2 Общие сведения о языке программирования Free Pascal

В этой главе читатель познакомится со структурой проекта в среде Lazarus и основными элементами языка программирования Free Pascal: переменными, константами, их типами, основными операциями и функциями языка.

2.1 Структура проекта Lazarus

Любой *проект* в Lazarus – это совокупность файлов, из которых создается единый *выполняемый файл*. В простейшем случае список файлов проекта имеет вид:

- файл описания проекта (`.lpi`);
- файл проекта (`.lpr`);
- файл ресурсов (`.lrs`);
- модуль формы (`.lfm`);
- программный модуль (`.pas`);

После *компиляции* программы из всех файлов проекта создается единый выполняемый файл, имя этого файла совпадает с именем проекта.

Программный модуль, или просто модуль, – это отдельно компилируемая программная единица, которая представляет собой набор типов данных, констант, переменных, процедур и функций. Любой модуль имеет следующую структуру:

```
unit имя_модуля;    //Заголовок модуля.
interface
//Раздел описаний.
implementation
//Раздел реализаций.
end.              //Конец модуля.
```

Заголовок модуля – это зарезервированное слово `unit`, за которым следует имя модуля и точка с запятой. В разделе описаний, который открывается служебным словом `interface`, описывают *программные элементы* – типы, классы, процедуры и функции:

```
interface
uses список_модулей;
type список_типов;
```

```
const список_констант;
var список_переменных;
procedure имя_процедуры;
…
function имя_функции;
…
```

Раздел `implementation` содержит *программный код*, реализующий механизм работы описанных программных элементов (тексты процедур обработки событий, процедуры и функции, созданные программистом). *Процедуры* и *функции* в Lazarus также построены по модульному принципу[23].

Наряду с визуальными приложениями, Lazarus позволяет разрабатывать и обычные консольные приложения, которые также могут быть созданы в оболочке Free Pascal, и в текстовом редакторе Geany. Авторы настоятельно рекомендуют начинать изучение программирования именно с создания консольных приложений. Поэтому рассмотрим подробно структуру консольного приложения.

2.2 Структура консольного приложения

Структура консольного приложения имеет вид:

```
заголовок программы;
uses modul1, modul2, …, moduln;
раздел описаний;
тело программы.
```

Заголовок программы состоит из служебного слова *program*, имени программы, образованного по правилам использования идентификаторов (см. п. 2.3), и точки с запятой, например:

```
program my_prog001;
```

Предложение `uses modul1, modul2, …, moduln` предназначено для подключения модулей. В модулях находятся функции и процедуры языка. Для использования функций и процедур, находящихся в модуле, необходимо в тексте программы подключить их с помощью предложения `uses`.

Раздел описаний включает следующие подразделы:

```
раздел описания констант;
раздел описания типов;
раздел описания переменных;
```

23 Подробно о процедурах и функциях см. в главе 4.

раздел описания процедур и функций.

В языке Free Pascal должны быть описаны все переменные, типы, константы, которые будут использоваться программой. В стандартном языке Pascal порядок следования разделов в программе жестко установлен, во Free Pascal такого строгого требования нет. В программе может быть несколько разделов описания констант, переменных и т.д., но все они должны быть до тела программы. Более подробно структуру консольной программы на языке Free Pascal можно представить следующим образом:

```
program имя_программы;
uses modul1, modul2, …, moduln;
const описания_констант;
type описания_типов;
var описания_переменных;
begin
    операторы_языка;
end.
```

Тело программы начинается со слова begin, затем следуют операторы языка Pascal, реализующие алгоритм решаемой задачи. Операторы в языке Pascal отделяются друг от друга точкой с запятой и могут располагаться в одну строчку или начинаться с новой строки (в этом случае их также необходимо разделять точкой с запятой). Назначение символа « ; » - отделение операторов друг от друга. Тело программы заканчивается служебным словом end. Несмотря на то что операторы могут располагаться в строке как угодно, рекомендуется размещать их по одному в строке, а в случае сложных операторов отводить для каждого несколько строк. Рассмотрим более подробно структуру программы:

```
program имя_программы;
uses modul1, modul2, …, moduln;
const описания_констант;
type описания_типов;
var описания_переменных;
begin
оператор_1;
оператор_2;
...
end.
```

Приведем пример текста программы на Free Pascal:

```
program one;
const a=7;
var b,c: real;
begin
 c:=a+2; b:=c-a*sin(a)
end.
```

2.3 Элементы языка

Программа на языке Free Pascal может содержать следующие символы:

- латинские буквы A, B, C…, x, y, z;
- цифры 0, 1, 2…, 9;
- специальные символы + ,–, /, =, <, >, [,], ., (,), ;, :, {, }, $, #, _, @, ‘, ^.

Из символов алфавита формируют ключевые слова и идентификаторы. *Ключевые слова* – это зарезервированные слова, которые имеют специальное значение для компилятора. Примером ключевых слов являются операторы языка, типы данных и т.п. Ключевые слова используются только так, как они определены при описании языка. *Идентификатор* – это имя программного объекта[24], представляющее собой совокупность букв, цифр и символа подчеркивания. Первый символ идентификатора – буква или знак подчеркивания, но не цифра. Идентификатор не может содержать пробел. Прописные и строчные буквы в именах не различаются, например `ABC`, `abc`, `Abc` – одно и то же имя. Каждое имя (идентификатор) должно быть уникальным и не совпадать с ключевыми словами.

В тексте программы можно использовать *комментарии*. Комментарий это текст, который компилятор игнорирует. Комментарии используют для пояснений программного кода или для временного отключения команд программного кода при отладке. Комментарии бывают однострочные и многострочные. Однострочный комментарий начинается с двух символов «косая черта» `//` и заканчивается символом перехода на новую строку. Многострочный комментарий заключен в фигурные скобки `{ }` или располагается между парами символов `(*` и `*)`. Понятно, что фигурные скобки `{ }` или символы `(*` и `*)`

24 К программным объектам относятся константы, переменные, метки, процедуры, функции, модули и программы.

можно использовать и для однострочного комментария.

