Первое, что, на мой взгляд, неизменно для всех сред программирования и вообще для всего, связанного с созданием программ – это три закона:

1) Компьютер думать не умеет никогда и ни при каких обстоятельствах. Вместо компьютера думает программист.
2) Пользователь думает всегда не так, как предполагает разработчик. Или пользователь вообще не хочет думать.
3) Количество ошибок в программе не зависит от размера программы. И при каждом тестировании остается не менее 10% ошибок, бывших до начала этапа проверки.

Написание текстовой игры, по крайней мере на АХМА, наполовину состоит из программирования. В остальных средах/платформах я работать еще не пробовала, но мне кажется, что там ситуация принципиально аналогичная, отличающаяся только «правилами орфографии». Поэтому далее я представлю несколько советов, руководствуясь которыми вместе с данными из справки по конкретной платформе, вы сможете работать, не испытывая существенных трудностей.

Первое, даже если это не требует используемая среда, пропишите шапку программы/игры. В шапке укажите все переменные, которые вы собираетесь использовать и присвойте им начальные значения (например A:=0). Благодаря этому простому действию исключается целый класс ошибок с неопределенными переменными и ошибок, связанных с неожиданными значениями по умолчанию (например, автоматически неопределенным переменным присваивается 1 или 0). Кстати, немного о синтаксисе/орфографии: как правило, что бы не путать знак равенства при сравнении и присвоение значения, в первом случае ставят «=» или «==», а во втором «:=».

Переменные, которые используются в программе, могут подвергаться математическим операциям. **Внимание!** Не все из представленных ниже возможностей могут быть предусмотрены, поэтому уточните в руководстве пользователя соответствующей программы. Так же обращаю внимание, что пишу все объяснения исходя из своего опыта программирования на Паскале в старших классах и на младших курсах. Поэтому пособие предназначено для **облегчения понимания** процесса использования элементов программы в текстовой игре, но **не является сферическим пособием в вакууме** для

гениев работы с переменными. Так же приводимые фрагменты программ будут написаны с опорой на Паскаль, как на самый простой для понимания языков программирования и наиболее просто показывающие логику машины. Все операторы могут быть переведены на иные языки программирования простым изменением «орфографии».

1) Сложение/вычитание. Записывается обычно в виде А:= Б+С или А:= К-5. С помощью того же приема может быть реализован стандартный счетчик: А:=А+1. В некоторых программах допускается сокращенная запись: А:=А++, но я не рекомендую ее использовать т.к. надежность сокращенных форм записи всегда иже, чем полных). Так же будьте аккуратны с выбором что с чем складывать и следите, что бы участвующие в операции переменные были одного типа (а то можно попытаться сложить 5 с ведром, что скорей всего программой будет воспринято как ошибка).

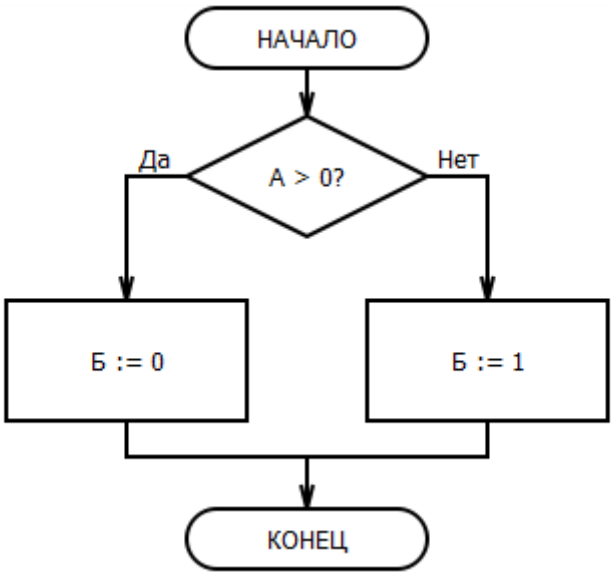
2) Умножение, деление. Ситуация с ними аналогична ситуации с сложением и вычитанием. При работе так же следует избегать деления 7 на тучку, следить, что бы у всех участвующих переменных были числовые значения и что бы в знаменателе не получался «0».

