

## § 1

### *Назначение программы*

ALGO - одна из первых оболочек, которая разработана не для профессионального программирования, а для обучения программированию.

Так исторически сложилось, что все изучали программирование, пользуясь профессиональными системами. Это было понятно и на старых больших вычислительных машинах, и на первых персональных компьютерах со встроенным языком Бейсик и однозадачной операционной системой MS DOS. Многозадачные операционные системы с интуитивно понятным для пользователя интерфейсом (MAC OS, Windows, Linux и пр.) потеснили старые системы вместе со старыми компьютерами.

Однако программное обеспечение для программирования настолько усложнилось, что стало практически недоступным для начинающих. Возвращаться с целью изучения понятий переменной, выражения, условия, цикла и т.д. к текстовому Бейсику или Turbo Pascal для DOS не хотелось, поэтому и было создано ALGO - среда с современным общепринятым графическим интерфейсом и простым языком программирования.

Мы имеем лабораторное оснащение для изучения большинства предметов. ALGO - учебная система программирования, то есть аналог лабораторного оснащения для изучения программирования. Это первая реализация. Конечно эта среда будет совершенствоваться, будут расширяться ее возможности и количественно и качественно. Как сегодня уже принято, каждое новое программное пространство имеет достаточно большое соответствие с иными системам программирования.

В среде ALGO реализован язык программирования Pascal. Правды ради, надо сказать, что выбраны только те элементы языка, которые необходимы начинающим для усвоения основ программирования. В этом разделе перечислены основные отличия реализованного языка от языка среды Turbo Pascal, это:



- 1) Не поддерживаются указатели, объекты, UNIT, USES.
- 2) Не реализованы множественные, перечислимые и диапазонные типы, а также тип STRING.
- 3) Файлы поддерживаются только текстовые.
- 4) Описание меток должно быть последним при объявлении, а метки задаются только целыми числами.

В следующих версиях ALGO будут поочередно сниматься эти ограничения, о чем будет указано в этом разделе (*от автора*).

### ***Пользование оболочкой.***

Для того, чтобы выполнить программу, написанную языком Паскаль, необходим компилятор – тоже программа, встроенная или вызываемая специальной командой, который переведет эту программу в коды компьютера. Но для этого нужен текстовый файл с программой на Паскале. Чтобы подготовить такой текст нужно воспользоваться экранным редактором. Кроме того, нужно уметь запустить программу на пошаговое выполнение, знать способы работы с файлами, просмотра промежуточных результатов и многое другое. Все эти способы, собранные вместе, назовем программной средой, а вид этой среды на экране (окна, меню, кнопки) – графической оболочкой или просто оболочкой.

На (Рис. 1) представлен вид оболочки ALGO и показаны основные его элементы.

Оболочка выполнена двумя языками - английским и русским. Для перехода на английский язык нужно нажать кнопку  , а на русский - кнопку  . Это же можно сделать с клавиатуры нажатием Ctrl+E и Ctrl+U соответственно, или через меню (Опции Английский язык; Опции Русский язык)

Для перехода к меню указывают мышкой на соответствующий его пункт или нажимают клавишу Alt, выбирают нужный пункт клавишами управления курсором, после чего нажимают Enter.

Язык оболочки никак не связан с режимом работы клавиатуры. Для переключения клавиатуры с латинских букв на русские и наоборот в большинстве случаев используют правый Ctrl+Shift. Это нужно выяснять в преподавателя или собственника компьютера. Но всегда можно переключить режим работы клавиатуры с помощью иконки в правом нижнем уголке экрана,

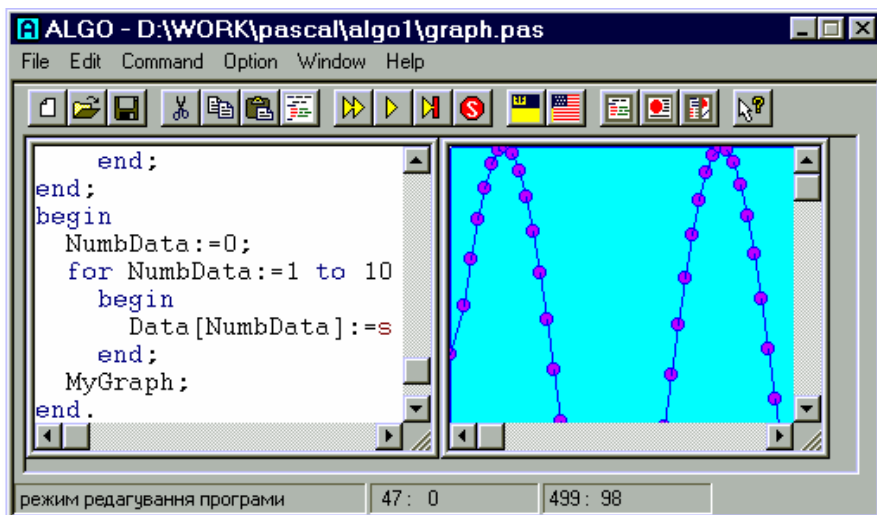
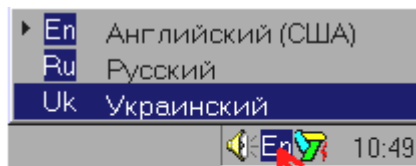


Рис. 1

как показано на рисунке (Рис. 2).

Обратите внимание на окно режима работы программы. Подготовка и исправление текста программы возможно только в режиме редактирования. В остальных случаях для перехода в этот


режим нужно нажать кнопку  или клавишу F6. В окне заголовка указана полный путь и имя файла программы, с которой




Для выбора режима работы клавиатуры нажмите ЗДЕСЬ


Вы работаете. Когда этой информации нет, то это означает, что Вы не записывали программы на диск. Рекомендуется записывать программу после набора каждых 10-20 строк текста и перед первым выполнением. Иначе в результате ошибочных действий или сбоя программы подготовленный Вами текст программы может быть утрачен.


### ***Считывание и запись программ.***

Для того, чтобы считать (загрузить) программу с диска, нужно нажать кнопку  на панели инструментов или выбрать пункт **Файл Открыть** в меню, или нажать клавишу **F3**. На экране появится окно системного диалога, которое имеет такой вид *Рис.2*.

С помощью этого диалога Вы можете выбрать и прочитать любой доступный текстовый файл. Нельзя вводить имена файлов, которых нет.

Для того чтобы создать новый файл, нужно нажать кнопку  на панели инструментов или выбрать пункт **Файл Новый** в меню, или нажать клавишу **F4**.

Аналогичным способом записывают (сохраняют) файлы на диске. Для этого нужно нажать кнопку  на панели инструментов или выбрать пункт **Файл Сохранить** в меню, или нажать клавишу **F2**. Окно системного диалога имеет такой же вид. Когда Вы записываете новый файл, то обязательно надлежит ввести с клавиатуры его имя в соответствующем окне диалога. Если при этом будет указано расширение (.pas, .dat, .txt), то файл будет записан с этим расширением, иначе система автоматически запишет файл с расширением (.pas).

При необходимости записи файла в новую папку (подкаталог) можно создать ее с помощью этого же диалога, нажав кнопку .

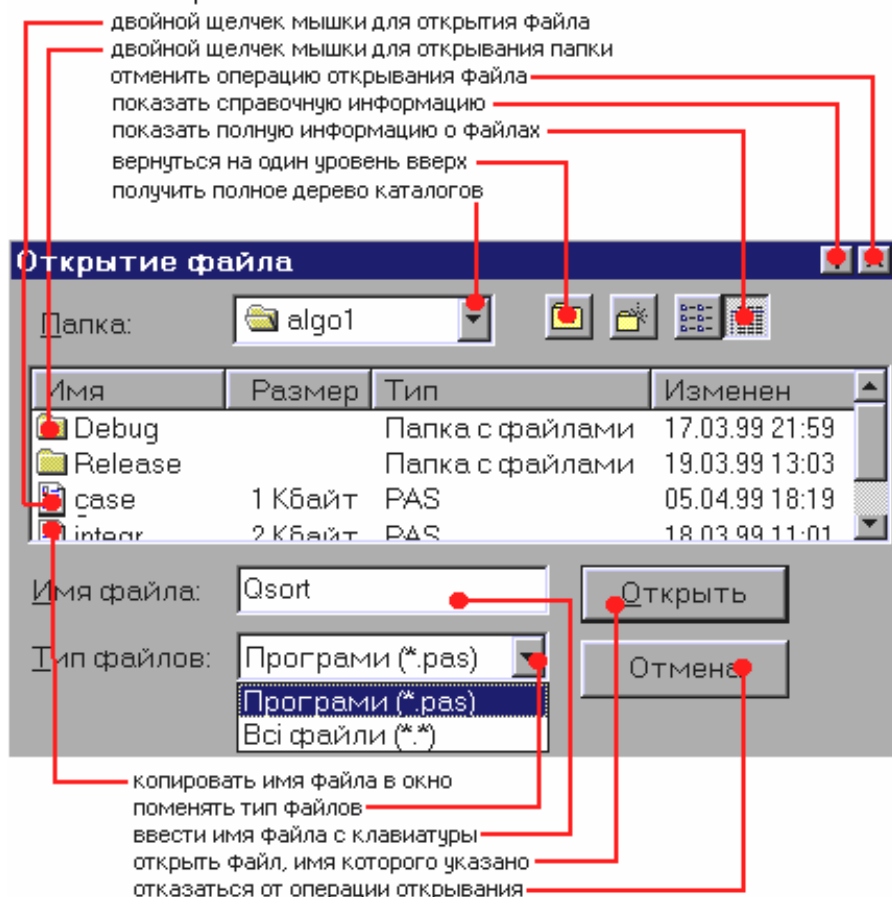



Рис. 2


## Выполнение программы.

Чтобы выполнить программу нужно нажать кнопку  на панели инструментов или выбрать пункт **Команды Выполнить** программу в меню, или нажать клавишу **F9**.

Как не досадно, но в большинстве случаев вместо ожидаемого результата выполнения получаем сообщения об ошибке

в программе. ALGO выделяет красным цветом строку, в которой прекратилась компиляция и устанавливает курсор на месте остановки. Это не всегда означает, что как раз в этой строке допущена ошибка. Например, если курсор установлен в начале выделенной строки, а в тексте сообщения есть слово "пропущен", то это в большинстве случаев означает, что пропущено что-то в предыдущей строке.

При выполнении программы зеленым цветом выделяется строка, которая выполняется, а в окне режима работы указан текущий режим.

Кнопкой  или клавишей F6 всегда можно прекратить выполнение программы.

Кнопками  и  или клавишами F8 и F7 можно выполнять программу пошагово и до курсора соответственно.

