КИПР

ТУСУР

Кафедра конструирования и производства радиоаппаратуры

УТВЕРЖДАЮ

Заведующий кафедрой КИПР

_____В.Н. ТАТАРИНОВ

“___” _____2012 г.

Циклические программы

Лабораторная работа по дисциплинам **«**Информатика**»** для студентов специальностей 211000.62 «Конструирование и технология электронных средств» (бакалавриат) и 162107.65 «Информатика и информационные технологии» (специалитет)

Разработчик:

Доцент кафедры КИПР

_____**Ю.П. Кобрин**

Томск 2012

СОДЕРЖАНИЕ

1 Цели работы

✓ Изучение работы операторов цикла.

✓ Освоение приемов программирования и отладки простейших циклических алгоритмов.

✓ Закрепление навыков практической работы в интегрированной среде ***Borland Pascal.***

2 Порядок выполнения работы

1) Изучить операторы организации циклов, используя материал раздела «Методика разработки циклических программ на языке Pascal» а также, при необходимости, дополнительной литературы [1] [2] [3] [4] [5] [6].

2) Ответить письменно на контрольные вопросы.

3) Используя условия задания предыдущей лабораторной работы [4] составить три варианта циклической программы[1] (с циклами типа ***for, while*** и ***repeat .. until***), позволяющей рассчитать таблицу изменения напряжения на сопротивлении нагрузке $U_{RH}(t)$, если время меняется в диапазоне $0 < t < T_u$ с шагом Δt. Для всех вариантов $T_u = 6 * A$. Сделать выводы о целесообразности использования разных вариантов цикла для решения поставленной задачи.

4) Войти в свой личный каталог, загрузить и настроить систему ***Borland Pascal 7.0***.

5) Ввести разветвленную программу, сделанную на прошлом лабораторном занятии.

6) Сделать копию программы, записав ее под новым именем (варианта циклической программы) с помощью команды ***Save as*** ...

7) Ввести разработанный Вами вариант циклической программы, корректируя и дополняя его.

8) Добиться с помощью отладки, чтобы программа дала правильные результаты [9].

9) Повторить пункты 6 - 8 для остальных вариантов цикла.

10) Оформить отчет и защитить его у преподавателя.

3 Контрольные вопросы

Ответьте на следующие контрольные вопросы:

1) Какие управляющие конструкции повторения поддерживаются в языке ***Borland Pascal***?

2) Как записывается и как работает оператор ***for***?

3) Какие ограничения накладываются на использование оператора ***for***?

4) Как работает оператор цикла ***while***?

5) В чем отличие оператора ***while*** от оператора ***repeat***?

6) Какие существуют отличия и особенности при работе с операторами ***while, repeat*** и ***for***?

---

[1] Каждый новый вариант программы легко создать на основе коррекции предыдущего текста программы, копию которой следует сохранить в своем каталоге под новым именем.

7) Как программируются циклические алгоритмы с явно заданным числом повторений цикла?

8) Как программируются циклические алгоритмы с незаданным числом повторений цикла?

4 Защита отчета

Отчет должен состоять из следующих разделов:

- ✓ Тема и цель работы.
- ✓ Индивидуальное задание.
- ✓ Ответы на контрольные вопросы.
- ✓ Текст программы и вводимые тестовые исходные данные.
- ✓ Результаты выполнения программы.
- ✓ Выводы.

При защите отчета по работе для получения зачета студент должен:

- ✓ уметь отвечать на контрольные вопросы;
- ✓ обосновать структуру выбранного алгоритма и показать его работоспособность;
- ✓ уметь пояснять работу программы;
- ✓ продемонстрировать навыки работы в среде ***Borland Pascal***.

5 Методика разработки циклических программ на языке Pascal

5.1 Общие понятия

Алгоритмы решения многих задач являются циклическими, т.е. для достижения результата определенная последовательность действий должна быть повторена несколько раз. Использование циклов позволяет существенно сократить схему алгоритма и длину соответствующей ему программы.

Например, программа контроля знаний выводит вопрос, принимает ответ, добавляет оценку за ответ к сумме баллов, затем повторяет это действие еще и еще раз, и так до тех пор, пока испытуемый не ответит на все вопросы.

Другой пример. Для того чтобы найти фамилию человека в списке, надо проверить первую фамилию списка, затем вторую, третью и т.д. до тех пор, пока не будет найдена нужная фамилия или не будет достигнут конец списка.

Алгоритм, в котором есть последовательность операций (группа инструкций), которая должна быть выполнена несколько раз, называется ***циклическим***, а сама последовательность операций именуется ***циклом***.

