КИПР

ТУСУР

Кафедра конструирования
и производства радиоаппаратуры

Циклические программы с графическим интерфейсом в среде Lazarus

Томск 2017

Кобрин Юрий Павлович

Циклические программы с графическим интерфейсом в среде Lazarus. Методические указания к лабораторной работе и по организации самостоятельной работы по дисциплинам «Информатика» и «Информационные технологии» для студентов очного и заочного обучения специальностей 211000.62 (бакалавриат) и 162107.65 «Информатика и информационные технологии» (специалитет). - Томск: Томский государственный университет систем управления и радиоэлектроники (ТУСУР), кафедра КИПР, 2017. – 27 с.

Lazarus — открытая бесплатная кроссплатформенная среда визуальной разработки программного обеспечения для компилятора *Free Pascal*, максимально приближённая к *Delphi*. Методические указания посвящены освоению основ объектно-ориентированного программирования (ООП) в интегрированной среде *Lazarus*, а также с особенностями технологии визуального программирования простейших циклических программ с графическим интерфейсом.

Методические указания предназначены для помощи в подготовке бакалавров и магистрантов в Информатике, выполнения курсовых и дипломных проектов.

Оглавление

1 Цели и задачи работы

Цель работы:

- Закрепление навыков работы в интегрированной среде Lazarus.
- Знакомство с операторами цикла в программах в интегрированной среде Lazarus.
- Освоение приёмов программирования и отладки циклических алгоритмов в интегрированной среде Lazarus

2 Порядок выполнения работы

Перед выполнением этой работы следует:

1) Ознакомиться с разделом «Методика разработки циклических программ» настоящей работы. В качестве дополнительной литературы использовать [1,2,3,4,5,6,7,8].

2) Войти в свой личный каталог, загрузить и настроить систему программирования ***Lazarus***.

3) Используя условия задания предыдущей лабораторной работы [7] составить три варианта циклической программы (с циклами типа $\boldsymbol{for}$, $\boldsymbol{while}$ и $\boldsymbol{repeat}$ .. $\boldsymbol{until}$), позволяющими рассчитать таблицу изменения напряжения на сопротивлении нагрузке $\boldsymbol{U\_R}_{\text{Н}}(\boldsymbol{t})$, если время меняется в диапазоне $\boldsymbol{0 \le t \le T}$ и с шагом $\boldsymbol{\Delta t}$. Для всех вариантов период импульса $\boldsymbol{T}_{\text{И}} = \boldsymbol{6 \cdot A}$, где $\boldsymbol{A}$ - длина одного из участков импульса (например, A=10). Заметим, что каждый следующий вариант программы легко получить на основе коррекции текста предыдущего варианта программы, копию которой следует сохранить в своём каталоге под новым именем.

4) Ввести разработанную Вами программу, корректируя и дополняя свой шаблон.

5) Добиться с помощью отладки, чтобы программы давали правильные результаты.

6) Ответить на контрольные вопросы.

7) Оформить отчёт и защитить его у преподавателя.

3 Отчётность

Отчёт должен быть выполнен в соответствии с [9] и состоять из следующих разделов:

1) Тема и цель работы.
2) Индивидуальное задание.
3) Блок-схема алгоритма.
4) Откомпилированные тексты программ (в электронном виде).
5) Ответы на контрольные вопросы.
6) Результаты выполнения программ.
7) Выводы.

При защите отчёта по работе для получения зачёта студент должен:

- уметь отвечать на контрольные вопросы;
- обосновать структуру выбранного алгоритма и доказать, что разработанные приложения работает правильно;
- уметь пояснить назначение элементов разработанных приложений;
- продемонстрировать навыки работы в интегрированной среде *Lazarus*.

4 Контрольные вопросы

Ответьте на следующие контрольные вопросы:

1) Что такое цикл? В каких случаях его необходимо применять? Какие операторы цикла Вы знаете?

2) Что такое «зацикливание» программы? Как изменяются данные, если программа «зациклилась»? Какие изменения нужно сделать в алгоритме Рис. 4.1, чтобы избежать зацикливания?

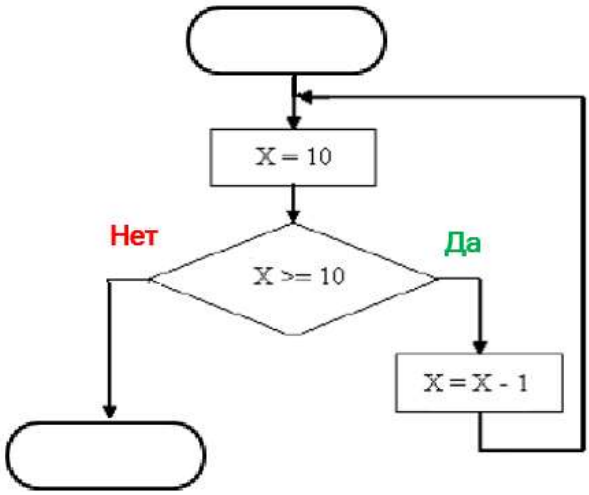


Рис. 4.1 - Циклический алгоритм

3) Какие значения примут переменные в результате работы алгоритмов, представленных на блок-схемах Рис. 4.2 *а), б), в).*

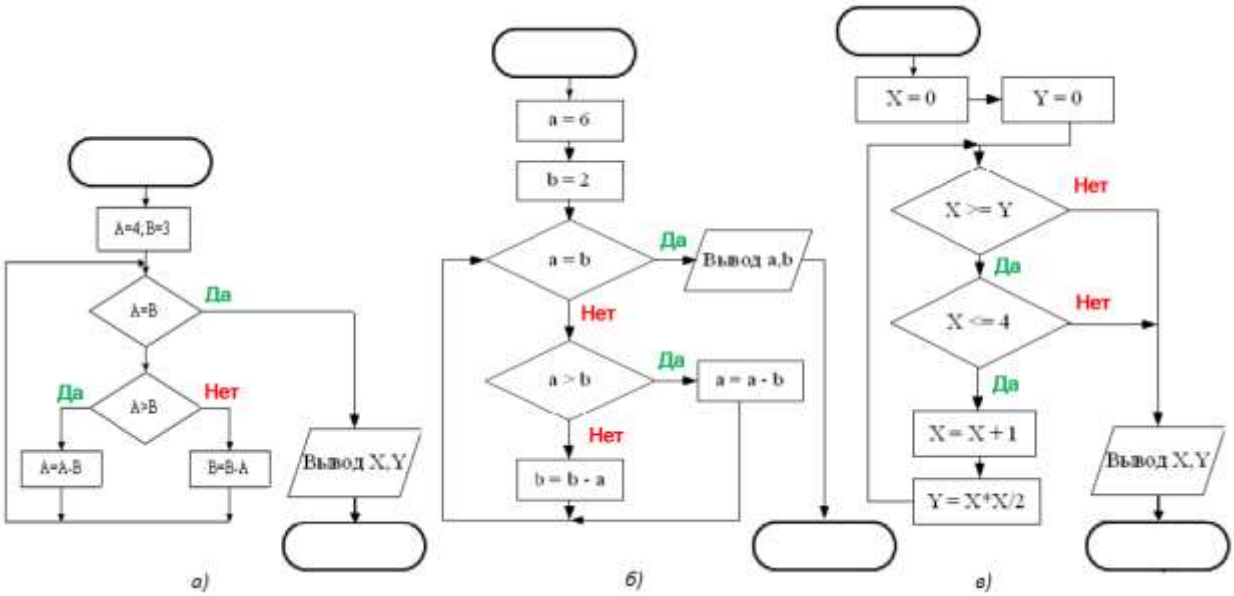


Рис. 4.2 – Анализ условий выхода из циклических алгоритмов

4) В каких случаях оператор ***for*** предпочтительнее использовать для организации циклов? Как записывается и как работает оператор ***for***? Какие ограничения по типу накладываются на управляющего переменного оператора ***for***? С каким шагом может изменяться параметр оператора ***for***?

5) Может ли тело оператора цикла со счётчиком не выполниться ни разу?

6) Как можно рассчитать число итераций в операторе ***for***? Сколько итераций совершится при использовании следующего цикла:

```
for i := 5 to 15 do
  writeln(i);
```

7) Чему равно количество повторений тела оператора цикла со счётчиком, если управляющая переменная цикла принимает все целые значения от 1 до 10?