## *Структура языка программирования Algo*

### *Пользование оболочкой*

Программная среда Robot как и Algo организует общение с пользователем, т.е. с вами уважаемые учащиеся, используя нарисованное окно с несколькими полями, кнопками, надписями, которые и образуют так называемый интерфейс – диалоговый контакт. Произошло это название от двух слов: inter – между, face – лицо. На первом уроке на рисунках эти поля были указаны.

Для нас, начинающих учиться писать программы нужным будет поле для ввода текста программы. Оно расположено слева вверху и занимает довольно обширную часть окна программы. Вы заметили, что этот подпункт называется «Пользование оболочкой». Так вот оболочкой как раз и называется все это окно, посредством ее мы и будем общаться со средой нашего программного приложения.


Забегая немного вперед, следует отметить, что для того, чтобы выполнить программу, написанную языком Паскаль (Robot), необходима специальная встроенная программа - компилятор, который переведет написанную вами программу в специальный машинный язык - коды компьютера и только после этого ее выполняет. Если компилятор обнаружил ошибку в программе, он немедленно выводит сообщение и прекращает дальше работать, надо ошибку в программе исправить и заново запустить ее на выполнение. Как видите нажатием кнопки «Исполнение



программы» запускается первым компилятор и лишь, если в программе нет ошибок, она выполняется. Но для этого нужен текстовый файл с программой на языке Robot. Чтобы подготовить такой текст и нужно воспользоваться экранным редактором – полем, о котором только что шла речь.

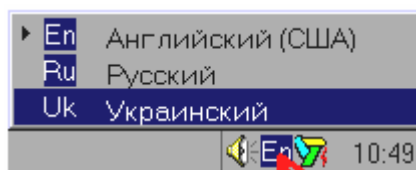
Оболочка выполнена двумя языками - английским и русским. Для перехода на английский язык нужно нажать кнопку



, а на русский - кнопку . Это же можно сделать с клавиатуры нажатием Ctrl+E и Ctrl+U соответственно, или через меню (Опции Английский язык; Опции Русский язык)

Для перехода к меню указывают мышкой на соответствующий его пункт или нажимают клавишу Alt, выбирают нужный пункт клавишами управления курсором, после чего нажимают Enter.

Язык оболочки никак не связан с режимом работы клавиатуры. Для переключения клавиатуры с латинских букв на русские и, наоборот, в большинстве случаев используют правый Ctrl+Shift. Это нужно выяснять в преподавателя или собственника компьютера. Но всегда можно переключить режим работы клавиатуры мышкой с помощью иконки в правом нижнем углу экрана, как показано на рисунке.



Для выбора  
режима работы  
клавиатуры  
нажмите ЗДЕСЬ

## ***Структурный язык программирования***

Одним из первых языков, которые начали использовать в школе после введения предмета Информатика в учебный план, был язык Basic. Это довольно простая среда программирования, не требовала никаких заранее установок и объявлений, никаких ограничений использования значений, хотя конечно были и свои правила. Длинные программы на языке Basic читать было очень трудно, так как они порой были настолько запутаны переходами



и возвратами, что не опытному программисту, а тем более, начинающему или ученику это было не под силу.

Для упорядочивания чтения текстов программ, обращения к отдельным ее частям и были созданы языки, примером которых и является ROBOT (ALGO, PASCAL). По-другому, была разработана определенная последовательность блоков, связанных с основной частью, закреплено за ними место следования – очередность, а также были введены обязательные служебные слова, без которых нельзя обойтись в работоспособной программе. Эти блоки и организовали структуру языка программирования, а сам язык такого строения стали называть структурным.

Обычный разговорный язык состоит из четырех основных элементов: символов, слов, словосочетаний и предложений. Возник разговорный язык из естественных потребностей человека общаться между собой. Разговорный язык имеет две формы: устную, состоящую из звуков, сочетания которых образуют слова, а слова – предложения и письменную – символьную или ее иногда называю текстовую, состоящую из набора букв, который называют алфавитом и знаков препинания. Разговорные языки в литературе названы естественными или национальными, каждый из которых отличается количеством и начертанием букв

ЯЗЫК программирования (его часто называют алгоритмическим языком) содержит подобные элементы, только слова называют элементарными конструкциями, словосочетания - выражениями, предложения - операторами. Символы, элементарные конструкции, выражения и операторы составляют иерархическую структуру, поскольку элементарные конструкции образуются из последовательности символов, выражения - это последовательность элементарных конструкций и символов, а оператор - последовательность выражений, элементарных конструкций и символов.

ОПИСАНИЕ ЯЗЫКА есть описание четырех названных элементов. Описание символов заключается в перечислении допустимых символов языка. Под описанием элементарных

конструкций понимают правила их образования. Описание выражений- это правила образования любых выражений, имеющих смысл в данном языке. Описание операторов состоит из рассмотрения всех типов операторов, допустимых в языке. Описание каждого элемента языка задается его СИНТАКСИСОМ и СЕМАНТИКОЙ. Синтаксические определения устанавливают правила построения элементов языка. Семантика определяет смысл и правила использования тех элементов языка, для которых были даны синтаксические определения.

СИМВОЛЫ языка - это основные неделимые знаки, в терминах которых пишутся все тексты на языке.

ЭЛЕМЕНТАРНЫЕ КОНСТРУКЦИИ -это минимальные единицы языка, имеющие самостоятельный смысл. Они образуются из основных символов языка.

ВЫРАЖЕНИЕ в алгоритмическом языке состоит из элементарных конструкций и символов, оно задает правило вычисления некоторого значения.

ОПЕРАТОР задает полное описание некоторого действия, которое необходимо выполнить. Для описания сложного действия может потребоваться группа операторов. В этом случае операторы объединяются в СОСТАВНОЙ ОПЕРАТОР или БЛОК.

В общем алгоритм структуры текста программы таков:

1. Заголовок
2. Объявления:
  - переменных;
  - процедур и функций;
  - меток.
3. Начало
4. Тело программы
5. Конец

*Заголовок программы*

В своей простейшей форме программа Robot (Algo) состоит из заголовка программы, который именуется программой, и основного программного блока, выполняющего назначение программы. В основном программном блоке находится секция кода, заключенная между ключевыми словами `begin` и `end`. Приведем простейшую программу, иллюстрирующую эти принципы:

```
program Привет;  
begin  
  Writeln('Добро пожаловать в среду программиро-  
вания Algo-Robot');  
end.
```

Первая строка - это заголовок программы, который именуется данной программой. Остальная часть программы - это исходный код, который начинается ключевым словом `begin` и заканчивается `end`.

На русском языке это будет выглядеть так:

```
программа Привет;  
начало  
  вывестистр('Добро пожаловать в среду програм-  
мирования Algo-Robot');  
конец.
```

Хотя данная конкретная программа содержит только одну строку, их может быть много. В любой программе Pascal, в том числе и Algo, все действия выполняются между `begin` и `end`.

Код между последними операторами `begin` и `end` программы управляет логикой программы. В очень простой программе в этой секции кода может содержаться все, что вам нужно. В более крупных и сложных программах размещение в этой секции всего программного кода может затруднить чтение и

понимание программы. К тому же ее будет труднее разрабатывать.

## **Объявления**

Объявление переменных, процедуры и функции, метки позволяют разделить логику программы на более мелкие и управляемые фрагменты и аналогичны подпрограммам в других языках, например, в Basic. Как и в основном блоке программы, все действия в процедурах и функциях заключаются в `begin` и `end`, после последнего только нужно ставить точку с запятой. Каждый из этих сегментов кода выполняет конкретную задачу.

### **✓ Переменные величины, имена**

Переменными величинами называют числа, слова, целые тексты, которые в ходе выполнения программы могут менять свои значения. Обозначаются переменные буквами латинского алфавита. Имя переменной (ее часто называют идентификатором) может иметь любую длину, однако только первые его 16 символов являются значимыми.

Идентификатор должен начинаться с буквы и не может содержать пробелов. После первого символа идентификатора можно использовать буквы, цифры и символы подчеркивания. Как и в зарезервированных словах, в идентификаторах можно использовать как строчные, так и прописные буквы. Имя не должно содержать пробелов. Не разрешается в языке Algo использовать в качестве имен служебные слова и стандартные имена, которыми названы стандартные константы, типы, процедуры, функции и файлы.

Примеры имен языка Algo:

`A b12 r1m SIGMA gamma I80_86 ... G, alfa,`  
`test17, x2y, Сума2Чисел, конец_массива.`

В математике принято классифицировать переменные в соответствии с некоторыми важными характеристиками. Производится строгое разграничение между вещественными, целыми и логическими переменными, между переменными, представляющими отдельные значения и множество значений и так далее.

При обработке данных на ЭВМ такая классификация еще более важна. В любом алгоритмическом языке каждая переменная, выражение или функция бывают определенного типа. В языке Algo существует правило: тип явно задается в описании переменной, которое предшествует их использованию. Например:

**Var i, f, : integer**

На русском языке:

**Переменные i, f, : целые;**

Любая переменная наряду с именем обязательно должна иметь и значение. Запомните, пожалуйста, следующую схему:



При описании переменной, как указывалось раньше, необходимо указать ее тип. Тип переменной определяет набор значений, которые она может принимать, форму записи их в памяти и действия, которые могут быть над ней выполнены. Типы разделяются на простые и сложные. Переменная простого типа всегда имеет одно значение (число, символ и т.п.), переменная сложного типа имеет таблицу значений одинакового типа (мас-

сив) или набор полей различного типа (запись). К простым типам в ALGO относятся:

- логический тип `boolean` - логические;
- символьный тип `char` - символы;
- целый тип `integer` - целые;
- действительный тип `real` - действительные.

По мере изучения языка программирования мы постепенно познакомимся со всеми типами переменных.

## **✓ *Функции и процедуры***

О функциях и процедурах, как и о типах переменных, сейчас, на первых уроках, вести разговор нет необходимости. К ним мы еще вернемся, когда будем изучать и решать задачи с этими данными.

## **✓ *Метки***

В этом разделе оглашаются метки, которые определяются операторам перехода. Каждая метка может использоваться не один раз, а количество их в программе зависит от конкретной задачи и надобности в них. Все метки перед употреблением должны быть объявлены с помощью такой записи:

**Label 1, 2,...N;**  
**Метки 1, 2,...N;**

Метка есть последовательность цифр, то есть целое число без знака, которое находится в диапазоне от 0 до 9999. Нули перед числами игнорируются. Организация перехода организуется так:

```
program Привет;  
  label 1;  
  begin  
    1;;
```

```
Writeln('Добро пожаловать в среду программи-  
рования Algo-Robot');  
goto 1;  
end.
```

На русском языке эта программа выглядеть будет так:

```
программа Привет;  
метки 1;  
начало  
1;;  
вывестистр('Добро пожаловать в среду про-  
граммирования Algo-Robot');  
перейти 1;  
конец.
```

Если исполнить эту программа, то экран монитора будет заполняться выражением «Добро пожаловать в среду программирования Algo-Robot» до тех пор, пока не будет нажата кнопка Stop.

