* языка программирования MSX BASIC.

Ещё раз напомним, что если после имени переменной не следует один из специальных символов, определяющих её тип (%, #, $, !), то по принципу умолчания считается, что имя принадлежит числовой переменной двойной точности.

Это соглашение нарушается в случае использования четырёх модификаций оператора `DEF` («to DEFine» — «определять»), пользуясь которыми можно задавать тип переменной одной *начальной* буквой её имени.

Можно также определить интервал или несколько интервалов букв для именования переменных (интервальные пары букв также отделяются запятыми). Интервал записывается с помощью двух букв, разделённых знаком «-» (минус).

Например:

```
DEFINT A,J-M
```

В программе, содержащей указанный оператор, любая переменная, имя которой начинается с A, J, K, L, M, являются переменной *целого типа* (независимо от того, оканчивается ли её имя знаком %). Так, переменные с именами AARD, J2, K, LOOP будут определяться как целые переменные.

Итак, магическое служебное слово, используемое в операторе DEF для описания типа *целой* переменной — это INT («INTeger» — «целое число»).

Аналогично с помощью операторов:

- DEFSTR («STRing» — «строка»),
- DEFSNG («SiNGle» — «одиночный»),
- DEFDBL («DouBLe» — «двойной»)

можно задавать начальные буквы имён *строковых* переменных и переменных *одинарной* и *двойной* точности соответственно. Однако, действие оператора DEF в любой его модификации отменяется, если тип переменной указан в явном виде (символами %, #, $, ! после имени переменной), то есть, говоря другими словами, действие оператора DEF не распространяется на переменные, после имени которых следует один из символов:

```
% , # , $ , !
```

Таким образом, при определении типа переменной происходит следующее:

1. если за именем переменной следует признак типа, то тип переменной определяется в соответствии с этим признаком. Если признак отсутствует, проверяется пункт 2;
2. если в программе имеется какой–либо из операторов DEFINT, DEFSNG, DEFDBL или DEFSTR и его действие затрагивает имя переменной, то её тип определяется в соответствии с правилами, заданными этим оператором;
3. если тип переменной остался не определённым этими двумя проверками, то переменная рассматривается как переменная двойной точности.

Выполнение оператора DEF должно предшествовать первому использованию определяемых им переменных. Кроме того, этот оператор присваивает начальные значения всем переменным, имена которых начинаются с указанных в нем букв, а именно: *числовым* переменным присваивается значение *нуль*, а *строковым* — значение строки, имеющей длину 0 (такой строкой является пустая строка "").

Во многих программах используются только целые и символьные переменные. В начале таких программ обычно ставится оператор DEFINT A-Z, а символьные переменные указываются индивидуально признаком $. Задание всех числовых переменных как целых значительно сокращает время вычислений, а также уменьшает размер памяти, необходимый для хранения значений этих переменных.

I.5. Массивы переменных. Оператор ERASE

Фома идёт в реку. Фома не труслив,
Хоть там аллигаторов целый *массив*.

—Почти по С.Михалкову

Для возможности *однотипной* обработки большой совокупности данных в программировании введено понятие *массива*.

Массив — это конечное множество значений (числовых или строковых), обозначенных одним именем (*именем массива*), причём каждый элемент этого множества идентифицируется с помощью одного или нескольких числовых индексов.

Правила образования имени массива то же, что и для имени простой переменной.

Индексы — это своего рода числовые координаты, указывающие местонахождение конкретного элемента как в массиве данных, так и в массиве ячеек памяти, отведённых для хранения этих данных.

Массив характеризуется *размерностью*, то есть количеством индексов, необходимых для поиска элемента в массиве, а также *границами измерения*, то есть границами изменения каждого индекса.

Массив описывается (объявляется) с помощью *описателя* размерности. Описатель записывается следующим образом:

сразу после имени массива в круглых или квадратных скобках указывается верхняя граница изменения индекса. Если индексов несколько, то верхние границы изменения каждого индекса разделяются запятой.

Нижние границы индексов всегда равны 0 (*нулю*).

При обращении к элементу массива указывается *имя элемента массива* (имя индексируемой переменной). Имя элемента массива состоит из имени массива и следующего(их) за ним в круглых или квадратных скобках индекса(ов) (если индексов два, то они разделяются запятой).

В качестве индекса может использоваться *выражение* (разумеется не строковое!), частными случаями которого являются *константа* или *имя переменной* (понятие *выражения* см. в разделе I.8).

Старайтесь в качестве индексов использовать *целочисленные* выражения (выражения, которые могут принимать только целочисленные значения), так как в противном случае компьютер автоматически преобразует вещественные константы, представляющие значения индексов, в целые константы, на что расходуется машинное время.

Например, целесообразно писать A(K%), а не A(K).

Примеры:

- 1)

C%(100)	
имя массива	C
тип элементов массива	целочисленный
размерность	одномерный
количество элементов	100 + 1 = 101
имена элементов массива	C%(0), C%(1),... ,C%(100)

- 2)

A(5,10)	
имя массива	A
тип элементов массива	двойная точность
размерность	двумерный
количество элементов	(5+1)*(10+1) = 66
имена элементов массива	A(0,0), A(0,1), ..., A(0,10), A(1,0), A(1,1), ..., A(1,10), ... A(5,0), A(5,1), ..., A(5,10)

- 3)

E$[20]	
имя массива	E
тип элементов массива	строковый
размерность	одномерный
количество элементов	20 + 1 = 21
имена элементов массива	E$[0],E$[1], ..., E$[20]

Максимальная размерность вещественного массива — 11, целого или строкового — 13, максимальное число элементов массива определяется размером оперативной памяти компьютера и типом массива.

Перед использованием в Вашей программе массивов необходимо выделить для них место в памяти

(«зарезервировать место»). Это осуществляется при помощи специального оператора *объявления* (описания) *массивов*.

Его структура:

```
DIM α, σ, β,...
```
,

где

- `DIM` («DIMension» — «размерность») — служебное слово;
- α, σ, β, ... — список описателей размерности встречаемых в программе массивов; описатели отделяются друг от друга запятой.

Например, `DIM C%(100),A(5,10),E$(20)`, здесь `C%(100),A(5,10),E$(20)` — описатели размерности массивов.

Запомните: в любом месте программы, где имя массива встречается *в первый* раз (конечно, не в операторе `DIM`), используется *неявный* оператор `DIM` с верхней границей индекса(ов), равной 10. Таким образом, например, одномерный массив, содержащий *менее* 11 элементов, можно не описывать оператором `DIM` !