Например:

```
{Комментарий может
 выглядеть так!}
(*Или так.*)
//А если вы используете такой способ,
//то каждая строка должна начинаться
//с двух символов «косая черта».
```

2.4 Данные в языке Free Pascal

Для решения задачи в любой программе выполняется обработка каких-либо данных. Они хранятся в памяти компьютера и могут быть самых различных типов: целыми и вещественными числами, символами, строками, массивами. *Типы данных* определяют способ хранения чисел или символов в памяти компьютера. Они задают размер ячейки, в которую будет записано то или иное значение, определяя тем самым его максимальную величину или точность задания. Область памяти (ячейка), в которой хранится значение определенного типа, называется *переменной*. У переменной есть *имя* (*идентификатор*), *тип* и *значение*. Имя служит для обращения к области памяти, в которой хранится значение. Во время выполнения программы значение переменной можно изменить. Перед использованием любая переменная должна быть описана. Описание переменной в языке Free Pascal осуществляется с помощью служебного слова `var`:

```
var имя_переменной: тип_переменной;
```

Если объявляется несколько переменных одного типа, то описание выглядит следующим образом:

```
var переменная_1,…,переменная_N: тип_переменных;
```

Например:

```
var
//Объявлена целочисленная переменная.
ha: integer;
//Объявлены две вещественные переменные.
hb, c: real;
```

Константа – это величина, которая не изменяет своего значения в процессе выполнения программы. Тип константы определяется ее значением. Описание константы имеет вид:

```
const имя_константы = значение;
```

Например:

```
const
h=3;          //Целочисленная константа.
bk=-7.521;    //Вещественная константа.
c='abcde';    //Символьная константа.
```

Рассмотрим основные типы данных.

2.4.1 Символьный тип данных

Данные *символьного типа* в памяти компьютера всегда занимают один байт. Это связано с тем, что обычно под величину символьного типа отводят столько памяти, сколько необходимо для хранения одного символа.

Описывают символьный тип с помощью служебного слова `char`. Например:

```
var
 c: char;
```

В тексте программы значения переменных и константы символьного типа должны быть заключены в апострофы: `'a'`, `'b'`, `'+'`.

2.4.2 Целочисленный тип данных

Целочисленные типы данных могут занимать в памяти компьютера один, два, четыре или восемь байт. Диапазоны значений данных целочисленного типа представлены в табл. 2.1.

Таблица 2.1 Целочисленные типы данных

Тип	Диапазон	Размер
`Byte`	0 .. 255	1 байт
`Word`	0 .. 65535	2 байта
`LongWord`	0 .. 4294967295	4 байта
`ShortInt`	-128 .. 127	1 байт
`Integer`	-2147483648 .. 2147483647	4 байта
`LongInt`	-2147483648 .. 2147483647	4 байта
`Smallint`	-32768 .. 32767	2 байта
`Int64`	-2^{63} .. 2^{63}	8 байт
`Cardinal`	0 .. 4294967295	4 байта

Описание целочисленных переменных в программе может быть таким:

```
var
b: byte;
i, j: integer;
W: word;
L_1, L_2: longint;
```

2.4.3 Вещественный тип данных

Внутреннее представление *вещественного числа* в памяти компьютера отличается от представления целого числа. Оно представлено в экспоненциальной форме $m\text{E} \pm p$, где m – мантисса (целое или дробное число с десятичной точкой), p – порядок (целое число)[25]. Чтобы перейти от экспоненциальной формы к обычному представлению числа[26], необходимо мантиссу умножить на десять в степени порядок. Например:

$-36.142\text{E} + 2 = -36.142 \cdot 10^2 = -3614.2$;

$7.25\text{E} - 5 = 7.25 \cdot 10^{-5} = 0.0000725$.

Вещественное число в Pascal может занимать от четырех до десяти байтов. Диапазоны значений вещественного типа представлены в табл. 2.2.

Таблица 2.2 Вещественные типы данных

Тип	Диапазон	Количество значащих цифр	Размер
`Single`	1.5E-45 .. 3.4E+38	7 — 8	4 байта
`Real`	2.9E-39 .. 1.7E+38	15 — 16	8 байтов
`Double`	5.0E-324 .. 1.7E+308	15 — 16	8 байтов
`Extended`	3.4E-4932 .. 3.4E+4932	19 — 20	10 байтов
`Comp`	-2^{63} .. 2^{63}	19 — 20	8 байтов
`Currency`	-922337203685477.5808 .. 922337203685477.5807	19 — 20	8 байтов

Примеры описания вещественных переменных:

```
var
r1, r2: real; D: double;
```

25 Действия над числами, представленными в экспоненциальной форме, называют арифметикой с плавающей точкой, так как положение десятичной точки меняется в зависимости от порядка числа.

26 Число в обычном его представлении называют числом с фиксированной точкой.

2.4.4 Тип дата-время

Тип данных дата-время `TDateTime` предназначен для одновременного хранения даты и времени. В памяти компьютера он занимает восемь байтов и фактически представляет собой вещественное число, где в целой части хранится дата, а в дробной – время.

2.4.5 Логический тип данных

Данные *логического типа* могут принимать только два значения: истина (`true`) или ложь (`false`). В стандартном языке Pascal был определен лишь один логический тип данных – `boolean`. Логические типы данных, определенные в среде Lazarus, представлены в табл. 2.3.

Таблица 2.3 Логические типы данных

Тип	Размер
`Boolean`	1 байт
`ByteBool`	1 байт
`WordBool`	2 байта
`LongBool`	4 байта

Пример объявления логической переменной:

```
var
 FL: boolean;
```

2.4.6 Создание новых типов данных

Несмотря на достаточно мощную структуру встроенных типов данных, в среде Lazarus предусмотрен механизм создания *новых типов*. Для этого используют служебное слово `type`:

```
type новый_тип_данных = определение_типа;
```

Когда новый тип данных создан, можно объявлять соответствующие ему переменные:

```
var список_переменных: новый_тип_данных;
```

Применение этого механизма мы рассмотрим в следующих параграфах.

2.4.7 Перечислимый тип данных

Перечислимый тип задается непосредственным перечислением значений, которые он может принимать:

```
var имя_переменной:(знач_1,знач_2,…,знач_N);
```

Такой тип может быть полезен, если необходимо описать данное, которое принимает ограниченное число значений. Например:

```
var
animal: (fox, rabbit);
color: (yellow, blue, green);
```

Применение перечислимых типов делает программу нагляднее:

```
//Создание нового типа данных – времена года.
type
 year_times = (winter, spring, summer, autumn);
//Описание соответствующей переменной.
var  yt: year_times;
```

2.4.8 Интервальный тип

Интервальный тип задается границами своих значений внутри базового типа:

```
var  имя_переменной: мин_знач .. макс_знач;
```

Обратите внимание, что в данной записи два символа точки рассматриваются как один, поэтому пробел между ними не допускается. Кроме того, границы диапазона могут быть только целого или символьного типов и левая граница диапазона не должна превышать правую. Например:

```
var
 date: 1..31;
 symb: 'a'..'h';
```

Применим механизм создания нового типа данных:

```
type
//Создание перечислимого типа данных –
//дни недели.
Week_days = (Mo, Tu, We, Th, Fr, Sa, Su);
//Создание интервального типа данных –
//рабочие дни.
Working_days = Mo.. Fr
//Описание соответствующей переменной.
var
 days: Working_days;
```

2.4.9 Структурированные типы

Структурированный тип данных характеризуется множественностью образующих его элементов. В языке Free Pascal это массивы, строки, записи, множества и файлы.

