Глава 3
ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ КОНСТРУКЦИИ

§ 8. Алгоритмы и исполнители

Алгоритмы построения чертежей человек разрабатывает с глубокой древности. Появление чертежей связано с практической деятельностью человека — возведением укреплений и городских построек. Первые сведения о чертежах, напоминающих современные, связаны с именем Леонардо да Винчи (1452—1519) — итальянского ученого и художника, который в технических рисунках и эскизах раскрывал свои идеи в области техники и строительства.

8.1. Понятие алгоритма

Вспомним некоторые понятия, изученные в 6-м классе.

Алгоритм — понятная и конечная последовательность точных действий (команд), формальное выполнение которых позволяет получить решение поставленной задачи.

Исполнитель алгоритма — человек, группа людей или техническое устройство, которые понимают команды алгоритма и умеют правильно их выполнять.

Система команд исполнителя — команды, которые понимает и может выполнить исполнитель.

Любой исполнитель имеет ограниченную систему команд. Все они разделяются на группы:

1. Команды, которые непосредственно выполняет исполнитель.
2. Команды, меняющие порядок выполнения других команд исполнителя.

Компьютер является универсальным исполнителем.

Запись алгоритма в виде последовательности команд, которую может выполнить компьютер, называют **программой**.

Существуют следующие способы представления алгоритмов:

- **словесный** (описание алгоритма средствами естественного языка с точной и конкретной формулировкой фраз);
- **графический (блок-схема)** (графическое изображение команд алгоритма с использованием геометрических фигур, или блоков, и стрелок, соединяющих эти блоки и указывающих на порядок выполнения команд);
- **программный** (запись алгоритма в виде программы).

(Схематически данные способы представлены в примере 8.1.)

8.2. Исполнитель Чертежник

В 6-м классе вы познакомились с исполнителем Чертежник, предназначенным для построения рисунков и чертежей на координатной плоскости (пример 8.2).

Чертежник имеет перо, которым он может рисовать отрезки на плоскости. В исходном положении перо поднято и находится над точкой (0, 0) — началом координат. После завершения рисования перо также должно быть поднято.

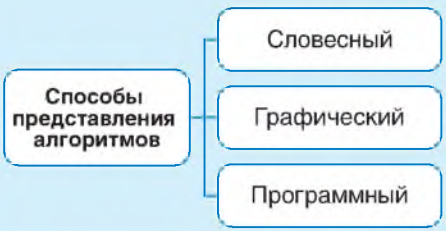
В настоящее время чертежи широко применяются в различных отраслях строительства, сельского хозяйства, промышленности и т. д. Сегодня для построения чертежей используются специальные программы, позволяющие автоматизировать процесс черчения. Вот логотипы подобных программ:

AutoCAD

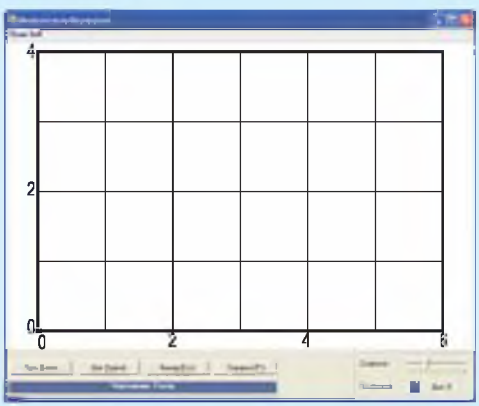

Компас-3D


NanoCAD

Пример 8.1.



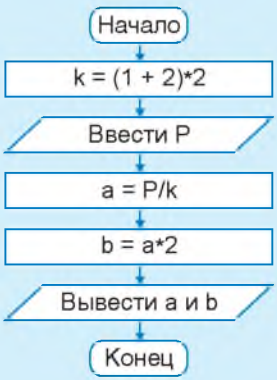
Пример 8.2. Поле исполнителя Чертежник.



Пример 8.3. Запись алгоритма по действиям:

1) 1 + 2 = 3 (части);
2) 3 · 2 = 6 (частей);
3) 120 : 6 = 20 (м);
4) 20 · 2 = 40 (м).

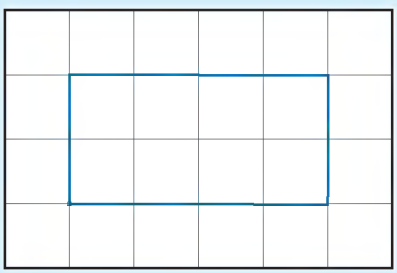
Блок-схема алгоритма:



Программа для исполнителя:

```
uses Drawman;
begin
  Field(6, 4);
  ToPoint(1, 1);
  PenDown;
  OnVector(4, 0);
  OnVector(0, 2);
  OnVector(-4, 0);
  OnVector(0, -2);
  PenUp;
end.
```

Результат работы программы:



Система команд Чертежника:

Команда	Действие
`ToPoint (x,y)`	Переместить перо в точку `(x,y)`
`PenUp`	Поднять перо
`PenDown`	Опустить перо
`Field (n,m)`	Создать поле размером $n \times m$
`OnVector (a,b)`	Сместить перо на a единиц по горизонтали и b единиц по вертикали

Пример 8.3. Составим алгоритм решения задачи.

Прямоугольный участок, длина которого в 2 раза больше ширины, огородили забором длиной 120 м. Определите длину и ширину участка. Напишите программу, выполнив которую исполнитель Чертежник построит чертеж забора этого участка. Масштаб: 1 клетка равна 10 м.

Словесное описание алгоритма:

1. Длина участка в 2 раза больше ширины, поэтому в сумме длина и ширина составят 3 одинаковые части. Забор огораживает участок по периметру, равному удвоенной сумме длины и ширины, т. е. периметр равен 6 одинаковым частям.

2. Ширина: 120 : 6 = 20 м.

3. Длина в 2 раза больше ширины: 20 · 2 = 40 м.

8.3. Алгоритмическая конструкция *следование*

Существует большое количество алгоритмов, в которых все команды выполняются последовательно одна за другой в том порядке, в котором они записаны. В подобных алгоритмах отсутствуют команды, меняющие порядок выполнения других команд. Такие программы вы составляли в прошлом году для исполнителя Чертежник.

Алгоритмическая конструкция *следование* — последовательность команд алгоритма, которые выполняются в том порядке, в котором они записаны.

Алгоритмическая конструкция *следование* отображает естественный, последовательный порядок выполнения действий в алгоритме.

Следование использовалось в примере 8.3, в котором описывались алгоритмы вычисления длины и ширины участка и построения прямоугольника исполнителем Чертежник.

Алгоритмическая конструкция *следование* представлена в примерах 8.4 и 8.5.

Пример 8.4. Алгоритм изготовления бутерброда:

1. Отрезать ломтик батона.
2. Положить на батон лист салата.
3. Отрезать кусочек ветчины.
4. Положить ветчину на лист салата.
5. Отрезать кусочек помидора.
6. Положить помидор на ветчину.

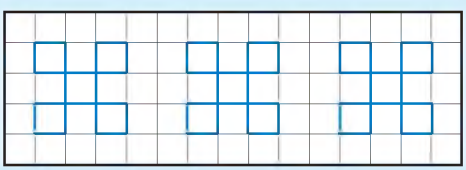
Пример 8.5. Алгоритм выполнения лабораторной работы по биологии «Строение инфузории туфельки»:

1. Рассмотреть внешний вид и внутреннее строение инфузории туфельки.
2. Зарисовать инфузорию туфельку и обозначить названия ее органов.
3. Подвести итог работе.

Пример 8.6. Программа для исполнителя Чертежник будет следующей:

```
uses Drawman;
procedure figura;
begin
  PenDown;
  OnVector(1,  0);
  OnVector(0,  3);
  OnVector(-1,  0);
  OnVector(0,  -1);
  OnVector(3,  0);
  OnVector(0,  1);
  OnVector(-1,  0);
  OnVector(0,  -3);
  OnVector(1,  0);
  OnVector(0,  1);
  OnVector(-3,  0);
  OnVector(0,  -1);
  PenUp;
end;
begin
  Field(15,  5);
  ToPoint(1,  1);
  Figura;
  ToPoint(6,  1);
  Figura;
  ToPoint(11,  1);
  Figura;
end.
```

Результат выполнения программы:



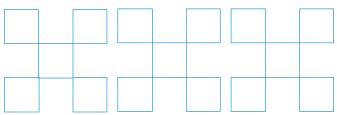
8.4. Вспомогательные алгоритмы

Часто в одной программе нужно рисовать одно и то же изображение несколько раз. Получение этого изображения удобно оформить в виде вспомогательного алгоритма, который можно использовать нужное число раз.

Вспомогательный алгоритм — алгоритм, целиком используемый в составе другого алгоритма.

Вспомогательный алгоритм решает некоторую подзадачу основной задачи. Вызов вспомогательного алгоритма в программе заменяет несколько команд одной.

Пример 8.6. Напишем программу, выполнив которую Чертежник нарисует изображение:



Данный рисунок состоит из одинаковых фигур. Для рисования одной из них можно оформить вспомогательный алгоритм `figura`.

Описание основного алгоритма:

перемещение в начальную точку;

рисование фигуры;

перемещение ко второй фигуре;

рисование фигуры;

перемещение к третьей фигуре;

рисование фигуры.

При решении задач над проектом могут работать несколько человек. Каждый из членов коллектива делает часть своей работы и оформляет ее как отдельный вспомогательный алгоритм.

Построение алгоритмов часто выполняют **методом пошаговой детализации**. При этом сложная задача разбивается на ряд более простых. Для каждой подзадачи составляется свой вспомогательный алгоритм. Подзадачи могут разбиваться на еще более простые подзадачи.

1. Что такое алгоритм?
2. Какие способы записи алгоритмов вам известны?
3. Что называют алгоритмической конструкцией *следование*?
4. Какой алгоритм называется вспомогательным?
5. Для чего нужны вспомогательные алгоритмы?

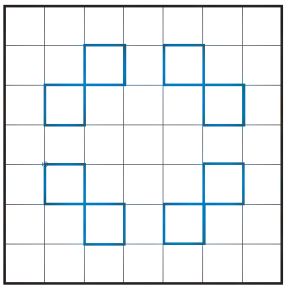
Упражнения

1 Какой рисунок получится после выполнения Чертежником следующей программы? Изобразите рисунок и проверьте правильность своих действий, выполнив программу на компьютере.

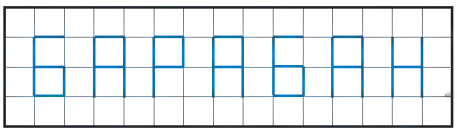
```
uses Drawman;
begin
  Field(8, 8);
  ToPoint(2, 1);
  PenDown;
  OnVector(4, 0);
  OnVector(0, 1);
  OnVector(1, 0);
  OnVector(0, 4);
  OnVector(-1, 0);
  OnVector(0, 1);
  OnVector(-4, 0);
  OnVector(0, -1);
  OnVector(-1, 0);
  OnVector(0, -4);
  OnVector(1, 0);
  OnVector(0, -1);
  PenUp;
end.
```

2 Напишите для исполнителя Чертежник программы получения следующих изображений:

а



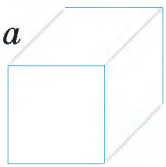
б



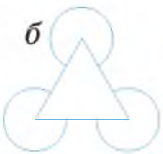
3 Придумайте свои рисунки и составьте программы для их рисования с помощью исполнителя Чертежник.

4* Проанализируйте рисунки. Какие из них мог выполнить исполнитель Чертежник? Почему? Какие команды вы можете предложить добавить исполнителю для выполнения остальных рисунков?

а

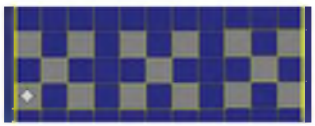


б



в

г



§ 9. Исполнитель Робот

Роботы развозят заказы в ресторане в г. Харбин (Китай)[1].

9.1. Роботы в жизни человека

Человек с глубокой древности мечтал об искусственном создании, которое могло бы выполнять его приказы. Сегодня эта мечта стала реальностью — в жизни людей появились роботы. Они способны выполнять практически любую работу, доступ-

[1]Материалы о роботах взяты с сайтов http://www.robogeek.ru и http://fishki.net/1211999-roboty-v-nashej-zhizni.html (дата доступа: 07.02.2017).

ную человеку, а также делать многие вещи, которые людям выполнить сложно или вообще невозможно. Роботы используются на производстве и в быту, могут работать в сфере услуг и развлечений. Есть роботы, похожие на человека, а есть совсем непохожие.

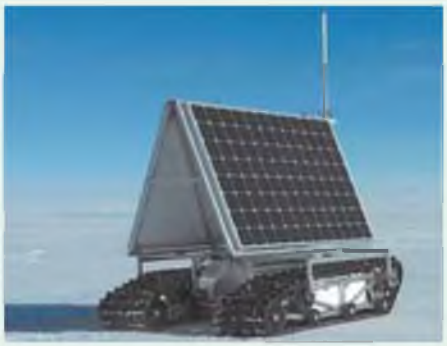
Робот — автоматическое устройство, которое действует по заранее составленной программе.

Робот получает информацию о внешнем мире от датчиков — аналогов органов чувств живых организмов — и предназначен для осуществления различных операций.

Мир роботов очень разнообразен. В быту современного человека используются автоматические стиральные и посудомоечные машины, роботы-пылесосы и др. С помощью роботов можно выращивать растения или управлять домом.

Робот может быть материальным или виртуальным. Виртуальный робот — специальная программа, выполняющая определенные действия.

Робот LS3, созданный для транспортировки грузов по пересеченной местности.



Автономный робот GROVER, который изучает слои льда на ледниковом щите Гренландии.

Роботизированная система, предназначенная для выращивания овощей. Управление данной системой осуществляется через Wi-Fi. Есть возможность удаленного контроля через Интернет.

Робот-пылесос с помощью системы камер и сенсоров может ориентироваться в комнате и строить маршрут уборки.

Роботы являются исполнителями. Для исполнителей обычно определяют среду обитания и систему команд.

Общим для всех роботов является то, что человек может ими управлять. Робот получает команды от оператора и выполняет их по одной или действует автономно по предварительно составленной программе.

