МИНОБРНАУКИ РОССИИ

политехнический колледж филиала федерального государственного бюджетного образовательного учреждения высшего образования «Майкопский государственный технологический университет» в поселке Яблоновском

МАТЕРИАЛЫ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

по дисциплине

ОП.04 Основы алгоритмизации и программирования

специальность

09.02.07 Информационные системы и программирование

форма обучения

очная

квалификация выпускника

программист

Яблоновский, 2018

УДК 004 (07)

ББК 73

М-34

Одобрено предметной (цикловой) комиссией

информационных и математических дисциплин

Разработчик: Хуаде Р.А. - преподаватель первой категории политехнического колледжа филиала федерального государственного бюджетного образовательного учреждения высшего образования «Майкопский государственный технологический университет» в поселке Яблоновском

Материалы для подготовки к экзамену

(решение типовых задач)

Линейные алгоритмы

Задача № 1. Вывести на экран сообщение «Hello World!»

Формулировка. Вывести на экран сообщение «Hello World!». Некоторые учебные курсы по программированию рассматривают эту задачу как самую первую при изучении конкретного языка или основ программирования.

Решение. Эта задача включает в себя лишь демонстрацию использования оператора вывода write (или writeln), который будет единственным в теле нашей маленькой программы. С помощью него мы будем осуществлять вывод на экран константы 'Hello World!' типа string (или, как допускается говорить, строковой константы). В данном случае будем использовать оператор writeln. Напомню, что при использовании оператора write курсор останется в той же строке, в которой осуществлялся вывод, и будет находиться на одну позицию правее восклицательного знака во фразе «Hello World!», а при использовании оператора writeln – на первой позиции слева в следующей строке.

Код:

```
1.      program HelloWorld;
2.
3. begin
4. writeln('Hello World!')
5. end.
```

Задача № 2. Вывести на экран три числа в порядке, обратном вводу

Формулировка. Вывести на экран три введенных с клавиатуры числа в порядке, обратном их вводу. Другими словами, мы ввели с клавиатуры три числа (сначала первое, потом второе и третье), и после этого единственное, что нам нужно сделать – это вывести третье, затем второе и первое.

Решение. Так как с клавиатуры вводится три числа, необходимо завести три переменные. Обозначим их как a, b и c. Ввиду того, что нам ничего не сказано о том, в каком отрезке могут располагаться введенные числа, мы возьмем тип integer, так как он охватывает и положительные, и отрицательные числа в некотором диапазоне (от –2147483648 до 2147483647). Затем нам нужно использовать оператор вывода write (writeln), в списке аргументов которого (напомним, что список аргументов write (writeln) может содержать не только переменные, но и константы и арифметические выражения) эти переменные будут находиться в обратном порядке. В данном случае будем использовать оператор writeln, который после вывода результата переведет курсор на следующую строку: writeln(c, b, a); Однако если мы оставим его в таком виде, то увидим, что при выводе между переменными не будет никакого пробела, и они будут слеплены и визуально смотреться как одно число. Это связано с тем, что при вводе мы использовали пробелы для разделения чисел, а сами пробелы никаким образом не влияют на содержимое переменных,

которые будут последовательно выведены оператором writeln без каких-либо дополнений. Чтобы избежать этого, нам нужно добавить в список аргументов Данил Душистов: «Решение 50 типовых задач по программированию на языке Pascal» 3 writeln две текстовые константы-пробелы. Проще говоря, пробельная константа – это символ пробела, заключенный в одиночные апострофы (апостроф – символ «'»). Первая константа будет разделять переменные a и b, вторая – b и c. В результате наш оператор вывода будет таким: writeln(c, ' ', b, ' ', a); Теперь он работает так: выводит переменную c, затем одиночный символ пробела, затем переменную b, потом еще один символ пробела и, наконец, переменную a.

Код:

```
1. program WriteThree;
2.
1.     var
4. a, b, c: integer;
5.
6. begin
7. readln(a, b, c);
8. writeln(c, ' ', b, ' ', a)
9. end.
```

Задача № 3. Вывести на экран квадрат введенного числа

Формулировка.Дано натуральное число меньше 256. Сформировать число, представляющее собой его квадрат. Решение. Для ввода числа нам необходима одна переменная. Обозначим эту переменную как a. Так как нам ничего не сообщается о необходимости сохранить исходное число, то для получения квадрата мы можем использовать ту же самую переменную, в которую считывали число с клавиатуры. В условии задачи дается ограничитель величины вводимого числа – фраза «меньше 256». Это означает, что оно может быть охвачено типом byte. Но что произойдет, если в переменную a будет введено число 255, и затем мы попытаемся присвоить ей его квадрат, равный 65025? Естественно, это вызовет переполнение типа данных, так как используемой для переменной a ячейки памяти не хватит для того, чтобы вместить число 65025. Значит, для ее описания мы должны использовать более емкий числовой тип. При этом типом минимальной размерности, охватывающим данный отрезок (от 1 (это 12) до 65025), является тип word. Его мы и будем использовать при описании a. Далее нужно сформировать в переменной a квадрат. Для этого присвоим ей ее прежнее значение, умноженное само на себя: a := a * a; Теперь остается вывести результат на экран. Для этого будем использовать оператор writeln.

Код:

```
1. program SqrOfNum;
2.
3. var
4. a: word;
```

5.

6. begin

7. readln(a);

8. a := a * a;

9. writeln(a)

10. end.

Задача № 4. Получить реверсную запись трехзначного числа

Формулировка. Сформировать число, представляющее собой реверсную (обратную в порядке следования разрядов) запись заданного трехзначного числа. Например, для числа 341 таким будет 143. Давайте разберемся с условием. В нашем случае с клавиатуры вводится некоторое трехзначное число (трехзначными называются числа, в записи которых три разряда (то есть три цифры), например: 115, 263, 749 и т. д.). Нам необходимо получить в некоторой переменной число, которое будет представлять собой реверсную запись введенного числа. Другими словами, нам нужно перевернуть введенное число «задом наперед», представить результат в некоторой переменной и вывести его на экран. Решение. Определимся с выбором переменных и их количеством. Ясно, что одна переменная нужна для записи введенного числа с клавиатуры, мы обозначим ее как n. Так как нам нужно переставить разряды числа n в некотором порядке, следует для каждого из них также предусмотреть отдельные переменные. Обозначим их как a (для разряда единиц), b (для разряда десятков) и c (для разряда сотен). Теперь можно начать запись самого алгоритма. Будем разбирать его поэтапно: 1) Вводим число n; 2) Работаем с разрядами числа n. Как известно, последний разряд любого числа в десятичной системе счисления – это остаток от деления этого числа на 10. В терминах языка Pascal это означает, что для получения разряда единиц нам необходимо присвоить переменной a остаток от деления числа n на 10. Этому шагу соответствует следующий оператор: a := n mod 10; Получив разряд единиц, мы должны отбросить его, чтобы иметь возможность продолжить работу с разрядом десятков. Для этого разделим число n на 10. В терминах Pascal, опять же, это означает: присвоить переменной n результат от деления без остатка числа n на 10. Это мы сделаем с помощью оператора n := n div 10; 3) Очевидно, что после выполнения п. 2 в переменной n будет храниться двухзначное число, состоящее из разряда сотен и разряда десятков исходного. Теперь, выполнив те же самые действия еще раз, мы получим разряд десятков исходного числа, но его уже нужно присваивать переменной b. 4) В результате в переменной n будет храниться однозначное число – разряд сотен исходного числа. Мы можем без дополнительных действий присвоить его переменной c. 5) Все полученные в переменных числа – однозначные. Теперь переменная n нам больше не нужна, и в ней нужно сформировать число-результат, в котором a будет находиться в разряде сотен, b – десятков, c – единиц. Легко понять, что для этого нам следует умножить a на 100, прибавить к полученному числу b, умноженное на 10 и c без изменения, и весь этот результат присвоить переменной c. Это можно записать так: n := 100 * a + 10 * b + c; 6) Далее остается только вывести полученное число на экран.

Код:

1. program ReverseNum;

2.

```
3. var
4. n, a, b, c: word;
5.
6. begin
7. readln(n);
8. a := n mod 10;
9. n := n div 10;
10. b := n mod 10;
11. n := n div 10;
12. c := n;
13. n := 100 * a + 10 * b + c;
14. writeln(n)
15. end.
```

Проверим работу программы на произвольном варианте введенных данных. Для этого выполним ее «ручную прокрутку», проделав с введенным числом те же действия, которые должен выполнить алгоритм. Пусть пользователем введено число 514. Покажем в таблице, какие значения будут принимать переменные после выполнения соответствующих строк. При этом прочерк означает, что значение переменных на данном шаге не определено, а красным цветом выделены переменные, которые изменяются:

№ строки	n	a	b	c
7	514	-	-	-
8	514	4	-	-
9	51	4	-	-
10	51	4	1	-
11	5	4	1	-
12	5	4	1	5
13	415	4	1	5

Нетрудно понять, что написанная программа будет выводить правильный ответ для любого заданного трехзначного числа, так как в соответствии с алгоритмом заполнение данной таблицы возможно лишь единственным образом. Это значит, что мы можем представить число в виде аб-

страктного трехзначного числа ***xyz***, (в нем каждая буква должна быть заменена на любое число от 0

до 9, конечно, за исключением тех случаев, когда оно перестает быть трехзначным), и работая с

разрядами этого числа, показать, что в результате работы ответом будет число ***zyx***.

