ВВЕДЕНИЕ

§ 1. Алгоритм и его свойства

Термин «алгоритм» произошел от фамилии математика IX в. Мухаммеда ибн Муса аль-Хорезми, который сформулировал правила четырех основных арифметических действий. Именно эти правила вначале и назывались алгоритмами, но позже алгоритмом стали называть любой способ вычислений, единый для некоторого класса исходных данных.

Пример 1.1. Алгоритм «Решето Эратосфена» позволяет получить простые числа, не превосходящие N.

1. Выпишем подряд все натуральные числа от 2 до N.
2. Возьмем первое число 2 и зачеркнем каждое второе число, начиная отсчет со следующего за двойкой числа.
3. Возьмем первое незачеркнутое число, которое больше 2 (число 3), и зачеркнем каждое третье число, начиная отсчет от числа, стоящего после 3 (учитывая и ранее зачеркнутые числа).
4. Продолжим действия до тех пор, пока первое незачеркнутое число не окажется больше N.
5. В результате незачеркнутыми окажутся все простые числа, не превосходящие N, и только они.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

В информатику понятие алгоритма пришло из математики.

Алгоритм — точно определенная система понятных исполнителю предписаний, формальное выполнение которых позволяет получить решение задачи для любого допустимого набора исходных данных за конечное число шагов.

Приведенное определение не является определением в математическом смысле слова, поскольку в нем использованы другие неопределенные понятия — «система предписаний», «формальное выполнение» и др. Это описание понятия «алгоритм», раскрывающее его сущность. (Рассмотрите пример 1.1.) Описание можно уточнить, указав общие свойства, которые характерны для алгоритмов. К ним относятся: дискретность, детерминированность (определенность), понятность, результативность, конечность, массовость.

Дискретность. Алгоритм разбивается на отдельные действия (шаги). Выполнение очередного действия возможно только после завершения предыдущего. При этом набор промежуточных данных конечен и получается по определенным правилам из данных предыдущего действия. **Команда** является специальным указанием для исполнителя, предписывающим ему произвести каждое отдельное дей-

ствие. Команды, которые относят к системе команд исполнителя, называют простыми; другие команды могут быть определены через простые.

Детерминированность. Если алгоритм неоднократно применить к одним и тем же исходным данным, то каждый раз должны получаться одни и те же промежуточные результаты и один и тот же выходной результат. Данное свойство означает, что результат выполнения алгоритма определяется только входными данными и командами самого алгоритма и не зависит от исполнителя алгоритма.

Понятность. Алгоритм не должен содержать команд, смысл которых исполнитель может воспринимать неоднозначно. Запись алгоритма должна быть четкой, полной и понятной. У исполнителя не должно возникать необходимости в принятии каких-либо самостоятельных решений.

Результативность. При точном выполнении команд алгоритма результатом должен быть ответ на вопрос задачи. Если способ получения последующих величин из каких-либо исходных не приводит к результату, то должно быть указано, что следует считать результатом исполнения алгоритма. В качестве одного из возможных ответов может быть установление того факта, что задача не имеет решения.

Конечность. Реализуемый по заданному алгоритму процесс должен остановиться через конечное число шагов и выдать искомый результат. Это свойство тесно связано со свойством результативности.

Необходимость построения формального определения алгоритма привела к появлению в 20—30-х гг. XX в. теории алгоритмов. Для определения различными математиками были предложены:

- машина Тьюринга;
- машина Поста;
- нормальный алгоритм Маркова.

Существовали и другие определения алгоритма. Впоследствии было доказано, что все они эквивалентны.

Алан Мэтисон Тьюринг (1912—1954) — английский логик и математик, оказавший существенное влияние на развитие информатики. Предложенная им в 1936 г. абстрактная вычислительная Машина Тьюринга позволила формализовать понятие алгоритма, которое используется в теоретических и практических исследованиях.

Эмиль Леон Пост (1897—1954) — американский математик и логик. Известен своими трудами по математической логике. Предложил абстрактную вычислительную машину — машину Поста (1936).

Пример 1.2. Часто рецепты приготовления каких-либо блюд называют алгоритмами. В данном случае нарушается свойство детерминированности, поскольку при приготовлении блюда разными людьми результат может быть разным (например, он может зависеть от того, на какой плите готовили, или от качества продуктов). Кроме того, в рецептах часто бывают фразы «посолить по вкусу», «добавить 2—3 ложки сахара» и т. д., которые нарушают свойство понятности.

Пример 1.3. Задача может иметь решение, но сформулировать алгоритм решения этой задачи не всегда удается. Если человеку предложить фотографии животных, то он достаточно быстро сможет разделить их на две группы: дикие и домашние. Однако сформулировать алгоритм, согласно которому он это сделал, на сегодняшний день не представляется возможным. Некоторые задачи классификации сегодня успешно решаются системами искусственного интеллекта с помощью методов машинного обучения. Однако характерной чертой таких методов является не прямое решение задачи, а процесс обучения в ходе анализа множества решений сходных задач.

Пример 1.4. Способы проверки алгоритма на правильность работы:

- математическое доказательство;
- использование специально подобранных тестов.

Массовость. Алгоритм пригоден для решения любой задачи из некоторого класса задач, т. е. начальная система величин может выбираться из некоторого множества исходных данных, которое называется **областью применимости алгоритма**.

Для практического решения задач на компьютере наиболее существенно свойство массовости. Как правило, ценность программы для пользователя будет тем выше, чем больший класс однотипных задач она позволит решать.

Если разработанная последовательность действий не обладает хотя бы одним из перечисленных выше свойств, то она не может считаться алгоритмом. Для понимания свойств алгоритма рассмотрите примеры 1.2 и 1.3.

Для одного и того же алгоритма могут существовать различные формы записи: текстовое описание, блок-схема, машина Тьюринга и др. Независимо от формы записи любой алгоритм может быть представлен с использованием базовых алгоритмических конструкций: **следование, цикл** и **ветвление**.

В рамках теории алгоритмов происходит анализ различных алгоритмов решения задачи для выбора наиболее эффективного (оптимального). Разработка инструментов для анализа эффективности алгоритмов — одна из задач теории алгоритмов.

Любой алгоритм нужно проверять на правильность работы (пример 1.4).

?

1. Что такое алгоритм?
2. Какие свойства характеризуют алгоритм?
3. Какие базовые алгоритмические конструкции используются при составлении алгоритмов?

Упражнения

1 Прокомментируйте основные свойства алгоритма для решета Эратосфена.

2 Аль-Хорезми составил алгоритмы арифметических действий еще в IX в. Составьте алгоритмы выполнения арифметических действий в столбик. Проверьте для составленных алгоритмов основные свойства.

3 Приведите примеры алгоритмов, обладающих указанными признаками.

1. Используется только алгоритмическая конструкция *следование*.
2. Присутствует алгоритмическая конструкция *ветвление*.
3. Присутствует алгоритмическая конструкция *цикл*.
4. Используются подпрограммы.

4 Опишите последовательность действий для решения задачи «Волк, коза и капуста».

Однажды крестьянину понадобилось перевезти через реку волка, козу и капусту. У крестьянина есть лодка, в которой может поместиться, кроме самого крестьянина, только один объект — или волк, или коза, или капуста. Если крестьянин оставит без присмотра волка с козой, то волк съест козу; если крестьянин оставит без присмотра козу с капустой, коза съест капусту. Как крестьянину перевезти на другой берег и волка, и козу, и капусту в целости и сохранности?

Будет ли полученная последовательность действий алгоритмом? Какое свойство алгоритма не выполняется? Возможно ли переформулировать задачу так, чтобы аналогичная последовательность действий стала алгоритмом?

5 Есть двое песочных часов на 3 мин и на 8 мин. Как с их помощью отмерить 7 мин? Определите систему команд исполнителя, который может решать эту задачу, и составьте для него последовательность действий, приводящую к ответу. Какие алгоритмические конструкции были использованы при реализации? Можно ли считать полученную последовательность действий алгоритмом? Какое свойство алгоритма не выполняется? Возможно ли переформулировать задачу так, чтобы аналогичная последовательность действий стала алгоритмом?

§ 2. Языки программирования

2.1. Высокоуровневые языки программирования

Любой алгоритм рассчитан на конкретного исполнителя, имеющего свою систему команд. Алгоритм для компьютера должен быть записан с помощью команд, которые компьютер может выполнять.

Первым высокоуровневым языком программирования, который был реализован практически, стал в 1949 г. Краткий код (Short Code). Операции и переменные в этом языке программирования кодировались двухсимвольными сочетаниями.

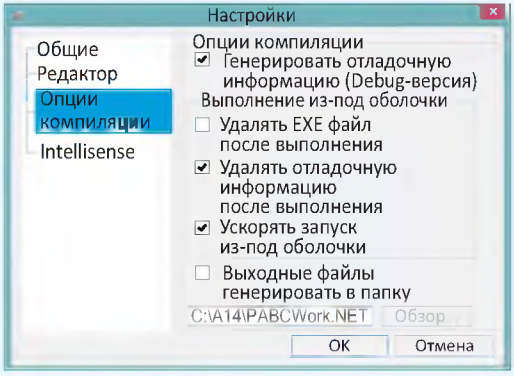
Пример 2.1. Некоторые языки программирования высокого уровня:

- C++;
- Python;
- Pascal (Delphi);
- C#;
- Java;
- JavaScript;
- Perl;
- Fortran;
- VisualBasic;
- Lisp.

Пример 2.2. При трансляции программы происходит преобразование текста с одного языка на другой. Различают следующие виды трансляции: компиляция и интерпретация.

Компилятор — транслятор, преобразующий исходный код с какого-либо языка программирования на машинный. В результате создается исполняемый файл, который может быть выполнен непосредственно в операционной системе.

Среда программирования PascalABC имеет встроенный компилятор, опции которого можно настроить, выполнив команду **Сервис → Настройки**:



Интерпретатор — транслятор, который может работать двумя способами:

- читать код и исполнять его сразу (чистая интерпретация);
- читать код, создавать в памяти промежуточное представление кода (байт-код или p-код), выполнять промежуточное представление кода.

Чистая интерпретация применяется для языков JavaScript, VBA, Lisp и др. Примеры интерпретаторов, создающих байт-код: Perl, PHP, Python и др.

Устройством, выполняющим команды в компьютере, является процессор. Систему команд процессора называют машинным языком или машинным кодом. Каждая команда процессора записывается в двоичном коде. Для записи этих команд в символьной форме используют язык Ассемблер. Ассемблер является языком низкого уровня, поскольку содержит команды, отражающие машинный код.

В языках программирования высокого уровня используются команды, которые объединяют последовательности машинных команд. Программы, написанные на таких языках, проще для понимания человеком, поскольку для обозначения команд используют слова естественного языка (чаще всего английского).

Языки программирования высокого уровня, или высокоуровневые языки программирования (пример 2.1), были разработаны для того, чтобы суть алгоритма могла не зависеть от аппаратной реализации компьютера. Для преобразования текста программы, написанной на языке высокого уровня, в элементарные машинные команды используются специальные программы — **трансляторы** (пример 2.2). Например, в тексте программы достаточно написать «while», а уже транслятор языка переведет эту команду в последовательность машинных кодов. Принципы работы трансляторов зависят от аппаратного и программного обеспечения компьютера.

Языки программирования высокого уровня являются формальными

искусственными языками. У каждого языка программирования можно выделить две составляющие: синтаксис и семантику. **Синтаксис** (грамматика языка) — совокупность правил для написания программы. **Семантика** — смысловая сторона языка (определяет смысловое содержание языковой конструкции).

Текст программы на языке высокого уровня представляет собой обычный текстовый файл. Для его «чтения» и превращения в последовательность машинных команд выполняется анализ текста программы — проверка на соответствие синтаксическим правилам и семантике данного языка программирования. В случае обнаружения несоответствия компилятор может выдавать сообщения об ошибках (пример 2.3).

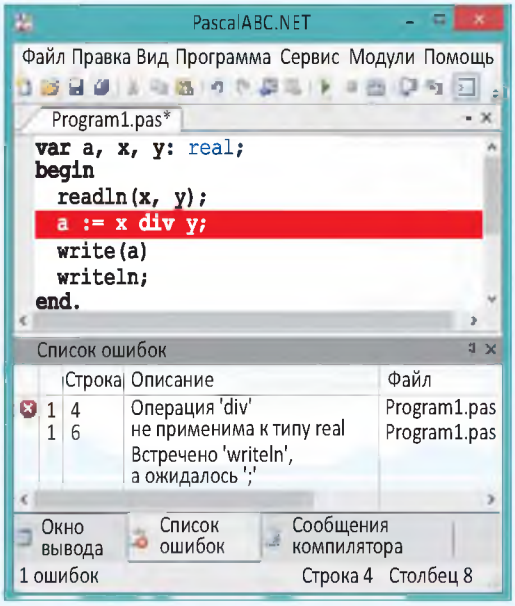
За время существования вычислительных машин было создано более 8 тыс. языков программирования, и ежегодно появляются новые. Некоторые из них являются узкоспециальными, другие используются миллионами программистов по всему миру.

Существуют различные подходы к классификации языков программирования. По степени отличия семантики языка от машинного кода языки делят на низкоуровневые и высокоуровневые. Часто языки программирования делят на компилируемые и интерпретируемые, однако такое деление условно, поскольку для любого традиционно компилируемого языка (такого, как Паскаль) можно написать интерпретатор.

В 1951 г. американский ученый Грейс Мюррей Хоппер (1906—1992) создала первый в мире компилятор и ввела сам этот термин.

Компилятор осуществлял функцию объединения команд, производил выделение памяти компьютера и преобразование команд высокого уровня (в то время псевдокодов) в машинные команды.

Пример 2.3. Семантическая (строка 4) и синтаксическая (строка 6) ошибки в среде PascalABC.



Роберт В. Флойд (1936—2001) — американский ученый в области теории вычислительных систем. Лауреат премии Тьюринга. Впервые термин «парадигма программирования» был применен Р. Флойдом в 1978 г. во время получения премии Тьюринга: «Если прогресс искусства программирования в целом требует постоянного изобретения и усовершенствования парадигм, то совершенствование искусства отдельного программиста требует, чтобы он расширял свой репертуар парадигм».

Флойд отмечал, что парадигмы программирования не являются взаимоисключающими, они могут сочетаться, обогащая инструментарий программиста.