8) Почему в программировании цикла со счётчиком существует правило: нельзя изменять управляющую переменную цикла в теле цикла?

9) Как работает оператор цикла ***while***? Может ли тело цикла с предусловием не выполниться ни разу?

10) Сколько раз выполнится цикл ***while*** в программах на Рис. 4.3, *a), b)*:

```
a : = 3; b := 7;
while (a / 2) ≤ (b / 3)
  begin
    a := a + 2; b := b+3
  end;
```

a)

```
I := -10;
while i > 0 do
i:=i-2;
```

b)

Рис. 4.3 - Анализ работы цикла While

11) Как работает оператор цикла ***Repeat***? Поясните, в каком случае прекращаются повторения.

12) Может ли тело цикла с предусловием ***Repeat*** выполняться бесконечное число раз?

13) Может ли тело цикла с постусловием ***Repeat*** не выполниться ни разу?

14) В чем отличие оператора ***while*** от оператора ***Repeat***? Когда удобнее использовать оператор ***while***, а когда оператор ***Repeat*** ?

15) При помощи какого оператора можно досрочно выйти из цикла? Перейти к следующему шагу выполнения цикла?

16) Чему равно «a» после выполнения

```
a := 0;
repeat
  a := a - 5
until a >= 25
```

a)

```
a := 0;
for i := 1 to 10 do
  a := a + 2;
```

б)

```
a := 0; i := 1;
while i <= 10 do
begin
  a := a + 1; i := i + 1
end
```

в)

5 Методика разработки циклических программ

5.1 Постановка задачи

Одними из важнейших алгоритмических структур при решении многих задач являются циклические структуры, в которых для достижения результата определённая последовательность действий повторяется несколько раз (Рис. 5.1). Возможность многократного повторения некоторой группы операторов позволяет существенно упростить схему алгоритма и сократить длину соответствующей ему программы.

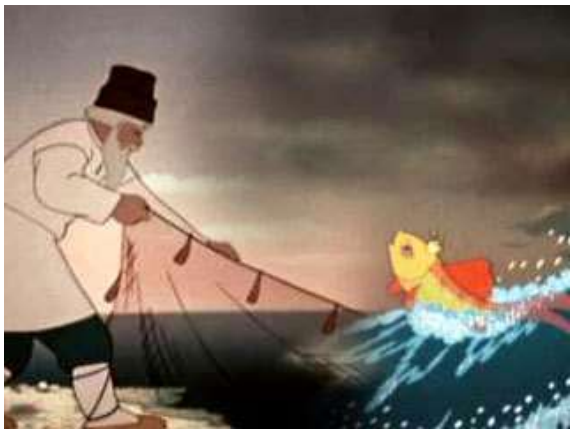


Рис. 5.1 – Есть ли здесь цикл?

Программирование циклов - одна из основных и в тоже время важных задач, которые часто приходится решать программисту.

Запомните!

Цикл – это многократно повторяющаяся последовательность одних и тех же действий.

Параметр цикла (**управляющая переменная цикла**) определяет момент выхода из цикла на продолжение программы.

Тело цикла – последовательность инструкций, предназначенная для многократного исполнения. В теле цикла может быть всего одно действие, может быть несколько действий, может не быть ни одного действия. Последнюю разновидность цикла называют *пустым циклом*. Пустой цикл иногда применяют, например, для задержки времени.

Итерация – однократное выполнение тела цикла.

Шаг цикла — это значение, на которое будет увеличиваться или уменьшаться параметр цикла при каждой итерации.

Условие выхода или **условие окончания цикла** – выражение определяющее - будет ли в очередной раз выполняться итерация или цикл завершится. Если условие выхода из цикла задано неправильно – цикл может никогда не кончиться, произойдет зацикливание (бесконечный цикл).

Счётчик цикла (счётчик итераций) – переменная, хранящая текущий номер итерации. Цикл не обязательно содержит счётчик циклов.

Счётчиков цикла может быть несколько, так как условие выхода из цикла может зависеть от нескольких изменяемых в цикле переменных, а может определяться и внешними условиями (например, наступлением определённого времени). В этом случае счётчик вообще может не понадобиться.

Циклические вычисления можно распределить по трём группам:

– **Счётные циклы** (циклы с заданным количеством повторений) – циклические процессы, для которых количество повторений известно. Например, заполнение массива или вычисления (табулирования) функции $y = f(x)$, у которой x меняется на конечном интервале от начального $X_Н$ до конечного значения $X_К$ с заданным шагом Δx (Рис. 5.2). Здесь много раз используется одна и та же формула с разными значениями аргумента x.

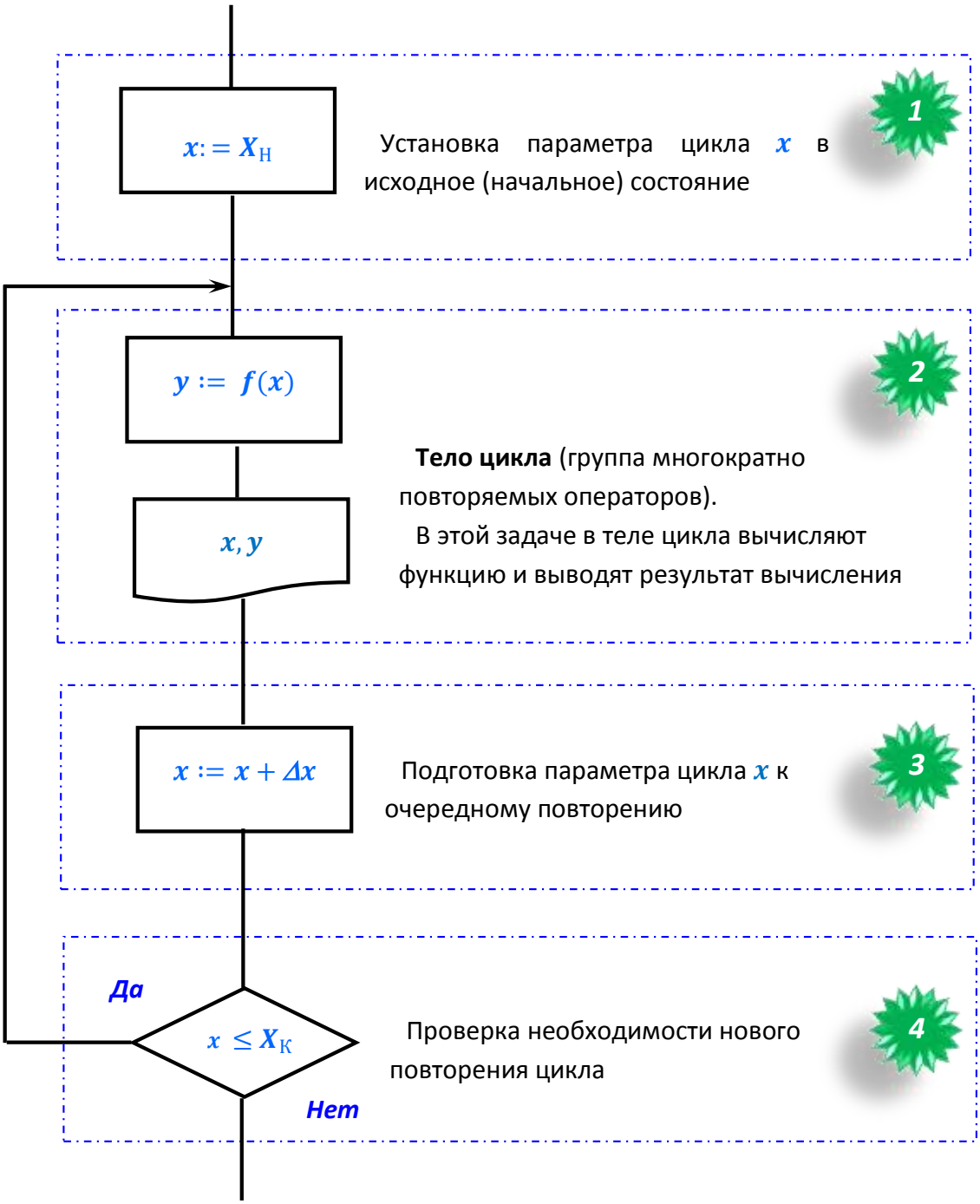


Рис. 5.2 - Блок-схема циклической программы вычисления значений функции $y = f(x)$ в диапазоне $X_Н$ .. $X_К$ с шагом Δx

– **Итерационные циклы** – циклические процессы, завершающиеся по достижении или нарушении некоторых условий. Например, нахождение суммы бесконечного сходящегося ряда с заданной точностью.