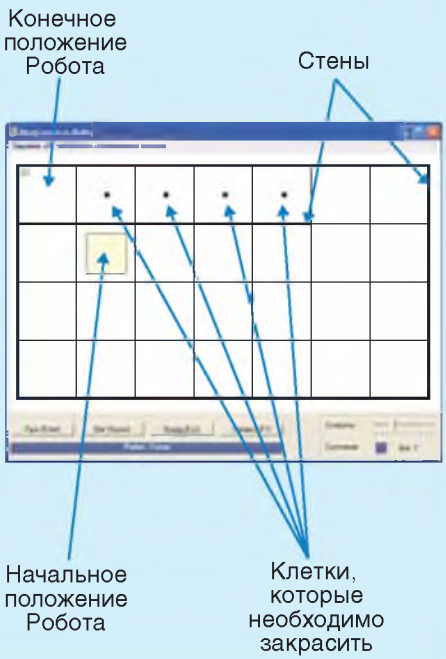
9.2. Среда обитания и система команд исполнителя Робот

В среде программирования PascalABC, кроме исполнителя Чертежник, можно выбрать исполнителя Робот.

Средой обитания исполнителя Робот является прямоугольное клетчатое поле. Размеры поля, как и для исполнителя Чертежник, задаются командой `Field(n, m)`. Первоначально Робот находится в центральной клетке поля.

Между некоторыми клетками, а также по периметру поля находятся стены. Робот может передвигаться по полю и закрашивать указанные клетки. Большой желтый квадрат внутри клетки означает начальное положение Робота, маленький — конечное (пример 9.1).

Пример 9.1. Поле исполнителя Робот с начальной обстановкой имеет следующий вид:



Поле Робота, на котором определено положение стен, начальное и конечное положение исполнителя, называют **обстановкой**.

Для подключения исполнителя Робот в программе прописывается команда **uses** Robot. Готовые задания с обстановками для Робота хранятся в задачнике, встроенном в систему программирования, и вызываются командой task. Эта же команда использовалась для Чертежника.

Система команд исполнителя:

Команда	Действие
Right	Перемещает Робота вправо
Left	Перемещает Робота влево
Up	Перемещает Робота вверх
Down	Перемещает Робота вниз
Paint	Закрашивает текущую ячейку

Робот может становиться на обычную и на закрашенную клетку, но не может переместиться с клетки на клетку, если между ними стена. Робот не может переместиться за границы поля. Эти действия вызывают ошибку (пример 9.2). Робот может закрасить уже закрашенную клетку. Такое действие ошибку не вызывает.

Пример 9.2. Вызов задачи a1 из встроенного задачника:

```
•Program1.pas
uses Robot;
begin
  Task('a1');
end.
```

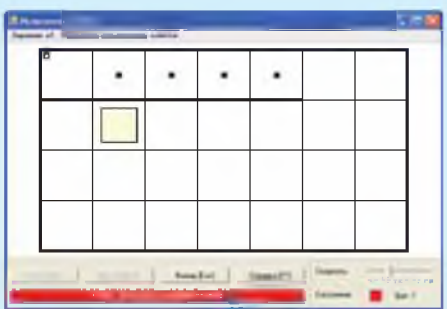
Запись команд исполнителя:

```
•Program1.pas*
uses Robot;
begin
  Task('a1');
  right;
  right;
end.
```

Запишем в программе команду up.

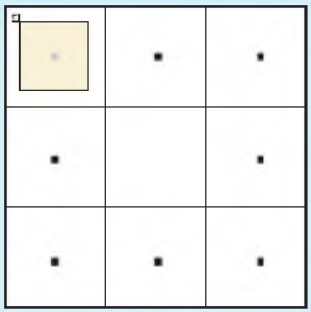
```
uses Robot;
begin
  Task('a1');
  up;
end.
```

Сверху находится стена, поэтому перемещение Робота вверх невозможно:



Сообщение об ошибке

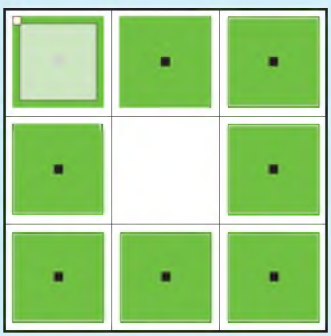
Пример 9.3. Начальная обстановка:



Программа для исполнителя Робот:

```
uses Robot;
begin
  Task('a2');
  paint; right;
  paint; right;
  paint; down;
  paint; down;
  paint; left;
  paint; left;
  paint; up;
  paint; up;
end.
```

Результат работы программы имеет следующий вид:



9.3. Использование алгоритмической конструкции *следование* для исполнителя Робот

Рассмотрим примеры решения задач для исполнителя Робот.

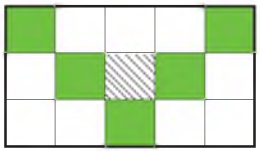
Пример 9.3. Робот находится на поле размером 3 × 3 клетки. Нужно закрасить все клетки, кроме средней (задача а2).

Для решения задачи Робот должен выполнить следующий алгоритм:

```
закрасить;
вправо;
закрасить;
вправо;
закрасить;
вниз;
закрасить;
вниз;
закрасить;
влево;
закрасить;
влево;
закрасить;
вверх;
закрасить;
вверх.
```

В данном алгоритме Робот обходит клетки, двигаясь по часовой стрелке. Тот же результат можно получить, если Робот будет обходить поле против часовой стрелки, изначально двигаясь вниз.

Пример 9.4. Составим программу для закрашивания клеток поля Робота по образцу:



Такой обстановки нет в задачнике, поэтому вначале нужно создать поле Робота размером 5 × 3. Начальное положение Робота на таком поле отмечено заштрихованной клеткой.

Для решения задачи Робот должен выполнить алгоритм:

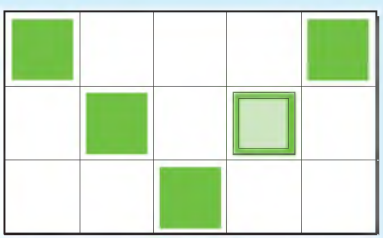
```
создать поле;
вниз;
закрасить;
влево;
вверх;
закрасить;
влево;
вверх;
закрасить;
вправо;
вправо;
вправо;
вправо;
закрасить;
влево;
вниз;
закрасить.
```

**Какими еще способами можно решить данную задачу?*

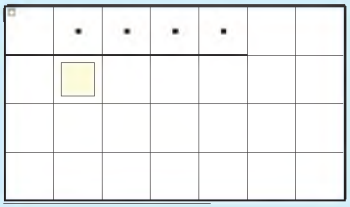
Пример 9.4. В данном случае программа для учебного компьютерного исполнителя Робот может быть составлена таким образом:

```
uses Robot;
begin
  Field(5, 3);
  down;
  paint;
  left;
  up;
  paint;
  left;
  up;
  paint;
  right;
  right;
  right;
  right;
  paint;
  left;
  down;
  paint;
end.
```

Результат работы записанной выше программы будет иметь следующий вид:



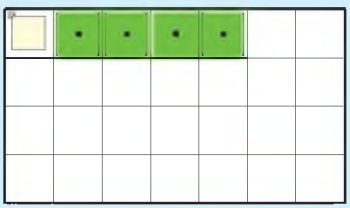
Пример 9.5. Начальная обстановка:



Программа для Робота:

```
uses Robot;
begin
  Task('a1');
  right;  right;
  right;  right;
  up;
  left;  paint;
  left;  paint;
  left;  paint;
  left;  paint;
  left;
end.
```

Результат работы программы:



Пример 9.5. Решим задачу а1 из встроенного задачника.

Для решения данной задачи Робот должен обойти линию (стену), закрасить указанные клетки и переместиться в клетку, определяющую конечное положение исполнителя.

Алгоритм решения задачи:

сдвинуться вправо на 4 клетки;

вверх;

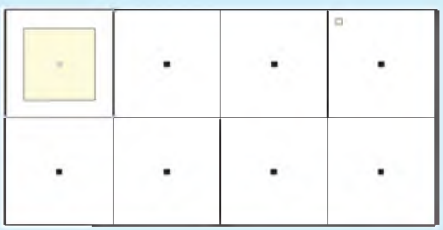
сдвинуться влево на 4 клетки, закрашивая их по пути;

влево.

В примерах 9.3 — 9.5 команды исполнителя Робот выполнялись последовательно, одна за другой, в том порядке, в котором они были записаны. Поэтому все приведенные алгоритмы реализованы с использованием алгоритмической конструкции *следование*.

9.4. Вспомогательные алгоритмы

Пример 9.6. Начальная обстановка:



Пример 9.6. Решим задачу а3 из встроенного задачника.

Робот должен закрасить все клетки поля. Но двигаться по прямой ему мешают стены, которые исполнитель должен обходить.

Алгоритм решения задачи:

закрасить; вниз;

закрасить; вправо;

```
закрасить;  вверх;
закрасить;
вправо;
закрасить;  вниз;
закрасить;  вправо;
закрасить;  вверх;
закрасить.
```

Если проанализировать данный алгоритм, то можно заметить, что дважды повторяется последовательность команд, которая закрашивает квадрат из четырех клеток:

```
закрасить;  вниз;
закрасить;  вправо;
закрасить;  вверх;
закрасить.
```

Оформим эти команды как вспомогательный алгоритм, который назовем **квадрат**. Тогда алгоритм решения исходной задачи может быть записан так:

```
квадрат;
вправо;
квадрат.
```

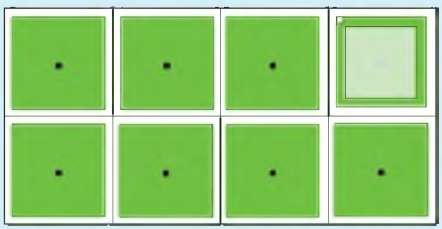
При решении данной задачи использование вспомогательного алгоритма позволило не записывать дважды одну и ту же последовательность команд.

Вспомогательные алгоритмы можно использовать и в том случае, когда исходная задача разбивается на несколько независимых друг от друга задач. Тогда

Программа 1 для исполнителя Робот:

```
uses Robot;
begin
  Task('a3');
  paint; down;
  paint; right;
  paint; up;
  paint;
  right;
  paint; down;
  paint; right;
  paint; up;
  paint;
end.
```

Результат работы программы:



Программа 2 (с использованием вспомогательного алгоритма) для исполнителя Робот:

```
uses Robot;
procedure kvadrat;
begin
  paint; down;
  paint; right;
  paint; up;
  paint;
end;
begin
  Task('a3');
  kvadrat;
  right;
  kvadrat;
end.
```

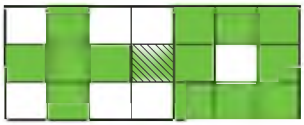
Пример 9.7. Программа для учебного компьютерного исполнителя Робот:

```
uses Robot;
procedure krest;
begin
  left; paint;
  down; left; paint;
  up; left; paint;
  right; paint;
  up; paint;
end;
procedure kvadrat;
begin
  paint; right;
  paint; right;
  paint; down;
  paint; down;
  paint; left;
  paint; left;
  paint; up;
  paint; up;
end;
begin
  field(7,3);
  krest;
  right; right; right;
  kvadrat;
end.
```

Результат работы записанной выше программы будет иметь следующий вид:

каждую из них можно оформить как вспомогательный алгоритм.

Пример 9.7. Напишем программу для закрашивания клеток поля Робота по образцу:



Такой обстановки нет в задачнике, поэтому создадим поле 7×3. Начальное положение Робота отмечено заштрихованной клеткой.

В данной задаче Робот должен нарисовать две отдельные фигуры: крест и квадрат. Составим два вспомогательных алгоритма.

Вспомогательный алгоритм **крест**:

```
влево; закрасить;
вниз; влево; закрасить;
вверх; влево; закрасить;
вправо; закрасить;
вверх; закрасить.
```

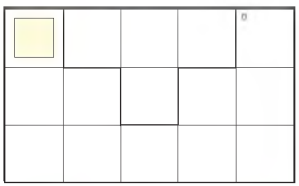
В качестве вспомогательного алгоритма для рисования квадрата можно использовать алгоритм решения задачи а2 (пример 9.3). Для перехода от одной фигуры к другой Робот должен сдвинуться на 3 клетки вправо:

```
крест;
вправо; вправо; вправо;
квадрат.
```

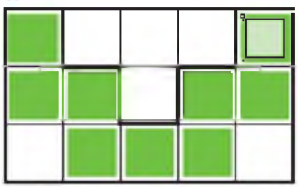
1. Что такое робот?
2. Какие команды входят в систему команд учебного компьютерного исполнителя Робот?
3. Опишите среду обитания учебного исполнителя Робот.

Упражнения

1 Начальная обстановка на поле Робота изображена на рисунке справа. Трое учащихся составили и выполнили алгоритм, по которому Робот закрасил все клетки пути от начальной к конечной. На каком из рисунков — *а*, *б* или *в* — изображено решение данной задачи? Почему?



а



б

в

2 Какой из приведенных алгоритмов решает задачу, сформулированную в предыдущем задании? Объясните, почему другие программы не могут быть алгоритмами решения данной задачи.

а)
```
paint;  down;
right;
paint;  down;
right;
paint;  right;
up;
paint;  right;
up;
paint;
```

б)
```
paint;  down;
paint;  right;
paint;  down;
paint;  right;
paint;  right;
paint;  up;
paint;  right;
paint;  up;
paint;
```

в)
```
ToPoint(0,3);
PenDown;
OnVector(1,0);
OnVector(0,-1);
OnVector(1,0);
OnVector(0,-1);
OnVector(1,0);
OnVector(0,1);
OnVector(1,0);
OnVector(0,1);
OnVector(1,0);
```

3 Для какого исполнителя приведен алгоритм в задании 2, *в*? Сформулируйте для этого исполнителя задачу, решением которой будет приведенный алгоритм.

4 Для исполнителя Робот была составлена следующая программа:

```
paint;
right;  up;
paint;
right;  down;
```

Изобразите в тетради «узор», который нарисует Робот. При каких минимальных размерах поля Робот сможет выполнить данную программу?

5 Все команды в программе из задания 4 учащийся скопировал три раза. Как изменится «узор» после выполнения программы? Как можно по-другому записать этот алгоритм? Какого размера поле нужно создать? Подсказка. Воспользуйтесь вспомогательным алгоритмом.

6 Программа решения задачи была записана на доске. Два учащихся, переписывая этот алгоритм для исполнителя Робот, пропустили из-за невнимательности по одной команде. Какую команду пропустил каждый из учащихся? Что будет результатом работы каждой программы?