Задача № 5. Посчитать количество единичных битов числа

Формулировка. Дано натуральное число меньше 16. Посчитать количество его единичных

битов. Например, если дано число 9, запись которого в двоичной системе счисления равна 10012 (подстрочная цифра 2 справа от числа означает, что оно записано в двоичной системе счисления), то количество его единичных битов равно 2.

Решение. Нам необходима переменная для ввода с клавиатуры. Обозначим ее как **n**. Так как мы должны накапливать количество найденных битов, то возникает потребность в еще одной переменной. Обозначим ее как **count** («*count*» в переводе с англ. означает «считать», «подсчет» и т. д.). Переменные возьмем типа **byte** (они могут принимать значения от 0 до 255), и пусть в данном случае такой объем избыточен, но это не принципиально важно.

Как же сосчитать количество битов во введенном числе? Ведь число же вводится в десятичнойсистеме счисления, и его нужно переводить в двоичную?

На самом деле все гораздо проще. Здесь нам поможет одно интересное правило:

*Остаток от деления любого десятичного числа **x** на число **p** дает нам разряд единиц числа **x** (его крайний разряд справа) в системе счисления с основанием **p**.* То есть, деля некоторое десятичное число, например, на 10, в остатке мы получаем разряд единиц этого числа в системе счисления с основанием 10. Возьмем, например, число 3468. Остаток от деления его на 10 равен 8, то есть разряду единиц этого числа. Понятно, что такие же правила господствуют и в арифметике в других системах счисления, и в том числе в двоичной системе. Предлагаю поэкспериментировать: запишите на бумаге десятичное число, затем, используя любой калькулятор с функцией перевода из одной системы счисления в другую, переведите это число в двоичную систему счисления и также запишите результат. Затем разделите исходное число на 2 и снова переведите в двоичную систему. Как оно изменилось в результате? Вполне очевидно, что у него пропал крайний разряд справа, или, как мы уже говорили ранее, разряд единиц. Но как это использовать для решения задачи? Воспользуемся тем, что в двоичной записи числа нет цифр, кроме 0 и 1. Легко убедиться в том, что сложив все разряды двоичного числа, мы получаем как раз таки количество его единичных битов. Это значит, что вместо проверки значений разрядов двоичного представления числа мы можем прибавлять к счетчику сами эти разряды – если в разряде был 0, значение счетчика не изменится, а если 1, то повысится на единицу.

Теперь, резюмируя вышеприведенный итог, можно поэтапно сформировать сам алгоритм:

1) Вводим число **n**;

2) Обнуляем счетчик разрядов **count**. Это делается потому, что значения всех переменных при запуске программы считаются неопределенными, и хотя в большинстве компиляторов **Pascal** они обнуляются при запуске, все же считается признаком «хорошего тона» в программировании обнулить значение переменной, которая будет изменяться в процессе работы без предварительного присваивания ей какого-либо значения.

3) Прибавляем к **count** разряд единиц в двоичной записи числа **n**, то есть остаток от деления **n** на 2: count := count + n mod 2; Строго говоря, мы могли бы не прибавлять предыдущее значение переменной **count** к остатку от деления, так как оно все равно было нулевым. Но мы поступили так для того, чтобы сделать код более однородным, далее это будет видно. Учтя разряд единиц в двоичной записи **n**, мы должны отбросить его, чтобы исследовать число далее. Для этого разделим **n** на 2. На языке **Pascal** это будет выглядеть так:

n := n div 2;

4) Теперь нам нужно еще два раза повторить **п. 3** , после чего останется единственный двоичный разряд числа **n**, который можно просто прибавить к счетчику без каких-либо дополнений:

count := count + n;

5) В результате в переменной **count** будет храниться количество единичных разрядов в двоичной записи исходного числа. Осталось лишь вывести ее на экран.

Код:

```
1. program BinaryUnits;

2.

3. var

4. n, count: byte;

5.

6. begin

7. readln(n);

8. count := 0;

9. count := count + n mod 2;

10. n := n div 2;

11. count := count + n mod 2;

12. n := n div 2;

13. count := count + n mod 2;

14. n := n div 2;

15. count := count + n;

16. writeln(count)

17. end.
```

Программа работает правильно на всех вариантах правильных исходных данных, в чем не сложно убедиться с помощью простой проверки.

Условные операторы

Задача № 6. Вывести на экран наибольшее из двух чисел

Формулировка.Даны два числа. Вывести на экран то из них, которое больше.

Решение. Собственно, это самая простая задача, с помощью которой можно продемонстрировать использование условного оператора **if**. Напомним, как нужно использовать этот оператор. Мы вводим с клавиатуры числа в переменные **a** и **b** типа **integer**, затем в операторе **if** проверяем булевское выражение «**a > b**»: если оно истинно, то выполняется **then**-блок оператора, если ложно – **else**- блок. Соответственно, если **a** больше **b** (условие в заголовке истинно), то в **then**-блоке мы выводим **a**, а если **a** не больше **b** (условие в заголовке ложно), то выводим **b** (хотя сюда попадает и случай, когда **a** = **b**, что, впрочем, не нарушает решения).

На языке **Pascal** мы можем записать весь оператор с **if**- и **then**-блоками в одну строчку следующим образом: if a > b then writeln(a) else writeln(b); Данная строка легко понятна и читаема по причине того, что мы выполняем столь простой набор операторов в обоих блоках ветвления оператора **if**. Однако в более сложных примерах мы будем с первых же написанных строчек следовать *принципу аккуратного оформления кода*, чтобы не появлялось привычки «вытягивать» операторы ветвлений и другие конструкции в одну строчку, так как в будущем это может сильно сказаться на удобочитаемости и простоте понимания написанного программного кода, особенно при увеличении количества вложенных в блок операторов (которые, например, тоже могут быть операторами ветвления). Не стоит забывать о том, что при вложенности в тело какого-либо оператора хотя бы одного составного оператора или другой сложной конструкции требуется равномерный отступ для подчиненной конструкции с адекватной расстановкой операторных скобок! Например, для оператора **if** это распределение конструкций по мнемонической модели **if-end**, **else-end**, согласно которой эти ключевые слова должны стоять на одном уровне по вертикали, а их содержимое должно быть немного смещено вправо.

Конечно, для простейшей конструкции с условным оператором это вовсе не самоцель, и можно разместить ее в одной строке, если оби ветви оператора (и **if**-блок, и **else**-блок) не содержат составного оператора. В нашем же примере «аккуратное оформление» показывается лишь в качестве введения.

Код:

```
1. program MaxOfTwo;

2.

3. var

4. a, b: integer;

5.

6. begin

7. readln(a, b);

8. if a > b then begin

9. writeln(a)

10. end

11. else begin

12. writeln(b)
```

13. end

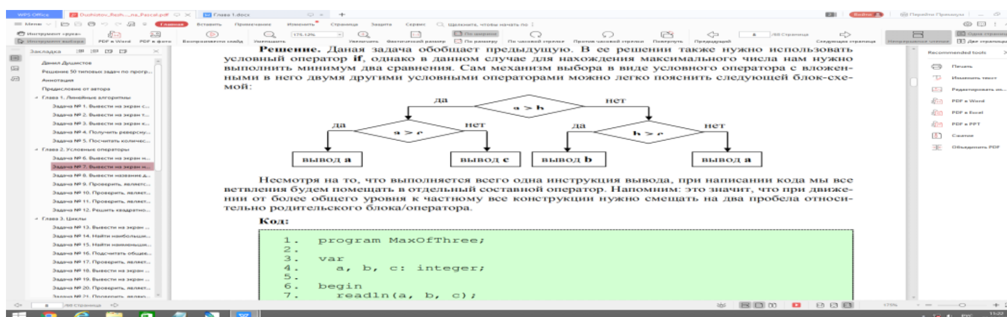
14. end.

При таком оформлении хорошо видно, какой код выполняется при истинности условия, а какой – при его ложности.

Задача № 7. Вывести на экран наибольшее из трех чисел

Формулировка.Даны три числа. Вывести на экран то из них, которое больше.