Оператор `DIM` может находиться в любом месте программы, но обязательно *перед* работой с массивом, к которому он относится.

При выполнении оператора `DIM` всем элементам объявляемых числовых массивов присваивается значение 0 («нуль»), а всем элементам объявляемых строковых массивов присваивается значение "" («нуль–строка» или «пустая» строка).

Если для размещения всех массивов программы объем памяти оказывается недостаточным, на экран дисплея выдаётся сообщение:

«Out of memory»
(*«Не хватает памяти»*)

или

«Out of string memory»
(*«Не хватает строковой памяти»*)

для случая массива символьных строк, и оператор `DIM` резервирует место только для первых перечисленных в нем массивов, которые полностью уместились в памяти.

Кроме очевидного использования для создания массивов требуемого типа и размера, имеет смысл применять оператор `DIM` для объявления всех переменных. Например, `DIM MIN%, AB1#, X, C[15,4], X(6), AB1#(4,8)`.

Обратите внимание на то, что, хотя простая переменная X и массив X(6) имеют одинаковые имена, для компьютера же — это разные объекты!

Если в программе уже описаны массивы, содержащие большое количество элементов, то для создания новой простой переменной потребуется несколько секунд машинного времени (оно уходит на поиск места для новой переменной в оперативной памяти), определение же всех простых переменных оператором `DIM` в начале программы позволяет сэкономить это время (см. пример 8 в разделе 7.5).

Приведём очень важную таблицу, указывающую количество байт, занимаемых элементом массива, в зависимости от типа массива.

Тип массива	**Количество байт на один элемент**	**Количество элементов**	
		MSX 1	**MSX 2**
Целый	2	14343	14323
Одинарной точности	4	7171	7161
Двойной точности	8	3585	3580

Тип массива	Количество байт на один элемент	Количество элементов	
		MSX 1	MSX 2
Строковый	Длина области, занятой строкой+3	9562	9548

Определив размерность некоторого массива посредством оператора DIM, её уже нельзя изменить с помощью некоторого другого оператора DIM. Для этого необходимо сначала удалить соответствующий массив из оперативной памяти компьютера с использованием «стирающего» оператор ERASE.

Оператор состоит из служебного слова ERASE («to erase» — «вычёркивать») и следующего за ним списка имён уничтожаемых массивов. Имена массивов в этом списке перечисляются через запятую, причём скобки и максимальное значение индексов массивов не указываются.

Например, если массив A в программе описан как DIM A(30), то оператор ERASE A «уничтожает» одномерный массив A(30); после этого мы можем оператором DIM A(4,10) определить двумерный массив A(4,10) с тем же именем!

Заметим, что если попытаться объявить заново с помощью оператора DIM массив без предварительного использования оператора ERASE, то выдаётся сообщение об ошибке

«Redimensioned array»
(*«Переопределение массива»*).

Следующий раздел при первом прочтении можно пропустить!
Однако *обязательно вернитесь* к нему позднее!

I.6. Имена, значения и типы

Часто оказывается труднее хранить
богатства, чем добывать их.

—Демосфен

Ясно, что цель программы состоит в вычислении *значений*. В свою очередь компьютер оперирует не со значениями, а скорее с *представлениями* значений, которые являются конфигурациями байт памяти ЭВМ. Так как физические представления зависят от изображаемых объектов, то для того, чтобы оперировать со значениями, необходимо специфицировать их *типы*. Это кажущееся ограничение может оказаться на деле выгодным, потому что оно заставляет программиста точно определить все используемые им объекты и лучше контролировать свою программу.

Итак, *каждый* программный объект имеет *тип* и *значение*. Разумеется, он должен иметь также и *имя* в программе, чтобы мы могли отличить его от других объектов с аналогичными типом и значением.

1. *Константа* имеет фиксированное имя, фиксированный тип и фиксированное значение.

   Например, обозначение 134 есть *имя* константы типа «целое» и фиксированного *значения* 134 (сто тридцать четыре).

   Для констант тип и значение выводятся непосредственно из *имени*. Язык MSX BASIC, как и многие другие языки программирования позволяют программисту выбрать другое имя для обозначения константы. Например, программист может связать с константой 3.14159 имя PI при помощи оператора

   ```
   PI = 3.14159
   ```

   В таких случаях говорят, что PI есть *символическая константа* (т.е. такая, которая обозначается некоторым символом вместо принятого для константы обозначения объявлением её значения).

   Использование символических констант — хороший приём программирования: он исключает ситуации, при которых

значения, являющиеся по существу параметрами выполняемой программы, используют в явной форме многократно в разных местах программы. Такие ситуации осложняют модификацию и расширение программ. Значение символической константы появляется только в одном месте программы, в *объявлении символической* константы, которое позволяет связать значение константы с выбранным именем; если появляется необходимость перейти к другому значению, модифицируется только это объявление.(В некоторых языках программирования строго применяется этот принцип, *запрещая* использование констант, имеющих отличную от символической форму!).

2. *Переменная* имеет фиксированное имя, фиксированный тип и переменное значение.

   Имя переменной называется *идентификатором*. Тип переменной связывается с её именем при помощи *объявления типа*, которое может быть неявным.

   В заключение определим *переменную* в информатике как совокупность трёх элементов: идентификатора, типа и значения , где только третий элемент является переменным (однако, в некоторых языках программирования (например, Рапира) тип является переменным!).

   Заметим, что понятие переменной в информатике *отличается* от понятия переменной в математике!

3. *Массив* имеет фиксированное имя, фиксированный тип и конечное множество значений.

   Объявление массива уточняет его тип (который может задаваться неявно), число его измерений и границы каждого измерения.

   Отметим, что любое объявление массива определяет две категории имён: имя массива в целом и имена для обозначения каждого элемента массива(имена индексируемых переменных).

I.7. Операции

Операция в языке программирования — это совокупность действий, вырабатывающая некоторое значение, называемое *результатом* операции.

Исходными данными для операции являются её *операнды* (переменные, константы, функции), причём могут использоваться один или два операнда.Соответственно операции называются *одноместными* и *двухместными* (*унарными* и *бинарными*).

Операции в зависимости от типов операндов и результата делятся на следующие группы: арифметические, отношения, логические, строковые, операции–функции.

I.7.1. Арифметические операции

В таблице в порядке уменьшения приоритета перечислены 8 арифметических операций.