3) Деление с остатком и остаток от деления – это полезнейшие функции, которые, к сожалению, не везде реализованы априори. Наиболее распространенные обозначения div и mod. Где А:=Б div С – это выражение, результатом выполнения которого будет присвоение переменной «А» целой части от деления Б на С. То есть если Б=7, а С=2, то переменной А будет присвоено значение 3 т.к. 7/2=3,5 или 3 целых и 1 в остатке (2*3+1=7). Вторая функция - А:=Б mod С – это выражение, результатом выполнения которого будет присвоение переменной «А» остатка от целочисленного деления Б на С. То есть если Б=7, а С=2, то переменной А будет присвоено значение 1 т.к. 7/2=3,5 или 3 целых и 1 в остатке (2*3+1=7).
Что бы стало понятнее, вспомните, как вас учили операции деления столбиком в младших классах: то число, которое получалось в результате – это div, а число, остававшееся в нижней строке столбца – это mod.
 # Если вы используете сложные выражения, содержащие скобки, то ставьте открывающую и закрывающую скобочки одновременно. Благодаря этому вы избежите некорректного выполнения действия

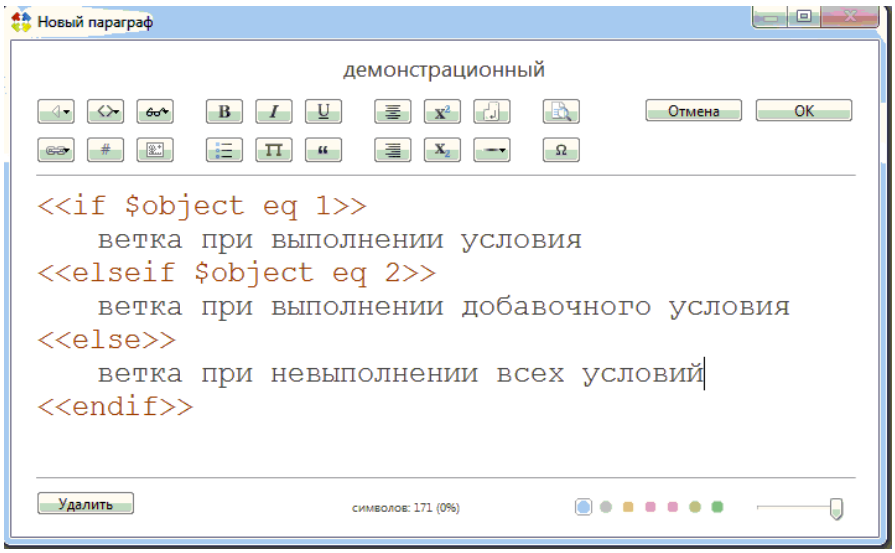
из-за забытой закрывающей скобки. Так же для надежности можете «фразы» div и mod отделять скобками. Например А:= 7 + (Б div С)/(В mod С) – (Л mod К).

4) Ветвление или выбор вероятности, или использование «if».