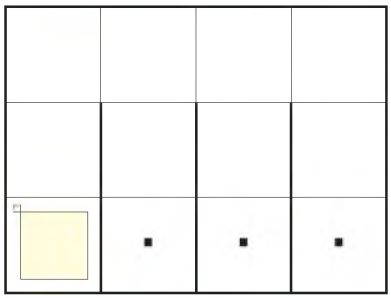
Программа, записанная первым учащимся	Программа, записанная вторым учащимся

`uses Robot;`
`begin`
`Field(15,15);`
`paint; right;`
`paint; right;`
`paint; down;`
`paint; down;`
`paint;`
`down; left;`
`down; left;`
`paint; down;`
`paint; down;`
`paint; right;`
`paint; right;`
`paint; up;`
`paint; up;`
`paint; up;`
`left; up;`
`left;`
`paint; up;`
`paint; up;`
`end.`
	`uses Robot;`
	`begin`
	`Field(15,15);`
	`paint; right;`
	`paint; right;`
	`paint; down;`
	`paint; down;`
	`paint;`
	`down; left;`
	`paint;`
	`down; left;`
	`paint; down;`
	`paint; down;`
	`paint; right;`
	`paint; right;`
	`paint; up;`
	`paint; up;`
	`paint; up;`
	`left; up;`
	`paint; up;`
	`paint; up;`
	`end.`

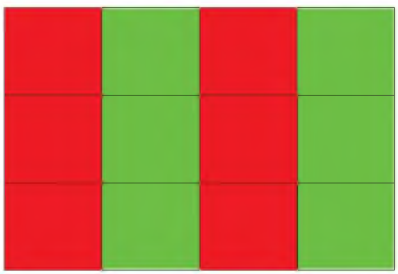
7 Составьте программу для решения задачи а4 из встроенного задачника (см. рис. справа). Предложите два алгоритма:

1. С использованием алгоритмической конструкции *следование*.
2. С использованием вспомогательного алгоритма.

Сравните полученные решения.



8* Робот-огородник может разбить грядку на посадочные зоны-клетки. На рисунке справа изображена схема посадки овощей (красная клетка — томаты, зеленая — огурцы). Предложите систему команд для робота-огородника и разработайте алгоритм посадки овощей (робот сажает одно растение в одну клетку).



§ 10. Алгоритмическая конструкция *повторение*

10.1. Алгоритмы с циклами

В окружающем мире можно наблюдать ситуации, при которых различные действия и процессы повторяются. Некоторые повторяются несколько раз и завершаются. Другие могут повторяться очень долго (например, круговорот воды в природе, движение планет в космическом пространстве, смена времен года и т. д.). Человеку тоже регулярно приходится выполнять повторяющиеся действия: умываться, одеваться, посещать парикмахерскую, завтракать, ходить на работу и др.

Понятие цикла используется в различных сферах человеческой деятельности.

Под циклом понимают совокупность явлений, процессов, составляющих кругооборот в течение определенного промежутка времени. С этой точки зрения можно говорить о годовом цикле вращения Земли вокруг Солнца или о производственном цикле.

Циклом является законченный ряд каких-либо произведений, чего-либо излагаемого, исполняемого: цикл лекций, цикл стихотворений.

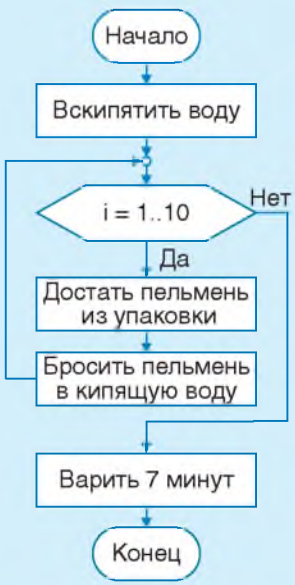
Пример 10.1. Приготовление пельменей.

Алгоритм:

1. Вскипятить воду.
2. Для i = 1..10 повторять:
   2.1. Достать пельмень из упаковки.
   2.2. Бросить пельмень в кипящую воду.
3. Варить 7 минут.

В данном примере параметр цикла i изменяется от 1 до 10. Действия «достать пельмень из упаковки» и «бросить пельмень в кипящую воду» выполняются 10 раз и составляют тело цикла.

Блок-схема данного алгоритма выглядит таким образом:



Как правило, человек составляет программы, в которых каждая команда в отдельности и весь алгоритм в целом выполняются за конечное число повторений.

Алгоритмическая конструкция *повторение* (цикл) определяет последовательность действий, выполняемых многократно. Эту последовательность действий называют **телом цикла**.

Существует несколько возможностей управлять тем, сколько раз будет повторяться тело цикла.

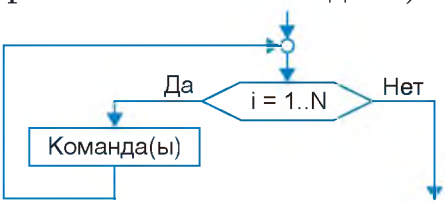
Алгоритмическая конструкция *цикл с параметром (цикл со счетчиком)* — способ организации цикла, при котором количество повторов зависит от начального и конечного значений параметра цикла.

Таким образом, цикл с параметром организует выполнение команд тела цикла заранее известное число раз (пример 10.1).

Параметр цикла определяет нумерацию действий в цикле. Параметр цикла может принимать только целые значения. Часто нумерацию начинают с 1 и закан-

чивают числом N (пример 10.2). В этом случае цикл выполнится N раз. Если нумерация установлена двумя произвольными числами $N1$ (начальное значение) и $N2$ (конечное значение), то цикл выполнится ($N2 - N1 + 1$) раз.

Алгоритмическая конструкция цикла с параметром может изображаться на блок-схеме следующим образом (значение параметра изменяется от 1 до N):



В данной конструкции в прямоугольнике(-ах) записываются повторяющиеся команды алгоритма (тело цикла), которые выполняются N раз (Да). При этом после каждого выполнения команд тела цикла происходит проверка, который раз выполняется цикл. На блок-схеме переход на проверку условия изображается в виде стрелки, выходящей из тела цикла и возвращающейся к проверке. Как только команды тела цикла выполнятся N раз (Нет), цикл завершится (пример 10.3). Если $N \leqslant 0$, то команда тела цикла не выполнится ни разу.

Пример 10.2. Вычислим a^n (например, $3^5 = 243$).

Алгоритм возведения числа в степень:

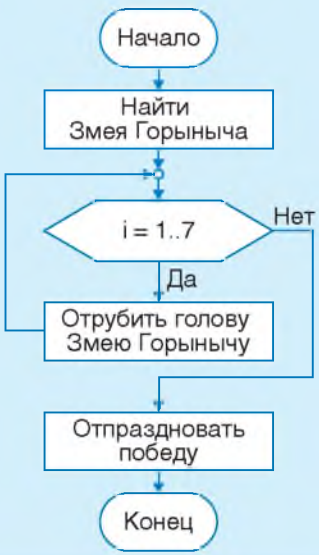
1. Ввести значения a и n.
2. Определить начальное значение результата $r = 1$.
3. Для $i = 1..n$ повторять:
   3.1. Умножить результат на a.
4. Записать результат.

Пример 10.3. В фольклорных произведениях часто встречается многоголовый Змей Горыныч (количество голов может быть, например, 7).

Алгоритм победы над Змеем Горынычем:

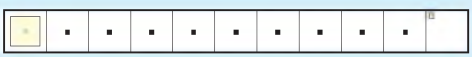
1. Найти Змея Горыныча.
2. Для $i = 1..7$ повторять:
   2.1. Отрубить голову Змею Горынычу.
3. Отпраздновать победу.

Блок-схема данного алгоритма:



Многие роботы, которые используются в быту и на производстве, могут выполнять циклические алгоритмы. Примером такого робота является суши-робот, который может производить от 450 до 4000 заготовок для суши за 1 час.

Пример 10.4. Начальная обстановка:



Программа для исполнителя Робот:

```
uses Robot;
begin
  Task('c2');
  for var i:= 1 to 10 do
  begin
    paint;
    right;
  end;
end.
```

Результат работы программы:

10.2. Использование команды цикла с параметром для исполнителя Робот

Чтобы составлять алгоритмы с циклами для компьютерного исполнителя Робот, нужно знать, как записывается команда цикла.

Для записи цикла с параметром используется команда **for**. Формат записи команды:

```
for var i:= N1 to N2 do[1]
begin
  тело цикла;
end;
```

Строка **for var** i:= N1 **to** N2 **do** является заголовком цикла. Его читают так: «Для переменной i от $N1$ до $N2$ делай». Если $N2 \geqslant N1$, то команды тела цикла выполнятся $(N2 - N1 + 1)$ раз, иначе цикл не выполнится ни разу.

Слова **begin** и **end;** являются операторными скобками в языке Pascal. Если тело цикла состоит из одной команды, операторные скобки можно опустить.

Операторные скобки — пара слов, определяющих в языке программирования блок команд, воспринимаемый как единое целое, как одна команда.

Пример 10.4. Решим задачу c2 из встроенного задачника.

[1] Команда в таком формате записывается только в среде PascalABC.Net.

Робот должен закрасить все клетки поля (кроме последней), перемещаясь вправо. Для этого в цикле нужно 10 раз выполнить команды:

```
закрасить;
вправо.
```

Данные команды образуют тело цикла.

Командами, образующими тело цикла, могут быть любые команды из системы команд исполнителя. Кроме того, в теле цикла может вызываться вспомогательный алгоритм. Использование вспомогательного алгоритма позволит сократить запись тела цикла и сделает программу более понятной.

Пример 10.5. Решим задачу с7 из встроенного задачника.

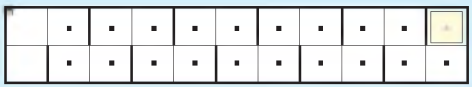
На поле исполнителя Робот есть стены. При обходе стен Робот выполняет следующие команды:

```
закрасить;  вниз;
закрасить;  влево;
закрасить;  вверх;
закрасить;  влево.
```

Чтобы решить задачу, Робот должен повторить эти команды 5 раз. Оформим данные команды как вспомогательный алгоритм `kvadrat` и вызовем его в цикле.

В данном примере тело цикла состоит из одной команды `kvadrat`, поэтому операторные скобки можно не использовать.

Пример 10.5. Начальная обстановка для учебного компьютерного исполнителя Робот:



Программа для исполнителя Робот составляется следующим образом:

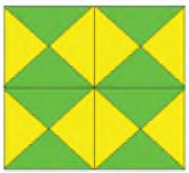
```
uses Robot;
procedure kvadrat;
begin
  paint;
  down;
  paint;
  left;
  paint;
  up;
  paint;
  left;
end;
begin
  Task('c7');
  for var i:= 1 to 5 do
    kvadrat;
end.
```

Результат работы записанной выше программы будет иметь следующий вид:

1. Что понимают под алгоритмической конструкцией *повторение*?
2. Что такое цикл с параметром?
3. Что такое операторные скобки?
4. Приведите примеры использования цикла.

Упражнения

1 Опишите словесно или изобразите с помощью блок-схемы следующие алгоритмы:

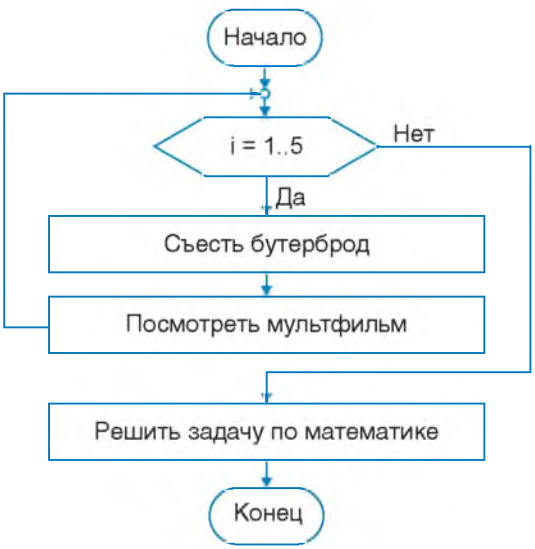


1. Рисование в графическом редакторе изображения из 4 квадратов с диагоналями и закрашенными областями (см. рис. справа).
2. Каждую минуту бактерия делится на две. Изначально есть одна бактерия. За бактериями наблюдали 10 минут. Определите количество бактерий в конце наблюдения. Заполните таблицу.

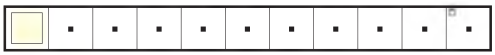
Время, мин	0	1	2	3	4	5	6	7	8	9	10
Количество бактерий	1	2									

3. Сверление 10 отверстий.
4. Сервировка стола к обеду на 6 персон.

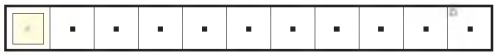
2 Семиклассник Андрей после школы пригласил своего друга Юру помочь ему в решении 5 задач по математике. В гостях Юра посоветовал Андрею провести остаток дня, воспользовавшись алгоритмом, записанным в виде блок-схемы (см. рис. справа). Объясните, почему за выполнения этого задания Андрей получил двойку по математике.



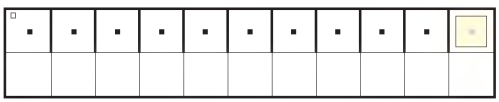
3 Составьте программу для решения задачи с3 из встроенного задачника. Сравните алгоритм решения этой задачи с примером 10.4. Что у них общего? Чем они отличаются?



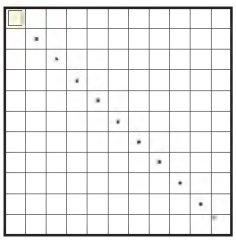
4 Составьте программу для решения задачи с4 из встроенного задачника. Сравните ее решение с предыдущим упражнением и с примером 10.4.



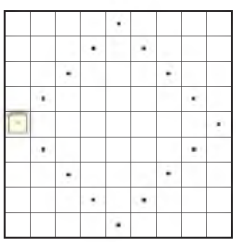
5 Составьте программу для решения задачи с8 из встроенного задачника. Используйте вспомогательный алгоритм.



6 Составьте программу для решения задачи с5 из встроенного задачника.



7 Для решения задачи с14 Петя составил алгоритм и записал программу. Петин младший брат Олег удалил несколько команд. Определите, сколько команд удалил Олег. Восстановите программу, которую написал Петя.



```
uses Robot;
begin
 Task('c14');
 paint;
 for var i:= 1 to 4 do
 begin
  paint;
  right;
  down;
 end;
 for var i:= 1 to 4 do
 begin
  right;
  up;
 end;
 for var i:= 1 to 4 do
 begin
  paint;
 end;
end.
```

8 Максим пытается представить, как можно было бы использовать роботов в различных ситуациях, описанных в литературных произведениях. Например, для Тома Сойера, которого тетушка Полли отправила красить забор, Максим придумал робота-маляра и решил, что такому роботу достаточно одной команды: покрась доску. Алгоритм покраски забора из 20 досок Максим записал так:

1. Установить робота у левого края забора.
2. Для i = 1..20 повторять:
   2.1. Покрась доску.