– **Поисковые циклы** – циклические процессы, из которых возможны два варианта выхода: выход по завершению процесса и досрочный выход по какому-либо дополнительному условию, например, нахождению первого отрицательного элемента массива.

Составляя циклические программы необходимо помнить, что результаты вычислений, запоминаемые переменной в левой части формулы оператора присвоения на каждом цикле, записываются *вместо предыдущих значений*. Чтобы не потерять эти значения на следующей итерации, нужно их как-то использовать: своевременно выводить на экран или принтер, запоминать (например, в качестве элемента массива) или употреблять как-то иначе. Так, например, в алгоритме на Рис. 5.2 выполняется циклическое вычисление функции $y = f(x)$, а результаты вычислений выводятся на документ в каждом цикле, например, в виде строки таблицы.

Любой циклический алгоритм обязательно начинается с блока установки параметра цикла (**блок 1**) в исходное состояние (см. Рис. 5.2). В циклической части, следующей за блоком установки параметра цикла в исходное состояние, можно выделить три блока:

– тело цикла (**блок 2**) – группу операторов, которую нужно повторять;

– подготовка очередного значения параметра цикла (**блок 3**);

– проверка необходимости нового повторения цикла (**блок 4**).

Порядок следования **блоков 2, 3, 4** в принципе может быть *любым*, поэтому возможны несколько вариантов организации циклов (Рис. 5.3).

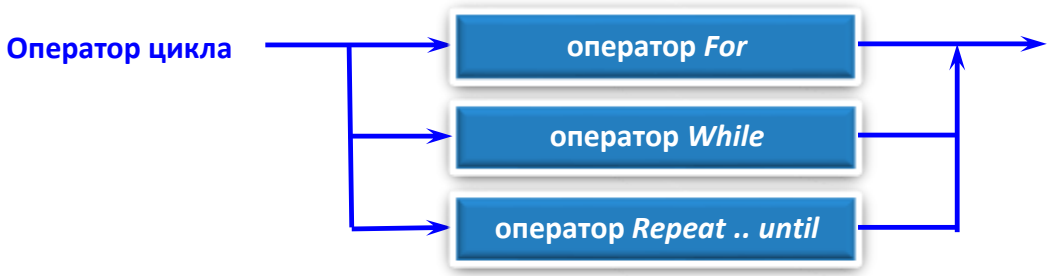


Рис. 5.3 – Операторы циклов

Различают циклы ***с заданным*** и ***с неизвестным*** числом повторений:

– если число повторений цикла *известно заранее* или *может быть вычислено*, то целесообразно использовать один из вариантов цикла со счётчиком ***For***;

– если же *момент* завершения *цикла зависит от выполнения некоторого условия* (например, в итерационных циклах, характеризующихся последовательным приближением к искомому результату с учётом заданной точности вычислений), то применяются операторы цикла с предусловием ***While*** или с постусловием ***Repeat***.

В общем случае в соответствии с ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения» циклы изображают с помощью символа, состоящего из двух частей, отображающих начало и конец цикла (Рис. 5.4). Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т.д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие.

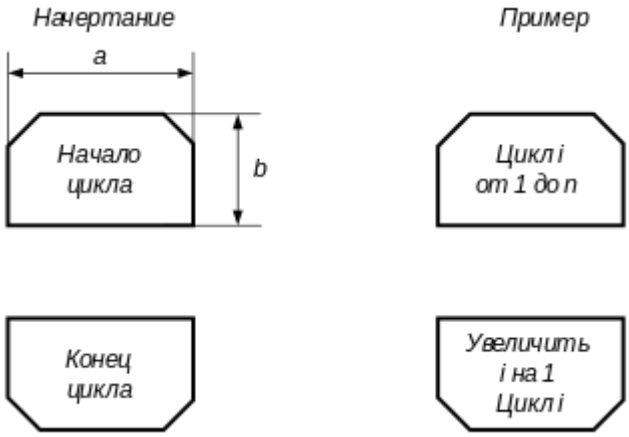


Рис. 5.4 - Изображение циклов по ГОСТ 19.701-90

Хорошее понимание и владение операторами повторения, а также рассмотренными ранее другими управляющими конструкциями, в огромной степени определяют профессиональный уровень программиста.

5.2 Оператор цикла с заданным числом повторений (со счётчиком) *For*

Цикл со счётчиком Рис. 5.5 — цикл, в котором некоторая переменная изменяет своё значение от заданного начального значения до конечного значения с заданным шагом, и для каждого значения этой переменной тело цикла выполняется один раз.

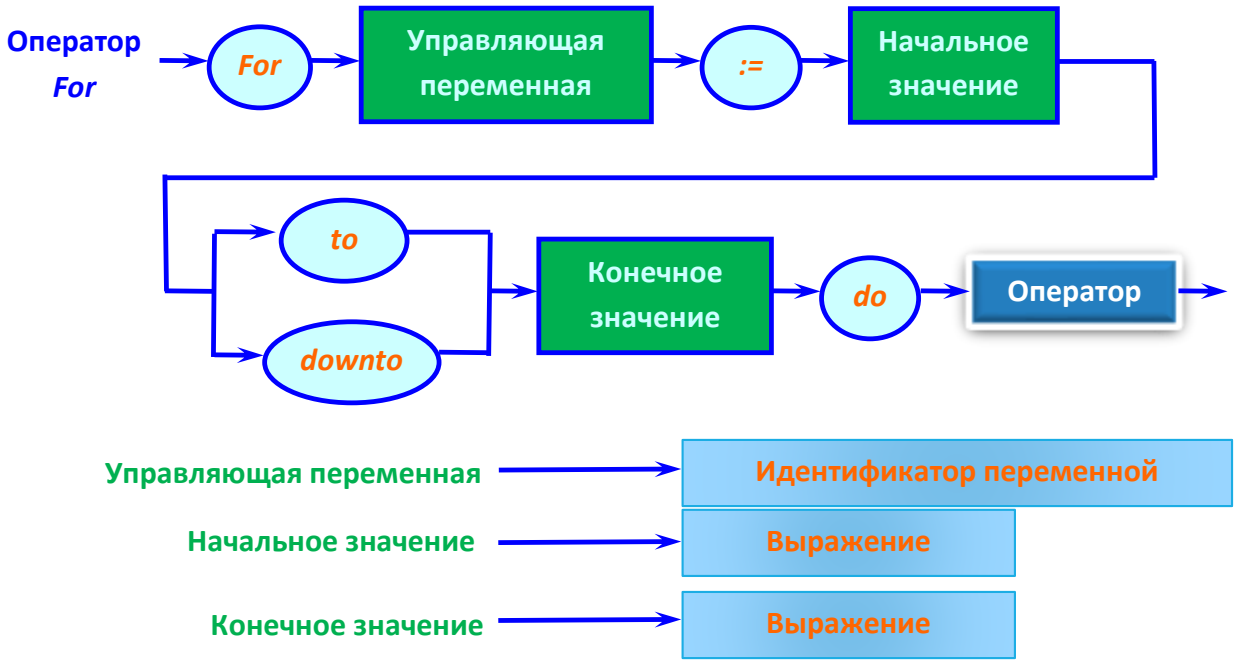


Рис. 5.5 - Синтаксическая диаграмма оператора цикла ***For***

Цикл со счётчиком ***for*** подходит для программирования таких циклических программ, в которых до выполнения цикла известны начальное и конечное значения счётчика повторений цикла (например, индекс первого и последнего элемента массива). Поэтому **цикл со счётчиком *for*** называют *циклом с заданным числом повторений*.

Возможны два варианта исполнения цикла.

1. Если счётчик при выполнении цикла наращивает своё значение на **+1:**

For <**Управляющая переменная**> ***:=*** <**Нач. значение**> ***to*** <**Кон. значение**> ***do***
<***Оператор***>***;***

Пусть вычисленные значения выражений <**Исходное значение**> и <**Конечное значение**> равны, соответственно, $i_{нач}$ и $i_{кон}$ и переменой цикла (счётчику циклов) i присвоено начальное значение $i_{нач}$. Совершается проверка - не превосходит ли переменная цикла конечное значение $i_{кон}$? Если $i_{нач} \leq i_{кон}$, то выполняется тело цикла (оператор или группа операторов). Далее значение управляющей переменной увеличивается на единицу $i := i + 1$ (шаг приращения переменной цикла задан неявно – служебным словом ***to***).