Пример 2.4. Основная идея структурного программирования заключается в том, что программа должна иметь простую структуру, быть хорошо читаемой и легко модифицируемой. Структурированность кода поддерживается посредством подпрограмм, которые вызываются из других подпрограмм. Структурное программирование поддерживают такие языки, как Pascal, Go, C и многие другие языки программирования.

Пример 2.5. Особенность языков процедурного программирования заключается в том, что задачи разбиваются на шаги и решаются шаг за шагом. Решение для каждого отдельного шага оформляется в виде отдельной процедуры. К процедурным языкам относят: C, Pascal, Lua и др.

2.2. Парадигмы программирования

В истории развития языков программирования можно выделить различные парадигмы программирования — совокупность идей и понятий, определяющих стиль программирования. Между парадигмами и языками программирования нет жесткой связи. Парадигма показывает один из возможных способов использования средств языка программирования для написания кода программы. Часто язык программирования, созданный в рамках одной парадигмы, через некоторое время модернизируется, расширяется и начинает использоваться в рамках другой парадигмы.

Рассмотрим некоторые парадигмы программирования.

Структурное программирование — парадигма программирования, в основе которой лежит представление программы в виде блоков иерархической структуры. Разработана в конце 1960-х — начале 1970-х гг. В соответствии с данной парадигмой любая программа состоит из трех базовых управляющих структур: *ветвление, цикл* и *последовательность*; кроме того, используются подпрограммы (пример 2.4). Разработка программы ведется пошагово, методом «сверху вниз» (нисходящее программирование): сначала определяются цели решения задачи, а затем идет детализация каждого шага, который, став отдельной задачей, также может детализироваться.

Процедурное программирование — парадигма программирования, при которой последовательно выполняе-

мые команды можно собрать в подпрограммы с помощью механизмов самого языка (пример 2.5). Это концепция программирования «снизу вверх», или концепция восходящего программирования — разработка программ начинается с разработки подпрограмм (процедур, функций), в то время как проработка общей схемы не закончилась.

Парадигмы структурного и процедурного программирования основаны на подпрограммах. Разница заключается в порядке их разработки: «сверху вниз» или «снизу вверх».

Функциональное программирование — парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании. Основывается функциональное программирование на вычислении результатов функций от исходных данных и результатов других функций и не предполагает явного хранения состояния программы (пример 2.6).

Объектно-ориентированное программирование (ООП) — парадигма программирования, основанная на представлении программы в виде совокупности объектов и отражении их взаимодействия. Концепция ООП позволяет объединить данные и алгоритмы их обработки в единую структуру, называемую классом. Каждый объект является экземпляром какого-либо класса (пример 2.7). В ООП программа представляет собой набор объектов, имеющих состояние и поведение. Состояние и поведение объекта может измениться в результате события.

Пример 2.6. В основе функциональных языков лежит лямбда-исчисление (λ-исчисление) — математическая теория для формализации понятия вычислимости. К ним относят: Haskell, Lisp, F# и др.

Пример 2.7. В программе *текстовый редактор* объектами могут быть абзац текста, меню, кнопки и т. д. В классе, который описывает кнопку, содержатся данные о размере кнопки, ее внешнем виде, алгоритмы, позволяющие «нажать» кнопку или «навести на нее мышь» и др.

Конкретная кнопка, например Ч, является объектом. Такое событие, как «клик мышью по такой кнопке», изменит начертание текста. Событие «двойной клик мышью по абзацу текста» выделит его и т. д.

К объектно-ориентированным языкам относят:

- C++;
- Java;
- Delphi;
- Python;
- Ruby и др.

Развитие объектно-ориентированного программирования часто связывают с понятиями «событие» (событийно-ориентированное программирование) и «компонент» (компонентное программирование).

Среда PascalABC предоставляет возможность создавать оконные приложения. При разработке интерфейса программы используются визуальные компоненты, а программный код основан на событийном программировании. Создавать такие приложения вы научитесь в 11-м классе.

Пример 2.8. Мультипарадигменные языки программирования чаще всего поддерживают процедурную (структурную), объектно-ориентированную и функциональную парадигмы: C++, Python, JavaScript, Ruby, C#.

Существуют и другие классификации и способы сравнения различных языков программирования[1].

Пример 2.9. Учебный язык обеспечивает простоту, ясность и удобочитаемость конструкций языка. Как учебные языки программирования разрабатывались: Basic, Pascal, Logo, Scratch.

Пример 2.10. Некоторые эзотерические языки служат для проверки математических концепций (Thue, Unlambda), другие создаются для развлечения. Часто они пародируют «настоящие» языки программирования или являются абсурдным воплощением концепций программирования (Smetana, Var'aq, FiM++ и др.).

Пример 2.11. Псевдокод алгоритма нахождения суммы квадратов первых n натуральных чисел.

```
ввод n;
S = 0
нц для i от 1 до n
  S = S + i * i
кц
```

Служебные (ключевые) слова — зарезервированные слова, которые имеют специальные значения для компилятора. Их нельзя использовать как идентификаторы в программах.

Программа в целом — это объект. Для выполнения своих функций она обращается к входящим в нее объектам, которые, в свою очередь, могут обращаться к другим объектам, реализовывать свои методы или реагировать на события. Объектно-ориентированное программирование особенно важно при реализации крупных проектов.

Большинство современных языков программирования являются **мультипарадигменными** — поддерживают сразу несколько парадигм программирования (пример 2.8).

Отдельно рассматривают такие классы языков программирования, как **учебные** (пример 2.9) и **эзотерические** языки программирования (пример 2.10).

Часто для записи алгоритмов применяют **псевдокод** — язык описания алгоритмов, использующий ключевые слова языков программирования, но опускающий детали, несущественные для понимания алгоритма (например, описания переменных). Главная цель использования псевдокода — обеспечить понимание алгоритма человеком, сделать описание более воспринимаемым, чем исходный код на языке программирования. При написании псевдокода может использоваться лексика русского языка (пример 2.11).

2.3. Основные структурные элементы языка программирования

Для записи элементов языка программирования используется алфавит. **Алфавиты** большинства языков

[1] https://ru.wikipedia.org/wiki/Сравнение_языков_программирования (дата доступа: 10.02.2019).

программирования близки друг другу по синтаксису и, как правило, используют буквы латинского алфавита, арабские цифры и общепринятые спецсимволы (знаки препинания, знаки математических операций и сравнений, разделители, служебные слова). Большинство распространенных языков программирования содержат в своем алфавите следующие элементы:

- буквы — {AaBbCcDd...};
- цифры — {0 1 2 3 4 5 6 7 8 9};
- знаки арифметических операций — {× / + −...};
- знаки сравнения — {< > = ...};
- разделители — {. , ; : ( ) { } [ ]... };
- служебные слова — {**if** **while** **for** и т. д.};
- комментарии — любой набор символов и др.

Несмотря на значительные различия между языками, многие фундаментальные понятия в большинстве языков программирования схожи между собой (пример 2.12). Согласно известной формуле Н. Вирта «Алгоритмы + структуры данных = программы», рассматривая язык программирования, нужно говорить о способах записи команд алгоритма с помощью операторов и организации работы с данными (пример 2.13).

Пример 2.12. Некоторые служебные слова (в алфавитном порядке) в разных языках программирования.

Pascal	Python	C++
const, do, else, for, if, then, var, while	def, elif, else, for, if, return, while	const, do, else, for, if, return, while

Пример 2.13. Структурная схема процедурного языка программирования.

Операторы

Одним из основных понятий всех языков программирования является **оператор**, который представляет собой законченную фразу языка и является предписанием на выполнение конкретных действий по обработке данных. Программа строится из операторов так же, как текст литературного

Выражения строятся из величин (констант и переменных), функций, скобок, знаков операций и т. д. Тип выражения определяется результатом вычислений. Выражения могут принимать числовые, логические, символьные, строковые и другие значения.

Выражение 5 + 3 является числовым, а выражение $A + B$ может иметь самый разный смысл в зависимости от того, что стоит за идентификаторами A и B (если A и B — строки, то результат — строка, которая получилась при конкатенации исходных строк).

Пример 2.14. Запись оператора цикла (поиск суммы квадратов первых *n* натуральных чисел).

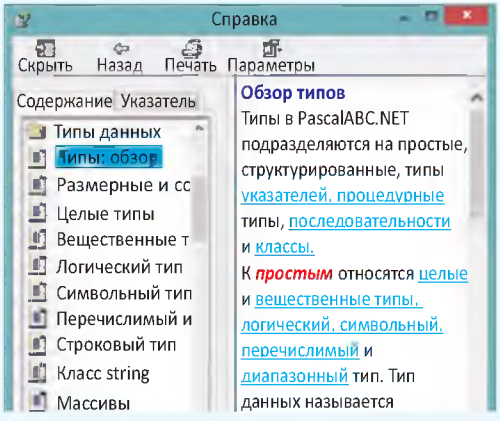
Pascal:
```
for var i := 1 to n do
  s := s + i * i;
```

Python:
```
for i in range(n + 1):
    s += i * i
```

C++:
```
for (int i = 1; i <= n; i++)
    s += i * i;
```

Пример 2.15. Классификация данных в языке программирования.

Подобная структура типов данных присуща многим языкам программирования. С другими типами данных, которые используются в языке PascalABC, можно познакомиться в справочной системе.



произведения формируется из предложений.

Выделяют следующие операторы: оператор присваивания, оператор условного перехода (ветвления), оператор цикла (пример 2.14), оператор выбора, составной оператор, иногда используют пустой оператор, оператор безусловного перехода и др.

Все операторы языка в тексте программы отделяются друг от друга с помощью явных или неявных разделителей (в Pascal таким разделителем является «;»). Операторы выполняются в том порядке, в котором они записаны в тексте программы. Порядок выполнения операторов может быть изменен только посредством управляющих операторов: ветвления, цикла и др.

Данные

Большая часть операторов предназначена для обработки величин. Величина характеризуется типом, именем и значением. Типы данных могут быть простыми и структурированными (пример 2.15). Величина простого типа в каждый момент имеет одно значение. Величина структурированного типа состоит из величин других типов. Например, строка состоит из символов, каждый отдельный символ строки имеет свое значение (код). Самым распространенным структурированным типом данных является массив, с которым вы познакомитесь в этом учебном году.

Всем объектам в языках программирования (переменным, функциям, процедурам и др.) даются имена. Имя объекта в программе называют **иден-**

тификатором (от слова «идентифицировать»). Идентификатором является любая конечная последовательность букв и цифр, начинающаяся с буквы (пример 2.16). Имя может содержать знак подчеркивания «_». Использовать служебные слова языка в качестве идентификатора запрещается в большинстве языков программирования.

Величины могут быть постоянными (константы) и переменными. **Переменная** может принимать некоторое значение в результате выполнения команды ввода или с помощью оператора присваивания. В ходе выполнения программы значения переменной могут неоднократно меняться. После описания переменная отождествляется с некоторым блоком памяти, содержимое которого является ее значением. Переменная хранит значение, соответствующее ее типу (например, переменная целого типа не может принимать значение вещественного числа). Это значение может извлекаться из памяти для выполнения с ним операций, соответствующих типу переменной.

Пример 2.16. Имена переменных задает программист. Существуют рекомендации, как можно (нужно) именовать переменные в коде:

- имя переменной должно быть понятным, наглядным и отражать суть обозначаемого объекта (sum, number, count_of_positive);
- вводить переменным короткие имена (*s*, *i*, *n*) можно в том случае, когда они используются в небольшом фрагменте кода и их применение очевидно.

Некоторым идентификаторам заранее предписан определенный смысл, например sin, cos, sqrt, abs — имена математических функций.

Многие компании разрабатывают свои правила по оформлению кода, в которых прописаны также и правила именования переменных. В компании Microsoft используют так называемую «венгерскую нотацию»[1]. Известными являются также:

- «верблюжья нотация»;
- «змеиная нотация»;
- «шашлычная нотация»[2].

Правила оформления кода, разработанные в некоторых компаниях, являются открытыми и могут использоваться другими разработчиками. Например, в Google разработаны styleguide («гид по стилю») для разных языков программирования (C++, Java, Python, Lisp и др.)[3].

Подпрограммы

Алгоритм, реализующий решение отдельной части основной задачи, называют **вспомогательным**, а его запись на языке программирования — **подпрограммой**. Подпрограммы могут быть реализованы в виде функций или процедур.

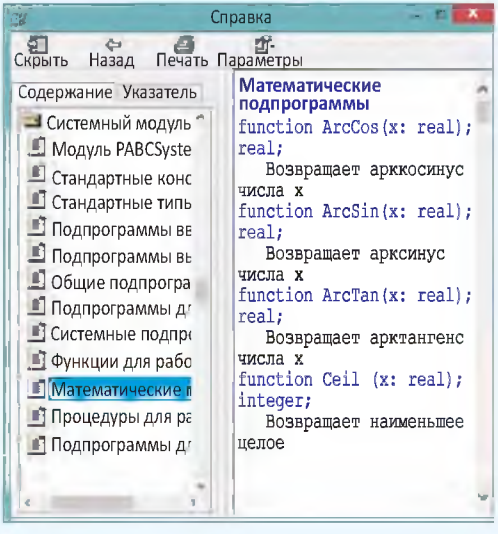
---

[1] https://ru.wikipedia.org/wiki/Венгерская_нотация#cite_note-hunganotat-1 (дата доступа: 25.02.2019).

[2] https://ru.hexlet.io/blog/posts/naming-in-programming (дата доступа: 25.02.2019).

[3] http://google.github.io/styleguide/ (дата доступа: 25.02.2019).

Пример 2.17. Список стандартных функций языка программирования PascalABC можно посмотреть в справочной системе:



Пример 2.18. Процедуры в языках программирования.

В языке VisualBasic процедуры объявляются как sub (сокращение от англ. *subroutine* — подпрограмма).

В языке C++ нет отдельных конструкций для описания процедур. Их роль выполняют функции, которые имеют тип *void* (англ. «пустота»).

На сайте Tiobe[1] ежемесячно публикуется рейтинг языков программирования.

Рейтинг составляется на основе подсчета результатов поисковых запросов, содержащих название языка, в Google, Blogger, Wikipedia, YouTube, Baidu, Yahoo!, Bing, Amazon.

Функция описывает процесс вычисления определенного значения, зависимого от некоторых аргументов, поэтому для функции всегда указывается тип возвращаемого значения. Для каждого языка высокого уровня разработана библиотека стандартных функций: арифметических, логических, символьных и т. д. (пример 2.17). Функции (как стандартные, так и задаваемые программистом) используются в программе в выражениях.