Для организации цикла используется переменная, называемая ***параметром*** *(****управляющей переменной****)* ***цикла***. По значению этой переменной определяется момент выхода из цикла на продолжение программы.

Любой циклический алгоритм начинается с ***блока установки параметра цикла*** (блок ***1***) ***в исходное состояние*** (*рис. 5.1*). В циклической программе, следующей за этим блоком можно выделить три части:

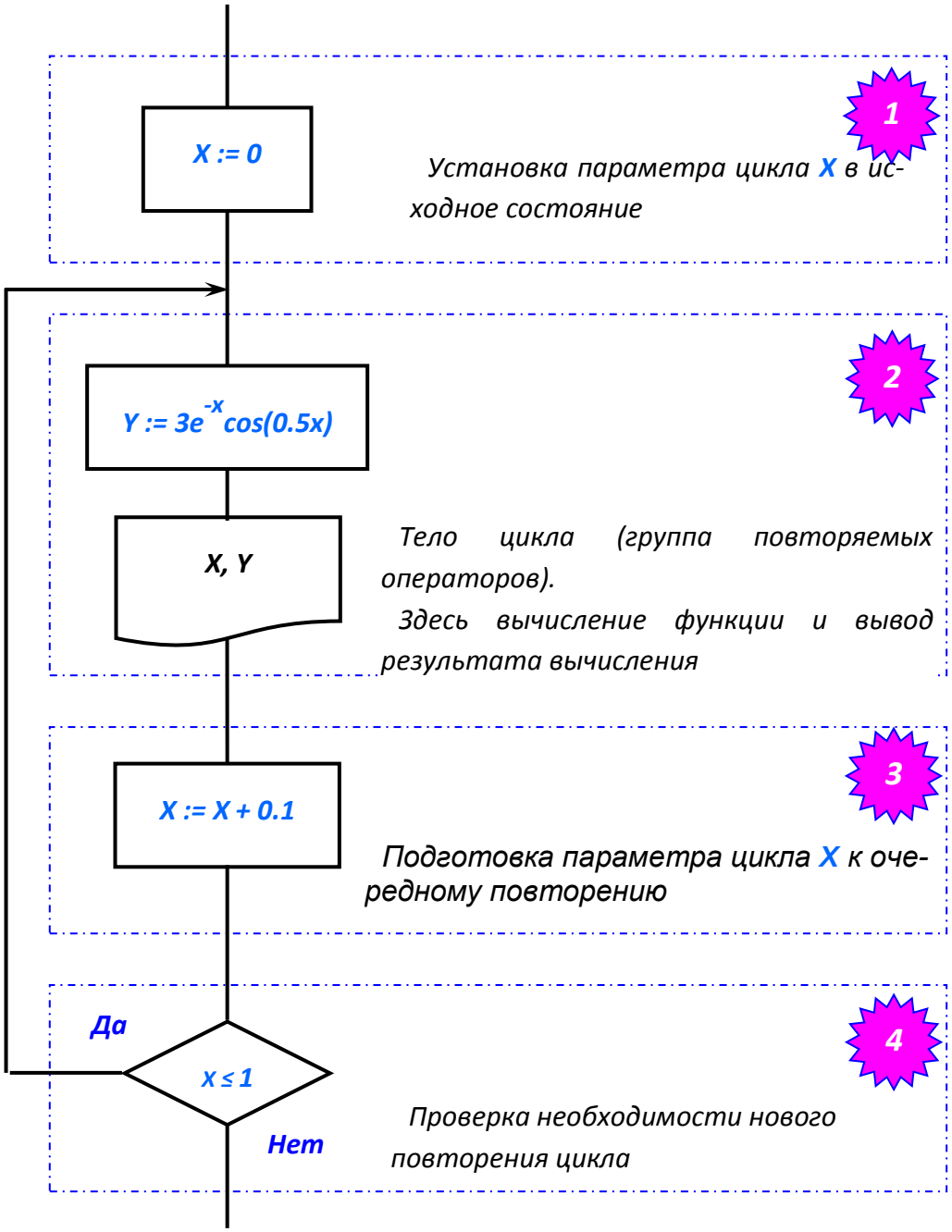

Рисунок 5.1 - Блок-схема циклической программы вычисления значений функции ***y = f(X)*** в диапазоне ***0 .. 1*** с шагом ***0.1***

- ✓ ***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 участков ***2, 3, 4*** в принципе может быть *любым*.

