**Информатика 21 марта 9В класс.**

**!!!Задание отмечено жёлтым цветом. Буду проверять после каникул!!!!**

# Подпрограммы на паскале. Процедуры и функции. Рекурсия

**1. Дедуктивный метод программирования**

Отвлечемся на некоторое время от составления программ и поговорим о творческом процессе вообще, не только программиста или математика, а, например, художника или архитектора.

Допустим, что художник собирается нарисовать картину: портрет человека или что-то другое. Прежде, в глубине его сознания созревает общий образ будущего произведения, затем начинается ее реальное воплощение на холсте, бумаге, дереве или на чем-то другом. И вот здесь начинается тяжелейшая работа. Художник выполняет эскизы, рисунки отдельных фрагментов картины, а потом создает единое произведение, воплощающее в красках, цвете и тени, выношенный им образ.

Архитектор задолго до создания проекта своей новой конструкции также видит его целиком, а затем воплощает по отдельным частям единый проект здания или сооружения.

Подобно им и программист должен видеть в целом программу, которая решает какую-то задачу, а потом разбивает ее на отдельные части, составляет на выбранном языке программирования эти части программы, объединяет их в единое целое и получает программу.

Итак, весь творческий процесс можно разбить (разумеется, чисто условно) на следующие этапы:

1) основная идея решения задачи;

2) общая конструкция программы;

3) выделение отдельных, элементарных частей программы;

4) практическая реализация на языке программирования этих частей программы;

5) объединение их в единую программу.

Такой процесс программирования называют структурным или нисходящим. Более подробно с этим процессом мы познакомимся позже, когда изучим хотя бы основы языка программирования, но об отдельных частях, "кирпичиках", составляющих программу узнаем на этом занятии.

***Подпрограммой*** называется группа операторов, к которой обращаются из основной программы несколько раз. Иногда это может быть 2, 3 раза, а очень часто, каждый раз из выполняемого цикла основной программы.

Вполне понятно, что писать несколько раз одинаковые группы операторов трудно, проделывается много "*технической*" работы, а в некоторых случаях просто невозможно (если обращаться приходиться каждый раз при выполнении цикла).

Для облегчения такой работы и созданы **подпрограммы**.

***Использование подпрограмм позволяет***:

1) сделать основную программу более наглядной и компактной;

2) уменьшить объем используемой памяти ЭВМ;

3) сократить время отладки программы.

На языке Паскаль ***подпрограммы*** бывают двух видов, - это **процедуры и функции**.

**2. Процедуры**

Рассмотрим следующий простой пример, с помощью которого попробуем разобраться в конструкции процедур на Паскале.

***Пример 1***. Составить программу, которая бы проверяла, являются ли три числа взаимно простыми.

Мы знаем, что числа называются взаимно простыми, если их наибольший общий делитель (НОД) равен 1. Значит, для решения этой задачи нам придется дважды находить НОД чисел. Если заданы три числа: a, b, c, то найти НОД(a, b), а затем найти НОД(НОД(a, b), c).

Дважды писать операторы для нахождения НОД нам не хочется, поэтому оформим операторы для НОД в виде процедуры.

Посмотрите, как это будет выглядеть в ***программе***:

**Program** Problem1;

**uses** WinCrt;

**var**

a, b, c, k : integer;

{----------------------------------------------------------------------------------------}

**Procedure** nod(a, b : integer; **var** n : integer);

**var**

r : integer;

**begin**

**repeat**

r := a **mod** b;

a := b; b := r

**until** b = 0;

n := a

**end**;

{---------------------------------------------------------------------------------------}

**begin**

write('Введите три натуральных числа '); readln(a, b, c);

nod(a, b, k);

a := k; b := c;

nod(a, b, k);

**if** k = 1 **then** writeln('Числа взаимно простые')

**else** writeln('Числа не взаимно простые')

**end**.

В разделе описаний, после описания переменных, записывается заголовок процедуры: **Procedure**

Это слово является служебным и зарезервировано в Паскале. В одной строке с ним, через пробел, записывается имя процедуры, которое должно удовлетворять всем требованиям, предъявляемым к именам, основными из которых являются: начинаться с буквы и не иметь пробелов, т. е., требования такие же, как и к имени программы (имя нашей процедуры - nod):

**Procedure** nod(a, b : integer; **var** n : integer);

Далее, в скобках, записываются имена переменных и их типы, значения которых будут ***вводиться*** в процедуру из основной программы, в нашем случае, их две (**a**, **b**) и они имеют тип **integer**.