### ***Начало, тело и конец программы***

Начало состоит из одного служебного слова begin (начало), конец аналогично – из одного end. (конец.)

Все операторы, команды и т.д. записываются между этими двумя служебными словами, и называется этот блок телом программы.

### § 3

#### Следование

В системе «алгоритмики» основными конструкциями исторически принято считать всего три, это: следование, циклы, ветвления.

Конструкция – это связанные логически между собою набор служебных слов (команд) с обязательными параметрами (операндами), имеющая завершение закрепленных за ней действий.

Например: *Следование* – последовательное, друг за другом, выполнение команд; *цикл* – повторение одних и тех же команд; *ветвление* – выбор одного из двух действий или команды.

В данном разделе повторения мы остановимся на самой простой и по смыслу и графическому представлению конструкции – *следование*. Так как этот материал вам уже знаком, остается только напомнить следующие основные, важные моменты:

#### Блок-схема следования

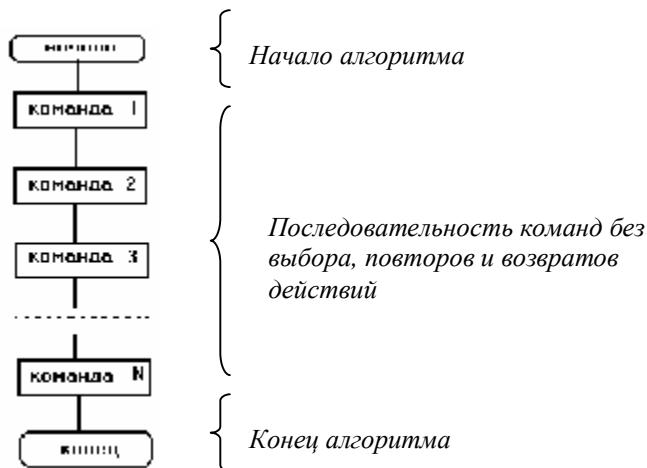


Рис. 3



Примеры записи текста алгоритма на языке программирования в среде ROBOT с показом его исполнения (Рис. 4).

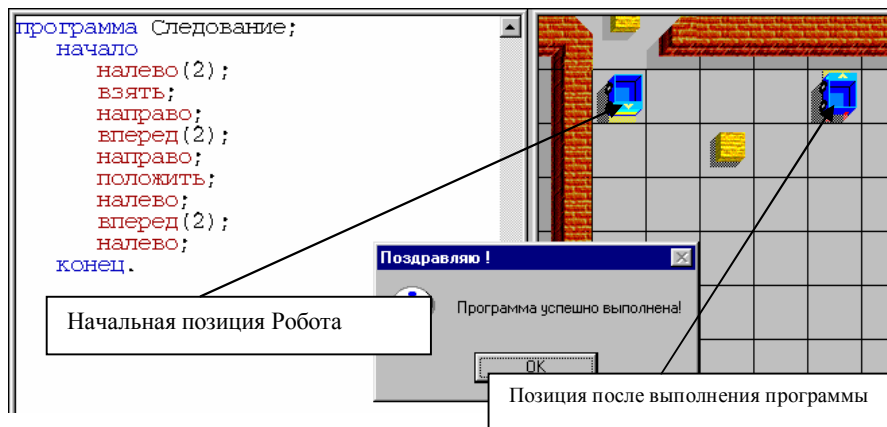


Рис. 4

Эта же программа на английском языке будет выглядеть так:

```

program Следование;
begin
  left(2);
  take;
  right;
  forward(2);
  right;
  put;
  left;
  forward(2);
  left;
end.

```

### Задание 1

Составить программу решения выражения:

$$(24 + 37) \cdot 4 + 12 \cdot (67 - 31 - 30)$$

## Математическая модель

Начнем с составления плана решения. Такой процесс называется в информатике математическим моделированием. Значит, мы составим нечто иное, как математическую модель.

- 1) Определим порядок действий так, чтобы можно было эти действия объединить в отдельные блоки.
- 2) Эти блоки могут содержать несколько действий подряд выполняемых.
- 3) В данном выражении таких блоков три:

$$\underbrace{(24 + 37) \cdot 4}_{1\text{-й}} + \underbrace{12 \cdot (67 - 31 - 30)}_{2\text{-й}}$$

3-й

- 4) Обозначим 1-й блок буквой  $x$ , второй –  $y$ , третий –  $z$ .
- 5) То есть, вначале выполним сложение и умножение и в памяти ПК запоем под именем  $x$ , затем выполним действия вычитания и умножения – запоем под именем  $y$  и наконец, выполним действие сложение, которое будет тоже запоем под именем  $z$  ( $z=x+y$ ).
- 6) Результат нужно вывести на экран.

*Составление программы в редакторе ALGO.*

**программа Следование;**

**переменные  $x,y,z$ : целые;**

**начало**

**$x:=(24+37)*4;$**

**$y:=12*(67-31-30);$**

**$z:=x+y;$**

**вывестистр('Результат выражения =' , $z$ );**

**конец.**

*На английском языке программа выглядит так:*

**Program Следование;**

**Var  $x,y,z$ : integer;**

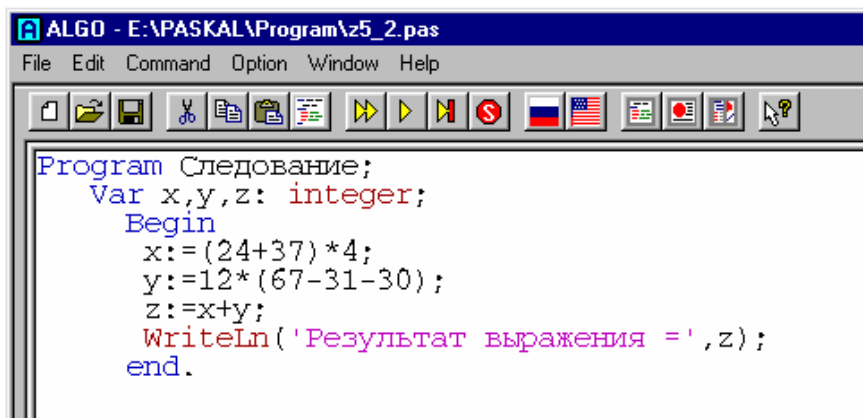
**Begin**

```

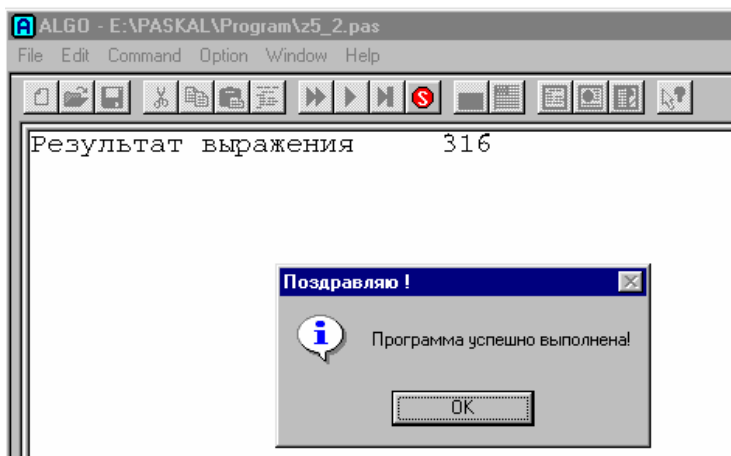
x:=(24+37)*4;
y:=12*(67-31-30);
z:=x+y;
WriteLn('Результат выражения =',z);
End.

```

Экран ALGO представлен на *Рис.5* и *Рис. 6*



*Рис. 5*



*Рис. 6*

## ***Задание 2***

Дано равенство в неявном виде:

$$y = 2 \cdot (x + 4) \cdot (x + 4) + 3 \cdot (z - 5) + c$$

Найти значение  $y$ , если  $x=7$ ,  $z=11$  и  $c=13$ . Написать программу, предварительно составив математическую модель.

## § 4

### *Ветвление*

В условиях, когда наши программы становятся все сложнее, тексты их длиннее и структура все полнее, для детального понимания блоков и конструкций, а также назначения некоторых данных добросовестный программист, отдающий себе отчет, что программа пишется для другого, может быть не вполне владеющего основами программирования, включает в текст программы комментарии. Что это такое и как они вводятся?

#### ♦ *Пояснения к программе (комментарии).*

Любой текст, взятый в фигурные скобки или ограниченный парами символов (\* \*) является комментарием и игнорируется компилятором или:

{любой текст, не содержащий правую фигурную скобку}  
(\*любой текст, не содержащий звездочку и правую круглую скобку\*).

Комментарии в ALGO выделяются зеленым цветом. При написании программ, работа которых не является очевидной, желательно писать комментарии, чтобы иной программист мог понять их.

Например, фрагмент программы с комментариями выглядит так:

```
Program калькулятор;  
Const  
  N=15; { количество кнопок }  
  светлый = 255; { интенсивность светлого  
цвета }  
  темный = 128; { интенсивность темного цвета }  
Var коорд : array [0..N,1..4] of integer;  
{координаты кнопок}  
  Данные : text; {текстовый файл данных}
```

**i,j,k : integer; {рабочие переменные для цик-  
лов}**

**число,предыдущее,операция: integer;  
надпись : char; {обозначение кнопок}**

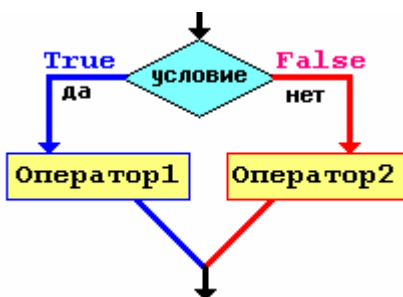
### ♦ Условный оператор

Полный условный оператор позволяет выбрать для выполнения один из двух операторов.

**If выражение then оператор1  
else оператор2**

**если выражение то оператор1  
иначе оператор2**

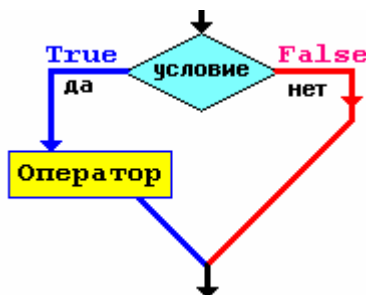
Блок-схема его приводится на рисунке ниже



Неполный условный оператор выполняет внутренний оператор только в том случае, когда значение выражения есть истинным (при выполнении условия).

**If выражение then опе-  
ратор**

**если выражение то  
оператор**



В выражении должен образовываться результат, что имеет стандартный логический тип. Если результатом выражения есть истинное значение (true), то выполняется оператор, который следует за ключевым словом then. Если результатом выражения есть ложное значение false и присутствующее ключевое слово else, то выполняется оператор, который следует за ключевым словом else, а если оно отсутствует, то никакой оператор не выполняется.

