

МОУ СОШ №1  
г.Белинский

Язык программирования Бейсик

(для элективных курсов и факультативов)

Основные команды Бейсика. Метод  
анализа Бейсик - программ.  
Основные конструкции.

Разработал:

учитель информатики

Дедов А. И.

# План занятий

1. История возникновения языков программирования.
2. Компиляция и интерпретация.
3. Диаграммный метод анализа Бейсик - программ.
4. Основные команды языка Бейсик и их исполнение.

Под **ЯЗЫКОМ** понимают любую систему знаков

( знак-это объект, специально выделенный для передачи информации: буква на бумаге, жест, выражение лица, положение переключателя и тому подобное).

Вместе с языками возникла проблема трансляции, т.е. перевода (**translator**) с одного языка на другой.

Слова составляющие язык ЭВМ (машинный язык) весьма далеки от понятий которыми оперирует человек: регистр, переслать ...- и все это в двоичных кодах.

Первым шагом на пути «очеловечивания» машинного языка стало создание программ, переводящих символические имена в машинные коды. Такие программы называют **ассемблерами** .

В 1958 году вступил в строй транслятор **Фортрана**-первого широко используемого языка высокого уровня (ЯВУ).

Количество существующих языков в настоящее время составляет несколько тысяч. Вот некоторые из них:

**Алгол-60** -явился основой многих современных языков.

**Кобол**- для решения экономических задач.

**Снобол- 4** - для обработки текста.

**Лисп** – в области искусственного интеллекта.

Современные языки, такие как, Паскаль , Си , объектно-ориентированный язык Visual Basic и др. , предназначены для решения широкого круга задач.

**Basic** - самый простой язык высокого уровня, создан специально для обучения программированию.

Для обучения туземцев английскими миссионерами в позапрошлом веке был разработан так называемый «бейсик-инглиш»-небольшое подмножество английского языка ( в нём было , например, всего 18 глаголов).  
Различия между языком Бейсик и, современными языками программирования вполне соответствуют различиям между «бейсик-инглиш» и языком Шекспира.  
Все же изучение Бейсика позволяет легко понять многие существенные черты современных языков программирования.

Существуют два способа трансляции.

Первый - *интерпретация*; он соответствует устному переводу (*interpretation*), т.е. каждая инструкция исходной программы переводится и сразу выполняется. При этом способе производится многократное декодирование повторяющихся инструкций.

Второй – *компиляция*; он соответствует письменному переводу, когда перед выполнением собирается перевод всего текста программы (*compile*- собирать).

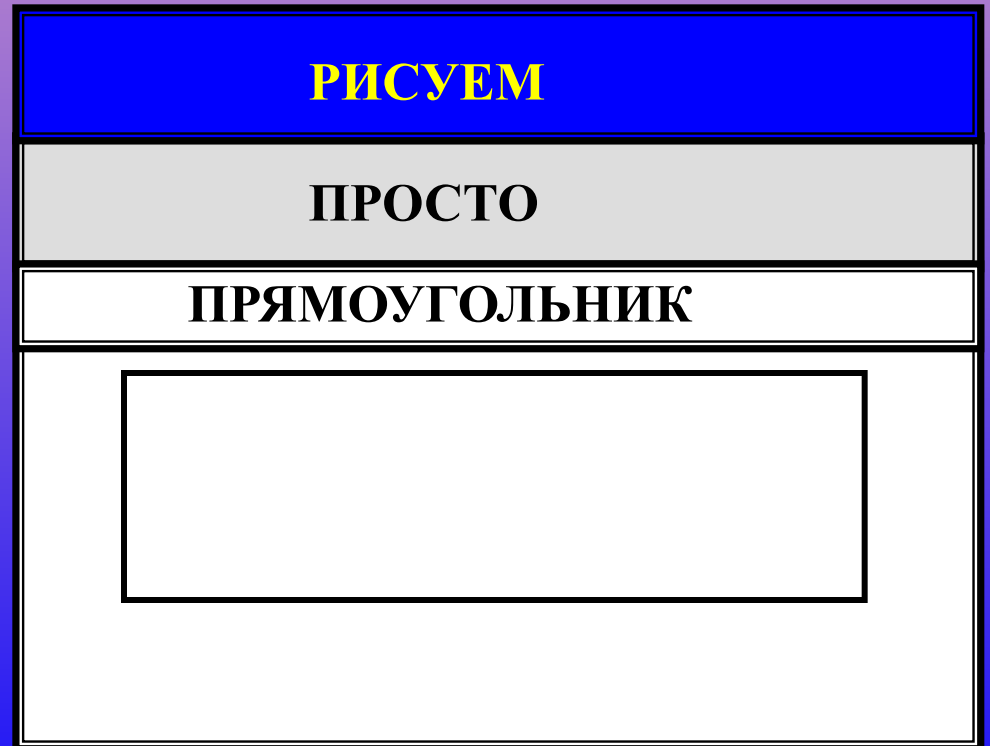
Интерпретация проще реализуется, обеспечивает большую гибкость, но компиляция, как правило, дает гораздо более эффективную программу. Часто используются оба варианта вместе: интерпретатор- для отладки и компилятор для трансляции отлаженной программы.