Сразу надо заметить, что имена этих переменных могут не совпадать с именами переменных в основной программе, скажем мы могли их обозначить m, n или любыми другими именами.

После точки с запятой и зарезервированного слова **var**, записываются переменные и их типы, значения которых будет являться результатом работы процедуры и ***выводятся*** из нее в основную программу. Такая переменная в нашем примере одна - n. Она выведет значение НОД чисел a и b. Ее имя также может иметь одноименное в основной программе и это нисколько не отразится на работе процедуры.

***Обратите внимание***, что перед переменными, значения которых *вводятся* из основной программы, не ставится слово **var**, а перед переменной, значение которой *выводится* в основную программу, это слово записано. Это очень важное обстоятельство!

Так, если поставить **var** перед **a** и **b**, то компилятор будет воспринимать эти переменные как выходные и вводимые для них значения воспринимать не будет, и, наоборот, если **var** не будет записано перед выходной переменной, то компилятор воспримет ее как входную и выводить ее значение в основную программу не будет.

Дальнейшее построение процедуры строится также, как и основная программа на Паскале.

Описываются переменные, которые будут участвовать в ее работе, но их имена не должны повторять имена уже описанных входных и выходных параметров в заголовке программы. Далее описываются необходимые для работы операторы.

В нашем примере ***процедура*** nod будет такой:

**Procedure** nod(a, b : integer; **var** n : integer);

**var**

r : integer;

**begin**

**repeat**

r := a **mod** b;

a := b; b := r

**until** b = 0;

n := a

**end**;

Основная программа строится обычным образом, но там, где необходимо найти НОД чисел, обращается к процедуре. Как?

Для этого обращаются к ней по имени, а в скобках записывают фактические значения входных переменных (в нашем случае для переменных a и b), а также имена выходных переменных (в нашем случае k).

Из приведенного ниже участка программы видно, что при первом обращении к процедуре nod определяется НОД чисел a и b (nod(a, b, k) и результат запоминается в переменную k, далее, изменяются значения переменных a и b   и снова вызывается процедура nod, которая уже находит НОД чисел k и c и результат присваивает переменной k.

Вы можете видеть основную часть **программы**:

**begin**

write('Введите три натуральных числа '); readln(a, b, c);

nod(a, b, k);

a := k; b := c;

nod(a, b, k);

**if** k = 1 **then** writeln('Числа взаимно простые')

**else** writeln('Числа не взаимно простые')

**end**.

**Сделаем общие выводы для построения и работы процедур**

Процедуры помещаются в разделе описаний и начинается зарезервированным (служебным) словом

**Procedure**

Процедуре *обязательно* дается имя, которое должно удовлетворять тем же требованиям, что и имена переменных, т.е. это может быть одна или несколько букв, комбинация букв и целых чисел, но без пробелов, начинаться с буквы и т.д.

После имени, в скобках записываются переменные - параметры и их тип: входные, значения которых используются для вычисления в качестве аргументов.

Выходные параметры - это те переменные, в которых получается результат выполнения процедуры.

**Входные и выходные параметры процедуры называются *формальными параметрами.***

Фактические, конкретные, значения формальные параметры должны получить в основной программе после обращения к ней (а пока в процедуре они являются не чем иным, как "*пустышками*").

После формальных параметров, описываются переменные, которые необходимы непосредственно для работы процедуры.

Это *параметры процедуры*. Они нужны в ней, как и в любой другой программе и описываются также. Их имена должны отличаться от имен входных и выходных параметров.

Надо заметить, что процедура может быть такой, что в ней не будет вообще параметров, достаточно тех, которые будут введены из программы.

Описание процедуры имеет вид:

**Procedur**e <имя> (<входные параметры> : <их тип>;

**var**

<выходные параметры> : <их тип>);

(раздел описаний)

**begin**

(раздел операторов)

**end**;

Она помещается в основной программе в разделе описаний.

***По входным и выходным параметрам процедуры могут быть следующих типов:***

1) ***иметь и входные и выходные параметры***:

**Procedure** <имя>(<входные параметры> : <их тип>;

**var** <выходные параметры> : <их тип>);

Мы только познакомились с программой такого типа.

2) ***иметь входные параметры, но не иметь выходных***:

**Procedure** <имя>(<входные параметры> : <их тип>);

3) ***иметь выходные параметры, но не иметь входных***:

**Procedure** <имя>(**var** <выходные параметры> : <их тип>);

4) ***не иметь ни входных, ни выходных параметров***:

**Procedure** <имя>;

***В зависимости от этого различаются процедуры по своей конструкции и выполняемым функциям.***

Далее следует раздел операторов, который составляется по тем же правилам, как и в других программах.