Составляя циклические программы необходимо помнить, что результаты вычислений, присваиваемые переменной в левой части формулы оператора присвоения на каждом цикле, записываются *вместо предыдущих значений*. Поэтому их нужно своевременно выводить на экран или принтер, запоминать (например, в качестве элемента массива) или использо-

вать как-то иначе. Так, например, в алгоритме на *рис. 5.1* выполняется циклическое вычисление функции

$$y = 3*e^{-x}*Cos(0.5*x) \text{ при } 0 \le x \le 1.$$

Результаты вычислений выводятся на документ в каждом цикле в виде строки таблицы.

Фрагмент программы, реализующий этот циклический алгоритм на языке ***Pascal*** (без использования специальных операторов цикла), может выглядеть так:

```
...
Label
   Zikl;                  {Описание метки начала тела цикла}

Var
   x,   {аргумент функции y = f(x), параметр (переменная) цикла}
   y: real;       {текущее значение функции y = f(x) каждом цикле}

...
x := 0;           {установка параметра цикла в исходное состояние}

Zikl:   {метка начала тела цикла}

   y := 3 * exp(-x) * cos(0.5 * x);          {вычисление функции y = f(x)}

   Writeln('x =  ', x:3:1, '    y = ', y:10:3);  {вывод на экран строки таблицы}

   x := x + 1;              {подготовка параметра цикла к очередному повторению}

   if x <= 1.0              {проверка необходимости нового повторения}
      then Goto Zikl; {перейти на начало тела цикла,
                       если значение параметра цикла x в заданном промежутке}
...
```

Бесспорно, подобный вариант циклической программы не очень нагляден. Кроме того, в программе *используются метки*, что противоречит технологии *структурного программирования*. Вместе с тем, в языке ***Borland Pascal*** существуют более эффективные операторы организации циклов (*рис. 5.2*), чем использование примитивных операторов с метками.

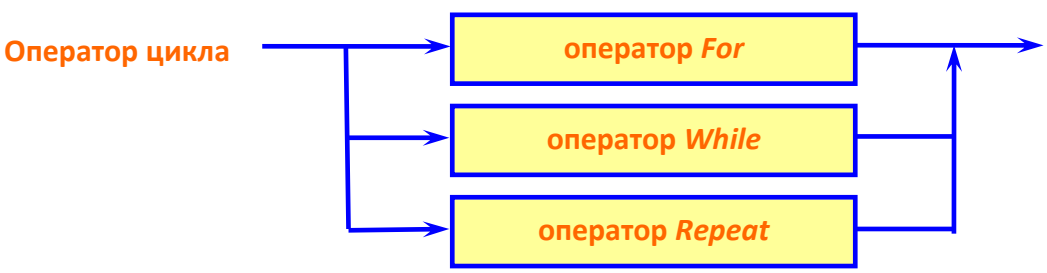

Рисунок 5.2 – Операторы повторения в **Borland Pascal**

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 счетчиком ***For***;

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

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

5.2 Оператор цикла с заданным числом повторений (со счетчиком) For

Оператор цикла со счетчиком подходит для программирования только таких циклических программ, в которых *до выполнения цикла* известны начальное и конечное значения счетчика повторений цикла.

Для реализации циклов с заданным числом повторений в ***Паскале*** имеется оператор