В обеих частях условного оператора можно употреблять лишь по одному оператору. При необходимости поместить группу операторов, нужно образовать из них составной оператор.

Синтаксическая неоднозначность, которая возникает в конструкции:

**if e1 then if e2 then s1else s2**

решается путем такой интерпретации этой конструкции:

```
if e1 then  
    begin  
        if e2 then  
            s1  
        else  
            s2  
    end;
```

В общем случае ключевое слово else связывается с самым близким ключевым словом if, которое еще не связано с ключевым словом else.

Приведем пример оператора if:

```
if X < 1.5 then  
    Z:= X+Y  
else  
    Z:= 1.5;
```

### ◆ *Оператор выбора.*

Оператор выбора (варианта) состоит из выражения (переключателя) и списка операторов, каждому из которых предшествует список констант, которые называются константами выбора, или ключевое слово *else*. В каждом списке есть хотя бы одна константа. Переключатель должен иметь порядковый тип. Все константы выбора должны быть уникальными (встречаться лишь один раз в данном операторе) и иметь тип, совместимый с типом переключателя.

**Case выражение of**  
**K11,...,K1M : оператор1;**  
**.....**  
**KN1,...,KNL : операторN;**  
**else операторN1;...операторNk**  
**end.**

**Выбор выражение из**  
**K11,...,K1M : оператор1;**  
**.....**  
**KN1,...,KNL : операторN;**  
**иначе операторN1;...операторNk**  
**конец.**

В каждом варианте записывают лишь один оператор, который может быть составным оператором. Оператор завершается точкой с запятой. Часть оператора, начинающаяся словом *else*, может отсутствовать.

Оператор выбора выполняет оператора, перед которым есть константа выбора, равная значению переключателя в момент выполнения оператора. Если такой константы выбора нет и есть слово *else*, то выполняется оператор, который следует за ключевым словом *else*. Если же ветвь *else* отсутствует и константы выбора нет, то не выполняется никакой оператор.



### **Пример оператора выбора**

Пусть при тестировании ученик получил N баллов из 20 возможных. Нужно вывести сумму баллов с коротким комментарием. Сделаем это с помощью оператора case (z5\_3.pas).

```
Program Выбрать;  
  Var N : integer;  
  Begin  
      WriteLn('Введите число баллов полученных Вами при тестировании');  
      ReadLn(N);  
      Case N of  
        20: WriteLn('Лучше не бывает!')  
          19,18,17 : WriteLn('Отлично!')  
          16,15,14,13 : WriteLn('Хорошо.')  
          12,11,10,9 :  
            WriteLn('Удовлетворительно.')  
            8,7 : WriteLn('Еще немного, и все было бы в порядке.')  
            else WriteLn('Если ничего не знаете, то хоть бы что-то угадали!')  
      End;  
      WriteLn('Сума баллов - ',N:2,' из 20 возможных');  
  End.
```

### **Задание 1**

Используя оператор выбора напишите программу, которая по номеру цвета выводит его названия. Все данные помещены в таблице ниже.

Номер цвета	0	1	2	3	4	5	6	7
Цвет	Черный	Синий	Коричневый	Красный	Зеленый	Сиреневый	Желтый	Белый

## ***Задание 2***

Используя конструкцию ветвления, как составной оператор, напишите программу, которая по цвету светофора определяет действия человека, стоящего у пешеходного перехода.

## Циклы

### *Некоторые дополнения использования типами данных*

Решая математические задачи, вернее составляя программы их решения, часто приходится использовать данные целого и действительного типа одновременно. Поэтому целесообразно будет внести некоторые разъяснения по их использованию. Так, если оба операнда в операциях **+**, **-**, **\***, **div** или **mod** есть операндами целого типа, то тип результата будет целым. Если один или больше операндов в операциях **+**, **-**, или **\*** имеют действительный тип, то тип результата будет действительным. Значение выражения **x/v** всегда будет действительного типа, независимо от типов операндов. Если **v** равно 0, то будет выведено сообщение об ошибке. Значение выражение **i div j** представляет собой *целую часть* от **i/j**, округленную в меньшую сторону к значению целого типа. Если **j** равно 0, то будет выведено сообщение об ошибке. Операция **mod** возвращает *остаток* от *деления* двух ее операндов, то есть:

$$i \bmod j = i - (i \operatorname{div} j) * j$$

Знак результата операции **mod** будет тем же, что и знак **i**, если **j** равно нулю, то результатом будет ошибка.

Результаты логических операций отвечают обычной булевой логике. Например, выражение **a and b** есть истинным, принимает значение **true (истина)** только в том случае, если оба операнда **a** и **b** имеют истинное значение (**true**).

При полном вычислении подразумевается, что каждый операнд логического выражения, построенный с помощью операций **or (или)** и **and (и)**, всегда будет вычисляться, даже если результат всего выражения уже известный.

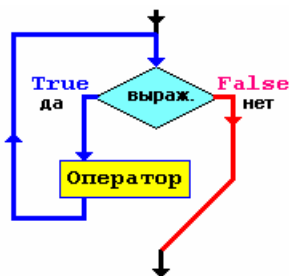
Об алгебре логики вам будет подробно изучено в старших классах. В наших задачах эти логические операции мы будем применять крайне редко.

### ♦ *Цикл с предусловием*

Оператор цикла с предусловием содержит в себе логическое выражение, которое управляет повторным выполнением оператора (который может быть составным оператором).

**While выражение do  
оператор**

**Пока выражение выполнять  
оператор**



Выражение, с помощью которого осуществляется управление повторением оператора, должно иметь логический тип. Вычисление его проводится до того, как внутренний оператор будет выполнен. Внутренний оператор выполняется повторно до тех пор, пока выражение не примет значение false (ложь). Если выражение с самого начала принимает значение false (ложь), то оператор, что находится внутри оператора цикла с предусловием, не выполняется.

Пример оператора цикла с предусловием:

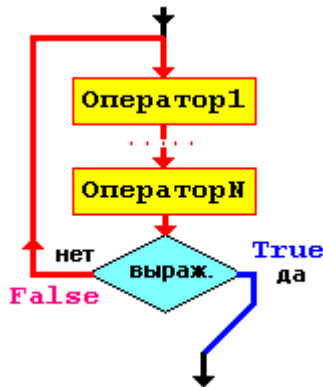
**while Data[I] <> X do I := I + 1;**

### ♦ Цикл с послеусловием

Оператор цикла с послеусловием записывают в общем виде так:

**Repeat**  
    **оператор1;**  
    .....  
    **операторN**  
**until выражение**

**Повторять**  
    **оператор1;**  
    .....  
    **операторN**  
**пока Не выражение**



Операторов может быть произвольное количество, или не быть совсем. Результат выражения должен иметь логический тип. Операторы, помещенные между ключевыми словами **repeat** (повторять) и **until** (пока не), выполняются последовательно до тех пор, пока результат выражения не примет значение **true** (истина). Последовательность операторов выполнится, по крайней мере, один раз, поскольку вычисление выражения проводится после выполнения последовательности операторов.

Приведем пример оператора цикла с послеусловием:

```

repeat
    K := I mod J;
    I := J;
    J := K;
until J = 0;

```

♦ *Цикл с параметром*

Оператор цикла с параметром вызывает повторяемое выполнение оператора (что может быть составным оператором) для всех значений управляющей переменной в границах заданного диапазона.

**For** переменная := НачЗн **to** КонЗн **do**  
Оператор

Для переменная:= НачЗн **к** КонЗн **выпол-**  
**нять**

Оператор



В качестве управляющей переменной должен использоваться идентификатор, обозначающий переменную, объявленную локально в блоке, в котором находится оператор **for (для)**. Управляющая переменная должна иметь простой порядковый тип. Начальное и конечное значение должны иметь тип, совместимый по присваиванию с порядковым типом.

Когда начинает выполняться оператор **for (для)**, начальное и конечное значения определяются один раз, и эти значения хранятся на протяжении всего выполнения оператора **for (для)**. Оператор, находящийся в теле оператора **for (для)**, выполняется один раз для каждого значения в диапазоне между начальным и конечным значением. Управляющая переменная всегда инициализируется начальным значением. При каждом следующем повторении оператора **for (для)** берется следующее значение управляющей переменной. Если начальное значение превышает конечное значение, то оператор, находящийся в теле оператора **for (для)**, не выполняется. Когда в операторе цикла вместо слова **do (выполнять)** используется ключевое слово **downto (обратно)**, значения управляющей переменной изменяются в обратном порядке. Если начальное значение в этом случае меньше, чем конечное значение, то оператор не выполняется.

Оператор, находящийся в теле оператора **for**, не должен изменять значения управляющей переменной. После выполнения оператора **for** значение управляющей переменной становится неопределенным, если только выполнение оператора **for** не было прервано с помощью оператора перехода.

Приведем примеры оператора цикла с параметром:

- 1)        **for sym := 'z' downto 'o' do**  
          **if str[i] = sym then goto 25;**
  
- 2)        **for I := 1 to 10 do**  
          **for J := 1 to 10 do**  
              **begin**  
                  **X := 0;**  
                  **for K := 1 to 10 do**

<b>Mat1[I,K]*Mat2[K,J];</b>	<b>X:=X+</b>
<b>end;</b>	<b>Mat[I,J] := X;</b>

♦ ***Оператор перехода.***

Оператор перехода вызывает передачу управления оператору, которому предшествует метка, указанная в данном операторе перехода. Оператора перехода имеет такой вид:

**GoTo метка**  
**Перейти метка**

Метка есть последовательность цифр в диапазоне от 0 до 9999. Начальные нули не являются значащими. Следующим после оператора перехода будет выполнен оператор, обозначенный указанной меткой.

При использовании оператора перехода должны соблюдаться такие правила:

- 1 Метка, которая указывается в операторе перехода, должна находиться в том же модуле, что и сам оператор перехода. Иными словами, не допускаются переходы с процедуры или функции наружу или извне внутрь ее.
- 2 Переход извне внутрь структурного оператора (то есть переход на более глубокий уровень вложенности) может вызывать непредвиденные эффекты, хотя компилятор не выдает сообщение об ошибке. Например, переход внутрь оператора цикла с параметром приводит к выполнению части тела цикла при неизвестных значениях параметра, начального и конечного его значений.

Частое употребление оператора перехода делает программу запутанной, поэтому его не рекомендуют использовать.

Пример:



В случае, когда нужно немедленно завершить выполнение процедуры при  $X < 0$ , можно воспользоваться переходом на пустой оператор, что есть последним в процедуре.

**Procedure Test;**

.....

**Label 11;**

**Begin**

.....

**If  $X < 0$  then GoTo 11;**

.....