Процесс, включающий проверку, выполнение тела цикла и изменение управляющей переменной, повторяется до тех пор, пока проверка не даст положительный результат, т.е. $i > i_{кон}$. В этом случае оператор ***For*** завершает работу, переменная цикла объявляется неопределённой и осуществляется переход на операторa, следующий за оператором ***For.***

2. Если счётчик при выполнении цикла уменьшает своё значение на **-1:**

For <**Управляющая переменная**> ***:=*** <**Нач. значение**> ***downto*** <**Кон. значение**> ***do***
<***Оператор***>***;***

Если <**Конечное значение**> больше <**Начального значения**>, тело цикла не выполняется ни разу. При каждом повторении в этом варианте цикла ***For*** переменная цикла i уменьшается на единицу $i := i - 1$ (отрицательный шаг приращения переменной цикла задан неявно – служебным словом ***downto***). Проверяется: *если* значение переменной цикла больше или равно конечного значения $i \geq i_{кон}$, то тело цикла выполняется. Если же нет, и $i < i_{кон}$, то оператор ***For*** завершает работу, переменная цикла объявляется *неопределённой* и осуществляется переход на оператор, следующий за оператором ***For***.

К сожалению, никакой другой шаг, кроме ±1, в цикле *For* не предусмотрен!

В блок-схемах алгоритмов программ цикл со счётчиком ***For*** может также изображаться так, как показано на Рис. 5.6– с помощью блочного символа *«Модификация»,* задающего закон изменения управляющей переменной.

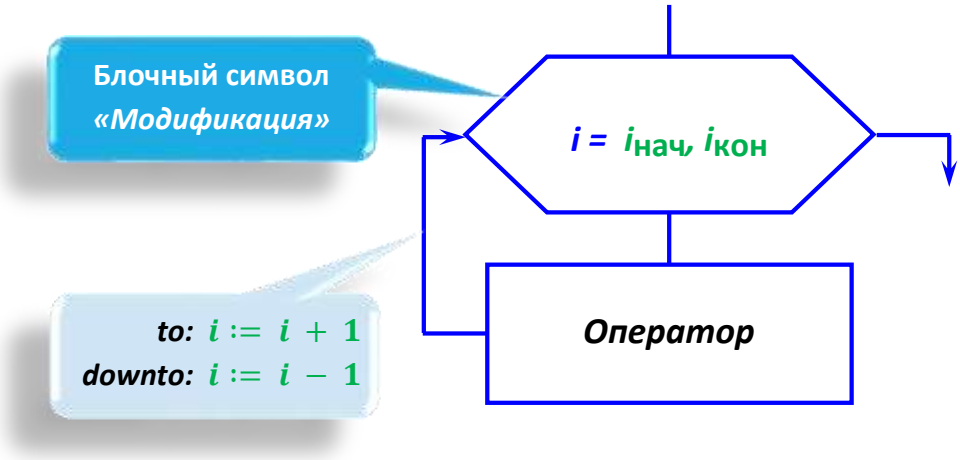


Рис. 5.6 – Изображение цикла ***For*** с помощью символа *«Модификация»*

В обоих вариантах записи оператора **For**:

- < **Управляющая переменная**> ⇒ переменная порядкового типа, к которым относятся данные типа ***shortint, integer, word, byte*** и ***Char***; она принимает последовательные значения от заданного <**Нача.значения**> до заданного <**Кон. значения**> ;
- <**Нач. значение**> и <**Кон. значение**> – выражения порядкового типа, определяющие, соответственно, начальное и конечное значение управляющей переменной цикла. Они рассчитываются только один раз перед началом выполнения оператора ***For***, и не должны меняться на протяжении всего процесса его выполнения.

Число повторений цикла легко найти по формуле

M = <**Кон. значение**> - <**Нач. значение**> ***+1;***

<***Оператор***> – единственный повторяемый оператор. Если необходимо выполнить несколько действий, то они должны быть объединены в один составной оператор путём заключения в составной оператор (группа операторов, заключённая между операторными скобками ***begin*** и ***end***).

Пример 5.1

Вычислить с помощью цикла ***For*** сумму всех целых чисел от ***1*** до ***N.***

Решение 5.1

Блок-схема вычисления суммы всех целых чисел от **1** до ***N*** изображена на Рис. 5.7.

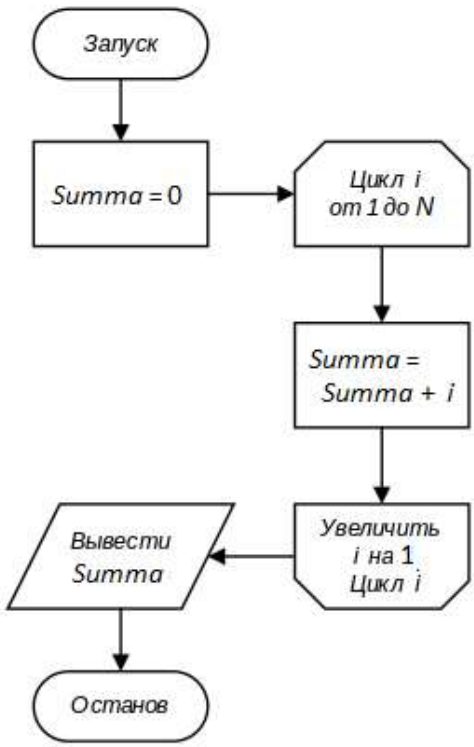


Рис. 5.7 - Блок-схема вычисления суммы всех целых чисел от ***1*** до ***m***

Фрагмент программы примера 5.1 может выглядеть так:

```
...
Var
   i,  //счётчик циклов
   N, //количество суммируемых чисел
   Summa: integer; //сумма всех целых чисел от 1 до N
...
   Summa := 0;  //Начальное значение суммы}
   For  i :=  1 to N do  //Цикл подсчета суммы
      Summa := Summa + i;
...
```

Пример 5.2 (с ошибкой)

После окончания итерации наращивание/уменьшение значения цикла происходит автоматически. Следовательно, специального оператора для увеличения значения счётчика цикла (типа $i := i + 1$) не требуется, Более того, подобный оператор приведёт к **непредсказуемой работе цикла!**

Фрагмент решения предыдущего примера 5.1 с ошибкой, может выглядеть так:

```
...
   Summa := 0;  //Начальное значение суммы}
   For  i :=  1 to N do  //Цикл подсчета суммы
    begin
      Summa := Summa + i;
      i := i +1;  ⇐   Ошибка! Нельзя самому изменять значение счётчика циклов!
    end
...
```

Пример 5.3

Вычислить и вывести на экран таблицу значений функции:

$$z := \frac{a^3}{a^2 + x^2}$$

если x изменяется от – 2 до 5 с шагом $h = 0{,}2$.

Решение 5.3

Возможная схема алгоритма приведена на Рис. 5.8.