Массив – совокупность данных одного и того же типа[27]. Число

27 Подробно о массивах см. в главе 5.

элементов массива фиксируется при описании типа и в процессе выполнения программы не изменяется. Для описания массива используют ключевые слова `array … of`:

```
имя: array [список_индексов] of тип_данных;
```

где:

• `имя` – любой допустимый идентификатор;

• `тип_данных` – любой тип языка.

• `список индексов` – перечисление диапазонов изменения номеров элементов массива; количество диапазонов совпадает с количеством измерений массива; диапазоны отделяются друг от друга запятой, а границы диапазона, представляющие собой интервальный тип данных, отделяют друг от друга двумя символами точки:

```
[индекс1_начальный..индекс1_конечный,
    индекс2_начальный..индекс2_конечный, …, ]
```

Например:

```
var
//Одномерный массив из 10 целых чисел.
a:array [1..10] of byte;
//Двумерный массив вещественных чисел
//(3 строки, 3 столбца).
b:array [1..3, 1..3] of real;
```

Еще один способ описать массив – создать новый тип данных. Например, так:

```
type
//Объявляется новый тип данных –
//трехмерный массив целых чисел.
massiv=array [0..4, 1..2, 3..5] of word;
//Описывается переменная соответствующего типа.
var
M: massiv;
```

Для *доступа к элементу массива* достаточно указать его порядковый номер, а если массив многомерный (например, таблица), то несколько номеров:

```
имя_массива[номер_элемента]
```

Например: `a[5]`, `b[2, 1]`, `M[3, 2, 4]`.

Строка – последовательность символов. В Lazarus строка трактуется как массив символов, то есть каждый символ строки пронумерован, начиная с единицы.

При использовании в выражениях строка заключается в апострофы. Описывают переменные строкового типа так:

```
имя_переменной: string;
```

или:

```
имя_переменной: string[длина_строки];
```

Например:

```
const S='СТРОКА';
var
 Str1: string;
 Str2: string[255];
 Stroka: string[100];
```

Если при описании строковой переменной длина строки не указывается, это означает, что она составляет 255 символов. В приведенном примере строки `Str1` и `Str2` одинаковой длины.

Запись[28] – это структура данных, состоящая из фиксированного количества компонентов, называемых полями записи. В отличие от массива поля записи могут быть разного типа. При объявлении типа записи используют ключевые слова `record … end`:

```
имя_записи = record список_полей end;
```

здесь, `имя_ записи` – любой допустимый идентификатор, `список_полей` – описания полей записи. Например:

```
//Описана структура записи.
//Каждая запись содержит информацию
//о студенте и состоит из двух полей –
//имя и возраст.
type
 student = record
     name: string;
     age: byte;
     end;
var
//Объявлены соответствующие переменные –
//три объекта типа запись.
 a, b, c: student;
```

Доступ к полям записи осуществляется с помощью составного имени:

```
имя_записи.имя_поля
```

28 Подробно о структурах см. в главе 1 .

Например:

```
a.name:='Ivanov Ivan';
a.age:=18;
b.name:=a.name;
```

Оператор присоединения `with имя_записи do` упрощает доступ к полям записи:

```
with a do
begin
name:='Petrov Petr';
age:=19;
end;
```

Множество – это набор логически связанных друг с другом объектов. Количество элементов множества может изменяться от 0 до 255. Множество, не содержащее элементов, называется пустым. Для описания множества используют ключевые слова `set of`:

```
имя_множества = set of базовый_тип_данных;
```

Например:

```
type  TwoNumbers = set of 0..1;
var   Num1, Num2, Num3: TwoNumbers;
```

Файл[29] – это именованная область внешней памяти компьютера. Файл содержит компоненты одного типа (любого типа, кроме файлов). Длина созданного файла не оговаривается при его объявлении и ограничивается только емкостью диска, на котором он хранится. В Lazarus можно объявить *типизированный файл*:

```
имя_файловой_переменной = file of тип_данных;
```

бестиповый файл:

```
имя_файловой_переменной = file;
```

и *текстовый файл*:

```
имя_файловой_переменной = TextFile;
```

Например:

```
var
f1: file of byte;
f2: file;
f3: TextFile;
```

2.4.10 Указатели

Как правило, при обработке оператора объявления переменной

```
имя_переменной: тип_переменной
```

29 Подробно о файлах см. в главе 7 .

компилятор автоматически выделяет память под переменную имя_переменной в соответствии с указанным типом. Доступ к объявленной переменной осуществляется по ее имени. При этом все обращения к переменной заменяются адресом ячейки памяти, в которой хранится ее значение. При завершении подпрограммы, в которой была описана переменная, память автоматически освобождается.

Доступ к значению переменной можно получить иным способом – определить собственные переменные для хранения адресов памяти. Такие переменные называют указателями.

Указатель[30] – это переменная, значением которой является адрес памяти, где хранится объект определенного типа (другая переменная). Как и любая переменная, указатель должен быть объявлен. При объявлении указателей всегда указывается тип объекта, который будет храниться по данному адресу:

```
var имя_переменной: ^тип;
```

Такие указатели называют *типизированными*. Например:

```
var p: ^integer;
//По адресу, записанному в переменной p
//будет храниться переменная типа int
//или, другими словами p, указывает
//на тип данных integer.
```

В языке Free Pascal можно объявить указатель, не связанный с каким-либо конкретным типом данных. Для этого применяют служебное слово pointer:

```
var имя_переменной: pointer;
```

Например:

```
var x, y: pointer;
```

Подобные указатели называют *нетипизированными* и используют для обработки данных, структура и тип которых меняется в процессе выполнения программы, то есть динамически.

2.5 Операции и выражения

Выражение задает порядок выполнения действий над данными и состоит из операндов (констант, переменных, обращений к функциям), круглых скобок и знаков операций, например

```
a+b*sin(cos(x)).
```

30 Подробно об указателях см. 5.11

В табл. 2.4 представлены *основные операции* языка программирования Free Pascal.