Часто используют подпрограммы, которые не возвращают конкретное значение, а представляют собой самостоятельный этап обработки данных. В языке Pascal их называют **процедурами**, в других языках они могут называться по-другому или не иметь собственного названия и описываться так же, как функции (пример 2.18).

Язык программирования — инструмент для решения конкретной задачи. Не существует единственного самого лучшего языка программирования. Для решения задач разного рода и уровня сложности требуется применять разные языки и технологии программирования. В простейших случаях достаточно освоить основы структурного написания программ, например на языке PascalABC. Для создания же сложных проектов требуется не только свободно владеть каким-то языком в полном объеме, но и иметь представление о других языках и их возможностях. Как правило, чем сложнее задача, тем больше времени требуется на освоение инструментов, необходимых для ее решения.

[1] https://www.tiobe.com/tiobe-index/ (дата доступа: 26.06.2019).

1. Для чего предназначен транслятор?
2. Какие функции выполняет компилятор? Интерпретатор?
3. Что определяется парадигмой программирования?
4. Из каких элементов может состоять алфавит языка программирования?
5. Что представляет собой оператор языка программирования?
6. Какие типы данных вам известны?
7. Для чего используются функции и процедуры?

Упражнения

1 Напишите программы для решения следующих задач.

1. Определите последнюю цифру натурального числа N.
2. Два отрезка на плоскости задаются координатами своих концов. Определите, какой из них короче.
3. Найдите сумму $1+\frac{1}{2^2}+ — + \ldots + —$ для заданного N.
4. Вводится строка текста. Определите, является ли она палиндромом.
5. Вводятся два целых числа, являющихся числителем и знаменателем дроби. Сократите дробь, выведите полученные числитель и знаменатель. Подсказка: можно воспользоваться алгоритмом Евклида.

6*. Дед Мазай и заяц играют в очень простую игру. Перед ними — гора из N одинаковых морковок. Каждый из игроков во время своего хода может взять из нее любое количество морковок, равное неотрицательной степени числа 2 (1, 2, 4, 8, ...). Игроки ходят по очереди. Кто возьмет последнюю морковку, тот и выигрывает. Составьте алгоритм, который при заданном значении N определяет победителя в этой игре. Учтите, что каждый из игроков хочет выиграть и не делает лишних ходов, т. е. играет оптимально.

2* Предложенные ниже алгоритмы записаны разными способами. Определите, что делает каждый из предложенных алгоритмов, и реализуйте их на языке Pascal.

1. Алгоритмический язык

```
ввод a
n := Длина(a)
m := 1
b := Извлечь(a, m)
нц для i от 7 до n
  c := Извлечь(a, i)
  b := Склеить(b, c)
кц
вывод b
```

Для слова «энергетика» программа выводит «этика».

2. Python

```
a = int(input())
k = 0
s = 1
while k < a:
  k = k + 1
  s = s + 1.0/k
print(s)
```

При $a = 5$ программа выводит 3.2833333333333337.

3. Basic

```
INPUT X
L = 0
M = 0
WHILE X > 0
   M = M + 1
    IF X MOD 3 <> 0 THEN
     L = L + 1
    END IF
   X = X \ 3
WEND
PRINT L
PRINT M
```

Для значения 5637 программа выводит 4 и 8.

4. C++

```
int F(int x)
{
   return x*x + 16*x + 15;
}
int main()
{
  int a, b;
  cin >> a >> b;
  int M = 0;
  for (int t = a; t <= b; t++)
    if (F(t) > 0)
      M = M + 1;
   cout << M;
   return 0;
}
```

При $a = -3$, $b = 5$ программа выводит 6.

3* Изобразите любой алгоритм из упражнения 2 в виде блок-схемы.

Глава 1
АЛГОРИТМЫ ОБРАБОТКИ МАССИВОВ

§ 3. Структурированный тип данных массив

Впервые тип данных *массив* появился в языке Фортран (создан в период с 1954 по 1957 г. в корпорации IBM). Уже первые версии языка поддерживали трехмерные массивы (в 1980 г. максимальная размерность массива была увеличена до 7). Массивы были необходимы для создания математических библиотек, в частности содержащих процедуры решения систем линейных уравнений.

Пример 3.1. В 10 А классе 25 учащихся. Известен рост каждого в сантиметрах. Для хранения значений роста можно использовать массив A, состоящий из 25 целых чисел. Индекс каждого элемента — порядковый номер учащегося из списка в классном журнале. Тогда запись A[5] — рост учащегося под пятым номером.

3.1. Понятие массива

В современном мире ежесекундно происходит обработка огромного числа данных с помощью компьютера. Если необходимо обрабатывать данные одного типа (числа, символы, строки и др.), то для их хранения можно воспользоваться типом данных, который называется **массив**.

Массив — упорядоченная последовательность данных, состоящая из конечного числа элементов, имеющих один и тот же тип, и обозначаемая одним именем.

Массив является структурированным (составным) типом данных. Это означает, что величина, описанная

как массив состоит из конечного числа других величин. Так, например, можно создать массивы из 10 целых или 100 вещественных чисел. Тип элементов массива называют **базовым типом**. Все элементы массива упорядочены по индексам, определяющим местоположение элемента в массиве.

Массиву присваивается имя, посредством указания которого можно ссылаться на данный массив как на единое целое. Элементы, образующие массив, упорядочены так, что каждому из них соответствует номер (индекс), определяющий место элемента в общей последовательности (примеры 3.1—3.3). Индексы представляют собой выражения любого простого типа, кроме вещественного. Доступ к каждому отдельному элементу осуществляется обращением к имени массива с указанием индекса нужного элемента, индекс элемента записывается после имени в квадратных скобках (пример 3.4).

Если обращение к элементам массива осуществляется при помощи только одного индекса, то такой массив называют **одномерным** или **линейным**. Элементы данного массива располагаются цепочкой друг за другом. Количество индексов, по которым обращаются к элементу в массиве, определяет **размерность массива**. Кроме одномерных, могут использоваться двумерные, трехмерные и другие массивы.

3.2. Описание массивов

Описание массива в языке Паскаль происходит следующим образом:

```
var <имя массива>: array [<тип индекса>] of <тип элементов>;
```

Пример 3.2. В 10 Б классе 27 учащихся. В классном журнале указаны фамилия и имя каждого из них. Для хранения списка учащихся можно использовать массив `S`, состоящий из 27 строк. Индекс каждого элемента — порядковый номер учащегося из списка в классном журнале. Тогда запись `S[5]` — фамилия и имя учащегося под номером 5.

Пример 3.3. Каждый день в декабре измеряли температуру воздуха. Для хранения значений температуры можно использовать массив `T`, состоящий из 31 вещественного числа. Индекс элемента — номер дня в декабре. Запись `T[15]` — температура воздуха 15 декабря.

Пример 3.4. Обращение к элементу массива: `a[3]`, `T[i]`, `S[n-1]`.

Двумерный массив — массив, элементами которого являются одномерные массивы. Его можно представить как таблицу с данными, в которой каждая строка — линейный массив. Обращение к элементу осуществляется по двум индексам: `a[3][5]` — элемент, находящийся в третьей строке и пятом столбце. Примером использования двумерного массива является лист электронной таблицы.

Массив `p`, описанный следующим образом:

```
var p: array [1..30] of array [1..30]
        of boolean;
```

можно использовать для определения свободных мест в зрительном зале. Тогда запись

```
p[2][13] := true;
```

будет означать, что во втором ряду место 13 свободно, а запись

```
p[5][7]  := false;
```

— в пятом ряду место 7 занято.

Пример 3.5. Опишем массив, рассмотренный в примере 3.1. Размер описанного массива — 25 элементов:

```
var a: array[1..25] of integer;
```

Пример 3.6. Опишем массив, рассмотренный в примере 3.2:

```
var S: array[1..27] of string;
```

Пример 3.7. Опишем массив, рассмотренный в примере 3.3. Размер описанного массива — 31 элемент.

```
var T: array[1..31] of real;
```

Пример 3.8*. Описать массив для хранения следующих данных: имеется здание склада, в котором есть два подземных этажа, цокольный и три верхних. Необходимо хранить количество пустых отсеков склада на каждом этаже. Размер описанного массива — 6 элементов:

```
const verh = 3;
      niz = -2;
var Sklad: array[niz..verh] of
                          integer;
```

Пример 3.9. Команда `b := a;` допустима для массивов `a` и `b`, описанных следующим образом:

```
var a, b: array[1..10] of integer;
```

Но команда `b := a;` выдаст ошибку, если массивы будут описаны так:

```
var a: array[1..10] of integer;
    b: array[1..10] of integer;
```

или так:

```
var a: array[1..10] of integer;
    b: array[1..15] of integer;
```

или так:

```
var a: array[1..10] of integer;
    b: array[1..10] of real;
```

Имя массива является идентификатором и задается по тем же правилам, что и имена любых других переменных.

Тип индекса определяет, как будут нумероваться элементы в массиве. Для задания типа индекса указывают номер первого элемента в массиве, затем ставят две точки, после которых указывают номер последнего элемента. Данные, которые используются для задания индексов, должны быть константами. Диапазон индексов определяет максимально возможное количество элементов в массиве — **размер массива**.

Тип элементов задает значение базового типа для данного массива. Базовый тип может быть любым из известных вам типов (примеры 3.5—3.8).

3.3. Операции над массивами

Массивы, описанные одинаково (в одной команде описания), можно использовать в операциях присваивания. В результате выполнения этой команды все элементы одного массива будут переписаны во второй (пример 3.9). Если массивы описаны одинаково, но в разных строках или описаны поразному, то при попытке присваивания возникнет ошибка о невозможности преобразовать типы.

Никакие другие операции для массива как для типа данных не определены.

Операции, выполняемые с элементами массива, соответствуют операциям, выполняемым над базовым типом. Если, например, описан массив из чисел типа integer, то с элемента-

ми такого массива можно выполнять такие же операции, как и с целыми числами. Элементы массива называются **индексированными переменными**. Они могут использоваться так же, как и простые переменные (пример 3.10).

3.4. Ввод и вывод элементов массива

Чтобы работать с массивом, необходимо задать начальные значения элементов массива. Сделать это можно несколькими способами:

1) ввод элементов массива с клавиатуры;

2) использование случайных чисел для определения значений;

3) использование функций (стандартных или собственных) для определения значений;

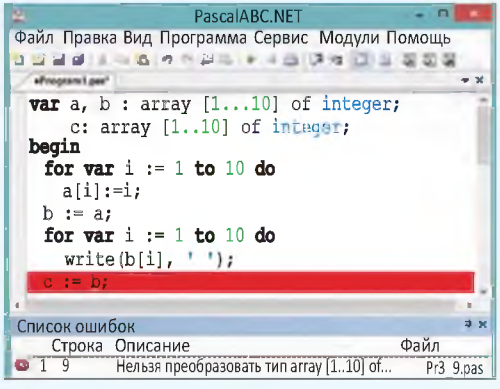
4) определение элементов массива как констант.

При вводе элементов массива с клавиатуры каждый элемент должен вводиться отдельно. Если количество вводимых элементов определено, то можно воспользоваться циклом **for** (пример 3.11).

При вводе элементов массива следует помнить, что количество вводимых элементов не может быть больше размера массива. В массив, описанный в примере 3.11, можно ввести любое количество чисел от 1 до 10, изменив значение 10 в заголовке цикла.

При описании массива размер определяет максимальное количество возможных элементов. При вводе можно определять количество элементов, которое необходимо для обработки в каждом конкретном случае (пример 3.12).

Пример 3.9. *Продолжение.*



Пример 3.10. Операции над индексированными переменными:

```
a[3] := 25 mod 7;
s := (t[1] + t[30])/2;
a[k] := b[k]*2;
Sum := Sum + a[i];
if a[i] < 0 then ...
```

Пример 3.11. Ввести 10 элементов массива a.

```
var a: array[1..10] of integer;
begin
  writeln('Введите 10 чисел
          через пробел');
  for var i := 1 to 10 do
    read(a[i]);
...
end.
```

Пример 3.12. Ввести заданное количество элементов массива a.

```
var a: array[1..100] of integer;
  n: integer;
begin
  writeln('Введите количество
          чисел в массиве');
  readln(n);
  writeln('Введите', n,
          'чисел через пробел');
  for var i := 1 to n do
    read(a[i]);
...
end.
```

Пример 3.13. Ввод массива строк.

```
var a: array[1..100] of string;
    n: integer;
begin
  writeln('Введите количество
          строк в массиве');
  readln(n);
  writeln('Введите', n,'строк,
         каждую с новой строки');
  for var i := 1 to n do
    readln(a[i]);
...
end.
```

Пример 3.14. Случайным образом задать n элементов массива a. Каждый элемент — число из отрезка [0; 100].

```
var a: array[1..100] of integer;
    n: integer;
begin
  writeln('Введите количество
          чисел в массиве');
  readln(n);
  for var i := 1 to n do
    a[i] := random(101);
...
end.
```

Пример 3.15. Описание массива, элементы которого являются числовыми константами.

```
const simple_numb: array[1..5]
      of integer = (2, 3, 5, 7, 11);
```

Пример 3.16. Описание массива, элементы которого являются строковыми константами.

```
const c_rgb: array of string =
     ('красный','синий','зеленый');
```

Пример 3.17. Вывод элементов массива в столбец (по одному в строке).

```
for var i := 1 to n do
  writeln(a[i]);
```

Пример 3.18. Вывод элементов массива в строку (через пробел).

```
for var i := 1 to n do
  write(a[i], ' ');
```

Если необходимо вводить массив из строк, нужно помнить, что каждый элемент вводится в отдельной строке с использованием команды readln (пример 3.13). Использовать пробел как разделитель не получится, поскольку пробел будет воспринят как очередной символ строки.

Иногда бывает удобно задавать элементы массива случайным образом. Для этого используется функция random(k), которая генерирует случайное целое число из промежутка [0; k) (пример 3.14).

Если элементы массива должны принадлежать отрезку [a; b], то можно использовать функцию random(a, b) или определить значение элемента массива так:

```
a[i]:= random(b-a+1)+ a;
```

Если элементы массива не будут изменяться при решении задачи, то массив может быть описан как константа (примеры 3.15, 3.16). При таком описании можно не указывать индексы элементов в массиве, тогда нумерация будет осуществляться от нуля до количества элементов в списке минус один.