Операция	Действия	Примеры
^	Возведение в степень	X^Y
-	Изменение знака (унарный минус)	-U
*, /	Умножение, деление	X*Y, X/Y
\	Деление нацело	X\Y
MOD	Нахождение остатка	XMODY
+, -	Сложение, вычитание (бинарный минус)	X+Y, X-Y

Следовательно, при отсутствии скобок вначале выполняется операция «^», затем «-» (унарный минус) и т.д. При прочих равных условиях операции одинакового приоритета выполняются слева направо.

Существует фраза: «my dear aunt Sally», которая помогает запомнить, что для получения правильных результатов надо сначала умножать(«multiply») и делить(«divide»), а затем складывать(«add») и вычитать(«subtract»).

Отметим, что операнд X операции «^» должен быть неотрицательным (X⩾0) (однако, если операнд Y принимает

целые значения, то операнд X может быть и отрицательным!).

Поэтому, например, выражение $\sqrt[n]{X}$ для целого нечётного n следует представлять в виде `SGN(X)*ABS(X)^(1/n)` (X≠0).

Теперь рассмотрим подробно операции:

- *деление нацело* ;
  перед выполнением деления операнды преобразуются к целому типу путём отбрасывания дробных частей(операнды должны находиться в интервале от -32768 до +32767), полученное после деления частное преобразуется к целому типу путём отбрасывания дробной части.

  Примеры:

  - 10\4 возвращает 2 ,
  - 25.68\6.99 возвращает 4 ;

- *нахождение остатка* (вычисление остатка);
  перед выполнением операции `XMODY` операнды X и Y преобразуются к целому типу путём отбрасывания дробных частей; результатом операции является целое число, равное остатку от деления полученных целых чисел (остаток имеет знак делимого).

  Примеры:

  - 10.4 `MOD` 4 возвращает 2 (10/4=2 и остаток 2),
  - 25.68 `MOD` 6.99 возвращает 1 (25/6=4 и остаток 1),
  - 33567 `MOD` 2 возвращает сообщение «Overflow» («Переполнение»)(подумайте, почему?),
  - -7 `MOD` 8 возвращает -7.

Заметим, что если при вычислениях встретилось деление на нуль, то фиксируется ошибка

«Division by zero» (*«Деление на нуль»*),

и выполнение программы прекращается.

При переполнении фиксируется ошибка

«Overflow» (*«Переполнение»*),

и выполнение программы также прекращается. Переполнение возникает оттого, что в ячейку памяти компьютера программист пытается поместить «очень длинное» число. Слишком большая (по модулю) константа может не уместиться в отведённую для неё ячейку!

При попытке работать с числом, большим, чем максимальное, компьютер выведет на экран сообщение об ошибке «Overflow» (переполнение диапазона представимых в компьютере чисел).

Попытка представления чисел, меньших, чем минимальное, не вызывает сообщения об ошибке, но само число будет округлено до нуля, что подразумевает невозможность его использования в качестве делителя.

I.7.2. Операции отношения. Логические операции

Tertium non datur.

—Лат. изречение

В операциях отношения используются следующие символы (знаки операций отношения):

операция	проверяемое условие	примеры
=	равенство	X=Y
< > или > <	неравенство	X<>Y, X><Y
<	меньше	X<Y
>	больше	X>Y
< = или = <	меньше или равно	X<=Y, X=<Y
> = или = >	больше или равно	X>=Y, X=>Y

Операции отношения обладают равным приоритетом.

Они применяются для сравнения двух величин, например:

- PT<=23 ,
- K%(7,J)=-4 ,
- A$>"КГПИ" .

Результатом операции отношения для *программиста* может быть одно из двух логических значений:

- T (первая буква английского слова «true» — «истина»), если соответствующее отношение истинно, и
- F (первая буква английского слова «false» — «ложь»), если соответствующее отношение ложно.

Результатом операции отношения для *ЭВМ* является:

- `&B1111111111111111` = -1 , если соответствующее отношение *истинно*;
- `&B0000000000000000` = 0 , если соответствующее отношение *ложно*.

Результат операции отношения может быть использован для принятия решения о дальнейшем ходе выполнения программы (см. раздел III.2.).

По приоритету операции отношения следуют за арифметическими операциями и операцией конкатенации (см.далее).

К логическим операциям относятся:

- `NOT` — логическое *отрицание* (инверсия),
- `AND` — логическое *умножение* (конъюнкция),
- `OR` — логическое *сложение* (дизъюнкция),
- `XOR` — («eXclusive OR») исключающее «*или*» (симметрическая разность),
- `EQV` — («EQuiValence или exclusive or negation») *эквивалентность*,
- `IMP` — («IMPlication») *импликация* (следование).

Логические операции выполняются над 16–*битными целыми* операндами со знаком, которые принадлежат отрезку [-32768, 32767], в противном случае фиксируется ошибка: «Overflow» («*Переполнение*»).

Логические операции выполняются над *каждой парой соответствующих битов* операндов, т.е. каждый бит результата определяется значениями соответствующих битов в операндах.

Операции над соответствующими битами выполняются по правилам, приведённым в следующей таблице истинности (диаграмма Вейча).

X	Y	NOT X	X AND Y	X OR Y	X XOR Y	X EQV Y	X IMP Y
1	1	0	1	1	0	1	1
1	0	0	0	1	1	0	0
0	1	1	0	1	1	0	1
0	0	1	0	0	0	1	1

Операции в таблице перечислены в порядке убывания приоритетов.

Следующие примеры демонстрируют выполнение логических операций. Символы $()_2$ указывают на то, что число в круглых скобках записано в двоичной системе счисления.

Примеры:

- 1)

```
15 AND 14 = 14

15 = (1111)₂  =  1   1   1   1
                AND AND AND AND  = (1110)₂ = 14
14 = (1110)₂  =  1   1   1   0
```

- 2)

```
63 AND 16 = 16

63 = (11111)₂  =  1   1   1   1   1   1
                  AND AND AND AND AND AND  = (10000)₂ =16
16 = (10000)₂  =  0   1   0   0   0   0
```

- 3)

```
(-1) AND 8 = 8 , т.к. -1=(1111111111111111)₂
                       8=(1000)₂
```

- 4)

```
4 OR 2 = 6 , т.к. 4=(100)₂, а 2=(10)₂
```

- 5)

```
(-1) OR (-2) = (-1) , т.к. -1=(1111111111111111)₂, а
-2=-(10)₂ =(1111111111111101 + 0000000000000001)₂ = (1111111111111110)₂
```

- 6)

```
NOT(-1) = 0;
NOT(2) = -3
```

- 7)

```
&H0F AND &H08 = &H08
```

При выполнении операции AND над аргументами &H0F и &H08 из &H0F выделяется четвёртый бит, начиная с младшего бита, а все остальные биты маскируются (зануляются). Подобная операция, при выполнении которой для какого либо байта производится маскирование одного или нескольких битов, называется *маскированием* битов. Оно позволяет указать компьютеру, какие органы управления должны быть включены (состояние 1), а какие — выключены (состояние 0).