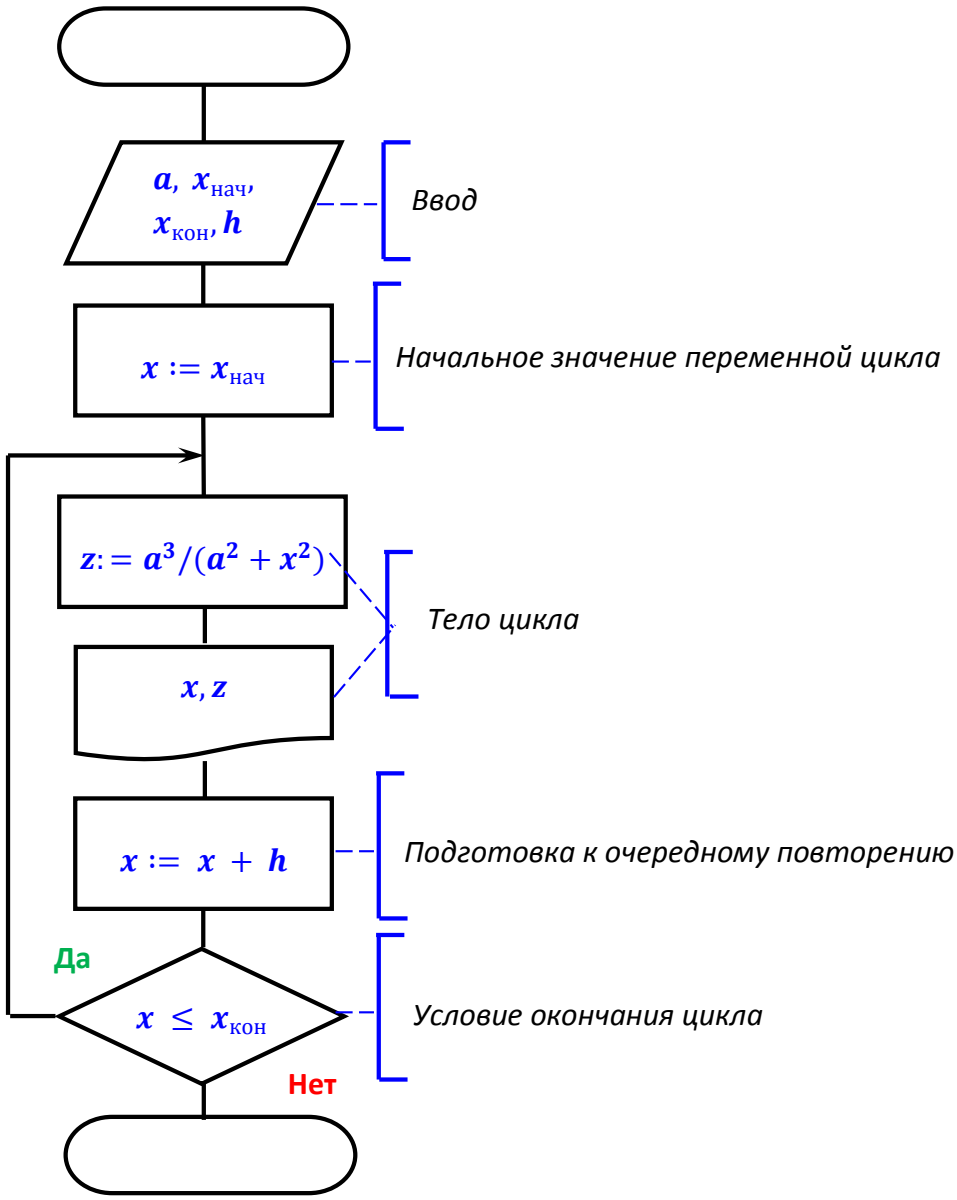


Рис. 5.8 - Использование цикла ***For*** для табуляции функции

В этом примере шаг приращения аргумента функции $h = 0{,}2$ дробный и не является порядковым типом. Следовательно, аргумент функции $\boldsymbol{x}$ в качестве управляющей переменной цикла ***For*** напрямую использовать нельзя.

Чтобы решить данную задачу с использованием цикла с заданным числом повторений для организации цикла ***For*** понадобится специально ввести управляющую переменную (счётчик циклов) $\boldsymbol{i}$.

Определим число повторений цикла ***n***:

$$\boldsymbol{n = \frac{(x_{кон} - x_{нач})}{h} + 1 = (5 + 2)/0{,}2 = 36.}$$

где $x_{нач}$ и $x_{кон}$ – соответственно начальное и конечное значение аргумента, h – шаг приращения x. Однако здесь есть одна тонкость: результат деления чисел - вещественное число, т.е. имеет дробную часть. ***Free Pascal*** самостоятельно не может решить проблему, что делать с этой дробной частью – отбросить или округлить. Существуют встроенные функции, позволяющие выбрать способ перехода от вещественного числа с плавающей запятой к целому, порядковому типу (Таблица 5.1).

Таблица 5.1- Функции приведения вещественного числа к целому типу

Функция	Тип аргументов	Тип результата	Действие	Примеры
round(x)	Вещественный	Целый	Округление числа	***round(12.75) = 13***
trunc(x)	Вещественный	Целый	Отсечение дробной части	***trunc(12.75) = 12***

С учётом этого, программа на Паскале может иметь вид:

```
Program DemoFor;  //Демонстрация использования цикла с заданным числом повторений
Var                 //объявления переменных
  a,                //параметр формулы
  x,                //текущее значение аргумента функции
  xN, xK,           //начальное и конечное значение аргумента
  h: real;          //шаг приращения аргумента
  i,                //счетчик циклов
  N: integer;       //количество циклов
Begin //начало блока DemoFor
  {Вывод назначения программы:}
  writeln(' Таблица значений функции z = a*a*a / (a*a + x*x)');
  {Диалог при вводе исходных данных:}
  write('Параметр  a : ');  readln(a);
  write('Начальное значение xN: ');  readln(xN);
  write('Конечное значение xK: ');   readln(xK);
  write('Шаг табулирования  h : ');  readln(h);
  writeln; {вывод пустой строки}
```

```
{Вывод заголовка таблицы;}
writeln('x':10, 'z':12);     //цифра после двоеточия – ширина зоны вывода
{Подготовка к циклу расчёта таблицы:}
N := trunc((xK - xN) / h + 1); {расчёт числа циклов}
x := xN;  {начальное значение аргумента функции}
{Организация цикла расчёта и вывода строки таблицы:}
for i := 1 to N  do //Заголовок цикла с заданным числом повторений
  begin {начало составного оператора с телом цикла}
    z := a*a*a / (a*a + x*x);     //расчёт очередного значения функции
    {Вывод строки таблицы}
    {1-я цифра после двоеточия – ширина зоны вывода,
    2-я – число знаков после запятой}
    writeln(x:10:2, z:12:4);
    x := x + h  {подготовка з аргумента функции к очередному циклу}
  end  {конец составного оператора – тела цикла For}
end. {Конец программы DemoFor}
```

5.3 Оператор цикла с предусловием *While*

Оператор цикла с предусловием ***While*** (***Пока***) позволяет многократно выполнять одни и те же действия, пока выполняется некоторое логическое условие (Рис. 5.9).

Рис. 5.9 - Синтаксическая диаграмма оператора цикла ***While***

Запомните!

В цикле ***While*** переменная цикла и шаг приращения (в отличие от цикла ***For***) могут быть и ***вещественными, т.е. иметь дробную часть***!

Чтобы цикл ***While*** был выполнен хотя бы один раз, необходимо, чтобы перед выполнением оператора ***While*** значение логического выражения ***Условие*** было истинно (***True***).

Типичными примерами использования цикла ***While*** являются итерационные вычисления до достижения заданной точности, поиск в массиве или в файле и т.п.

Оператор ***While*** выполняется следующим образом:

– Сначала вычисляется значение логического выражения **Условие**.

– Если значение выражения **Условие** равно ***True*** (условие выполняется), то выполняется ***Оператор*** (тело цикла).

– Если значение выражения **Условие** равно ***False*** (условие не выполняется), то выполнение цикла ***While*** завершается, и управление передаётся оператору, следующему за телом цикла.

– **Условие** вычисляется перед каждой итерацией цикла и каждый раз вновь проверяется до тех пор, пока **Условие** не станет ложным (***False***).

– Если при первой же проверке **Условие** ложно (***False***), цикл не выполнится ни разу.

В блок-схемах алгоритмов программ цикл с неизвестным числом повторений ***While*** изображается так, как показано на Рис. 5.11 или Рис. 5.11.