Таблица 2.4. Основные операции языка Free Pascal

Операция	**Действие**	**Тип операндов**	**Тип результата**
+	сложение	целый/вещественный	целый/вещественный
+	сцепление строк	строковый	строковый
−	вычитание	целый/вещественный	целый/вещественный
*	умножение	целый/вещественный	целый/вещественный
/	деление	целый/вещественный	вещественный
div	целочисленное деление	целый	целый
mod	остаток от деления	целый	целый
not	отрицание	целый/логический	целый/логический
and	логическое И	целый/логический	целый/логический
or	логическое ИЛИ	целый/логический	целый/логический
xor	исключающее ИЛИ	целый/логический	целый/логический
shl	сдвиг влево	целый	целый
shr	сдвиг вправо	целый	целый
in	вхождение в множество	множество	логический
<	меньше	числовой/символьный/логический	логический
>	больше	числовой/символьный/логический	логический
<=	меньше или равно	числовой/символьный/логический	логический
>=	больше или равно	числовой/символьный/логический	логический
=	равно	числовой/символьный/логический	логический
<>	не равно	числовой/символьный/логический	логический

В сложных выражениях порядок, в котором выполняются операции, соответствует приоритету операций. В языке Free Pascal приняты следующие приоритеты:

1. `not`.
2. `*`, `/`, `div`, `mod`, `and`, `shl`, `shr`.
3. `+`, `-`, `or`, `xor`.
4. `=`, `<>`, `>`, `<`, `>=`, `<=`.

Использование скобок в выражениях позволяет менять порядок вычислений.

Перейдем к подробному рассмотрению основных операций.

2.5.1 Арифметические операции

Операции `+`, `-`, `*`, `/` относят к *арифметическим операциям*. Их назначение понятно и не требует дополнительных пояснений.

Операции целочисленной арифметики (применяется только к целочисленным операндам):

• `div` – целочисленное деление (возвращает целую часть частного, дробная часть отбрасывается), например, 17 `div` 10=1;

• `mod` – остаток от деления, например, 17 `mod` 10=7.

К *операциям битовой арифметики* относятся следующие операции: `and`, `or`, `xor`, `not`, `shl`, `shr`. В операциях битовой арифметики действия происходят над двоичным представлением целых чисел.

Арифметическое И (`and`)[31]. Оба операнда переводятся в двоичную систему, затем над ними происходит логическое поразрядное умножение операндов по следующим правилам:

1 `and` 1=1, 1 `and` 0=0, 0 `and` 1=0, 0 `and` 0=0.

Например, если A=14 и B=24, то их двоичное представление – A=0000000000001110 и B=0000000000011000. В результате логического умножения A `and` B получим 0000000000001000 или 8 в десятичной системе счисления (рис. 2.1). Таким образом, A&B=14&24=8.

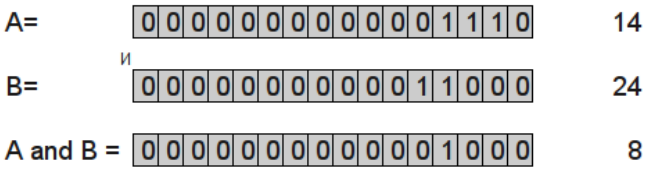

Рисунок 2.1: Пример логического умножения A and B

31 Конъюнкция.

Арифметическое ИЛИ (`or`)[32]. Здесь также оба операнда переводятся в двоичную систему, после чего над ними происходит логическое поразрядное сложение операндов по следующим правилам:

1 `or` 1=1, 1 `or` 0=1, 0 `or` 1=1, 0 `or` 0=0.

Например, результат логического сложения чисел A=14 и B=24 будет равен A `or` B=30 (рис. 2.2).

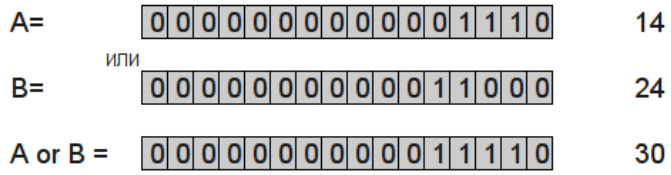

Рисунок 2.2: Пример логического сложения A or B

Арифметическое исключающее ИЛИ (`xor`). Оба операнда переводятся в двоичную систему, после чего над ними происходит логическая поразрядная операция `xor` по следующим правилам:

1 `xor` 1=0, 1 `xor` 0=1, 0 `xor` 1=1, 0 `xor` 0=0.

Арифметическое отрицание (`not`). Эта операция выполняется над одним операндом. Применение операции `not` вызывает побитную инверсию двоичного представления числа (рис. 2.3).

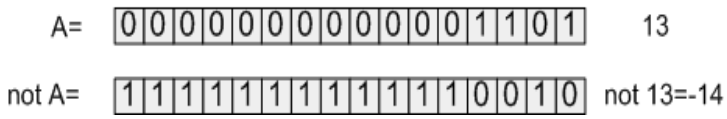

Рисунок 2.3: Пример арифметического отрицания not A

Сдвиг влево (`M shl L`). Двоичное представление числа M сдвигается влево на L позиций. Рассмотрим операцию 15 `shl` 3. Число 15 в двоичной системе имеет вид 1111. При сдвиге его на 3 позиции влево получим 1111000. В десятичной системе это двоичное число равно 120. Итак, 15 `shl` 3 =120 (рис. 2.4).

Рисунок 2.4: Сдвиг влево 15 shl 3

32 Дизъюнкция

Заметим, что сдвиг на один разряд влево соответствует умножению на два, на два разряда – умножению на четыре, на три – умножению на восемь. Таким образом, операция M `shl` L эквивалентна умножению числа M на 2 в степени L.

Сдвиг вправо (`M shr L`). В этом случае двоичное представление числа M сдвигается вправо на L позиций, что эквивалентно целочисленному делению числа M на 2 в степени L. Например, 15 `shr` 1=7 (рис. 2.5), 15 `shr` 3= 2.

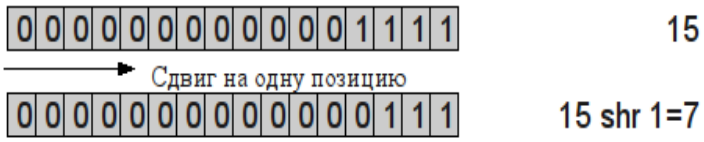

Рисунок 2.5: Сдвиг вправо 15 shr 1

2.5.2 Операции отношения

Операции отношения применяются к двум операндам и возвращают в качестве результата логическое значение. Таких операций семь: >, >=, <, <=, =, <>, `in`. Результат операции отношения – логическое значение `true` (истина) или `false` (ложь).