Если значения операндов принадлежат числовому множеству, состоящему из двух элементов: 0 и (-1), то в результате логической операции будет получен *либо* 0, *либо* (-1). Таким образом, можно составить таблицу:

X	Y	NOT X	X AND Y	X OR Y	X XOR Y	X EQV Y	X IMP Y
-1	-1	0	-1	-1	0	-1	-1
-1	0	0	0	-1	-1	0	0
0	-1	-1	0	-1	-1	0	-1
0	0	-1	0	0	0	-1	-1

Результатом логической операции для *программиста* может быть одно из двух логических значений:

- T (первая буква английского слова true — «истина») или
- F (первая буква английского слова false — «ложь»).

Результатом логической операции для *компьютера* является либо:

- целое число, отличное от нуля (этот случай соответствует логическому значению T);
- нуль (этот случай соответствует логическому значению F).

По приоритету логические операции следуют за арифметическими операциями и операциями отношения. Так же, как

и операции отношения, логические операции могут быть использованы для принятия решения о дальнейшем ходе выполнения программы (см. раздел III.2.).

I.7.3. Строковые операции

1. Значения строковых переменных могут объединяться с помощью знака «+» — операция *конкатенации* или *сцепления* («concatenation» — «сцепление»); отметим, что иногда(!) в операторе `PRINT` знак «+» можно опускать.
   Пример: в результате выполнения операции конкатенации вида «NEW» + «-YORK» получим строковую константу «NEW-YORK».
2. Строковые переменные могут сравниваться с помощью тех же операций отношения, что и числовые: = , < , > , <> , <= , >= .

Строковую переменную можно сравнивать *только* со *строковыми* переменными !

Для компьютера значением символа является его *код* (цифровое представление символа), т.к. вычислительная машина работает только с числами. При вводе в компьютер символы преобразуются в *код* в соответствии с Американским стандартным 8–разрядным кодом для обмена информацией ASCII («American Standard Code for Information Interchange»).

Например, прописная латинская буква A имеет код 65, а строчная — 97.

Полная таблица символов приведена здесь.

Забегая вперёд, заметим, что код любого символа может быть получен при помощи функции преобразования `ASC("символ")`; так значением `ASC(" ")` является 32.

При выполнении операции отношения две строки сравниваются посимвольно до выявления двух первых несовпадающих символов. Затем сравниваются (как обычные целые числа) коды ASCII этих несовпадающих символов и определяется наибольший код; соответствующая строка считается большей. Если первая строка короче второй, и все её символы совпадают с соответствующими символами второй строки, то большей считается вторая,более длинная строка. Две строки одинаковой длины с совпадающими в одинаковых позициях символами считаются равными;в частности, равными считаются две «пустые» строки ("").

Отметим,что пробелы в начале и конце строки *значимы*.

Примеры:

- "AA"<>"AB" ,
- "ABC"="ABC" ,
- "SM">"S" ,
- "KQ">"KG" ,
- "1AAA"<"2aaa" ,
- " "<"1" (код символа "1" равен 49) .

Заметим, что результатом всех приведённых операций отношения в данных примерах является (-1) — для ЭВМ и T («истина») — для программиста.

Напомним, что компьютер различает строчные и прописные буквы в значениях строковых переменных по величине кода ASCII !

Ясно, что строковые операции могут быть эффективно использованы, например для *сортировки* строк (расположения их в алфавитном порядке (см.пример 5 в разделе IV.4.1.).

Однако, будьте очень осторожными, используя выражения с *несколькими* операциями отношения в случае строковых операндов. Например, запись в программе выражения типа

```
A$=B$=C$
```

вызовет сообщение об ошибке:

«Type mismatch»

(*«Несоответствие типов»*),

поскольку результатом выполнения первой операции отношения A$=B$ является целое число (0 или -1), которое нельзя сравнивать со значением строковой переменной C$!

Отметим, конструкция вида (A$=B$)=(C$=D$) *допустима*!

По приоритету операции отношения следуют за операцией конкатенации, а операция конкатенации — за арифметическими операциями.

I.7.4. Операции-функции

Многие функции, часто встречающиеся при программировании на языке MSX BASIC представлены в виде составной части языка и получили название *операции — функции* (или *встроенные функции*). Они обеспечивают выполнение различных стандартных операций обработки числовых и символьных данных.

В таблице приведены наиболее часто употребляемые *числовые* встроенные функции.

***Имя* функции**	***Значение* функции**
ABS(X)	| X |
LOG(X)	ln X , X>0
EXP(X)	exp(X) -147.366 < X < 145.063
SQR(X)	√X , X≠0
SIN(X)	sin X , X в радианах
COS(X)	cos X , X в радианах
TAN(X)	tg X , X в радианах
ATN(X)	arctg X , X в радианах arctg X ∈ [-π/2,π/2]
SGN(X)	1, если X>0, sign X={ 0, если X=0, -1, если X<0
INT(X)	[X] (целая часть X) Определение наибольшего целого числа, не превосходящего значения X.
FIX(X)	У значения X отбрасывается десятичная точка и все цифры,следующие за ней.
RND(X)	Генерируется псевдослучайное число из интервала (0,1).
CINT(X)	Значение X преобразуется к целому типу («Convert to INTeger type»)
CDBL(X)	Значение X преобразуется к типу «двойная точность» («Convert to DouBLe precision type»).
CSNG(X)	Значение X преобразуется к типу «одинарная точность» («Convert to SiNGle precision type»).

Перечисленные в таблице числовые функции составляют тот «джентльменский набор», которым наделена каждая ЭВМ, «обученная» языку программирования высокого уровня. Эти функции называются *встроенными*,поскольку алгоритмы вычисления их значений записаны, «встроены» в MSX BASIC.

В качестве аргумента X числовых встроенных функций может быть использовано *арифметическое выражение* (данное понятие рассмотрено в разделе I.8.).

