МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Г. Р. Кадырова

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Учебное пособие

Ульяновск
УлГТУ
2014

УДК 681.3.06(075)
ББК 22.18я7
К 13

Рецензенты: д-р техн. наук, профессор кафедры «Информационные технологии» УлГУ, Семушин И. В.;

зав. кафедрой «Гуманитарные и естественно-научные дисциплины» филиала ПАГС, канд. пед. наук, доцент Ж. В. Болтачева

Утверждено редакционно-издательским советом университета в качестве учебного пособия.

Кадырова, Г. Р.

К 13 Основы алгоритмизации и программирования : учебное пособие / Г. Р. Кадырова. – Ульяновск : УлГТУ, 2014. – 95 с.

ISBN 978-5-9795-1273-0

Данное пособие рассматривает основы построения алгоритмов и программирования на базе языка Паскаль, очерченные стандартом дисциплины для данных специальностей; содержит большое число задач, которые могут быть использованы для организации практических и лабораторных занятий.

Учебное пособие предназначено для студентов дневной и заочной форм обучения для направления 08010062 «Экономика» профили: «Финансы и кредит», «Бухгалтерский учет и аудит», «Налоги и налогообложение» при подготовке к практическим и лабораторным занятиям по курсу «Информатика» и для индивидуального обучения основам программирования.

Подготовлено на кафедре «Прикладная математика и информатика».

УДК 681.3.06 (075)
ББК 22.18я7

ISBN 978-5-9795-1273-0

Оглавление

Предисловие

Одним из базовых понятий информатики, вычислительной техники и программирования является понятие алгоритма, как некоторого правила преобразования информации. Умение составлять алгоритмы и программы решения различных практических задач является элементом алгоритмической культуры современного специалиста в области информационных технологий.

Целью данного учебного пособия является изучение основ алгоритмизации, уяснение понятия и этапов эволюции технологии программирования.

Тему «Алгоритм и его свойства» традиционно считают главной темой теоретической информатики, вводящей в обширные практические разделы алгоритмизации. Разработка алгоритмов и программ для решения прикладных задач является одним из важнейших направлений дисциплины «Информатика» и остается наиболее стабильным компонентом подготовки специалиста в вузе. Эти темы, а также этапы создания программ, классификация языков программирования, методологии программирования рассмотрены в данном пособии.

Основы программирования излагаются на языке Паскаль, который является основным языком, изучаемым в ходе подготовки специалистов.

Пособие содержит большое число задач, которые могут быть использованы для организации практических и лабораторных занятий.

Материалы пособия многократно апробировались на занятиях, зачетах и экзаменах со студентами Ульяновского государственного технического университета.

Раздел 1. Основы алгоритмизации

Понятие алгоритма. Свойства и виды алгоритма

Алгоритм – четкое описание последовательности действий, которые необходимо выполнить для получения результата.

Термин «алгоритм» происходит от латинской формы имени среднеазиатского математика Аль-Хорезми – Algorithmi. Алгоритм является одним из основных понятий информатики и математики.

К основным свойствам алгоритмов относятся:

1) **дискретность** (прерывность, раздельность) – алгоритм должен представлять процесс решения задачи как последовательное исполнение простых (или ранее определенных) шагов (этапов);

2) **определенность** – каждое правило алгоритма должно быть четким, однозначным и не оставлять места для произвола. Это свойство обеспечивает выполнение алгоритма механически, не требуя никаких дополнительных указаний или сведений о решаемой задаче;

3) **результативность** (или конечность) – алгоритм должен приводить к решению задачи за конечное число шагов;

4) **массовость** – алгоритм решения задачи производится в общем виде, т. е. его можно будет применять для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из определенной области, которая называется областью применимости алгоритма.

На практике чаще всего встречаются следующие формы представления алгоритмов:

- **словесная** – записывается на естественном языке;
- **графическая** – с помощью блок-схемы;
- **псевдокоды** – полуформализованные описания алгоритмов на некотором условном алгоритмическом языке, которые включают в себя как элементы языка программирования, так и фразы

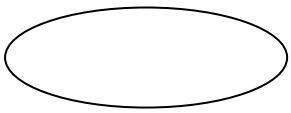
естественного языка, общепринятые математические обозначения и др.

Рассмотрим пример алгоритма на естественном языке:

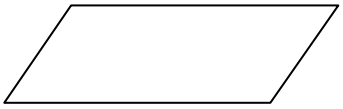
1. Ввести в компьютер числовые значения переменных a, b и c.
2. Вычислить дискриминант по формуле $d = b^2 - 4ac$.
3. Если d < 0, то напечатать сообщение «Корней нет» и перейти к п.4. Иначе вычислить $x_1 = \frac{-b+\sqrt{D}}{2a}$, $x_2 = \frac{-b-\sqrt{D}}{2a}$ и напечатать значения x_1 и x_2.
4. Прекратить вычисления.

Блок-схемой называется наглядное графическое изображение алгоритма, когда отдельные его этапы изображаются при помощи различных геометрических фигур – блоков, а связи между этапами (последовательность выполнения этапов) указываются при помощи стрелок, соединяющих эти фигуры. Блоки сопровождаются надписями. Типичные действия алгоритма изображаются следующими геометрическими фигурами:

Блок начала-конца алгоритма. Надпись в блоке: «начало» («конец»).

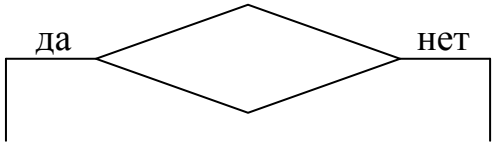


Блок ввода-вывода данных. Надпись в блоке: слово «ввод» («вывод» или «печать») и список вводимых (выводимых) переменных.



Блок решения или **действия**. Надпись в блоке: конкретная операция или группа операций.

Условный блок. Надпись в блоке: проверяемое условие. В результате проверки условия осуществляется выбор одного из возможных путей (ветвей) вычислительного процесса. Если условие выполняется, то следующим выполняется этап по ветви «да», если условие не выполняется, то выполняется этап по ветви «нет».



В качестве примера приведем блок-схему алгоритма решения уравнения, описанного выше.

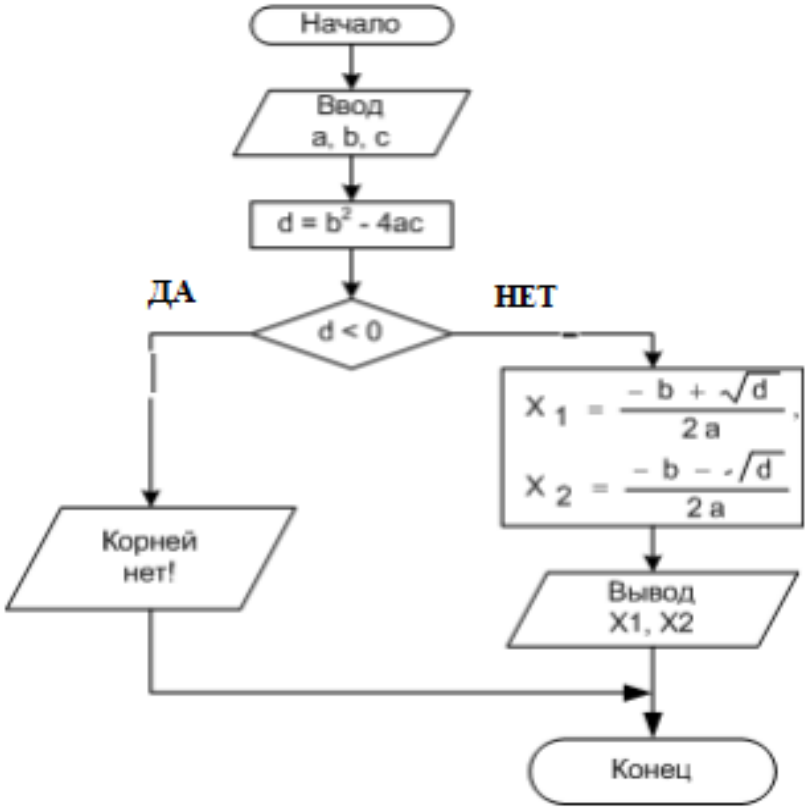


Рис. 1.1. Блок-схема алгоритма решения квадратного уравнения

Виды алгоритма:

1. **Линейный алгоритм**. В котором все операции выполняются последовательно одна за другой (рис. 1.1).

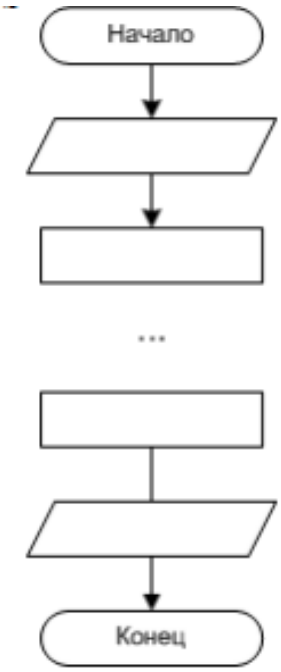


Рис. 1.2. Блок-схема линейного алгоритма

Рассмотрим пример линейного алгоритма.

Зная длины трех сторон треугольника, вычислить площадь и периметр треугольника.

Пусть a, b, c – длины сторон треугольника. Необходимо найти S – площадь треугольника, P – периметр. Для нахождения площади можно воспользоваться формулой Герона:$r = \sqrt{\mathrm{r(r-a)(r-b)(r-c)}}$, где r – полупериметр.

Входные данные: a, b, c. Выходные данные: S, P.

Блок-схема алгоритма представлена на рис. 1.3.

Внимание! В этих блоках знак «=» означает не математическое равенство, а операцию присваивания. Переменной, стоящей слева от оператора, присваивается значение, указанное справа. Причем это значение может быть уже определено или его необходимо вычислить с помощью выражения. Например, операция $r = (a+b+c)/2$ – имеет смысл (переменной r присвоить значение $(a+b+c)/2$), а выражение $(a+b+c)/2=r$ – бессмыслица.

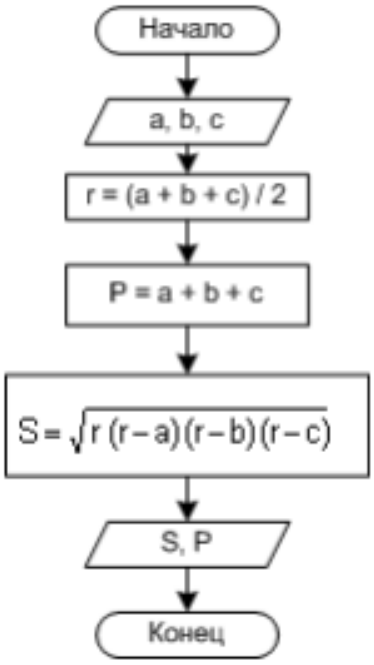


Рис. 1.3. Пример линейного алгоритма

2. **Разветвленный алгоритм**. Алгоритмы разветвленной структуры применяются, когда в зависимости от некоторого условия необходимо выполнить либо одно, либо другое действие.

В блок-схемах разветвленные алгоритмы изображаются так, как показано на рис. 1.4.

Рис. 1.4. Полный и неполный разветвленный алгоритм

В качестве примера разветвленного алгоритма можно рассмотреть блок-схему алгоритма решения квадратного уравнения (см. рис. 1.1).

Рассмотрим еще один пример разветвленного алгоритма.

Даны три числа *a*, *b*, *c*. Найти наибольшее из них.

Блок-схема алгоритма представлена на рис. 1.5.

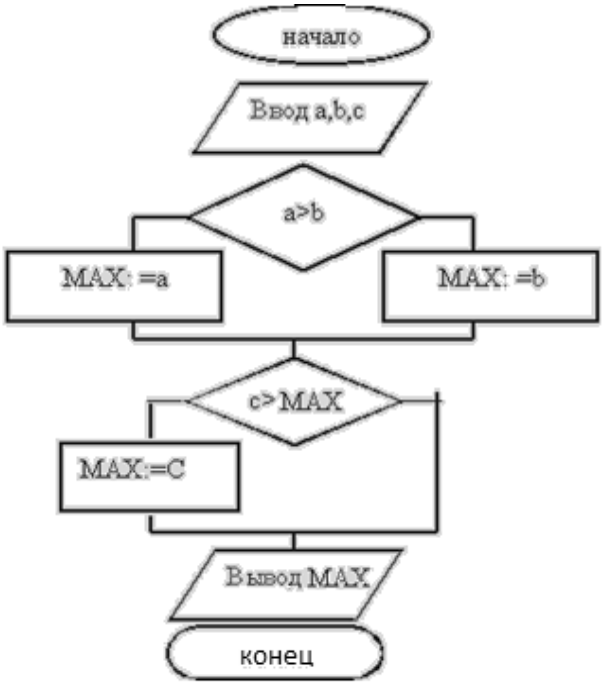


Рис. 1.5. Алгоритм поиска наибольшего из трех чисел

3. **Циклический алгоритм**. **Циклом** называют повторение одних и тех же действий (шагов). Последовательность действий, которые повторяются в цикле, называют **телом цикла**.

Существует два типа алгоритмов циклической структуры. На рис. 1.6 изображен **цикл с предусловием**, а на рис. 1.7 – **цикл с постусловием**.

Эти циклы взаимозаменяемы и обладают некоторыми отличиями:

- в цикле с предусловием условие проверяется до тела цикла, в цикле с постусловием – после тела цикла;

• в цикле с постусловием тело цикла выполняется хотя бы один раз, в цикле с предусловием тело цикла может не выполниться ни разу;

• в цикле с предусловием проверяется условие продолжения цикла, в цикле с постусловием – условие выхода из цикла.