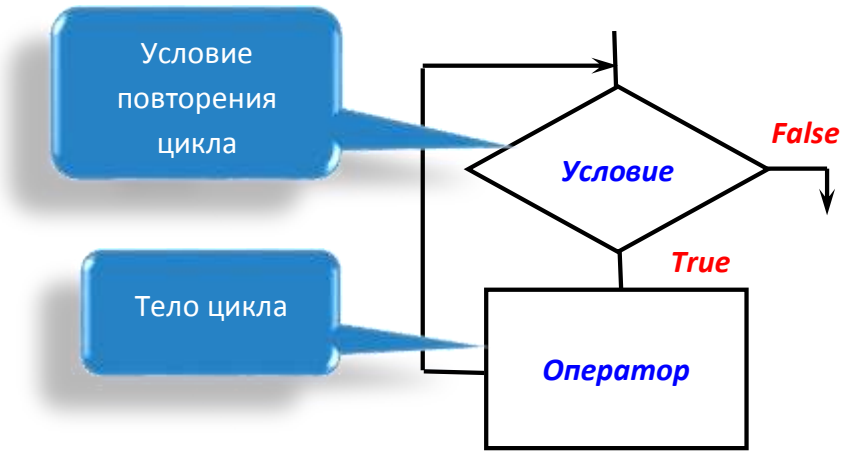


Рис. 5.11 – Цикл с предусловием ***While***

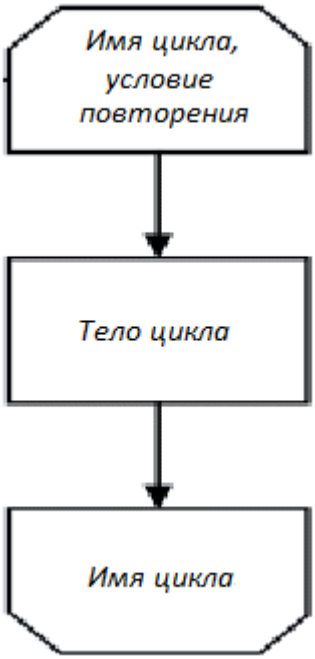


Рис. 5.11 - Изображение цикла ***While*** по ГОСТ 19.701-90

Как видно из синтаксической диаграммы (Рис. 5.9), оператор ***While*** повторяет *единственный оператор*, пока **Логическое условие** принимает значение ***True.*** Чтобы обойти это ограничение, как и в случае оператора цикла ***For*** применяют *составной оператор*.

Запомните!

Чтобы не "зациклиться", нужно, чтобы в теле цикла непременно менялись значения переменных, входящих в выражение ***Условие*** так, чтобы когда-либо значение выражения ***Условие*** неизбежно стало бы **ложным** (***False***).

Таблица 5.2 показывает варианты оптимальной записи для разных случаев применения оператора ***While***.

Таблица 5.2 - Наиболее популярные варианты записи оператора ***While***

Если тело цикла состоит из одного оператора	Если тело цикла включает более одного оператора

while <Условие> do
Оператор;
	while <Условие> do
	begin {составной оператор}
	Оператор 1;
	Оператор 2;
	…
	Оператор N
	end;

Пример 5.4

Вычислить с помощью цикла ***While*** сумму всех целых чисел от ***1*** до ***N*** (та же задача, что и в примере 5.1).

Решение *5.4*

```
…
Var
   i,  {управляющая переменная цикла}
   N, {количество суммируемых чисел}
   Summa: integer; {сумма всех целых чисел от 1 до N}
…
 {Обязательны начальные установки переменных, используемых в цикле!}
  Summa := 0;  {Начальное значение суммы}
  i := 1;          {Первое число}
  {Цикл подсчёта суммы чисел}
  While  i <= N do    {В заголовке цикла – условие его повторения}
    begin
      Summa := Summa + i;
      i := i + 1 {подготовка к следующей итерации}
    end;
  Writeln('Сумма чисел = ', Summa); {Выводим результат}
…
```

Пример 5.5

Вычислить и вывести на экран таблицу значений функции $z = a^3 / \left(a^2 + x^2\right)$, если ***x***, изменяется от ***–2*** до ***5*** с шагом ***0,2*** (та же задача, что и в примере 5.3). Написать фрагмент программы, решающий эту задачу с помощью цикла ***While***.

Решение 5.5

```
...
x := xN;                    {начальное значение переменной цикла}
while x <= xK  do           {проверка условия повторения цикла «пока»}
  begin                     {начало составного оператора (тела цикла while)}
    z := a*a*a / (a*a + x*x); {очередное значение функции}
    writeln(x:10:2, z:12:4); {вывод строки таблицы}
    x := x + h              {подготовка значения аргумента к очередному циклу}
end;                        {конец составного оператора (тела цикла while)}
...
```

5.4 Оператор цикла с послеусловием *Repeat*

Оператор цикла с послеусловием ***Repeat*** (до выполнения условия) (Рис. 5.12) также, как и оператор ***While***, используется в том случае, если необходимо выполнить повторные вычисления (организовать цикл), но число повторений во время разработки программы *неизвестно* и может быть определено только во время работы программы, т.е. определяется

Рис. 5.12 - Синтаксическая диаграмма оператора цикла ***Repeat***

В общем случае оператор ***Repeat*** записывается следующим образом:

```
Repeat
{группа повторяемых в цикле операторов - тело цикла}
  Оператор 1;
  Оператор 2;
   ...
  Оператор N
Until Условие Выхода из цикла;
```

где ***Условие*** - выражение логического типа, определяющее условие *завершения цикла*.

Оператор ***Repeat*** выполняется следующим образом:

- Сначала выполняются находящиеся между ***Repeat*** и ***until*** операторы тела цикла.
- Затем вычисляется значение выражения ***Условие***. Если ***Условие*** ложно (значение выражения ***Условие*** равно ***False***), то операторы тела цикла выполняются ещё раз.
- Если ***Условие*** истинно (равно ***True***), то выполнение цикла прекращается.

В блок-схемах алгоритмов программ цикл с неизвестным числом повторений ***Repeat*** изображается так, как показано на Рис. 5.13 или Рис. 5.14.

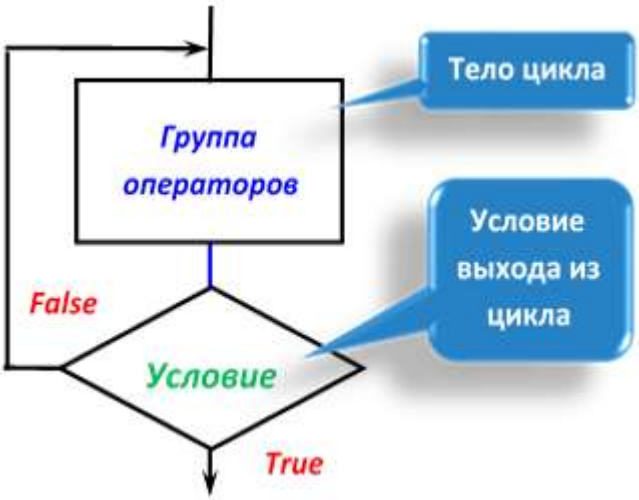


Рис. 5.13 - Цикл с послеусловием «До выполнения условия»

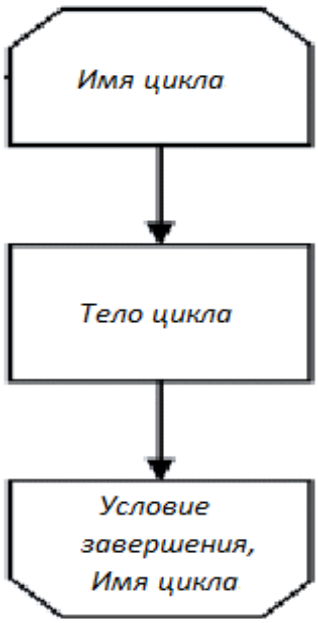


Рис. 5.14- Изображение цикла ***Repeat .. until*** по ГОСТ 19.701-90

Таким образом, операторы цикла, находящиеся между ***Repeat*** и ***until***, повторяются, пока ***Условие*** ложно (в отличие от оператора ***While***).

Запомните!

В циклах ***While*** и ***Repeat .. until*** переменная цикла и шаг приращения (в отличие от цикла ***For***) могут быть и ***вещественными, т.е. иметь дробную часть!***

Так как проверяемое ***Условие*** выхода из цикла располагается *в конце тела цикла*, то операторы, находящиеся в теле цикла, в отличие от цикла ***While***, всегда выполняются как минимум один раз.

Как и в случае цикла ***While***, в теле цикла обязательно должна меняться некоторая переменная, используемая в ***Условии***, чтобы оно когда-либо выполнилось.

Заметим, что в цикле ***Repeat*** и ***until*** составной оператор для объединения повторяемой группы операторов *не требуется*.

Пример 5.6.

Вычислить с помощью цикла ***repeat .. until*** сумму всех целых чисел от ***1*** до ***N*** (та же задача, что и в примере 5.1).