Выводить элементы массива можно в столбец (пример 3.17) или в строку (пример 3.18). Если элементы массива выводятся в строку, то между ними нужно выводить символ-разделитель (чаще всего используют пробел), иначе все числа будут распечатаны подряд как одно число с большим количеством цифр. Выводить элементы массива можно не только

в прямом, но и в обратном порядке (пример 3.19).

3.5. Решение задач с использованием ввода-вывода массивов

Пример 3.20. Написать программу, которая введет элементы массива с клавиатуры и выведет сумму третьего и пятого элементов.

Этапы выполнения задания

I. Исходные данные: массив a и количество элементов n.

II. Результат: S — сумма третьего и пятого элементов.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Вычисление суммы.
3. Вывод результата.

IV. Описание переменных: a — **array**[1..10] **of** integer; n, S — integer.

Пример 3.21. Написать программу, которая сформирует массив из n чисел из отрезка [0; 100] случайным образом. Вывести массив на экран.

Этапы выполнения задания

I. Исходные данные: массив a и количество элементов n.

II. Результат: полученный массив.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Генерация массива.
3. Вывод результата.

IV. Описание переменных: a — **array**[1..100] **of** integer; n — integer.

Пример 3.19. Вывод элементов массива в строку (в обратном порядке).

```
for var i := n downto 1 do
  write(a[i], ' ');
```

Пример 3.20.

V. Программа:

```
var a: array[1..10] of integer;
   n, S: integer;
begin
  writeln('Введите количество
          чисел в массиве >=5');
  readln(n);
  writeln('Введите ', n,
          'чисел через пробел');
  for var i := 1 to n do
    read(a[i]);
  S := a[3] + a[5];
  write('Сумма чисел = ', S);
end.
```

VI. Тестирование:

Окно вывода

```
Введите количество чисел в массиве
7
Введите 7 чисел через пробел
12 3 4 2 1 19 7
Сумма чисел = 5
```

VII. Анализ результатов. Третий элемент массива равен 4, пятый элемент равен 1, сумма элементов равна 5.

Пример 3.21.

V. Программа:

```
var a: array[1..100] of integer;
    n: integer;
begin
writeln('Введите количество
         чисел в массиве');
  readln(n);
  for var i := 1 to n do
    a[i] := random(101);
  for var i := 1 to n do
    write(a[i],' ');
end.
```

VI. Тестирование:

Окно вывода

```
Введите количество чисел в массиве
8
66 44 80 3 100 66 3 78
```

Пример 3.22.

V. Программа:

```
var a: array[1..100] of integer;
      n, k: integer;
begin
 writeln('Введите количество
          чисел в массиве');
 readln(n);
 for var i:=1 to n do
 begin
  a[i]:=2*random(10, 35);
  write(a[i], ' ');
 end;
 writeln;
 writeln('Введите k');
 readln(k);
 write(a[k]);
end.
```

VI. Тестирование:

```
Окно вывода
Введите количество чисел в массиве
7
50 56 66 60 32 24 66
Введите k
5
32
```

Пример 3.23.

V. Программа:

```
var s: array [1..20] of string;
   n, k1, k2: integer;
begin
 writeln('Количество учащихся ');
 readln(n);
 writeln('Фамилии');
 for var i := 1 to n do
  readln(s[i]);
 writeln('k1 и k2');
 readln(k1, k2);
 for var i := k1 to k2 do
  writeln(s[i]);
end.
```

Пример 3.22. Написать программу, которая сформирует массив из n четных чисел из отрезка [20; 70] случайным образом. Вывести на экран k-й элемент массива.

Этапы выполнения задания

I. Исходные данные: массив a и количество элементов n.

II. Результат: искомый элемент.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Генерация массива.

1.1. Чтобы элементы массива были только четными, необходимо каждый полученный элемент умножать на 2.

1.2. Поскольку элементы умножаются на два, границы исходного отрезка нужно уменьшить в два раза.

1.3. Вывод массива по элементам.

3. Ввод значения k и вывод результата.

IV. Описание переменных: a — **array**[1..100] **of** integer; n, k — integer.

Пример 3.23. Написать программу, которая введет с клавиатуры список фамилий учащихся и выведет из него фамилии с номерами от k1 до k2.

Этапы выполнения задания

I. Исходные данные: массив s и количество учащихся n, номера фамилий — k1 и k2.

II. Результат: список заданных фамилий.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Вывод результата.

IV. Описание переменных: s — **array**[1..20] **of** string; n, k1, k2 — integer.

Пример 3.24. Задать случайным образом два массива X и Y, содержащих по n чисел из отрезка [100; 300], и массив R, содержащий n чисел из отрезка [5; 100]. Построить на экране окружности, координаты центров которых хранятся в массивах X и Y, а радиусы в массиве R.

Этапы выполнения задания

I. Исходные данные: массивы X, Y, R и количество элементов n.

II. Результат: рисунок n окружностей.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Генерация массивов.
3. Установка прозрачного стиля заливки для того, чтобы изображались окружности, а не круги.
4. Вывод результата.

IV. Описание переменных: X, Y, R — **array**[1..100] **of** integer; n — integer.

Все рассмотренные выше способы ввода и вывода массивов универсальны и могут использоваться для разных компиляторов языка Pascal. В среде PascalABC.Net дополнительно реализованы команды print и println, с помощью которых массив можно вывести без использования команды цикла. Команда print(b); выведет элементы массива b в квадратных скобках через запятую: [1,2,3,4,5,6].

Команда println после вывода массива дополнительно переводит курсор на новую строку. Элементы выводятся так же, как и при использовании команды print.

Пример 3.23. *Продолжение.*

VI. Тестирование:

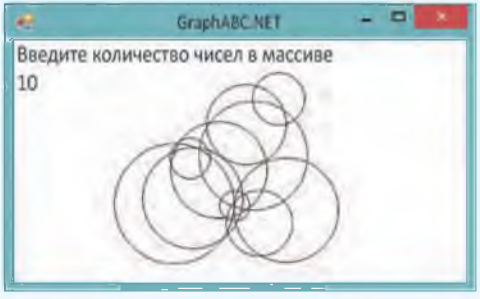
```
Окно вывода
Количество учащихся
6
Фамилии
Белов
Иванов
Королев
Петров
Сидоров
Яшкин
k1 и k2
3
5
Королев
Петров
Сидоров
```

Пример 3.24.

V. Программа:

```
uses graphABC;
var X, Y, R: array[1..100]
      of integer;
  n: integer;
begin
 SetWindowSize(400,400);
 writeln('Введите количество
          чисел в массиве');
 readln(n);
 writeln(n);
 for var i := 1 to n do
 begin
  X[i]:= random(100,300);
  Y[i]:= random(100,300);
  R[i]:= random(5,100);
 end;
 SetBrushStyle(bsClear);
 for var i := 1 to n do
   circle(X[i],Y[i],R[i])
end.
```

VI. Тестирование.



1. Что такое массив?
2. Как описываются массивы?
3. Что такое размер массива?
4. Какие операции допустимы для массивов?
5. Какие способы задания значений элементам массива вы знаете?
6. Как можно вывести массив?

Упражнения

1 Используя примеры 3.14—3.18, выполните следующие задания.

1. Введите 5 чисел и выведите их в одной строке.
2. Введите 7 чисел и выведите их в одной строке в обратном порядке.
3. Задайте 10 случайных чисел и выведите их по одному в строке.
4. Выведите на экран элементы массива, заданного в примере 3.16.

2 Измените программу из примера 3.19 так, чтобы выводилось произведение первых трех элементов.

3 Используя программы из примера 3.19 или 3.20, задайте массив из n случайных чисел из отрезка [–10; 10]. Выведите: первый элемент; последний элемент; элемент, стоящий на среднем месте.

4 Введите массив из n строк с клавиатуры. Выведите элементы массива в обратном порядке.

5 Для массива, описанного в примере 3.2, введите данные с клавиатуры. Задайте номер учащегося. Выведите его фамилию.

6* Введите рост учащихся своего класса, организовав ввод следующим образом:

```
Введите количество учащихся в классе: 15
Вводите рост учащихся
учащийся номер 1: 165
учащийся номер 2: 170
учащийся номер 3: 156
```

7* Для массива, описанного в примере 3.3, задайте значения случайными вещественными числами из интервала (–20; 10). Выведите значения температур для указанного диапазона дат. Пример вывода для диапазона дат от 1 декабря до 8 декабря:

```
1 декабря температура была = 9.4
2 декабря температура была = -11.8
3 декабря температура была = -16.6
4 декабря температура была = 8
5 декабря температура была = 0.9
6 декабря температура была = -9.3
7 декабря температура была = -11.5
8 декабря температура была = 6.6
```

8 Измените программу из примера 3.24 так, чтобы окружности рисовались разными цветами.

§ 4. Выполнение арифметических действий над элементами массива

4.1. Вычисление сумм и произведений элементов массива

Операции, выполняемые с элементами массива, соответствуют операциям, которые выполняются над базовым типом элементов массива (пример 4.1).

Пример 4.2. Задан одномерный массив из целых чисел. Найти сумму и произведение элементов этого массива.

I. Исходные данные: массив a и количество элементов n.

II. Результат: S — сумма элементов и P — произведение элементов массива.

III. Алгоритм решения задачи.

1. Ввод исходных данных. Массив вводится поэлементно с клавиатуры.
2. Определение начального значения для суммы (S := 0) и для произведения (P := 1).
3. В цикле добавляем очередной элемент массива к сумме и к произведению.
4. Вывод результата.

IV. Описание переменных: a — **array**[1..10] **of** integer; n, S, P — integer.

Пример 4.3. Известны отметки по информатике всех учащихся 10 Б класса за первую четверть. Успеваемость в классе будем считать хорошей, если средний балл больше 7, плохой, если средний балл ниже 4, в остальных случаях — успеваемость средняя. Определить успеваемость класса по заданным отметкам.

I. Исходные данные: массив a для хранения отметок и количество учащихся n.

Пример 4.1.

Если базовым типом элементов массива является тип integer, то для элементов массива допустимы следующие операции: +, –, *, **div**, **mod**.

Если в массиве хранятся числа типа real, то допустимыми будут операции +, –, *, /.

Если в массиве хранятся строки, то для каждого его элемента допустимы строковые функции и процедуры.

Пример 4.2.

V. Программа:

```
var a: array[1..10] of integer;
  n, S, P: integer;
begin
 write('Введите n =');
 readln(n);
 writeln('Вводите элементы');
 for var i := 1 to n do
  read(a[i]);
 S := 0;
 P := 1;
 for var i := 1 to n do
 begin
  S := S + a[i];
  P := P * a[i];
 end;
 writeln('Сумма = ', S);
 writeln('Произведение = ', P);
end.
```

VI. Тестирование.

```
Окно вывода
Введите n = 5
Вводите элементы
3 2 44 -1 3
Сумма = 51
Произведение = -792
```

VII. Анализ результатов. Проверить правильность вычислений можно на калькуляторе.

Пример 4.3.

V. Программа:

```
var a: array[1..30] of integer;
 n, S: integer; Sr: real;
begin
 write('Количество учащихся ');
 readln(n);
 writeln('Вводите отметки');
 for var i := 1 to n do
  read(a[i]);
 S := 0;
 for var i := 1 to n do
  S := S + a[i];
 Sr := S / n;
 if Sr > 7 then
  writeln('Хорошая')
 else
  if Sr < 4 then
   writeln('Плохая')
  else
   writeln('Средняя');
end.
```

VI. Тестирование.

Окно вывода

```
Количество учащихся 5
Вводите отметки
10 5 6 8 9
Хорошая
```

Пример 4.4.

V. Программа:

```
var Kol, Cen: array[1..50] of
    integer;
   n, Sum: integer;
begin
 write('Введите количество
       видов товаров ');
 readln(n);
 for var i := 1 to n do
 begin
  writeln('Введите количество
      товара', i, ' и его цену ');
  read(Kol[i], Cen[i]);
 end;
 Sum := 0;
 for var i := 1 to n do
  Sum := Sum + Kol[i]*Cen[i];
 writeln('Суммарная стоимость
          товаров =', Sum);
end.
```

II. Результат: одно из слов — «хорошая», «средняя», «плохая» в зависимости от значения среднего балла.

III. Алгоритм решения задачи.

1. Ввод исходных данных. Сначала вводим количество учащихся в классе, затем массив отметок (поэлементно с клавиатуры).

2. Для определения успеваемости нужно вычислить средний балл (переменная Sr). Средний балл определяется как сумма (переменная S) всех отметок, деленная на количество учащихся в классе. Начальное значение для суммы — S := 0.

3. В цикле добавляем очередной элемент массива к сумме.

4. Делим полученную сумму на количество учащихся в классе.

5. Проверяем значение среднего балла и выводим результат.

IV. Описание переменных: a — **array**[1..30] **of** integer; n, S — integer; Sr — real.

4.2. Вычисление сумм и произведений при работе с двумя массивами

Пример 4.4. На складе хранятся товары. Для каждого вида товара известно количество единиц товара и цена за единицу товара. Определить суммарную стоимость всех товаров, хранящихся на складе.

I. Исходные данные: Cen — одномерный массив для хранения цены единицы товара каждого вида, Kol — массив для хранения количества товара каждого вида, n — количество видов товаров.

II. Результат: Sum — значение суммарной стоимости товаров на складе.

III. Алгоритм решения задачи.

1. Ввод исходных данных. Для каждого вида товара задается его цена и количество.

2. Стоимость всех товаров одного вида определяется как произведение количества на цену. Суммарная стоимость — сумма всех таких произведений. Начальное значение суммы `Sum := 0`; В цикле к сумме прибавляются произведения `Kol[i]*Cen[i]`.

3. Вывод результата.

IV. Описание переменных: `Cen`, `Kol` — **array**[1..50] **of** integer; `n`, `Sum` — integer.

4.3*. Использование массива, элементы которого являются константами

Пример 4.5. Задано натуральное число `n` (`n < 5000`). Определить, является ли это число простым.

I. Исходные данные: `n` — натуральное число.

II. Результат: вывод сообщения «простое» или «составное».

III. Алгоритм решения задачи.

1. Ввод исходных данных.

2. Известно, что число `n` является простым, если оно не делится ни на одно простое число, не большее $\sqrt{n}$. Максимальное число по условию — 5000, $\sqrt{} \approx$. Создадим массив констант `s_n` из простых чисел, не больших 71.