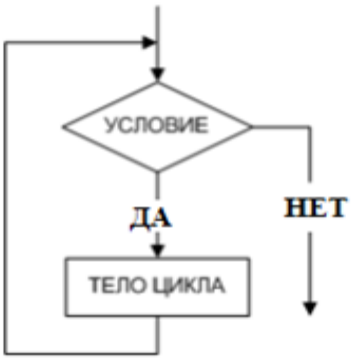


Рис. 1. 6. Алгоритм циклической структуры с предусловием

Рис. 1.7. Алгоритм циклической структуры с постусловием

При написании условных циклических алгоритмов следует помнить следующее. Во-первых, чтобы цикл имел шанс когда-нибудь закончиться, содержимое его тела должно обязательно влиять на условие цикла. Во-вторых, условие должно состоять из корректных выражений и значений, определенных еще до первого выполнения тела цикла.

Обязательные блоки для организации цикла:

1. Установка начального значения параметра цикла.

2. Проверка условия достижения конечного значения параметра цикла.

3. Изменение параметра цикла.

Рассмотрим пример циклического алгоритма (рис. 1.8).

«Семь раз отмерь, один раз отрежь»
I – счетчик, **параметр цикла**, по которому осуществляется выход из цикла.

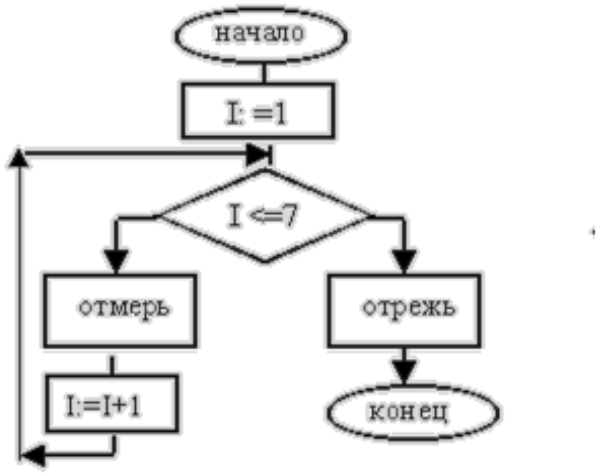


Рис. 1.8. Пример циклического алгоритма

Контрольные вопросы

1. Что такое алгоритм? Свойства алгоритма.
2. Формы представления алгоритма. Перечислить и зарисовать блоки алгоритма.
3. Перечислить виды алгоритма. Примеры.
4. Что такое цикл? Типы циклов. Привести блок схемы. Назвать отличия.

Раздел 2. Языки и методологии программирования

Классификация языков программирования

В настоящее время в мире существует несколько сотен реально используемых языков программирования. Для каждого есть своя область применения.

Любой алгоритм, есть последовательность предписаний, выполнив которые можно за конечное число шагов перейти от исходных данных к результату. В зависимости **от степени детализации** предписаний обычно определяется уровень языка программирования – чем меньше детализация, тем выше уровень языка.

По этому критерию можно выделить следующие уровни языков программирования:

- машинные;
- машинно-ориентированные (ассемблеры);
- машинно-независимые (языки высокого уровня).

Машинные языки и машинно-ориентированные языки – это **языки низкого уровня**, требующие указания мелких деталей процесса обработки данных. Языки же **высокого уровня** имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны для человека.

На заре компьютерной эры **машинный код** был единственным средством общения человека с компьютером. Огромным достижением создателей языков программирования было то, что они сумели заставить сам компьютер работать переводчиком с этих языков на машинный код.

Существующие языки программирования можно разделить на две группы: **процедурные** и **непроцедурные** (рис. 2.1).

Процедурные (или алгоритмические) языки позволяют создавать программы, представляющие собой систему предписаний для решения конкретной задачи (выполнение команд программы определяется их последовательностью, командами перехода, цикла или обращениями к процедурам). Роль компьютера сводится к механическому выполнению этих предписаний.

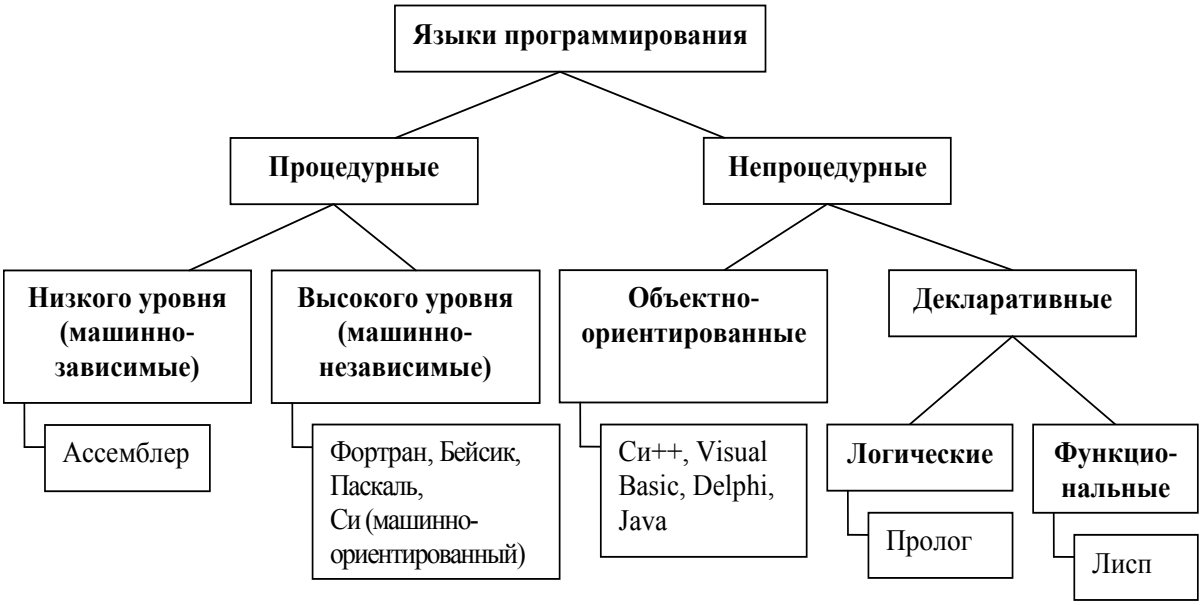


Рис. 2.1. Общая классификация языков программирования

Процедурные языки разделяют на **языки низкого** и **высокого уровня**.

Разные типы процессоров имеют разные наборы команд. Если язык программирования ориентирован на конкретный тип процессора и **учитывает его особенности**, то он называется ***языком программирования низкого уровня***. Имеется в виду, что операторы языка близки к машинному коду и ориентированы на конкретные команды процессора.

С помощью **языков низкого уровня** создаются очень **эффективные** (работают быстрее) и **компактные программы** (занимают меньше места в памяти), так как разработчик получает доступ ко всем возможностям процессора. С помощью этих языков

удобнее разрабатывать системные программы, драйверы (программы для управления устройствами компьютера), некоторые другие виды программ.

Языком низкого уровня (машинно-ориентированным) является *Ассемблер*, который просто представляет каждую команду машинного кода, но не в виде чисел, а с помощью условных символьных обозначений, называемых мнемониками.

Языки программирования **высокого уровня** значительно ближе и **понятнее человеку**, нежели компьютеру. **Особенности** конкретных **компьютерных архитектур** в них **не учитываются**, поэтому создаваемые **программы** на уровне исходных текстов легко **переносимы на другие платформы**, для которых создан транслятор этого языка. Разрабатывать программы на языках высокого уровня с помощью понятных и мощных команд значительно проще, а **ошибок** при создании программ допускается гораздо **меньше**. Для перевода исходных программ с языка высокого уровня на машинный язык используются специальные программы – **трансляторы**.

Работа всех трансляторов строится по одному из двух принципов: **интерпретация** или **компиляция**.

Интерпретация подразумевает пооператорную трансляцию и последующее выполнение оттранслированного оператора исходной программы. В связи с этим можно отметить **два недостатка** метода интерпретации: во-первых, интерпретирующая программа должна находиться в памяти ЭВМ в течение всего процесса выполнения исходной программы, т. е. занимать определенный объем памяти; во-вторых, процесс трансляции одного и того же оператора повторяется столько раз, сколько раз должна исполняться эта команда в программе, что резко снижает производительность работы программы.

Несмотря на указанные недостатки, трансляторы-интерпретаторы получили достаточное распространение, так как они **удобны при разработке** и **отладке исходных программ**.

При **компиляции** процессы трансляции и выполнения разделены во времени: **сначала исходная программа полностью переводится** на машинный язык (после чего наличие транслятора в оперативной памяти становится ненужным), а **затем оттранслированная программа может многократно исполняться**. Следовательно, для одной и той же программы трансляция методом компиляции обеспечивает более **высокую производительность вычислительной системы** при **сокращении требуемой оперативной памяти**.

Компиляция программы включает два действия: **анализ**, т. е. определение правильности записи исходной программы в соответствии с правилами построения языковых конструкций входного языка, и **синтез** – генерирование эквивалентной программы в машинных кодах.

Объектно-ориентированный язык создает окружение в виде множества независимых объектов. Каждый объект ведет себя подобно отдельному компьютеру, их можно использовать для решения задач как «черные ящики», не вникая во внутренние механизмы их функционирования. Из языков объектного программирования, популярных среди профессионалов, следует назвать прежде всего *Cи++*, для более широкого круга программистов предпочтительны среды типа *Delphi* и *Visual Basic*.

При использовании **декларативного языка** программист указывает **исходные** информационные структуры, **взаимосвязи между ними** и то, какими **свойствами должен обладать результат**. При этом процедуру его получения («**алгоритм**») **программист не строит** (по крайней мере, в идеале). В этих языках отсутствует понятие «оператор» («команда»). Декларативные языки можно

подразделить на два семейства – **логические** (типичный представитель – *Пролог*) и **функциональные** (*Лисп*).

С развитием глобальной сети было создано много **языков программирования**, адаптированных специально **для Интернета**. Характерные особенности: языки являются интерпретируемыми, интерпретаторы для них распространяются бесплатно, сами программы – в исходных текстах. Такие языки называются **скрипт-языками**.

Языки веб-программирования – это языки, которые в основном предназначены для работы с веб-технологиями. Языки веб-программирования можно условно разделить на две пересекающиеся группы: **клиентские** и **серверные**.

Как следует из названия, программы на **клиентских языках** обрабатываются на стороне пользователя, как правило их выполняет браузер. Это и создает главную проблему клиентских языков – результат выполнения программы (скрипта) зависит от браузера пользователя. То есть если пользователь запретил выполнять клиентские программы, то они исполняться не будут, как бы ни желал этого программист. Кроме того, может произойти такое, что в разных браузерах или в разных версиях одного и того же браузера один и тот же скрипт будет выполняться по-разному. С другой стороны, если программист возлагает надежды на серверные программы, то он может упростить их работу и снизить нагрузку на сервер за счет программ, исполняемых на стороне клиента, поскольку они не всегда требуют перезагрузку (генерацию) страницы. Самыми распространенными клиентскими языками программирования являются:

- HTML
- CSS
- JavaScript

- VBScript
- ActionScript
- Java

Серверные языки. Когда пользователь дает запрос на какую-либо страницу (переходит на неё по ссылке или вводит адрес в адресной строке своего браузера), то вызванная страница сначала обрабатывается на сервере, то есть выполняются все программы, связанные со страницей, и только потом возвращается к посетителю по сети в виде файла. Этот файл может иметь расширения: HTML, PHP, ASP, ASPX, Perl, SSI, XML, DHTML, XHTML.

Работа программ уже полностью зависима от сервера, на котором расположен сайт, и от того, какая версия того или иного языка поддерживается.

К серверным языкам программирования можно отнести:

- PHP
- Perl
- Python
- Ruby
- любой .NET язык программирования (технология ASP.NET)
- Java
- Groovy

Охарактеризуем наиболее известные языки программирования.

1. **Фортран** (FORmula TRANslating system – система трансляции формул); старейший и сегодня активно используемый в решении **задач математической** ориентации язык. Является классическим языком для программирования на ЭВМ математических и инженерных задач.

2. **Бейсик** (Beginner's All-purpose Symbolic Instruction Code – универсальный символический код инструкций для начинающих); несмотря на многие недостатки и изобилие плохо совместимых

версий – самый популярный по числу пользователей. Широко употребляется при написании простых программ.

3. **Паскаль** (Pascal – назван в честь ученого Блеза Паскаля); чрезвычайно популярен как при изучении программирования, так и среди профессионалов. Создан в начале 70-х годов швейцарским ученым Никлаусом Виртом. Язык Паскаль первоначально разрабатывался как учебный, и, действительно, сейчас он является одним из основных языков обучения программированию в школах и вузах. Однако качества его в совокупности оказались столь высоки, что им охотно пользуются и профессиональные программисты.

Не менее впечатляющей, в том числе и финансовой, удачи добился Филип Кан, француз, разработавший систему Турбо-Паскаль. Суть его идеи состояла в объединении последовательных этапов обработки программы – компиляции, редактирования связей, отладки и диагностики ошибок – в едином интерфейсе. Версии Турбо-Паскаля заполонили практически все образовательные учреждения, программистские центры и частные фирмы. На базе языка Паскаль созданы несколько более мощных языков (Модула, Ада, Дельфи).

4. **АДА**; является языком, победившим (май 1979 г.) в конкурсе по разработке универсального языка, проводимым Пентагоном с 1975 году. Разработчики – группа ученых во главе с Жаном Ихбиа. Победивший язык окрестили АДА, в честь Огасты Ады Лавлейс. Язык АДА – прямой наследник языка Паскаль. Этот язык предназначен для создания и длительного (многолетнего) сопровождения больших программных систем, допускает возможность параллельной обработки, управления процессами в реальном времени и многое другое, чего трудно или невозможно достичь средствами более простых языков.

5. **Си** (С – «си»); широко используется при создании системного программного обеспечения. Наложил большой отпечаток

на современное программирование (первая версия – 1972 г.), является очень популярным в среде разработчиков систем программного обеспечения (включая операционные системы). Си сочетает в себе черты как языка высокого уровня, так и машинно-ориентированного языка, допуская программиста ко всем машинным ресурсам, чего не обеспечивают такие языки, как Бейсик и Паскаль.

6. **Си++** (C++); объектно-ориентированное расширение языка Си, созданное Бьярном Страуструпом в 1980 году. Множество новых мощных возможностей, позволивших резко повысить производительность программистов, наложилось на унаследованную от языка Си определенную низкоуровневость.