Числовые функции, которые не входят в MSX BASIC, вычисляются следующим образом:

десятичный логарифм	LG(X)=LOG(X)/LOG(10)
секанс	SEC(X)=1/COS(X)
косеканс	CSC(X)=1/SIN(X)
котангенс	COT(X)=1/TAN(X)
арксинус	ARCSIN(X)=ATN(X/SQR(-X*X+1))

арккосинус	ARCCOS(X)=-ATN(X/SQR(-X*X+1))+2*ATN(1)
арккотангенс	ARCCOT(X)=ATN(X)+2*ATN(1)
гиперболический синус	SINH(X)=(EXP(X)-EXP(-X))/2
гиперболический косинус	COSH(X)=(EXP(X)+EXP(-X))/2
гиперболический тангенс	TANH(X)=(EXP(X)-EXP(-X))/(EXP(X)+EXP(-X))

Помните, что указанные формулы справедливы лишь при тех значениях аргумента, при которых все используемые выражения имеют смысл!

Заметим, что людольфово число π (так иногда называют число π в честь голландского математика Людольфа ван Цойтена (1540-1610)) можно получить следующим обращением к встроенной функции ATN(X): 4*ATN(1) = 3.1415926535898 (π с 14 знаками) или ATN(9.9999999999999E62)*2 = 3.1415926535898, а неперово число e: EXP(1) = 2.7182818284588.

Обратите внимание, что из–за особенностей машинной арифметики в последних двух цифрах результата — ошибка: мы должны были бы иметь 80 вместо 88!

Имейте ввиду, что EXP(X)=

- { 0, если -149.668≤X≤-147.366 (потеря всех значащих цифр!);
- { «Overflow» («Переполнение»), если X≥145.063 или -297.033≤X<-149.668.

Вызов встроенной функции осуществляется путём указания в нужном месте программы *имени* функции (ABS, LOG, EXP и т.д.) и её аргумента, заключённого в круглые скобки. После вычисления значения встроенной функции вызов функции заменяется вычисленным значением и затем продолжается вычисление выражения, содержащего вызов функции.

Поговорим теперь о функциях CSNG(X), CINT(X) и CDBL(X).

Функция CSNG(X) вычисляет значение выражения X. Результат округляется до 6 значащих цифр, при округлении проверяется седьмая значащая цифра (счёт идёт слева направо), если она больше или равна 5, то к шестой цифре прибавляется 1.

Если аргумент отсутствует, является строкой или имеет значение экспоненты от недопустимого аргумента, то выдаётся сообщение об ошибке

«Type mismatch»
(*«Несоответствие типов»*)

или

«Overflow
(*«Переполнение»*).

Эта функция позволяет округлить результат вычислений, т.е. избавиться от «лишних» цифр результата.

Например, CSNG(5.1234567890)= 5.12346 .

Функция CINT(X) вычисляет значение выражения X и отбрасывает дробную часть результата. Если значение выражения X выходит за границы диапазона представления целых чисел (от -32768 до 32767), то будет сообщение об ошибке «Overflow».

Например,

- CINT(5.123)=5 ,
- CINT(-4.56)=-4 ,
- CINT(400012.4) даёт «Overflow».

Функция CDBL(X) вычисляет значение выражения X и преобразует его в формат двойная точность: четырнадцать десятичных цифр.

Например, `CDBL(5.456456456456456)` даёт 5.4564564564565 .

Если значение выражения X слишком велико (≥ `EXP(145.063)`), то выдаётся сообщение «Overflow».

Однако, на практике функция `CDBL()` не применяется, т.к. она *неявно* выполняется всякий раз, когда происходит присвоение значения переменной двойной точности. Основное её назначение — обеспечить *совместимость* версии MSX BASIC с другими версиями языка BASIC.

I.7.5. Функция RND. Псевдопеременная TIME

Подробнее остановимся на функции `RND`. При *первом* прочтении раздел, касающийся описания этой функции можно (и, пожалуй, нужно!) *пропустить*!

Вначале отметим, что *случайные и псевдослучайные* числа — числа, последовательность появления которых обладает теми или иными статистическими закономерностями. Различают *случайные* числа, генерируемые (образуемые) каким–либо стохастическим устройством, и *псевдослучайные* числа, конструируемые с помощью арифметических алгоритмов.

Надо сказать, что используемые в современных компьютерах арифметические алгоритмы обладают недостатком, несовместимым с понятием случайности: последовательность выдаваемых ими чисел периодична. И хотя повторяющийся фрагмент этой последовательности может быть весьма длинным (миллионы, миллиарды чисел), случайной её уже не назовёшь. Поэтому и принято название: *датчик* (или *генератор*) псевдослучайных чисел.

Случайные и псевдослучайные числа используются на практике в теории игр, математической статистике, методе статистических испытаний для конкретной реализации недетерминированных алгоритмов и поведения, предсказуемого лишь «в среднем».

Например, если очередное случайное число равно 0, то игрок выбирает первую стратегию, а если 1, то — вторую.

Общий вид обращения к функции `RND`:

```
RND(α)
```
,

где:

- `RND` («RaNDom» — «случайный») — имя встроенной функции;
- α — арифметическое выражение.

Значением функции является *псевдослучайное* число β. β не является полностью случайным, однако, период повторения так велик, что его можно рассматривать как случайное.

В дальнейшем под *инициализацией* переменной (аргумента) будем понимать присвоение переменной значения.

Для аргумента α возможны три варианта:

- `α < 0`; говорят, что при `α<0` происходит «выбор случайной последовательности из уже имеющейся совокупности случайных последовательностей».

  Генератор псевдослучайных чисел выдаёт новое псевдослучайное число при каждом *новом* отрицательном значении α. Последующие обращения к генератору псевдослучайных чисел с помощью функции `RND` будут возвращать в Вашу программу такое же псевдослучайное число.

  Пример 1.
  01-01.bas
  01-01.bas

  ```
  10 INPUT N
  20 FOR I=1 TO 3:PRINT CSNG(RND(-ABS(N))):NEXT I
  ```

  ```
  run
  ? 1
  ```

```
 .0438982
 .0438982
 .0438982
Ok

run
? 2
 .943898
 .943898
 .943898
Ok

run
? 3
 .843898
 .843898
 .843898
Ok
```

Обратите внимание, что при выборе случайной последовательности используется *неслучайный* аргумент! Чуть ниже будет указано, как выбрать последовательность «псевдослучайно».