Назначение операций >, >=, <, <=, =, <> понятно (см. табл. 2.4). Поясним, как работает операция `in`. Первым операндом этой операции должно быть любое выражение, вторым – множество, состоящее из элементов того же типа. Результат операции `true` (истина), если левый операнд принадлежит множеству, указанному справа.

2.5.3 Логические операции

В языке Free Pascal определены следующие *логические операции* `or`, `and`, `xor`, `not`. Логические операции выполняются над логическими значениями `true` (истина) и `false` (ложь). В табл. 2.5 приведены результаты логических операций.

Таблица 2.5. Логические операции

A	B	Not A	A and B	A or B	A xor B
t	t	f	t	t	f
t	f	f	f	t	t
f	t	t	f	t	t
f	f	t	f	f	f

В логических выражениях могут использоваться операции отношения, логические и арифметические.

2.5.4 Операции над указателями

При работе с указателями используют операции получения адреса и разадресации.

Операция получения адреса `@` возвращает адрес своего операнда. Например:

```
Var
a:real; //Объявлена вещественная переменная a
adr_a:^real; //Объявлен указатель на тип real
…
//Оператор записывает в переменную adr_a
//адрес переменной a
adr_a:=@a;
```

Операция разадресации `^` возвращает значение переменной, хранящееся по заданному адресу:

```
Var
a:real; //Объявлена вещественная переменная a
adr_a:^real; //Объявлен указатель на тип real
…
//Оператор записывает в переменную a
//значение, хранящееся по адресу adr_a.
a:=adr_a^;
```

2.6 Стандартные функции

В языке Free Pascal определены *стандартные функции* над арифметическими операндами (табл. 2.6):

Таблица 2.6. Некоторые арифметические функции

Обозначение	Тип аргументов	Тип результата	Действие
`abs(x)`	целый/вещественный	целый/вещественный	модуль числа
`sin(x)`	вещественный	вещественный	синус
`cos(x)`	вещественный	вещественный	косинус
`arctan(x)`	вещественный	вещественный	арктангенс
`pi`	без аргумента	вещественный	число π
`exp(x)`	вещественный	вещественный	экспонента e^x
`ln(x)`	вещественный	вещественный	натуральный логарифм

Обозначение	Тип аргументов	Тип результата	Действие
`sqr(x)`	вещественный	вещественный	квадрат числа
`sqrt(x)`	вещественный	вещественный	корень квадратный
`int(x)`	вещественный	вещественный	целая часть числа
`frac(x)`	вещественный	вещественный	дробная часть числа
`round(x)`	вещественный	целый	округление числа
`trunc(x)`	вещественный	целый	отсекание дробной части числа
`random(n)`	целый	целый	случайное число от 0 до n
Функции, определенные в программном модуле Math[33]			
`arccos(x)`	вещественный	вещественный	арккосинус
`arcsin(x)`	вещественный	вещественный	арксинус
`arccot(x)`	вещественный	вещественный	арккотангенс
`arctan2(y,x)`	вещественный	вещественный	арктангенс y/x
`cosecans(x)`	вещественный	вещественный	косеканс
`sec(x)`	вещественный	вещественный	секанс
`cot(x)`	вещественный	вещественный	котангенс
`tan(x)`	вещественный	вещественный	тангенс
`lnXP1(x)`	вещественный	вещественный	логарифм натуральный от (x+1)
`log10(x)`	вещественный	вещественный	десятичный логарифм
`log2(x)`	вещественный	вещественный	логарифм по основанию два
`logN(n,x)`	вещественный	вещественный	логарифм от x по основанию n

Определенную проблему представляет возведение X в степень n. Если значение степени `n` – целое положительное число, то можно n раз перемножить X (что дает более точный результат и при целом `n` предпочтительней) или воспользоваться формулой[34]:

33 Эти функции будут работать только в том случае, если в тексте основной программы после ключевого слова Unit указать имя Math.

34 Формула формируется так: логарифмируем выражение x^n, получается n ln(x), экспоненцируем последнее.

$$\begin{cases} X^n = e^{n\cdot\ln(X)}, X>0 \\ X^n = -e^{n\cdot\ln(X)}, X<0 \end{cases},$$

которая программируется с помощью стандартных функций языка:

- `exp(n*ln(x))` – для положительного *X*;
- `-exp(n*ln(abs(x)))` – для отрицательного *X*.

Данную же формулу можно использовать для возведения *X* в дробную степень *n*, где *n* – обыкновенная правильная дробь вида *k*/*l*, а знаменатель *l* — нечетный. Если знаменатель *l* – четный, это означает извлечение корня четной степени, следовательно, есть ограничения на выполнение операции: подкоренное выражение не должно быть отрицательным.

При возведении числа *X* в отрицательную степень следует помнить, что

$$x^{-n} = \frac{1}{x^n} .$$

Таким образом, для программирования выражения, содержащего возведение в степень, надо внимательно проанализировать значения, которые могут принимать *X* и *n*, так как в некоторых случаях возведение *X* в степень *n* невыполнимо.

Некоторые *функции*, предназначенные для *работы со строками*, представлены в табл. 2.7.

Таблица 2.7. Функции обработки строк

Обозначение	**Тип аргументов**	**Тип результата**	**Действие**
Работа со строками			
`length(S)`	строка	целое	текущая длина строки `S`
`concat(S1,S2, …)`	строки	строка	объединение строк `S1`, `S2`, …
`copy(S, n, m)`	строка, целое, целое	строка	копирование `n` символов строки `S` начиная с `m`-й позиции
`delete(S, n, m)`	строка, целое, целое	строка	удаление `n` символов из строки `S` начиная с `m`-й позиции

Обозначение	Тип аргументов	Тип результата	Действие
`insert(S, n, m)`	строка, целое, целое	строка	вставка `n` символов в строку `S` начиная с `m`-й позиции
`pos(S1, S2)`	строки	целое	номер позиции, с которой начинается вхождение `S2` в `S1`
`chr(x)`	целое	символ	возвращает символ с кодом `x`
`ord(c)`	символ	целое	возвращает код символа `c`
Преобразование строк в другие типы			
`StrToDateTame(S)`	строка	дата и время	преобразует символы из строки `s` в дату и время
`StrToFloat(S)`	строка	вещественное	преобразует символы из строки `s` в вещественное число
`StrToInt(S)`	строка	целое	преобразует символы из строки `s` в вещественное число
`Val(S,X,Kod)`	строка		преобразует строку символов `S` во внутреннее представление числовой переменной `X`, если преобразование прошло успешно, `Kod`=0.
Обратное преобразование			
`DataTimeToStr(V)`	дата и время	строка	преобразует дату и время в строку.
`FloatToStr(V)`	вещественное	строка	преобразует вещественное число в строку
`IntToStr(V)`	целое	строка	преобразует целочисленное число в строку
`FloatToStrF( V, F,P,D)`	вещественное	строка	преобразует вещественное число `V` в строку символов с учетом формата `F` и параметров `P`, `D`.