**11 : ;**

**end;**

### ***Задание 1***

По тексту данной программы ниже определите: какие конструкции здесь используются, какую эту программа решает задачу. При запуске программы будет выдана ошибка, определите ее и отредактируйте текст.

(z5\_4.pas)

**program summa;**

**var sum,n,i:integer;**

**begin**

**sum:=0;**

**writeln('Введи последнее число от 1');**

**read(n);**

**for i:=1 to n do**

**begin**

**sum:=sum+i**

**end;**

**writeln('сумма этих чисел равна ',sum)**

**end.**

### ***Задание 2***

При забивании гвоздя  $n$  см длинны с каждым ударом гвоздь входит в дерево на 1 см. Написать программу, которая подсчитает количество ударов при забивании гвоздя, длину  $n$  которого вы должны указать по запросу программы. Длина гвоздя не должна быть меньше чем 1 см.

(z5\_5.pas)

### ***Задание 3***

Одному ученику для развития памяти была предложена такая тренировочная работа, называя четные числа на любом последовательном промежутке, одновременно сосчитать их количество. Вернувшись со школы, он быстро составил программу на своем компьютере, тем самым избавил себя от изнурительной работы. Постарайтесь и вы составить такую программу.

(z5\_6.pas)

## ГРАФИКА В СРЕДЕ ПРОГРАММИРОВАНИЯ ALGO

Рисунок всегда, во все времена занимал и занимает очень важное место в жизни человека. Древние племена, не имевшие письменности, рассказали нам о своей жизни при помощи на-кальных рисунков. Рисунок – это не только художественное произведение. Созданное талантливым художником. Это может быть и простая картинка, схема, набросок, чертеж, обозначающие что-то.

Появившийся на свет младенец не умеет ни говорить, ни ходить. Он еще очень беспомощен. Но уже с рождения ему подарена удивительная способность – видеть мир своими глазами. Пройдет не мало времени, прежде чем этот малыш сможет выразить свои мысли и желания словами и далек тот день, когда он напишет первое слово. Но получив в руки карандаш, он неумело, но уже настойчиво пытается рисовать мир. Эти рисунки – одно из средств самовыражения маленького человека.

За длительное свое существование человек придумал много инструментов для рисования. Это: цветные карандаши, фломастеры, краски, кисти. Ластик – помощник для стирания лишних штрихов. Есть специальные баллончики с краской – распылители. Существуют наборы чертежных инструментов. И многое, многое другое, позволяющее создавать прекрасные рисунки.

Рисовать можно на любых плоских или кривых поверхностях, картоне, материале. Делать роспись на посуде, стенах квартир, одежде и т.д. Такие работы называли графическими, от слово графит – вещество используемое для производства карандашей и способное оставлять след на поверхности (бумаге, картоне, полотне и т.д.). Сам жанр искусства был естественно назван ГРАФИКОЙ.

Но вот настало время массовой компьютеризации (конец прошлого века и начало нынешнего) и человек быстро создал массу замечательных программ, способных создавать не хуже,

если не лучше, замечательные рисунки с помощью компьютера. При всем этом еще иметь и массу удобств.

Такие программы называются Графическими редакторами, а изображения созданные с помощью компьютера - Машинной (Компьютерной) графикой

Чем же так удобны компьютерные рисунки? При работе в графическом редакторе карандаш никогда не ломается, ластик не стирается, а краски не кончаются. При этом пальцы, нос и одежда остаются чистыми. Если рисунок на бумаге не удался, остается одно – выбросить его в корзину и начать все заново. Зато компьютерный рисунок можно исправлять до тех пор. Пока вы не останетесь довольны своей работой. И еще одно из множества удобств – это то, что рисунок можно сохранить, распечатать на принтере, копировать дарить друзьям, близким, пересылать по электронной почте во все концы мира, редактировать (добавлять объекты, надписи и др.).

Все это вы уже слышали от учителя в 3-м классе, когда осваивали графический редактор **Paint**.

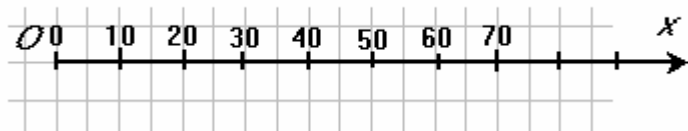
В настоящее время есть и другие способы и средства создания рисунков: цифровое фото – цифровая фото или видео камеры, сканированный рисунок – сканер. Эти средства позволяют почти не прикасаться к электронным карандашам, кистям и др. инструментам. Однако такие изображения тоже относятся к компьютерной графике.

И наконец – программные средства. Это языки программирования, алфавит. И набор функций и процедур которых позволяют создавать компьютерный рисунок. Вот такой небольшой экскурсией от прошлого до настоящего, мы подошли к нашей теме «Графика в среде программирования Algo».

Но прежде чем приступить к освоению способов программирования графических объектов, нам необходимо рассмотреть некоторые математические понятия, которые заложены в компьютерном графическом моделировании.

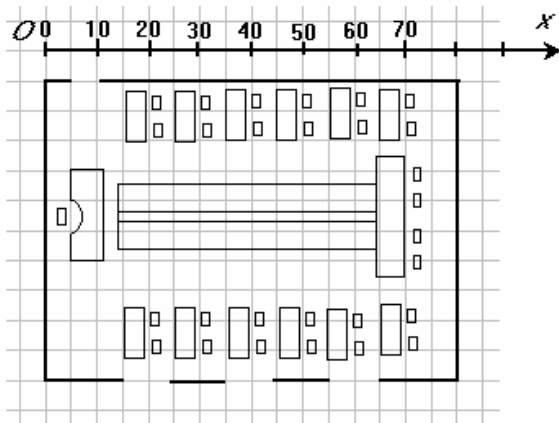
## ***Прямоугольная система координат. Координаты точки на плоскости.***

Из курса начальной школы вы уже знаете что такое координатный луч. Не будем давать ему определение, а сошлемся на рисунок, на мой (автора) взгляд, дающий полное представление об этом понятии.



*Рис. 5*

На данном рисунке единичный отрезок составляет две клеточки и равен 10. Что такое 10? Эта единица длины какой-то меры может быть 10 миллиметров, 10 километров. Все это зависит какую мы решаем с вами задачу. Если это поверхность пола нашего кабинета то за 10 можно принять единицу измерения дециметр и тогда одну из сторон пола, а он прямоугольный, можно измерить и изобразить с использованием нашего координатного луча (Рис. 6).



*Рис. 6 Измерение размеров кабинета Информатики с помощью нашей координатной оси*

Однако ответить на вопрос как размещена мебель, указать точный «адрес» ее в кабинете этим способом нам не удастся.

Для этого мы введем дополнительный координатный луч с такими же компонентами (Рис. 7).

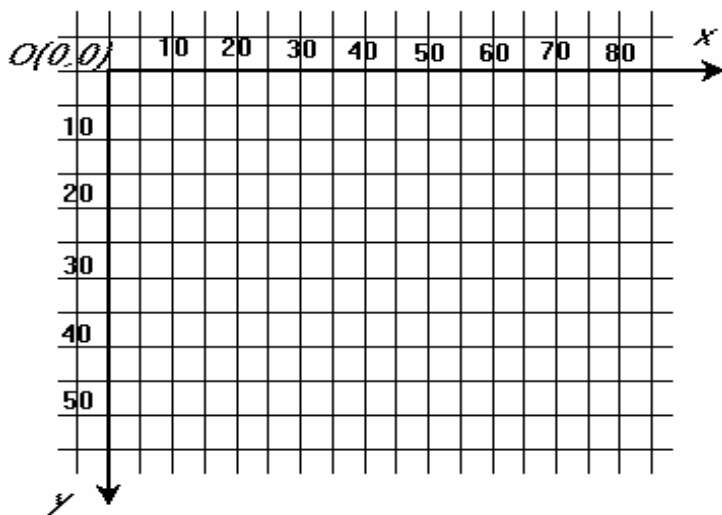


Рис. 7

Как видно из этого рисунка эти оба координатных луча расположены относительно друг друга под прямым,  $90^0$  углом и выходят из одной и той же точки, которая обозначена буквой *O*.

Уместно напомнить, что две прямые линии или отрезка могут образовывать множество различных углов (Рис. 8).

Обратите внимание на их величину, которую можно измерять специальным прибором – *транспортиром* (Рис. 9).

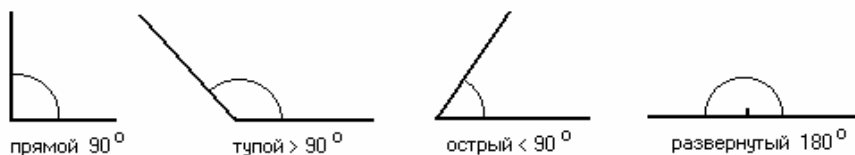


Рис. 8

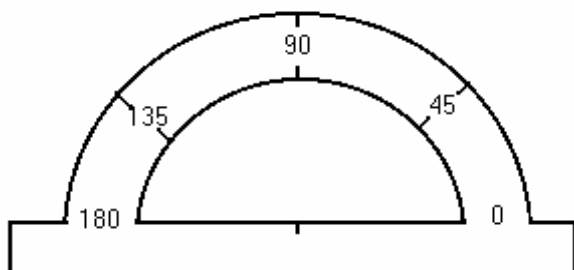


Рис. 9 Транспортир

Это нам понадобится при определении и построении фигур на координатной плоскости.

Вообще говоря, любые два пересекающиеся луча плоскость делят на четыре части. Значит, два луча исходящие из одной точки образуют только четвертую часть или, как говорят в математике, четверть. Но для краткости мы будем часть плоскости образованной двумя лучами, исходящими из одной точки под углом  $90^\circ$ , называть *координатной плоскостью*.

Горизонтальный координатный луч в такой системе называют осью и обозначают  $Ox$ , а координатный вертикальный луч называют – осью  $Oy$ .

Кроме этого названия, ось  $Ox$  называется осью *абсцисс*, а ось  $Oy$  – осью *ординат*. Все значения по осям называются *координатами*, соответственно *абсциссами* и *ординатами*.

Все фигуры на плоскости занимают какую-то ее площадь. Из ранее вами изученного площадь фигуры можно найти умножением ее двух измерений. У прямоугольника  $S = a \times b$ , у

квадрата -  $S = a \times a$ . Аналогично площадь и других фигур определяется по их формулам, но все они имеют только два измерения. В старших классах вы эти формулы изучите, научитесь определять площадь. Наша задача сводится только к построению их в системе *координат*.

Итак, *система координат – это часть плоскости образованная двумя пересекающимися координатными лучами (в нашем случае - исходящими из одной точки)*. Название она имеет по буквам, определяющим ось абсцисс, начало координат и ось ординат, т.е. –  $xOy$ .