Сможет ли робот-маляр покрасить забор? В чем ошибка Максима? Исправьте алгоритм, добавив необходимую(-ые) команду(-ы).

§ 11. Использование условий

Условия используются в правилах дорожного движения. Так, если горит зеленый свет, то можно переходить улицу.

Условия также встречаются в фольклоре, например при выборе пути сказочными героями.

В. Васнецов. «Витязь на распутье» (фрагмент). 1882 г.

11.1. Понятие условия

Принятие решений зачастую зависит от различных условий. Если на улице дождь, то нужно взять зонт; если хорошо подготовился к уроку, то получишь высокую отметку, иначе низкую и др.

Человек способен понимать условия, сформулированные в произвольной форме. Но для того чтобы Робот или другой исполнитель мог принимать решения, нужно «научить» его «понимать» условия.

Условием для исполнителя является понятное ему высказывание, которое может быть истинным (соблюдаться) либо ложным (не соблюдаться).

Исполнитель может проверить истинность условий, входящих в его систему условий.

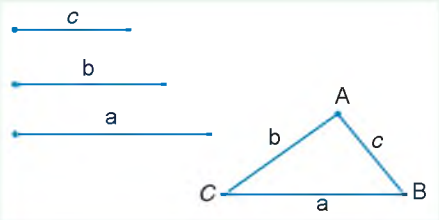
Рассмотрим систему условий для исполнителя Робот.

`WallFromLeft`	Истинно, если слева от Робота стена
`WallFromRight`	Истинно, если справа от Робота стена
`WallFromUp`	Истинно, если сверху от Робота стена
`WallFromDown`	Истинно, если снизу от Робота стена
`FreeFromLeft`	Истинно, если слева от Робота свободно
`FreeFromRight`	Истинно, если справа от Робота свободно
`FreeFromUp`	Истинно, если сверху от Робота свободно
`FreeFromDown`	Истинно, если снизу от Робота свободно
`CellIsPainted`	Истинно, если ячейка, в которой находится Робот, закрашена
`CellIsFree`	Истинно, если ячейка, в которой находится Робот, не закрашена

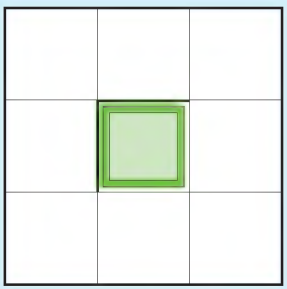
Образцы истинных и ложных условий для исполнителя Робот представлены в примере 11.1.

Применяются условия в математике, например:

Треугольник существует, если для большей стороны a выполняется неравенство $a < b + c$.



Пример 11.1. Рассмотрим начальную обстановку поля исполнителя Робот:



В данном случае для Робота будут истинны следующие условия:

`WallFromLeft`
`WallFromUp`
`FreeFromRight`
`FreeFromDown`
`CellIsPainted`

Ложными при такой начальной обстановке будут условия:

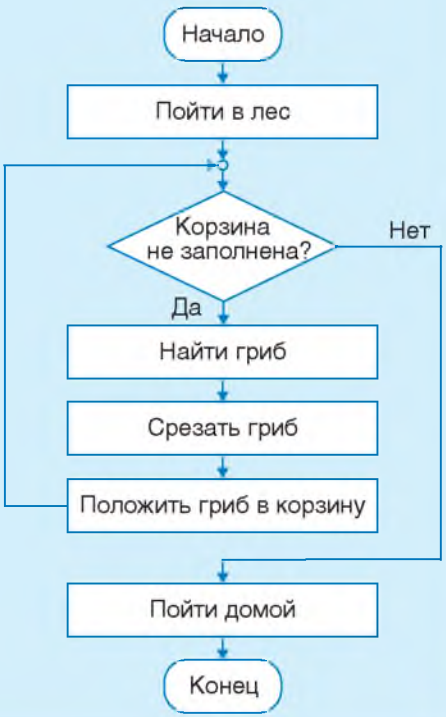
`WallFromRight`
`WallFromDown`
`FreeFromLeft`
`FreeFromUp`
`CellIsFree`

Пример 11.2. Сбор грибов.

Использование цикла с параметром при составлении алгоритма решения этой задачи может привести к разным результатам.

Корзина может быть или полупустой, или не все найденные грибы в нее поместятся.

Если использовать цикл с предусловием, то в результате домой можно унести полную корзину грибов.



11.2. Цикл с предусловием

Цикл с параметром используется при составлении алгоритма в том случае, когда заранее известно количество повторений. Однако часто до выполнения цикла количество повторений не известно.

Пример 11.2. Вы с родителями пошли в лес за грибами. Ваши действия можно описать командами: найти гриб, срезать гриб, положить гриб в корзину. Эти действия будут выполняться в цикле, но вы заранее не знаете, сколько грибов войдет в корзину. Поэтому следует говорить не о количестве повторений (количестве грибов), а об условии, при котором вы будете продолжать сбор грибов: пока корзина не заполнена.

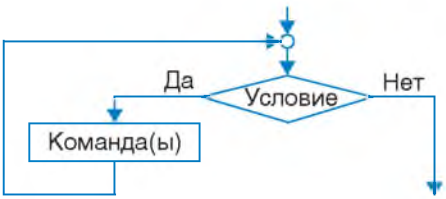
Алгоритмическая конструкция *цикл с предусловием (цикл «пока»)* — способ организации цикла, при котором количество выполнений команд тела цикла зависит от истинности или ложности условия цикла.

Цикл с предусловием используется, когда количество повторений тела цикла заранее не известно, но известно условие продолжения работы.

Условие цикла определяет, как долго будет выполняться цикл.

Пока условие истинно, выполняются команды, составляющие тело цикла. Цикл прекращает выполняться тогда, когда условие становится ложным. Цикл с предусловием имеет такое название, поскольку проверка условия предваряет выполнение команд тела цикла.

Алгоритмическая конструкция цикла с предусловием изображается на блок-схеме так:



В данной конструкции в прямоугольнике(-ах) записываются повторяющиеся команды алгоритма (тело цикла), которые совершаются, пока верно условие (Да). При этом после каждого выполнения команд тела цикла происходит проверка, истинно ли условие. Как только условие станет ложным (Нет), цикл завершается. Если условие сразу ложно, то цикл не выполнится ни разу.

Если условие в цикле будет всегда истинно (всегда Да), то такой цикл не сможет завершиться. Возникшую ситуацию называют **зацикливанием**.

Российский академик Андрей Андреевич Марков (младший) (1903—1979) в своих исследованиях в области теории алгоритмов показал, что в общем случае алгоритмы должны содержать предписания двух видов:

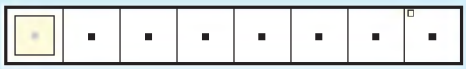
1) функциональные операторы, направленные непосредственно на преобразование информации;

2) логические операторы, определяющие дальнейшее направление действий.

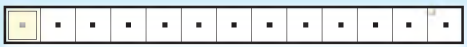
Оператор — элемент языка, задающий полное описание действия, которое необходимо выполнить. В английском языке данное понятие обозначается словом *statement*, означающим также ‘предложение’.

Если применить вышесказанное к компьютерным исполнителям, то предписания первого вида составляют систему команд исполнителя, а предписания второго вида — систему условий исполнителя.

Пример 11.3. Одна из возможных начальных обстановок:



Другая возможная начальная обстановка:



Запишем программу для учебного компьютерного исполнителя Робот:

```
uses Robot;
begin
 Task('w2');
 while FreeFromRight do
 begin
  paint;
  right;
 end;
 paint;
end.
```

Результат работы указанной выше программы для первой начальной обстановки будет иметь следующий вид:

Результат работы программы для второй начальной обстановки:

Для записи цикла с предусловием используется команда **while**. Формат записи команды:

```
while <условие> do
begin
 тело цикла;
end;
```

Строка **while** <условие> **do** является заголовком цикла. Эту строку можно прочитать следующим образом: «Пока верно условие, делай». Команды **begin** и **end**; в данном случае играют роль операторных скобок.

Пример 11.3. Напишем программу для решения задачи w2 из встроенного задачника.

Робот должен закрасить коридор переменной длины.

В данной задаче нам точно не известна длина коридора, но известно, что Робот может двигаться, пока справа пусто, и закрашивать клетки:

```
Пока справа пусто, повторять
 закрасить;
 вправо.
```

После прохода всего коридора Робот должен закрасить последнюю клетку. Это происходит после выполнения цикла, так как для последней клетки условие «справа пусто» уже не выполняется.

Пример 11.4. Напишем программу для решения следующей задачи. Робот находится в верхнем левом углу поля. Снизу от него вдоль всего поля расположена стена с проходом в одну клетку. Составить алгоритм, выполнив который Робот сможет пройти через проход и закрасить клетку. Расположение прохода заранее не известно.

Проход не ограничен стеной снизу. Робот может двигаться вправо, пока внизу есть стена:

Пока снизу стена, **повторять** вправо.

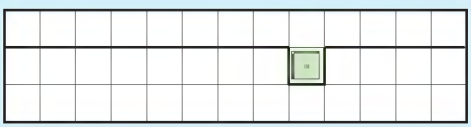
Робот остановится в той клетке, у которой снизу нет стены. После этого Робот должен сдвинуться вниз и закрасить клетку.

Пример 11.4. Одна из возможных начальных обстановок:

Программа для исполнителя Робот:

```
uses Robot, RobTasks[1];
begin
 Task('myrob3');
 while WallFromDown do
  right;
 down;
 paint;
end.
```

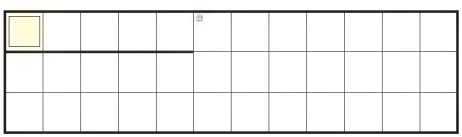
Результат работы программы будет следующим:

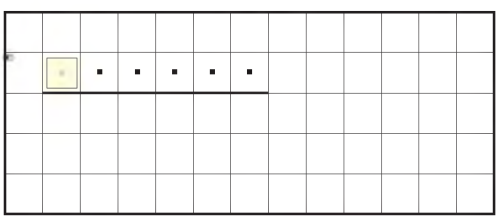


1. Что понимают под условием для исполнителя?
2. Когда используется цикл с предусловием?
3. В каком случае возникает ситуация зацикливания?

Упражнения

1 Напишите программу для решения задач w3 и w8 из встроенного задачника. Обращайте внимание на начальное и конечное положение Робота.





[1] Модуль RobTasks, содержащий данную обстановку и задачу, можно скачать по адресу: http://e-vedy.adu.by/course/view.php?id=423.

2 Для исполнителя Робот был написан следующий алгоритм:

```
uses Robot;
begin
 Field( , );
 while FreeFromRight do
 begin
  paint;
  down;
  right;
  paint;
  up;
  right;
 end;
end.
```

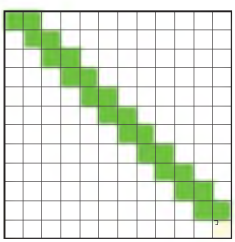
Нарисуйте в тетради результат работы данного алгоритма. Какими должны быть размеры поля, чтобы Робот не врезался в стену? Определите начальное положение Робота.

3 Составьте алгоритм, выполнив который Робот нарисует «узор» из задания 2 вдоль левого края поля исполнителя. Каким должен быть вертикальный размер поля исполнителя? (Задача myrob5 из модуля RobTasks.)

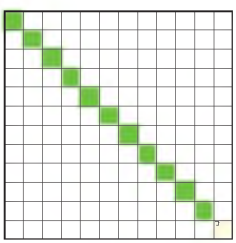
4 Робот находится на квадратном поле неизвестного размера. Начальное положение Робота — верхний левый угол. Составьте и выполните алгоритм, по которому Робот переместится из начального положения в нижний правый угол и закрасит все клетки своего пути. На каком (на каких) из рисунков изображено решение этой задачи? Почему?

а

б

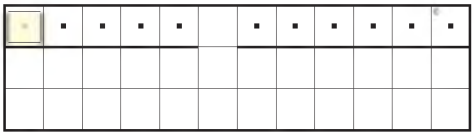


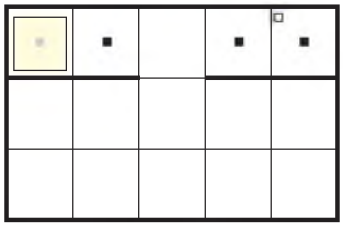
в



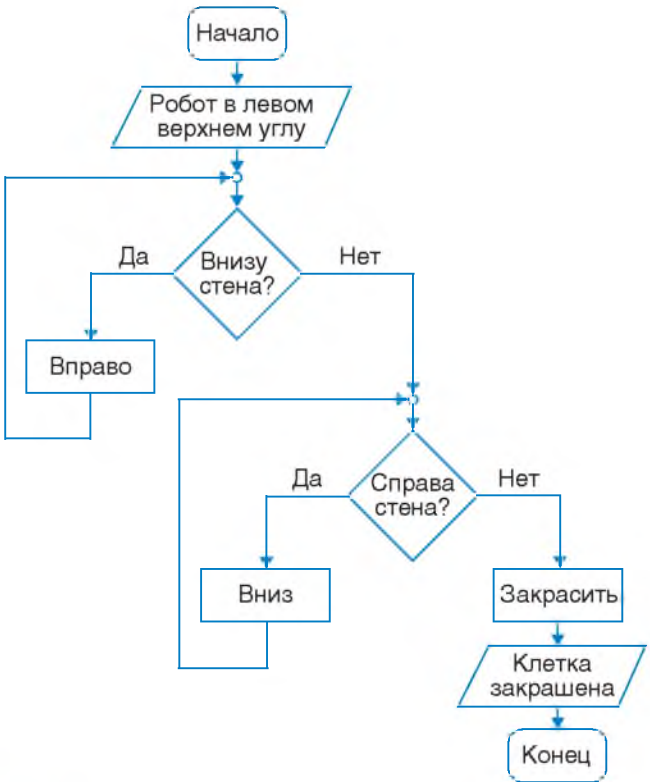
5 На поле Робота размещен «забор» — горизонтальная стена. Забор нужно «покрасить» — закрасить все клетки сверху стены. В «заборе» могут быть одни «ворота» — клетка без линий. Длина «забора» и

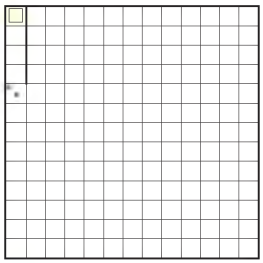
расположение «ворот» не известны. (Задача myrob7 из модуля RobTasks.)