Решение. Даная задача обобщает предыдущую. В ее решении также нужно использовать условный оператор **if**, однако в данном случае для нахождения максимального числа нам нужно выполнить минимум два сравнения. Сам механизм выбора в виде условного оператора с вложенными в него двумя другими условными операторами можно легко пояснить следующей блок-схемой:



Несмотря на то, что выполняется всего одна инструкция вывода, при написании кода мы все ветвления будем помещать в отдельный составной оператор. Напомним: это значит, что при движении от более общего уровня к частному все конструкции нужно смещать на два пробела относительно родительского блока/оператора.

Код:

1. program MaxOfThree;

2.

3. var

4. a, b, c: integer;

5.

6. begin

7. readln(a, b, c);

8. if a > b then begin

9. if a > c then begin

10. writeln(a)

11. end

12. else begin

```
13. writeln(c)
14. end
15. end
16. else begin
17. if b > c then begin
18. writeln(b)
19. end
20. else begin
21. writeln(c)
22. end
23. end
24. end.
```

Задача № 8. Вывести название дня недели по его номеру

Формулировка.Вывести название дня недели по его номеру. Решение. Задача простейшим образом решается с помощью оператора выбора case. Напомним, что этот оператор позволяет организовать ветвления в зависимости от значений некоторой переменной, для каждого из которых можно предусмотреть выполнение различных действий. Причем если значению переменной не соответствует ни один вариант, выполняется else-блок (если он присутствует). Кстати, не стоит забывать, что после перечисления всех вариантов оператора case необходимо написать ключевое слово end (выходит, ключевое слово case является еще и открывающей операторной скобкой). Для того чтобы воспользоваться оператором case, нам необходимо произвести ввод номера дня недели в некоторую переменную i типа byte и по этому номеру вывести название текущего дня недели. Кстати, благодаря else-блоку в этой программке мы впервые предусмотрим сообщение об ошибке, связанной с некорректно введенным номером, которому не соответствует ни один из дней недели.

Код:

```
1. program DaysOfTheWeek;
2.
3. var
4. i: byte;
5.
6. begin
7. readln(i);
8. case i of
9. 1: writeln('Monday');
```

```
10. 2: writeln('Tuesday');
11. 3: writeln('Wednesday');
12. 4: writeln('Thursday');
13. 5: writeln('Friday');
14. 6: writeln('Saturday');
15. 7: writeln('Sunday')
16. else writeln('This day of the week does not exist!')
17. end
18. end.
```

Кстати, в каждом из вариантов ветвлений может быть помещен составной оператор, но при описании вариантов мы не стали использовать операторные скобки, так как на этот раз они наоборот испортили бы все оформление кода, которое сейчас является достаточно гармоничным.

Задача № 9. Проверить, является ли четырехзначное число палиндромом

Формулировка.Дано четырехзначное число. Проверить, является ли оно палиндромом.

Примечание: палиндромом называется число, слово или текст, которые одинакового читаются слева направо и справа налево. Например, в нашем случае это числа 1441, 5555, 7117 и т. д.

Примеры других чисел-палиндромов произвольной десятичной разрядности, не относящиеся к решаемой задаче: 3, 787, 11, 91519 и т. д.

Решение. Для ввода числа с клавиатуры будем использовать переменную **n**. Вводимое число принадлежит множеству натуральных чисел и четырехзначно, поэтому оно заведомо больше 255, так что тип **byte** для ее описания нам не подходит. Тогда будем использовать тип **word**.

Какими же свойствами обладают числа-палиндромы? Из указанных примеров легко увидеть, что в силу своей одинаковой «читаемости» с двух сторон в них равны первый и последний разряд, второй и предпоследний и т. д. вплоть до середины. Причем если в числе нечетное количество разрядов, то серединную цифру можно не учитывать при проверке, так как при выполнении названного правила число является палиндромом вне зависимости от ее значения.

В нашей же задаче все даже несколько проще, так как на вход подается четырехзначное число. А это означает, что для решения задачи нам нужно лишь сравнить 1-ю цифру числа с 4-й и 2-ю цифру с 3-ей. Если выполняются оба эти равенства, то число – палиндром. Остается только получить соответствующие разряды числа в отдельных переменных, а затем, используя условный оператор, проверить выполнение обоих равенств с помощью булевского (логического) выражения.

Однако не стоит спешить с решением. Может быть, мы сможем упростить выведенную схему? Возьмем, например, уже упомянутое выше число 1441. Что будет, если разделить его на два числа двузначных числа, первое из которых будет содержать разряд тысяч и сотен исходного, а второе – разряд десятков и единиц исходного. Мы получим числа 14 и 41. Теперь, если второе число заменить на его реверсную запись (это мы делали в **задаче 4**), то мы получим два равных числа 14 и 14! Это преобразование вполне очевидно, так в силу того, что палиндром читается одинаково в обоих направлениях, он состоит из дважды раза повторяющейся комбинации цифр, и одна из

копий просто повернута задом-наперед. Отсюда вывод: нужно разбить исходное число на два двузначных, одно из них реверсировать, а затем выполнить сравнение полученных чисел с помощью условного оператора **if**. Кстати, для получения реверсной записи второй половины числа нам необходимо завести еще две переменные для сохранения используемых разрядов. Обозначим их как **a** и **b**, и будут они типа **byte**.

Теперь опишем сам алгоритм:

1) Вводим число **n**;

2) Присваиваем разряд единиц числа **n** переменной **a**, затем отбрасываем его. После присваиваем разряд десятков **n** переменной **b** и также отбрасываем его:

a := n mod 10;

n := n div 10;

b := n mod 10;

n := n div 10;

3) Присваиваем переменной **a** число, представляющее собой реверсную запись хранящейся в переменных **a** и **b** второй части исходного числа **n** по уже известной формуле:

a := 10 * a + b;

4) Теперь мы можем использовать проверку булевского выражения равенства полученных чисел **n** и **a** помощью оператора **if** и организовать вывод ответа с помощью ветвлений:

if n = a then writeln('Yes') else writeln('No');

Так как в условии задачи явно не сказано, в какой форме необходимо выводить ответ, мы будем считать логичным вывести его на интуитивно понятном пользователю уровне, доступном в средствах самого языка **Pascal**. Напомним, что с помощью оператора **write** (**writeln**) можно выводить результат выражения булевского типа, причем при истинности этого выражения будет выведено слово 'TRUE' («true» в пер. с англ. означает «истинный»), при ложности – слово 'FALSE' («false» в пер. с англ. означает «ложный»). Тогда предыдущая конструкция с **if** может быть заменена на

writeln(n = a);

Код:

1. program PalindromeNum;

2.

3. var

4. n: word;

5. a, b: byte;

6.

7. begin

8. readln(n);

9. a := n mod 10;

10. n := n div 10;

11. b := n mod 10;

12. n := n div 10;

13. a := 10 * a + b;

14. writeln(n = a)

15. end.

Задача № 10. Проверить, является ли четырехзначное число счастливым билетом

Формулировка.Дано четырехзначное число. Проверить, является ли оно «счастливым билетом». Примечание: счастливым билетом называется число, в котором: а) при четном количестве цифр в числе сумма цифр его левой половины равна сумме цифр его правой половины; б) при нечетном количестве цифр – то же самое, но с отбрасыванием серединной цифры. Например, рассмотрим число 1322. Его левая половина равна 13, а правая – 22, и оно является счастливым билетом (т. к. 1 + 3 = 2 + 2). Аналогично: 1735 (1 + 7 = 3 + 5), 1111 (1 + 1 = 1 + 1) и т. д.

Примеры других счастливых билетов за рамками условия текущей задачи: 7 (отбросили единственную цифру), 39466 (3 + 9 = 6 + 6, а 4 отбросили), 11 (1 = 1), и т. д.

Решение. Для ввода достаточно одной переменной **n** типа **word**. Все, что необходимо сделать для решения – это последовательно получить все разряды исходного числа, причем из двух младших разрядов (единиц и десятков) сформировать первую сумму, а из двух старших разрядов – вторую, после чего вывести на экран результат булевского выражения равенства полученных сумм. Первую сумму будем хранить в переменной **right**, а вторую – в переменной **left** (выбрано по расположению цифр в записи числа). Значение обоих из них не может превосходить 18 (т. к. для наиболь- шего допустимого числа 9999 обе суммы равны 9 + 9 = 18), поэтому для их описания используем

тип **byte**.

Более детальный разбор алгоритма:

1) Вводим число **n**;

2) Присваиваем переменной **right**значение последней цифры числа **n**, потом отбрасываем эту

цифру, затем повторяем то же самое, но на этот раз уже прибавляем добытую цифру к прежнему значению **right**:

right := n mod 10;

n := n div 10;

right := right + n mod 10;

n := n div 10;

3) Присваиваем переменной **left** последнюю цифру **n**, отбрасываем ее и прибавляем к **right** единственную оставшуюся в переменной **n** цифру:

left := n mod 10;

n := n div 10;

left := left + n;

4) Выводим на экран результат сравнения накопленных сумм:

writeln(left = right);

Код:

1. program HappyTicket;

2.