Таким образом, если Вы хотите при каждом прогоне Вашей программы получать одно и то же псевдослучайное число, поместите в начале этой программы строку вида:

```
Z = RND(отрицательное число)
```

- α > 0 ; возвращается следующее псевдослучайное число из последовательности, заданной последней инициализацией генератора; последовательность псевдослучайных чисел одна и та же для любого положительного значения α. Говорят, что при α>0 происходит «выбор случайного числа из выбранной ранее случайной последовательности»;

- α = 0 ; повторяется вывод предыдущего случайного числа. Это удобно, поскольку таким образом можно сохранить последнее используемое в программе псевдослучайное число.

Пример 2.
01-02.bas
01-02.bas

```
10 INPUT N
20 FOR I=1 TO 3:?CSNG(RND(ABS(N))):NEXTI:?CSNG(RND(0))
```

```
run
? 1
 .595219
 .106586
 .765977
 .765977
Ok

run
? 2
 .595219
 .106586
 .765977
 .765977
Ok
```

Все генерируемые псевдослучайные числа содержат 14 значащих цифр и находятся в интервале (0,1).

Для генерации целого псевдослучайного числа, лежащего на отрезке [X,Y], применяется оператор присваивания вида:

```
Z = INT((Y-X+1)*RND(1)+X)
```

Пример 3.
01-03.bas
01-03.bas

```
10 INPUT X,Y
20 PRINT INT((Y-X+1)*RND(1)+X);:GOTO 20
run
? 1,5
 3  1  4  3  4  1  2  5  4  3  5  3
Break in 20      (нажато "CTRL"+"STOP")
Ok
```

При повторном запуске программы будем иметь:

```
run
? 1,5
 3  1  4  3  4  1  2  5  4  3  5  3
Break in 20      (нажато "CTRL"+"STOP")
Ok
```

Ясно, что для получения целого псевдослучайного числа, лежащего на отрезке [0,9], можно применить оператор присваивания:

```
Z = INT(10*RND(1))
```

Обычно, желательно получать «абсолютно непредсказуемые» псевдослучайные числа. Чтобы добиться этого, прежде всего необходимо инициализировать генератор псевдослучайных чисел в программе также псевдослучайным числом. Для этого в начале программы используется оператор вида:

```
X = RND(-TIME)
```

Помните, что такая инициализация должна осуществляться только один раз!

Всюду далее, в тех местах программы, где необходимо получить случайное число, пишется выражение `RND(1)` .

Пример 4.
01-04.bas
01-04.bas

```
10 INPUT X,Y:G=RND(-TIME)
20 PRINT INT((Y-X+1)*RND(1)+X);:GOTO 20
run
? 1,5
 1  2  4  3  4  4  2  2  2  2  1  2  4
Break in 20     (нажато "CTRL"+"STOP")
Ok

run
? 1,5
 4  1  1  3  2  3  1  2  3  4  1
Break in 20    (нажато "CTRL"+"STOP")
Ok
```

Отметим, что конструкция языка программирования, которая может быть использована в контексте, предполагающем присваивание значения, называется *псевдопеременной*.

В MSX BASIC имеется шесть псевдопеременных:

1. `TIME`
2. `SPRITE$()`
3. `MID$()`
4. `VDP()`
5. `MAXFILES`
6. `BASE()`

Поговорим о псевдопеременной `TIME`.

MSX–компьютер обладает счётчиком, который называется *таймером* и запускается автоматически при включении машины. Таймер принимает целые значения из диапазона [0,65535]. Дойдя до 65535, он снова начинает вести отсчёт от нуля; таймер обновляется каждые 18.2 минуты. Более длительные интервалы времени можно задавать программным путём или с помощью специальной аппаратуры. Очевидно, что 60 «тиков» таймера соответствуют одной секунде времени. Показания таймера и значение псевдопеременной `TIME` *всегда совпадают*! Поэтому Вы всегда можете узнать показания таймера при помощи оператора присваивания вида:

```
X = TIME
```
,

где `TIME` («time» — «время») — служебное слово.

Более того, Вы имеете возможность устанавливать новое значение таймера, используя оператор присваивания

```
TIME = α
```
,

где α — арифметическое выражение.

В результате псевдопеременная `TIME` получает значение, равное целой части значения арифметического выражения α (0≤`INT(`α`)`≤65535).

Пример 5.
01-05.bas
01-05.bas

```
NEW
Ok
10 INPUT T:TIME=T
20 FOR M=1 TO 1000:NEXT
30 PRINT TIME;:GOTO 20
run
? 0
 134  269  403  538  672  807  941 …
Ok

run
? 64999
 65132  65267  65402  0  134  269  403 …
Ok
```

Пример 6.
01-06.bas
 01-06.bas

```
10 TIME=0
20 IF TIME=65535 THEN PRINT TIME/60;"сек" ELSE GOTO 20
run
 1092.25 сек
Ok

далее
print 1092.25/60
 18.204166666667
Ok
```

Используя псевдопеременную `TIME`, можно учитывать время работы программы.

Пример 7.
01-07.bas
01-07.bas

```
NEW
```

```
Ok
10 TIME=0
20 FOR K=1 TO 10000:NEXT
30 PRINT TIME/60;"сек"
```

Используя эту программу, получим забавные таблицы для компьютеров MSX 1!

Дисплей монохроматический! Время измеряется в секундах!	**K%**	**K!**	**K#**	**K**
`FOR K=1 TO 10000:NEXTK`	15.3	24.7	27.1	27.3
`FOR K=1 TO 10000:NEXT`	12.3	21.6	23.9	23.9
Дисплей цветной! Время измеряется в секундах!	**K%**	**K!**	**K#**	**K**
`FOR K=1 TO 10000:NEXTK`	15.58	25.20	27.67	27.87
`FOR K=1 TO 10000:NEXT`	12.53	22.03	24.42	24.42

... а теперь для компьютеров MSX 2 (сетевой вариант):

Дисплей монохроматический! Время измеряется в секундах!	**K%**	**K!**	**K#**	**K**
`FOR K=1 TO 10000:NEXTK`	12.90	22.00	24.13	24.18
`FOR K=1 TO 10000:NEXT`	10.25	19.02	21.08	21.15
Дисплей цветной! Время измеряется в секундах!	**K%**	**K!**	**K#**	**K**
`FOR K=1 TO 10000:NEXTK`	14.28	24.08	26.60	26.70
`FOR K=1 TO 10000:NEXT`	11.37	20.77	23.37	23.27

... и наконец, для компьютера MSX 2, отключённого от локальной сети:

Дисплей монохроматический! Время измеряется в секундах!	**K%**	**K!**	**K#**	**K**
`FOR K=1 TO 10000:NEXTK`	7.98	17.22	19.58	19.80
`FOR K=1 TO 10000:NEXT`	5.05	14.18	16.48	16.48
Дисплей цветной! Время измеряется в секундах!	**K%**	**K!**	**K#**	**K**
`FOR K=1 TO 10000:NEXTK`	8.37	18.07	20.55	20.77
`FOR K=1 TO 10000:NEXT`	5.28	14.87	17.30	17.28

Запомните, что большинство операций ввода-вывода выключают таймер, и он начинает отсчёт заново после окончания ввода-вывода!