Процедура описана и после этого начинается основная программа.

**Вызов процедуры из программы**

Как происходит вызов подпрограммы - процедуры?

***Обязательн***о указывается имя процедуры. В скобках задаются ***фактические*** значения входных параметров и те переменные, в которые будут "*запоминаться*" выходные значения.

Рассмотрим пример, где может быть использована процедура второго типа: имеет входные параметры, но не имеет выходных.

***Пример 2***. Составить программу, которая устанавливает, какие числа из заданного промежутка [a; b] можно представить в виде суммы двух квадратов целых чисел?

В этой программе, нам придется проверять каждое из чисел промежутка [a; b] можно ли его представить в виде суммы квадратов двух чисел, поэтому было бы разумно разработать процедуру, которая бы проверяла одно число и затем обращаться к ней из основной программы для проверки каждого числа из промежутка.

Процедуру составим по следующему способу. Пусть задано число **n**. Нам необходимо найти такие два числа **a** и **b**, чтобы сумма их квадратов была равна n, т.е. решить в целых числах уравнение: 

Возникает естественное желание испытывать натуральные числа от 1 и до ...? А вот до какого значения неизвестно. Если их брать до числа n, то это будет слишком много лишней и бесполезной работы.

Чтобы выяснить этот вопрос, можно *организовать цикл*, в котором проверять сколько чисел **a** надо, чтобы выполнялось неравенство:  Здесь, в качестве **b** взято наименьшее натуральное число 1. Организовав такой цикл, и подсчитав, сколько чисел a потребуется, мы узнаем сколько чисел надо просматривать, чтобы найти решение уравнения.

Этот **цикл** может быть таким:

a := 1; k := 1;

**while** a\*a + 1<=n **do**

**begin**

k := k + 1;

a := a + 1

**end**;

Теперь ясно, что для испытания чисел, следует устроить цикл от 1 до k:

**for** a := 1 **to** k **do**

***Второй цикл*** должен быть для значений b. Но если его организовать тоже от 1 до k, тогда могут повторяться дважды одинаковые значения, только на разных местах, например, для числа 20 могут быть выданы следующие значения:

22 + 42 = 20 и 42 + 22 = 20.

Чтобы избежать повторения чисел, цикл для чисел b можно организовать либо от 1 до **a**, либо от **k** до **а**.

Нами выбран первый вариант.

***Процедура***

**Procedure** to\_square(n : integer);

**label** 1;

**var**

a, b, k : integer;

**begin**

a := 1; k := 1;

**while** a\*a + 1<=n **do**

**begin**

k := k + 1;

a := a + 1

**end**;

**for** a := 1 **to** k **do**

**for** b := 1 **to** a **do**

**if** a\*a + b\*b = n

**then**

**begin**

writeln(n, '=', a, '\*', a,' +', b, '\*', b); **goto** 1

**end**;

1: **end**;

Процедура выполнена с досрочным прерыванием цикла, так как нет необходимости выяснять всевозможные значения пар чисел, удовлетворяющих этому уравнению, а достаточно просто выяснить возможность такого представления.

Выполнив такую процедуру, не составляет труда решить полностью задачу. Для этого в основной программе выполнить цикл для всех чисел из промежутка, и каждое из которых, с помощью процедуры проверять. Кстати говоря, эта процедура имеет только один ***формальный параметр*** - ***входной***, - значение проверяемого числа из промежутка и не имеет выходных параметров.

***Программа***

**Program** Problem2;

**uses** WinCrt;

**var**

a, b, i : integer;

{---------------------------------------------------------------------------------------}

**Procedure** to\_square(n : integer);

**label** 1;

**var**

a, b, k : integer;

**begin**

a := 1; k := 1;

**while** a\*a + 1 <= n **do**

**begin**

k := k + 1;

a := a + 1

**end**;

**for** a := 1 **to** k **do**

**for** b := 1 **to** a **do**

**if** a\*a + b\*b = n

**then**

**begin**

writeln(n, '=', a, '\*', a, '+', b,'\*', b); **goto** 1

**end**;

1: **end**;

{----------------------------------------------------------------------------------------}

**begin**

write('Введите начало промежутка '); readln(a);

write('Введите конец промежутка '); readln(b);

write('Числа, которые можно представить в виде суммы ');

writeln('квадратов следующих чисел');

**for** i := a **to** b **do** to\_square(i);

**end**.

### Задание 1

1. Покажите, что квадрат числа, являющегося суммой двух точных квадратов, также можно представить в виде суммы двух точных квадратов.