3. В цикле будем делить число `n` на каждое из чисел, не больших $\sqrt{n}$

Пример 4.4. *Продолжение.*

VI. Тестирование.

Окно вывода

```
Введите количество видов товаров 3
Введите количество товара 1 и его цену
5 2
Введите количество товара 2 и его цену
7 3
Введите количество товара 3 и его цену
4 5
Суммарная стоимость товаров =51
```

Пример 4.5.

Условие `s_n[i] <= sqrt(n)` проверяется долго за счет вызова функции `sqrt(n)`. Это условие обычно заменяют эквивалентным: `s_n[i]*s_n[i] <= n`.

V. Программа:

```
const s_n: array of integer =
(2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
1, 37, 41, 43, 47, 53, 59, 61, 67, 71);
var n,i: integer;
begin
 writeln('Введите число');
 read(n);
 i := 0;
 while (s_n[i] * s_n[i] <= n)
      and (n mod s_n[i] <> 0) do
  i := i + 1;
 if s_n[i]*s_n[i] > n then
  writeln('Простое')
 else
  writeln('Составное')
end.
```

VI. Тестирование.

Окно вывода

```
Введите число
2027
Простое
```

Окно вывода

```
Введите число
2021
Составное
```

VII. Анализ результатов. Проверить правильность вычислений можно на калькуляторе или посмотреть в таблице простых чисел[1].

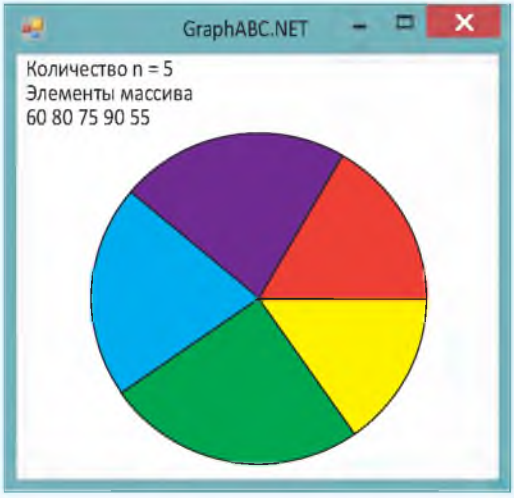
[1] Технические таблицы: http://tehtab.ru/guide/guidemathematics/guidemathematicsfigurestables/simplefigures/ (дата доступа: 10.02.2019).

Пример 4.6.

V. Программа:

```
uses graphABC;
var a: array[1..10] of integer;
    n, S, u0, u1: integer;
begin
 write('Количество n =');
 readln(n);
 writeln(n);
 writeln('Элементы массива');
 for var i := 1 to n do
  read(a[i]);
 for var i := 1 to n do
  write(a[i], ' ');
 S := 0;
 for var i := 1 to n do
  S := S + a[i];
 u0 := 0;
 for var i := 1 to n do
 begin
  u1:= u0+trunc(a[i]*360/S);
  SetBrushColor(clRandom);
  Pie(150,150,100,u0,u1);
  u0 := u1;
 end;
end.
```

VI. Тестирование.



и хранящихся в массиве констант. Если число n не разделилось ни на одно из них, то оно — простое, иначе — составное.

4. Проверяем, с каким условием закончил работу цикл: число является простым, если последний просмотренный элемент массива больше $\sqrt{n}$ (число ни на что не разделилось).

5. Вывод результата.

IV. Описание переменных: s_n — const **array of** integer; n, i — integer.

4.4. Построение круговой диаграммы

Пример 4.6. Задан одномерный массив из целых чисел. Построить круговую диаграмму по числовым данным, хранящимся в массиве. Например, для 5 элементов массива — 60, 80, 75, 90, 55.

I. Исходные данные: массив a для хранения данных и n — количество данных.

II. Результат: круговая диаграмма.

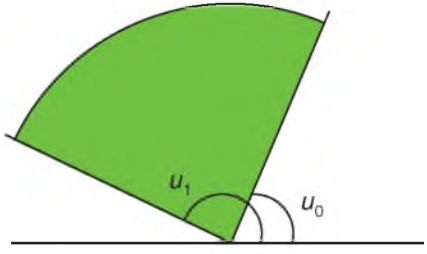
III. Алгоритм решения задачи.

1. Ввод исходных данных. Массив вводится поэлементно с клавиатуры.

2. Круговая диаграмма состоит из n секторов. Градусная мера сектора определяется числовым значением соответствующего элемента в массиве. Суммарное значение всех элементов массива (переменная S) соответствует величине в 360°. Тогда значению элемента массива A[i] будет соответствовать величина — ——.

3. Вычисляем сумму всех элементов массива.

4. В цикле строим секторы, градусная мера которых равна целой части величины —.



Для построения сектора нужно знать величины двух углов: u_0 и u_1. Значение = + —. Вначале u_0 = 0. Затем в цикле меняем значение u_0 на u_1. Цвет сектора будем задавать случайным образом. Для вычисления целой части можно использовать функции trunc и round.

IV. Описание переменных: a — **array**[1..10] **of** integer; n, S, u_0, u_1 — integer.

Пример 4.6. *Продолжение.*

VII. Постройте по этим данным диаграмму в Excel и сравните.

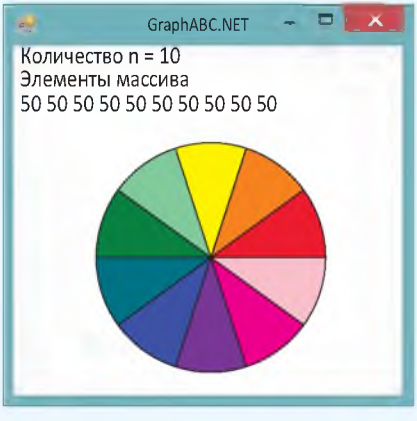
Для задания цвета сектора можно использовать массив, содержащий цветовые константы:

```
const d_color: array [1..10]
of Color = (clRed, clOrange,
clYellow, clLightGreen, clGreen,
clTeal, clBlue, clDarkViolet,
clMagenta, clPink);
```

Команду задания цвета сектора нужно будет заменить на:

```
SetBrushColor(d_color[i]);
```

Результат:



1. Какие операции допустимы для элементов массива целых чисел?
2. Какие операции допустимы для элементов массива вещественных чисел?
3. Как записать данные в массив констант?

Упражнения

1 Для задачи из примера 4.2 выполните перечисленные задания.

1. Заполните таблицу.

	n	a	S	P
1	3	−2 −3 −5		
2	5	1 2 3 4 5		
3	10	1 −3 −2 3 4 3 2 4 3 2		

2. Добавьте в таблицу свои значения n и a.

3. Попытайтесь подобрать такие значения элементов массива, чтобы S = P, для n = 2,5.

4. Для n = 10 ввели все элементы массива, равные 9. Какой результат получили? Почему? Что нужно исправить в программе для получения правильного результата?

2 Для задачи из примера 4.3 добавьте вывод среднего балла.

3 В ходе хоккейного матча удалялись игроки обеих команд. Для каждого удаленного игрока известно время его отсутствия на поле. Определите, какая из команд провела больше времени на скамейке штрафников.

4 Для задачи из примера 4.5 выполните следующее задание:

Введите число 5557. Почему появилась ошибка? Дополните массив констант простыми числами так, чтобы программа могла выдавать ответ для чисел, меньших 10 000. (Для этого можно воспользоваться самой программой или таблицей простых чисел.)

5 Для задачи из примера 4.6 выполните перечисленные задания.

1. Внесите в программу изменения так, чтобы цвет сектора выбирался из массива констант.

2*. Измените программу так, чтобы диаграмма всегда строилась в центре графического окна. Диаметр круга определяется меньшей из двух величин — шириной или высотой окна.

6* В массивах X и Y хранятся координаты точек. Постройте многоугольник, заданный этими координатами. Запросите у пользователя номера двух точек и постройте диагональ многоугольника, соединяющую эти точки.

§ 5. Поиск элементов с заданными свойствами

Человек постоянно сталкивается с задачами поиска требуемой информации. Типичным примером может служить работа со справочниками или библиотечной картотекой. В современном мире информацию ищут с использованием сети Интернет.

Чтобы поиск был результативным и быстрым, разрабатывают эффективные алгоритмы поиска. Важную роль в процессе поиска информации играет способ хранения данных. Одной из самых простых структур для этого является массив.

5.1. Линейный поиск

Рассмотрим, как осуществляется поиск для данных, хранящихся в массиве.

Среди разновидностей простейших задач поиска, встречающихся на практике, можно выделить следующие типы:

1. Найти хотя бы один элемент, равный заданному элементу X. В результате необходимо получить i —

индекс (номер) элемента массива, такой, что a[i] = X.

2. Найти все элементы, равные заданному X. В результате необходимо получить количество таких элементов и (или) их индексы.

Иногда поиск организуется не по совпадению с элементом X, а по выполнению некоторых условий. Примером может служить поиск элементов, удовлетворяющих условию: $X_1 \leqslant a[i] \leqslant X_2$, где X_1 и X_2 заданы.

Если нет никакой добавочной информации о разыскиваемых данных, то самый простой подход — последовательный просмотр элементов массива.

Алгоритм, при котором для поиска нужного элемента последовательно просматривают все элементы массива в порядке их записи, называется **линейным** или **последовательным поиском**.

5.2. Поиск одного элемента, удовлетворяющего условию поиска

Пример 5.1. Задан одномерный массив из n чисел. Определить, есть ли в нем хотя бы один элемент, равный x (значение x вводится).

I. Исходные данные: массив a, количество чисел n, искомое число x.

II. Результат: вывод сообщения «Элемент найден» или «Элемент не найден».

III. Алгоритм решения задачи.

1. Ввод исходных данных.

2. Пусть p — переменная логического типа, которая имеет значение «истина», если элемент

Алгоритмы поиска можно разделить на алгоритмы, использующие неупорядоченные наборы данных, и на алгоритмы, работающие с предварительно упорядоченным набором данных.

Примером поиска в неупорядоченном наборе данных может служить поиск тетради конкретного учащегося в стопке тетрадей, сданных на проверку. Чтобы найти нужную тетрадь, возможно, придется пересмотреть все. Поиск в словаре — поиск в упорядоченном наборе данных, т. к. все слова расположены в алфавитном порядке.

Пример 5.1.

V. Программа:

```
var a: array[1..10] of integer;
  n, x: integer;
  p: boolean;
begin
  write('Количество n =');
  readln(n);
  writeln('Элементы массива');
  for var i := 1 to n do
    read(a[i]);
  write('Число x =');
  readln(x);
  //линейный поиск элемента
  p := false;
  for var i := 1 to n do
    if a[i] = x then
      p := true;
  if p then
    writeln('Элемент найден')
  else
    writeln('Элемент не найден');
end.
```

VI. Тестирование.

Окно вывода	Окно вывода
Количество n = 5	Количество n = 5
Элементы массива	Элементы массива
1 2 3 4 5	1 3 5 7 9
Число x = 4	Число x = 6
Элемент найден	Элемент не найден

Пример 5.2.

V. Программа:

```
var a: array[1..10] of integer;
 n, x, k: integer;
begin
 write('Количество n=');
 readln(n);
 writeln('Элементы массива');
 for var i := 1 to n do
  read(a[i]);
 write('Число x =');
 readln(x);
 //линейный поиск элемента
 k := 0;
 for var i := 1 to n do
  if a[i] = x then
   k := i;
 if k = 0 then
  writeln('Элемент не найден')
 else
  writeln('Элемент найден
            на месте ', k);
end.
```

VI. Тестирование.

Окно вывода	Окно вывода
Количество n = 5 Элементы массива 1 3 3 3 5 Число x = 3 Элемент найден на месте 4	Количество n = 5 Элементы массива 1 3 5 7 9 Число x = 4 Элемент не найден

Пример 5.3.

Фрагмент программы:

```
//линейный поиск элемента
 k:=1;
 while (k<=n) and (a[k]<>x) do
  k:=k+1;
 if k = n+1 then
  writeln('Элемент не найден')
 else
  writeln('Элемент найден
            на месте ', k);.
```

в массиве найден, и «ложь» — в противном случае. До просмотра элементов массива p := false.

3. В цикле будем просматривать все числа в массиве и сравнивать их с числом x.

4. После окончания поиска возможна одна из двух ситуаций:

4.1. Искомый элемент найден (p := true), т. е. в массиве есть такой элемент a[i], что a[i] = x.

4.2. Весь массив просмотрен, и совпадений не обнаружено (p := false).

5. Вывод результата.

IV. Описание переменных: a — **array**[1..10] **of** integer; n, x — integer; p: boolean.

Часто требуется не только определить, есть ли в массиве искомый элемент, но и установить, на каком месте он находится.

Будем хранить индекс найденного элемента (пример 5.2) в переменной k. После выполнения данного алгоритма по значению переменной k можно определить, есть ли в массиве искомый элемент, и если есть, то где он стоит. Если в массиве несколько таких элементов, то в переменной k будет храниться номер последнего из них. Если такого элемента нет, то значение переменной k не изменится (k останется равным 0).

На практике операцию поиска приходится выполнять достаточно часто, и скорость работы программы находится в прямой зависимости от используемого алгоритма поиска.

В рассмотренных выше алгоритмах требуется просмотреть весь массив, даже в том случае, если искомый элемент находится в массиве на первом месте.

Для сокращения времени поиска можно останавливаться сразу после того, как элемент найден. В этом случае весь массив придется просмотреть только тогда, когда искомый элемент последний или его нет вообще (пример 5.3). Цикл заканчивает работу, когда будет найден искомый элемент либо когда k = n + 1, т. е. элемента, совпадающего с x, не существует.

*При такой реализации на каждой итерации цикла требуется увеличивать индекс k и вычислять логическое выражение. Ускорить поиск можно, упростив логическое выражение. Поместим в конец массива дополнительный элемент со значением x. Тогда совпадение с x обязательно произойдет, и можно не проверять условие a[k] <> x. Такой вспомогательный элемент часто называют **«барьером»** или **«часовым»**, так как он препятствует выходу за пределы массива. В исходном массиве теперь будет n + 1 элемент (пример 5.4).

