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.

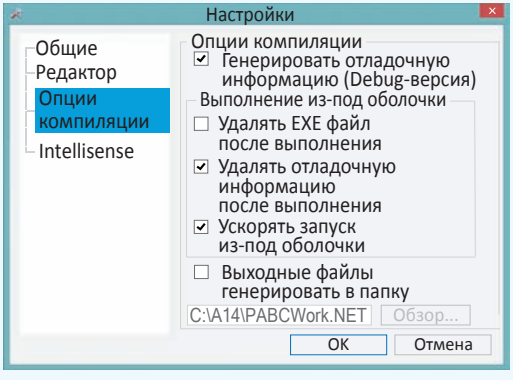
Пример 2.1. Некоторые языки программирования высокого уровня:

- 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).

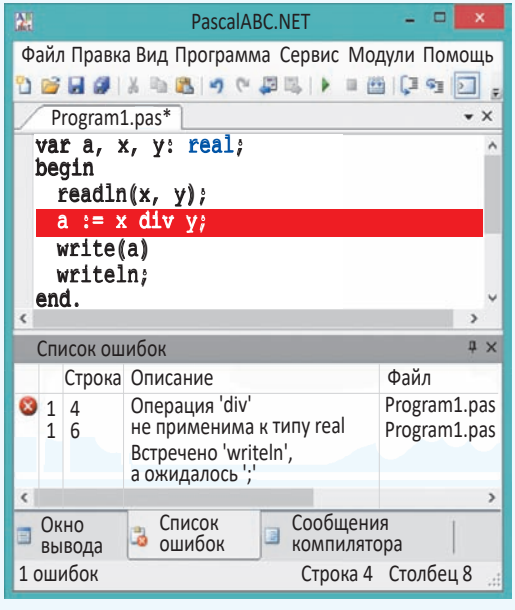
За время существования вычислительных машин было создано более 8 тыс. языков программирования, и ежегодно появляются новые. Некоторые из них являются узкоспециальными, другие используются миллионами программистов по всему миру.

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).

Операторы

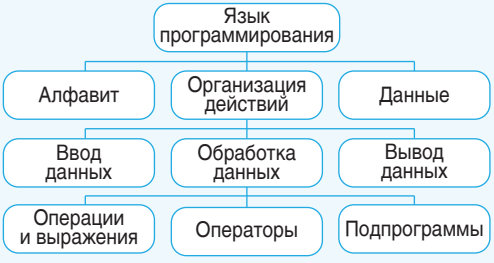
Одним из основных понятий всех языков программирования является **оператор**, который представляет собой законченную фразу языка и является предписанием на выполнение конкретных действий по обработке данных. Программа строится из операторов так же, как текст литературного

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.



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).

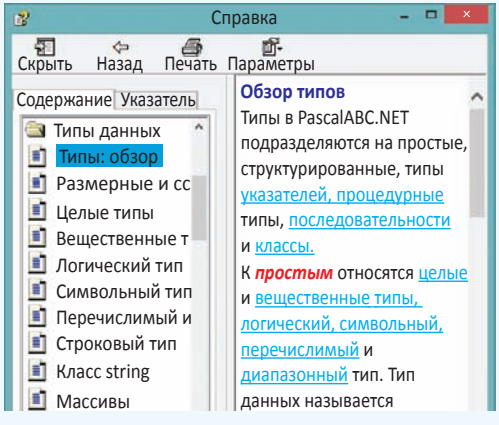
Pascal:
```
for var i := 1 to n do
  s := s + i * i;
```

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.

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.

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].

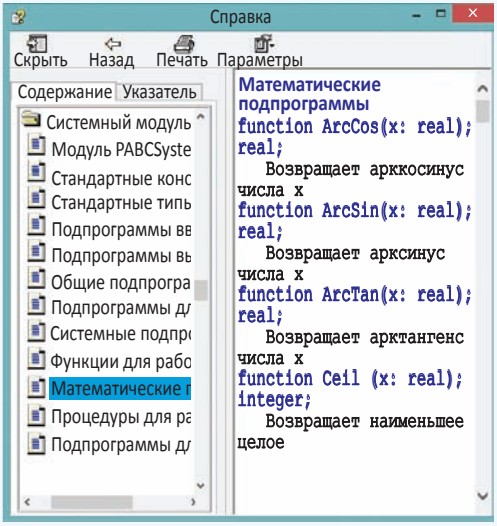
Правила оформления кода, разработанные в некоторых компаниях, являются открытыми и могут использоваться другими разработчиками. Например, в Google разработаны styleguide («гид по стилю») для разных языков программирования (C++, Java, Python, Lisp и др.)[3].

[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 $1+\frac{1}{2^2}+\frac{1}{3^2}+\ldots+\frac{1}{N^2}$ 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