2. Покажите, что произведение двух целых чисел, из которых каждое есть сумма квадратов двух целых чисел, можно представить в виде суммы двух точных квадратов.

***Пример 3***. Составить программу нахождения и вывода на экран всех простых чисел из заданного промежутка [n; m]. (Массивы не использовать.)

Следует сразу заметить, что эта задача решается иначе с помощью программы "**Решето Эратосфена**", но ее мы будем разбирать при изучении работы с массивами чисел и множествами.

А сейчас приведем другой вариант решения этой задачи. Ясно, что для ее решения нам потребуется процедура, которая устанавливает, является ли число простым. На предыдущих занятиях мы уже составляли такую программу. Сейчас вспомним ее работу и возможно внесем некоторые изменения.

***Алгоритм*** составления процедуры такой.

*Во-первых*, вспомним, какие числа называются *простыми*.

Натуральные числа, которые имеют только два делителя 1 и само себя называются ***простыми***. Остальные называются ***составными***.

Из их числа исключается число 1, которое не относится ни к простым, ни к составным.

Первое простое число - это 2. **Если** оно есть, **тогда** его сразу надо выводить на экран:

**if** p = 2 **then** write(p,' ')

Ясно, что все остальные *четные* числа являются составными, а значит их нет смысла проверять. Для исключения из рассмотренных четных чисел введем для проверки условный оператор:

**else if** p **mod** 2 <> 0 **then**

**Если** число нечетное, **тогда** его надо проверять. Сущность проверки будет заключаться в определении числа делителей. Но нам уже известен математический факт, что все делители некоторого натурального числа p находятся в промежутке от 1 до корня квадратного из этого числа (исключая само число). Значит, **если** имеются делители, в данном случае, от 3 до trunc(sqrt(p)), **то** оно является составным, а **если** таких делителей нет (четные числа мы уже исключили, а единицу и само число **p** не рассматриваем), тогда число будет простым.

Для проверки этого факта устроен следующий **цикл**:

**begin**

i := 3; k := 0;

**while** i <= trunc(sqrt(p)) **do**

**begin**

**if** p **mod** i = 0 **then** k := k + 1;

i := i + 2

**end**;

**if** k = 0 **then** write(p, ' ')

**end**

Полностью ***процедура***, устанавливающая является ли число простым будет следующей (probleme number -простое число):

**Procedure** Probleme\_number(p : integer);

**var**

i, k : integer;

**begin**

**if** p=2 **then** write(p, ' ') **else if** p **mod** 2<>0

**then**

**begin**

i := 3; k := 0;

**while** i<=trunc(sqrt(p)) **do**

**begin**

**if** p **mod** i = 0

**then** k := k + 1;

i := i + 2

**end**;

**if** k = 0 **then** write(p, ' ')

**end**

**end**;

Снова, эта процедура имеет ***только входной формальный параметр и не имеет выходных***.

***Программу*** составить нетрудно. В ней достаточно организовать цикл для проверки чисел из заданного промежутка и каждое из них проверять с помощью процедуры.

**Program** Problem3; { Простые числа из промежутка [n; m] }

**uses** WinCrt;

**var**

n, m, i : integer;

{----------------------------------------------------------------------------------------}

**Procedure** probleme\_number(p : integer);

**var**

i, k : integer;

**begin**

**if** p=2 **then** write(p, ' ')

**else if** p **mod** 2 <> 0

**then**

**begin**

i := 3; k := 0;

**while** i <= trunc(sqrt(p)) **do**

**begin**

**if** p **mod** i = 0 **then** k := k + 1;

i := i + 2

**end**;

**if** k = 0 **then** write(p, ' ')

**end**

**end**;

{----------------------------------------------------------------------------------------}

**begin**

write('Введите левую границу промежутка > 1 '); readln(n);

write('Введите правую границу промежутка '); readln(m);

writeln('Простые числа из промежутка [', n, ' ', m, ']');

**for** i := n **to** m **do** probleme\_number(i);

writeln

**end**.

***Пример 4***. Французский физик М. Мерсен (1588 - 1648) заметил, что многие простые числа имеют вид

2**p** - 1,

где **p** также простое число. Все числа такого вида называются числами Мерсена. Составить программу, которая находит числа Мерсена на заданном промежутке.