Поясним назначение функции `FloatToStrF(V,F,P,D)`. Обычно ее используют для форматированного вывода вещественного числа. В табл. 2.8 содержатся значения параметров этой функции.

Таблица 2.8. Параметры функции `FloatToStrF`

Формат	Назначение
`ffExponent`	Экспоненциальная форма представления числа, *P* – мантисса, *D* – порядок: 1.2345E+10.
`ffFixed`	Число в формате с фиксированной точкой, *P* – общее количество цифр в представлении числа, *D* – количество цифр в дробной части: 12.345.
`ffGtneral`	Универсальный формат, использует наиболее удобную форму представления числа.
`ffNumber`	Число в формате с фикс. точкой, использует символ разделителя тысяч при выводе больших чисел.
`ffCurency`	Денежный формат, соответствует `ffNumber`, но в конце ставит символ денежной единицы.

Примером работы функции `FloatToStrF` служит фрагмент программы:

```
var
n:integer;
m:real;
St:string;
begin
n:=5;
m:=4.8;
St:='Иванов А.';
//Выражение chr(13) – символ окончания строки.
//Для вывода вещественного числа m отводится
//четыре позиции вместе с точкой,
//две позиции после точки.
Label1.Caption:='Студент'+St+'сдал '
        +IntToStr(n)+'экзаменов.'+chr(13)+
                'Средний балл составил '+
                   FloatToStrF(m,ffFixed,4,2);
End;
```

Результатом работы программы будет фраза:

```
Студент Иванов А. сдал 5 экзаменов.
Средний балл составил 4.80.
```

В табл. 2.9 приведены функции, предназначенные для работы с датой и временем.

Таблица 2.9. Функции для работы с датой и временем

Обозначение	Тип аргументов	Тип результата	Действие
`date`	без аргумента	дата-время	возвращает текущую дату
`now`	без аргумента	дата-время	возвращает текущую дату и время
`time`	без аргумента	дата-время	возвращает текущее время

Если в процессе работы над программой возникает необходимость в создании переменной, размер которой неизвестен заранее, используют *динамическую память*[35].

В языке Free Pascal *операции распределения памяти* осуществляются по средством функций представленных в табл. 2.10.

Таблица 2.10. Функции для работы с памятью

Обозначение	Действие
`adr(x)`	Возвращает адрес аргумента `x`.
`dispose(p)`	Возвращает фрагмент динамической памяти, который ранее был зарезервирован за типизированным указателем `p`.
`Freemen(p,size)`	Освобождает фрагмент динамической памяти, который ранее был зарезервирован за бестиповым указателем `p`.
`GetMem(p,size)`	Резервирует за бестиповым указателем p фрагмент динамической памяти размера `size.`
`New(p)`	Резервирует фрагмент динамической памяти для размещения переменной и помещает в типизированный указатель `p` адрес первого байта.
`SizeOf(x)`	Возвращает длину в байтах внутреннего представления указанного объекта `x`.

С подробным описанием приведенных в этой главе функций и множеством других функций можно ознакомиться в справочной системе Lazarus.

Рассмотрим решение задачи с использованием стандартных функций.

ЗАДАЧА 2.1. Известны длины сторон треугольника a, b и c. Вычислить площадь S, периметр P и величины углов α, β и γ треугольника (рис. 2.6).

35 Динамическая память – это область оперативной памяти, которая выделяется программе при ее работе.

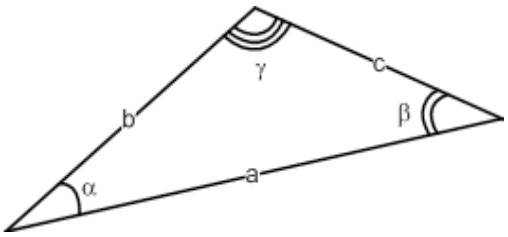

Рисунок 2.6: Иллюстрация к задаче 2.1

Прежде чем приступить к написанию программы, вспомним математические формулы, необходимые для решения задачи.

Для вычисления площади треугольника применим теорему Герона:

$$S=\sqrt{r(r-a)(r-b)(r-c)}\ ,$$

где полупериметр:

$$r=\frac{a+b+c}{2}\ ;$$

один из углов найдем по теореме косинусов:

$$cos(\alpha)=\frac{b^2+c^2-a^2}{2\cdot b\cdot c}\ ;$$

второй — по теореме синусов:

$$sin(\beta)=\frac{b}{a}\cdot\sin(a)\ ;$$

третий — по формуле:

$$\gamma=\pi-(\alpha+\beta);$$

Решение задачи можно разбить на следующие этапы:

1. Определение значений *a*, *b* и *c* (ввод величин *a, b* и *c* в память компьютера).
2. Расчет значений *S*, *P*, *α, β и γ* по приведенным формулам.
3. Вывод значений *S*, *P*, *α, β и γ*.

Процесс разработки несложного программного интерфейса описан в главе 1. Попытайтесь самостоятельно разработать внешний вид данной программы. Разместите на форме десять меток, три поля ввода и одну кнопку. Измените их заголовки (свойство `Caption`) в соответствии с табл. 2.11. В результате форма должна выглядеть так, как показано на рис. 2.7.

Таблица 2.11. Заголовки компонентов формы (рис. 2.7)

Компонент	**Свойство** `Caption`
`Form1`	`Параметры треугольника`
`Label1`	`Введите длины сторон`
`Label2`	`a=`
`Label3`	`b=`
`Label4`	`c=`
`Label5`	`Величины углов`
`Label6`	`alfa=`
`Label7`	`betta=`
`Label8`	`gamma=`

Компонент	**Свойство** Caption
Label9	Периметр P=
Label10	Площадь S=
Button1	ВЫЧИСЛИТЬ

Рисунок 2.7: Проект формы к задаче 2.1

Итак, проект формы готов. В окне программного кода среды Lazarus автоматически сформировал структуру модуля, перечислив названия основных разделов. Двойной щелчок по кнопке Вычислить приведет к созданию процедуры

TForm1.Button1Click

в разделе implementation:

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

и ее описанию в разделе interface. Понятно, что созданная процедура не содержит ни одной команды. Задача программиста заполнить шаблон описаниями и операторами. Все команды, указанные в процедуре между словами begin и end, будут выполнены при щелчке по кнопке Выполнить.