7. **Дельфи** (Delphi); язык объектно-ориентированного «визуального» программирования; в данный момент чрезвычайно популярен. Созданный на базе языка Паскаль специалистами фирмы Borland язык Delphi, обладая мощностью и гибкостью языков Си и Си++, превосходит их по удобству и простоте интерфейса при разработке приложений, обеспечивающих взаимодействие с базами данных и поддержку различного рода работ в рамках корпоративных сетей и сети Интернет.

8. **Ява** (Java); платформенно-независимый язык объектно-ориентированного программирования, чрезвычайно эффективен для создания интерактивных веб-страниц. Этот язык был создан компанией Sun в начале 90-х годов на основе Си++. Он призван упростить разработку приложений на основе Си++ путем исключения из него всех низкоуровневых возможностей.

9. **C#** (произносится си шарп) – объектно-ориентированный язык программирования. Разработан в 1998–2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-

334 и ISO/IEC 23270. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java.

С тех пор язык сильно вырос в плане популярности и стал чуть ли не самым предпочитаемым языком среди разработчиков Windows- и Web-приложений, которые используют .NET Framework. Отчасти привлекательность языка C# связана с его понятным синтаксисом, который происходит от синтаксиса C/C++, но упрощает некоторые вещи. Несмотря на это упрощение, язык C# обладает той же мощью, что и C++.

10. **Оберон** (Oberon) – язык общего назначения, созданный автором Pascal и Modula-2 Никлаусом Виртом (Niklaus Wirth) и его коллегами из Швейцарского федерального технического института г. Цюрих (ETH Zurich) в ходе разработки одноименной операционной системы для однопользовательской рабочей станции Ceres. Язык и операционная система названы именем одного из спутников планеты Уран – Оберона. Имеет долгую историю создания, является наследником Algol 60 (1960), Pascal (1970) и Modula (1979). Oberon синтезировал более четверти века исследований Н.Вирта по методологии и языкам программирования. Ему с учениками удалось добиться точного синтеза «старых» достижений структурного и модульного программирования с «новыми» объектными методами. Вот, что говорил сам Вирт о своем «детище»: «Он (Oberon) включает в себя средства, необходимые для объектно-ориентированного программирования, сохраняя стиль Паскаля, и является результатом моего стремления к простоте без потери выразительности. В этом должна состоять сущность языка, равно пригодного как для учебной аудитории, так и для профессиональной деятельности. » Неслучайно, что в качестве эпиграфа к сообщению о языке Oberon Н. Вирт выбрал высказывание А.Эйнштейна: «Сделай так просто, как возможно, но не проще того». Наращивание мощи языка без его усложнения –

принцип, которому неуклонно следует Н. Вирт. В 1992 году сотрудничество Н.Вирта с Ханспетером Мёссенбёком (Hanspeter Mössenböck) привело к добавлению в язык ряда новых средств. Новая версия получила название Oberon-2. Оберон-2 представляет собой почти правильное расширение Оберона и является фактическим стандартом языка, который поддерживается большинством современных Оберон-систем.

11. **Лисп** (Lisp) – функциональный язык программирования. Ориентирован на структуру данных в форме списка и позволяет организовать эффективную обработку больших объемов текстовой информации.

12. **Пролог** (PROgramming in LOGic – логическое программирование). Главное назначение языка – разработка интеллектуальных программ и систем. Пролог – это язык программирования, созданный специально для работы с базами знаний, основанными на фактах и правилах (одного из элементов систем искусственного интеллекта). В языке реализован механизм возврата для выполнения обратной цепочки рассуждений, при котором предполагается, что некоторые выводы или заключения истинны, а затем эти предположения проверяются в базе знаний, содержащей факты и правила логического вывода. Если предположение не подтверждается, выполняется возврат и выдвигается новое предположение. В основу языка положена математическая модель теории исчисления предикатов.

13. **HTML** (HyperText Markup Language). Общеизвестный язык для оформления документов. Не является алгоритмическим языком программирования, а язык разметки гипертекста. Очень прост, содержит элементарные команды форматирования текста, добавления рисунков, задания шрифтов и цветов, организации ссылок

и таблиц. Все Web-страницы написаны на языке HTML или используют его расширения.

14. **PHP** – скриптовый язык программирования, применяющийся для создания сайтов. Важное его достоинство языка php – это создания динамических веб-сайтов, работа с базами данных (mysql).

Методологии программирования

Большая часть работы программистов связана с написанием исходного кода, тестированием и отладкой программ на одном из языков программирования. Исходные тексты и исполняемые файлы программ являются объектами авторского права и являются интеллектуальной собственностью их авторов и правообладателей.

Различные языки программирования поддерживают различные стили программирования (методологии или парадигмы программирования). Отчасти искусство программирования состоит в том, чтобы выбрать один из языков, наиболее полно подходящий для решения имеющейся задачи.

Разные языки требуют от программиста различного уровня внимания к деталям при реализации алгоритма, результатом чего часто бывает компромисс между простотой и производительностью (или между временем программиста и временем пользователя).

Методология программирования – это совокупность идей, понятий, принципов, способов и средств, определяющая стиль написания, отладки и сопровождения программ.

Перечислим основные методологии (парадигмы) программирования вместе с присущими им видами абстракций:

- процедурно-ориентированные – алгоритмы;
- объектно-ориентированные – классы и объекты;

– логически-ориентированные – цели, выраженные в исчислении предикатов, правила «если…, то…»;
– параллельное программирование – потоки данных.

Существуют и другие парадигмы.

Структурное программирование

Классическое **процедурное программирование** требует от программиста **детального описания** того, как решать задачу, т. е. формулировки **алгоритма** и его специальной записи. При этом ожидаемые свойства результата обычно не указываются. **Основные понятия** языков этих групп – **оператор** и **данные**.

При процедурном подходе операторы объединяются в группы – процедуры. **Структурное программирование** в целом не выходит за рамки этого направления, оно лишь дополнительно фиксирует некоторые полезные приемы технологии программирования.

Структурное программирование, наиболее отчетливо выраженное в языке Паскаль (PASCAL), возникло в ходе развития процедурно-ориентированного подхода, заложенного в исторически первом из языков программирования Фортране (FORTRAN).

Основу технологии **структурного программирования** составляют следующие **положения**:

1. **Сложная задача разбивается на более мелкие**, функционально лучше управляемые **задачи**. Каждая задача имеет один вход и один выход. В этом случае управляющий поток программы состоит из совокупности элементарных подзадач с ясным функциональным назначением.
2. Простота управляющих структур, используемых в задаче. Логическая структура программы может быть выражена комбинацией **трех базовых структур: следования, ветвления и цикла**.

3. Детально проработанные алгоритмы изображаются в виде **блок-схемы**.
4. Используя базовые структуры, можно полностью **исключить использование безусловного перехода**, что является важным признаком структурного программирования

Три базовые структуры были рассмотрены в разделе «Основы алгоритмизации» (линейный алгоритм, разветвленный алгоритм, циклический алгоритм). Здесь лишь напомним их.

Следование – самая важная из структур. Она означает, что действия могут быть выполнены друг за другом

Ветвление – это структура, обеспечивающая выбор между двумя альтернативами. Выполняется проверка, а затем выбирается один из путей.

Эта структура называется также «ЕСЛИ – ТО – ИНАЧЕ», или «развилка». Каждый из путей (ТО или ИНАЧЕ) ведет к общей точке слияния, так что выполнение программы продолжается независимо от того, какой путь был выбран.

Может оказаться, что для одного из результатов проверки ничего предпринимать не надо. В этом случае можно применять только один обрабатывающий блок (структура «ЕСЛИ – ТО»).

Цикл «Пока» (цикл с предусловием) начинается с проверки логического выражения. Если оно *истинно*, то выполняется тело цикла, затем все повторяется снова, пока логическое выражение сохраняет значение «истина». Как только оно становится *ложным*, управление передается по программе дальше (см. рис. 1.6).

В цикле «До» (цикл с постусловием) проверка условия выполняется после операторов тела цикла. Цикл повторяется, если условие *ложно*. Как только оно становится *истинным*, управление передается по программе дальше (см. рис. 1.7).

Пожалуй, самым важным достижением **структурного подхода** к разработке алгоритмов является **нисходящее проектирование программ**.

Этот метод основан на идее уровней абстракции, которые становятся уровнями **модулей** в разрабатываемой программе. Это позволяет программисту сначала сконцентрировать внимание на определении того, что надо сделать в программе, а лишь затем решать, как это надо делать. При нисходящем проектировании исходная, подлежащая решению задача разбивается на ряд подзадач, подчиненных по своему содержанию главной задаче. Такое разбиение называется детализацией или декомпозицией.

На следующем этапе эти задачи, в свою очередь, разбиваются на более мелкие подчиненные подзадачи и так далее, до уровня относительно небольших подзадач, которые требуют для решения небольших модулей.

Модуль – это последовательность логически связанных операций, оформленных как отдельная часть программы. Модули связаны между собой только по входным и выходным данным.

Использование модулей имеет следующие преимущества:

1) возможность создания программы несколькими программистами;

2) простота проектирования и последующих модификаций программы;

3) упрощение отладки программы – поиска и устранения в ней ошибок;

4) возможность использования готовых библиотек наиболее употребительных модулей.

Сегодня дополняющим структурное программирование, создающим основу для разработки современных программных комплексов стало объектное **объектно-ориентированное**

программирование, а противостоящим ему при решении определенных классов задач является **декларативное программирование**, выраженное двумя разными подходами – **функциональным** и **логическим**.

Объектно-ориентированное программирование

Концепция ООП возникла в середине 80-х годов. Главная ее идея в том, что программное приложение, как и окружающий нас мир, должно состоять из **объектов**, обладающих собственными свойствами и поведением. Каждый объект объединяет данные и процедуры обработки этих данных и относится к определенному классу. Объектно-ориентированный подход основан на:

- выделении классов объектов;
- установлении свойств и методов обработки;
- создании иерархии классов, наследовании свойств объектов и методов их обработки.

ООП позволяет резко сократить объем и трудоемкость разработки программ, имеющих дело с множеством связанных друг с другом объектов.

При объектно-ориентированном подходе программные задачи распределяются между объектами программы. Объекты обладают определенным набором свойств, методов и способностью реагировать на события (нажатие кнопок мыши, интервалы времени и т.д.).

В отличие от процедурного программирования, где порядок выполнения операторов программы определяется порядком их следования и командами управления, **в ООП порядок выполнения процедур и функций определяется**, прежде всего, **событиями**.

Чтобы проект можно было считать объектно-ориентированным, объекты должны удовлетворять некоторым требованиям. Этими требованиями являются инкапсуляция, наследование и полиморфизм.

Инкапсуляция – это механизм, который объединяет данные и код, манипулирующий этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования. В объектно-ориентированном программировании код и данные могут быть объединены вместе; в этом случае говорят, что создаётся так называемый «черный ящик». Когда коды и данные объединяются таким способом, создаётся объект.

Наследование – свойство объектов порождать своих «потомков», означает, что новый объект можно определить на основе уже существующих объектов, при этом он будет содержать все свойства и методы родительского. Наследование полезно, когда требуется создать новый объект, обладающий дополнительными свойствами по сравнению со старым.

Полиморфизм – многие объекты могут иметь одноименные методы, которые могут выполнять разные действия для разных объектов. Например, оператор «+» для числовых величин выполняет сложение, а для текстовых – склеивание. Выполнение каждого конкретного действия будет определяться типом данных.

В ООП центральным является понятие класса. **Класс** – это шаблон, по которому создаются объекты определенного типа. Класс объединяет в себе данные и методы их обработки.

Объекты – это экземпляры определенного класса. Например, кнопки или текстовые поля, устанавливаемые на форме являются экземплярами соответствующих стандартных классов.

Декларативное программирование

Направлен на относительно узкий круг задач искусственного интеллекта.

Программист описывает свойства исходных данных, их взаимосвязи, свойства, которыми должен обладать результат, а не алгоритм получения результата.

Разумеется, для получения результата этот алгоритм все равно нужен, но он должен порождаться автоматически той системой, которая поддерживает декларативно-ориентированный язык программирования.

При **логическом** варианте такого подхода (прежде всего это относится к языку *Пролог*, *PROLOG*) задача описывается совокупностью фактов и правил в некотором формальном логическом языке, при **функциональном** варианте – в виде функциональных соотношений между фактами (язык *Лисп*, *LISP*).

Языки логического программирования базируются на классической логике и применимы для систем логического вывода, в частности, для так называемых **экспертных систем**.

На языках логического программирования естественно формализуется логика поведения, и они применимы для описаний правил принятия решений.

Важным преимуществом такого подхода является достаточно высокий уровень машинной независимости, а также возможность откатов – возвращения к предыдущей подцели при отрицательном результате анализа одного из вариантов в процессе поиска решения (скажем, очередного хода при игре в шахматы), что избавляет от необходимости поиска решения путем полного перебора вариантов и увеличивает эффективность реализации.

Функциональный подход к программированию появился в результате проведения фундаментальных математических исследований.

Важнейшей характеристикой функционального подхода является то обстоятельство, что всякая программа, разработанная на языке

функционального программирования, может рассматриваться как функция, аргументы которой, возможно, также являются функциями.

Функциональный подход породил целое семейство языков, родоначальником которых стал язык программирования *LISP*.

Параллельное программирование

Используется для распараллеливания обработки информации в многопроцессорных и мультипрограммных ЭВМ с целью ускорения вычислений и эффективного использования ресурсов ЭВМ.

В отличие от программирования последовательных вычислений, концептуальную основу которого составляет понятие алгоритма, реализуемого по шагам строго последовательно во времени, в параллельном программировании программа порождает совокупность параллельно протекающих процессов обработки информации, полностью независимых или связанных между собой статическими или динамическими пространственно-временными или причинно-следственными отношениями.