Всеми этими рассуждениями о введенными новыми понятиями, мы подготовились научиться определять местоположение точки на плоскости, указывать адреса вершин фигур, а значит строить на плоскости. Для этого следует хорошо уяснить несколько правил:

- 1) **Правило.** Координаты точки обозначаются заглавными буквами латинского алфавита или цифрами.
- 2) **Правило.** Рядом с именем точки записываются ее координаты – числа, первое число абсциссы, второе – ордината. Например:  $A(30,60)$ ;  $1(45,20)$  и т.д.
- 3) **Правило.** Чтобы построить точку  $A(30,60)$ , нужно на оси абсцисс найти число 30 и тонкой штриховой линией провести вниз до пересечения другой штриховой линии. Проведенной из точки 60 горизонтально. На пересечении этих линий и находится данная точка (Рис. 10).
- 4) **Правило.** Нахождение координат точки на плоскости проводится в обратном порядке. От точки, вертикально и горизонтально проводят штриховые линии до пересечения с осями, читают числа, которые и есть координаты точки.
- 5) **Правило.** Построение фигур сводится к построению точек – вершин данной фигуры по их координатам. А затем, последовательно соединяя полученные точки, получают искомую фигуру (Рис. 11).



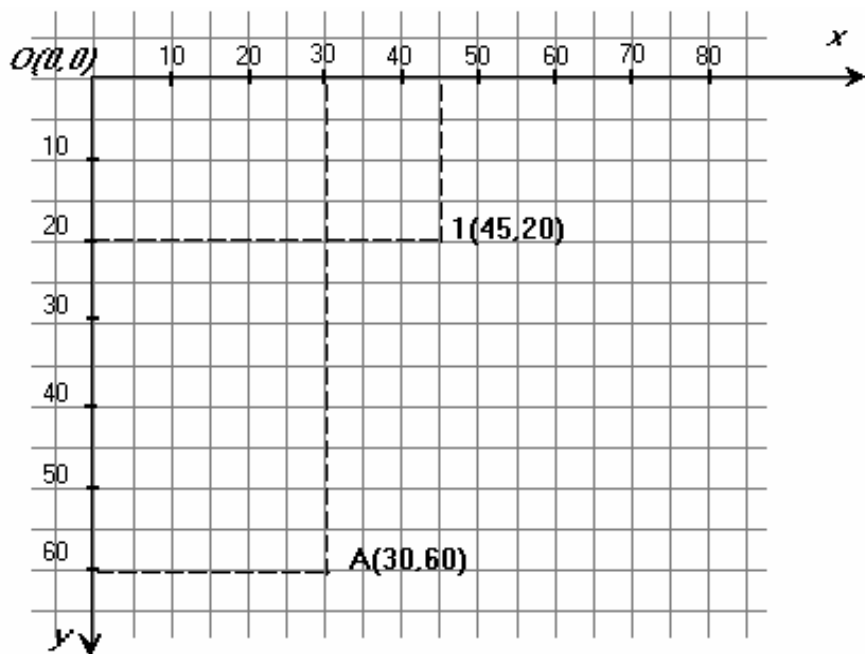


Рис. 10 Построение точек на координатной плоскости

- 6) **Правило.** В записи первой всегда записывается координата абсциссы, второй – ордината –  $B(x,y)$ , где  $x$  – абсцисса, т.е. число на оси  $Ox$ ,  $y$  – ордината, число на оси  $Oy$ .

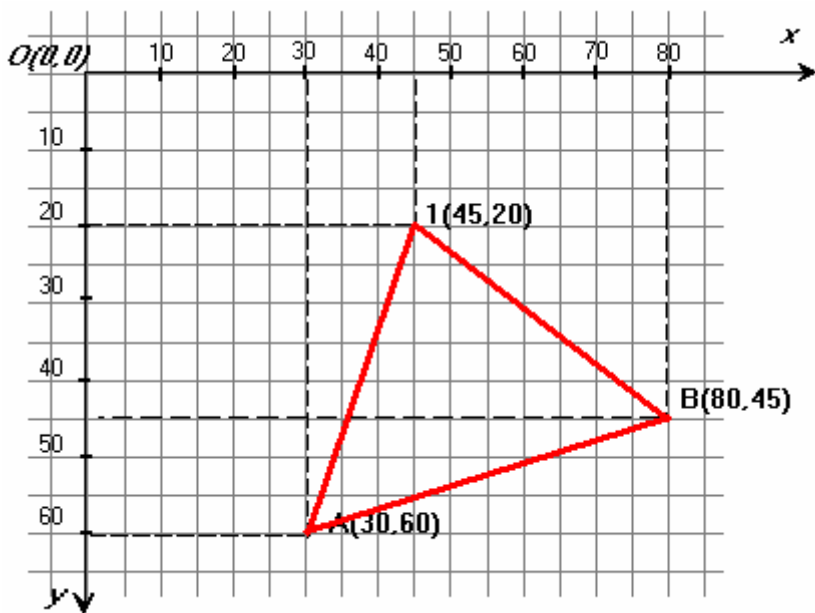


Рис. 11. Построение фигуры «Треугольник» по точкам (А-І-В-А)

### Задание 1

Построить систему координат. Единичный отрезок – 2 клетки и равен 10 единиц. В построенной системе координат построить точки А(20,80), В(50,20), С(80,80) по всем изученным правилам.

### Задание 2

Дополнить Задание 1 следующими построениями: соединить точки по схеме А-В-С-А. Назвать построенную фигуру. Вписать такую же фигуру так, чтобы стороны ее были на единичный отрезок удалены и параллельны сторонам первой. Определить и записать координаты вершин полученной второй фигуры.

### ***Задание 3***

Подготовить систему координат с единичным отрезком в две клетки 30 единиц длина. По указанию учителя построить 4 точки, определить их координаты и назвать полученную фигуру.

## § 7

### *Графический экран*

Для отображения информации в среде ALGO выделено рабочее поле размером 800 точек по горизонтали и 800 точек по вертикали. Часть этого поля видна в окне вывода. С помощью вертикального и горизонтального ползунков можно просматривать весь графический экран.

Точки экрана, как вы уже знаете, называются пикселями и их размер настолько мал, что невооруженным глазом их видеть довольно сложно. Последовательность точек не имеет между собой промежутков (дырок). Отсюда можно заключить следующее: если построить несколько точек подряд, то на экране будет видна линия, а если ими заполнить часть экрана сплошную – получим заполненную однородную или цветную (без дырок) область.

Всего на графическом экране среды Algo 800x800 точек, т.е. 640 000 штук. Запись (800x800) называется разрешением экрана. Чем точек больше, тем качество изображения лучше, а значит разрешение выше. В информатике приняты несколько стандартных видов разрешения и цветности

#### *Разрешения:*

640x320 – низкое;

800x600 – среднее;

1024x768 – высокое;

#### *Количество цветов:*

16 – низкое качество;

256 – среднее;

16 бит (65536) – High Color;

24 бит (16777216) – True Color.

В среде Algo достаточно низкая и разрешаемость экрана и количество цветов определяется комбинацией из трех основных: красного (**red**), зеленого (**green**) и синего (**blue**). Сокра-

ценно такую комбинацию называют **RGB**. Далее мы встретимся с этими цветами, когда речь пойдет о программировании цветов линий и заполнения фигур.

Как видно из Рис.12, началом координат считают верхний левый угол экрана. Ось X направлена слева направо, Y - сверху вниз. Для всех графических функций координаты задают в форме (X,Y).

Напоминание. Первой в записи обязательно должны быть координата абсцисса, второй – ордината. В противном случае можно получить изображение совершенно не по условию задачи и более того, выходящее за пределы экрана.

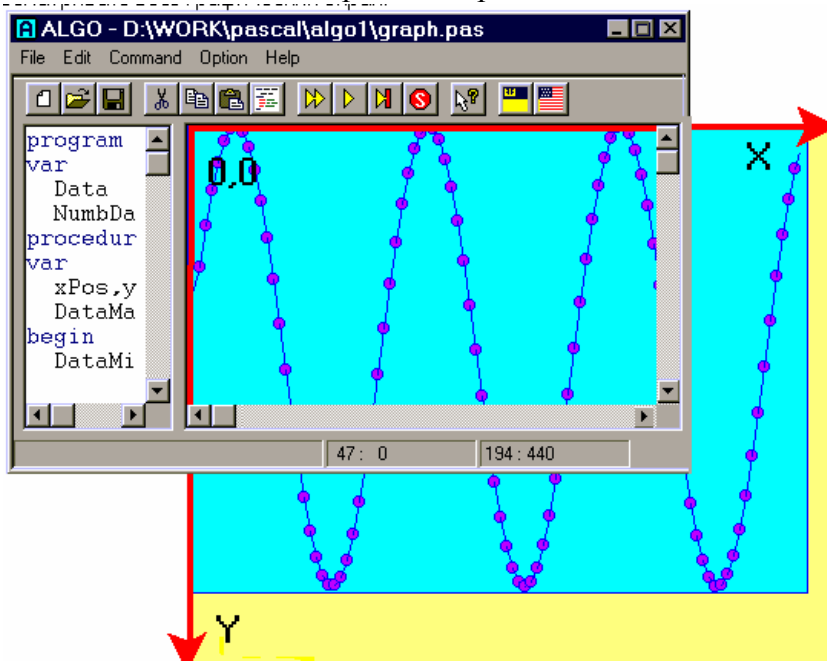


Рис. 12

Правда, если координаты при построении изображения выходят за границы графического экрана, то информация вне границ экрана игнорируется. Исключение составляют операторы **readln** и **writeln**. При попытке вывода за нижнюю границу экра-

на все изображение на экране двигается вверх на величину высоты символа.

### ◆ *Текущие координаты и цвета.*

При выводе текстовой или графической информации на экран необходимо указать, куда именно выводить: сверху, посередине или внизу, а также цвет текста, линий и заполнения. Указывать эту информацию в каждом операторе было бы очень неудобно. Поэтому в среде Algo заложены функции запоминать значения координат точки, в которой завершился вывод информации и текущие цвета. Эти значения называют активными. То есть, если продолжать построения, то отсчет вывода линий и точек начинается от последней построенной. При запуске программы устанавливаются нулевые координаты (верхний левый угол экрана) и черный цвет линий и текста. Цвет заполнения замкнутых фигур (цвет кисти) выбирается прозрачный, то есть замкнутые фигуры не закрашиваются.

В процессе работы программы активные координаты меняются так, чтобы следующий вывод осуществлялся там, где закончился предыдущий.

### ◆ *Построение точек.*

Чтобы поставить (нарисовать) точку в позиции с координатами  $x, y$ , надо обратиться к процедуре **Point**, которая объявляется следующим образом

**procedure Point(  $x, y$  : integer);**  
**процедура Точка(  $x, y$  : целые);**

Фактическими параметрами при обращении к процедуре **Point** должны быть переменные, константы или выражения целого типа. Цвет точки определяется активным цветом линий.