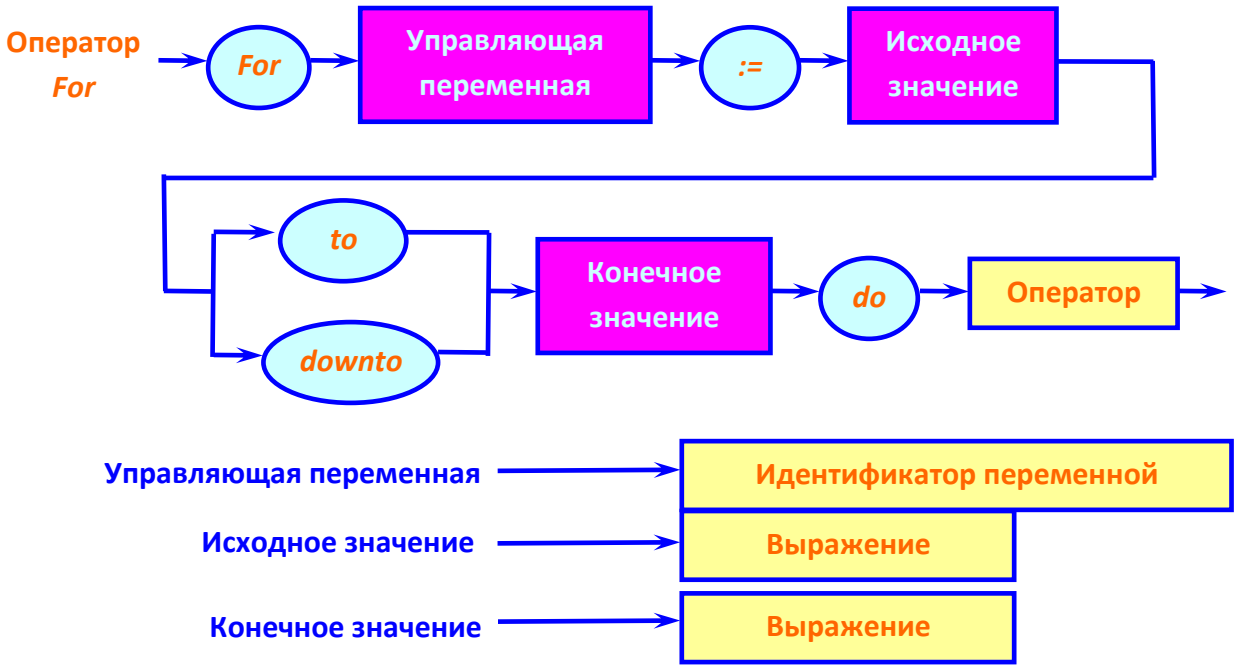

Рисунок 5.3 - Синтаксическая диаграмма оператора цикла ***For***

For, синтаксическая диаграмма которого приведена на *рис. 5.3.*

Варианты записи оператора ***For*** можно представить одним из двух способов.

1. Если счетчик при выполнении цикла *наращивает свое значение на* ***+1***:

```
For <Управляющая переменная> := <Исх. значение> to <Кон. значение>
  do <Оператор>;
```

2. Если счетчик при выполнении цикла *уменьшает свое значение на* ***-1:***

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

В обоих вариантах записи оператора ***For*****:**

✓ **<** Управляющая переменная> **::=** переменная **порядкового типа**[2];

✓ <Исходное значение> и <Конечное значение> – выражения, типы которых должны быть совместимы по присваиванию с типом управляющей переменной цикла. Значения этих выражений определяют, соответственно, начальное и конечное значение управляющей переменной цикла. Они рассчитываются ***только один раз*** при входе в оператор ***For***, и сохраняются неизменными на протяжении всего процесса его выполнения.

✓ **<** ***Оператор*****>** – *единственный* повторяемый оператор[3]. Для снятия ограничения по количеству операторов следует употреблять составные операторы ***begin .. end***.

В блок-схемах алгоритмов программ цикл с заданным числом повторений изображается так, как показано на *рис. 5.4*.

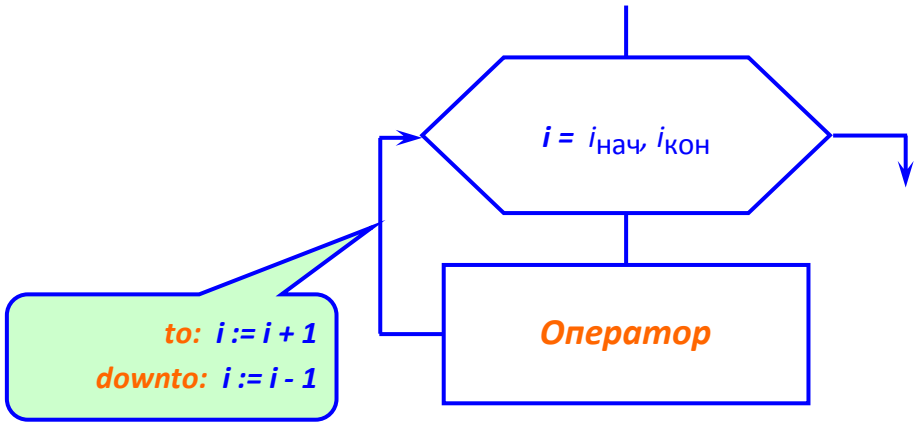

Рисунок 5.4 - Цикл с заданным числом повторений

Рассмотрим работу *первого варианта цикла* ***For***, Пусть значения выражений <Исходное значение> и <Конечное значение> равны, соответственно, $i_{нач}$ и $i_{кон}$ (см. *рис.5.4*) и переменой цикла присвоено начальное значение $i_{нач}$. После этого производится проверка, *не превосходит ли* содержимое переменной цикла конечное значение $i_{нач}$. Если не превосходит, выполняется оператор (тело цикла). Далее значение переменной *увеличивается на единицу* (шаг приращения переменной цикла[4] задан неявно)**.** Процесс, включающий проверку и выполнение тела цикла, повторяется до тех пор, пока проверка не даст положитель-

---

[2] Вспомним, что к порядковому типу относятся данные типа ***shortint, integer, word, byte*** и ***Char***. К любому из них можно применить функцию ***Ord(X)***, возвращающую порядковый номер значения выражения ***X***.