3. var

4. n: word;

5. left, right: byte;

6.

7. begin

8. readln(n);

9. right := n mod 10;

10. n := n div 10;

11. right := right + n mod 10;

12. n := n div 10;

13. left := n mod 10;

14. n := n div 10;

15. left := left + n;

16. writeln(left = right)

17. end.

Задача № 11. Проверить, является ли двоичное представление числа палиндромом

Формулировка.Дано число типа **byte**. Проверить, является ли палиндромом его двоичное представление с учетом того, что сохранены старшие нули. Пример таких чисел: 102 (т. к. 102 = 0110 01102, а это палиндром), 129 (129 = 1000 00012) и т. д.

Решение. Данная задача частично повторяет **задачу 9**. Сходство состоит в том, что и там, и здесь у проверяемых чисел фиксированная разрядность (длина), ведь здесь нам уже задан тип и получено указание сохранить старшие нули, так что в данном случае двоичные представления всех подаваемых на вход программе чисел будут восьмизначными. Но как же упростить решение, чтобы не делать сравнение всех разрядов «в лоб»? Для этого нам нужно вспомнить правило, упомянутое в **задаче 5**, на этот раз несколько уточненное и дополненное:

– *Остаток от деления любого числа* ***x*** *в системе счисления с основанием* ***p*** *на само число* ***p*** *дает*

*нам крайний справа разряд числа **x**.*

*– Умножение любого числа **x** в системе счисления с основанием **p** на само число **p** добавляет числу **x** новый разряд справа.*

Для примера возьмем число 158 в десятичной системе счисления. Мы можем получить его крайнюю цифру справа, которая равна 8, если возьмем остаток от деления 158 на число 10, являющееся в данном случае основанием системы счисления. С другой стороны, если мы умножим 158 на 10, то появляется новый разряд справа, и в результате мы получаем число 1580. Согласно правилу те же самые арифметические законы актуальны и для двоичной системы счисления. А это в свою очередь означает, что мы можем разработать алгоритм наподобие того, который использовался в **задаче 9** для формирования числа, представляющего собой правую половину исходного числа, которая записана в реверсном порядке. Для этого нам нужно использовать четыре переменных для хранения двоичных разрядов правой половины двоичной записи введенного числа, добыть эти самые разряды с удалением их в исходном числе, сформировать из них двоичную реверсную запись и выполнить сравнение. Обозначим эти переменные типа **byte** как **a**, **b**, **c**, и **d**.

Опишем сам алгоритм:

1) Вводим число **n**;

2) Последовательно получаем 4 крайних справа разряда двоичной записи числа **n**: присваиваем их значение соответствующей переменной, а затем отбрасываем в исходном числе:

a := n mod 2;

n := n div 2;

b := n mod 2;

n := n div 2;

c := n mod 2;

n := n div 2;

d := n mod 2;

n := n div 2;

3) Теперь нужно подумать, как видоизменится формула, с помощью которой мы получали реверсную запись числа в **задаче 4** и **задаче 9**. Очевидно, что в десятичной системе счисления реверсную запись четырехзначного числа, разряд единиц которого находится в переменной **k**, разряд десятков – в переменной **l**, сотен – в **m** и тысяч – в **n** мы можем найти по следующей формуле (**x** в данной случае – любая переменная типа **word**): x := 1000 * k + 100 * l + 10 * m + n; Можно представить, что мы формируем четыре числа, которые затем складываем. Первое число 1000 * k – это разряд единиц исходного числа, к которому справа приписано три разряда (три нуля), то есть, трижды произведено умножение на основание системы счисления 10, проще говоря, число **k** умножено на 103. Аналогично, к **l** нужно приписать два нуля, к **m** – один ноль, а **n** оставить без изменения, так как эта цифра будет находиться в разряде единиц формируемого «перевертыша». Вспомнив правило, высказанное немного выше, преобразуем предыдущую формулу для двоичной системы счисления (будем умножать цифры на двойку в соответствующих степенях). Она получится такой (для формирования числа используется переменная **a**): a := 8 * a +

4 * b + 2 * c + d;

4) После применения вышеприведенной строки останется лишь вывести на экран результат сравнения полученных чисел:

writeln(n = a);

Код:

1. program BinaryPalindrome;

2.

3. var

4. n, a, b, c, d: byte;

5.

7. readln(n);

8. a := n mod 2;

9. n := n div 2;

10. b := n mod 2;

11. n := n div 2;

12. c := n mod 2;

13. n := n div 2;

14. d := n mod 2;

15. n := n div 2;

16. a := 8 * a + 4 * b + 2 * c + d;

17. writeln(n = a)

18. end.

Выполним «ручную прокрутку» программы при вводе числа 102. Покажем в таблице, какие значения будут принимать переменные после выполнения соответствующих строк (операторов) кода. Значения переменных для наглядности представлены как в десятичной, так и в двоичной системе счисления (при этом дописаны старшие нули до заполнения тетрады). При этом прочерк означает, что значение переменных на данном шаге не определено, а красным цветом выделены переменные, которые изменяются:

стеме счисления (при этом дописаны старшие нули до заполнения тетрады). При этом прочерк означает, что значение переменных на данном шаге не определено, а красным цветом выделены переменные, которые изменяются:

№ строки	Десятичная система					Двоичная система				
	n	a	b	c	d	n	a	b	c	d
7	102	—	—	—	—	0110 0110	—	—	—	—
8	102	0	—	—	—	0110 0110	0000	—	—	—
9	51	0	—	—	—	0011 0011	0000	—	—	—
10	51	0	1	—	—	0011 0011	0000	0001	—	—
11	25	0	1	—	—	0001 1001	0000	0001	—	—
12	25	0	1	1	—	0001 1001	0000	0001	0001	—
13	12	0	1	1	—	0000 1100	0000	0001	0001	—
14	12	0	1	1	0	0000 1100	0000	0001	0001	0000
15	6	0	1	1	0	0000 0110	0000	0001	0001	0000
16	**6**	**6**	1	1	0	**0000 0110**	**0110**	0001	0001	0000

Задача № 12. Решить квадратное уравнение

Формулировка. Даны вещественные числа **a**, **b** и **c**, причем **a** отлично от 0. Решить квадратное уравнение *ax*2 + *bx* + *c* = 0 или сообщить о том, что действительных решений нет.

Решение. Из алгебры известно, что:

Квадратное уравнение ax2 + bx + c = 0, *выражение D = b*2 – 4*ac – дискриминант:*

– если D > 0, имеет два решения: 1

2

b D x a

− + = , 2

2

b D x a

−− = *;*

– если D = 0, имеет единственное решение: 1

2

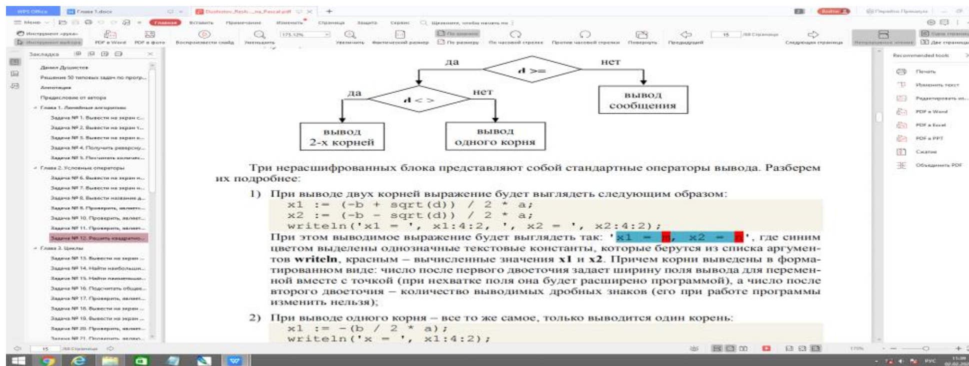
b x a

= − ;

– если D < 0, не имеет действительных решений.