Решение 5.6

```
...
Var
  i,                      //управляющая переменная цикла
  N,                      //количество суммируемых чисел
  Summa: integer;         //сумма всех целых чисел от 1 до N
...
  Summa := 0;  //Начальное значение суммы
  i := 1;          //Первое число (и счётчик циклов!
  Repeat                  //Цикл подсчета суммы чисел
    Summa := Summa + i;   //Накапливаем сумму чисел
    i := i + 1            //подготовка к очередной итерации
  until i > N;            //Условие завершения цикла

Writeln('Сумма чисел = ', Summa);  //Выводим результат
...
```

Пример 5.7

Вычислить и вывести на экран таблицу значений функции $z = a^3 / \left(a^2 + x^2\right)$, если ***x*** изменяется от ***–2*** до ***5*** с шагом ***0,2*** (та же задача, что и в примере 5.3). Написать фрагмент программы, решающий эту задачу с помощью цикла ***repeat .. until***.

Решение 5.7

```
...
  x := xN;                    //Начальное значение переменной цикла
  repeat                      //Заголовок цикла «до выполнения условия»
    z := a*a*a / (a*a + x*x);  //Расчёт очередного значения функции
    writeln(x:10:2, z:12:4);  //вывод строки таблицы
    x := x + h                //подготовка к очередному циклу
  until x > xN;               //проверка условия окончания цикла
...
```

5.5 Досрочное завершение циклов *While*, *Repeat* и *For*

Для гибкого управления циклическими операторами ***While***, ***Repeat*** и ***For*** в состав ***Free Pascal*** включены две процедуры-команды ***Break*** и ***Continue***.

Команда досрочного выхода ***Break*** (англ. *прервать*) позволяет досрочно выйти из любого типа цикла, не дожидаясь выполнения условия выхода. Такое бывает, например, когда при выполнении тела цикла обнаруживается ошибка, после которой дальнейшая работа цикла не имеет смысла.

Процедура ***Continue*** (англ. *продолжать*) применяется, когда в текущей итерации цикла необходимо пропустить все команды до конца тела цикла. При этом сам цикл прерываться не должен, а условия продолжения или выхода должны вычисляться обыкновенным образом. Например, в результате дополнительной проверки в теле цикла выясняется, что необходимо перейти к следующей итерации, не исполняя дальнейшие команды тела цикла.

Введение в язык этих команд практически исключает необходимость использования операторов безусловного перехода ***GoTo***.

5.6 Сравнение работы операторов *While*, *Repeat* и *For*

Подытожим отличия и особенности работы с различными операторами цикла.

Цикл со счётчиком For:

- Начальная установка переменной (счётчика) циклов до заголовка не требуется
- Изменение в теле цикла значений переменных, стоящих в заголовке цикла не допускается
- Количество итераций цикла неизменно и точно определяется значениями нижней и верхней границ и шага изменения счётчика циклов (+1 или -1)
- Нормальный ход работы цикла может быть нарушен оператором ***GoTo*** (передача управления на метку вне цикла) или процедурами ***Break*** и ***Continue***
- Цикл может не выполниться ни разу, если шаг цикла будет изменять значение счётчика от нижней границы в направлении, противоположным верхней границе

Цикл с предусловием *While* работает, пока Условие истинно

- Цикл завершается, когда **Условие** становится ложным (***False***)
- Цикл может не выполниться *ни разу*, если при входе в цикл исходное значение условия ***True***
- Если в теле цикла требуется повторять более одного оператора, то необходимо использовать ***Составной оператор***

Цикл с постусловием *Repeat .. until* работает пока **Условие ложно**

- Цикл завершается, когда **Условие** становится истинным (***True***)
- Цикл обязательно выполняется как минимум *один раз*
- Независимо от количества операторов в теле цикла ***использование составного оператора не требуется.***

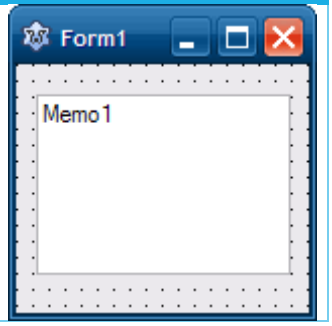
Циклы с неизвестным числом повторения *While* и *Repeat..until*:

- До начала цикла должны быть сделаны начальные установки переменной цикла, чтобы обеспечить корректный вход в цикл
- В теле цикла должны присутствовать операторы, изменяющие переменные условия так, чтобы цикл через некоторое число итераций завершился
- Шаг цикла может быть дробным
- Нормальный ход работы цикла может быть нарушен оператором ***GoTo*** (передача управления на метку вне цикла) или процедурами ***Break*** и ***Continue***

6 Компоненты для вывода информации

6.1 Компонент TMemo (Ввод, отображение и редактирование текста)

Основное предназначение компонента ***TMemo*** – работа с большим количеством строк (ввод, отображение и редактирование текстового материала).

Место в палитре	Вид
	

Окно компонента ведёт себя как обычный текстовый редактор типа «Блокнот», т.е. доступны все стандартные функции редактирования (выделение, копирование, вставка, удаление и пр.). Для работы с буфером обмена можно использовать общепринятые горячие клавиши: $Ctrl - X$ — вырезать, $Ctrl - C$ — копировать; $Ctrl - V$ — вставить.

Информация в $TMemo$ содержится в виде совокупности (массива) строк типа $TStrings$. Каждый элемент массива содержит ровно одну строку. Доступ к отдельной строке осуществляется с помощью свойства $Lines$ по ее номеру (индексу). Индекс указывается, как и положено для массивов, в квадратных скобках. Нумерация строк начинается с нуля. Общее количество строк содержится в свойстве $Lines.Count$.

Если строка не умещается целиком в окне, то можно установить свойство $WordWrap = true$ и тогда не уместившаяся часть строки будет автоматически перенесена на следующую строку.

Таблица 6.1 - Основные свойства компонента $TMemo$ (ввод, отображение и редактирование текстов)

Свойство	Описание
Name	**Имя компонента.** Используется в программе для доступа к компоненту и его свойствам, в частности для доступа к тексту, введённому в поле редактирования.
Text	**Текст, находящийся в поле *Memo***. Рассматривается как единое целое.
Lines	**Массив строк, соответствующий содержимому поля.** Доступ к строке осуществляется по номеру. Строки нумеруются с нуля.
Lines.Count	**Количество строк текста в поле *Memo*.**
Left	Расстояние от левой границы поля до левой границы формы.
Top	Расстояние от верхней границы поля до верхней границы формы.
Width, Height	**Ширина, высота поля.**
Font	**Шрифт**, используемый для отображения вводимого текста.
ParentFont	Признак наследования свойств шрифта родительской
WantReturns	Клавиша для ввода конца строки: *True* – клавиша Enter; *False* – сочетание клавиш *Ctrl + Enter*.
WordWrap	**Переход в начало следующей строки при вводе длинных строк**: *True* –производится автоматически; *False* – не производится. При включённой горизонтальной полосе прокрутке это свойство игнорируется.
ScrollBar	**Использование полосы прокрутки**, если текст большой и не помещается в компоненте *Memo*: ssNone – Нет полосы прокрутки; ssHorizontal – Установлена горизонтальная прокрутка; ssVertical – Установлена вертикальная прокрутка; ssBoth – Установлены две полосы прокрутки.
ReadOnly	**Разрешает или запрещает редактирование текста** (программно текст все равно можно добавлять).

Для сохранения содержимого текстового поля $\boldsymbol{Memo}$ в файл используется функция $\boldsymbol{SaveToFile}(\text{'}\boldsymbol{mytetxt.txt}\text{'})$, а для извлечения - $\boldsymbol{LoadFromFile}(\text{'}\boldsymbol{mytetxt.txt}\text{'})$, где $\boldsymbol{mytetxt.txt}$ – текстовый файл расположенный в каталоге программы.

Если строка не умещается целиком в окне, то можно установить свойство $\boldsymbol{WordWrap} = \boldsymbol{true}$ и тогда не уместившаяся часть строки будет автоматически перенесена на следующую строку.

Можно также установить полосы прокрутки свойством $\boldsymbol{ScrollBars}$. Возможные значения свойства:

– $\boldsymbol{ssNone}$ - установлено по умолчанию, полосы прокрутки отсутствуют.

– $\boldsymbol{ssVertical}$ - установить вертикальную полосу прокрутки.

– $\boldsymbol{ssHorizontal}$ - установить горизонтальную полосу прокрутки.

– $\boldsymbol{ssBoth}$ - установить и вертикальную и горизонтальную полосы.

– $\boldsymbol{ssAutoVertical}$ - вертикальная полоса в окне компонента видна, но не доступна, пока окно не заполнится по вертикали.