[3] Чтобы обойти это ограничение, как и в случае разветвленного алгоритма, следует применить составной оператор (операторные скобки).

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

ный результат. В этом случае оператор ***For*** завершается, переменная цикла объявляется неопределенной и осуществляется переход на оператор, следующий за оператором ***For.***

Второй вариант цикла ***For*** противоположен по действию рассмотренному выше. В нем проверка определяет: *не меньше ли* значение переменной цикла конечного значения, и лишь в случае отрицательного ответа выполняется тело цикла. Если конечное значение больше начального, тело цикла не выполняется ни разу. При каждом повторении переменная цикла здесь *уменьшается на единицу*.

Пример 5.1.

Ввести с клавиатуры число ***N*** и вычислить с помощью цикла ***For*** сумму всех целых чисел от ***1*** до ***N.***

Решение.

```
...
Var
   i,  {счетчик циклов}
  N, {количество суммируемых чисел}
  Summa: integer; {сумма всех целых чисел от 1 до N}
 ...
  Write('N = ');
  Readln(N); {Вводим  N}

  Summa := 0;  {Начальное значение суммы}
  For  i :=  1 to N do   {Цикл подсчета суммы}
     Summa := Summa + i;
 Writeln('Сумма чисел = ', Summa); {Выводим результат}
 ...
```

***Пример 5.2* (с ошибкой).**

После окончания тела цикла наращивание/уменьшение значения цикла происходит автоматически. Следовательно, специального оператора для увеличения значения счетчика типа ***i := i + 1*** не требуется, Более того, подобный оператор приведет к *непредсказуемой работе цикла* и считается плохим стилем программирования Например, ошибочный фрагмент решения предыдущего примера 5.1, 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!
end;
...
```

Пример 5.3.

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*.

Решение.

Возможная схема алгоритма приведена на *рис. 5.5*.

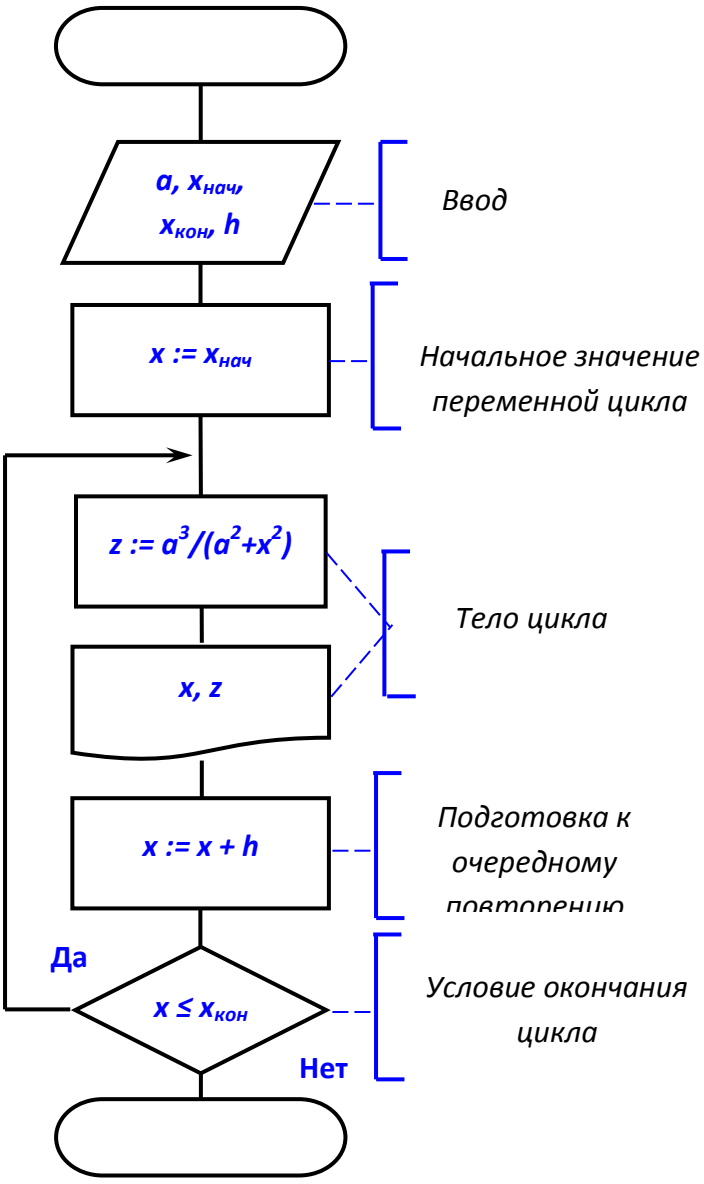

Рисунок 5.5 - Использование цикла ***For*** для табуляции функции

В этом примере шаг приращения аргумента функции (*0,2*) не является порядковым типом. Следовательно, аргумент функции ***x*** в качестве управляющей переменной цикла ***For*** использовать нельзя.

Чтобы решить данную задачу с использованием цикла с заданным числом повторений понадобится для организации цикла ***For*** специально ввести управляющую переменную (счетчик циклов) ***i*** .

Число повторений цикла ***n*** несложно вычислить по элементарной формуле

$$n = (x_{кон} - x_{нач})/h + 1 = (5 + 2)/0.2 + 1 = 36,$$

где $x_{нач}$ и $x_{кон}$ – соответственно начальное и конечное значение аргумента, ***h*** – шаг приращения.

С учетом этого, программа на Паскале может иметь вид:

```
Program DemoFor;  {Демонстрация использования цикла с заданным числом повторений}
Uses {подключаем модуль CRT }
   CRT;
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 программы DemoFor}
    {Подготавливаем экран}
    TextColor(Yellow);                {желтые символы}
    TextBackGround(Blue);   {голубой фон}
    ClrScr;                           {очищаем экран}
    {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 расчета таблицы:}
    N := (xK - xN) / h + 1; {расчет числа циклов}
```

```
  x := xN;  {начальное значение аргумента функции}

  {Организация цикла расчета и печати строки таблицы:}
   for i := 1 to N  do {Заголовок цикла с заданным числом повторений}
     begin {начало составного оператора, объединяющего группу операторов, входящих в тело цикла}
        z := a*a*a / (a*a + x*x);     {расчет очередного значения функции}
       {вывод строки таблицы}
        writeln(x:10:2, z:12:4);  {1-я цифра после двоеточия – ширина зоны вывода, 2-я – число знаков после запятой}
          x := x + h  {подготовка значения 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. 5.6*). Типичными примерами использования цикла ***While*** являются вычисления с заданной точностью, поиск в массиве или в файле.

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

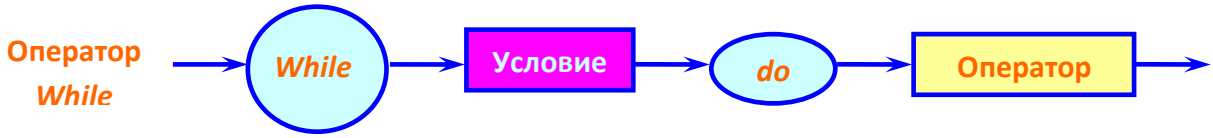

Рисунок 5.6 - Синтаксическая диаграмма оператора цикла ***While***

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

✓ Если значение выражения ***Условие*** равно ***False*** (условие не выполняется), то на этом выполнение цикла ***While*** завершается.

✓ Если значение выражения условие равно ***True*** (условие выполняется), то выполняется ***Оператор*** тела цикла. После этого снова проверяется выполнение условия. Если условие выполняется, то цикл выполняется еще раз. И так до тех пор, пока условие не станет ложным (***False***).

В блок-схемах алгоритмов программ цикл с неизвестным числом повторений **«Пока»** изображается так, как показано на *рис. 5.7*.

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

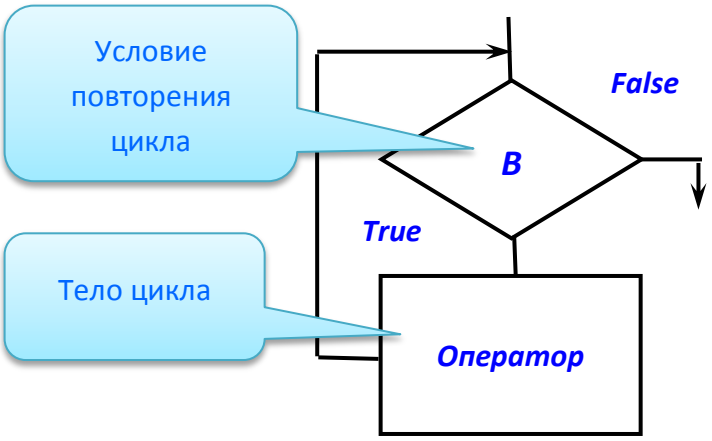

Рисунок 5.7 – Цикл с предусловием «Пока»

Обобщенные формы записи оптимальной записи для стандартных случаев оператора ***While*** приведены в *табл. 5.1.*

Таблица 5.1 - Наиболее популярные варианты записи опреаторa ***While***

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

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

Как видно из синтаксической диаграммы *(см. рис. 5.6)* и *рис. 5.7*, оператор ***While*** повторяет *единственный оператор*, пока логическое условие ***B*** принимает значение ***True.*** Чтобы обойти это ограничение, как и в случае оператора цикла ***For*** применяют составной оператор (второй столбец *табл. 5.1*).