Контрольные вопросы

1. Что такое системы программирования и к какому классу программ они относятся?
2. Что входит в состав систем программирования?
3. На каком языке программирования создавались первые программы?
4. Приведите классификацию языков программирования.
5. Охарактеризуйте языки низкого уровня. Приведите пример языка низкого уровня. Достоинства языков низкого уровня.
6. Охарактеризуйте языки высокого уровня. Назовите языки высокого уровня.
7. Для чего предназначены трансляторы? Назовите отличие компиляции от интерпретации.

8. Недостатки интерпретации (как вид транслятора).
9. Какие действия выполняются при компиляции?
10. Особенность декларативных языков.
11. Охарактеризуйте кратко языки программирования: Фортран, Бейсик, Паскаль, Оберон.
12. Охарактеризуйте кратко языки программирования: Ада, Си, Си++, Си#, Delphi, Java.
13. Приведите примеры объектно-ориентированных языков.
14. Перечислите методологии программирования.
15. Назовите положения структурного программирования.
16. Назовите и зарисуйте три базовые структуры структурного программирования.
17. На чем основано нисходящее проектирование? Что вы можете сказать о концепции модульного проектирования? Перечислите преимущества использования модулей.
18. Что такое объект в объектно-ориентированном программировании? Три принципа объектно-ориентированного программирования.
19. Особенность декларативного программирования.
20. Что такое параллельное программирование?

Раздел 3. Разработка программ для компьютера

Этапы создания программ

Программа – это последовательность команд компьютера, приводящая к решению задачи. Программа является результатом интеллектуального труда, для которого характерно творчество.

Хотя программирование в значительной степени искусство, тем не менее можно систематизировать и обобщить накопленный профессиональный опыт. Проектирование и разработку программ целесообразно разбить на ряд последовательных этапов:

1) постановка задачи;
2) проектирование программы;
3) построение модели;
4) разработка алгоритма;
5) написание программы;
6) отладка программы;
7) тестирование программы;
8) документирование.

Кратко остановимся на каждом из этих этапов.

Чтобы приступить к решению задачи необходимо **точно ее сформулировать**. В первую очередь, это **означает определение исходных и выходных данных**, т. е. ответы на вопросы: а) что дано; б) что нужно найти. Дальнейшая детализация постановки задачи представляет собой ответы на серию вопросов такого рода:

- как определить решение;
- каких данных не хватает и все ли они нужны;
- какие сделаны допущения и т. п.

Таким образом, кратко можно сказать, что на этапе **постановки задачи** необходимо:

– описание исходных данных и результата;
– формализация задачи;

– описание поведения программы в особых случаях (если таковые есть).

В ходе этой работы выявляются свойства, которыми должна обладать система в конечном виде (замысел), описываются функции системы, характеристики интерфейса.

Проектирование программы. Сначала производится проектирование архитектуры программной системы. Следующим шагом является детальное проектирование. На этом этапе происходит процедурное описание программы, выбор и оценка алгоритма для реализации каждого модуля. Входной информацией для проектирования являются требования и спецификации системы.

Для проектирования программ существуют различные подходы и методы. Современный подход к проектированию основан на декомпозиции, которая, в свою очередь, основана на использовании абстракции. Целью при декомпозиции является создание модулей, которые взаимодействуют друг с другом по определенным и простым правилам. Декомпозиция используется для разбиения программы на компоненты, которые затем могут быть объединены.

Построение модели в большинстве случаев является непростой задачей. Чтобы приобрести опыт в моделировании, необходимо изучить как можно больше известных и удачных моделей.

При построении моделей, как правило, используют два принципа: дедуктивный (от общего к частному) и индуктивный (от частного к общему).

Разработка алгоритма – самый сложный и трудоемкий процесс, но и самый интересный в творческом отношении. Выбор метода разработки зависит от постановки задачи, ее модели.

При построении алгоритма для сложной задачи используют системный подход с использованием декомпозиции (нисходящее

проектирование сверху-вниз) и синтеза (программирование снизу-вверх).

Одним из системных методов разработки алгоритмов является структурное программирование, которое мы рассмотрели в предыдущем разделе.

На этапе **написания программы** по разработанному алгоритму на выбранном языке программирования составляется программа.

Отладка программы – это процесс обнаружения и исправления ошибок. Программные ошибки можно разделить на два класса: ***синтаксические*** (синтаксис языка программирования) и ***алгоритмические*** (логические). Синтаксические ошибки выявляются в процессе компилирования программы – это наиболее простые с точки зрения исправления ошибки. Алгоритмические ошибки программы выявить гораздо труднее: программа работает, а результат выдает неправильный. Для обнаружения ошибок этого класса требуется этап тестирования программы.

Тестирование – это процесс исполнения программ с целью выявления (обнаружения) ошибок.

Существуют различные способы тестирования программ.

Тестирование программы как «черного ящика» (стратегия «черного ящика» определяет тестирование с анализом входных данных и результатов работы программы). Критерием исчерпывающего входного тестирования является использование всех возможных наборов входных данных.

Тестирование программы как «белого ящика» заключается в стратегии управления логикой программы, позволяет использовать ее внутреннюю структуру. Критерием выступает исчерпывающее тестирование всех маршрутов и управляющих структур программы.

Разумная и реальная стратегия тестирования – сочетание моделей «черного» и «белого ящиков».

При проектировании процедуры тестирования предусматривают серии тестов, имеющих наивысшую вероятность обнаружения большинства ошибок. Для целей исчерпывающего тестирования создают эквивалентные разбиения входных параметров, причем предусматривают два класса: правильные входные данные и неправильные (ошибочные входные значения). Для каждого класса эквивалентности строят свой тест. Классом эквивалентности тестов можно назвать такое множество тестов, что выполнение алгоритма на одном из них гарантирует аналогичный результат прогона для других.

Из всего выше сказанного следует, что тестирование заключается в составлении наборов тестов (входные данные – ожидаемый результат), которые бы охватывали все ветки прохождения алгоритма.

Есть золотое правило программистов – оформляй свои программы в том виде, в каком бы ты хотел видеть программы, написанные другими. К каждому конечному программному продукту необходимо **документированное сопровождение** в виде помощи (help), файлового текста (readme.txt).

Контрольные вопросы

1. Перечислите и охарактеризуйте этапы создания программ.
2. Что представляет собой декомпозиция?
3. Что такое отладка программы?
4. Какие классы программных ошибок вы знаете и когда они выявляются?
5. Назначение тестирования программы.
6. Какие способы тестирования вы знаете?

Раздел 4. Основы структурного языка программирования Паскаль

В 1970 г. профессор Никлаус Вирт из Швейцарии обосновал и разработал язык высокого уровня – *Паскаль*. Этот язык отличается простотой и стройностью, качествами, которые обеспечивают Паскалю популярность уже на протяжении нескольких десятилетий.

Паскаль – это процедурный язык, имеющий блочную структуру. Набор операторов языка отражает принципы структурного программирования.

Паскаль-программа является текстовым файлом с собственным именем и с расширением *pas*.

Структура программы

В целом программа на языке Паскаль состоит из двух основных частей: описания всех данных, с которыми производятся действия и описания самих действий. Кроме этого, в самом начале программы может присутствовать ее название – заголовок. В самом конце программы ставиться точка «.».

Правила языка Паскаль предусматривают единую для всех программ форму основной структуры:

ЗАГОЛОВОК	**program <Имя программы>;** Содержит служебное слово **program;**
РАЗДЕЛ ОПИСАНИЙ	**Раздел** внешних модулей, процедур и функций **– Uses** *Пример:* **uses** Crt; **Раздел констант – const** **Константа** – переменная, которая не изменяется в процессе выполнения программы. Под константу не выделяется память. Тип константы определяется ее значением. *Пример:* **const** N=10; p=0.14;

РАЗДЕЛ ОПИСАНИЙ	**Раздел типов – type** В Паскале существуют стандартные типы, которые описывать не надо (они считаются уже описанными: integer, real, char, Boolean и т.д.) В Паскале есть возможность создать свой новый тип данных. **Раздел переменных – var** **Переменная** – это величина способная изменяться в процессе выполнения программы. Под переменную выделяется память. Каждая переменная, до работы с ней должна быть описана, т.е. указан тип переменной. Попытка в процессе выполнения программы присвоить переменной значение иного типа расценивается как ошибка в программе. *Пример*: **var** I, j, r: integer; X, h, sum: real; D, f, r: string; **Раздел процедур и функций – procedure** и **function** Раздел процедур и функций не начинается каким-то специальным служебным словом – начало данного раздела легко определяется по служебным словам **procedure** или **function**.
БЛОК ОСНОВНЫХ ОПЕРАТОРОВ	**begin** Оператор 1; Оператор 2; Оператор N **end.** Это основной раздел программы – именно здесь задаются те действия, которые должны быть выполнены в данной программе.

Имена программы и используемых величин (констант, переменных) выбираются программистом самостоятельно в соответствии с правилами построения идентификаторов:

• идентификатор должен быть уникальным, то есть одним и тем же именем разные объекты не могут быть названы;

• идентификатор имеет ограничение по длине (зависит от конкретной реализации языка на компьютере);

• идентификатор может состоять только из символов латинского алфавита, цифр и знака подчеркивания («_»);

• идентификатор не может начинаться с цифры.

Данные. Типы данных

Совокупность величин, с которыми работает компьютер, принято называть **данными**. У всякой величины имеются три основных свойства: имя, значение и тип. В алгоритмах и языках программирования величины делятся на **константы** и **переменные**

Константа – это величина, которая не изменяется в процессе выполнения программы. Под константу не выделяется память. Тип константы определяется ее значением. Объявляется в разделе ***Const*** (примеры смотри в следующем разделе).

Переменная – это величина способная изменяться в процессе выполнения программы. Под переменную выделяется память. Каждая переменная, до работы с ней должна быть описана в разделе ***Var***, т.е. указан тип переменной (примеры смотри в следующем разделе).

Все данные характеризуются своим **типом**.

Тип данных определяет множество значений, допустимых для переменной, операции выполняемые на этих значениях, количество выделяемой памяти. То есть переменная может принимать только значения, определяемые ее типом и участвовать только в тех операциях, которые допустимы для этого типа.

В языке Паскаль тип величины задают заранее. Все переменные, используемые в программе, должны быть объявлены в разделе описания с указанием их типа. Обязательное описание типа приводит к избыточности в тексте программ, но такая избыточность является важным вспомогательным средством разработки программ и

рассматривается как необходимое свойство современных алгоритмических языков высокого уровня.

В языке Паскаль существует пять простых типов данных:

Integer	**Целочисленные** данные, во внутреннем представлении занимают 2 байта
Real	**Вещественные** данные, занимают 6 байтов
Char	**Символ**, занимает 1 байт
String	**Строка** символов, занимает *MAX+1* байт, где *MAX* – максимальное число символов в строке
Boolean	**Логический** тип, занимает 1 байт и имеет два значения: false (ложь) и true (истина)

Целый и вещественный тип имеют подтипы. В таблице приведем подтипы целого типа:

Название	Кол-во памяти (байт)	Диапазон значений
Byte	1	0 … 255
ShortInt	1	-128… +127
Word	2	0 … 65535
Integer	2	-32768 … +32767
LongInt	4	-2147483648 … +2147483647

Над данными **целого типа** определены следующие арифметические операции:

Знак операции	Назначение	Приоритет
+	Сложение	2
-	Вычитание	2
*	Умножение	1
/	Деление	1
div	Целая часть от деления	1
mod	Остаток от деления	1

Результат выполнения этих операций над целыми операндами получается также целого типа (исключение составляет операция / – **результат всегда вещественное число**).

Над данными целого типа определены следующие операции отношения: =, <>, <, >, <=, >=. Результат выполнения этих операций – логический тип.

Приоритет – это последовательность выполнения действий в строке операций. Если приоритет = 1, то эти действия выполняются в первую очередь, если приоритет = 2, то эти действия выполняются во вторую очередь. Для изменения приоритета используются круглые скобки.

Пример выполнения операций div и mod:

7 div 2 = 3

3 div 5 = 0

7 mod 2 = 1

3 mod 5=3

Список стандартных функции, дающие **целый результат**:

Функция	**Тип аргумента**	**Назначение**
Abs(x)	X – целое	Абсолютная величина X
Sqr (x)	X – целое	Возведение X в квадрат
Trunc (x)	X – веществ.	Выделение целой части числа X
Round (x)	X – веществ.	Округление X до целого числа
Succ (x)	X – целое	Следующее за X число
Pred (x)	X – целое	Предыдущее перед X число
Random (x)	X – целое	Случайное число от 0 до x-1 .Если функция не содержит аргумента, то генерируется случайное число от 0 до 1.
Randomize		Оператор, позволяющий генерировать новую последовательность случайных чисел при новом запуске программы

Вещественные числа могут быть представлены в форме с фиксированной точкой и в форме с плавающей точкой:

С фиксированной точкой	**С плавающей точкой**	
	Математическая. запись	Запись на языке Паскаль
5600	$0.56 * 10^{4}$	0.56E+04
-0.023	$-23 * 10^{-3}$	-23E-03
570	$0.57 * 10^{3}$	0.57E+03
0.26	$26 * 10^{-2}$	26E-02
-0.003	$-3 * 10^{-3}$	-3E-03

Над данными вещественного типа определены следующие арифметические операции:

Знак операции	**Назначение**	**Приоритет**
+	Сложение	2
-	Вычитание	2
*	Умножение	1
/	Деление	1

Операции div и mod над вещественными величинами не допустимы!

Список стандартных функции, дающие **вещественный результат**:

Математическая запись	Запись на языке Паскаль	Назначение
sin x	**sin (x)**	Синус числа X
cos x	**cos (x)**	Косинус числа X
arctg x	**arctan (x)**	Арктангенс числа X
ln x	**ln (x)**	Натуральный логарифм числа X
$\mathbf{e^x}$	**exp (x)**	Экспонента числа X
$\sqrt{x}$	**sqrt (x)**	Корень квадратный числа X

Функция *ln (x)* и *exp (x)* используются для возведения в степень по правилу: $x^n = e^{n\ln(x)}$. Например, выражение x^9 вычисляется по формуле *exp(9*ln(x))*.