В нашем случае процедура TForm1.Button1Click будет иметь вид:

```
procedure TForm1.Button1Click(Sender: TObject);
//Описание переменных:
//   a, b, c – стороны треугольника;
//   alfa, betta, gamma – углы треугольника;
//       S - площадь треугольника;
//       r - полупериметр треугольника
//Все переменные вещественного типа.
var a, b, c, alfa, betta, gamma, S, r: real;
begin
//Из полей ввода Edit1, Edit2, Edit3
//считываются введенные строки,
```

```
//с помощью функции StrToFloat(x)
//преобразовываются в вещественные числа
//и записываются в переменные a, b, c.
a:=StrToFloat(Edit1.Text);
b:=StrToFloat(Edit2.Text);
c:=StrToFloat(Edit3.Text);
//Вычисление значения полупериметра.
R:=(a+b+c)/2;
//Вычисление значения площади,
//для вычисления применяется функция:
//  sqrt(x) – корень квадратный из x.
S:=sqrt(r*(r-a)*(r-b)*(r-c));
//Вычисление значения угла alfa в радианах.
//Для вычисления применяем функции:
//         arccos(x) - арккосинус x;
//         sqr(x) – возведение x в квадрат.
alfa:=arccos((sqr(b)+sqr(c)-sqr(a))/2/b/c);
//Вычисление значения угла betta в радианах.
//Для вычисления применяем функции:
//          arcsin(x) - арксинус x;
betta:=arcsin(b/a*sin(alfa));
//Вычисление значения угла gamma в радианах.
//Математическая постоянная определена
//функцией без аргумента pi.
gamma:=pi-(alfa+betta);
//Перевод радиан в градусы.
alfa:=alfa*180/pi;
betta:=betta*180/pi;
gamma:=gamma*180/pi;
//Для вывода результатов вычислений используем
//операцию слияния строк «+»
//и функцию FloatToStrF(x), которая
//преобразовывает вещественную переменную x
//в строку и выводит ее в указанном формате,
//в нашем случае под переменную отводится
//три позиции, включая точку
//и ноль позиций после точки.
//Величины углов в градусах выводятся на форму
```

```
//в соответствующие объекты типа надпись.
Label6.Caption:='alfa='+
                FloatToStrF(alfa,ffFixed,3,0);
Label7.Caption:='betta='+
                FloatToStrF(betta,ffFixed,3,0);
Label8.Caption:='gamma='+
                FloatToStrF(gamma,ffFixed,3,0);
//Используем функцию FloatToStrF(x)
//для форматированного вывода, в нашем случае
//под все число отводится пять позиций,
//включая точку, и две позиций после точки.
//Значения площади и периметра
//выводятся на форму.
Label9.Caption:='Периметр P='+
                FloatToStrF(2*r,ffFixed,5,2);
Label10.Caption:='Площадь S='+
                FloatToStrF(S,ffFixed,5,2);
end;
```

Обратите внимание, что было написано всего десять команд, предназначенных для решения поставленной задачи. Остальной текст в окне редактора создается автоматически. В результате весь программный код имеет вид:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
Classes, SysUtils, LResources, Forms, Controls,
             Graphics, Dialogs, StdCtrls, Math;
type
 { TForm1 }
 TForm1 = class(TForm)
     Button1: TButton;
     Edit1: TEdit;
     Edit2: TEdit;
     Edit3: TEdit;
     Label1: TLabel;
     Label10: TLabel;
     Label2: TLabel;
```

```
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
 procedure Button1Click(Sender: TObject);
 private
    { private declarations }
 public
    { public declarations }
 end;
var
 Form1: TForm1;
implementation
{ TForm1 }
procedure TForm1.Button1Click(Sender: TObject);
 var a, b, c, alfa, betta, gamma, S, r: real;
 begin
    a:=StrToFloat(Edit1.Text);
    b:=StrToFloat(Edit2.Text);
    c:=StrToFloat(Edit3.Text);
    r:=(a+b+c)/2;
    S:=sqrt(r*(r-a)*(r-b)*(r-c));
    alfa:=arccos((sqr(b)+sqr(c)-
                        sqr(a))/2/b/c);
    betta:=arcsin(b/a*sin(alfa));
    gamma:=pi-(alfa+betta);
    alfa:=alfa*180/pi; betta:=betta*180/pi;
    gamma:=gamma*180/pi;
    Label6.Caption:='alfa='+
               FloatToStrF(alfa,ffFixed,3,0);
    Label7.Caption:='betta='+
               FloatToStrF(betta,ffFixed,3,0);
    Label8.Caption:='gamma='+
               FloatToStrF(gamma,ffFixed,3,0);
    Label9.Caption:='Периметр P='+
```

```
                FloatToStrF(2*r,ffFixed,5,2);
    Label10.Caption:='Площадь S='+
                FloatToStrF(S,ffFixed,5,2);
 end;
initialization
 {$I unit1.lrs}
end.
```

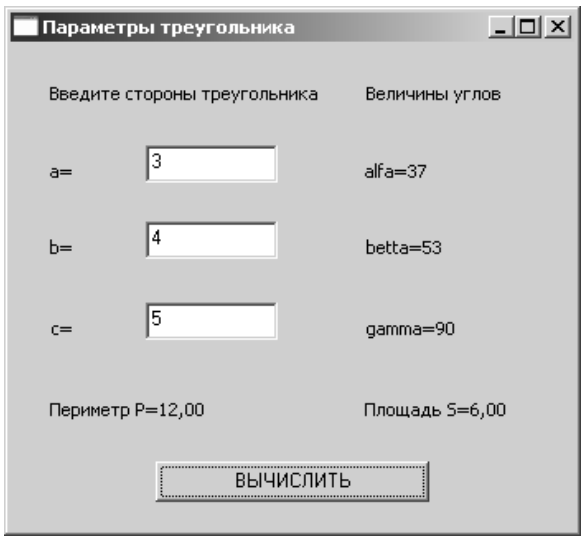

Рисунок 2.8: Результат работы программы к задаче 2.1

На рис. 2.8 представлено диалоговое окно, которое появится, если запустить эту программу, щелкнув по кнопке Вычислить.

Теперь напишем консольное приложение для решения этой задачи. Для этого запустим текстовый редактор Geany. Выполним команду **Файл — New with Tamplate — Pascal Source File**.

В открывшемся окне редактора введем следующий текст программы.