# Интерпретация

Программа

```
10 PRINT "РИСУЕМ"  
20 PRINT "ПРОСТО"  
30 PRINT "ПРЯМОУГОЛЬНИК"  
40 LINE(20,30)-(150,80),8,B
```

Исполнение



# Компиляция

Программа

```
10 PRINT "РИСУЕМ"  
20 PRINT "ПРОСТО"  
30 PRINT "ПРЯМОУГОЛЬНИК"  
40 LINE(20,30)-(150,80),8,B
```

Исполнение

**РИСУЕМ**

**ПРОСТО**

**ПРЯМОУГОЛЬНИК**





# Интерпретатор и основные команды Бейсика

К интерпретатору одна за другой поступают строки программы на Бейсике.

Оператор Бейсика всегда начинается с ключевого слова (например **PRINT**- печатать).

Получив исходный текст программы, из неё выделяется отдельный оператор и распознаётся по ключевому слову. Но операторов не мало, и в одиночку с ними не справиться. У Бейсика есть помощники –это программы, реализующие операторы языка (каждая свой). Так что остается только передать «остаток» оператора (без ключевого слова), который представляет собой указание, с чем и что нужно сделать, соответствующей программе (далее, для удобства, будем называть программы по имени соответствующего оператора).

# Оператор LET (оператор присваивания)

Распознав оператор, интерпретатор передает управление программе, которая умеет выполнять присваивания. Получив остаток оператора, имеющий в нашем случае вид  $A=B$ , программа LET

присвоит переменной с именем **A** значение переменной с именем **B**.

Программа LET понимает и арифметические выражения, например:

$A=B+C$

A присвоить B+C

То, что находится справа от знака  $=$ , LET передаст некой программе (назовём ее Арифмометр), которая умеет вычислять значения разных выражений; он вернет программе LET значение выражения (число), оно будет присвоено переменной, имя которой указано слева от знака  $=$

## Откуда берутся переменные ?

**Переменная**, это область памяти (ячейка) предназначенная для хранения данных. В качестве имени переменной используется латинская буква (идентификатор). Если букв не хватает, то используется сочетание типа «латинская буква с цифрой», например W2, X0 и т.п.

( Существуют также варианты интерпретатора, воспринимающие и более длинные имена, например до 6 символов.)

После просмотра программы у интерпретатора остается следующая таблица для всех встретившихся переменных.

Имя переменной	Ссылка на ячейку в которой хранится переменная
----------------	--

## Оператор **PRINT** (оператор печати)-

читает слева направо остаток, который должен представлять последовательность выражений и символов, управляющих печатью.

Например:

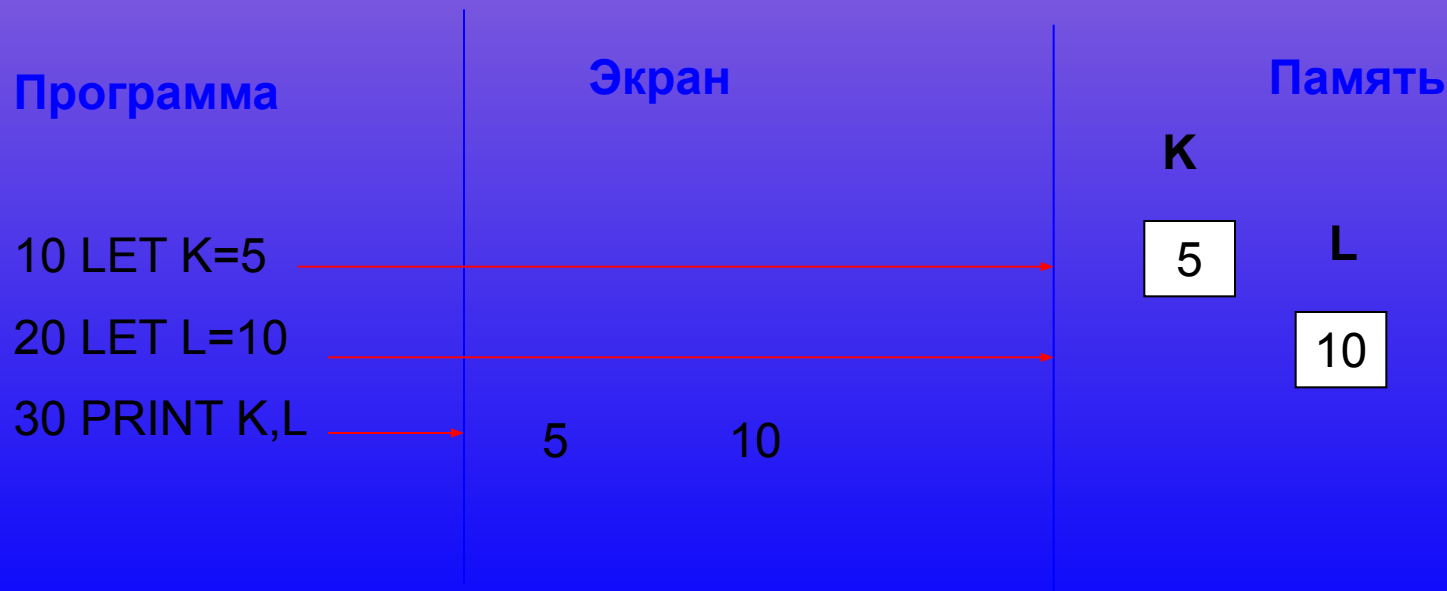
- а) **PRINT S** - ЭВМ печатает на экране видеомонитора значение переменной S
- б) **PRINT A/5** – прочитав выражение , PRINT передаёт его Арифмометру, а полученное значение печатает на экране видеомонитора.
- в) **PRINT "ЭВМ"**- компьютер печатает на экране текст {ЭВМ }

# Исполнение Бейсик - программ.

Процесс исполнения программ иллюстрируется диаграммами, в основе которых текст программы и объекты действия операторов Бейсика: фрагмент экрана дисплея и используемые ячейки памяти машины. Диаграмма состоит из трёх самостоятельных частей:

текста программы, экрана и набора ячеек памяти.

С помощью стрелок указывают на объекты действия исполняемого оператора либо осуществление передачи управления другому оператору программы.



Перед пуском программы в памяти машины имеется участок, специально отведённый под хранение данных. Состоит этот участок из достаточно обширной последовательности ячеек памяти, каждая такая ячейка первоначально не имеет определённого имени и не содержит в себе конкретных значений. Этот факт можно отразить при первоначальном оформлении диаграммы, включая в её заголовок последовательность пустых, безымянных ячеек .

**Программа**

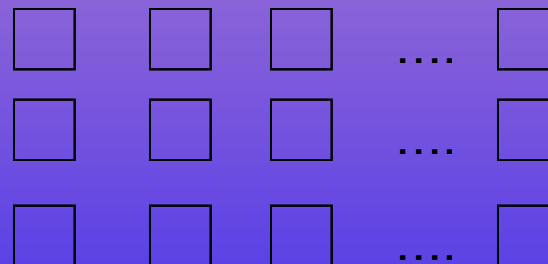
10 LET P=7

20 LET K=P\*P

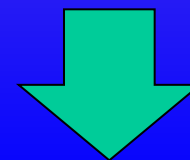
30 PRINT K

**Экран**

**Память**



Только после запуска программы, в процессе выполнения, происходит обращение к ячейкам, в результате чего они получают соответствующие имена и значения. Таким образом исполнив предыдущую программу, мы получим диаграмму.