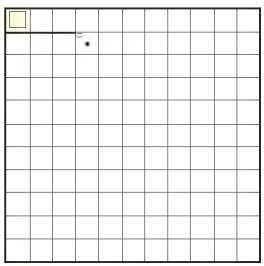


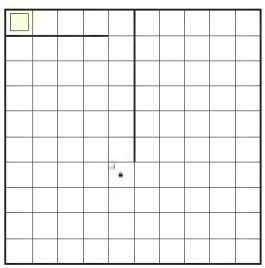


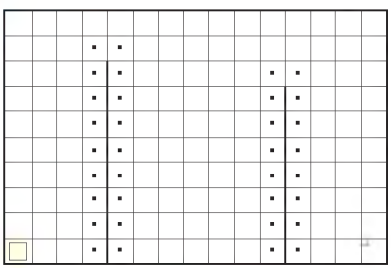
6 По блок-схеме запишите программу для исполнителя Робот. Каким будет результат для каждой из предложенных начальных обстановок? (Задача myrob8 из модуля RobTasks.)









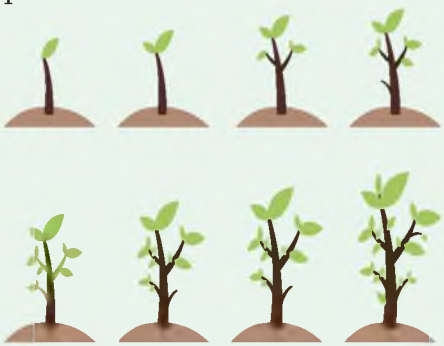


7* Решите задачу w10 из встроенного задачника. Напишите вспомогательный алгоритм для обхода одной стены.

§ 12. Алгоритмическая конструкция *ветвление*

Понятие ветвления используется в различных сферах человеческой деятельности.

В ботанике под ветвлением побегов понимают процесс образования боковых побегов у растений.



При употреблении термина в переносном смысле под ветвлением понимают наличие нескольких путей, направлений, сюжетных линий и т. д.

Ветвления используются в дорожной разметке и картографии.

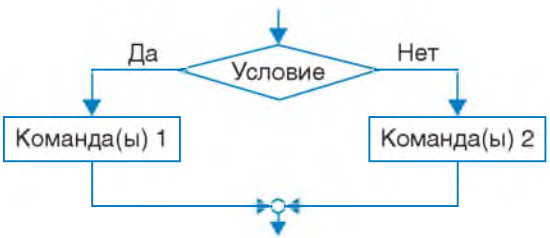
12.1. Команда ветвления

Довольно часто на поставленный вопрос человек получает ответ «Да» или «Нет». В зависимости от ответа он определяет свои действия и выполняет одну или другую команду (группу команд).

Роботы и другие технические устройства тоже могут выполнять различные действия в зависимости от условия. Если условие истинно (на вопрос получен ответ «Да»), то выполняются одни действия, если ложно, то другие.

Алгоритмическая конструкция *ветвление* обеспечивает выполнение одной или другой последовательности команд в зависимости от истинности или ложности некоторого условия.

Ветвление может изображаться на блок-схеме таким образом:



В данной конструкции в прямоугольнике(-ах) записываются команды алгоритма. При

такой организации алгоритма может выполниться **только** одна из двух команд (последовательностей команд). Другая последовательность будет проигнорирована (пример 12.1).

Для записи конструкции ветвления в языке программирования Pascal используется команда **if**. Формат записи команды:

```
if <условие> then
begin
 команды 1;
end
else
begin
 команды 2;
end;
```

Строка **if** <условие> **then** является заголовком ветвления. Эту строку можно прочитать так: «Если условие верно, то». После слова **then** записывается последовательность команд 1, которая выполнится, если условие истинно. После слова **else** записывается последовательность команд 2, которая выполнится, если условие ложно. Слова **begin** и **end**; в данном случае играют роль операторных скобок. Обратите внимание, что перед словом **else** точка с запятой не ставится.

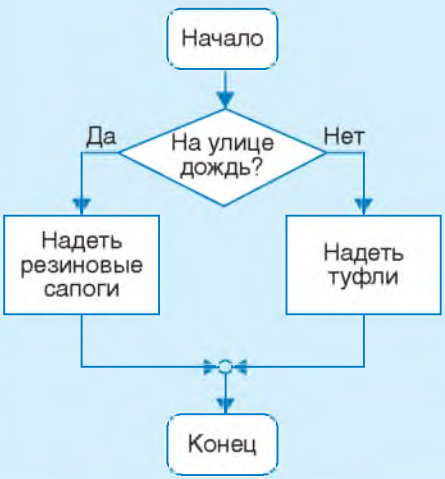
Ветвление может быть записано в **полной** или **сокращенной** форме.

Пример 12.1. Выбор обуви весной в зависимости от погоды:

```
Если на улице дождь, то
  надеть резиновые сапоги;
Иначе
  надеть туфли.
```

В данном примере в текущий момент времени может быть выполнена только одна команда из двух: или надеть сапоги, или надеть туфли.

Блок-схема данного алгоритма будет выглядеть следующим образом:

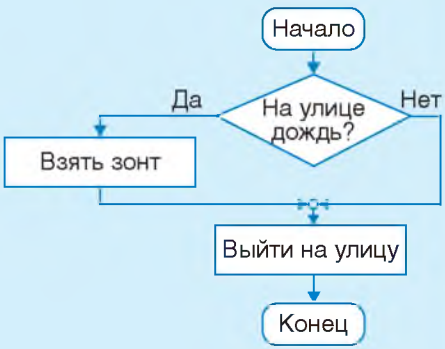


Пример 12.2. Выход на улицу осенью.

```
Если на улице дождь, то
  взять зонт;
выйти на улицу.
```

Здесь используется сокращенная форма команды ветвления. Если условие истинно, выполняется команда «взять зонт». Если условие ложно, никаких действий не происходит. Команда «выйти на улицу» выполняется всегда.

Блок-схема алгоритма:

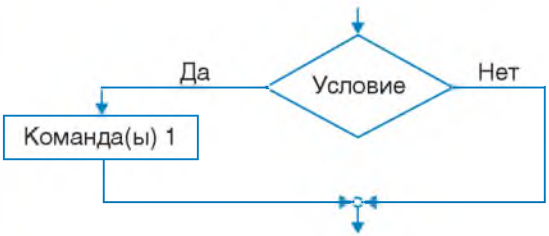


Пример 12.3. Имеется три монеты, среди которых одна фальшивая. Фальшивая монета легче настоящих. Найдем фальшивую монету за минимальное число взвешиваний на чашечных весах без гирь:

```
Положить на каждую чашу
весов монеты 1 и 2;
Если весы в равновесии, то
  фальшивая монета 3;
Иначе
  Если монета 1 тяжелее, то
    фальшивая монета 2;
  Иначе
    фальшивая монета 1.
```

Полная форма ветвления предусматривает организацию выполнения двух разных наборов команд, из которых выполняется только один. В сокращенной форме один из наборов команд (чаще по ответу «Нет») опускается. В этом случае, если условие ложное, то никакие действия не выполняются.

Блок-схема сокращенной формы ветвления:



(Рассмотрите пример 12.2.)

На языке программирования Pascal команда запишется так:

```
if <условие> then
begin
 команды 1;
end;
```

Алгоритм может содержать более одной конструкции ветвления (пример 12.3).

Пример 12.4. Решим задачу if1 из встроенного задачника.

Робот должен закрасить клетку, которая находится за стеной. В зависимости от обстановки обход стены может осуществляться по-разному.

Вначале Робот должен сдвинуться вправо. Если стена снизу, то сверху свободно и можно обойти стену сверху, в противном случае Робот обходит стену снизу.

После обхода стены Робот закрашивает клетку. Алгоритм можно записать так:

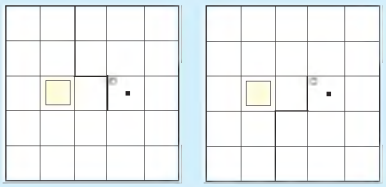
```
вправо;
Если сверху свободно, то
  вверх; вправо; вниз;
Иначе
  вниз; вправо; вверх;
закрасить.
```

Пример 12.5. Робот находится на неизвестной клетке поля без линий. Он должен закрасить клетку слева от себя.

Для того чтобы закрасить клетку слева от себя, Робот должен переместиться влево, а затем закрасить клетку. Однако сделать это Робот сможет только тогда, когда не находится в клетках, являющихся левой границей поля. Поэтому, прежде чем сдвинуться влево, Робот должен проверить, свободно ли слева.

Результат работы данной программы зависит от начального положения Робота. Поэтому для проверки правильности работы программы необходимо подготовить начальные обстановки, которые дают разные ответы на вопрос: слева пусто?

Пример 12.4. Возможные начальные обстановки:



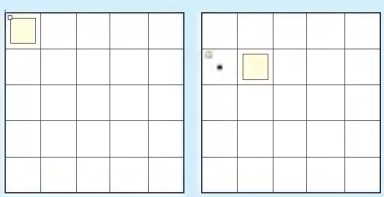
Программа для Робота:

```
uses Robot;
begin
 Task('if1');
 right;
 if FreeFromUp then
 begin
  up; right; down;
 end
 else
 begin
  down; right; up;
 end;
 paint;
end.
```

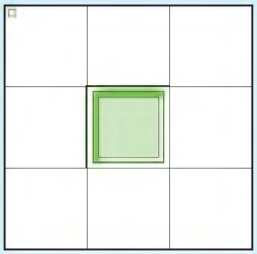
Пример 12.5. Программа для исполнителя Робот:

```
uses Robot, RobTasks;
begin
 Task('myrob9');
 if FreeFromLeft then
 begin
  left; paint;
 end;
end.
```

Возможные начальные обстановки:



Пример 12.6. Рассмотрим следующую начальную обстановку поля для исполнителя Робот:



Проверим для Робота следующие составные условия:

1. `WallFromLeft and CellIsPainted`.
2. `WallFromUp or WallFromDown`.
3. `Not (WallFromRight or FreeFromUp)`.

Первое условие состоит из двух простых: `WallFromLeft` (условие *A*) и `CellIsPainted` (условие *B*). Условие может быть записано как «*A* **И** *B*». Это условие верно только тогда, когда верны и *A*, и *B*. Условие *A* — `WallFromLeft` — истинно, условие *B* — `CellIsPainted` — истинно, условие *A* **И** *B* — истинно.

Второе условие может быть записано как «*A* **ИЛИ** *B*», где *A* — `WallFromUp`, *B* — `WallFromDown`. Условие *A* — истинно, условие *B* — ложно. Значит, условие «*A* **ИЛИ** *B*» — истинно.

12.2. Составные условия

В качестве условия в алгоритмах с циклами и ветвлениями используется любое понятное исполнителю этого алгоритма высказывание, которое может быть либо истинным, либо ложным.

Все условия, с которыми нам приходилось до сих пор встречаться при составлении алгоритмов для Робота, были простыми высказываниями. Однако можно строить и составные условия.

Составное условие — условие, которое образуется из нескольких простых условий, соединенных друг с другом логическими операциями.

С логическими операциями над высказываниями вы уже знакомы. В PascalABC используются следующие логические операции:

Логическая операция	Запись в PascalABC
Не	`Not`
И	`And`
Или	`Or`

(Рассмотрите пример 12.6.)

Система условий для Робота построена так, что можно обойтись без использования логической операции отрицания.

Для условия FreeFromLeft отрицанием будет условие not

FreeFromLeft. Но условие «слева не свободно» означает, что там стена. Поэтому вместо условия not FreeFromLeft можно писать WallFromLeft. Отрицания других условий показаны в таблице:

Условие	Отрицание
WallFromLeft	FreeFromLeft
WallFromRight	FreeFromRight
WallFromUp	FreeFromUp
WallFromDown	FreeFromDown
CellIsPainted	CellIsFree

В третьем условии частица Not отрицает составное условие WallFromRight or FreeFromUp. Условие может быть записано как **НЕ** («*A* **ИЛИ** *B*»). Для того чтобы определить, истинно или ложно это условие, нужно сначала определить истинность условия «*A* **ИЛИ** *B*». Условие *A* — ложно, условие *B* тоже ложно. Поэтому ложным будет и условие «*A* **ИЛИ** *B*», но тогда условие **НЕ** «*A* **ИЛИ** *B*» будет истинным.

1. Что такое алгоритмическая конструкция *ветвление*?
2. Чем отличается полная конструкция ветвления от сокращенной?
3. Что такое составное условие?
4. Какие логические операции можно использовать для записи составных условий?
5. Какими способами можно построить отрицание условия для компьютерного исполнителя Робот?

Упражнения

1 Выделите конструкцию ветвления в отрывке из поэмы А. С. Пушкина «Руслан и Людмила» и изобразите ее с помощью блок-схемы.

У лукоморья дуб зеленый;
Златая цепь на дубе том:
И днем и ночью кот ученый
Все ходит по цепи кругом;
Идет направо — песнь заводит,
Налево — сказку говорит.
Там чудеса: там леший бродит,
Русалка на ветвях сидит...[1]

[1] Пушкин, А. С. Руслан и Людмила : поэма. — М. : Изд. Дом «Прибой». — 1996. — С. 5.

2 Для заданной обстановки поля Робота определите, какие из составных условий истинны, а какие ложны.

Начальная обстановка	Условия

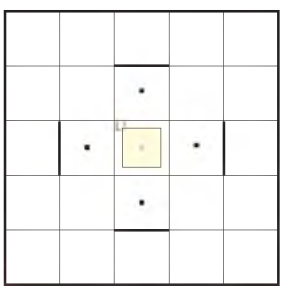
	`WallFromLeft or CellIsPainted;`
	`WallFromUp and WallFromDown;`
	`Not CellIsPainted and FreeFromRight;`
	`Not (WallFromUp or FreeFromRight);`
	`WallFromDown and CellIsFree;`
	`(WallFromUp or WallFromDown) and FreeFromRight.`

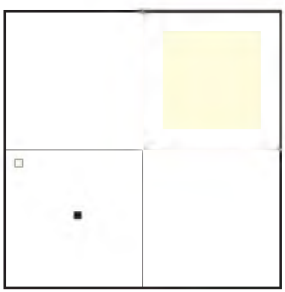
3* В задании 2 замените условия, содержащие not, соответствующими условиями без использования отрицания.

4 Для каждого из ложных условий задания 2 придумайте обстановку, в которой данное условие будет верным, а для каждого истинного — обстановку, в которой условие будет ложным.

5 Измените программу из примера 12.5 так, чтобы Робот закрашивал клетку справа (снизу, сверху) от себя. Нарисуйте в тетради различные начальные обстановки для проверки данного условия.

6 Решите задачи if2 и if3 из встроенного задачника.