Логический тип данных имеет всего два значения **True** (истина), **False** (ложь) и является упорядоченным типом **True > False**.

В программе логический тип переменной задается служебным словом **Boolean**.

Существуют следующие **логические операции**:

1. ***Операции сравнения:***

   > – больше;

   < – меньше;

   = – равно;

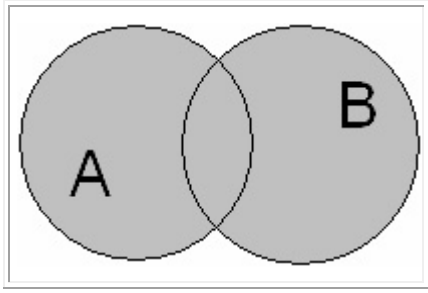
   <> – не равно;

   >= – больше либо равно;

   <= – меньше либо равно.

2. or (или) – логическое сложение (дизъюнкция).

В физическом смысле логическое сложение – это объединение двух областей.

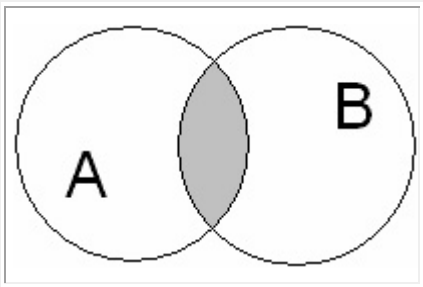


A =True	B =False
A **or** A = True A **or** B = True B **or** A = True B **or** B = False	

Логическое сложение дает ложный результат только в том случае, когда оба операнда ложные.

3. and (и) – логическое умножение (конъюнкция).

В физическом смысле логическое умножение – это пересечение двух областей.

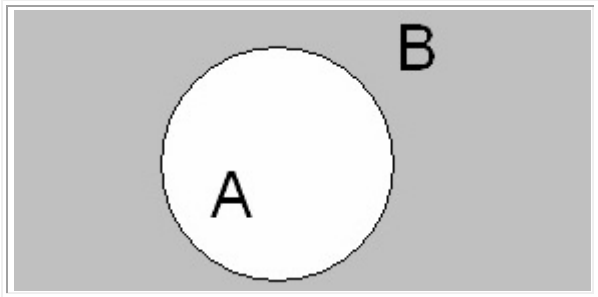


A =True	B =False
A **and** A = True A **and** B = False B **and** A = False B **and** B = False	

Логическое умножение дает истинный результат только в том случае, когда оба операнда истинны.

4. not (не) – логическое отрицание.

Логическое отрицание – это унарная операция, то есть операция, выполняемая над одним операндом.



A =True	B =False
not A = B **not** B = A	

Операция отрицания дает ложный результат, если операнд истинный, и, наоборот, истинный результат, если операнд ложный.

Практикум

Задание 1. Определите тип величины, если её значение равно:

1) 25;
2) 36,6;
3) 'нет';
4) 48,2;
5) 'число';
6) '29';
7) 't'
8) '147';
9) -27
10) -16,01.

Задание 2. Выберите значения, допустимые для величин целого типа.

1) -5
2) 3,7
3) 38
4) 'три'
5) 20,2
6) '23'
7) 6,0
8) 589

Задание 3. Выберите значения, допустимые для величин вещественного типа.

1) 'есть решение';
2) 4,65E-2;
3) 3,45;
4) 23,79;
5) -47;
6) '98E+6'
7) 29;
8) 87,05;

9) -946,9;
10) -99;
11) '38,762';
12) -0,007;
13) -2,4;
14) '27';

Задание 4. Сопоставьте величинам, подходящие им типы.

Величина	Тип
Число квартир в доме	real (вещественный)
Название месяца	string (строковый)
Название города	char (символьный)
Арифметический знак	real (вещественный)
Число p	string (строковый)
Масса Земли	integer (целый)

Задание 5. Определите тип для величин.

- Высота здания
- Число этажей в здании
- Количество игроков в команде
- Температура человека
- Название вида дерева
- Скорость машины
- Название фильма
- Кличка собаки
- Вес контейнера
- Количество книжных полок
- Результат прыжка в длину
- Год вашего рождения
- Марка автомобиля

Задание 6. Правильно ли определен тип величины?

- Количество тетрадей – тип Real;
- Кличка собаки – тип Char;
- Вес контейнера – тип Integer;
- Название хоккейной команды – тип String;
- Объем шара – тип Integer;
- Количество книжных полок – тип Integer.
- Название месяца – тип Char;
- Количество раб. дней в месяце – тип Integer;
- Скорость велосипеда – тип Real;
- Количество планет в солн. системе – тип Real;
- Название континента – тип Char;
- Площадь земной поверхности – тип Integer.
- Расстояние между городами – тип Real;
- Марка стали – тип Integer;
- Количество букв в слове – тип Char;
- Масса воды в сосуде – тип Integer;
- Количество слов в телеграмме – тип Integer;
- Порода собаки – тип String.

Задание 7. Для величины КОЛИЧЕСТВО СТРАНИЦ В КНИГЕ выберите допустимое значение:

1) 'двести';
2) 137,5;
3) -18;
4) 243;
5) 5;
6) -46,3;
7) 'девяносто';
8) 175;
9) 65,1;
10) -187
11) 567;
12) -148;
13) 'двадцать пять';
14) 1481;

15) 314,2; 16) 'Шесть'

Задание 8. Переменные X, У, Z имеют соответственно тип целый, вещественный, строковый. Из приведенных ниже значений укажите запись с допустимыми значениями для x, y, z:.

1) 15,3; 6; 'пять'
2) 38; 26,04; 'семь'
3) 'да'; 18; 10.3
4) 'нет'; 13; 'один'
5) 27,6; 13; 'один'
6) 3; 37,77; 'Азия'

Задание 9. Определите, является ли величина константой или переменной.

- Расстояние от дома до школы
- Время, за которое вы добираетесь до школы
- Количество дней в недели
- Количество дней в январе
- Количество дней в месяце
- Стоимость билета в кино
- Температура воздуха
- Скорость света
- Скорость ракеты
- Ваше имя

Задание 10. Сопоставить тип переменной, объем памяти, выделяемой для переменной этого типа и диапазон чисел, соответствующий этому типу:

ShorInt	1 байт	0…255
Integer	1 байт	0…65535
LongInt	2 байта	-32768…32767
Byte	2 байта	-2147483648… +2147483647
Word	4 байта	-128…127

Задание 11. Вычислите значение выражений там, где это возможно:

1) 20 div 6
2) 20 mod 6
3) 500 div 10
4) 500 mod 100
5) 2 div 5
6) 2 mod 5
7) 123 div 0
8) 2.0 mod 3

Задание 12. Запишите числа в форме с фиксированной точкой:

- 2.3E+05
- -1.5E+02
- 5.7E+00
- 4.4E-02

Задание 13. Запишите числа в форме с плавающей точкой:

- 35050
- 15.23
- -50.7
- 0.034

Выражения

Арифметические выражения

Выражения строятся из *операндов* (переменные, константы, вызовы функций), *знаков операций* и *круглых скобок*.

При составлении выражений необходимо знать следующие правила:

1) все выражение записывается в строку, то есть двухэтажные выражения, а также верхние и нижние индексы не допускаются. Например: *(a1*x1 - a2*x2)/(x1-x2)*;
2) в выражении можно использовать только круглые скобки.

Примеры арифметических выражений:

*b+a*x*

*sin(x)+2*x*

При составлении выражения следует учитывать приоритет операций:

1) *,/,div, mod
2) +,-

Для изменения приоритета используются круглые скобки.

Выражения используются для вычисления новых значений. Тип выражения определяется в зависимости от типов операндов, участвующих в выражении и входящих в него операций.

Правила определения типа выражения:

1) Если в выражении есть хотя бы один *вещественный операнд*, то выражение будет иметь *вещественный тип*.
2) Если все операнды в выражении *целого типа*, выражение будет иметь *целый тип*, **исключение** – наличие операции деления (/) – всегда вещественный тип.

Например, если переменная *x* имеет вещественный тип, а переменные *a* и *b* – целый тип, то тип выражений будет определен следующим образом:

*2*x+3* – вещественный тип (т.к. *x* – вещественный тип);

*2*a-3* – целый тип (т.к. все операнды – целые);

a/b+3 – вещественный тип (т.к., хотя все операнды – целые, но есть операция деления).

Логические выражения

При решении задач часто возникают ситуации, когда последующие действия зависят от выполнения некоторого условия, например, вычислять корни квадратного уравнения можно только в случае, когда дискриминант положителен. Для этого используется структура ветвления, которая реализуется в языке *Паскаль* условным оператором. В качестве условия такого оператора используется логическое выражение. Логическое выражение дает либо истинное, либо ложное значение (true, false).

В логическом выражении могут учувствовать несколько логических операций, приоритет выполнения операций следующий:

1) логическое отрицание;
2) конъюнкция;
3) дизъюнкция;
4) операции сравнения.

Для изменения очередности предназначены круглые скобки.

Примеры. Записать логические выражения, истинные при соблюдении следующих условий:

1. В волейбольную секцию примут детей не старше 13 лет и с ростом не ниже 160 см.

Ответ: *(x<=13) and (y>=160)*, где *x* – возраст ребенка, а *y* – его рост.

2. В кружок английского языка примут школьников младше 5 класса или старше 8 класса.

Ответ: *(x<5) or (x>8)*, где *x* – класс, в котором учится школьник.

Примеры записи логических выражений:

- (a>3) and (a<5) or (b>2) and (b<10)
- not (a<15) or (b>30)
- c or d and (b=10)

Приоритет всех операций (от высшего к низшему):

1) операции и функции в скобках;
2) not;
3) *, /, and, div, mod;
4) +, -, or;
5) =, <>, <, >, <=, >=.

Сведем основные математические функции в Паскале в одну таблицу:

	Функция	**Назначение**	**Тип аргумента**	**Тип функции**
1	ABS(X)	Вычисление абсолютного значения X	real integer	real integer
2	SQR(X)	Вычисление квадрата X (X*X)	real integer	real integer
3	SQRT(X)	Вычисление квадратного корня X	real integer	real real
4	COS(X)	Вычисление COS(X)	real integer	real real
5	SIN(X)	Вычисление SIN(X)	real integer	real real
6	ARCTAN(X)	Вычисление ARCTANG (X)	real integer	real real
7	EXP(X)	Вычисление экспоненты X	real integer	real real
6	LN(X)	Вычисление натурального логарифма X	real integer	real real
9	TRUNC(X)	Нахождение целой части X (отбрасывание дробной)	real, integer	integer integer
10	FRAC(X)	Выделение дробной части числа X	real integer	real real
11	INT(X)	Нахождение целой части X (меньшее число)	real integer	integer integer

12	ROUND(X)	Округление X в сторону ближайшего целого	real integer	integer integer
13	RANDOM(X)	Нахождение случайного числа от 0 до X-1	integer	integer
14	ODD(X)	TRUE, если X – нечетное, FALSE, если X – четное	integer	boolean

Практикум

Задание 1. Назовите константы и переменные в выражениях :

1) 2 + x – y – 1.7

*2) 2 * xy*

*3) 2 + a4 – 1 / 3 * z2*

*4) (1 / x+1 / y) * 0.2 / 0.5*

Задание 2. Записать на *Паскале* следующие выражения:

1) x^{-1}

2) x^{-2}

3) x^{100}

4) $x^{\sqrt{2}}$

5) x^4

6) x^{x^2}

7) 2^{1+x}

8) $\sqrt[3]{1+x}$

Задание 3. Записать на *Паскале* следующие выражения:

1) $\dfrac{x^2+y^2}{1-\dfrac{y}{2}}$

2) $1+\left|\dfrac{1}{x+y}\right|$

3) $\sqrt{1+\sqrt{|x|}}$

4) $\dfrac{a+b-1.7}{c+\dfrac{d}{e+f+0.5}}$

Задание 4. Укажите порядок выполнения операций при вычисления выражения:

*(x-1/2)*y-3/10+4/(5-x)*

Задание 5. Записать на *Паскале* следующие выражения:

1) *[(ax-b)x+c]x-d*
2) $\frac{ab}{c}+\frac{c}{ab}$
3) $\frac{x+y}{a_1}\cdot\frac{a_2}{x-y}$
4) $10^4\alpha-\frac{3}{5}\beta$

Задание 6. Вычислить выражение:

*24/(3*4)-24/3/4+24/3*4*

Задание 7. Записать на *Паскале* следующие выражения:

1) $(1+x)^2$
2) $\sqrt{1+x^2}$
3) $|a+bx|$
4) *sin8*
5) cos^2x^3
6) *tg x*
7) $\log_2\frac{x}{5}$
8) $arctg\ 10^3$

Задание 8. Записать на *Паскале* следующие выражения:

1) $\sqrt[8]{x^8+8^x}$
2) $\frac{xyz-3.3\left|x+\sqrt[4]{y}\right|}{10^7+\sqrt{\lg 4}}$
3) $e^{|x-y|}+\ln(1+e)\log_2 tg2$

Задание 9. Вычислить значения выражений:

1) *trunk(6.9)*
2) *round(6.9)*
3) *trunk(6.2)*
4) *round(6.2)*
5) *trunk(-1.8)*
6) *round(-1.8)*
7) *round(0.5)*
8) *round(-0.5)*

Задание 10. Вычислить значения выражений:

1) *3*7div2mod7/3-trunc(9.7)*
2) *succ(round(5/2)-pred(3))*

Задание 11. Определить тип выражений:

1) *1+0.0*
2) *20/4*
3) *sqr(4)*
4) *sqr(5.0)*
5) *sqrt(16)*
6) *sin(0)*
7) *succ(-2)*
8) *trunk(-3.14)*

Задание 12. Запишите логические выражения, определив их истинность при *x=3*, *y=10*, *z=-5*.

1) *y* – четное число;
2) *z* – неотрицательное число;
3) *x* больше y;
4) *x* в сумме с y больше 12;
5) разность *x* и *z* – отрицательное число;
6) произведение *y* и *x* – нечетное число;
7) произведение *y* и *z* – отрицательное число;
8) *y* и *z* имеют одинаковую четность.