– $\boldsymbol{ssHorizontal}$ - горизонтальная полоса в окне компонента видна, но не доступна, пока окно не заполнится по горизонтали.

– $\boldsymbol{ssAutoBoth}$ - объединяет два предыдущих значения.

Запретить пользователю редактирование можно, установив свойство $\boldsymbol{ReadOnly} = \boldsymbol{true}$.

Добавление новой строки при вводе данных с клавиатуры осуществляется нажатием клавиши $\boldsymbol{Enter}$, при этом свойство $\boldsymbol{WantReturns}$ должно быть установлено равным $\boldsymbol{true}$. Если $\boldsymbol{WantReturns} = \boldsymbol{false}$, то для перехода на новую строку необходимо нажать $\boldsymbol{Ctrl} + \boldsymbol{Enter}$.

Свойство $\boldsymbol{SelStart}$ указывает начало выделенного текста, а $\boldsymbol{SelLength}$ - длину выделенного текста (количество символов).

В свойстве $\boldsymbol{Text}$ весь набор строк представляется в виде одной строки с разделителями возврат каретки и перенос строки ($\#\mathbf{13}\#\mathbf{10}$) между строками.

Посмотрим, как программно заполнять поле ввода компонента. Чтобы добавить строку в $\boldsymbol{TMemo}$ необходимо воспользоваться методами:

– $\boldsymbol{function\ Add}\ (\boldsymbol{const\ S}: \boldsymbol{string}) : \boldsymbol{integer}$ - добавляет строку $\boldsymbol{S}$ в конец набора строк $\boldsymbol{TMemo}$ и возвращает ее индекс.

– **procedure Append (const S: string);** - просто добавляет строку S в конец набора строк.

Чтобы добавить целый набор строк, например, из другого компонента $\boldsymbol{TMemo}$, можно применить методы:

– $\boldsymbol{procedure\ AddStrings}(\boldsymbol{TheStrings}: \boldsymbol{Tstrings})$; - где The- Strings набор строк типа $\boldsymbol{TStrings}$, добавляет этот набор строк к существующему.

– $\boldsymbol{procedure\ Assign}\ (\boldsymbol{Source} : \boldsymbol{TPersistent})$; - полностью очищает содержимое $\boldsymbol{TMemo}$ и загружает новый набор строк из $\boldsymbol{Source}$.

Для вставки строки в произвольное место списка строк существует метод $\boldsymbol{procedure\ Insert}(\boldsymbol{Index}: \boldsymbol{integer}; \boldsymbol{const\ S}: \boldsymbol{string})$; - где $\boldsymbol{Index}$ номер (индекс) строки куда вставляется строка $\boldsymbol{S}$. При этом старая строка не исчезает, а сдвигается вниз

вместе со всеми нижележащими строками (их индексы автоматически увеличиваются на единицу).

Заменить содержимое какой-либо строки можно простым оператором присваивания, например, чтобы заменить содержимое строки с индексом k достаточно записать оператор:

$$Memo1.Lines[\,k\,] := '\text{Содержимое заменяемой строки}';$$

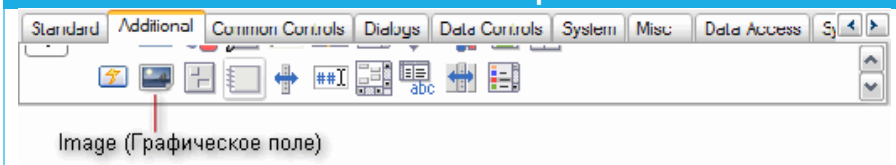
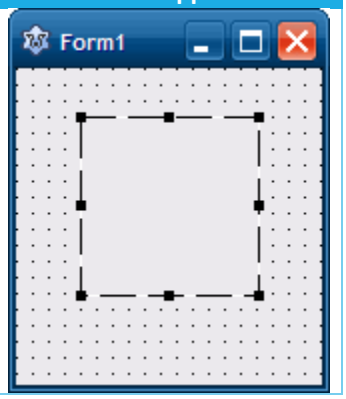
Добавлять строки оператором присваивания тоже можно, но при одном условии, нельзя применять этот способ во время создания формы, т.е. в обработчике ***OnCreate*** формы.

Чтобы удалить строку применяется метод

$$Delete(Index: integer);$$

6.2 Компонент Изображение (*Timage*)

Компонент (***TImage***) обеспечивает вывод на поверхность формы графического рисунка в ***bmp***–формате Он поддерживает многочисленные форматы графических файлов. Но для того, чтобы компонент можно было использовать для отображения иллюстраций в формате *JPG*, надо подключить модуль ***JPEG*** – указать имя модуля в директиве ***uses***.

Место в палитре	Вид
	

Для загрузки рисунка в поле ***Image*** на этапе разработки интерфейса приложения нужно присвоить свойству ***Picture*** (Таблица 6.2) файл, содержащий рисунок.

Таблица 6.2 - Основные свойства компонента ***TImage*** (вывод иллюстраций)

Свойство	Описание
Picture	Иллюстрация, которая отображается в поле компонента.
Width, Height	Размер компонента. Если размер компонента меньше размера иллюстрации, и значение свойств ***AutoSize***, ***Strech*** и ***Proportional*** равно ***False***, то изображается часть иллюстрации.
Proportional	Признак автоматического масштабирования картинки без искажения. Чтобы масштабирование было выполнено, значение свойства ***AutoSize*** должно быть ***False***.
Strech	Признак автоматического масштабирования (сжатия или растяжения) иллюстрации в соответствии с реальным размером компонента. Если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена. Обратите внимание: свойство ***Stretch*** не влияет на файлы рисунков типа *.ico*.

Свойство	Описание
AutoSize	Признак автоматического изменения размера компонента в соответствии с реальным размером иллюстрации.
Center	Признак определяет расположение картинки в поле компонента по горизонтали, если ширина картинки меньше ширины поля компонента. Если значение свойства равно $\boldsymbol{False}$, то картинка прижата к правой границе компонента, если $\boldsymbol{True}$ – то картинка располагается по центру.
Visible	Отображается ли компонент и соответственно, иллюстрация на поверхности формы.
Canvas	Поверхность, на которую можно вывести графику.

Для присвоения изображению файла с рисунком во время выполнения приложения используется метод $\boldsymbol{LoadFromFile}$, принадлежащий объекту $\boldsymbol{Picture}$.

Например, для вывода в изображение $\boldsymbol{imgIxample}$ файла рисунка $\boldsymbol{myPicture.jpg}$ во время выполнения используется следующий оператор:

$$\boldsymbol{imgExample.Picture.LoadFromFile}(\text{'}\boldsymbol{myPicture.jpg}\text{'})$$

7 Список литературы

1. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В. Free Pascal и Lazarus: Учебник по программированию / Е.Р. Алексеев, О.В. Чеснокова, Т.В. Кучер. - М.: Издательский дом ДМК-пресс, 2010. - 440 с.
2. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В.. Самоучитель по программированию на Free Pascal и Lazarus.. - Донецк: ДонНТУ, Технопарк ДонНТУ УНИТЕХ, 2011. - 503 с.
3. Кетков Ю.Л. Свободное программное обеспечение. FREE PASCAL для студентов и школьников / Ю.Л. Кетков, А.Ю. Кетков. — СПб.: БХВ-Петербург, 2011. — 384 с.
4. Мансуров К.Т. Основы программирования в среде Lazarus. - М.: Нобель пресс, 2013. – 772 с.
5. Фаронов В.В. Turbo Pascal. Наиболее полное руководство (в подлиннике). — СПб.: БХВ-Петербург, 2004. — 1056 с.
6. Фленов М.Е. Библия Delphi. — 2-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2008. - 800 с.
7. Lazarus Tutorial/ru [Электронный ресурс] // База знаний о Free Pascal, Lazarus и родственных проектах: [сайт]. URL: http://wiki.freepascal.org/Lazarus_Tutorial/ru
8. Программирование на Lazarus [Электронный ресурс] // «ИНТУИТ» Национальный открытый университет: [сайт]. URL: http://www.intuit.ru/studies/courses/13745/1221/lecture/23276?page=1
9. ОС ТУСУР 01-2013 (СТО 02069326.1.01-2013). Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления. - Томск: ТУСУР, 2013. – 57 с.
10. Кобрин Ю.П. Основные понятия языка Free Pascal. - Томск: ТУСУР, кафедра КИПР, 1916. - 37 с/.