Следовательно, нам необходимо вычислить дискриминант (заведем для него вещественную переменную **d** типа **real**) и в зависимости от его значения организовать ветвления. Сначала нужно проверить, имеет ли уравнение действительные решения (для решений заведем переменные **x1** и **x2** типа **real**). Если да, и если дискриминант не равен нулю, то вычисляем оба решения по формулам, а если дискриминант равен нулю, то вычисляем единственное решение. Если же действительных решений нет, выводим текстовое сообщение об этом. Основной алгоритм можно

проиллюстрировать следующей блок-схемой:



Три нерасшифрованных блока представляют собой стандартные операторы вывода. Разберем их подробнее:

1) При выводе двух корней выражение будет выглядеть следующим образом:

x1 := (-b + sqrt(d)) / 2 * a;

x2 := (-b - sqrt(d)) / 2 * a;

writeln('x1 = ', x1:4:2, ', x2 = ', x2:4:2); При этом выводимое выражение будет выглядеть так: 'x1 = m, x2 = n', где синим цветом выделены однозначные текстовые константы, которые берутся из списка аргументов **writeln**, красным – вычисленные значения **x1** и **x2**. Причем корни выведены в форматированном виде: число после первого двоеточия задает ширину поля вывода для переменной вместе с точкой (при нехватке поля она будет расширено программой), а число после второго двоеточия – количество выводимых дробных знаков (его при работе программы изменить нельзя);

2) При выводе одного корня – все то же самое, только выводится один корень:

x1 := -(b / 2 * a);

writeln('x = ', x1:4:2);

3) При отсутствии действительных корней выводим сообщение:

writeln('No real solutions!');

В итоге внутренний условный оператор с телом включительно будет выглядеть так:

if d <> 0 then begin

x1 := (-b + sqrt(d)) / 2 * a;

x2 := (-b - sqrt(d)) / 2 * a;

writeln('x1 = ', x1:4:2, ', x2 = ', x2:4:2)

end

else begin

x1 := -(b / 2 * a);

```
writeln('x = ', x1:4:2)
end;
```

Код:

```
1. program QuadraticEquation;
2.
3. var
4. a, b, c, d, x1, x2: real;
5.
6. begin
7. readln(a, b, c);
8. d := b * b - 4 * a * c;
9. if d >= 0 then begin
10. if d <> 0 then begin
11. x1 := (-b + sqrt(d)) / 2 * a;
12. x2 := (-b - sqrt(d)) / 2 * a;
13. writeln('x1 = ', x1:4:2, ', x2 = ', x2:4:2)
14. end
15. else begin
16. x1 := -(b / 2 * a);
17. writeln('x = ', x1:4:2)
18. end
19. end
20. else begin
21. writeln('No real solutions!');
22. end
23. end.
```

Циклы

Задача № 13. Вывести на экран все натуральные числа до заданного

Формулировка. Дано натуральное число. Вывести на экран все натуральные числа до заданного включительно.

Решение. Данная задача решается с использованием оператора цикла **for**. Напомним, что с помощью цикла **for** можно совершить заданное количество итераций (повторений) некоторых

операторов, которые синтаксически заключены в содержимое его тела (так называемого тела цикла). При этом некоторая целочисленная переменная изменяется от некоторого стартового значения до некоторого конечного (оба значения включительно), увеличиваясь на единицу с каждым повторением тела цикла. Так как нам необходимо выводить натуральные числа, это означает, что вывод должен всегда начинаться с единицы, и при этом выводятся все следующие за ней натуральные числа до тех пор, пока значение переменной цикла (обычно используют переменную **i**) не достигнет конечного **n** (на последнем шаге значение переменной цикла будет равно **n**). После этого цикл завершится, и будут выполнены те операторы, которые следуют непосредственно за ним. Кстати, не стоит забывать, что после выхода из цикла **for** его переменная цикла считается неопределенной!

Код:

```
1. program FromOneToN;

2.

3. var

5.

6. begin

7. readln(n);

8. for i := 1 to n do begin

9. write(i, ' ')

10. end

11. end.
```

Пусть введено число 5, например. При входе **i** станет равно 1 и будет проверено существование отрезка в заданных границах. Так как 1 меньше 5, то произойдет вход в цикл, и будут выполняться следующие команды, пока **i** не превысит **n**:

1) Выполнение команд в теле цикла;

2) Увеличение **i** на 1;

3) Возвращение на шаг 1.

Нетрудно понять, что в нашем случае **i** будет принимать значения 1, 2, 3, 4, 5 и будет выведена на экран строка '1 2 3 4 5 '. Здесь красным цветом выделены изменяющиеся значения переменной цикла, а синим – выводящаяся неизменной пробельная константа.

Задача № 14. Найти наибольший нетривиальный делитель натурального числа

Формулировка. Дано натуральное число. Найти его наибольший нетривиальный делитель или вывести единицу, если такового нет.

Примечание 1: делителем натурального числа **a** называется натуральное число **b**, на которое **a** делится без остатка. То есть выражение «**b** – делитель **a**» означает: **a / b = k**, причем **k** – натуральное число.

Примечание: нетривиальным делителем называется делитель, который отличен от 1 и от самого

числа (так как на единицу и само на себя делится любое натуральное число).

Решение. Пусть ввод с клавиатуры осуществляется в переменную **n**. Попробуем решить задачу перебором чисел. Для этого возьмем число на единицу меньшее **n** и проверим, делится ли **n** на него. Если да, то выводим результат и выходим из цикла с помощью оператора **break**. Если нет, то снова уменьшаем число на 1 и продолжаем проверку. Если у числа нет нетривиальных делителей, то на каком-то шаге проверка дойдет до единицы, на которую число гарантированно поделится, после чего будет выдан соответствующий условию ответ. Хотя, если говорить точнее, следовало бы начать проверку с числа, равного **n div 2** (чтобы отбросить дробную часть при делении, если **n** нечетно), так как ни одно натуральное число не имеет делителей больших, чем половина это этого числа. В противном случае частное от деления должно

быть натуральным числом между 1 и 2, которого просто не существует. Данная задача также решается через **for**, но через другую его разновидность, и теперь счетчик будет убывать от **n div 2** до 1. Для этого **do** заменится на **downto**, при позиции начального и конечного значений остаются теми же.

Алгоритм на естественном языке:

1) Ввод **n**;

2) Запуск цикла, при котором **i** изменяется от **n div 2** до 1. В цикле:

1. Если **n** делится на **i** (то есть, остаток от деления числа **n** на **i** равен 0), то выводим **i** на экран и выходим из цикла с помощью **break**.

Код:

```
1. program GreatestDiv;

2.

3. var

4. i, n: word;

5.

6. begin

7. readln(n);

8. for i := n div 2 downto 1 do begin

9. if n mod i = 0 then begin

10. writeln(i);

11. break

12. end

13. end

14. end.
```

Кстати, у оператора ветвления **if** в цикле отсутствует **else**-блок. Такой условный оператор называется оператором ветвления с одной ветвью.

Задача № 15. Найти наименьший нетривиальный делитель натурального числа

Формулировка. Дано натуральное число. Найти его наименьший нетривиальный делитель или вывести само это число, если такового нет.

Решение. Задача решается аналогично предыдущей. При этом необходимо начать обычный цикл с увеличением, при котором переменная цикла **i** изменяется от 2 до **n** (такая верхняя граница нужна для того, чтобы цикл всегда заканчивался, так как когда **i** будет равно **n**, выполнится условие **n mod i = 0**). Весь остальной код при этом не отличается.

Код:

```
1. program SmallestDiv;

2.

3. var

4. i, n: word;

5.

6. begin

7. readln(n);

8. for i := 2 to n do begin

9. if n mod i = 0 then begin

10. writeln(i);

11. break

12. end

13. end

14. end.
```

Задача № 16. Подсчитать общее число делителей натурального числа

Формулировка. Дано натуральное число. Подсчитать общее количество его делителей.

Решение. Задача достаточно похожа на две предыдущие. В ней также необходимо провести перебор в цикле некоторого количества натуральных чисел на предмет обнаружения делителей **n**, но при этом необходимо найти не первый из них с какого-либо конца отрезка [1, **n**] (это отрезок, содержащий все числа от 1 до **n** включительно), а посчитать их. Это можно сделать с помощью счетчика **count**, который нужно обнулить непосредственно перед входом в цикл. Затем в условном операторе **if** в случае истинности условия делимости числа **n** (**n mod i = 0**) нужно увеличивать счетчик **count** на единицу (это удобно делать с помощью оператора **inc**).

Алгоритм на естественном языке:

1) Ввод **n**;

2) Обнуление переменной **count** (в силу необходимости работать с ее значением без предварительного присваивания ей какого-либо числа)

3) Запуск цикла, при котором **i** изменяется от 1 до **n**. В цикле:

1. Если **n** делится на **i** (то есть, остаток от деления числа **n** на **i** равен 0), то увеличиваем

значение переменной **count** на 1;

4) Вывод на экран значения переменной **count**.

Код:

```
1. program CountDiv;

2.

3. var

4. i, n, count: word;

5.

6. begin

7. readln(n);

8. count := 0;

9. for i := 1 to n do begin

10. if n mod i = 0 then inc(count)

11. end;

12. writeln(count)

13. end.
```

Задача № 17. Проверить, является ли заданное натуральное число простым

Формулировка. Дано натуральное число. Проверить, является ли оно простым. Примечание: простым называется натуральное число, которое имеет ровно два различных натуральных делителя: единицу и само это число.

Решение. Задача отличается от предыдущей только тем, что вместо вывода на экран числа делителей, содержащегося в переменной **count**, необходимо выполнить проверку равенства счетчика числу 2. Если у числа найдено всего два делителя, то оно простое и нужно вывести положительный ответ, в противном случае – отрицательный ответ. А проверку через условный оператор, как мы уже знаем, можно заменить на вывод результата самого булевского выражения с помощью оператора **write** (**writeln**).