## Программа

```
10 LET P=7
```

```
20 LET K=P*P
```

```
30 PRINT K
```

## Экран

49

## Память

P

7

K

4

9

Данный пример иллюстрирует как предварительную подготовку к исполнению (оформление заголовка диаграммы) и запись текста исполнения (последовательность исполнения операторов входящих в программу).

Дальше посмотрим как исполняются некоторые из основных операторов алгоритмического языка Бейсик.



# Основные команды

LET

GOTO

PRINT

IF-THEN

END

IF-THEN-ELSE

REM

GOSUB, RETURN

READ, DATA

FOR-TO, NEXT

INPUT

DIM

## Программа

## Экран

## Память

10 LET A=3

20 LET B=7

30 LET C=A\*B

40 LET D\$="ЭВМ"

50 LET E\$="П"+D\$

60 LET F=cos(3,14)\*A+B

70 LET G=C-B

A

3

B

7

C

21

D\$

ЭВМ

E\$

ПЭВМ

F

4

G

17

Выполняя оператор **LET** ЭВМ вычисляет значение выражения, стоящего справа от знака "=", а полученный результат заносит в ячейку памяти, имя которой указано слева от знака "=" или же просто заносит в переменную значение записанное справа от знака "=".



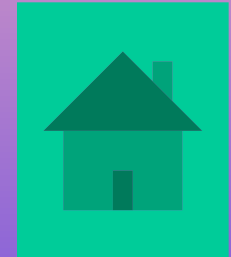
# Оператор PRINT (продолжение программы)

## Программа

```
90 PRINT 3.14
100 PRINT "константа"
110 PRINT F
120 PRINT E$
130 PRINT 13*2+4
140 PRINT INT(B/A)+INT(A/B)
150 PRINT E$+"*Корвет*"
160 PRINT "A*B=",C
170 PRINT A,B,C+B
180 PRINT "Числа:"; A,F
190 PRINT "Слова:";D$,E$
```

## Экран

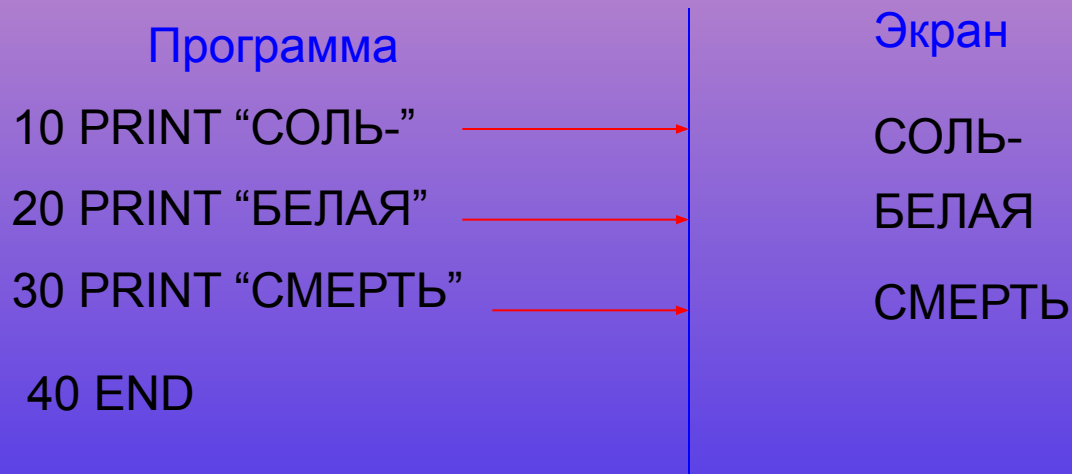
```
3.14
константа
4
ПЭВМ
30
2
ПЭВМ*Корвет*
A*B=21
3      7      28
Числа: 3      4
Слова: ЭВМ   ПЭВМ
```



Выполняя оператор **PRINT**, ЭВМ может печатать на экране: а) постоянное значение(90,100); б) содержимое переменной (110,120) ; в) значение выражений (130,140); г) совокупность перечисленных выше элементов, отделённых запятой или точкой с запятой (160,190)

## Оператор конца программы **END**.

Встретив его, ЭВМ прекращает выполнение программы, а для человека, исполняющего программу, данный оператор является признаком завершения работы.



**Работа программы завершена**



Оператор комментария **REM** является неисполняемым. Встретив в программе оператор **REM**, ЭВМ автоматически переходит к выполнению следующего по порядку оператора.