***Алгоритм***

Самый простейший алгоритм для составления программы, будет такой:

*во-первых*, необходима процедура для определения простых чисел, с которой мы уже знакомы;

*во-вторых*, нужна процедура, которая вычисляет степень натурального числа с натуральным показателем (эту процедуру вы уже должны были составлять при выполнении задания).

***Процедура*** определения простого числа:

**Procedure** Probleme\_number(p : longint; **var** v : longint);

**var**

i, k : longint;

**begin**

**if** p = 2 **then** v := p

**else if** p **mod** 2 <> 0

**then**

**begin**

i := 3; k := 0;

**while** i <= trunc(sqrt(p)) **do**

**begin**

**if** p **mod** i = 0 **then** k := k + 1;

i := i + 2

**end**;

**if** k = 0 **then** v := p

**end**

**end**;

***Процедура*** вычисления степени натурального числа с натуральным показателем (extent - степень):

**Procedure** extent(a, n : integer; **var** s : longint);

**var**

i : integer;

**begin**

s := 1;

**for** i := 1 **to** n **do** s := s\*a

**end**;

В основной программе надо пробовать каждое число из промежутка, является ли оно простым, а для этого необходимо обращение к процедуре probleme\_number(i, p).

**Если** получено простое число, **тогда** следует 2 возвести в эту степень, а для этого следует обратиться к процедуре extent(2, p, m).

Наконец, полученный результат также надо проверить, является ли он простым числом. **Если** является, **тогда** мы получаем число Мерсена.

Казалось бы все хорошо, но нас поджидают многие неприятности.

Неприятность первая! Выбрано число 3, процедурой probleme\_number(i, p) установлено, что оно простое число и его значение присвоено переменной p. Следующим числом из выбранного промежутка является 4, процедура probleme\_number(i, p) установила, что оно не является простым и его значение не присваивается переменной p. Тогда возникает вопрос, а чему равно значение переменной p? Оказывается оно осталось прежним, т. е. равным 3 - предыдущему простому числу.

Если продолжать дальше процесс в программе, тогда значения будут повторяться. Аналогичное повторение будет и при проверке предполагаемого числа Мерсена, является ли оно простым?

Чтобы избежать этих неприятностей, после каждой проверки, начиная с первой, необходимо запоминать полученные значения простых чисел в новые переменные, например: p1 := p; m1 := m; n1 := n, а затем перед повторным обращением к процедурам вычисления степени и проверке простых чисел, проверять, а не являются ли эти значения повторениями предыдущих.

Эта часть основной программы такая:

**for** i := 2 **to** b **do**

**begin**

probleme\_number(i, p);

**if** p <> p1 **then** extent(2, p, m);

**if** m <> m1 **then** probleme\_number(m-1, n);

**if** n <> n1 **then** write(n, ' ');

n1 := n; p1 := p; m1 := m

**end**;

***Программа***

**Program** Problem4; { Числа Мерсена }

**uses** WinCrt;

**var**

b, p, p1, m, m1, n, n1, i : longint;

{----------------------------------------------------------------------------------------}

**Procedure** Probleme\_number(p : longint; **var** v : longint);

**var**

i, k : longint;

**begin**

**if** p = 2 **then** v := p

**else** **if** p **mod** 2 <> 0

**then**

**begin**

i := 3; k := 0;

**while** i <= trunc(sqrt(p)) **do**

**begin**

**if** p **mod** i = 0 **then** k := k + 1;

i := i + 2

**end**;

**if** k = 0 **then** v := p

**end**

**end**;

{----------------------------------------------------------------------------------------}

**Procedure** extent(a, n : integer; **var** s : longint);

**var**

i : integer;

**begin**

s := 1;

**for** i := 1 **to** n **do** s := s\*a

**end**;

{---------------------------------------------------------------------------------------}

**begin**

write('Введите правую гран. знач. показ. степ. '); readln(b);

write('Числа Мерсена: ');

**for** i := 2 **to** b **do**

**begin**

probleme\_number(i, p);

**if** p <> p1 **then** extent(2, p, m);

**if** m <> m1 **then** probleme\_number(m - 1, n);

if n <> n1 **then** write(n,'; ');

n1 := n; p1 := p; m1 := m

**end**;

writeln

**end**.

***Замечание***. В дальнейшем, при работе с массивами чисел, мы найдем более простой способ нахождения чисел Мерсена с использованием "***Решета Эратосфена***".