Задание 13. Среди следующих выражений выберите логические:

1) *(x=3) or (x<>4)*
2) *2*x+5*
3) *x mod 2 =0*
4) *x div 5*

5) *x+y=10*
6) *x+y*
7) *(x+y<2) and x*
8) *(x>3) and (x<10)*

Задание 14. Укажите порядок выполнения операций при вычисления выражения:

*(x>=0) or true and (x=3) or (x*y<>4)*

Задание 15. Вычислить:

1) *not(odd(n))*, при *n=0*;
2) *t and (p mod 3=0)*, при *t=true*, *p=333*;
3) *(x*y<>0) and (y>x)*, при *x=2*, *y=1*;
4) *(x*y<>0) or (y>x)*, при *x=2*, *y=1*;
5) *f or not b*, при *a=false*, *b=true*.

Задание 16. Вычислить:

sqr(x)+ sqr(y) <= 4, при *x=0.3*, *y=-1.6*;

k mod 7 = k div 5-1, при *k=15*;

*odd(trunk(10*p))*, при *p=0.182*.

Задание 17. Запишите на *Паскале* выражения:

1) целое *k* делится на 7 нацело;
2) целое *x* не кратно 5.

Задание 18. Вычислить выражения при *a=true*, *b=false*:

1) a or b and not a;
2) (a or b) and not a;
3) not a and b;
4) not (a and b).

Задание 19. Запишите на *Паскале* выражения:

1) *x* принадлежит отрезку [0;1];
2) *x* лежит вне отрезка [0;1] ;
3) *x* принадлежит отрезку [2;5] или [-1;1] ;
4) *x* лежит вне отрезков [2;5] и [-1;1] ;
5) каждое из чисел *x*,*y*,*z* положительно;
6) хотя бы одно из чисел *x*,*y*,*z* положительно;
7) ни одно из чисел *x*,*y*,*z* не является положительным;
8) только одно из чисел *x*,*y*,*z* положительно.

Задание 20. Найдите значение выражения:

A mod (B div C) +4

1) при *A*=40, *B*=13, *C*=6 (4)
2) при *A*=17, *B*=42, *C*=13 (10)

Основные операторы языка

Оператор присваивания

Оператор используется, чтобы явно присвоить переменной результат вычисления выражения.

Формат оператора:

<Имя переменной> := <Выражение>;

Примеры:

```
S:=0;
Name:=’Оля’;
S:=S+1;
```

Выполнение оператора присваивания заключается в следующем: сначала вычисляется результат выражения, затем полученное значение записывается в переменную, имя которой стоит слева от знака присваивания.

Оператор присваивания считается верным, если тип выражения соответствует или может быть приведен к типу переменной слева от знака присваивания.

Переменной **вещественного** типа можно присвоить выражение **вещественного** или **целочисленного** типов. Переменной **целочисленного** типа можно присвоить значение выражения **только целочисленного** типа.

Например, если объявлены следующие переменные

```
Var
    i, n : integer;
    D : real;
```

то операторы присваивания:

i:=n/10; – неправильный,

i:=1.0; – неправильный,

d:=i; – правильный.

Если тип выражения не соответствует типу переменной, то компилятор выдает сообщение об ошибке.

Практикум

Задание 1. Переменной *y* присвоить дробную часть положительного числа *x*.

Задание 2. Если *y* – вещественная переменная, а *n* – целая, то какие из следующих операторов присваивания правильные, а какие нет и почему:

1) *y:=n+1*;
2) *n:=y-1*;
3) *n:=4.0*;
4) *y:=trunc(y)*;
5) *n:=n div 2*;
6) *y:=y mod 2*;
7) *n:=n/2*;
8) *n:=sqr(sqrt(n))*;

Задание 3. Поменять местами значения целых переменных *x* и *y*.

Задание 4. Какое значение будет иметь переменная *x* после выполнения операторов:

```
x:=10;
x:=x+3;
```

Задание 5. Записать оператор присваивания, который меняет знак у переменной *x*.

Задание 6. Записать хему равны значения переменных *x* и *y* после выполнения операторов:

```
x:=2;
y:=5;
x:=y;
y:=x;
```

Задание 7. Правильны ли следующие операторы присваивания? Ответ обосновать.

```
k:=k mod 3+k*cos(0);
x:=x*2 div 6 +x/4;
```

Задание 8. Найдите ошибку в программе:

```
program a1;
var
   a: integer;
   b: real;
begin
   a:=100;
   b:=2.5;
   writeln(a,b);
   a:=a+b/3.5;
```

```
    writeln(a)
end.
```

Задание 9. Найдите ошибку в программе:

```
program a1;
var
   x: integer;
   y: real;
   z: char;
begin
   z:=’100’;
   y:=2.5;
   x:=10.5;
   y:=x+y;
    writeln('x=',x, 'y=',y, 'z=',z)
end.
```

Задание 10. Какие из следующих последовательностей символов являются операторами присваивания:

1) a:=b;
2) a=c+1;
3) a:b-sqr(2);
*4) a*x+b:=0;*
5) z:=0;
6) z:=z+1;
7) z:=z+1,2;
8) y:=-y;
9) -y:=y

Задание 11. Пусть значения переменных *x* и *y* равны, соответственно, *0.3* и *–0.2*. Какие значения будут иметь эти переменные после выполнения операторов присваивания:

*1) x:=x+2*y; y:=y/2;*
2) y:=-y; x:=x+y; y:=y+1;

3) x:=1; y:=x+y.

Задание 12. Задать в виде оператора присваивания следующие действия:

1) переменной *z* присвоить значение, равное полусумме значений переменных *x* и *y*;
2) удвоить значение переменной *a*;
3) значение переменной *x* увеличить на *0.1*;
4) изменить знак значения переменной *t*.

Задание 13. Написать оператор присваивания, в результате выполнения которого переменная *y* получает значение, равное значению переменной *x*, возведенному в пятую степень.:

Задание 14. Две последовательности операторов отличаются только порядком следования операторов:

*u:=u+v; v:=2*v;*

и

*v:=2*v; u:=u+v.*

Верно ли, что для любых начальных значений *u* и *v* выполнение этих последовательностей операторов приводит к одинаковому изменению значений переменных *u* и *v*.

Задание 15. Какие значения будут получены в переменных *x* и *y* в результате выполнения последовательности операторов

x:=x+y;

y:=x-y;

x:=x-y;

если последовательность исходных данных была составлена из двух чисел: а) 3.5 и 2.4; б) 6.7 и –10.1

Задание 16. Даны *x*, *y*, *z*. Записать оператор присваивания для вычисления переменных *a* и *b*

а) $a = \dfrac{1+\sin^2(x+y)}{2+|x-2x/(1+x^2y^2)|}+x,\ b=\cos^2(arctg\dfrac{1}{z})$;

б) $a=\ln|(y-\sqrt{|x|})(x-\dfrac{y}{z+x^2/4})|,\ b=x-\dfrac{x^2}{3!}+\dfrac{x^5}{5!}$.

Составной оператор

Этот оператор, строго говоря, оператором не является. Дело в том, что также как арифметические действия иногда необходимо заключать в скобки, последовательности команд (операторов) тоже иногда требуют объединения. Это позволяют сделать так называемые операторные скобки.

Формат оператора:

```
Begin
    <Оператор 1>;
    <Оператор 2>;
    .....
    <Оператор N>
End;
```

Составной оператор предоставляет возможность выполнить произвольное количество команд там, где подразумевается использование только одного оператора. Такая необходимость встречается довольно часто.

Оператор ввода

Оператор ввода позволяет вводить данные в переменные во время выполнения программы с клавиатуры. Элементами списка ввода являются переменные, которые должны быть заполнены значениями, введенными с клавиатуры.

Формат оператора:

Read(<Список ввода>);

Readln(<Список ввода>);

Например,

read(x,y);

Выполнение оператора ввода происходит следующим образом: ход программы приостанавливается, на экран выводится курсор, компьютер ожидает от пользователя набора данных для переменных, имена которых указаны в списке ввода. Пользователь с клавиатуры вводит необходимые значения в том порядке, в котором они требуются списком ввода, нажимает клавишу Enter.

После этого набранные данные заносятся в соответствующие им переменные, и выполнение программы продолжается.

Практикум

Задание 1. Какие из следующих последовательностей символов являются операторами ввода:

1) read(x, y, z);
2) read x, y, z; в);
3) read(x);
4) read(a; b);
5) x:=read(x);
6) read(a, b+c)

Задание 2. Какие значения будут иметь переменные *x* и *y* в результате выполнения последовательности операторов, если последовательность исходных данных была составлена из чисел *5.2* и *18.7.*

read(x, y);

t:=x;

x:=y;

y:=t;

Оператор вывода

Оператор вывода позволяет выводить на экран монитора данные из списка вывода.

Формат оператора:

Write(<Список вывода>);

Writeln(<Список вывода>);

Например:

Write(‘Результат=’, Pi*r*r);

Элементами списка вывода могут являться имена переменных, констант, выражения (которые предварительно вычисляются), текст, заключенный в апострофы. Элементы списка, также как и в операторах ввода, разделяются запятыми.

Различие между операторами Write и Writeln: после выполнения оператора Writeln (от Write line) курсор переходит на новую строчку, а после выполнения оператора Write курсор остается на текущей строке вывода, и следующий оператор вывода начинает вывод именно с этой позиции.

Оператор Writeln без параметров (Writeln;) реализует переход к началу следующей строки.

После имени переменной или выражения через двоеточие можно указать **формат числа**, который задает ширину поля вывода. Для переменной целого типа – это одно число, указывающее число позиций, отводимых под целое значение (например, write(x:10);). Для переменной **вещественного типа** – это два числа, **первое** из которых указывает число позиций, отводимых под **все число**, включая десятичную точку, **второе – число позиций, отводимых под дробную часть числа** (например, write(y:7:2);). Если для вещественных величин формат вывода не задан, то значение выводится в форме с плавающей точкой.

Приведем примеры использования форматированного вывода.

1) Пусть имеется фрагмент программы:

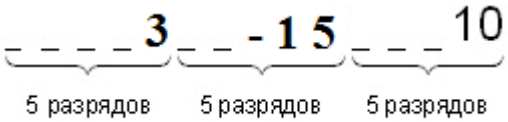
a := 3;

b := -15;

c := 10;

Write (a : 5, b : 5, c : 5);

На экране монитора данные будут расположены таким образом:



2) Пусть имеется фрагмент программы:

a := 52.6;

Write (a);

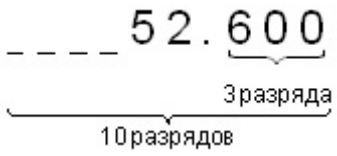
На экране будет отражена запись 5.2600000000E+01, т.е. $5.2600000000*10^{1}$

Для вывода значения вещественного типа с фиксированной точкой используем форматированный вывод:

a := 52.6;

Write (a : 10 : 3);

После выполнения оператора вывода на экране будет отражена запись:



Сведем полученные знания в таблицу:

Целый тип		**Вещественный тип**	
Неформатный вывод	Форматный вывод	Неформатный вывод	Форматный вывод
Var **A,B,C:Integer;** **A:=1; B:=2;** **C:=3;** **Write(A,B,C);** **На экране: 123**	WRITE(A:n,B:m,C:k), где n,m,k – ширина поля вывода Write(A:3,B:3,C:3); На экране: _ _1_ _2_ _3	Var A:Real; A:=12.56; Write (A); На экране: 1.2560000000E+01	WRITE(A:n:m), где n – ширина поля вывода, m-количество знаков после запятой Write (A:7:2); На экране: _ _12.56

Практикум

Задание 1. Какие из следующих последовательностей символов являются операторами вывода:

1) write(x, y);
2) write(x, x+1, x+2);
3) read(a);
4) write(100);
5) print(y, z);
6) print y, z
7) write (x; y; z);
8) write(x+2.2);
9) write(x+2,2)

Задание 2. Какие числа будут выведены в результате выполнения последовательности операторов:

read(x);

x:=x-1.2;

*x:=sqrt(x-1)*x+1;*

*write(x, 2-3*x);*

если в качестве исходного данного использовалось число:

а) 1.2; б) 2.2; в) 5.2; г) 10.2

Условный оператор

Условный оператор позволяет проверить некоторое условие и в зависимости от результатов проверки выполнить то или иное действие. Таким образом, условный оператор используется для реализации ветвлений в программе, которые происходят при выполнении некоторого условия.

Формат условного оператора:

If <условие>
Then <оператор 1>
Else <оператор 2>;

Работа оператора может быть выражена простыми словами:

Если <условие> {Если выполняется условие}
то < оператор 1> {то выполнить оператор № 1}
иначе < оператор 2> {иначе – выполнить оператор № 2}

Условный оператор работает по следующему алгоритму. Вначале вычисляется условное выражение, если результат вычисления имеет значение ИСТИНА, то выполняется оператор или группа операторов следующих за словом ***then***, если же выражение имеет значение ЛОЖЬ, то выполняется оператор следующий после слова ***else***.

Следует обратить ваше внимание на то что после <оператора 1> перед словом ***else*** не ставится точка с запятой.

Существуют конструкции когда условный оператор записывается без слова **else**, т.е.

If <условие> then <оператор 1>;

В таких случаях при вычислении логического выражения, когда результат имеет значение ЛОЖЬ, выполняется оператор следующий за оператором условия.

После слов ***then*** и ***else*** стоит только один оператор. Но что делать, если требуется по выполнению или невыполнению условия совершить не одно, а несколько действий? Здесь приходит на помощь

уже составной оператор. В операторные скобки можно заключить любое количество операторов.