После выполнения процедуры активная графическая позиция устанавливается в точку с координатами  $x, y$ .

Как и раньше, сама процедура не объявляется, она встроена в среду программирования. Чтобы построить точки, достаточно записать команду с координатами.

### **Задание 1**

Напишите программу построения с помощью точек линию длиной в 100 пикселей, приняв начало построения точку с координатами (50,20).

### **Задание 2**

Напишите программу построения прямого угла из точек в сплошную так, как показано на Рис. 13.

(z5\_14.pas)

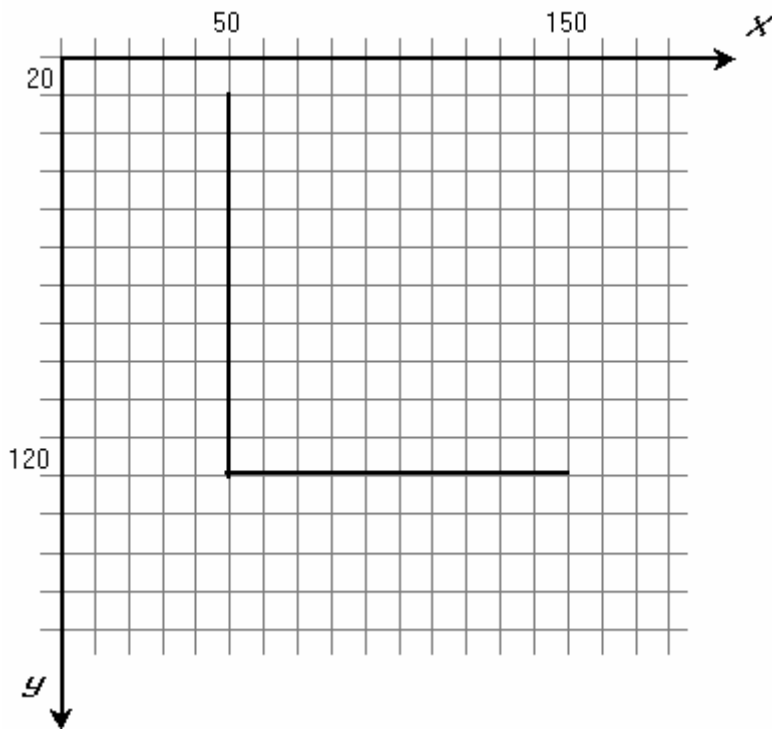


Рис. 13

### ***Задание 3***

Напишите программу построения прямого угла из точек с интервалом в одну точку. Можно использовать предыдущую задачу.

(z5\_15.pas)



## § 8

### *Установка текущих координат*

Активная графическая позиция устанавливается в точку с координатами (х,у) процедурой **MoveTo**, которая объявлена следующим образом:

**procedure MoveTo( x,y : integer);**  
**процедура Переместить( х,у : целые);**

Активную графическую позицию используют процедуры: LineTo, Write и WriteLn.

### *Проведение линии между точками*

В информатике наряду с понятием точки (пикселя) очень часто при построения используются линии. Но мы знаем, что линия это бесконечный след оставляемый каким либо графическим средством (мел, карандаш, ручка и др.) на поверхности или какой либо плоскости объекта. Различают несколько их видов: кривые, например, брошенная на землю веревка или нить, обруч – это замкнутая кривая; и прямые, например, натянутая нить, струна, под линейку проведенная на листе бумаги карандашом линия стрелка на брюках и др. Рис. 14.

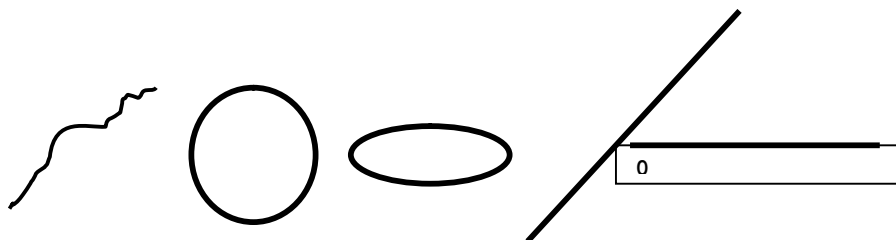


Рис. 14

*Кривые линии*

*Прямые линии*

Мы с вами в языке программировании будем иметь дело с линиями прямыми, а точнее с отрезками прямых линий (ограниченных с двух сторон точками, называемыми началом и концом линии. Как видите понятие отрезка не используем, хотя в действительности на экране построить бесконечную линию не возможно. И еще потому, что сказать слово «линия» гораздо быстрее, чем «отрезок прямой линии».

Что касается кривых линий, о них пойдет разговор чуть позже, на других уроках.

И так, линию проводят с помощью обращения к стандартной процедуре `line`, которая объявляется следующим образом

```
procedure line(x1, y1, x2, y2 : integer);  
процедура линия (x1, y1, x2, y2 : целые);
```

где `x1,y1` и `x2,y2` - координаты двух концов линии. Фактическими параметрами при обращении к процедуре `line` должны быть переменные, константы или выражения целого типа. Как мы раньше с вами договорились, в тексте программы мы не используем процедуры, используем команды, то есть, в программе построения линии будет написано:

```
Line(x1,y1,x2,y2);  
Линия(x1,y2,x2,y2);
```

Линия будет проведена активным цветом линий – по умолчанию он черный. Если бы можно было посмотреть построение в замедленном виде, то при выполнении этой команды линия строилась бы от точки с координатами `(x1,y1)` – начало отрезка, к точке с координатами `(x2,y2)` – конца его. После выполнения процедуры активная графическая позиция устанавливается в точку с координатами `x2,y2` – конец отрезка.

Из материала предыдущих уроков вам известно, что после запуска программной среды, активными координатами уста-

навливаются  $(0,0)$  – точки  $O$ , которую мы назвали началом отсчета или началом координат.

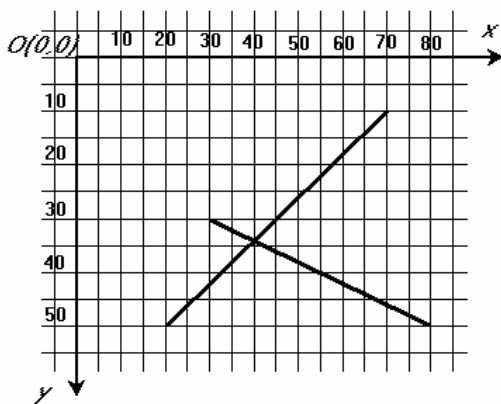
Если подумать, то можно сказать такое утверждение, что текущие координата в ходе построения линии между двумя точками менялись дважды. Первый раз они были текущими в точке начала отрезка  $(x_1, y_1)$ , после завершения построения, активными стали координаты конца отрезка  $(x_2, y_2)$ .

### ***Пример:***

Построить две линии с координатами начала отрезка  $(30,30)$  и конца отрезка  $(80,50)$  у одной и соответственно  $(20,50)$  и  $(70,10)$  у другой. В результате выполнения фрагмента программы

```
line(30,30,80,50);  
line(20,50,70,10);
```

получим такие линии как показаны на Рис. 15.



*Рис. 15*

### ***Проведение линии к точке***

Вы уже знаете, что из отрезков прямых линий можно строить фигуры: треугольники, квадраты, прямоугольники и

многоугольники. При их построении, следующая линия начинается с конца предыдущей. Чтобы такое можно было осуществить на изучаемом языке программирования, удобно пользоваться процедурой **LineTo**, которая объявляется следующим образом:

```
procedure LineTo( x,y : integer);  
процедура ЛинияК ( x,y : целые);
```

где (x,y) - координаты конца линии. Линия начинается с активной графической позиции. Фактическими параметрами при обращении к процедуре **LineTo** должны быть переменные, константы или выражения целого типа.

После выполнения процедуры активная графическая позиция устанавливается в точку с координатами (x,y), указанную командой.

### **Пример:**

Построить три отрезка выходящих один из другого и имеющих координаты: (30,30), (10, 50), (60,50). В результате выполнения фрагмента программы:

```
LineTo(30,30);  
LineTo(10,50);  
LineTo(60,50);
```

получим такие линии Рис. 16.

Вы видите, что первая линия, ее начало – это начало координат. Дальше, остальные линии строились из конца предыдущей, так как конец построенной линии имел текущие координаты.

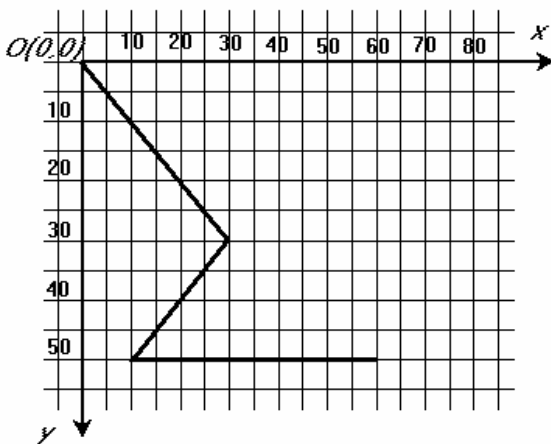


Рис. 16

Активную графическую позицию можно задавать командой **MoveTo(x,y)** или построить линию командой **Line(x1,y1,x2,y2)**.

### ***Задание 1***

Построить треугольник, вершины которого A(200,300), B(400,10), C(600, 300)

Объясните, почему координаты точек имеют такие большие значения?

(z5\_16.pas)

### ***Задание 2***

Используя команду **MoveTo(x,y)** придумать задачу построения квадрата в экранной системе координат.

(z5\_19.pas)

## § 9

### *Замкнутые фигуры из отрезков*

На уроках геометрии вам придется много и часто сталкиваться с фигурами, образованными отрезками прямых линий. Их великое множество. Мы с вами на уроках информатики остановимся на тех, которые часто применяются как в жизни, так и в предметных областях знаний.

Это треугольники, четырехугольники, прямоугольники и квадраты, пятиугольники и шестиугольники.

Мы не будем рассматривать их свойства, нас больше пока интересует их «внешний вид», т.е. форма. На рисунках подробно представлены треугольники, правда не все. И это не просто. Дальше мы много задач будем посвящать именно им.

И так, внимательно рассмотрите все рисунки, появятся вопросы, обратитесь к учителю.