§ 13. Использование основных алгоритмических конструкций для исполнителя Робот

Последовательное выполнение команд в программе определяется структурой *следование*. Для организации повторяющихся действий в алгоритме используется команда **цикла**. Команда **ветвления** позволяет выполнять одну или другую последовательность команд в зависимости от истинности условия.

Следование, цикл и ветвление — **базовые алгоритмические конструкции**. Используя эти конструкции как элементы некоего «конструктора», можно составлять и разрабатывать любые алгоритмы.

Команды цикла и ветвления управляют порядком выполнения других команд в программе и относятся к командам управления. Использование алгоритмической конструкции *следование* предполагает отсутствие управляющих конструкций.

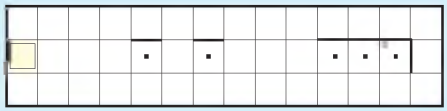
Рассмотрим подробнее примеры алгоритмов, содержащих несколько алгоритмических конструкций.

Перед человеком постоянно возникают разнообразные задачи, для которых существуют различные алгоритмы решения. При всем многообразии алгоритмов для их записи достаточно трех алгоритмических конструкций (структур): следование, цикл, ветвление.

Это положение было выдвинуто в середине 70-х гг. XX в. нидерландским ученым Эдсгером Вибе Дейкстрой (1930—2002).

Его труды оказали влияние на развитие информатики и информационных технологий. Э. Дейкстра является одним из разработчиков концепции структурного программирования, участвовал в создании языка программирования Алгол. Известен своими достижениями в области математической логики и теории графов.

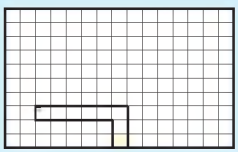
Пример 13.1. Одна из возможных начальных обстановок:

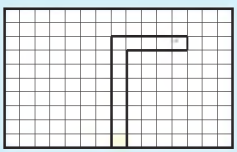


Программа для Робота:

```
uses Robot;
begin
 Task('cif1');
 while FreeFromRight do
 begin
  if WallFromUp then
   paint;
  right;
 end;
 if WallFromUp then
   paint;
end.
```

Пример 13.2. Возможные начальные обстановки:





Программа для Робота:

```
uses Robot;
begin
 Task('cif17');
 while FreeFromUp do
  up;
 if FreeFromLeft then
  while FreeFromLeft do
   left
 else
  while FreeFromRight do
   right;
end.
```

Пример 13.1. Решим задачу cif1 из встроенного задачника.

Робот передвигается вправо до тех пор, пока не встретит стену. По пути он должен закрасить клетки, над которыми есть стена.

Для решения задачи Робот должен проверять каждую клетку на своем пути. Если условие «сверху стена» выполняется, Робот закрашивает эту клетку. После проверки клетки Робот сдвигается вправо. Такие действия выполняются в цикле, пока справа пусто.

После цикла нужна команда ветвления, так как для крайней клетки поля команда «справа пусто» не выполняется и клетка в цикле не закрашивается. В этой задаче внутри структуры цикла используется структура ветвления.

Пример 13.2. Решим задачу cif17 из встроенного задачника.

Робот должен дойти до конца «коридора» переменного размера. «Коридор» может сворачивать влево или вправо.

Для решения задачи Робот сначала перемещается вверх до тех пор, пока вверху пусто. Стена, появившаяся сверху, означает, что начался поворот «коридора». «Коридор» поворачивает влево, если слева пусто, иначе «коридор» поворачивает вправо. Дальше Ро-

бот двигается в том направлении, где пусто, пока не встретит стену.

В данной задаче используется сначала структура цикла, а затем структура ветвления. Каждая последовательность команд в структуре ветвления, в свою очередь, является циклом.

Операторные скобки опущены, поскольку последовательность состоит из одной команды цикла.

Пример 13.3*. Решим задачу cc5 из встроенного задачника.

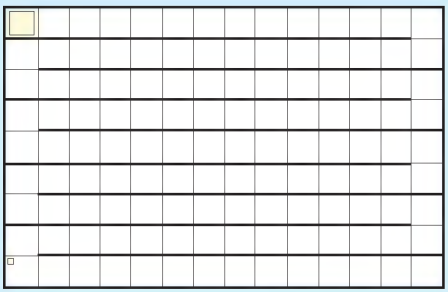
Робот находится в верхнем левом углу поля и должен переместиться в нижний левый угол. На поле присутствуют стены, которые Робот должен обойти. При этом он должен сначала двигаться до правой границы поля, затем спуститься вниз, а потом двигаться до левой границы поля и спуститься вниз. Эти действия Робот должен повторить 4 раза.

В данной задаче внутри цикла с параметром используются два других цикла с предусловием.

Структуру, когда внутри одного цикла выполняется другой, называют **вложенными** циклами.

Таким образом, базовые алгоритмические структуры можно комбинировать друг с другом.

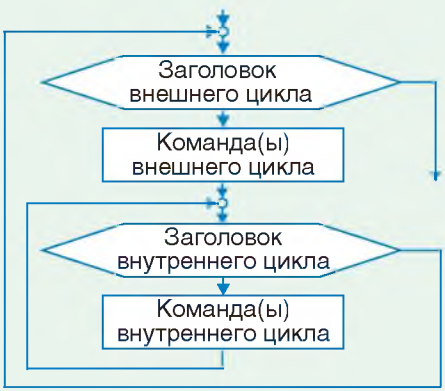
Пример 13.3*. Начальная обстановка:



Программа для Робота:

```
uses Robot;
begin
 Task('cc5');
 for var i:= 1 to 4 do
 begin
  while FreeFromRight do
   right;
  down;
  while FreeFromLeft do
   left;
  down;
 end;
end.
```

Блок-схема вложенных циклов:



1. Назовите базовые алгоритмические конструкции.

2. Приведите примеры использования базовых алгоритмических конструкций.

3*. Что такое вложенный цикл?

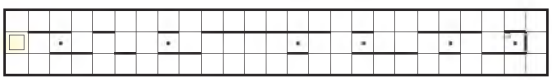
Упражнения

1 Какие алгоритмические конструкции используются в приведенных программах? Нарисуйте блок-схемы данных алгоритмов. Предложите пример начальной обстановки, в которой алгоритм выполнится корректно.

а)
```
uses Robot;
begin
 while WallFromLeft do
 begin
  down;
  paint;
 end;
end.
```

б)
```
uses Robot;
begin
 while CellIsPainted do
  if FreeFromleft then
   left;
end.
```

2 Для решения задачи cif3 из встроенного задачника Миша написал программу, но она работает неправильно. Какие ошибки допустил Миша?



```
uses Robot;
begin
 Task('cif3');
 while WallFromRight do
 begin
  if WallFromDown or
  WallFromUp then
   paint;
  right;
 end;
 if WallFromUp and
 WallFromDown then
  paint;
end.
```

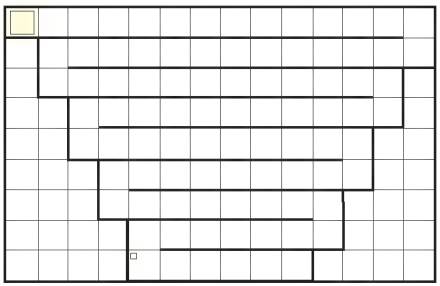
3 Используя базовые алгоритмические конструкции, запишите алгоритмы, соответствующие описаниям. Постройте для них блок-схемы.

1. Тело цикла, выполняющегося при условии `WallFromUp`, состоит из двух команд: `right` и `paint`.

2. Если условие `FreeFromRight` не выполняется, то, если клетка не закрашена, ее нужно закрасить, а если закрашена, то сдвинуться влево.

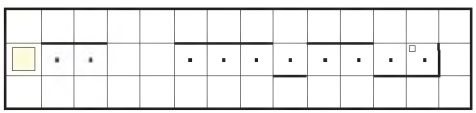
3. Проверку условия `CellIsPainted` нужно производить до тех пор, пока снизу нет стен. При выполнении условия сдвинуться вниз, при невыполнении условия закрасить клетку.

4 Заполните пропуски в программе решения задачи cc14 из встроенного задачника так, чтобы она работала верно.

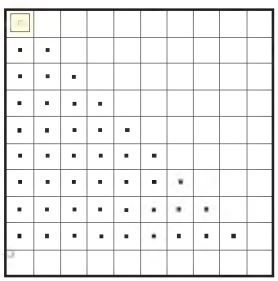


```
uses Robot;
begin
 Task('cc14');
 for var i: =1 to 4 do
 begin
  while … do
   right;
  down;
  while … do
   left;
  down;
 end;
end.
```

5 Решите задачу cif2 из встроенного задачника, используя внутри цикла команду ветвления.



6 Решите задачу cc7 из встроенного задачника, использовав внутри одного цикла два вложенных цикла.



7* Придумайте задачу для исполнителя Робот, в которой будут использоваться различные алгоритмические конструкции.

§ 14. Язык программирования Паскаль

Компьютер (от англ. *computer* — вычислитель) — устройство или система, способные выполнять заданную четко определенную изменяемую последовательность операций (чаще всего численных расчетов).

Электронно-вычислительная машина (ЭВМ) — комплекс технических средств, где основные функциональные элементы (логические, запоминающие, индикационные и др.) выполнены на электронных приборах, предназначенных для автоматической обработки информации в процессе решения вычислительных задач.

Никлаус Вирт (родился в 1934 г.) — швейцарский ученый, специалист по информатике, один из известнейших теоретиков в области разработки языков программирования, профессор компьютерных наук. Создатель и ведущий проектировщик языков программирования Паскаль, Модула-2, Оберон.

Желание упростить и ускорить всевозможные расчеты присуще человеку с древних времен. Сегодня компьютер способен выполнять сотни миллионов операций в секунду. Для решения вычислительных задач требуется сначала составить алгоритм их решения, а затем записать его в виде программы, используя какой-либо язык программирования.

Язык программирования устанавливает набор правил, определяющих внешний вид программы и действия, которые выполнит исполнитель под ее управлением.

Язык программирования Паскаль (Pascal) используется для обучения программированию и является базой для ряда профессиональных языков программирования.

Существует множество сред программирования, поддерживающих язык Паскаль: PascalABC, FreePascal, Delphi, GNU Pascal, Dev-Pascal, Rad Studio и др. В учебном курсе используется среда PascalABC (с ней вы работали, знакомясь с учебными компьютерными исполнителями).

14.1. Команда вывода

Демонстрировать работу любой программы имеет смысл только тогда, когда она выводит какую-либо информацию.

Программа на языке Pascal (тело программы) должна начинаться со слова **begin**, а заканчиваться словом **end** и точкой. Программа, состоящая из этих команд, разделенных пробелом или переводом строки, может быть запущена на выполнение, но она ничего не делает. Добавим в нее команду вывода приветствия:

```
begin
 write('Привет!');
end.
```

Результат работы программы отражается в нижней части окна программы PascalABC в окне вывода (пример 14.1).

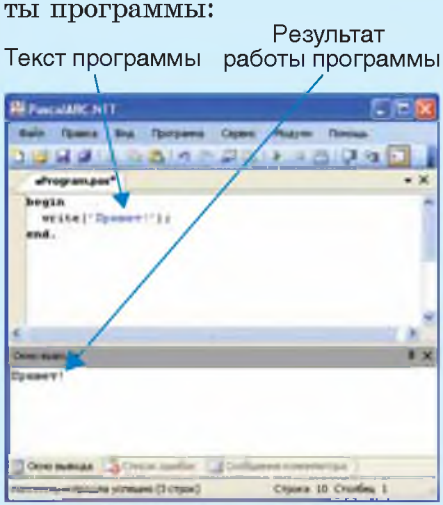
Команда `write( );` предназначена для **вывода данных.**

Текст, который нужно вывести на экран, заключают в апострофы (одинарные кавычки). Этот текст не анализируется и выводится в том виде, в котором он записан. Текст можно записать на любом языке. Текстом может быть произвольный набор символов.

В одной программе может быть несколько команд вывода. Для

По традиции, начавшейся в 1978 г. с примера из книги Б. Кернигана и Д. Ритчи «Язык программирования Си», первая программа на любом языке программирования должна выводить на экран приветствие миру:

Пример 14.1. Окно среды PascalABC с результатом работы программы:



Пример 14.2. Текст программы:

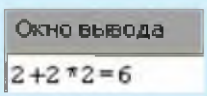
```
begin
 write('Привет!  ');
 writeln('Я компьютер!!!');
 write('Я умею выполнять ');
 writeln('программы!');
 write('Сегодня ты ');
 write('написал свою ');
 write('первую программу,');
 writeln(' а я ее выполнил.');
 write('Сейчас на экране -');
 writeln(' результат этой
программы.');
end.
```

Результат работы программы:

Пример 14.3. Текст программы:

```
begin
 write('2+2*2=');
 write(2+2*2);
end.
```

Результат работы программы:



Две команды write в программе можно объединить в одну, отделив текст от выражения запятой:

```
begin
 write('2+2*2=',  2+2*2);
end.
```

вывода текста, записанного в несколько строк, используют команду writeln( ). Сочетание «ln» (сокр. от англ. *line* — линия, строка), записанное в конце команды, означает, что после вывода нужно перевести курсор в новую строку.

Пример 14.2. Выведем на экран компьютера следующий текст: «Привет! Я компьютер!!! Я умею выполнять программы! Сегодня ты написал свою первую программу, а я ее выполнил. Сейчас на экране – результат этой программы».

Используя сочетание команд write и writeln, текст можно расположить по-разному.

Как вы уже знаете, текст в команде write( ), записанный в кавычках, не анализируется. Если кавычки опустить, то производится анализ данных, записанных в скобках. Так, если в скобках написать арифметическое выражение, то сначала вычисляется его значение, а затем выводится результат.

Пример 14.3. Посчитаем значение выражения 2 + 2 * 2.

Если записать выражение в кавычках, то будет выведено само выражение. При отсутствии кавычек на экран будет выведено значение данного выражения.

14.2. Понятие типа данных

На практике редко приходится писать программы, которые решают только одну задачу. Обычно программы пишутся для решения целого класса задач, которые можно сформулировать в общем виде.

С такими задачами вы уже сталкивались в курсе математики. Например, решение задачи «Найдите площадь прямоугольника» можно записать так: $S = a \cdot b$, где переменные a и b обозначают соответственно длину и ширину прямоугольника, а S — площадь. Зная эту формулу, можно найти площадь любого прямоугольника.

В программировании для решения задач в общем виде также используют переменные. Поскольку с такими переменными будет работать компьютер, то они должны храниться в его памяти.

Информацию, представленную в пригодном для обработки на компьютере виде, называют **данными**.