Вариант условного оператора в этом случае:

If <условие>

Then Begin <группа операторов 1> end

Else Begin < группа операторов 2> end;

Пример. Заданы целые значения *x* и *y*. Определить $z = max (x^2, y^2)$.

```
Program primer1;
Var
   z, x, y : integer;
Begin
  Readln (x, y);
  If sqr(x) > sqr(y) then z := sqr(x)
                     else z := sqr(y);
 Write ('Z = ', z);
End.
```

Практикум

Задание 1 (с решением). Составить программу для вычисления составной (сложной) функции, имеющей различный вид на разных участках аргумента:

Функция	**Участки аргумента**
$y=x^3$	–2<=x<0
$y=x^2$	0<= x<1
y=4ln(x)	1<= x<1.5
$y=-10(x-1.6)^2$	1.5<= x<=2

Порядок выполнения в системе Pascal ABC (см. раздел «Система программирования Pascal ABC»)**:**

1. Активизируйте пункт **Файл** и создайте новый файл (**Новый**).

2. Наберите текст программы (см. ниже). При наборе текста программы соблюдайте позиционирование (отступы) строк. Это не влияет на работу программы, но делает ее читабельной и облегчает поиск ошибок.

```
program Lab1;
uses crt;
var x, y:real;
begin
    writeln('Введите значение аргумента');
    read(x);
    if (x>=-2) and (x<0) then y:=sqr(x)*x;
    if (x>=0) and (x<1) then y:=sqr(x);
    if (x>=1) and (x<1.5) then y:=4*ln(x);
    if (x>=1.5) and (x<=2) then y:=-10*sqr(x-1.6);
    if (x>=-2) or (x<=2) then writeln('Y=',y:7:2)
                         else writeln('Функция не определена');
end.
```

3. Запустите программу на выполнение. Введите значение переменной X, лежащее в рабочем диапазоне (-2<=X<=2). На экране увидим результат вычисления Y (Y=).

4. Еще раз запустите программу на выполнение. Введите значение переменной X, не лежащее в рабочем диапазоне (X<=-2 или X>=2). На экране увидим сообщение «Функция не определена».

5. Рассмотрим, что делает программа.

В разделе **var** объявили две переменные X и Y вещественного типа.

В **теле программы**: вначале запрашиваем значение аргумена X (оператор вывода – writeln, который выводит приглашение для ввода,

оператор ввода – read(x); , который вводит зачение с клавиатуры в переменную X).

Затем следуют четыре условных операторов, проверяющих в какой диапазон попадает введенное значение X и вычисляет соответствующее выражение.

Пятый условный оператор нужен для корректного вывода результатов работы программы: если X попадает в один из четырех заданных в условии задачи диапазонов (т.е. лежит в отрезке [-2;2]), то выводится подсчитанное значение Y. В противном случае (else) – выводится сообщение «Функция не определена».

6. Сохраните программу в своей папке.

Задание 2. Заданы два целых числа, если первое число больше второго, то вывести на экран текст «Первое число больше», в противном случае «Второе число больше».

Задание 3. Заданы три целых числа *a*, *b*, *c*. Найти максимальное из них.

Задание 4. Заданы четыре переменные ***x, y, z,*** *k* целого типа. Вычислить *a* по правилу:

$$a=\begin{cases} x+y & k<0 \\ x+z \text{ , при} & 0<k<10 \\ x^2 & k\geq 10 \end{cases}$$

Задание 5. Является ли условным оператором последовательность символов:

1) if x < y then x :=0 else y :=0;
2) if x > y then x :=0 else read(y);
3) if x >= y then x :=0; y :=0 else write(z);
4) if a < b then 100 else z :=5;

5) if a < b < c then z :=z+1;
6) if sqrt(z) <= 3.17 then z :=z+1;
7) if a <> b then z :=z+1; a :=b+1

Задание 6. Даны действительные числа *x*, *y*, *z*. Получить:

1) max(x+y+z, xyz);
2) min(x, y,)

Задание 7. Дано действительное число *a*. Вычислить f(a), если

а) $f(x)=\begin{cases} x^2+4x+5 & \text{при } x \le 2, \\ \dfrac{1}{x^2+4x+5} & \text{в противном случае} \end{cases}$;

б) $f(x)=\begin{cases} 0 & \text{при } x \le 0, \\ x^2-x & \text{при } 0 < x \le 1, \\ x^2-\sin \pi x^2 & \text{в остальных случаях} \end{cases}$.

Задание 8. Вычислить значение функции:

$$Y=\begin{cases} x3, & \text{если } x \le -2 \\ \sqrt{x+3}, & \text{если } -2 \le x \le 2 \\ \dfrac{2}{x-2} & \text{в остальных случаях} \end{cases}$$

Задание 9.Проверьте, делится ли набранное с клавиатуры число на 5, на 11 или 13.

Операторы цикла

В Паскале существует три оператора цикла. Эти операторы имеют собственные условные названия: «Пока», «До» и «С параметром».

Формат цикла «Пока» (цикл с предусловием):

While <условие> Do <оператор>;

По-русски можно прочитать так: «Пока истинно условие, выполнять оператор».

Работает оператор следующим образом. Вначале вычисляется условие, если результат вычисления имеет значение ИСТИНА, то выполняется оператор в цикле, после чего вычисление выражения <условие> и его проверка повторяются. Если логическое выражение имеет значение ЛОЖЬ, оператор **While** прекращает свою работу.

Здесь, так же, как в формате условного оператора, подразумевается выполнение только одного оператора. Если необходимо выполнить несколько действий, то используется составной оператор. Тогда формат оператора принимает такой вид:

While <условие> Do Begin
    <оператор 1>;
    <оператор 2>;
    <оператор 3>;
    . . .
End;

Пример. Дана последовательность целых чисел за которой следует ноль. Вычислить сумму элементов последовательности.

```
Program primer;
Var i, a, s : integer;
Begin
  s := 0;
  Read (a);
  While a <> 0 do  begin
      s := s + a;
     Read (a);
  end;
  Write ('Сумма элементов равна ', s);
End.
```

Формат цикла «До» (цикл с постусловием):

Repeat

<оператор 1>;
<оператор 2>;
<оператор 3>;
. . .

Until <условие>;

Читается так: «Выполнять оператор 1, оператор 2. и т. д. до выполнения условия».

Здесь не требуется использование составного оператора, потому что сами слова Repeat и Until являются операторными скобками.

Оператор **Repeat – Until** работает следующим образом. В начале выполняется тело цикла, после чего вычисляется логическое выражение следующее за словом **Until**, если результатом вычисления выражения является ЛОЖЬ, то операторы в теле цикла будут выполнены повторно. в противном случае, если логическое выражение имеет значение ИСТИНА, оператор цикла с постусловием прекратит свою работу.

Пример. Дана последовательность целых чисел, последним элементом которой является число 100. Вычислить среднее арифметическое элементов этой последовательности.

```
Program primer;
Var i, a, s, x : integer;
Begin
  s := 0;
  x := 0;
  Repeat
    Read (a);
    s := s + a;
    x := x + 1;
  Until a = 100;
```

```
  s := s / x;
  Write ('среднее арифметическое: ', s);
End.
```

Цикл «С параметром»

В данном случае параметром будет являться целочисленная переменная, которая будет изменяться на единицу при каждой итерации цикла. Таким образом, задав начальное и конечное значения для такой переменной, можно точно установить количество выполнений тела цикла.

Форматов у этого вида цикла предусмотрено два:

For <П.Ц>:=<Н.З.> To <К.З.> Do <оператор>;

For <П.Ц>:=<Н.З.> Downto <К.З.> Do <оператор>;

Здесь П.Ц – параметр цикла, Н.З. – его начальное значение, К.З. – соответственно конечное значение параметра.

В первом случае параметр с каждой итерацией увеличивается на единицу, во втором – уменьшается на единицу.

Читается данная структура так: «Для переменной (далее следует ее имя) от начального значения до конечного выполнять оператор (являющийся телом цикла)». Иногда цикл с параметром называют «Для» («For»).

Выполняется этот цикл по следующему алгоритму:

1) параметру цикла присваивается начальное значение;

2) проверяется условие, лежит ли значение параметра цикла между начальной и конечной величинами. Если оно лежит внутри интервала, то выполняется тело цикла, иначе работа цикла прекращается;

3) выполняется тело цикла;

4) переменная-параметр автоматически увеличивается на 1 (или -1).

Для работы цикла ***For-To*** требуется, чтобы начальное значение параметра цикла было меньше или равно конечному значению, а для цикла ***For-Downto*** – начальное значение параметра цикла было больше или равно конечному значению.

Отметим два обстоятельства. Во-первых, условие, управляющее работой оператора ***For***, проверяется перед выполнением оператора <оператор>: если условие не выполняется в самом начале работы циклического оператора, исполняемый оператор не будет выполнен ни разу. Другое обстоятельство – шаг наращивания параметр строго постоянен и равен +1 или -1.

В случае использования в цикле не одного, а нескольких операторов, следует заключать эти группы операторов в логические скобки ***begin*** и ***end***.

Пример. Дана последовательность целых чисел, состоящая из 25 элементов. Вычислить сумму элементов последовательности.

```
Program primer1;
Var i, a, s : integer;
Begin
  x := 0;
  For i := 1 to 25 do begin
     Read (a);
     x := x + a;
  end;
  Write ('Сумма элементов равна ', x);
End.
```

Решим задание из **примера** используя конструкцию оператора **For-Downto**, в которой шаг наращивания будет равен -1.

```
Program primer2;
Var i, a, s : integer;
Begin
  x := 0;
  For i := 25 downto 1 do  begin
     Read (a);
     x := x + a;
  end;
  Write ('Сумма элементов равна ', x);
End.
```

Практикум

Задание 1 (с решением). Составить программу расчета конечной суммы и сравнения полученного результата с контрольным значением. Число членов суммы вводится с клавиатуры:

вид суммы	**контрольное значение**
$1+2+3+4+...+N$	$\frac{N(N+1)}{2}$

Порядок выполнения в системе Pascal ABC (см. раздел «Система программирования Pascal ABC»)**:**

1. Активизируйте пункт **Файл** и создайте новый файл (**Новый**).

2. Наберите текст программы (см. ниже). При наборе текста программы соблюдайте позиционирование (отступы) строк. Это не влияет на работу программы, но делает ее читабельной и облегчает поиск ошибок.

```
program Lab2;
uses crt;
var n, S, i: integer;
        K: real;
```

```
begin
    writeln('Введите число членов суммы');
    read(n);
    S:=0;
    i:=1;
    while i<=n do begin
        S:=S+i;
        i:=i+1;
    end;
    writeln('S = ', S);
    K:=n*(n+1)/2;
    writeln('Контрольная сумма = ', K);
end.
```

3. Запустите программу на выполнение. Введите число членов суммы (значение переменной n). На экране увидим результат вычисления суммы ряда (S=) и контрольного значения (Контрольная сумма =). Эти значения должны совпадать. При несовпадении результатов суммы и котрольной суммы, делаем вывод о наличии ошибки в программе. Следует найти ошибку и исправить ее.

4. Рассмотрим, что делает программа.

В разделе **var** объявили три переменных n, S, i целого типа и одна переменная K вещественного типа (поскольку при ее вычислении используется операция деления /).

В **теле программы**: вначале запрашиваем число членов суммы n (оператор вывода – writeln, который выводит приглашение для ввода, оператор ввода – read(n); , который вводит зачение с клавиатуры в переменную n).

Затем обнуляем начальное значение суммы (S:=0) и присваиваем параметру цикла начальное значение (в нашем задании это 1).

Используем цикл while. Задаем условие: пока i<=n (то есть ссуммируем все члены ряда) вычисляем сумму (S:=S+i) и увеличиваем параметр на шаг (в нашем задании шаг равен 1).

Вместо цикла **while** можно было использовать цикл **repeat**:

```
i:=1;
Repeat
    S:=S+i;
    i:=i+1;
Until i>n;
```

Обратите внимание, как изменилось условие!

Можно было использовать и оператор For, поскольку по заданию шаг изменения параметра цикла равен 1! В этом случае цикл выглядит следующим образом:

```
For i:=1 to n do
    S:=S+i;
```

После оператора цикла в программе следует оператор вывода на экран вычисленной суммы (writeln('S = ', S)), вычисление котрольной суммы по заданной формуле (K:=n*(n+1)/2) и вывод ее на экран (writeln('Контрольная сумма = ', K)).

6. Сохраните программу в своей папке.

Задание 2. Является ли оператором цикла:

1) while x<0 do x:=x+0.5;
2) while x<0 do x:=x-100;
3) while 0<y<1 do y:=sqr(y)+0.01;
4) while a>0 do y:=2*y;
5) while a>b do a:=a-1; b:=b+1

Задание 3. После выполнения фрагмента программы

```
C:=0; A:=17; B:=13; D:=2*A+3;
while D>=B do begin
        C:=C+1; D:=D-B;
end;
```

значения переменных *C* и *D* равны

a) C=3, D=8, b) C=3, D=9, c) C=2, D=9, d) C=2, D=11, e) C=2, D=8

Задание 4. Вычислить n! (n – факториал; n!=1*2*3*…*n).

Задание 5. После выполнения фрагмента программы

```
y:=0;   x:=1;
repeat
     y:=y+x*x;
     x:=x+1
until x>=4;
```

переменная *y* примет значение:

а) 5; б) 14; в) 16; г) 30.

Задание 6. Протабулировать (Построить таблицу соответствия «Аргумент x» и «Функция y») функцию y=sin(x), при x изменяющемся от -5 до 5 с шагом -0.6.

Задание 7. Используя цикл с постусловием, найдите сумму и призведение целых положительных чисел больших 13 и меньших 100 и кратных 4.

Задание 8. Напечатать таблицу перевода миль в километры для расстояний от 5 до 75 миль с шагом 5, если 1 миля составляет 1,609 км.

Задание 9. Написать программу, которая выводит на экран таблицу умножения, например, на 7.

Массивы

Рассмотренные выше простые типы данных – логический (boolean), целый (integer , word , byte , longint), вещественный (real), символьный (char) позволяют работать с одиночными объектами. В языке Паскаль могут использоваться также объекты, содержащие множество однотипных элементов. Массив – это упорядоченная последовательность однотипных данных, рассматриваемых как одно целое. Упорядоченность данных в массиве позволяет обращаться к любому элементу массива по его порядковому номеру (**индексу**). Элементы массива расположены последовательно в непрерывной области памяти.