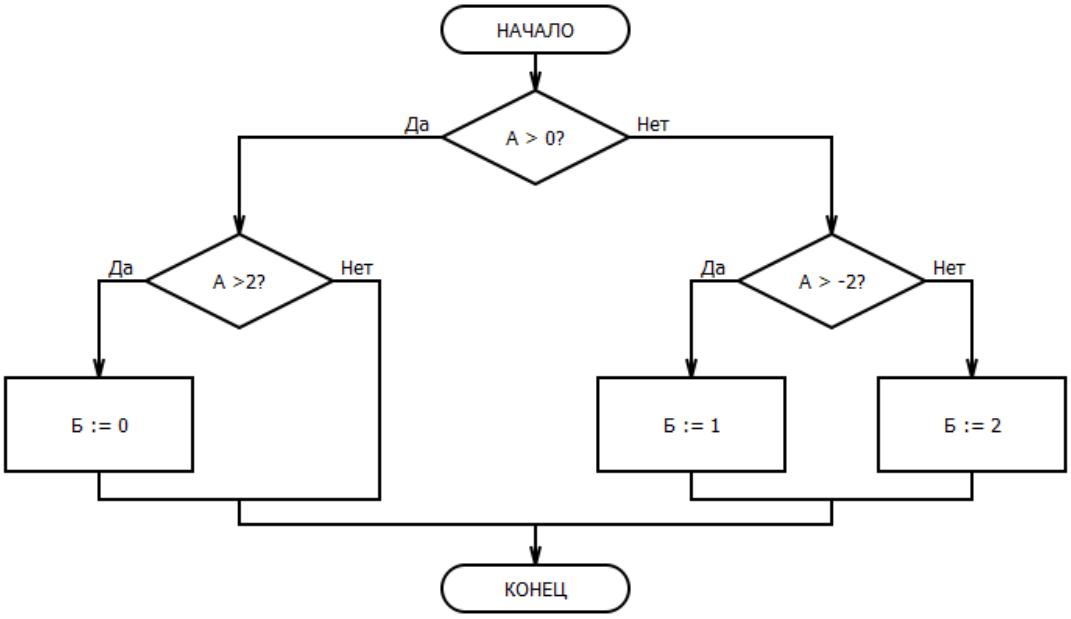
Для записи развилки сначала обычно указывается название операции – if, если – которое показывает компьютеру, что делать дальше. После ставят условие деления, например A>0, которое служит критерием выбора ветки. После указания условия пишут служебное слово then (тогда), после которого начинается ветка, соответствующая выполнению условия. Начало блока часто обозначается словом begin, а завершение – end, но не всегда. Что бы отделить ветки ветвления используется слово else, после которого идет блок, соответствующие не выполнению начального условия. По окончании описания обеих веток ставится служебное слово end(окончание), показывающее завершение ветвления. Итого, кусочек программы, представленный на рисунке, будет выглядеть следующим образом:




```
if A > 0 then
   begin
     Б := 0;
   end
  else
   begin
     Б := 1;
   end;
```



Многие среды программирования/платформы позволяют создавать деление более чем на две ветки за одно использование оператора «if». Так, вместо нескольких вложенных матрешкой операторов, можно использовать elseif, который обозначит дополнительную ветвь и условие ее выполнения. Кстати, синтаксис исполнения ветвления на 3 ветви в АХМА представлен на рисунке выше. А в виде блок-схемы, или алгоритма, варианты тройного и более ветвлений на рисунке ниже.



На представленном участке выполняется следующее, если алгоритм переводить на понятный всем язык:

a. Проверяется выполнение начального условии.

b. Если оно выполняется, то есть А больше 0, то далее исполняется левая ветвь алгоритма.

c. Проверяется добавочное условие, то есть сравнивается А с 2.

d. Если А больше 2, то выполняется присвоение переменой Б значение 0.

e. Если А меньше или равно 2 (не больше 2), то выполняется правая часть левой ветки. То есть мы сразу переходим к выходу из ветвления т.к. не поставили на эту вероятность никаких изменений.

f. Если начальное условие не выполнялось, то мы выходим в правую ветвь. В которой опять проверяем добавочное условие, отличное от добавочного условия левой ветви.

g. При А, находящейся в диапазоне значений от -2 до 0, включая 0. Мы выходим в левую часть правой ветви и выполняем присвоение переменной Б значения 1.

h. В случае не выполнения ни одного из условий (ни основного, ни добавочного), мы выходим на крайнюю правую часть и присваиваем переменной Б значение 2.
Вариант записи в виде операторов и служебных слов только что писаного кусочка будет выглядеть следующим образом

```
if A > 0 then
  begin
    if A >2 then
      begin
        Б := 0;
      end
    else
     begin
     end;
  end
 else
  begin
    if A > -2 then
      begin
        Б := 1;
      end
    else
     begin
       Б := 2;
     end;
  end;
```

#Что бы избежать ошибок, связанных с отсутствием того или иного служебного слова, советую сразу писать всю структуру, а потом наполнять ее содержанием.

5) Циклы предусловием, циклы с пост условием и цикл по счетчику– это элементы программы, позволяющие многократно, но не бесконечно выполнять последовательность действий.