### Программа

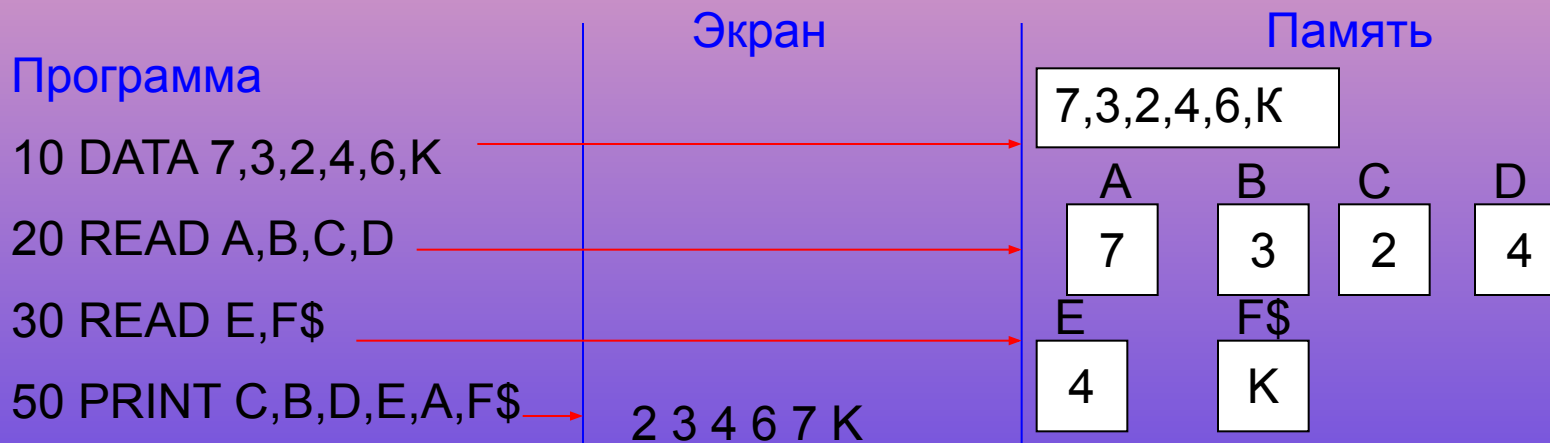
```
10 REM Вычисление площади  
20 REM куга, R=3 см  
30 REM  
40 PRINT "Площадь круга ="3.14*3^2  
50 END
```

### Экран

Площадь круга =38.27



## Операторы ввода списка данных READ , DATA



Область данных DATA можно также изобразить в памяти компьютера, а выполняя оператор READ, ЭВМ записывает постоянные значения, собранные в операторе DATA, в переменные в указанными в операторе READ именами.



## Оператор ввода значений с клавиатуры **INPUT**

Выполняя оператор **INPUT**, ЭВМ в качестве приглашения для ввода информации выдает на экран знак «?». В ответ человек должен с помощью клавиатуры набрать необходимые значения и нажать клавишу «**ВВОД**». После чего набранная информация записывается в переменные.

### Программа

```
10 PRINT "Введите длины катетов"
```

```
20 INPUT A,B
```

```
30 LET C=SQR(A^2+B^2)
```

```
50 PRINT "Гипотенуза=",C
```

### Экран

Введите длины катетов

?

3

4



Гипотенуза=5

### Память

A

3

B

4

C

5



## Оператор безусловного GOTO

Выполняя оператор **GOTO**, ЭВМ передаёт управление оператору программы, номер строки которой указан после ключевого слова **GOTO**

### Программа

```
10 PRINT "ПРИГОВОР "  
20 GOTO 40  
30 PRINT " КАЗНИТЬ "  
40 PRINT " ПОМИЛОВАТЬ "  
50 PRINT " xxxxxxxxxxxxxxxx "  
60 END
```

### Экран

```
ПРИГОВОР  
  
ПОМИЛОВАТЬ  
xxxxxxxxxxxxxxxxxx
```





## Оператор условного перехода **IF-THEN**

Выполняя оператор **IF-THEN**, ЭВМ в первую очередь проверяет, является ли верным условие, записанное после ключевого слова **IF**. Если условие является верным, то управление передаётся оператору, расположенному после ключевого слова **THEN**. Если условие не является верным, то управление передаётся следующему по порядку оператору программы.