Переменная в программировании — это именованная ячейка памяти, хранящая значение переменной.

До начала 1950-х гг. XX в. программисты ЭВМ при создании программ пользовались машинным кодом. Запись программы на машинном коде состояла из единиц и нулей. Машинный код принято считать языком программирования первого поколения. Типы данных не использовались.

Первым языком программирования, в котором появилась возможность создавать переменные, считается Ассемблер. В этом языке вместо машинных кодов стали использовать команды, записанные текстом. Ассемблер относится к языкам программирования второго поколения.

В 1957 г. появился язык Фортан, открывший эру языков программирования третьего поколения. Он позволил использовать разные числовые типы данных, необходимые для сложных расчетов: целые, вещественные (действительные) и комплексные.

Дальнейшее развитие языков программирования позволило добавить возможность работы с другими типами данных. Современные языки программирования позволяют работать с большим количеством типов данных.

В среде программирования PascalABC реализовано более 30 различных типов данных.

Обзор типов

Типы в PascalABC.NET подразделяются на простые, структурированные, типы указателей, процедурные типы, последовательности и классы.

К *простым* относятся целые и вещественные типы, логический, символьный, перечислимый и диапазонный тип.

Тип данных называется *структурированным*, если в одной переменной этого типа может содержаться множество значений.

К структурированным типам относятся массивы, строки, записи, кортежи, множества, файлы и классы.

Особым типом данных является последовательность, которая хранит по-существу алгоритм получения данных последовательности один за другим.

Все простые типы, кроме вещественного, называются *порядковыми*. Только значения этих типов могут быть индексами статических массивов и параметрами цикла for. Кроме того, для порядковых типов используются функции Ord, Pred и Succ, а также процедуры Inc и Dec.

Пример 14.4. Примеры описания переменных:

```
var x: real;
var x1, y1: real;
var a_1, a_2, a_3: real;
```

Диапазон возможных значений типа real задается числами в стандартном представлении от $-1.8 \cdot 10^{308}$ до $1.8 \cdot 10^{308}$. Наименьшее положительное число типа real приблизительно равно $5.0 \cdot 10^{-324}$. При вычислениях в числе хранится до 16 цифр.

Компьютер может обрабатывать данные разных типов: целые и действительные числа, символы, тексты и др.

Тип данных определяет способ хранения данных в памяти компьютера, диапазон возможных значений данных и операции, которые с этим типом данных можно выполнять.

Чтобы использовать какую-либо переменную, ее нужно описать. Описание переменных выполняется до начала программы (команды **begin**) (пример 14.4). При описании переменной выделяется память для хранения ее значения. В процессе выполнения программы значение переменной может изменяться.

Для описания переменных используется команда **var** (сокр. от англ. *variable* — переменная).

Формат записи команды:

var <имя переменной>: <тип>;

Для обозначения имени переменной используют буквы латинского алфавита, цифры и знак «_». Первым символом должна быть буква или знак подчеркивания.

Тип данных real в языке Pascal позволяет работать с числами и выполнять над ними арифметические действия.

14.3. Оператор присваивания

Одной из основных команд для обработки данных в программе является оператор присваивания.

Оператор присваивания предназначен для того, чтобы:

- задавать значения переменным;
- вычислять значения арифметического выражения (результат вычисления будет записан как значение переменной).

Формат записи оператора:

<имя переменной>:= <выражение>;

(Рассмотрите пример 14.5.)

В записи арифметического выражения используются знаки математических действий.

Математические операции	Запись в Pascal
+ (сложение)	+
– (вычитание)	–
· (умножение)	*
: (деление)	/

Приоритет выполнения операций соответствует принятому в математике: сначала выполняются умножение и деление, а затем сложение и вычитание. Для изменения порядка действий в выражениях используют скобки.

Пример 14.5. Примеры записи оператора присваивания:

```
x:= 7;
x1:= 3.5;
a_1: = 20 * (x + x1) - 32;
y:= y + 7;
```

Пример 14.6. Запишем оператор присваивания на Pascal для математических выражений:

Выражение	Запись на Pascal
$S = 2(a + b)$	S:=2*(a+b);
$S = a^2$	S:=a*a;
$a = \frac{x + y}{3}$	a:=(x+y)/3;

Пример 14.7. Запишем оператор присваивания, после выполнения которого значение переменной a увеличится в 2 раза, а переменной b уменьшится на 3.

В Pascal допустимы команды присваивания следующего вида:

```
a:= a * 2;
```

Смысл такой команды следующий: из ячейки памяти извлекается значение переменной a, затем оно умножается на 2, результат записывается в ту же ячейку памяти. Старое значение переменной a будет потеряно.

Запись оператора присваивания для изменения значения переменной b следующая:

```
b:= b – 3;
```

В PascalABC.NET определены операторы присваивания со значками +=, −=, *=, /=. Они позволяют изменить значение переменной. Например:

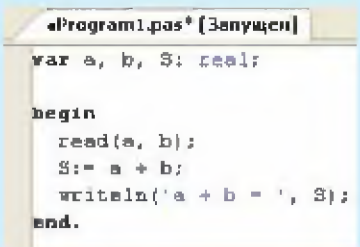
```
a *= 2;  // увеличить a в 2 раза;
b -= 3;  // уменьшить b на 3.
```

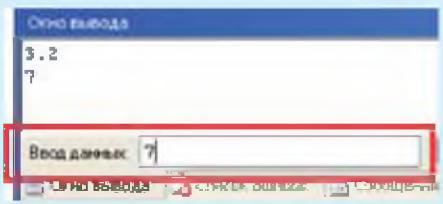
Пример 14.8. Ввести два числа, найти и вывести их сумму.

Текст программы:

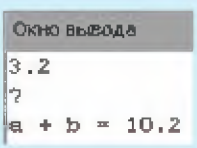
```
var a, b, S: real;
begin
 read(a, b);
 S:= a + b;
 writeln('a + b =', S);
end.
```

Ввод данных:





Результат:



Для записи обыкновенной дроби используется знак деления. Знак умножения опускать нельзя. Целая часть дробного числа отделяется от дробной части точкой. (Рассмотрите примеры 14.6 и 14.7 на с. 93.)

14.4. Ввод данных

Начальные значения переменным можно задавать не только с помощью оператора присваивания, но и путем ввода с клавиатуры. В этом случае, если необходимы вычисления с новым набором значений исходных данных, текст программы не нужно изменять.

Команда `read( )` предназначена для **ввода данных**. В скобках через запятую перечисляются имена переменных, значения которых необходимо ввести.

Ввод данных происходит в нижней части окна программы PascalABC. Для этого используется окно «Ввод данных». После нажатия кнопки «Ввести» или клавиши «Enter» введенные значения переносятся в окно вывода. После завершения работы программы в этом же окне будет выведен результат (пример 14.8).

14.5. Структура программы

Все программы на языке программирования Pascal имеют общую структуру. В программе можно выделить следующие разделы:

- заголовок (не обязателен);
- подключаемые библиотеки (модули) (если подключать дополнительные библиотеки не нужно, раздел отсутствует; известные библиотеки: `Drawman`, `Robot`, `RobTasks`);
- описание переменных с указанием их типа;
- описание вспомогательных алгоритмов (если использовать вспомогательные алгоритмы не нужно, раздел отсутствует);
- **begin** ... **end.** — служебные слова, обрамляющие тело основной программы, в которой находятся исполняемые команды; **begin** начинает исполняемую часть программы, а **end.** (точка в конце обязательна) ее завершает.

В минимально возможном наборе программа состоит из пустого тела программы: **begin end.** Программа, содержащая все разделы, представлена в примере 14.9.

Для каждого раздела определено ключевое служебное слово, которым он начинается. При написании программы ключевые слова выделяются полужирным шрифтом.

Пример 14.9. Программа, содержащая все разделы (подсчитывается количество закрашенных клеток в поле Робота размером 10 × 10):

```
//заголовок программы
//(необязательно)
program Primer;
//описание библиотек
uses Robot, RobTasks;
//описание переменных
var k: integer;
//описание вспомогательных
//алгоритмов
procedure line;
begin
 for var i:= 1 to 9 do
 begin
  if CellisPainted then
   k:= k + 1;
  right;
 end;
 if CellisPainted then
  k:= k + 1;
end;
procedure back;
begin
 for var i:= 1 to 9 do
  left;
end;
//основная программа
begin
//тело программы
 Task('myrob11');
 k:= 0;
 for var i:= 1 to 9 do
 begin
  line; back; down;
 end;
 line;
 writeln(k);
end.
```

1. Какая команда языка программирования Pascal предназначена для вывода данных?
2. Что определяет тип данных?
3. Для чего используется команда присваивания?
4. Какая команда языка программирования Pascal предназначена для ввода данных?
5. Из каких разделов состоит программа на языке программирования Pascal?

Упражнения

1 Для программы из примера 14.2 выполните следующие задания (файл с программой можно скачать):

1. Замените все команды `writeln` на команды `write` и выполните программу. Что произошло? Объясните почему.
2. Как изменится результат работы программы, если в исходном тексте заменить все команды `write` на `writeln`?
3. Измените программу так, чтобы текст на экране выглядел следующим образом:

```
Привет! Я компьютер!!! Я умею выполнять программы!
Ты сегодня написал свою первую программу!!!
Я выполнил твою программу. Посмотри на экране результат!
```

2 Внесите необходимые изменения в программу из примера 14.3, чтобы действия выполнялись в том порядке, в котором записаны, т. е. сначала сложение, а потом умножение.

3 Вводится возраст пользователя в годах. Определите возраст пользователя через 5 лет.

4 Напишите программу, в которой вводятся два числа *a* и *b*. Затем первое число уменьшается в 2 раза, а второе увеличивается на 30. Выведите измененные значения переменных.

5 Напишите программу для вычисления значения числовых выражений:

1. $23 + 45 \cdot 11 - 15$.
2. $\dfrac{37 + 2 \cdot 27}{41}$.
3. $\dfrac{5638 - 2347}{49} + \dfrac{123 \cdot 756}{4455}$.

§ 15. Организация вычислений

При решении любой задачи человеку приходится выполнять следующие действия:

- определение исходных данных (что дано в задаче);
- определение результатов (что нужно получить);
- обработка исходных данных в соответствии с известными правилами так, чтобы получить результат.

Применяя указанные правила к решению задачи по программированию, получим следующие этапы решения задачи:

I. Определение исходных данных.

II. Определение результатов.

II. Составление алгоритма решения задачи.

IV. Определение типов данных для переменных, используемых при реализации алгоритма.

V. Написание программы.

VI. Тестирование программы.

VII. Анализ результатов.

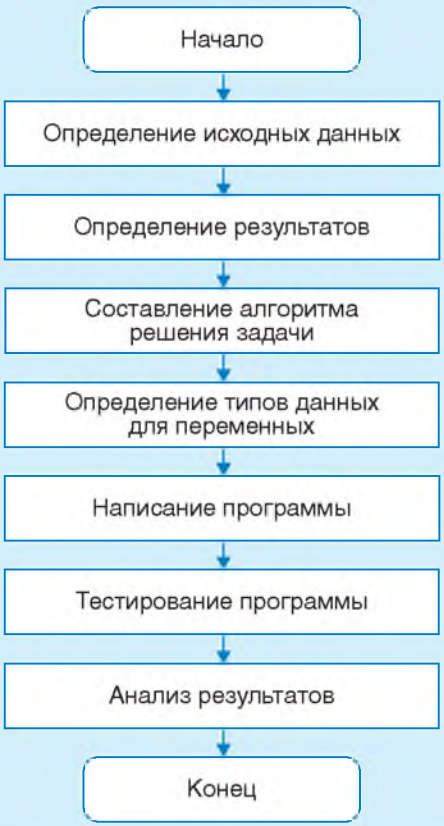
(Рассмотрите пример 15.1.)

Тестирование программы — проверка правильности работы программы при разных наборах исходных данных.

Пример 15.1. Решение задач по физике принято оформлять определенным образом.

Слева записывается то, что дано и что нужно получить, справа — последовательность действий, приводящая к решению задачи. Аналогично оформляются решения задач по химии, геометрии.

Этапы решения задачи по программированию можно представить следующим образом:



Пример 15.2.

V. Программа:

```
var  x, y, z, a:  real;
begin
 write('введите  x  =  ');
 read(x);
 write('введите  y  =  ');
 read(y);
 write('введите  z  =  ');
 read(z);
 a:=(2*x+3*y-z)/(3+x*x);
 writeln('a  =  ',a);
end.
```

VI. Тестирование программы.

Запустите программу и введите значения: $x = 2$, $y = 3$, $z = 1$.

Результат работы программы должен быть следующим:

```
Окно вывода
введите x = 2
введите y = 3
введите z = 1
a = 1.71428571428571
```

А для значений $x = 2$, $y = 4$, $z = 2$ получим:

```
Окно вывода
введите x = 2
введите y = 4
введите z = 2
a = 2
```

VII. Проверка правильности вычислений может быть выполнена на калькуляторе.

15.1. Вычисление значения арифметического выражения

Пример 15.2. Даны переменные x, y, z. Напишем программу для вычисления значения выражения $a = \dfrac{2x + 3y - z}{3 + x^2}$.

Этапы выполнения задания:

I. Определение исходных данных: переменные x, y, z.

II. Определение результатов: переменная a.

III. Алгоритм решения задачи:

1. Ввод исходных данных.
2. Вычисление значения выражения.
3. Вывод результата.

IV. Описание переменных.

Все переменные, определенные для решения задачи, имеют тип `real`.

В приведенном примере перед каждой командой ввода записана команда вывода с пояснениями о том, значение какой переменной нужно вводить.

При написании программ для вычисления значения арифметического выражения часто допускают следующие ошибки:

`2x+3;` `(x-4)(x+2);`	Пропущен знак *
`(2+y)/(x*x;`	Пропущена скобка

Будьте внимательны!

15.2. Использование языка программирования для решения задач

Пример 15.3. Напишем программу для решения геометрической задачи. Задан квадрат с длиной стороны a. Требуется найти его площадь и периметр.

Этапы выполнения задания:

I. Определение исходных данных: переменная a (длина стороны).

II. Определение результатов: переменные S (площадь) и P (периметр).

III. Алгоритм решения задачи:

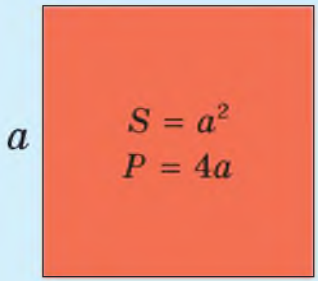
1. Ввод исходных данных.
2. Вычисление значений площади в математике производится по формуле $S = a^2$, а периметра — по формуле $P = 4a$. В программе этим формулам будут соответствовать команды присваивания: S:= $a * a$; P:= $4 * a$.
3. Вывод результата.