Цикл с предусловием (цикл while) перед выполнением своей внутренней начинки или тела проверяет на соответствие истине

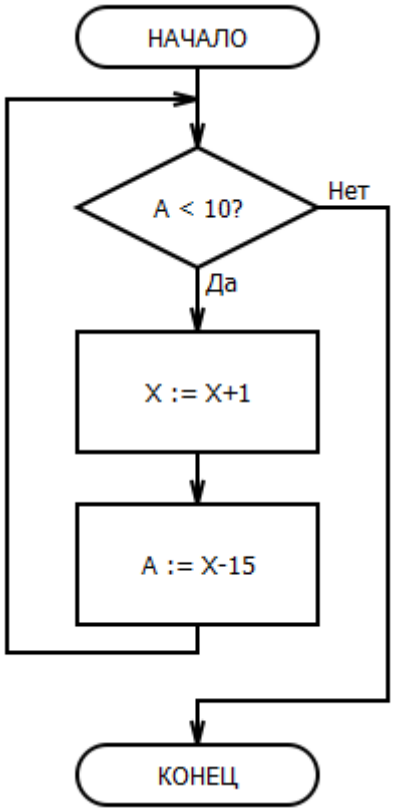
начальный параметр. После исполнения тела цикла программа возвращается к моменту перед входом в цикл. Если условие не выполняется, то и тело цикло не начинает исполняться, а просто продолжается выполнение программы так, как будто в ней нет цикла.

Цикл с пост условием (цикл until) сначала выполняет некую последовательность действий, заключенную в его тело, а потом проверяет соответствие условию выхода из цикла. До тех пор, пока финальное условие не начнет исполняться, тело цикла будет проигрываться раз за разом.

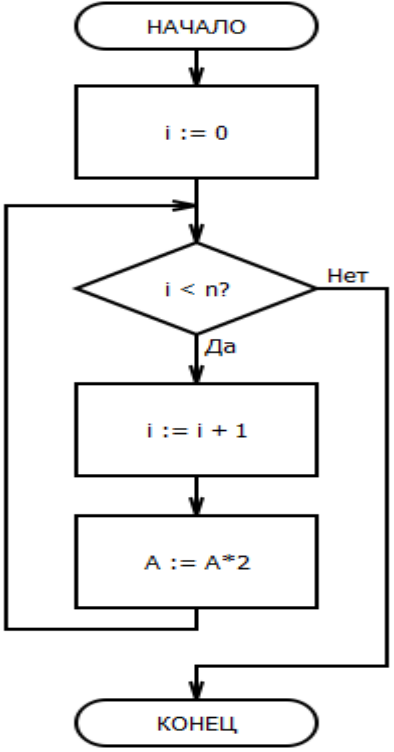
Цикл по счетчику (цикл for) – это разновидность цикла с предусловием, которая из-за большой востребованности обзавелась собственным вариантом орфографии. Использование данного типа цикла позволяет выполнить некоторую последовательность действий строго определенное количество раз. Особенностью реализации цикла является счетчик работающий по умолчанию. Стандартно переменная счетчика в каждом выполнении цикла (итерации) увеличивается на 1, но есть варианты уменьшения или иного шага изменения переменной.

Приведу ниже варианты исполнения каждого типа цикла в виде алгоритма и в виде кода.

```
while A < 10 do
  begin
    X := X+1;
    A := X-15;
  end;
```

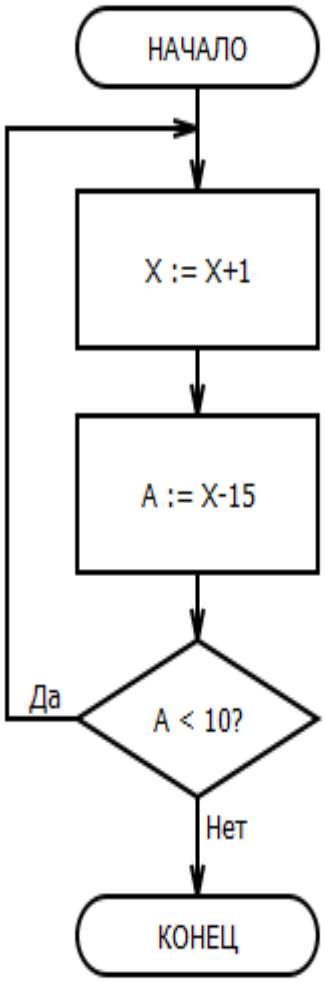


```
repeat
   X := X+1;
   A := X-15;
until not (A < 10);
```