### Программа

```
10 INPUT A,B
```

```
- 20 IF A>B THEN PRINT "1-е больше"
```

```
+ 30 IF B>A THEN PRINT "2-е больше"
```

```
40 END
```

### Экран

? 3 7

2-е больше

### Память

A

3

B

7

При исполнении программы результат проверки условия будет отмечаться знаками:

+ условие является верным;

- условие не является верным.



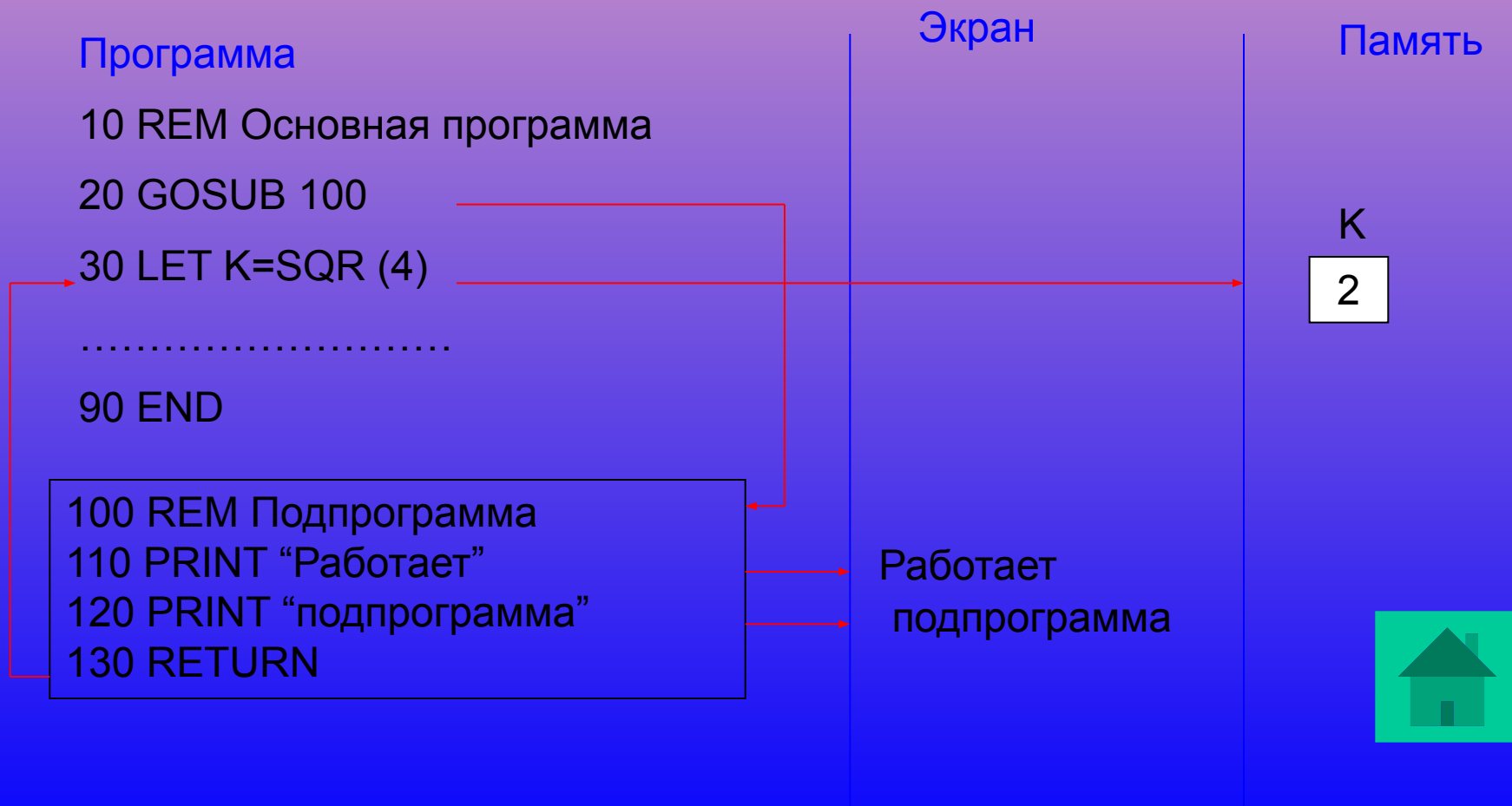
## Оператор IF-THEN-ELSE.