Пример 8.
01-08.bas
01-08.bas

```
NEW
Ok
10 DIM M(50,50)
20 TIME=0:A=1:PRINT TIME
run
 36
Ok

NEW
Ok
10 DIM M(50,50),A
20 TIME=0:A=1:PRINT TIME
```

```
run
 1
Ok
```

Приведём пример использования функции RND и псевдопеременной TIME.

Пример 9.
01-09.bas
01-09.bas

```
NEW
Ok
5 'Нахождение числа π методом Монте-Карло
10 X=RND(-TIME):INPUT"Количество бросаний точки  -";N
20 FOR I=1 TO N:X1=RND(1):X2=RND(1)
30 IF X1^2+X2^2<1 THEN IN=IN+1
40 NEXT:PRINT "π≈";4*IN/N
run
Количество бросаний точки - 100
π≈ 3.4                (π≈ 3.2)
Ok
run
Количество бросаний точки - 200
π≈ 3.12               (π≈ 3.24)
Ok
run
Количество бросаний точки - 500
π≈ 3.208               (π≈ 3.008)
Ok
run
Количество бросаний точки - 750
π≈ 3.2213333333333    (π≈ 3.2171428571429)
Ok
run
Количество бросаний точки - 1000
π≈ 3.204               (π≈ 3.156)
Ok
```

Справа в круглых скобках приведены результаты счёта при повторном запуске данной программы через некоторое время с теми же значениями N.

I.8. Выражения

Возможность использования выражений является одним из главных преимуществ языков программирования высокого уровня перед машинными языками.

Выражение — последовательность операндов, соединённых знаками операций, а при необходимости — и *круглыми* скобками так, что в результате выполнения операций получается единственное значение, которое называется *значением выражения*.

Напомним, что под *операндом* мы понимаем либо константу, либо переменную (простую или индексированную), либо встроенную функцию, либо функцию пользователя (см. раздел IV.3).

Заметим, что в частном случае выражение может содержать только константу, имя переменной, вызов встроенной функции или функции пользователя.

Итак, например, простейшими выражениями являются:

```
5
-.5E6
X1
```

```
"A"
COS(Z)
X-2>=0
```

Вычисление значения выражения заключается в том, что вместо имён переменных и функций подставляются их значения, и выполняются заданные операции. При этом учитываются *правила старшинства операций*. Напомним, что вычисление значения встроенной функции или функции пользователя имеет *наивысший* приоритет!

В отличие от обычной математической записи выражения в BASIC записываются в одну строку без подстрочных и надстрочных индексов. Причём, если в математике можно опустить знак умножения при записи алгебраических выражений (например, 2a+3b), то в BASIC это не допускается (надо писать: 2*a+3*b). Нетрудно понять, чем вызвано введение такого правила: без него невозможно определить, означает ли AB умножение A на B или это — имя переменной.

Менять порядок вычислений (чтобы застраховать себя в сомнительных случаях) программист может с помощью круглых скобок. Порядок работы со скобками соответствует общепринятому в математике, то есть вначале вычисляется значение выражения, стоящего в скобках.

Например, значение выражения `A+B*COS(Z)` вычисляется так же, как и значение выражения `A+(B*COS(Z))` .

Если в выражении *нет скобок*, то надо внимательно следить за *порядком старшинства операций*.

Например, многие начинающие программисты записывают для вычисления значения дроби $\frac{A}{2 \times B}$ в программе выражение: `A/2*B` , которое равносильно выражению $\frac{A}{2} \times B$ ибо приоритет операций «/» и «*» одинаков!

Верно записанное выражение должно иметь вид: `A/2/B` или `A/(2*B)`.

Аналогично, `2^2^K` означает `(2^2)^K` , а не `2^(2^K)`!

Скобки могут вкладываться одни в другие *практически* неограниченно: максимальное количество вложений скобок равно 126.

Если все же выражение не удовлетворяет этому ограничению, выдаётся сообщение:

«Out of memory»
(*«Не хватает памяти»*).

I.8.1. Арифметические и строковые выражения

Арифметическим выражением будем называть выражение,значением которого является число.

Примеры арифметических выражений:

- `-154.567E-1` и `-154.567*10^(-1)` (укажите различие!)
- `p1`
- `D-B^2+4*a*C`
- `SQR(A^2+B^2-2*A*B*cos(X))`
- `(-B+ABS(D))/(2*A)`
- `T%+c%*n%-4`

Строковым выражением будем называть выражение, значением которого является строка.

Примеры строковых выражений:

- `"А Б Б А"`
- `AH$`
- `s$+" "+"2"`

- `MID$(A$,3,7)`

I.8.2. Логические выражения

Но да будет слово ваше: да, да;
нет, нет; а что сверх этого, то от лукавого.

—Матф.,5,37

Наряду с арифметическими и строковыми выражениями можно рассматривать *логические выражения*. Разберём вначале частный случай логического выражения — *отношение*.

Отношение — выражение, состоящее из двух арифметических или строковых выражений, связанных знаком операции отношения.

Например,

- `X*X>=0`
- `5<2`
- `"A">"a"`
- `Z$+MID$(X$,2,N%)="a"`

Для *программиста* значением *отношения* может быть либо T (True — *истина*), либо F (False — *ложь*).

Так, первое из приведённых выражений имеет значение T, второе — F. Четвёртое выражение принимает то или иное значение в зависимости от значений строковых переменных Z$, X$ и целочисленной переменной N% .

Для компьютера значением *отношения* будет:

- -1, если *отношение* истинно;
- 0, если *отношение* ложно.