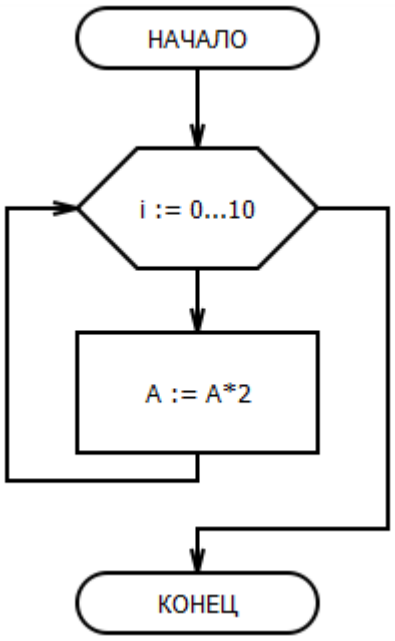


А вот цикл for представленный в двух вариантах: реализованный через while и обычный for

```
i := 0;
while i < n do
  begin
    i := i + 1;
    A := A*2;
  end;
```



```
for i := 0 to 10 do
   begin
      A := A*2;
   end;
```



На этой веселой картинке я завершаю обзор основных используемых элементов. Так же хотелось бы заметить, что не все из них могут быть возможны к реализации в используемой платформе напрямую. Если платформа не предусматривает использование того или иного действия у вас есть 2 варианта:

1) Подумать, как реализовать свою идею без цикла;
2) Придумать, как с помощью имеющихся возможностей организовать цикл.

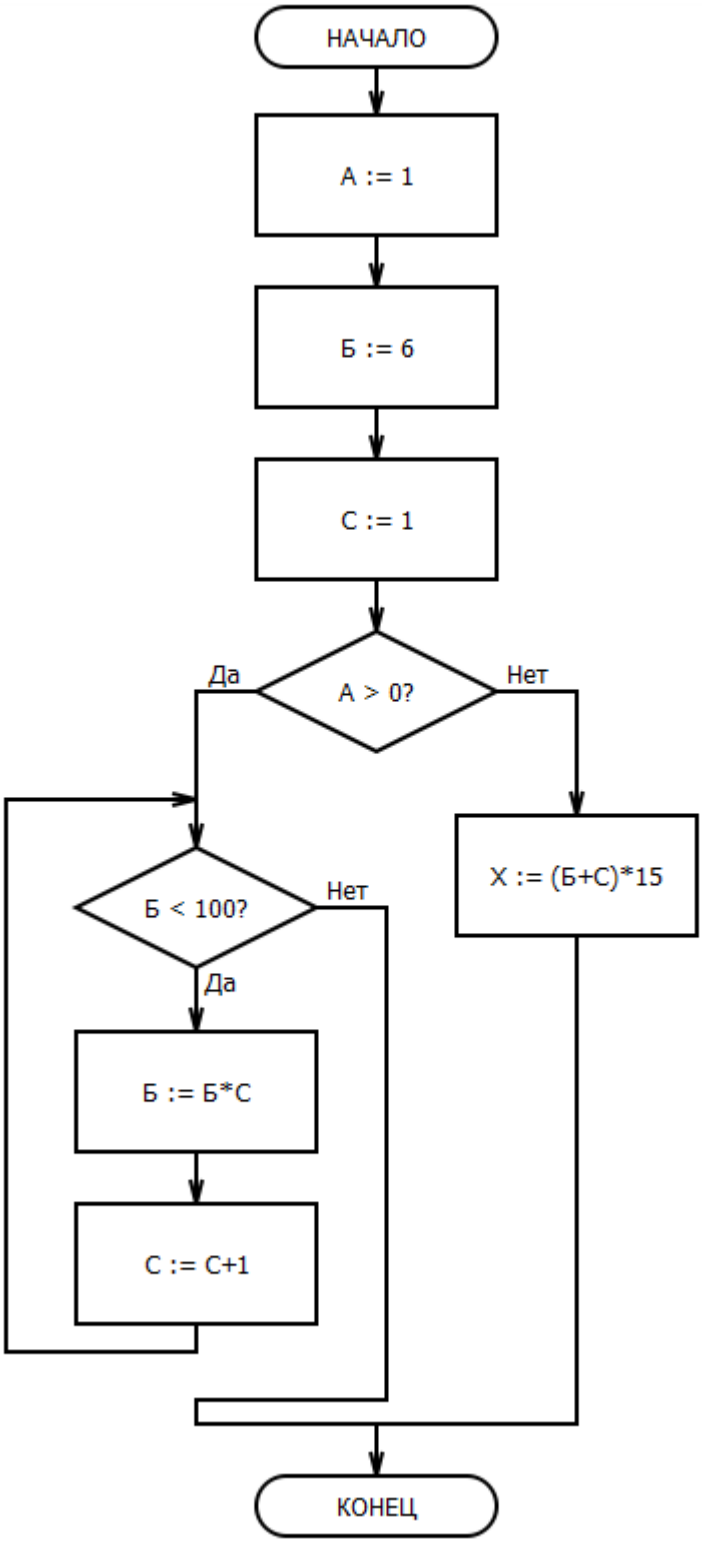
Все действия, которые вы пропишите в игре/программе, будут выполняться компьютером сверху вниз и справа налево. Поэтому будьте внимательны и не просите компьютер сначала разделить А на Б и приравнять полученное значение С, а потом присвоить переменной А значение, равное 3. Мы-то понимаем, что автор кода хотел получить деление 3х на Б и присвоить С, но компьютер думать и читать мысли не умеет.