Внимание! 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***).

Пример 5.4.

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

Решение.

```
...
Var
   i,  {управляющая переменная цикла}
   N, {количество суммируемых чисел}
   Summa: integer; {сумма всех целых чисел от 1 до N}
 ...
   Write('N = ');
   Readln(N); {Вводим  N}

{Обязательны начальные установки переменных, используемых в условии цикла!}
   Summa := 0;  {Начальное значение суммы}
   i := 1;          {Первое число}
  {Цикл подсчета суммы чисел}
  While  i <= N do    {В заголовке цикла – условие его повторения}
     begin
        Summa := Summa + i;
         i := i + 1 {Как минимум в одном операторе тела цикла должно быть  изменение значений
                          переменных в условии цикла, приводящее в итоге к ложности условия}
     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.

```
...
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 «До выполнения условия» ***Repeat***, (*рис. 5.8*) также как и оператор ***While***, используется в программе 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 ходом вычислений.

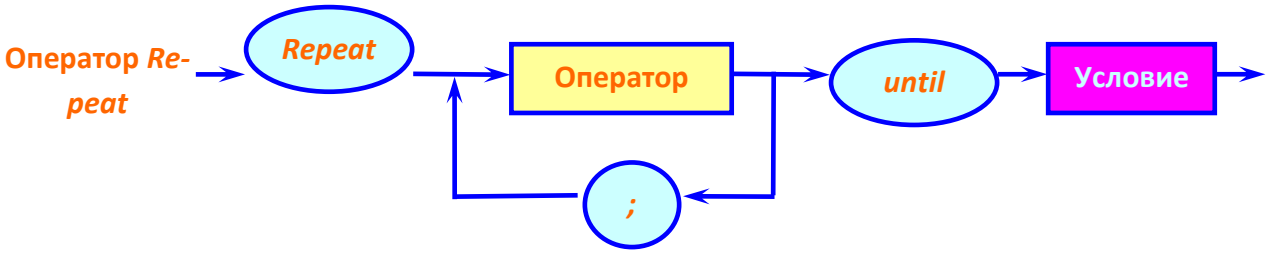

Рисунок 5.8 - Синтаксическая диаграмма оператора цикла ***Repeat***

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

```
Repeat
   Оператор 1; {группа повторяемых в цикле операторов - тело цикла}
   Оператор 2;
   …
   Оператор N
Until Условие B;
```

где ***Условие B*** - выражение логического (булевского) 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 B***. Если ***Условие B*** ложно (значение выражения условие равно ***False***), то операторы тела цикла выполняются еще раз.
- ✓ Если условие истинно (значение выражения условие равно ***True***), то выполнение цикла прекращается.

Таким образом, операторы цикла, находящиеся между ***Repeat*** и ***until***, повторяются, пока условие ложно (значение выражения условие равно ***False***).

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

Как и в случае цикла ***While***, в теле цикла обязательно должна меняться некоторая переменная, используемая в условии ***B***, чтобы это условие когда-либо выполнилось.

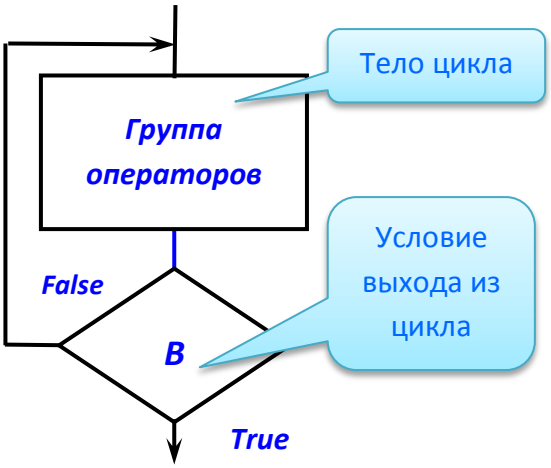

Рисунок 5.9 – Цикл с послеусловием «До выполнения условия»

Заметим, что здесь составной оператор для объединения повторяемой группы операторов *не требуется*.