5.3. Нахождение всех элементов, удовлетворяющих условию поиска

Если требуется определить количество элементов, удовлетворяющих какому-либо условию, то для этого определяют отдельную переменную, значение которой увеличивают на 1 каждый раз, когда найден нужный элемент. Такую переменную называют **счетчиком**. До начала просмотра

Пример 5.4*.
Фрагмент программы:

```
//линейный поиск с барьером
 a[n + 1] := x;
 k := 1;
 while a[k]<>x do
  k := k + 1;
 if k = n + 1 then
  writeln('Элемент не найден')
 else
  writeln('Элемент найден
          на месте ', k);
```

V. Тестирование.

Окно вывода
Количество n = 5
Элементы массива
1 5 6 7 1
Число x = 7
Элемент найден
на месте 4

Окно вывода
Количество n = 5
Элементы массива
1 2 3 4 5
Число x = -2
Элемент не найден

В современных языках программирования используются библиотеки, содержащие функции для поиска элементов в массивах (и других структурах данных). В PascalABC.Net такие функции реализованы только для динамических массивов (размер массива может изменяться во время выполнения программы, элементы нумеруются с нуля). Описание функций можно найти в справочнике в разделе «Методы расширения одномерных динамических массивов». Пример использования функции поиска всех элементов массива, больших 5:

```
var c : array of integer;
begin
 setlength(c, 10);
 for var i := 0 to 9 do
  c[i] := random(1, 10);
 println(c);
 c.FindAll(p -> p > 5). Println;
end.
```

Пример 5.5.

V. Программа:

```
var a: array[1..10] of integer;
  n, x, k: integer;
begin
  write('Количество n =');
  readln(n);
  writeln('Элементы массива');
  for var i := 1 to n do
    read(a[i]);
  write('Число x =');
  readln(x); k := 0;
  for var i := 1 to n do
    if a[i] mod x = 0 then
      k := k + 1;
  writeln('В массиве ', k, '
     элемент(-а,-ов), кратный(-х)', x);
end.
```

VI. Тестирование.

```
Окно вывода
Количество n = 5
Элементы массива
1 2 3 4 5
Число x = 2
В массиве 2 элемент(-а,-ов),
кратный(-х) 2
```

Пример 5.6.

V. Программа:

```
var a, b: array[1..10] of integer;
  n, x, k: integer;
begin
  write('Количество n =');
  readln(n);
  writeln('Элементы массива');
  for var i := 1 to n do
    read(a[i]);
  write('Число x =');
  readln(x); k := 0;
  for var i := 1 to n do
    if a[i] mod x = 0 then
    begin
      k := k + 1; b[k] := i;
    end;
  writeln('В массиве ', k, '
   элемент(-а,-ов), кратный(-х) ', x);
  writeln('Местоположение ');
  for var i := 1 to k do
    write(b[i],' ');
end.
```

элементов массива счетчику нужно задать начальное значение, или, другими словами, **инициализировать** значение переменной. В случае подсчета количества элементов, удовлетворяющих условию, счетчик инициализируется нулем. Для решения задачи нужно просматривать весь массив.

Пример 5.5. Задан одномерный массив из n чисел. Определить количество элементов, кратных x в линейном массиве.

I. Исходные данные: массив a, количество чисел n, искомое число x.

II. Результат: количество элементов, удовлетворяющих условию, — k.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Инициализация счетчика.
3. В цикле будем просматривать все числа в массиве и сравнивать с нулем их остатки от деления на число x. Если остаток равен нулю, то счетчик увеличиваем на 1.
4. Вывод результата.

IV. Описание переменных: a — **array**[1..10] **of** integer; n, x, k — integer.

Если необходимо не только посчитать, сколько элементов удовлетворяют условию, но и сохранить индексы таких элементов, то для этого можно воспользоваться дополнительным массивом. Создадим новый массив b. Как только будет найден необходимый элемент, его индекс будет заноситься в массив b. Переменная k будет хранить номер последнего занятого места в массиве b. Вначале k = 0 (пример 5.6).

После завершения работы первые k элементов массива b будут содержать индексы искомых элементов.

Если для решения задачи потребуются значения всех найденных элементов, то в программе возможно такое обращение к элементам массива: a[b[i]]. Адрес элемента в массиве a будет определяться значением элемента массива b по адресу i. Для вывода значений элементов в примере 5.6 последний цикл нужно заменить на

```
for var i := 1 to k do
  write(a[b[i]],' ');
```

5.4. Решение задач с использованием алгоритма линейного поиска

Пример 5.7. Известны результаты ЦТ по математике для n человек. Определить, есть ли среди них хотя бы один человек с баллом выше x. Значение x вводится с клавиатуры. Результаты экзамена получить случайным образом.

I. Исходные данные: массив a, количество чисел n, число x.

II. Результат: сообщение соответствует условию задачи.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Просмотр элементов с начала. Как только элемент найден, остановимся.
3. Если весь массив просмотрен, значит, в исходном массиве нет элемента, удовлетворяющего условию задачи, иначе выводим номер найденного элемента.

IV. Описание переменных: a — array[1..20] of integer; n, x, k — integer.

Пример 5.6. *Продолжение.*

VI. Тестирование.

Окно вывода

```
Количество n = 5
Элементы массива
6 3 2 4 5
Число x = 2
В массиве 3 элемент(-а,-ов),
кратный(-х) 2
Местоположение
1 3 4
```

Пример 5.7.

V. Программа:

```
var a: array[1..20] of integer;
 n, x, k: integer;
begin
 write('Количество n =');
 readln(n);
 writeln('Элементы массива');
 for var i := 1 to n do
 begin
  a[i] := random(0,100);
  write(a[i],' ');
 end;
 writeln;
 write('Число x =');
 readln(x);
 //линейный поиск с барьером
 a[n+1] := x + 1;
 k := 1;
 while (a[k] <= x) do
  k := k + 1;
 if k = n+1 then
  writeln('Нет таких')
 else
  writeln('Это человек с № ',
          k, ', его балл - ', a[k]);
end.
```

VI. Тестирование.

Окно вывода

```
Количество n = 10
Элементы массива
48 12 96 48 9 95 5 71 77 24
Число x = 80
Это человек с №3, его балл - 96
```

Пример 5.8.
V. Программа:

```
uses graphABC;
var X,Y: array [1..1000] of
     integer;
   n, k1, k2, R: integer;
begin
  write('Количество точек n =');
  read(n);
  writeln(n);
  for var i := 1 to n do
  begin
   X[i]:= random(-200,200);
   Y[i]:= random(-200,200);
  end;
  writeln('Радиус окружности');
  read(R);
  writeln(R);
  {Построение окружности и
  осей координат}
  circle(200,200,R);
  line(0,200,400,200);
  line(200,0,200,400);
  k1 := 0; k2 := 0;
  for var i := 1 to n do
   if X[i]*X[i]+Y[i]*Y[i]<=R*R then
   begin
    k1 := k1 + 1;
    SetPixel(X[i]+200,200-Y[i],
             clred);
  end
  else
  begin
   k2 := k2 + 1;
    SetPixel(X[i]+200,200-Y[i],
             clblue);
  end;
  if k1 > k2 then
   writeln('Внутри больше')
  else
  if k1<k2 then
    writeln('Снаружи больше')
   else
    writeln('Поровну')
end.
```

Пример 5.8. В двух линейных массивах x и y, заданных случайным образом, хранятся координаты точек плоскости (–200 ⩽ X[i], Y[i] ⩽ 200). Определить, каких точек больше — лежащих внутри или снаружи области, ограниченной окружностью радиуса R с центром в начале координат (будем считать, что точки, лежащие на окружности, лежат внутри области). Построить окружность и точки. Точки, принадлежащие внутренней области, нарисовать красным цветом, а внешней области — синим цветом.

I. Исходные данные: X, Y — массивы чисел, R — радиус окружности, n — количество точек.

II. Результат: рисунок, соответствующий условию задачи, и сообщение: «Внутри точек больше», «Снаружи точек больше» или «Точек поровну».

III. Алгоритм решения задачи.

1. Ввод исходных данных.

2. Инициализация счетчиков: k1:= 0; k2:= 0.

3. Будем просматривать все точки и для каждой проверять принадлежность области. Если $X^2 + Y^2 \leqslant R^2$, то точка лежит внутри области, тогда увеличим значение счетчика k1 на 1, если нет, то увеличим на 1 значение счетчика k2.

4. Сравним значения k1 и k2 и выведем результат.

5. Поскольку координаты точек принадлежат отрезку [–200, 200], то оси координат можно нарисовать пересекающимися в точке с координатами (200, 200). Для преобразования координат точек в экранные нужно к значению аб-

сциссы прибавить 200, а значение ординаты нужно отнять от 200 (ось Y на экране направлена вниз, поэтому нужно поменять знак ординаты). Строить точки можно в том же цикле, в котором происходит проверка.

IV. Описание переменных: X, Y — **array**[1..1000] **of** integer; n, k1, k2, R — integer.

Пример 5.9. На складе хранятся пустые ящики для упаковки товара. Известно, что масса одного пакета с конфетами x кг. Какова суммарная масса пакетов с конфетами, которые можно упаковать в такие ящики, заполнив ящик целиком?

I. Исходные данные: массив a, количество чисел n, число x.

II. Результат: количество ящиков — k, суммарная масса — S.

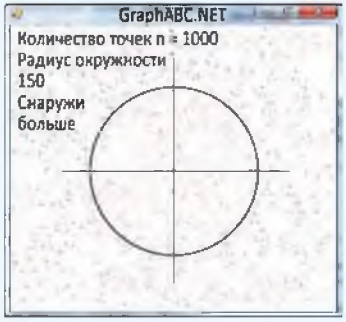
III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Инициализация счетчика и значения суммы: k := 0; S := 0.
3. Просматривая массив, проверим, является ли текущий элемент числом, кратным x (в этом случае ящик будет заполнен целиком). Как только элемент найден, увеличим счетчик k на 1, а переменную S — на значение найденного элемента массива.
4. Вывод результата.

IV. Описание переменных: a — **array**[1..20] **of** integer; n, x, k, S — integer.

Пример 5.10. Имеется список мальчиков 10 В класса и результаты их бега на 100 м. Для сдачи норматива

Пример 5.8. *Продолжение.*
VI. Тестирование.



Пример 5.9.
V. Программа:

```
var a: array [1..20] of integer;
    n, x, k, S: integer;
begin
 write('Количество ящиков:');
 readln(n);
 writeln('Вместимость ящиков');
 for var i := 1 to n do
  read(a[i]);
 write('Масса конфет:'); read(x);
 k := 0; S := 0;
 for var i := 1 to n do
  if a[i] mod x = 0 then
  begin
      k := k + 1;
      S := S + a[i];
   end;
 writeln('На складе',k,'ящ.');
 writeln('Суммарная масса', S);
end.
```

VI. Тестирование.

```
Окно вывода
Количество ящиков: 8
Вместимость ящиков
15 23 64 27 35 10 48 13
Масса конфет: 5
На складе 3 ящ.
Суммарная масса 60
```

VII. Анализ результатов. Ящики, удовлетворяющие условию задачи, имеют вес — 15, 25 и 10.

Пример 5.10.

V. Программа:

```
var r: array [1..20] of real;
  fam: array [1..20] of string;
  n, k: integer;
begin
 writeln('Количество учащихся: ');
  readln(n);
 writeln('Фамилия и результат: ');
  for var i := 1 to n do
  begin
    readln(fam[i]);
    readln(r[i]);
  end;
  writeln('Фамилии не сдавших
            норматив:');
  k := 0;
  for var i := 1 to n do
   if r[i] > 16 then
   begin
     k := k + 1;
     writeln(fam[i]);
   end;
 writeln('Не сдали норматив:', k);
end.
```

VI. Тестирование.

Окно вывода

```
Количество учащихся:
5
Фамилия и результат:
Иванов
13.5
Петров
14.0
Сидоров
21
Королев
16.1
Веремей
15.9
Фамилии не сдавших норматив:
Сидоров
Королев
Не сдали норматив: 2
```

VII. Анализ результатов. Результат Сидорова 21, а Королева 16.1, что превышает норматив.

необходимо пробежать дистанцию не более чем за 16 с. Вывести фамилии учащихся, которые не выполнили норматив по бегу. Сколько таких учащихся в классе?

I. Исходные данные: массивы fam (фамилии учащихся) и r (результаты бега в секундах), количество учащихся n.

II. Результат: фамилии тех учащихся, которые не выполнили норматив по бегу.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Инициализация счетчика: k := 0.
3. Будем просматривать массив с результатами и проверять, является ли текущий элемент числом больше 16 (норматив не сдан). Если такое значение найдено, то выведем элемент массива fam с соответствующим номером и увеличим значение счетчика на 1.
4. Вывод значения счетчика.

IV. Описание переменных: fam — **array**[1..20] **of** string; r — **array**[1..20] **of** real, n, k — integer.

Пример 5.11*. Задан одномерный массив из N целых чисел. Определить количество элементов, которые являются числами Смита. (Число Смита — это такое составное число, сумма цифр которого равна сумме цифр всех его простых сомножителей.) Например, числом Смита является $202 = 2 \times 101$, поскольку $2+0+2=4$, и $2+1+0+1=4$.

I. Исходные данные: a — массив чисел, n — количество чисел в массиве.

II. Результат: числа Смита и их количество в массиве.

III. Алгоритм решения задачи.

1. Ввод исходных данных.

2. Инициализация счетчика: k := 0.

3. Будем просматривать каждый элемент массива и определять, является ли он числом Смита. Для проверки создадим функцию check, которая будет получать в качестве параметра элемент массива, а также возвращать значение true, если число является числом Смита, и false в противном случае.

3.1. Найдем сумму цифр числа.

3.2. Будем раскладывать число на простые множители и для каждого множителя находить сумму цифр.

3.3. Для разложения числа на простые множители будем делить его сначала на 2 (пока делится), затем на 3. На 4 число уже делиться не будет, будем делить его на 5 и т. д. Закончится разложение тогда, когда после всех делений число станет равным 1.

4. Также нам понадобится функция sum, которая для числа будет возвращать его сумму цифр.

IV. Описание переменных: a — **array**[1..100] **of** integer; n, k — integer.

Пример 5.12*. Задан одномерный массив из n строк. Каждая строка является предложением из слов, разделенных пробелами. Найти и вывести те предложения, в которых нечетное количество слов.

Пример 5.11*.