В качестве заключения расскажу несколько правил красивого кода:

1) Визуально структурируйте свой код. Что бы понять необходимость данного шага просто сравните 3 записи одной и той же абсурдной программы.

Первый:

```
begin
   А := 1;
   Б := 6;
   С := 1;
   if А > 0 then
    begin
      while Б < 100 do
       begin
         Б := Б*С;
         С := С+1;
       end;
    end
    else
    begin
      Х := (Б+С)*15;
    end;
end;
```



Второй

```
begin
А := 1;
Б := 6;
С := 1;
if А > 0 then
begin
while Б < 100 do
begin
Б := Б*С;
С := С+1;
end;
end
else
begin
Х := (Б+С)*15;
end;
end;
```

Третий

```
begin
  A := 1; Б := 6; C := 1; if A > 0 then begin while Б < 100 do begin Б := Б*C; C := C+1; end; end else begin X := (Б+C)*15; end;
end;
```

Не знаю как вам, а мне намного удобнее работать с кодом, записанным по первому варианту. В первой форме записи глаз сразу выделяет структуру программы и исполняемые блоки, ассоциируясь с начальными условиями. То есть для вас не будет проблем с иерархией действий вашего кода.

2) Придумывайте переменным такие имена, что бы потом не гадать «А за что же она у меня отвечает…». Так же для создания имен переменных используйте одну и ту же логику. То есть если есть переменная «Возраст_Мамы», то для обозначения возраста папы не должна использоваться переменная «Папин_Возраст». А логичнее была бы единообразная переменная «Возраст_Папы». Если такое не возможно по какой либо причине, то см пункт 3.

3) Как можно чаще оставляйте комментарии (обычно они отмечаются «#» или «//»). Помните, что это сейчас вам все понятно и очевидно, вы прекрасно осознаете, что хотите сделать и зачем вам та или иная переменная. Но если вашу программу будет читать человек, то ему будет далеко не очевидна ваша логика. Да и вы сами, открыв код через несколько дней не вспомните, почему это количество огурцов задано функцией от помидоров. Не бойтесь оставить лишний комментарий, пользователи его не увидят, а вам, в случае чего, будет проще найти ошибки.

4) Сложные и громоздкие действия разбивайте на несколько последовательных. Таким образом вы существенно снизите вероятность появления ошибок и упростите их нахождение.

5) Старайтесь составлять и изменять код таким образом, что бы он был короче и проще. Чем менее громоздка ваша программа, тем легче в ней искать и исправлять ошибки, которые возникнут в любом случае.

6) Если вы сомневаетесь в своей логике, напишите программу в виде алгоритма. Для этого можно использовать карандаш и бумагу или воспользоваться специальными программами редакторами блок-схем. Для создания примеров в этой методичке я использовала AFCE Algorithm Flowchart Editor.

Конечно, это не все правила, но более полные их списки вы сможете найти сами, когда в этом возникнет необходимость. Приведенные 6 пунктов должны вполне удовлетворить текущие потребности.