Выполняя оператор **IF-THEN-ELSE** , ЭВМ поступает аналогично предыдущему случаю с той лишь разницей, что если проверяемое условие не является верным , то управление передаётся оператору, расположенному после ключевого слова **ELSE** .



Оператор вызова подпрограммы **GOSUB** , оператор возврата **RETURN**.

Выполняя оператор **GOSUB**, ЭВМ передаёт управление оператору, номер строки которого указан после ключевого слова (т.е. осуществляет переход на подпрограмму). Выполняя оператор **RETURN** , ЭВМ возвращает управление из подпрограммы в основную программу , в то место откуда был осуществлён вызов.



## Программа

```
10 PRINT "Цикл"
```

```
20 FOR I=1 TO 4
```

```
40 A=INT(RND(1)*9
```

```
70 PRINT A
```

```
50 NEXT I
```

## Экран

Цикл

6 1 2 9

## Память

1	2	3	4	I
6	1	2	9	A

На диаграмме показано изменение значений управляющей переменной **I** в памяти компьютера, изменение значения переменной **A** и вывод на экран значений этой переменной. Все действия выполняются в той последовательности, которая определена программой.



## Оператор определения массива DIM

Выполняя оператор DIM , ЭВМ отводит в памяти машины участок (или массив), состоящий из указанного количества переменных. Каждая такая переменная получает имя, зависящее от места её расположения в массиве .

Нумерация ячеек массива начинается с нуля. Заполнение ячеек массива производится аналогично заполнению обычных ячеек (переменных) памяти, то есть с помощью операторов LET , READ, INPUT .

Рассмотрим программу с массивом на следующем примере:

## Программа

```
10 PRINT "Одномерный массив"
```

```
20 DIM A (4)
```

```
30 FOR I=1 TO 4
```

```
40 A(I)=INT(RND(1)*9)
```

```
50 NEXT I
```

```
60 FOR I=1 TO 4
```

```
70 PRINT A(I);
```

```
80 NEXT I
```

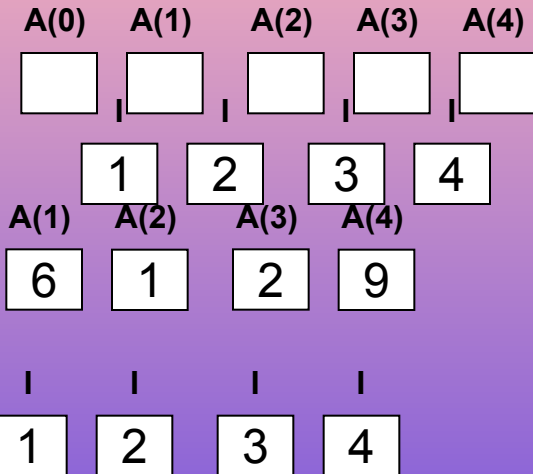
```
90 END
```

## Экран

Одномерный массив

6 1 2 9

## Память



На диаграмме показано как резервируется массив, и как затем идет его заполнение с применением цикла.

На своих занятиях при изучении операторов языка Бейсик я использую следующие задания:



1. Выполнить предлагаемую программу, при этом текст программы включает различные варианты применения рассматриваемого оператора.
2. По фрагменту диаграммы восстановить текст исходной программы.
3. Написать и выполнить программу для решения конкретной задачи.

Важным является сам процесс выполнения программы – наглядное и простое прослеживание последовательности действий, а не плакатная демонстрация уже исполненных программ.

Процесс исполнения программ можно демонстрировать и под управлением докладчика, а не в автоматическом режиме. Для этого достаточно при настройке анимации некоторых объектов в пункте **Начало**, указать не **После предыдущего**, а **По щелчку** и тогда каждая команда в программе будет исполняться после щелчка мышью, что даст возможность учителю делать более продолжительный комментарий.

При создании данной презентации использовалась литература:

А.Г. Гейн «Основы информатики и вычислительной техники».

В.А. Урнов Д.Ю.Климов «Преподавание информатики в компьютерном классе».

Журналы «Информатика и образование» 1989г.