V. Программа:

```
var a: array [1..100] of
   integer;
   n, k: integer;

function sum(x: integer):
          integer;
var s: integer;
begin
  s := 0;
  while x > 0 do
  begin
    s := s + x mod 10; x := x div 10;
  end;
  sum := s;
end;

function check(x: integer):
          boolean;
var s1, s2, d: integer;
begin
  s1 := sum(x); s2 := 0; d := 2;
//разложение на простые множители
  while x <> 1 do
  begin
    while x mod d = 0 do
    begin
      s2 := s2 + sum(d);
      x := x div d;
    end;
    d := d + 1;
  end;
  check := s1 = s2;
end;

begin
  writeln('Количество');
  readln(n);
  writeln('Элементы');
  for var i := 1 to n do
    read(a[i]);
  k := 0;
  writeln('Числа Смита');
  for var i := 1 to n do
    if check(a[i]) then
    begin
      inc(k);
      write(a[i], ' ');
    end;
  writeln;
  writeln('Всего - ', k);
end.
```

Пример 5.11*. *Продолжение.*
VI. Тестирование.

Окно вывода	Окно вывода
Количество 5 Элементы 202 3 323 85 117 Числа Смита 202 3 85 Всего - 3	Количество 5 Элементы 8 8 8 8 8 Числа Смита Всего - 0

Пример 5.12.
V. Программа:

```
var a: array [1..100] of
   string;
   n, k: integer;

function check(x: string):
          integer;
var
  s, len: integer;
begin
  s := 0;
  x := ' ' + x;
  len := length(x);
  for var i := 1 to len - 1 do
   if (x[i] = ' ') and (x[i + 1] <> ' ')
   then
     inc(s);
  check := s;
end;

begin
  writeln('Количество');
  readln(n);
  writeln('Элементы');
  for var i := 1 to n do
    readln(a[i]);
  k := 0;
  writeln;
  writeln('Искомые строки:');
  for var i := 1 to n do
   if check(a[i]) mod 2 <> 0 then
   begin
     inc(k);
     writeln(a[i]);
   end;
  writeln('Всего - ', k);
end.
```

I. Исходные данные: a — массив строк, n — количество строк в массиве.

II. Результат: искомые строки и их количество.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Инициализация счетчика: k := 0.
3. Будем просматривать каждую строку и определять, сколько в ней слов. Для проверки создадим функцию check, которая будет получать в качестве параметра элемент массива и возвращать количество слов в строке. Если количество слов является нечетным числом, то выведем строку и увеличим значение счетчика.

   3.1. Перед каждым словом предложения, кроме первого, стоит пробел, слово начинается с символа, который пробелом не является.

   3.2. Добавим пробел перед первым словом, тогда количество слов будет определяться количеством сочетаний пар символов: пробел и не пробел.

IV. Описание переменных: a — **array**[1..100] **of** string; n, k — integer.

При вводе данных для тестирования программы нужно помнить, что после каждого предложения необходимо нажимать клавишу Enter.

В данном случае строка как тип данных может не соответствовать строке в окне вывода. В примере 5.12 вводятся 3 строки:

1. «Многие компании разрабатывают свои правила по оформлению кода».

2. «В них прописаны также правила именования переменных».

3. «Компания Microsoft использует так называемую «венгерскую нотацию».

В окне вывода количество строк может быть больше (на рисунке их 6).

То, как будут выглядеть вводимые строки в окне вывода, зависит от ширины окна приложения PascalABC.NET. На большом мониторе, если окно приложения развернуто на весь экран, строк может быть три.

Компилятор определяет, что ввод строки закончен, если была нажата клавиша Enter. Внешний вид строк в окне вывода для компилятора не имеет значения.

Пример 5.12. *Продолжение.*

VI. Тестирование.

Окно вывода

```
Количество
3
Элементы
Многие компании разрабатывают
свои правила по оформлению кода.
В них прописаны также правила
именования переменных.
Компания Microsoft использует так
называемую «венгерскую нотацию».

Искомые строки
В них прописаны также правила
именования переменных.
Компания Microsoft использует так
называемую «венгерскую нотацию».

Всего – 2
```

1. Что называют последовательным поиском?
2. Как определить, что в массиве был найден элемент с определенными свойствами?
3. Для чего используют переменные «счетчики»?
4. Что такое инициализация переменной?

Упражнения

1 Для примеров 5.1—5.3 выполните перечисленные задания.

1. Заполните таблицу.

№	n	Массив	x	Результат
1	5	2 14 7 20 16	20	
2	5	2 3 4 5 6	8	
3	7	2 4 6 8 10 11 12	8	
4	5	6 3 9 12 15	3	

2. Добавьте в таблицу свои данные, такие, чтобы искомый элемент был первым в массиве; последним в массиве.

3. Какой ответ выдаст каждая из программ, если в массиве несколько элементов, удовлетворяющих условию задачи? Почему?

4. Что нужно изменить в программе 5.2, чтобы выдавался не последний из найденных элементов, а первый?

5. Что нужно изменить в программе 5.1, чтобы выдавался не последний из найденных элементов, а первый?

6. Измените условие цикла **while** примера 5.3 так, чтобы использовалась логическая операция not.

2 Рост учащихся класса представлен в виде массива. Определите количество учащихся, рост которых больше среднего роста по классу.

3 Заданы фамилии и рост учащихся 10-го класса. Вывести фамилии тех учащихся, рост которых меньше среднего роста по классу.

4 Известны данные о площади n стран (в млн кв. км) и численности населения (в млн жителей). Выведите номера тех стран, плотность населения которых больше x.

5 Для упражнения 4 добавьте возможность вводить и выводить названия стран.

6 Определите, есть ли в линейном массиве хотя бы один элемент, который является нечетным числом, кратным 7. Если да, то следует вывести его номер.

7* В линейном массиве найдите и выведите все простые числа с нечетной суммой цифр. Укажите, сколько чисел вывели.

8* В линейном массиве найдите и выведите все числа Армстронга. (Числом Армстронга называется такое число, которое равно сумме своих цифр, возведенных в степень, равную количеству его цифр. Например, числом Армстронга является число 371 : $371 = 3^3 + 7^3 + 1^3 = 27 + 343 + 1$.) Укажите, сколько чисел вывели.

9 Задан одномерный массив из N строк. Каждая строка является предложением из слов, разделенных пробелами. Найдите и выведите те предложения, в которых есть слова, начинающиеся на гласную (строчную или прописную).

§ 6. Максимальный и минимальный элементы массива

Пример 6.1.
V. Программа:

```
var a: array[1..20] of integer;
  n, max: integer;
begin
  write('Количество n =');
  readln(n);
  writeln('Элементы массива');
  for var i := 1 to n do
    read(a[i]);
  max := a[1];
  for var i := 2 to n do
    if a[i] > max then
      max := a[i];
  writeln('Максимум = ', max);
end.
```

6.1. Поиск максимального (минимального) элемента в массиве

Очень часто для решения задачи требуется находить не заданный элемент массива, а максимальный (наибольший) или минимальный (наименьший).

Рассмотрим задачу нахождения максимального элемента. Если в массиве один-единственный элемент, то он и есть максимальный. Если элементов больше одного, то максимальным в массиве из i элементов является максимум из a[i] и максимального

среди первых i-1 элементов. Находить максимум будем последовательно, сравнивая текущий элемент с максимумом, найденным на предыдущем шаге. Если текущий элемент больше, то значение максимума, найденное на предыдущем шаге, нужно обновить (пример 6.1).

Данный алгоритм находит значение максимального элемента, но не позволяет определить, на каком месте в массиве расположен этот максимальный элемент.

Будем использовать переменную n_max для хранения индекса максимального элемента. Значение переменной n_max будет изменяться тогда, когда изменяется значение максимального элемента (пример 6.2).

Если в массиве несколько элементов имеют максимальное значение, то значением переменной n_max будет индекс первого из них. Если использовать условие a[i] ⩾ max, то переменная n_max будет хранить индекс последнего из максимальных элементов.

В случае, когда известен индекс i элемента массива, значение элемента можно получить, обратившись к элементу по индексу: a[i]. Поэтому при поиске максимального элемента достаточно хранить только его индекс n_max. Значение максимального элемента — a[n_max] (пример 6.3).

Для поиска минимального элемента необходимо заменить знак > в условии оператора ветвления на знак < (пример 6.4).

Пример 6.1. *Продолжение.*
VI. Тестирование.

```
Окно вывода
Количество n = 5
Элементы массива
1 2 6 3 2
Максимум = 6
```

Пример 6.2.
V. Программа:

```
var a: array[1..20] of integer;
    n, max, n_max: integer;
begin
 write('Количество n =');
 readln(n);
 writeln('Элементы массива');
 for var i := 1 to n do
  read(a[i]);
 max := a[1]; n_max := 1;
 for var i := 2 to n do
  if a[i] > max then
  begin
   max := a[i]; n_max :=i;
  end;
 writeln('Максимум = ', max);
 writeln('Его место ', n_max);
end.
```

VI. Тестирование.

```
Окно вывода
Количество n = 5
Элементы массива
2 5 3 5 1
Максимум = 5
Его место 2
```

Пример 6.3. Фрагмент программы:

```
n_max := 1;
for var i := 2 to n do
 if a[i] > a[n_max] then
  n_max := i;
writeln('Максимум =',a[n_max]);
writeln('Его место ', n_max);
```

Пример 6.4. Фрагмент программы:

```
n_min := 1;
for var i := 2 to n do
 if a[i] < a[n_min] then
  n_min := i;
```

Пример 6.5.
V. Программа:

```
var a: array [1..20] of real;
    n, n_min: integer;
begin
 writeln('Количество
        спортсменов');
 readln(n); writeln('Время');
 for var i := 1 to n do
  read(a[i]);
 //поиск минимального элемента
 n_min := 1;
 for var i := 2 to n do
  if a[i] < a[n_min] then
    n_min := i;
 writeln('Победитель — лыжник
        номер ', n_min);
 writeln('Его время - ', a[n_min]);
end.
```

VI. Тестирование.

Окно вывода

```
Количество спортсменов
5
Время
6.31 6.17 7.32 6.54 7.03
Победитель - лыжник номер 2
Его время - 6.17
```

Пример 6.6.
V. Программа:

```
var a: array [1..20] of integer;
    n, min, k : integer;
begin
 write('Количество n =');
 readln(n); writeln('Числа');
 for var i := 1 to n do
  read(a[i]);
 //поиск минимального элемента
 min := a[1];
 for var i := 2 to n do
  if a[i] < min then
     min := a[i];
 //подсчет количества
 k := 0;
 for var i := 1 to n do
  if a[i] = min then
     k := k + 1;
 writeln('Минимальный ', min);
 writeln('Встретился ',k,
        ' раз(-а)');
end.
```

6.2. Решение задач с использованием алгоритма поиска максимального (минимального) элемента

Пример 6.5. В массиве хранится информация о результатах спортсменов, участвующих в лыжной гонке. Определить результат победителя и его номер.

I. Исходные данные: массив `a` — числа, являющиеся временем прохождения трассы, количество спортсменов — `n`.

II. Результат: `a[n_min]` — минимальное время, `n_min` — номер победителя.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Для решения задачи воспользуемся алгоритмом поиска минимального элемента в массиве и его номера (пример 6.4).
3. Вывод результата.

IV. Описание переменных: `a` — **array**`[1..20]` **of** `real`; `n`, `n_min` — `integer`.

Пример 6.6. Определить, сколько раз в линейном массиве встречается элемент, равный минимальному.

I. Исходные данные: массив `a`, количество чисел `n`.

II. Результат: `min` — минимальный элемент, `k` — количество минимальных.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Поиск минимального элемента.
3. Линейный поиск элементов, равных минимальному.
4. Вывод результата.

IV. Описание переменных: a — **array**[1..20] **of** integer; n, min, k — integer.

Пример 6.7. Задан массив из слов различной длины. Найти в нем самое длинное и самое короткое слово.

I. Исходные данные: массив a, количество слов n.

II. Результат: min_s — самое короткое слово, max_s — самое длинное слово.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Поиск самого короткого слова. Самое короткое слово — слово, в котором минимальное количество символов. Для его поиска можно воспользоваться алгоритмом поиска минимального элемента в массиве. Однако, если сравнивать сами элементы массива, то сравнение будет происходить не по длине[1]. Для сравнения строк по длине нужно использовать функцию вычисления длины строки length.
3. Для поиска самого длинного слова можно использовать алгоритм поиска максимального элемента и сравнивать элементы с использованием функции вычисления длины строки length.
4. Вывод результата.

IV. Описание переменных: a — **array**[1..20] **of** string; n — integer; min_s, max_s: string;

Пример 6.6. *Продолжение.*

VI. Тестирование.

```
Окно вывода
Количество n = 5
Числа
3 1 2 1 1
Минимальный 1
Встретился 3 раза(-а)
```

Пример 6.7.

V. Программа:

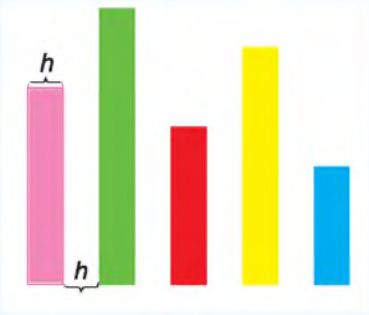
```
var a: array [1..20] of string;
    n: integer;
    min_s, max_s: string;
begin
 write('Количество n =');
 readln(n); writeln('Слова');
 for var i := 1 to n do
  readln(a[i]);
 //поиск короткого слова
 min_s := a[1];
 for var i := 2 to n do
  if length(a[i]) < length(min_s)
  then
    min_s := a[i];
 //поиск длинного слова
 max_s := a[1];
 for var i := 2 to n do
  if length(a[i]) > length(max_s)
  then
    max_s := a[i];
 writeln('Короткое - ',min_s);
 writeln('Длинное - ',max_s);
end.
```

VI. Тестирование.

```
Окно вывода
Количество n = 5
Слова
Все
дороги
ведут
в
Рим
Короткое - в
Длинное - дороги
```

[1] Сравнение строк осуществляется **лексикографически**: s1 < s2, если для первого несовпадающего символа с номером i верно неравенство s1[i] < s2[i] или все символы строк совпадают, но s1 короче s2.

Пример 6.8.



V. Программа:

```
uses graphABC;
var a: array[1..20] of integer;
  n, max, h, x, y1, y2: integer;
  m: real;
begin
  write('Количество n =');
  readln(n);
  writeln(n);
  writeln('Элементы массива');
  for var i := 1 to n do
  begin
    read(a[i]);
    write(a[i],' ');
  end;
  max := a[1];
  for var i := 2 to n do
    if a[i] > max then
      max := a[i];
  h := trunc(WindowWidth/(2*n+1));
  m := WindowHeight/max;
  x := h;
  for var i := 1 to n do
  begin
    SetBrushColor(clrandom);
    y1 := WindowHeight;
    y2 := y1 - trunc(a[i]*m);
    Rectangle(x, y1, x+h, y2);
    x := x + 2*h;
  end;
end.
```

6.3. Построение гистограммы (столбчатой диаграммы)

Пример 6.8. Дан одномерный массив из целых чисел. Построить гистограмму по числовым данным, хранящимся в массиве.