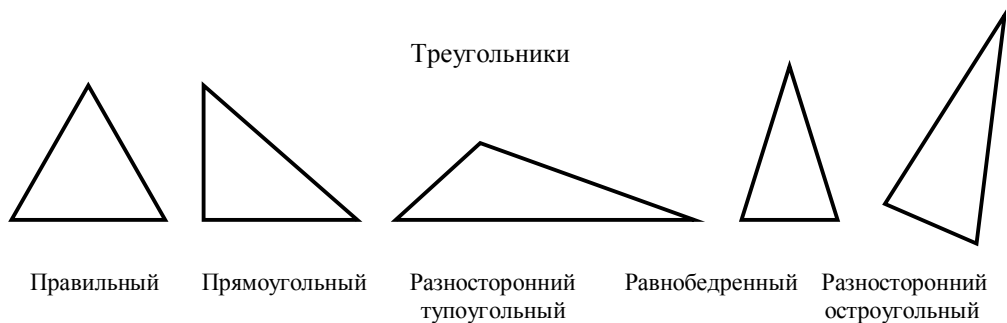


Рис. 17

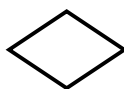
## Многоугольники



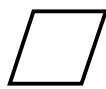
Прямоугольник



Квадрат



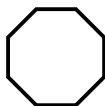
Ромб



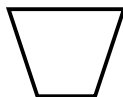
Параллелограмм



Шестиугольник



Восьмиугольник



Трапеция



Пятиугольник

Рис. 18

А мы продолжаем работать со средой Algo. Вам предоставляется возможность написать несколько программ для тренировки и закрепления своих знаний

### Задание 1

Построить в экранной системе координат квадрат, найти середины его сторон и соединить их между собой отрезками прямой линии. Какие фигуры и сколько получилось в результате построения (работы программы)? Если рисунок не просматривается в экранной области, что нужно сделать для его просмотра?

(z5\_20.pas)

### Задание 2

Построить два квадрата так, чтобы второй квадрат был смещен вправо вверх и перекрывался первым. Соедините их вершины параллельными линиями.

Указание:

- 1) Квадраты должны быть одинаковыми по размерам.
- 2) Сделайте вначале рисунок на бумаге.

- 3) *Определитесь с координатами их вершин*
- 4) *И только после этого составляйте программу*  
(z5\_21.pas)

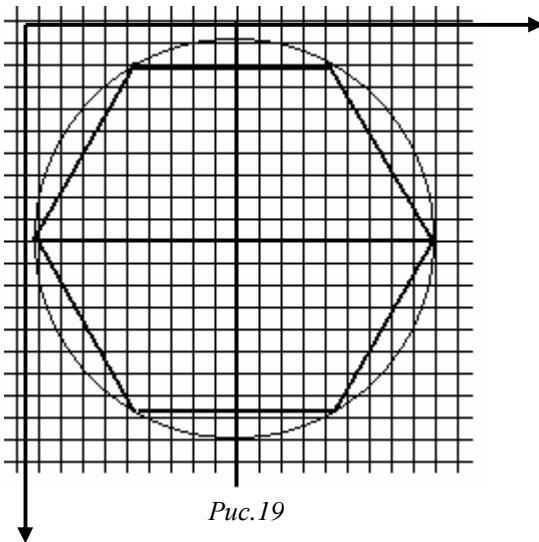
### **Задание 3**

Постройте шестиугольник, вершины которого соединить между собой. Сколько и каких получилось фигур внутри его?

*Указания:*

- 1) *Задача довольно трудная для построения, ее надо строить предварительно на тетрадном листе.*
- 2) *Выбрать единичный отрезок*
- 3) *Построить систему координат*
- 4) *Циркулем построить окружность*
- 5) *Этим же раствором циркуля разбить ее на 6 равных частей*
- 6) *Полученные точки соединить*
- 7) *Определить приблизительно координаты вершин*
- 8) *Записать их и ввести в программу для линий*
- 9) *Смотрите Рис. 19*

(z5\_23.pas)



*Рис.19*



## ***Построение прямоугольника***

Прямоугольник строят с помощью обращения к стандартной процедуре **rectangle**, которая объявляется следующим образом:

```
procedure Rectangle(x1, y1, x2, y2 : integer);  
процедура Прямоугольник (x1, y1, x2, y2 : целые);
```

где x1,y1 и x2,y2 - координаты двух противоположных вершин прямоугольника. Фактическими параметрами при обращении к процедуре **rectangle** должны быть переменные, константы или выражения целого типа.

После выполнения процедуры активная графическая позиция устанавливается в точку с координатами x2,y2.

Пример:

в результате выполнения фрагмента программы

```
rectangle(30,30,90,60);
```

```
rectangle(10,90,60,10);
```

получим такие прямоугольники

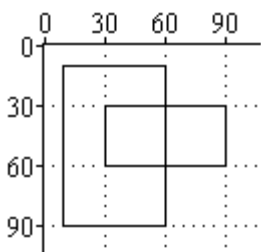


Рис. 20

## ***Построение эллипса***

Эллипс строят с помощью обращения к стандартной процедуре **ellipse**, которая объявляется следующим образом:

**procedure Ellipse(x1,y1,x2,y2 :integer);**  
**процедура Эллипс (x1, y1, x2, y2 : целые);**

где x1,y1 и x2,y2 - координаты двух противоположных вершин прямоугольника, в который вписывается эллипс. Фактическими параметрами при обращении к процедуре ellipse должны быть переменные, константы или выражения целого типа.

После выполнения процедуры активная графическая позиция не изменяется.

Пример:

в результате выполнения фрагмента программы

```
ellipse( 0, 0,90,60);  
ellipse(30,30,60,90);
```

получим такие эллипсы:

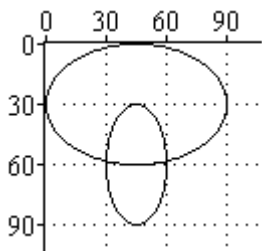


Рис. 21

#### **Задание 4**

Составить программу построения квадрата и вписанного в него круга

(z5\_24.pas)

#### **Задание 5**

Написать программу построения модели, представленной на Рис. 22.

(z5\_25.pas)

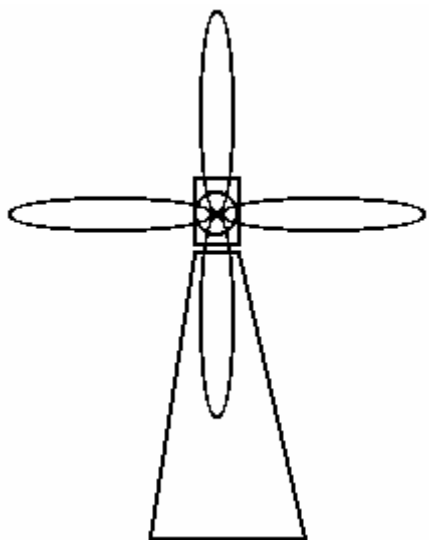


Рис. 22

*Это творческое задание, позволяющее сравнить работу в графическом редакторе Paint и получение аналогичного рисунка в среде программирования.*

*Сделайте необходимые заключения, проведя анализ своей деятельности в названных приложениях.*

*Успехов Вам, дорогие ребята!*

*Установка цвета линий.*

В изобразительном искусстве есть понятие основных цветов и их принято определить в количестве три. В физике, при рассмотрении темы «Разложение видимого света» на составляющие используется «магическое» число семь, т.е. наименьшее количество распознающих глазом цветов у человека со средними данными цветовосприятия.

В действительности цветов можно получить неограниченное количество.

Первые персональные компьютеры использовали два цвета (авт. не уточняем, какие именно, это зависит от вещества, нанесенного на экран монитора). В 80-е годы прошлого столетия уже использовали мониторы, палитра (набор цветов) которых имела 8 и 16 цветов. Даже сейчас во многих справочниках их нумерация от 0 до 15 остается в качестве параметров для вывода на экран того или иного цвета.

Мы работаем в среде программирования Pascal (Algo), она использует 256 цветов, хотя современные компьютеры это число определяют как минимальное и могут использовать 65536 цветов, более 16 млн. ...

Активный (текущий) цвет линий устанавливают с помощью обращения к процедуре Color, которая объявляется следующим образом

**procedure Color (r, g, b : integer);**  
**процедура Цвет (r, g, b : целые);**

где r, g, b - задают части красного, зеленого и синего в результирующем цвете линий. Установленным цветом будут отображаться все линии, которые выводятся на экран процедурами LineTo, Line, Point. Фактическими параметрами при обращении к процедуре Color должны быть переменные, константы или выражения целого типа, значения которых находится в диапазоне 0 -:- 255.

По умолчанию установлен черный цвет линий (0,0,0).

### ***Установка цвета заполнения.***

Активный (текущий) цвет заполнения устанавливается с помощью обращения к процедуре BrushColor, которая объявляется следующим образом:

**procedure BrushColor (r, g, b : integer);**

**процедура ЦветКисти (r, g, b : целые);**

где r, g, b - задают части красного, зеленого и синего в результирующем цвете заполнения. Установленным цветом будут закрашиваться фигуры процедурами Rectangle, Ellipse. Фактическими параметрами при обращении к процедуре BrushColor должны быть переменные, константы или выражения целого типа, значение которых находится в диапазоне 0 :- 255.

Если задать один из параметров отрицательным, то устанавливается так называемый прозрачный цвет заполнения, то есть фигуры не закрашиваются. Такой режим установлен по умолчанию.

Для получения чистых цветов в нужно задавать значение r, g, b такие, чтобы остаток от деления их на 8 был равен 7.

### ***Закрашивание замкнутых областей***

Чтобы закрасить запертую область, нужно обратиться к процедуре Fill, которая объявляется следующим образом:

**procedure Fill( x, y : integer);**

**процедура Закрасить( x, y : целые);**

Фактическими параметрами при обращении к процедуре Fill должны быть переменные, константы или выражения целого типа.

Начиная с точки с координатами x,y, процедура заполняет экран во всех направлениях активным цветом заполнения до тех пор, пока не встретится граница иного цвета, чем цвет ука-

занной точки. Иными словами процедура Fill меняет цвет замкнутой области, внутри которой лежит точка (x,y) на активный цвет заполнения, установленный процедурой BrushColor

Процедура Fill не выполняет никаких действий, если задан прозрачный цвет заполнения или цвет точки (x,y) совпадает с цветом заполнения.

В зависимости от типа монитора операционная система может использовать для заполнения большой набор цветов. В этом случае процедура Fill может работать некорректно. Для получения чистых цветов в этом случае нужно задавать значение r, g, b такие, чтобы остаток от деления их на 8 был равен 7.

Учитывая то, что данное пособие печатается в черно-белом оформлении, не приводится ни одного фрагмента программы, ни одного рисунка, связанных с цветностью форм фигур и их заполнения, а также варианты комбинаций r, g, b как параметров оператора Кисть. Предлагаем самостоятельно провести экспериментальную работу на ПК и заполнить в тетрадах таблицу, образец которой дан ниже, учитывая разъяснения в предыдущем абзаце.

Цвет заполнения	BrushColor (r, g, b)		
	r	g	b
	0	0	0
	255	255	255
	?	?	?

### **Задание 1**

Используя задачу z5\_24.pas внести необходимые изменения так, чтобы два графических объекта были окрашены в различные цвета

(z5\_26.pas )