Приведём пример нестандартного применения отношений. С помощью «смешанного» (оно содержит арифметические операции и операции отношения) выражения вида:

```
-(X>=1)*X^2-(X<1)*(X+4)
```

программист может вычислить значение кусочно–непрерывной функции Y, описываемой формулами:

$$Y=\begin{cases} X^2, if\ X \geqslant 1, \\ X+4, if\ X<1. \end{cases}$$

Отметим, что с помощью оператора `IF` (см. раздел III.2.) вычисление значения функции Y производится следующим образом:

```
IF X>=1 THEN Y=X^2 ELSE Y=X+4  .
```

Логическим выражением будем называть последовательность отношений, соединённых знаками *логических* операций и знаками операций отношения.

В логических выражениях, так же как в арифметических и строковых, можно (и нужно!) использовать круглые скобки. Гораздо легче запутаться в логических выражениях, чем в арифметических, поэтому если в логическое выражение входят несколько логических операций, то *обязательно* используйте *скобки*! При отсутствии скобок необходимо учитывать приоритет операций.

Значением логического выражения для *программиста* может быть одно из двух логических значений: T или F.

Значением логического выражения *для ЭВМ* является:

- целое число,отличное от нуля (этот случай соответствует логическому значению T);
- нуль (этот случай соответствует логическому значению F).

Логические выражения часто называют *условиями*. Если выражение имеет значение T, то говорят, что соответствующее условие *справедливо* (выполняется). В противном случае говорят, что условие не справедливо (не выполняется).

Примеры:

1.
```
   Z=1     AND     F<4
   ─▲─     ─▲─     ─▲─
    │       │       │
отношение  │  отношение
           │
знак логической операции
```

2.
```
    NOT  ( A<B    OR    C<D )
    ─▲─    ─▲─   ─▲─    ─▲─
     │      │     │      │
  │ отношение │ отношение
     │             │
знаки логических операций
```

3.
```
  (Z>1)   =   (F<=41)
  ──▲──   ▲   ───▲───
    │     │      │
отношение │ отношение
          │
знак операции отношения
```

4. написать логическое выражение, которое имеет значение T при выполнении следующего условия:

    1. значение переменной X находится в одном из двух интервалов (A,B) и (C,D).

       Ответ:

       ```
       (X>A AND X<B) OR (X>C AND X<D)  ;
       ```

    2. точка с координатами (x,y) принадлежит первой четверти единичного круга с центром в начале координат.

       Решение. Принадлежность точки (x,y) единичному кругу с центром в начале координат определяется значением отношения `X^2+Y^2<=1` . Отношение `X>=0` истинно для точек, принадлежащих правой полуплоскости, а отношение `Y>=0` — для точек верхней полуплоскости. Объединяя эти отношения логической операцией `AND`, получим искомое логическое выражение, истинное для точек, принадлежащих заданной области:

       ```
       X^2+Y^2<=1 AND X>=0 AND Y>=0  ;
       ```

    3. из отрезков длиной A, B и C можно построить треугольник.

       Решение. Как нетрудно видеть, искомое логическое выражение имеет вид:

       ```
       (A<B+C)AND(B<A+C)AND(C<A+B)
       ```

       или

       ```
       (A+B>C) AND (C>ABS(B-A)) ;
       ```

    1. ближайшее к значению переменной X целое число чётно и отлично от нуля. Ответ:

       ```
       FIX(X+.5)MOD2=0ANDFIX(X+.5)<>0  ;
       ```

    2. при делении значения целой переменной X на значение целой переменной Y в остатке получается число, делящееся нацело на значение целой переменной Z.

       Ответ:

```
(X%MODY%)MODZ%=0  ;
```

5. записать логическое выражение

   ```
   (A+5=0)AND(B=0)AND(K=5*B)  ,
   ```

   не прибегая к символам логических операций.

   Ответ:

   ```
   ABS(A+5)+ABS(B)+ABS(K-5*B)=0  .
   ```

   Таким образом, логическое выражение

   ```
   ((A+5=0) AND (B=0) AND (K=5*B)) EQV (ABS(A+5)+ABS(B)+ABS(K-5*B)=0)
   ```

   имеет значение T;

6. заметим, что, вообще говоря, при записи логических выражений (условий) можно обойтись без логических операций `XOR`, `EQV`, `IMP`, то есть логические выражения вида:

   - `ZIMPY=(NOTZ)ORY`,
   - `ZXORY=(ZAND(NOTY))OR((NOTZ)ANDY)`,
   - `ZEQVY=NOT(ZXORY)`

   истинны (имеют значение T).

Ещё раз настойчиво рекомендуем Вам, записывая логические выражения, не жалеть скобок и не пользоваться логическими операциями, смысл которых вам непонятен!

I.9. Дополнение

> Я уверен, вы согласитесь со мной, что если страница 534 застает нас только во второй главе,то первая должна быть невыносимо длинной.
>
> *—А.Конан Дойль*

Одним из критериев оценки производительности компьютеров является время выполнения определённых тестовых программ — «бенчмарков» («benchmark» — «эталонный текст», «benchmarking» — «определение эффективности системы (ЭВМ или программного обеспечения) посредством выполнения эталонных программ или обработки эталонных наборов данных»).

В книге [21] приведено сравнение результатов выполнения одной из таких программ на ряде компьютеров и микрокалькуляторов.

Программа — benchmark
bm.bas
Запуск программы в WebMSX

```
NEW
Ok
110 PRINT "Начало";:TIME=0
120 K=0:DIM M(5)
140 K=K+1               '◄──┐
150 A=K/2*3+4-5         '   │
160 GOSUB 230 '──►      '   │
170 FOR L=1 TO 5        '   │
180   M(L)=A            '   │
190 NEXT L              '   │
```

```
200 IF K<1000 THEN 140 '⟶┘
210 PRINT TIME/60;"сек":PRINT "Конец"
220 END
230 RETURN '⟶
```

Результаты эксперимента следующие:

Yamaha YIS-503IIR Компьютер серии MSX 1	57.2 сек
Yamaha YIS-805-128R2 Учительский компьютер серии MSX 2, отключённый от сети	47.45 сек
Персональные компьютеры	
IBM PC	37 сек.
Apple IIe	46 сек.
Искра 226	49 сек.
Tandy Color	51 сек.
Электроника НЦ–80–20	56 сек.
Epson HX–20	101 сек.
CM–1800	104 сек.
Калькулятор FX–702P фирмы «Casio»	>20 минут

Диск с примерами

Загрузить образ диска

Открыть диск в WebMSX

---

http://sysadminmosaic.ru/msx/basic_dialogue_programming_language/001

2023-05-18 21:57