I. Исходные данные: массив a, количество чисел n.

II. Результат: построенная диаграмма.

III. Алгоритм решения задачи.

1. Ввод исходных данных.

2. Гистограмма состоит из n прямоугольников одинаковой ширины. Элементы массива определяют высоту соответствующего прямоугольника. Максимальное значение элементов массива (переменная max) должно по высоте поместиться в окне (WindowHeight). Обозначим $= \frac{n}{\quad}$ (масштабный коэффициент). Тогда значению элемента массива a[i] будет соответствовать целая часть от величины a[i] * m.

3. Находим максимальный элемент массива.

4. В цикле строим прямоугольники. Все прямоугольники имеют одинаковую ширину (h), расстояние между ними можно определить равным ширине прямоугольника. Тогда ширина прямоугольника — целая часть от деления ширины окна на (2n + 1):

$$= \frac{n}{n+}.$$

5. При вычислении высоты прямоугольника нужно учесть то, что ось `Y` направлена сверху вниз.

6. Цвет прямоугольника будем задавать случайным образом.

7. Местоположение прямоугольника определяется переменной `x`. Начальное значение `x = h`. Новое значение получается из предыдущего увеличением на `2*h`.

8. Вывод результата.

IV. Описание переменных: `a` — **array**`[1..20]` **of** `integer; n, max, h, x, y1, y2` — `integer; m: real.`

Пример 6.8. *Продолжение.*
VI. Тестирование.

VII. Постройте по этим данным диаграмму в Excel и сравните.

1. Какой элемент массива является максимальным? Какой — минимальным?
2. Как найти максимальный элемент в массиве?
3. Как найти минимальный элемент?
4. Каким образом определить номер первого элемента, равного максимальному?
5. Как определить номер последнего элемента, равного минимальному?

Упражнения

1 Измените программы примеров 6.1 и 6.2 так, чтобы находился минимальный элемент в массиве.

2 Для примера 6.5 выполните перечисленные задания.

1. Найдите номер спортсмена, пришедшего на финиш последним.
2. Определите, был ли победитель единственным или есть еще лыжник, прошедший трассу с таким же результатом (см. пример 6.6).
3. Добавьте еще один массив. Введите в него фамилии спортсменов. Реализуйте пункты 1—3 так, чтобы выводилась фамилия, а не номер (см. пример 5.10).

3 В массиве хранится информация о стоимости автомобилей. Определите стоимость самого дорогого автомобиля и его номер в массиве. Если есть несколько таких автомобилей, то выведите все номера.

4 В массиве хранится информация о среднедневной температуре декабря. Определите, сколько в декабре было дней с самой низкой и с самой высокой температурой.

5 Задан массив из слов. Найдите в нем самое длинное слово, заканчивающееся буквой *а*.

6* Задан массив из слов. Найдите в нем самое короткое слово, начинающееся с прописной буквы.

7 Для примера 6.8 выполните перечисленные задания.

1. Измените программу так, чтобы при построении диаграммы использовалась не вся высота окна, а оставались поля сверху и снизу.
2. Измените программу так, чтобы столбики строились без промежутков между ними.
3. Создайте массив цветовых констант и используйте эти цвета для закрашивания столбиков.
4. Постройте линейчатую диаграмму.

8 По данным массива постройте диаграмму в виде ломаной линии. Соответствуют ли ординаты вершин ломаной значениям массива?

§ 7. Преобразование элементов массива

Пример 7.1.

V. Программа:

```
var a: array[1..20] of integer;
    n: integer;
begin
 write('Количество n =');
 readln(n);
 writeln('Элементы массива');
 for var i := 1 to n do
  read(a[i]);
 for var i := 1 to n do
 begin
  if a[i] > 0 then
     a[i] := a[i] * 2;
  if a[i] < 0 then
     a[i] := a[i] + 5;
 end;
 writeln('Преобразованный массив');
 for var i := 1 to n do
  write(a[i], ' ');
end.
```

VI. Тестирование.

Окно вывода

```
Количество n = 5
Элементы массива
3 -2 0 -1 5
Преобразованный массив
6 3 0 4 10
```

VII. Анализ результатов. Элементы 3 и 5 увеличены в 2 раза, элементы –2 и –1 увеличены на 5, элемент 0 остался неизмененным.

7.1. Основные задачи

Среди задач преобразования элементов массива можно выделить задачи следующих типов:

1. Изменение элементов массива в зависимости от условий.
2. Обмен местами элементов массива.
3. Удаление элемента из массива.
4. Вставка элемента в массив.

Рассмотрим каждую из задач.

7.2. Изменение элементов массива в зависимости от выполнения некоторых условий

Пример 7.1. Задан одномерный массив целых чисел. Преобразовать его элементы по следующему правилу: положительные элементы увеличить в 2 раза, а отрицательные — увеличить на 5.

I. Исходные данные: одномерный массив a, количество элементов n.

II. Результат: преобразованный массив a.

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. В цикле проверяем текущий элемент. Если он положительный,

умножаем его на 2. Если элемент отрицательный, то прибавляем к нему 5. Помним, что отрицание условия «элемент положительный» — условие «элемент не положительный», что подразумевает возможность равенства элемента нулю. Поэтому нужны два оператора ветвления для проверки условия задачи.

3. Вывод результата.

IV. Описание переменных: a — **array**[1..20] **of** integer; n — integer.

7.3. Обмен местами элементов в массиве

Для обмена местами двух элементов массива можно использовать дополнительную переменную, которую называют буфером. Буферу присваивают значение одного из элементов массива, этому элементу присваивают значение другого элемента массива, затем второму элементу присваивают значение буфера:

```
buf := a[i];
a[i] := a[k];
a[k] := buf;
```

Пример 7.2. Задан одномерный массив целых чисел. Поменять местами максимальный и минимальный элементы массива (минимальный и максимальный элементы встречаются в массиве только один раз).

I. Исходные данные: одномерный массив a, количество элементов n.

II. Результат: преобразованный массив a.

III. Алгоритм решения задачи.

1. Введем исходные данные.

2. Найдем максимальный элемент массива и его индекс (n_max).

Если обмен элементов осуществлять следующим образом:

```
a[i] := a[k]; a[k] := a[i];
```

то мы потеряем значение элемента, стоящего изначально на месте a[i], и получим два элемента со значением, равным a[k]. Для обмена элементов можно использовать встроенную функцию swap: swap(a[i], a[k]).

Пример 7.2.

V. Программа:

```
var a: array[1..20] of integer;
  n, n_min, n_max, buf: integer;
begin
 write('Количество n =');
 readln(n);
 writeln('Элементы массива');
 for var i := 1 to n do
  read(a[i]);
 n_min := 1;
 n_max := 1;
 for var i := 1 to n do
 begin
  if a[i] > a[n_max] then
   n_max := i;
  if a[i] < a[n_min] then
   n_min := i;
 end;
 buf := a[n_min];
 a[n_min] := a[n_max];
 a[n_max] := buf;
 writeln('Преобразованный
         массив');
 for var i := 1 to n do
  write(a[i], ' ');
end.
```

VI. Тестирование.

```
Окно вывода
Количество n = 5
Элементы массива
2 1 3 5 4
Преобразованный массив
2 5 3 1 4
```

Пример 7.3.
V. Программа:

```
var a: array[1..20] of integer;
    n, d, j: integer;
procedure del_mas(k: integer);
begin
  for var i := k + 1 to n do
    a[i - 1] := a[i];
  n := n - 1;
end;
begin
  write('Количество n =');
  readln(n);
  writeln('Элементы массива');
  for var i := 1 to n do
    read(a[i]);
  d := 0;
  j := 1;
  while j <= n do
  begin
    if a[j] mod 5 = 0 then
    begin
      del_mas(j);
      d := d + 1;
      j := j - 1;
    end;
    j := j + 1;
  end;
  writeln('Удалили ', d,
    'элемент(-а, -ов');
  writeln('Преобразованный
      массив');
  for var i := 1 to n do
    write(a[i], ' ');
end.
```

VI. Тестирование.

```
Окно вывода
Количество n = 7
Элементы массива
5 3 15 35 10 4 30
Удалили 5 элемент(-а,-ов)
Преобразованный массив
3 4
```

VII. Анализ результатов. Элементы 5, 15, 35, 10 и 30 кратны 5, поэтому их удалили из массива. Элементы 3 и 4 не кратны 5, поэтому они остались в массиве и сдвинулись, соответственно, на 1-е и 2-е места.

3. Найдем минимальный элемент массива и его индекс (n_min).

4. Поменяем местами элементы, стоящие на местах n_max и n_min.

5. Выведем результат.

IV. Описание переменных: a — **array**[1..20] **of** integer; n, n_min, n_max, buf — integer.

7.4*. Удаление элемента из массива

Для удаления элемента массива на месте k нужно сдвинуть на одну позицию влево все элементы, стоящие после него. Количество элементов при этом уменьшаем на 1.

```
for var i := k + 1 to n do
        a[i-1] := a[i];
n := n - 1;
```

Если в массиве нужно удалить не один, а несколько элементов, удовлетворяющих условию, то можно использовать вспомогательный алгоритм в виде соответствующей процедуры **procedure** del_mas(k: integer); Параметр k — номер удаляемого элемента.

Пример 7.3. Задан одномерный массив целых чисел. Удалить из линейного массива все числа, кратные 5. Сколько чисел удалили?

I. Исходные данные: одномерный массив a, количество элементов n.

II. Результат: преобразованный массив a и количество удаленных чисел d.

III. Алгоритм решения задачи.

1. Введем исходные данные.

2. Будем последовательно просматривать элементы массива. Если найдем число, кратное 5, то удалим его из массива, используя процедуру del_mas. Так как количество удаляемых элементов

заранее не известно, то применим цикл **while**.

3. При удалении элемента счетчик d будем увеличивать на 1.

4. Выведем результат.

IV. Описание переменных: a — **array**[1..20] **of** integer; n, d, j — integer.

7.5*. Вставка элемента в массив

Для вставки элемента на место k нужно освободить данное место в массиве. Для этого сдвинем на одну позицию вправо все элементы массива, стоящие после k-1. Сдвиг начинаем с последнего элемента. Количество элементов в массиве увеличится на 1.

Пример 7.4. Задан массив целых чисел. Вставить число x на k-е место.

I. Исходные данные: одномерный массив a, количество элементов n, число, которое нужно вставить в массив x, номер позиции в массиве, на которую нужно вставить число k.

II. Результат: преобразованный массив a.

III. Алгоритм решения задачи.

1. Ввод исходных данных.

2. Сдвигаем все элементы массива, стоящие после k – 1 на одну позицию вправо.

3. Увеличим n — количество элементов.

4. Вставляем число x на место k.

5. Выводим результат.

IV. Описание переменных: a — **array**[1..20] **of** integer; n, k, x — integer.

Пример 7.4.

V. Программа:

```
var a: array[1..20] of integer;
    n, k, x: integer;
begin
 write('Количество n =');
 readln(n);
 writeln('Элементы массива');
 for var i := 1 to n do
  read(a[i]);
 write('Число x =');
 readln(x);
 write('Номер позиции k =');
 readln(k);
 //сдвиг элементов вправо на 1
 for var i:= n downto k do
  a[i+1] := a[i];
 //вставка x на место k
 a[k] := x;
 n := n + 1;
 writeln('Преобразованный
    массив');
 for var i := 1 to n do
  write(a[i], ' ');
end.
```

VI. Тестирование.

Окно вывода

```
Количество n = 5
Элементы массива
3 2 5 -3 7
Число x = 6
Номер позиции k = 2
Преобразованный массив
3 6 2 5 -3 7
```

Методы расширения одномерных динамических массивов позволяют преобразовывать массивы с помощью встроенных функций ConvertAll, Replace, Transform и др. (см. справочную систему PascalABC.Net).

?

1. Какие типы задач преобразования массивов вы можете назвать?
2. Как можно поменять местами два элемента в массиве?
3. Как удалить элемент из массива?
4. Как вставить элемент в массив?

Упражнения

1 Для задачи из примера 7.1 выполните перечисленные задания.

1. Заполните таблицу.

№	n	a	Преобразованный массив
1	3	−2 −3 −5	
2	5	1 2 3 4 5	
3	10	1 −3 −2 0 4 0 2 −4 0 2	

2. Добавьте в таблицу свои значения n и a.
3. Можно ли заменить команды из п. 3.1. командами из п. 3.2?

3.1.
```
if a[i] > 0 then
  a[i] := a[i] * 2;
if a[i] < 0 then
  a[i] := a[i] + 5;
```

3.2.
```
if a[i] < 0 then
  a[i] := a[i] + 5;
if a[i] > 0 then
  a[i] := a[i] * 2;
```

4. В каких случаях программа будет давать неверный результат?

2 Задан одномерный массив. Преобразуйте его элементы по следующему правилу: из всех положительных элементов вычесть элемент с номером k, ко всем отрицательным добавить введенное число x. Нулевые элементы оставьте без изменения.

3 Задан одномерный массив из четного количества элементов. Поменяйте местами его «половинки».

4 В массиве записаны фамилии и имена учащихся класса. Из класса выбыли два учащихся. Известны их номера. Исключите данные этих учащихся из массива.

5 Для задачи из примера 7.4 выполните перечисленные задания.

1. Заполните таблицу:

№	n	a	x	k	Преобразованный массив
1	3	−2 −3 −5	0	2	
2	5	1 2 3 4 5	0	1	
3	10	1 −3 −2 0 4 0 2 −4 0 2	10	11	

2. Добавьте в таблицу свои значения n, a, x, k.
3. Какой результат выдаст программа, если ввести n = 5, а k = 120? Вставьте в программу проверку для числа k ($1 < k \leqslant n + 1$).
4. Какой результат получим, если заменить цикл из п. 4.1 циклом из п. 4.2?

4.1.
```
for var i := n downto k do
  a[i + 1] := a[i];
```

4.2.
```
for var i := k to n do
  a[i + 1] := a[i];
```

6 Переставьте первый элемент массива на последнее место, второй — на первое, третий — на второе и т. д.