IV. Описание переменных:

Все переменные, определенные для решения задачи, имеют тип `real`.

Обратите внимание: запись формул в операторе присваивания может отличаться от записи математических формул.

Пример 15.3.



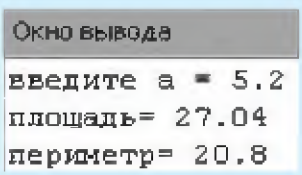
V. Программа:

```
var a,S,P: real;
begin
 write('введите a = ');
 read(a);
 s:= a*a;
 p:= 4*a;
 writeln('площадь = ',S);
 writeln('периметр = ',P);
end.
```

VI. Тестирование программы.

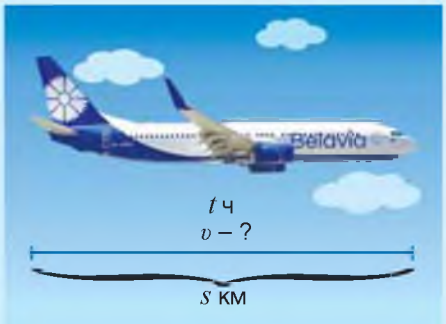
Запустите программу и введите значение $a = 5.2$.

Результат работы программы должен быть следующим:



VII. Проверка правильности вычислений может быть выполнена на калькуляторе.

Пример 15.4.



V. Программа:

```
var s, t, v: real;
begin
 write('введите s = ');
 read(s);
 write('введите t = ');
 read(t);
 v:= s / t;
 writeln('скорость = ', v);
end.
```

VI. Тестирование программы.

Запустите программу и введите значения $s = 3550$ и $t = 4$.

Результат работы программы должен быть следующим:

```
Окно вывода
введите s = 3550
введите t = 4
скорость = 887.5
```

VII. Проверка правильности вычислений может быть выполнена на калькуляторе.

Пример 15.4. Напишем программу для решения физической задачи. Расстояние между двумя городами составляет s км. Самолет пролетает это расстояние за t ч. Определите скорость самолета.

Этапы выполнения задания:

I. Определение исходных данных: переменные s (расстояние) и t (время).

II. Определение результатов: переменная v (скорость).

III. Алгоритм решения задачи:

1. Ввод исходных данных.
2. Согласно формуле расстояния: $s = vt$. Отсюда выразим v: $v = \frac{s}{t}$.
3. Вывод результата.

IV. Описание переменных:

Все переменные, определенные для решения задачи, имеют тип `real`.

При написании программ обращайте внимание на форматирование их текста:

- в первой позиции на экране пишут только слова **var**, **begin** **end**, а остальные со сдвигом на 2—4 позиции вправо;
- если в программе несколько частей, то их можно отделить друг от друга пустой строкой.

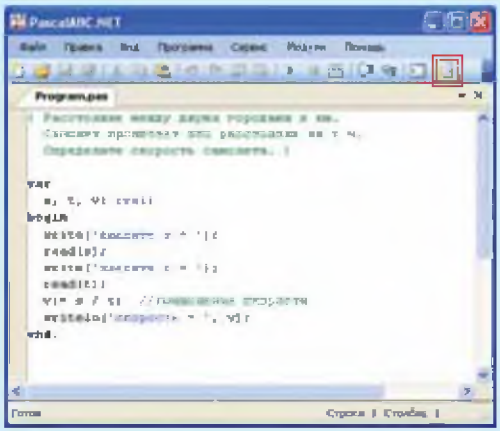
Выполнение этих правил повышает читаемость программы.

В программе можно использовать **комментарии** — текст, который не анализируется при запуске программы на выполнение.

Текст после символов // считается комментарием и выделяется зеленым цветом (пример 15.5).

В комментариях удобно записывать условие задачи. Пояснения к командам, записанные как комментарии, нужны для понимания действия, выполняемого командой. В языке программирования Pascal комментарии можно записать в несколько строк. Тогда текст, являющийся комментарием, заключают в фигурные скобки.

Пример 15.5. Программа с комментариями и отформатированным кодом:



Значок для форматирования кода на панели инструментов имеет следующий вид:

1. Перечислите этапы решения задачи по программированию.
2. Что понимают под тестированием программы?
3. Для чего можно использовать комментарии?

Упражнения

1 Даны x, y, z. Напишите программу для вычисления значения арифметических выражений.

1. $a = \dfrac{x + y - z}{x^2 + 2}$. 2. $a = 5\dfrac{2x - z}{3 + y^2}$. 3. $a = (1 + z)\dfrac{x + \dfrac{y}{x^2 + 4}}{2 + \dfrac{1}{x^2 + 4}}$.

2 Напишите программу для решения геометрической задачи.

1. Найдите длину окружности и площадь круга заданного радиуса.

2*. Найдите угол при основании равнобедренного треугольника, если известен угол при вершине.

3 Напишите программу для решения физической задачи.

1. Велосипедист едет с постоянной скоростью v км/ч. За сколько минут он проедет расстояние в s км?

2*. Автомобиль проходит первую часть пути длиной s_1 км за t_1 мин, участок пути длиной s_2 км за t_2 мин и участок длиной s_3 км за t_3 мин. Найдите среднюю скорость автомобиля в км/ч.

4 Напишите программу для решения химической задачи.

1. В организме человека на долю атомов кислорода приходится 65 % от массы тела. Найдите массу атомов кислорода для своей массы тела.

2*. Масса атома кислорода равна $26.56 \cdot 10^{-27}$ (это число на языке Pascal записывается так: 26.56E-27, буква E — английская). Сколько атомов кислорода содержится в вашем теле?

§ 16. Реализация алгоритмов работы с целочисленными данными

В PascalABC определены различные типы данных для работы с целыми числами, позволяющие выполнять действия над данными из разных числовых диапазонов. Чем больше диапазон, тем больше места в памяти компьютера отводится для хранения переменных.

Некоторые целочисленные типы данных:

Тип	Диапазон значений
shortint	–128..127
smallint	–32768..32767
integer, longint	–2147483648..2147483647
byte	0..255
word	0..65535

16.1. Целочисленный тип данных

Часто при решении задач нужно работать с целыми числами. Для этого в Pascal используется тип данных `integer`. С помощью переменных этого типа можно задавать целые числа из диапазона от –2147483648 до 2147483647. Для типа данных `integer` определены следующие операции:

Математические операции	Запись в Pascal
+ (сложение)	+
– (вычитание)	–
· (умножение)	*
целочисленное деление	div
нахождение остатка	mod

Для целочисленных данных не определена операция деления, как для действительных чисел. При попытке использовать операцию деления будет выдана ошибка (пример 16.1).

Для организации вычислений с целыми числами определены операции `div` и `mod`. Эти операции имеют такой же приоритет, как и операции деления и умножения.

Пример 16.2. Даны два целых числа *a* и *b*. Напишем программу, которая находит целую часть от деления *a* на *b* и остаток.

Этапы выполнения задания:

I. Определение исходных данных: переменные *a* и *b*.

II. Определение результатов: переменные *c* (целочисленное частное) и *d* (остаток).

III. Алгоритм решения задачи:

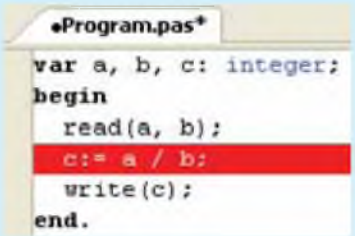
1. Ввод исходных данных.
2. Целочисленное частное находим как результат операции `a` **`div`** `b`, остаток — `a` **`mod`** `b`.
3. Вывод результата.

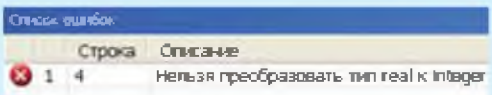
IV. Описание переменных.

Все переменные, определенные для решения задачи, имеют тип `integer`.

Значение, выдаваемое как результат операции `mod`, может

Пример 16.1. Ошибка использования операции деления для целочисленных типов данных:





Пример 16.2.

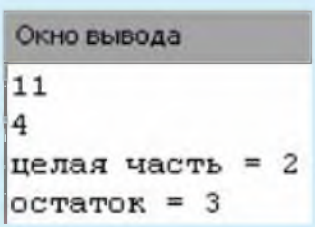
V. Программа:

```
var a, b, c, d: integer;
begin
 read(a, b);
 c:= a div b;
 d:= a mod b;
 writeln('целая часть = ',
 c);
 writeln('остаток = ', d);
end.
```

VI. Тестирование программы.

Запустите программу и введите значения $a = 11$ и $b = 4$.

Результат работы программы должен быть таким, как показано ниже:



Результат операций div и mod для разных чисел:

a	b	a div b	a mod b
17	3	5	2
−17	3	−5	−2
17	−3	−5	2
−17	−3	5	−2

Так, a **mod** b = a − (a **div** b) * b.

Пример 16.3.

V. Программа:

```
var c, m, s: integer;
begin
 write('введите c = ');
 readln(c);
 {Минуты}
 m:= c div 60;
 {Секунды}
 s:= c mod 60;
 write(m, ':', s);
end.
```

VI. Тестирование программы.

Запустите программу и введите значение $c = 137$.

Результат работы программы:

```
Окно вывода
введите c = 137
2:17
```

Для значения $c = 24$ получим:

```
Окно вывода
введите c = 24
0:24
```

отличаться от математического определения остатка (в математике под остатком понимают неотрицательное число). Если остаток не равен нулю, то знак числа, являющегося результатом операции mod, определяет знак делимого.

16.2. Использование целочисленных данных для решения задач

Пример 16.3. Пусть таймер показывает время только в секундах. Напишем программу, переводящую время в минуты и секунды.

Этапы выполнения задания:

I. Определение исходных данных: переменная c (время в секундах).

II. Определение результатов: переменные m (полное количество минут) и s (остаток секунд).

III. Алгоритм решения задачи:

1. Ввод исходных данных.
2. Для нахождения полного числа минут нужно найти целую часть от деления исходного числа секунд на 60.
3. Оставшиеся секунды находим как остаток от деления исходного числа секунд на 60.
4. Вывод результата.

IV. Описание переменных.

Переменные, определенные для решения задачи, имеют тип integer.

Пример 16.4. Задано двузначное число. Нужно поменять местами первую и вторую цифры числа.

Этапы выполнения задания:

I. Определение исходных данных: переменная *a* (исходное число).

II. Определение результатов: переменная *b* (преобразованное число).

III. Алгоритм решения задачи:

1. Ввод исходных данных.

2. Для преобразования числа необходимо выполнить следующие действия:

а) в переменной *a*1 сохраним вторую цифру числа. Для выделения цифры из числа нужно найти остаток от деления исходного числа на 10 (`a mod 10`);

б) для выделения первой цифры (переменная *a*2) нужно найти целую часть от деления числа на 10;

в) искомое число *b* получим, если умножим *a*1 на десять и к полученному произведению прибавим значение переменной *a*2.

3. Вывод результата.

IV. Описание переменных.

Все переменные, определенные для решения задачи, имеют тип `integer`.

Пример 16.4.

V. Программа:

```
var
 a, b, a1, a2: integer;
 begin
 write('введите a = ');
 readln(a);
 {Выделение последней
цифры}
 a1:= a mod 10;
 {Выделение первой цифры}
 a2:= a div 10;
 b:= a1 * 10 + a2;
 write('результат = ', b);
end.
```

VI. Тестирование программы.

Запустите программу и введите значение *a* = 25.

Результат работы программы должен быть следующим:

```
Окно вывода
введите a = 25
результат = 52
```

Издавна на Руси применялась система мер, отличная от современной Международной системы единиц (СИ). Например:

1 локоть = 45 см;
1 аршин = 16 вершков;
1 вершок = 4 ногтя;
1 ноготь ≈ 11 мм.

Пример 16.5.

V. Программа:

```
var l, m, s, x: integer;
begin
 write('введите l = ');
 readln(l);
 x:= l * 45;
 {метры}
 m:= x div 100;
 {сантиметры}
 s:= x mod 60;
 write(l, 'локтей = ');
 write(m, ' м ', s, ' см');
end.
```

VI. Тестирование программы.

Запустите программу и введите значение l = 7.

Результат работы программы должен быть следующим:

```
Окно вывода
введите l = 7
7 локтей = 3 м 15 см
```

Пример 16.5. В исторической книге длина отреза ткани измерялась в локтях. Напишем программу, которая переведет локти в метры и сантиметры.

Этапы выполнения задания:

I. Определение исходных данных: переменная l (локти).

II. Определение результатов: переменные m (метры) и s (сантиметры).

III. Алгоритм решения задачи:

1. Ввод исходных данных.
2. Сначала переведем локти в сантиметры. Для этого количество локтей нужно умножить на 45 и сохранить значение в переменной x.
3. Для определения числа метров найдем целую часть от деления x на 100.
4. Оставшиеся сантиметры можно найти как остаток от деления x на 100.
5. Вывод результата.

IV. Описание переменных:

Все переменные, определенные для решения задачи, имеют тип `integer`.

Какой тип данных можно использовать в Pascal для работы с целочисленными данными?

2. Какое максимальное значение можно задать переменной типа `integer`?

3. Какие операции определены для целочисленных данных?

Упражнения

1 Вася написал программу, которая переводит длину из метров в километры и метры. Но он не может решить, где нужно использовать `div`, а где `mod`. Помогите ему. Откройте файл и исправьте программу.

```
var d, m, k: integer;
begin
  write('введите d = ');
  readln(d);
  k:= d … 1000;
  m:= d … 1000;
  write(d, ' м = ');
  write(k, ' км ', m, ' м');
end.
```

2 Ответьте на вопросы для примера 16.4.

1. При каких значениях переменной a значение переменной b будет таким же?
2. Всегда ли в результате выполнения программы мы будем получать двузначное число? Почему?
3. Попробуйте ввести трехзначное число (например, 125). Объясните получившийся результат.

3 Напишите программы для решения задач. Используйте операции `div` и `mod`.

1. Задано двузначное число. Найдите среднее арифметическое цифр числа.
2. Задано двузначное число. Найдите разность между количеством десятков и единиц.
3. Дана масса в граммах. Переведите ее в килограммы и граммы.
4. Площадь участка измеряется в арах. Найдите количество полных км^2.

4* Для старорусской системы весов известны следующие соотношения:

1 берковец = 10 пудов = 400 фунтов = 38 400 золотников.

Напишите программу, которая переводит массу, заданную в золотниках, в фунты, пуды и берковцы.