Внимание! В циклах с неизвестным числом повторений ***While*** и ***Repeat .. until*** переменная цикла в отличие от цикла ***For*** может принимать и *нецелое (вещественное) значение*.

Алгоритм, соответствующий оператору ***Repeat***, представлен на *рис. 5.9*.

Пример 5.6.

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

Решение.

```
...
Var
   i,  {управляющая переменная цикла}
   N, {количество суммируемых чисел}
   Summa: integer; {сумма всех целых чисел от 1 до N}
 ...
   Write('N = ');
   Readln(N); {Вводим  N}

{Обязательны начальные установки переменных, используемых в условии цикла!}
   Summa := 0;  {Начальное значение суммы}
   i := 1;           {Первое число}
```

```
Repeat   {Цикл подсчета суммы чисел}
  Summa := Summa + i;
   i := i + 1 {Как минимум в одном операторе тела цикла должно быть  изменение
               значений переменных в условии цикла, приводящее в итоге к истинности
               условия}
 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.

```
...
x := xN;   {начальное значение переменной цикла}
repeat          {Заголовок цикла «до выполнения условия»}
  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 }
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 ***Borland Pascal*** включены две процедуры ***Break*** и ***Continue***.

Процедура ***Break*** позволяет досрочно выйти из любого типа цикла, не дожидаясь выполнения условия выхода. Например, выполнилось условие, требующее аварийного завершения цикла.

Процедура ***Continue*** позволяет начать новую итерацию любого типа цикла, даже если предыдущая не завершена. Например, дополнительная проверка в теле цикла показывает, что остальные операторы до конца тела цикла использовать не надо.

Введение в язык этих процедур 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 с предусловием *While* (пока условие истинно)	Цикл с постусловием *Repeat* (до истинности условия)
1.	До начала цикла должны быть сделаны начальные установки переменной цикла, чтобы обеспечить корректный вход в цикл	
2.	В теле цикла должны присутствовать операторы, изменяющие переменные условия так, чтобы цикл через некоторое число итераций завершился	
3.	Цикл работает, пока условие *истинно* (пока ***True***)	Цикл работает, пока условие *ложно* (пока ***False***)
4.	Цикл завершается, когда условие становится *ложным* (до ***False***)	Цикл завершается, когда условие становится *истинным* (до ***True***)
5.	Цикл может не выполниться *ни разу*, если при входе в цикл исходное значение условия ***True***	Цикл обязательно выполняется как минимум *один раз*
6.	Если в теле цикла требуется более одного оператора, то *необходимо использовать составной оператор*	Независимо от количества операторов в теле цикла *использование составного оператора не требуется.*
7.	Нормальный ход работы цикла может быть нарушен оператором ***GoTo*** (передача управления на метку вне цикла) или процедурами ***Break*** и ***Continue***	

№	Цикл со счетчиком For
1.	Начальная установка переменной (счетчика) циклов до заголовка *не требуется*
2.	Изменение в теле цикла значений переменных, стоящих в заголовке цикла *не допускается*
3.	*Количество итераций цикла неизменно* и точно определяется значениями нижней и верхней границ и шага изменения счетчика циклов (***+1*** или ***-1***)
4.	Нормальный ход работы цикла может быть нарушен оператором ***GoTo*** (передача управления на метку вне цикла) или процедурами ***Break*** и ***Continue***
5.	*Цикл может не выполниться ни разу*, если шаг цикла будет изменять значение счетчика от нижней границы в направлении, противоположным верхней границе

6 Список литературы

1. **Рапаков Г.Г., Ржеуцкая С.Ю.** *Программирование на языке Pascal.* — СПб. : БХВ-Петербург, 2004. - 480 с.

2. **Фаронов , В.В.** *Программирование на персональных ЭВМ в среде Турбо-Паскаль. :, .* - М. : Изд-во МГТУ, 1990.-580 с.

3. **Павловская , Т.А.** *Паскаль. Программирование на языке высокого уровня: Учебник для вузов.* — СПб. : Питер, 2007. — 393 с.

4. **Фаронов , В.В.** *Турбо Паскаль 7.0. Начальный курс. Учебное пособие.* -М. : ОМД Групп, 2003. - 616 с.

5. **Рютген Т., Франкен Г.** *Турбо Паскаль 7.0 .* — К. : Торгово-издательское бюро BHV, 1996-448 с.

6. **Попов, В.Б.** *Паскаль и Дельфи. Самоучитель.* — СПб. : Питер, 2004. — 544 с.