Код:

```
1. program PrimeTest;

2.

3. var

4. i, n, count: word;

5.
```

```
6. begin
7. readln(n);
8. count := 0;
9. for i := 1 to n do begin
10. if n mod i = 0 then inc(count)
11. end;
12. writeln(count = 2)
13. end.
```

Задача № 18. Вывести на экран все простые числа до заданного

Формулировка. Дано натуральное число. Вывести на экран все простые числа до заданного включительно.

Решение. В решении данной задачи используется решение предыдущей. Нам необходимо произвести тест простоты числа, который был описан в решении предыдущей задачи следующим кодом (обозначим его как **код 1**):

```
count := 0;
for i := 1 to n do begin
if n mod i = 0 then inc(count)
end;
writeln(count = 2);
```

Здесь **n** – проверяемое на простоту число. Напомним, что данный фрагмент кода в цикле проверяет, делится ли **n** на каждое **i** в отрезке от 1 до самого **n**, и если **n** делится на **i**, то увеличивает счетчик **count** на 1. Когда цикл заканчивается, на экран выводится результат проверки равенства счетчика числу 2.

В нашем же случае нужно провести проверку на простоту всех натуральных чисел от 1 до заданного числа (обозначим его как **n**). Следовательно, мы должны поместить **код 1** в цикл по всем **k** от 1 до **n**. Также в связи с этим необходимо заменить в **коде 1** идентификатор **n** на **k**, так как в данном решении проверяются на простоту все числа **k**. Кроме того, теперь вместо вывода ответа о подтверждении/опровержении простоты **k** необходимо вывести само это число в случае простоты:

```
if count = 2 then write(k, ' ');
```

Обобщая вышесказанное, приведем алгоритм на естественном языке:

1) Ввод **n**;

2) Запуск цикла, при котором **k** изменяется от 1 до **n**. В цикле:

1. Обнуление переменной **count**;

2. Запуск цикла, при котором **i** изменяется от 1 до **k**. В цикле:

а. Если **k** делится на **i**, то увеличиваем значение переменной **count** на 1;

3. Если **count** = 2, выводим на экран число **k** и символ пробела.

Код:

```
1. program PrimesToN;

2.

3. var

4. i, k, n, count: word;

5.

6. begin

7. readln(n);

8. for k := 1 to n do begin

9. count := 0;

10. for i := 1 to k do begin

11. if k mod i = 0 then inc(count)

12. end;

13. if count = 2 then write(k, ' ')

14. end

15. end.
```

Вычислительная сложность. В данной задаче мы впервые столкнулись с вложенным циклом (когда один цикл запускается внутри другого). Это означает, что наш алгоритм имеет нелинейную сложность (при которой количество выполненных операций равно размерности исходных данных (в нашем случае это **n** – количество чисел) плюс некоторое количество обязательных операторов).

Давайте посчитаем количество выполненных операций в частном случае. Итак, пусть необходимо вывести все натуральные простые числа до числа 5. Очевидно, что проверка числа 1 пройдет в 1 + 2 шага, числа 2 – в 2 + 2 шага, числа 3 – в 3 + 2 шага и т. д. (прибавленная двойка к каждому числу – два обязательных оператора вне внутреннего цикла), так как мы проверяем делители во всем отрезке от 1 до проверяемого числа. В итоге количество проведенных операций (выполненных операторов на низшем уровне) представляет собой сумму: 3 + 4 + 5 + 6 + 7 (также опущен обязательный оператор ввода вне главного (внешнего) цикла). Очевидно, что при выводе всех простых чисел от 1 до **n** приведенная ранее сумма будет такой:

1 + 3 + 5 + 6 + ... + (**n** – 1) + **n** + (**n** + 1) + (**n** + 2),

где вместо многоточия нужно дописать все недостающие члены суммы. Очевидно, что это сумма первых (**n** + 2) членов арифметической прогрессии с вычтенным из нее числом 2. Как известно,

сумма **k** первых членов арифметической прогрессии выражена формулой:

где **a1** – первый член прогрессии, **ak** – последний.

Подставляя в эту формулу наши исходные данные и учитывая недостающее до полной суммы число 2, получаем следующее выражение:

Чтобы найти асимптотическую сложность алгоритма, отбросим коэффициенты при перемен-

ных и слагаемые с низшими степенями (оставив, соответственно, слагаемое с самой высокой степенью). При этом у нас остается член ***n2*** , значит, асимптотическая сложность алгоритма – **O(n2)**.

Конечно, в дальнейшем мы не будем так подробно находить асимптотическую сложность алгоритмов, а тем более, вычислять количество требуемых операций, что интересно только теоретически. На самом деле, конечно, нас интересует лишь порядок роста времени работы алгоритма (количества необходимых операций), который можно выявить из анализа вложенности циклов и некоторых других характеристик.

Задача № 19. Вывести на экран первых n простых чисел

Формулировка. Дано натуральное число **n**. Вывести на экран **n** первых простых чисел.

Например, при вводе числа 10 программа должна вывести ответ: 2 3 5 7 11 13 17 19 23 29

Решение. Эта задача похожа на предыдущую тем, что здесь нам тоже необходимо проверить

все натуральные числа на некотором отрезке, на этот раз используя еще один счетчик для подсчета найденных простых. Однако на этот раз мы не можем узнать, сколько придется проверить чисел, пока не насчитается **n** простых. Следовательно, здесь нам не подходит цикл **for**, так как мы не можем посчитать необходимое количество итераций.

Здесь нам поможет цикл **while**. Напомним, что тело цикла **while** повторяется до тех пор, пока остается истинным булевское выражение в его заголовке (поэтому его еще называют циклом с предусловием). Так как **while** не имеет переменной цикла, которая увеличивается на 1 с каждым следующим шагом, то при его использовании необходимо обеспечить изменение некоторых переменных в теле цикла, от которых зависит условие в его заголовке, таким образом, чтобы при достижении требуемого результата оно стало ложным. Так как мы должны найти и вывести **n** первых простых чисел, подсчитывая их, и с каждый найденным простым числом некоторый счетчик (пусть это будет **primes** с начальным значением 0) увеличивается на 1, то условием в заголовке **while** будет выражение **primes < n**. Иными словами, цикл продолжается, пока число найденных чисел меньше требуемого. На последнем же шаге будет выведено последнее число, **primes** станет равным **n** и следующего шага цикла уже не будет, так как условие в его заголовке

станет ложным. На этом работа программы будет закончена. Проверяться на простоту будет некоторое число **k**, которое с **каждой** итерацией цикла обязательно будет увеличиваться на 1 (таким образом, мы будем двигаться по отрезку натуральных чисел, пока не выберем из него заданное количество простых).

Алгоритм на естественном языке:

1) Ввод **n**;

2) Обнуление переменной **primes**;

3) Присваивание переменной **k** значения 1;

4) Запуск цикла с предусловием **primes < n**. В цикле:

1. Обнуление переменной **count**;

2. Запуск цикла, при котором **i** изменяется от 1 до **k**. В цикле:

a. Если **k** делится на **i**, то увеличиваем значение переменной **count** на 1;

3. Если **count** = 2, выводим на экран число **k**, символ пробела и увеличиваем значение

счетчика **primes** на 1;

4. Увеличиваем значение **k** на 1.

Код:

```
1. program FirstNPrimes;
2.
3. var
4. i, k, n, count, primes: word;
5.
6. begin
7. readln(n);
8. k := 1;
9. primes := 0;
10. while primes < n do begin
11. count := 0;
12. for i := 1 to k do begin
13. if k mod i = 0 then inc(count)
14. end;
15. if count = 2 then begin
16. write(k, ' ');
```

17. inc(primes)

18. end;

19. inc(k)

20. end

21. end.

Выполним ручную прокрутку алгоритма, взяв в качестве **n** число 2. При этом будем уже попривычке красным цветом обозначать переменные, изменившиеся после выполнения данной строки, а прочерком те, которые на данном шаге не определены, так как алгоритм до них еще «не дошел». При повторении шагов цикла итерации явно считать не будем (хотя легко увидеть, что их номерам полностью соответствует изменяющаяся после каждого очередного выполнения тела переменная **k**), и в таблице будет указана лишь та строка, которая выполняется. На тех шагах, на которых переменные не изменяются, будем пояснять смысл выполняющихся операторов.

Для наглядности все же отделим друг от друга четные и нечетные шаги основного цикла **while**, при этом его внутренний цикл будем считать самоочевидным и в строке 12-14 будем фиксировать те значения переменных, которые будут получены по выходу из него.