```
program pr3;
//Подключение модуля для работы
//с математическими функциями, см. табл.2.6.
uses math;
//Описание всех используемых переменных.
var a,b,c,r,s,alfa,betta,gamma:real;
BEGIN
//Ввод сторон трегольника.
writeln('Введите стороны треугольника');
readln(a,b,c);
//Вычисление значения полупериметра.
R:=(a+b+c)/2;
//Вычисление значения площади,
//для вычисления применяется функция:
//   sqrt(x) – корень квадратный из x.
S:=sqrt(r*(r-a)*(r-b)*(r-c));
//Вычисление значения угла alfa в радианах.
```

```
//Для вычисления применяем функции:
//           arccos(x) - арккосинус x;
//           sqr(x) - возведение x в квадрат.
alfa:=arccos((sqr(b)+sqr(c)-sqr(a))/2/b/c);
//Вычисление значения угла betta в радианах.
//Для вычисления применяем функции:
//           arcsin(x) - арксинус x;
betta:=arcsin(b/a*sin(alfa));
//Вычисление значения угла gamma в радианах.
//Математическая постоянная определена
//функцией без аргумента pi.
gamma:=pi-(alfa+betta);
//Перевод из радиан в градусы.
alfa:=alfa*180/pi;
betta:=betta*180/pi;
gamma:=gamma*180/pi;
//Вывод площади и сторон треугольника.
writeln('Площадь треугольника =',S:6:2);
writeln('alf=',alfa:6:2,
        ' bet=',betta:6:2,' gam=',gamma:6:2);
end.
```

Для компиляции программы в Geany необходимо выполнить команду **Построить — Собрать (F8)**, для запуска — **Построить — Выполнить (F5)**. На рис. 2.9 представлены результаты работы программы.

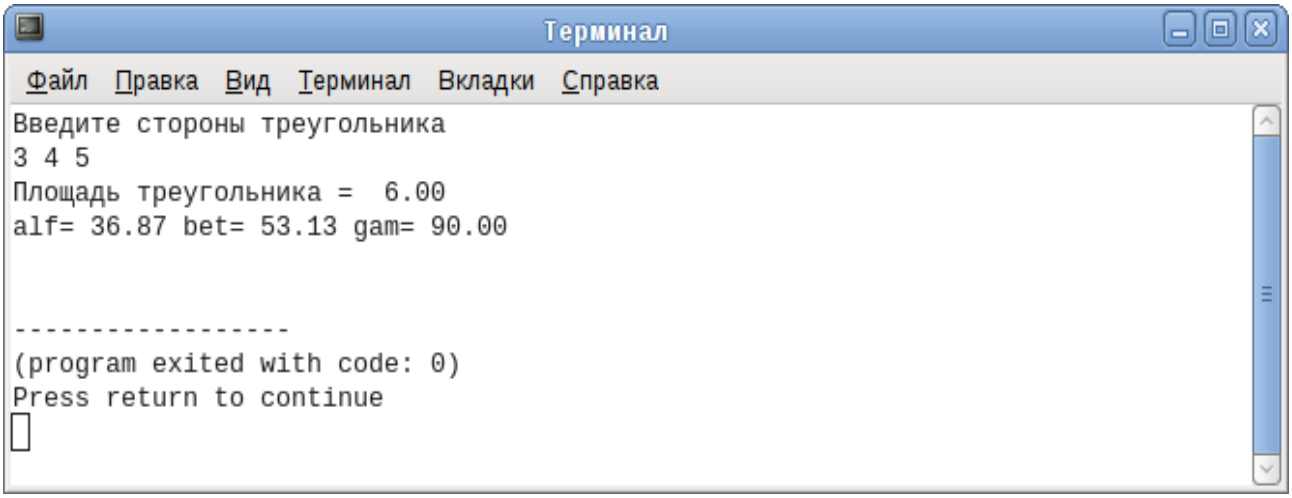

Рисунок 2.9: Результаты работы консольного приложения решения задачи 2.1

2.7 Задачи для самостоятельного решения

Разработать программу в среде программирования Lazarus. Для каждой задачи создать интерфейс, соответствующий условию.

1. Заданы два катета прямоугольного треугольника. Найти гипотенузу и углы треугольника.
2. Известна гипотенуза c и прилежащий угол α прямоугольного треугольника. Найти площадь треугольника.
3. Известна диагональ квадрата d. Вычислить площадь S и периметр P квадрата.
4. Известна диагональ прямоугольника d и угол α между диагональю и большей стороной. Вычислить площадь S прямоугольника.
5. Треугольник задан величинами своих сторон – a, b, c. Найти углы треугольника – α, β, γ.
6. Тело имеет форму параллелепипеда с высотой h. Прямоугольник в основании имеет диагональ d. Известно, что диагонали основания пересекаются под углом α. Найти объем тела V и площадь поверхности S.
7. В треугольнике известен катет a и площадь S. Найти величину гипотенузы c, второго катета b и углов α и β.
8. Известна площадь квадрата S. Вычислить сторону квадрата a, диагональ d и площадь S1 описанного вокруг квадрата круга.
9. В равнобедренном треугольнике известно основание c и угол при нем α. Найти площадь треугольника S и величину боковой стороны a.
10. Известны координаты трех вершин прямоугольника ABCD: A(x1,y1), B(x2,y2) и C(x3,y3). Найти его площадь и периметр.
11. Заданы два катета прямоугольного треугольника. Вычислить его площадь и периметр.
12. Известна гипотенуза c и противолежащий угол α прямоугольного треугольника. Найти периметр треугольника.
13. Известна диагональ ромба d. Вычислить его площадь S и периметр P.
14. Известна длина диагоналей прямоугольника d и угол α между ними. Вычислить площадь S прямоугольника.
15. В прямоугольном треугольнике известен катет b и площадь S. Вычислить периметр треугольника.

16. Известно значение периметра P равностороннего треугольника. Вычислить его площадь.

19. Задан периметр квадрата P. Вычислить сторону квадрата a, диагональ d и площадь S.

20. В равнобедренном треугольнике известно основание c и высота h. Найти площадь треугольника S и периметр P.

21. Известны координаты вершин треугольника ABC: A(x1,y1), B(x2,y2) и C(x3,y3). Найти его площадь и периметр.

22. Металлический слиток имеет форму цилиндра, площадь поверхности S, высота h, плотность α. Вычислить массу m слитка.

23. Задан первый член арифметической прогрессии и ее шаг. Вычислить сумму n членов арифметической прогрессии и значение n-го члена.

24. Задан первый член геометрической прогрессии и ее знаменатель. Вычислить сумму n членов геометрической прогрессии и значение n-го члена.

25. Тело падает с высоты h. Какова его скорость в момент соприкосновения с землей и когда это произойдет.