№ строки	n	k	primes	i	count
7	2	-	-	-	-
8	2	1	-	-	-
9	2	1	0	-	-
10	**(primes < n) = true** – входим в цикл				
11	2	1	0	-	0
12-14	2	1	0	От 1 до 1	1
15-18	**(count = 2) = false**				
19	2	2	0	-	0
10	**(primes < n) = true** – входим в цикл				
11	2	2	0	-	0
12-14	2	2	0	От 1 до 2	2
15	**(count = 2) = true**				

16	Вывод числа **k** (то есть 2)				
17-18	2	2	1	-	2
19	2	3	1	-	2
10	**(primes < n) = true** – входим в цикл				
11	2	3	1	-	0
12-14	2	3	1	От 1 до 3	2
15	**(count = 2) = true**				
16	Вывод числа **k** (то есть 3)				
17-18	2	3	2	-	2
19	2	4	2	-	2
10	**(primes < n) = false** – программа завершена				

Как видим, описать действия даже такого элементарного алгоритма и даже при столь малых исходных данных – достаточно трудоемкая и трудно проверяемая задача, поэтому очень важно понимать логику самой программы и уметь представлять себе целостную картину ее поведения с минимальными потребностями в моделировании.

Вычислительная сложность. Так как в данной задаче в главном цикле присутствует вложенный, в котором происходит ровно **k** операций (сложность этой конструкции мы определили в **задаче 18**), то его сложность явно не меньше **O(n2)** и превышает ее, так как нам явно необходимо выполнить более чем **n** шагов главного цикла. При этом точность расчета сложности зависит от распределения простых чисел, которое описывается с помощью достаточно сложных математических приемов и утверждений, так что мы не будем далее рассматривать эту задачу.

Задача № 20. Проверить, является ли заданное натуральное число совершенным

Формулировка. Дано натуральное число. Проверить, является ли оно совершенным.

Примечание: совершенным числом называется натуральное число, равное сумме всех своих собственных делителей (то есть натуральных делителей, отличных от самого числа). Например, 6 – совершенное число, оно имеет три собственных делителя: 1, 2, 3, и их сумма равна 1 + 2 + 3 = 6.

Решение. Эта задача напоминает **задачу 16**, в которой нужно было найти количество всех натуральных делителей заданного числа. Напомним код ее основной части (назовем его **кодом 1**):

```
count := 0;

for i := 1 to n do begin
```

if n mod i = 0 then inc(count)

end;

Как видно, в этом цикле проверяется делимость числа **n** на все числа от 1 до **n**, причем при каждом выполнении условия делимости увеличивается на 1 значение счетчика **count** с помощью функции **inc**. Чтобы переделать этот код под текущую задачу, нужно вместо инкрементации (увеличения значения) переменной-счетчика прибавлять числовые значения самих делителей к некоторой переменной для хранения суммы (обычно ее мнемонически называют **sum**, что в пер. с англ. означает «сумма»). В связи с этим оператор if n mod i = 0 then inc(count); в **коде 1** теперь уже будет выглядеть так:

if n mod i = 0 then sum := sum + i;

Кроме того, чтобы не учитывалось само число **n** при суммировании его делителей (насколько мы помним, этот делитель не учитывается в рамках определения совершенного числа), цикл должен продолжаться не до **n**, а до **n – 1**. Правда, если говорить точнее, то цикл следовало бы проводить до **n div 2** (также это обсуждалось в **задаче 14**), так как любое число **n** не может иметь больших делителей, иначе частное от деления должно быть несуществующим натуральным число между 1 и 2.

Единственное, что останется теперь сделать – это вывести ответ, сравнив число **n** с суммой его делителей **sum** как результат булевского выражения через **writeln**: writeln(n = sum);

Код:

```
1. program PerfectNumbers;

2.

3. var

4. i, n, count: word;

5.

6. begin

7. readln(n);

8. count := 0;

9. for i := 1 to n div 2 do begin

10. if n mod i = 0 then sum := sum + i

11. end;

12. writeln(n = sum)

13. end.
```

Задача № 21. Проверить, являются ли два натуральных числа дружественными

Формулировка. Даны два натуральных числа. Проверить, являются ли они дружественными.

Примечание: дружественными числами называются два различных натуральных числа, для которых сумма всех собственных делителей первого числа равна второму числу и сумма всех

собственных делителей второго числа равна первому числу.

Например, 220 и 284 – пара дружественных чисел, потому что:

Сумма собственных делителей 220: 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284

Сумма собственных делителей 284: 1 + 2 + 4 + 71 + 142 = 220

Решение. Эта задача напоминает **задачу 19**, так как в ней мы тоже считали сумму собственных делителей введенного числа, а затем сравнивали эту сумму с самим числом, проверяя его на предмет совершенности. В данном же случае нам нужно найти не только сумму собственных делителей первого числа (обозначим число как **n1**, а сумму его делителей **sum1**), но и второго числа (возьмем обозначения **n2** и **sum2** соответственно). Тогда ответом в задаче послужит сравнение: **(n1 = sum2) and (n2 = sum1)**. Кстати, здесь впервые в нашем повествовании мы используем логические операции (напомним, что логическое выражение **X1 and X2** принимает значение истины тогда и только тогда, когда истинны булевские выражения **X1** и **X2**, а в остальных случаях оно принимает ложное значение).

Однако предложенную схему можно упростить. Покажем это на примере: пусть даны числа 8 и 4. Считаем сумму собственных делителей числа 8: 1 + 2 + 4 = 7. Это число отлично от 4, поэтому пара уже не соответствует определению дружественных чисел. Можно сразу вывести отрицательный ответ, избежав подсчета суммы делителей второго числа. Если были бы даны числа 8 и 7, то необходимо было бы вычислить сумму собственных делителей числа 7, она равна 1 (так как оно простое). Теперь необходимо сравнить сумму собственных делителей второго с первым числом: так

как 1 отлично от 8, числа не дружественные.

Покажем на блок-схеме, как можно разветвить программу (вычисление обоих сумм не изображается):

вычисление **sum1**

sum1 =

вычисление **sum1**

вывод

'False'

writeln(**sum2 = n1**)

Таким образом, без логических операций можно и обойтись.

Код:

1. program AmicableTest;

2.

3. var

4. i, n1, n2, sum1, sum2: word;

5.

6. begin

7. readln(n1, n2);

8. for i := 1 to n1 div 2 do begin

9. if n1 mod i = 0 then sum1 := sum1 + i

10. end;

11. if sum1 = n2 then begin

12. for i := 1 to n2 div 2 do begin

13. if n2 mod i = 0 then sum2 := sum2 + i

14. end;

15. writeln(sum2 = n1)

16. end

17. else begin

18. writeln('False')

19. end

20. end.

Задача № 22. Найти наибольший общий делитель двух натуральных чисел

Формулировка. Даны два натуральных числа. Найти их наибольший общий делитель.

Примечание: наибольшим общим делителем (сокращенно пишут НОД) двух натуральных чисел **m** и **n** называется наибольший из их общих делителей. Обозначение: НОД(m, n).

Примечание 2: общим делителем двух натуральных чисел называется натуральное число, на которое натуральное число, которое является делителем обоих этих чисел.

Например, найдем НОД(12, 8):

Выпишем все делители числа 12: 1, 2, 3, 4, 6, 12;

Выпишем все делители числа 8: 1, 2, 4, 8;

Выпишем все общие делители чисел 12 и 8: 1, 2, 4. Из них наибольшее число – 4. Это и есть

НОД чисел 12 и 8.

Решение. Конечно, при решении мы не будем выписывать делители и выбирать нужный. В

принципе, ее можно было бы решить как **задачу 14**, начав цикл с наименьшего из двух заданных чисел (так как оно тоже может быть НОД, например, НОД(12, 4) = 4). Но мы воспользуемся так называемым алгоритмом Евклида нахождения НОД, который выведен с помощью математических методов. В самом простейшем случае для заданных чисел **m** и **n** он выглядит так:

*1) Если **m** неравно **n**, перейти к шагу 2, в противном случае вывести **m** и закончить алгоритм;*

*2) Если **m > n**, заменить **m** на **m – n**, в противном случае заменить **n** на **n – m;***

3) Перейти на шаг 1

Как видим, в шаге 2 большее из двух текущих чисел заменяется разностью большего и меньшего.

Приведем пример для чисел 12 и 8:

a. Так как 12 > 8, заменим 12 на 12 – 8 = 4;

b. Так как 8 > 4, заменим 8 на 8 – 4 = 4;

c. 4 = 4, конец.

Не проводя каких-либо рассуждений над алгоритмом и не доказывая его корректности, переведем

его описание в более близкую для языка **Pascal** форму:

Алгоритм на естественном языке:

1) Ввод **m** и **n**;

2) Запуск цикла с предусловием **m <> n**. В цикле:

1. Если **m > n**, то переменной **m** присвоить **m – n**, иначе переменной **n** присвоить **n – m**;

3) Вывод **m**:

Код:

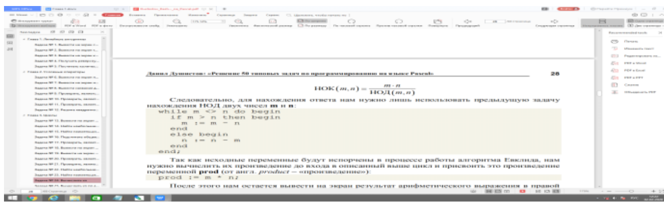
```
1. program GreatestCommonDiv;
2.
3. var
4. m, n: word;
5.
6. begin
7. readln(m, n);
8. while m <> n do begin
9. if m > n then begin
10. m := m - n
11. end
12. else begin
13. n := n - m
14. end
15. end;
16. writeln(m)
17. end.
```

Задача № 23. Найти наименьшее общее кратное двух натуральных чисел

Формулировка. Даны два натуральных числа. Найти их наименьшее общее кратное.

Примечание: наименьшим общим кратным двух чисел **m** и **n** называется наименьшее натуральное число, которое делится на **m** и **n**. Обозначение: НОК(m, n)

Решение. Из теории чисел известно, что НОК(m, n) связан с НОД(m, n) следующим образом:


«Решение 50 типовых задач по программированию на языке Pascal» 28

$$НОК(m,n) = \frac{m \cdot n}{НОД(m,n)}$$

Следовательно, для нахождения ответа нам нужно лишь использовать предыдущую задачу нахождения НОД двух чисел **m** и **n**:

```
while m <> n do begin
  if m > n then begin
    m := m - n
  end
  else begin
    n := n - m
  end
end;
```

Так как исходные переменные будут испорчены в процессе работы алгоритма Евклида, нам нужно вычислить их произведение до входа в описанный выше цикл и присвоить это произведение переменной **prod** (от англ. *product* – «произведение»):

```
prod := m * n;
```

После этого нам остается вывести на экран результат арифметического выражения в правой

Следовательно, для нахождения ответа нам нужно лишь использовать предыдущую задачу

нахождения НОД двух чисел **m** и **n**:

while m <> n do begin

if m > n then begin

m := m - n

end

else begin

n := n - m

end

end;

Так как исходные переменные будут испорчены в процессе работы алгоритма Евклида, нам нужно вычислить их произведение до входа в описанный выше цикл и присвоить это произведение переменной **prod** (от англ. *product* – «произведение»):

prod := m * n;

После этого нам остается вывести на экран результат арифметического выражения в правой части нашей формулы. В качестве самого НОД будет использоваться переменная **m**:

writeln(prod div m);

Кстати, деление в формуле будет целочисленным (через **div**) именно потому, что если два числа делятся на некоторое число, то и их произведение также делится на него.

Код:

1. program LeastCommonMult;

2.

3. var

4. m, n, prod: word;

5.

6. begin

7. readln(m, n);

8. prod := m * n;

9. while m <> n do begin

```
10. if m > n then begin
11. m := m - n
12. end
13. else begin
14. n := n - m
15. end
16. end;
17. writeln(prod div m)
18. end.
```

Задача № 24. Вычислить xn

Формулировка. Даны натуральные числа **x** и **n** (которое также может быть равно 0). Вычислить ***xn***.

Решение. Для того чтобы решить эту задачу, вспомним определение степени с натуральным показателем: запись ***xn*** означает, что число **x** умножено само на себя **n** раз.

Сразу из определения видно, что здесь заранее известно количество повторений при вычислении результата, так что задача легко решается через цикл **for**. Выходит, мы копируем исходное число **x** в некоторую переменную **res** (от англ. *result* – «результат»), а затем просто умножаем его на **x** **n** раз? Не стоит торопиться с ответом.

Рассмотрим пример: 34 = 3 * 3 * 3 * 3 = 81. Если посмотреть на эту запись, то мы видим, что возведение в четвертую степень как выражение содержит четыре слагаемых, но только три операции, так как мы с первого шага домножаем число 3 на три тройки. Тогда реализация идеи из абзаца выше будет давать число в степени на 1 больше, чем требуется.

Какой можно придумать выход? Например, можно сократить цикл на одну операцию, но что тогда будет при вычислении нулевой степени? Как известно, любое число в нулевой степени дает 1, а здесь при вводе в качестве **n** нуля приведет к тому, что не будет осуществлен вход в цикл (так как не существует целочисленного отрезка от 1 до 0) и в итоге на выход так и пойдет исходное число **x**.

А что, если изменить схему умножения так: 34 = 1 * 3 * 3 * 3 * 3 = 81? Так мы можем сравнять показатель степени и число требуемых операций, да и с нулевой степенью все становится просто, так как при вводе в качестве **n** нуля не будет осуществляться вход в цикл и на выход в программе пойдет число 1!

Теперь алгоритм на естественном языке:

1) Ввод **x** и **n**;

2) Присваивание переменной **res** числа 1;

3) Запуск цикла, при котором **i** изменяется от 1 до **n**. В цикле:

1. Присваиваем переменной **res** значение **res** * **x**;

4) Вывод переменной **res**.

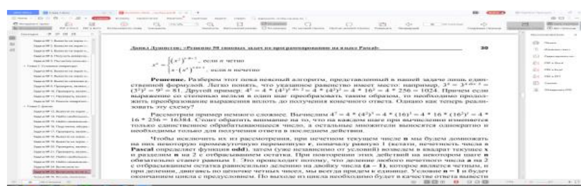
Код:

```
1. program Exponentiation;
2.
3. var
4. x, n, i, res: word;
5.
6. begin
7. readln(x, n);
8. res := 1;
9. for i := 1 to n do begin
10. res := res * x
11. end;
12. writeln(res)
13. end.
```

Кстати, стоит понимать, что объявление переменной **res** при использовании типа **word** достаточно условно, так как этот тип принимает значения от 0 до 65535, что на единицу меньше числа 2562 , хотя вводить в программу можно числа, предполагающие возведение в более высокую степень. Так как в условии задачи не сказано ничего о том, в каком числовом промежутке по **x** и **n** она должна выдавать корректный ответ, оставим это в таком виде, достаточном для проверки приложения на работоспособность.

Задача № 25. Вычислить x^n по алгоритму быстрого возведения в степень

Формулировка. Даны натуральные числа **x** и **n**. Вычислить x^n , используя алгоритм быстрого возведения в степень:



Решение. Разберем этот пока неясный алгоритм, представленный в нашей задаче лишь единственной формулой. Легко понять, что указанное равенство имеет место: например, 34 = 34 div 2 = (32)2 = 92 = 81. Другой пример: 45 = 4 * (42) 5 div 2 = 4 * (42)2 = 4 * 162 = 4 * 256 = 1024. Причем если выражение со степенью нельзя в один шаг преобразовать таким образом, то необходимо продолжить преобразование выражения вплоть до получения конечного ответа. Однако как теперь реализовать эту схему?

Рассмотрим пример немного сложнее. Вычислим 47 = 4 * (42)3 = 4 * (16)3 = 4 * 16 * (162)1 = 4 *

16 * 256 = 16384. Стоит обратить внимание на то, что на каждом шаге при вычислении изменяется только единственное обрабатывающееся число, а остальные множители выносятся однократно и необходимы только для получения ответа в последнем действии.

Чтобы исключить их из рассмотрения, при нечетном текущем числе **n** мы будем домножать на них некоторую промежуточную переменную **r**, поначалу равную 1 (кстати, нечетность числа в **Pascal** определяет функция **odd**), затем (уже независимо от условий) возведем в квадрат текущее **x** и разделим **n** на 2 с отбрасыванием остатка. При повторении этих действий на некотором шаге **n** обязательно станет равным 1. Это происходит потому, что деление любого нечетного числа **a** на 2 с отбрасыванием остатка равносильно делению на двойку числа **(a – 1)**, которое является четным, и при делении, двигаясь по цепочке четных чисел, мы всегда придем к единице. Условие **n = 1** и будет окончанием цикла с предусловием. По выходе из цикла необходимо будет в качестве ответа вывести последнее значение **x**, умноженное на **r**.

Теперь алгоритм на естественном языке:

1) Ввод **x** и **n**;

2) Присваивание переменной **r** числа 1;

5) Запуск цикла с предусловием **n <> 1**. В цикле:

1. Если **n** нечетно, домножаем **r** на **x**;

2. Переменной **x** присваиваем значение **x * x**;

3. Переменной **n** присваиваем результат от деления этой переменной на 2 с отбрасыванием остатка;

3) Вывод выражения **x * r**.

Код:

```
1. program FastExponentiation;
2.
3. var
4. x, n, r: word;
5.
6. begin
7. readln(x, n);
8. r := 1;
9. while n <> 1 do begin
10. if odd(n) then r := r * x;
11. x := x * x;
12. n := n div 2
```

13. end;

14. writeln(x * r)

15. end.