Нужно четко понимать, что индекс ячейки массива не является ее содержимым. Содержимым являются хранимые в ячейках данные, а индексы только указывают на них. Индексы элементов массива обычно целые числа, однако могут быть и символами, а также описываться другими порядковыми типами. Зачастую для задания количества элементов массива используется тип-диапазон. Тип-диапазон задается левой и правой границами изменения индекса массива.

Описание массива

Перед использованием массив, как и любая переменная, должна быть объявлена (описана).

Описание типа массива задается следующим образом:

```
type
   имя типа = array[ список индексов ] of тип;
```

Здесь *имя типа* – правильный идентификатор; *список индексов* – список одного или нескольких индексных типов, разделенных запятыми; *тип* – любой тип данных.

Пример.

```
const
  n = 5;
type
  mas = array[1..n] of integer;{Объявлен тип mas, являющийся
                                массивом (array) целых чисел
                                (integer), границы изменения
                                индекса массива – 1… n}
var
  a: mas;
```

Определить переменную как массив можно и непосредственно при ее описании в разделе *var*, без предварительного описания типа массива, например:

```
var
  a,b,c: array[1..10] of integer;{Объявлены три массива a, b, c,
                                  состоящие из десяти целых
                                  элементов}
```

Обращение к определенному элементу массива осуществляется путем указания имени переменной массива и в квадратных скобках индекса элемента, например *a[3]* – обращение к третьему элементу массива *a*, *a[i]* – обращение к *i*-му элементу массива *a*.

Простой массив является одномерным. Он представляет собой линейную структуру.

Основные действия с массивами

Единственное действие, которое можно выполнять над массивами целиком, причем только при условии, что массивы

однотипны, – это присваивание. Если в программе описаны две переменные одного типа, например,

```
Var
  a , b : array [1..10] of real;
```

то можно переменной *a* присвоить значение переменной *b* (*a:=b*). При этом каждому элементу массива *a* будет присвоено соответствующее значение из массива *b*. Все остальные действия над массивами Паскаля производятся поэлементно (это важно!).

Ввод массива

Для того чтобы ввести значения элементов массива, необходимо последовательно изменять значение индекса, начиная с первого до последнего, и вводить соответствующий элемент. Для реализации этих действий удобно использовать цикл с заданным числом повторений, где параметром цикла будет выступать переменная – индекс массива. Значения элементов могут быть введены с клавиатуры или определены с помощью оператора присваивания.

Пример фрагмента программы ввода массива с клавиатуры:

```
Var
   A : array [1..10] of integer;
   i : byte; {переменная i как индекс массива}
Begin
  For i:=1 to 10 do
    Read (a[i]); { ввод i- го элемента производится с клавиатуры }
```

Рассмотрим теперь случай, когда массив заполняется автоматически случайными числами. Для этого будем использовать генератор случайных чисел – random (N).

Пример фрагмента программы заполнения массива случайными числами:

```
Var
   A: array [1..10] of integer;
   i : byte ;
Begin
   For i :=1 to 10 do
      A [ i ]:= random (50)-25; {i-му элементу массива
                присваивается «случайное» целое
                число в диапазоне от 0 до 49 с
                вычетом 25, т.е. окончательный
                диапазон от -25 до 24} .
```

Вывод массива

Вывод массива в Паскале осуществляется также поэлементно, в цикле, где параметром выступает индекс массива, принимая последовательно все значения от первого до последнего.

Пример фрагмента программы вывода массива:

```
Var
   A: array [1..10] of integer;
   i : byte ; {переменная i как индекс массива}
Begin
   Writeln('Массив А');
   For i :=1 to 10 do
      Write ( a [ i ]:5); {вывод массива осуществляется в строку,
                под каждый элемент выделяется 5
                позиций, иначе элементы массива будут
                выведены слитно} .
   Writeln; (Для перевода курсора на следующую строку)
```

На экране мы увидим, к примеру, следующие значения:

Массив А

_ _ _ _ 5_ _ _ - 2_ _ 1 1 5 и т.д.

Вывод можно осуществить и в столбик (использовать оператор *Writeln*). Но в таком случае нужно учитывать, что при большой размерности массива все элементы могут не поместиться на экране и будет происходить скроллинг, т.е. при заполнении всех строк экрана будет печататься очередной элемент, а верхний смещаться за пределы экрана.

Пример программы вывода массива в столбик:

```
Var
   A: array [1..10] of integer;
   i : byte ;
Begin
   For i:=1 to 10 do
     Writeln (‘a[‘, i,’]=’, a[i]); {вывод элементов массива в столбик} .
```

На экране мы увидим, к примеру, следующие значения:

a [1]=2

a [2]=4

a [3]=1 и т.д.

Алгоритмы обработки

1. **Поиск минимального элемента:**

```
min:=a[1];
For i:=2 to 10 do
   If a[i]< min then min:= a[i];
```

2. Замена наибольшего элемента массива на 0:

```
max:=a[1]; imax:=1;
For i:=2 to 10 do
    If a[i] > max then begin
                        max:= a[i];
                       imax:=i;
    end;
a[imax]:=0;
```

В переменной *max* получаем наибольший элемент массива, а в переменной *imax* – индекс наибольшего элемента.

3. Удаление наибольшего элемента из массива:

```
max:=a[1]; imax:=1;
For i:=2 to 10 do
    If a[i] > max then begin
                        max:= a[i];
                       imax:=i;
    end;
For i:= imax to 9 do {сдвиг элементов, начиная с наибольшего и
    a[i]:=a[i+1];       до конца массива}
```

4. Перенос наибольшего элемента в начало массива:

```
max:=a[1]; imax:=1;
For i:=2 to 10 do
    If a[i] > max then begin
                        max:= a[i];
                       imax:=i;
    end;
For i:= imax downto 2 do {сдвиг элементов в противоположную
    a[i]:=a[i-1];              сторону, начиная с наибольшего и до
                               первого элемента массива}
a[1]:=max;
```

5. Поиск суммы четных элементов массива:

```
S:=0;
For i:=1 to 10 do
    If a[i] mod 2 = 0 then S:=S+ a[i];
```

Рассмотрим пример программы целиком.

Задача: Сформировать массив *b* из нечетных элементов массива *a*, состоящим из *n* элементов.

```
Program massiv;
Const n=10;
Var
    a, b: array [1..n] of integer;
    i, k: byte;       {i – индекс массива a, k – индекс массива b}
 Begin
   For i:=1 to n do       {Заполнение массива числами в диапазоне
                           от -25 до 24 случайным образом}
      a[i]:= random (50)-25;
   writeln('Массив A');
   For i:=1 to k-1 do
      write (a[i]:5);   { вывод массива a в строку}
   writeln;
   k:=1;
   For i:=1 to n do
      If a[i] mod 2 <> 0 then begin  {Проверка на нечетность}
                          b[k]:=a[i]; {Запись нечетного элемента в
                                       массив b}
                          k:=k+1;      {Подготовка индекса для
                                       следующего элемента}
      end;
   writeln('Массив B');
```

```
    For i:=1 to k-1 do
        write (b[i]:5);   {вывод массива b в строку}
end.
```

Система программирования Pascal ABC

В настоящее время удобной в учебном процессе является система программирования Pascal ABC (Паскаль ABC). Система предназначена для обучения программированию на языке Паскаль и ориентирована на школьников и студентов младших курсов. Кроме того, в пакете имеется Электронный задачник.

Эта система призвана осуществить переход от простейших программ к модульному, объектно-ориентированному, событийному и компонентному программированию.

Система Pascal ABC 3.0 & Programming Taskbook 4.5 Mini Edition (называемого в дальнейшем системой PABC-PT ME) является бесплатной и распространяется свободно при условии, что настоящий дистрибутив не изменен. Скачать программный комплекс можно в интернете совершенно бесплатно и установить на свой компьютер.

В скачанном пакете для установки запустите файл PABCInstall и в появившемся окне нажмите кнопку «Установка».

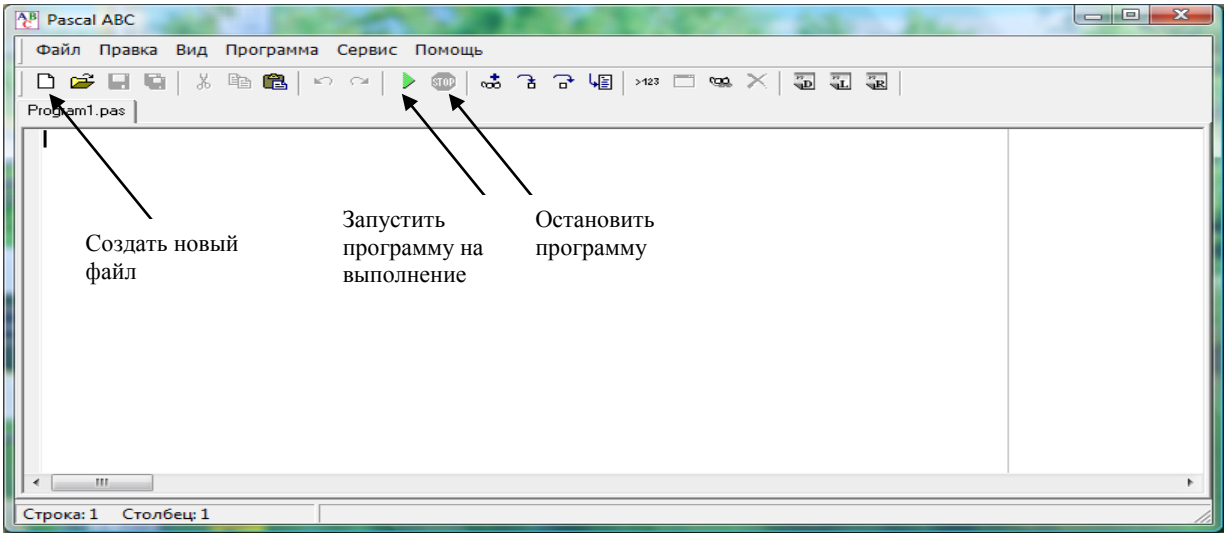
После установки автоматически запускается программа регистрации и настройки PABCSetup. В дальнейшем эту программу можно запустить повторно с помощью команды **PABC Setup – Регистрация и настройка** пункта **Pascal ABC** в группе **Программы** главного меню Windows.

Знакомство с системой программирования Паскаль ABC

Для запуска Паскаль ABC необходимо запустить ярлык **Pascal ABC**. На экране появится среда программирования Паскаль ABC (оболочка). Среда программирования – это пакет взаимосвязанных

файлов, которые позволяют набирать, редактировать, запускать и отлаживать программы.

После запуска ярлыка на рабочем столе открывается окно:



Первая строка экрана – строка заголовка, вторая – меню интегрированной среды, следующая строка – панель инструментов, нижняя строка экрана – строка состояния интегрированной среды. Между ними расположено окно редактирования – рабочее поле, в котором можно открывать несколько вкладок для разных программ.

Окно редактирования предназначено для ввода и редактирования текста программы. Место ввода информации обозначено курсором. В верхней левой части окна редактирования выводится служебное имя редактируемого файла, например: Program1.pas.

Знакомство с меню Паскаль АВС

Поочередно войдите в указанные ниже разделы **Меню** (активизируйте Меню мышью).

В меню Файл

Новый – создать новый фал

Открыть – открыть файл

Сохранить – сохранить файл

Сохранить как… – сохранить под новым именем

Выход – выйти из Паскаля

В меню Правка

Отменить – отменить изменение

Восстановить – вернуть изменение

В меню Программа

Выполнить – выполнить программу

Остановить – остановить программу.

Ввод текста программы

1. Набрать простейшую программу, соответствующую условию задачи:

Ввести в компьютер два целых числа, найти их сумму, результат вывести на экран с поясняющим текстом.

Внимание! Две косые черты (//) отделяют комментарии, их набирать не нужно.

```
program raschet;                    // название программы
uses crt;                           // подключаемые модули
var x, y, s:integer;                // объявление переменных
begin                               // начало исполнительной части
   writeln('Введите два целых числа');   // вывести на экран текст
   readln(x,y);                     //  прочитать данные с клавиатуры
                                    // и записать их в переменные x и y
   s:=x+y;                          // выполнить расчет и запомнить
                                    //  его в переменной s
   writeln('Сумма чисел =',s);      // Вывести на экран текст и значение
                                    //  переменной s
end.                                // конец программы
```

2. Просмотрите текст файла, обратите внимание на стуктуру программы.

Структура простейших программ выглядит следующим образом:

program ...; *заголовок программы и ее имя*

var ...; *блок объявления переменных и их типа*

begin *начало исполнительной части программы*

...; *операторы, обеспечивающие*

...; *выполнение*

...; *программы*

end. *конец программы (точка обязательна)*

В Паскале текст программы обычно начинается заголовком.

В качестве имени программы можно применять комбинацию английских букв и цифр (первый символ – буква), без пробелов и нельзя применять служебные слова языка.

Каждый оператор языка заканчивается точкой с запятой (;).

Обычно каждый раздел программы и оператор записывается с новой строки для наглядности и более легкого понимания текста. Для этих же целей используют отступы и выравнивания.

Комментарии предназначены для пояснения задачи и для временного исключения из текста программы некоторых операторов. В тексте они выделяются фигурными скобками { } или отделяются двумя косыми чертами //. Комментарии игнорируются компьютером при выполнении.

Под именем программы располагается ее **декларативная часть**, в которой компьютеру сообщается обо всех именах констант и переменных, определяемых программистом.

За декларативной частью следует **исполнительная часть** программы, обрамляемая словами-ограничителями (логическими скобками): begin и end. Между указанной парой слов и размещаются операторы, выполняющие в программе те или иные действия. Исполнительную часть программы называют **телом программы**.

Запуск программы на выполнение

1. Запустите набранную программу на выполнение (кнопка **Выполнить** на панели инструментов). Если после запуска программы внизу окна появляется красная строчка с сообщением (рис. 4.1), то в строке, где находится курсор или в предыдущей (но не всегда, это зависит от ошибки!) имеется ошибка. Внимательно просмотрите всю строчку, найдите и исправьте ошибку.

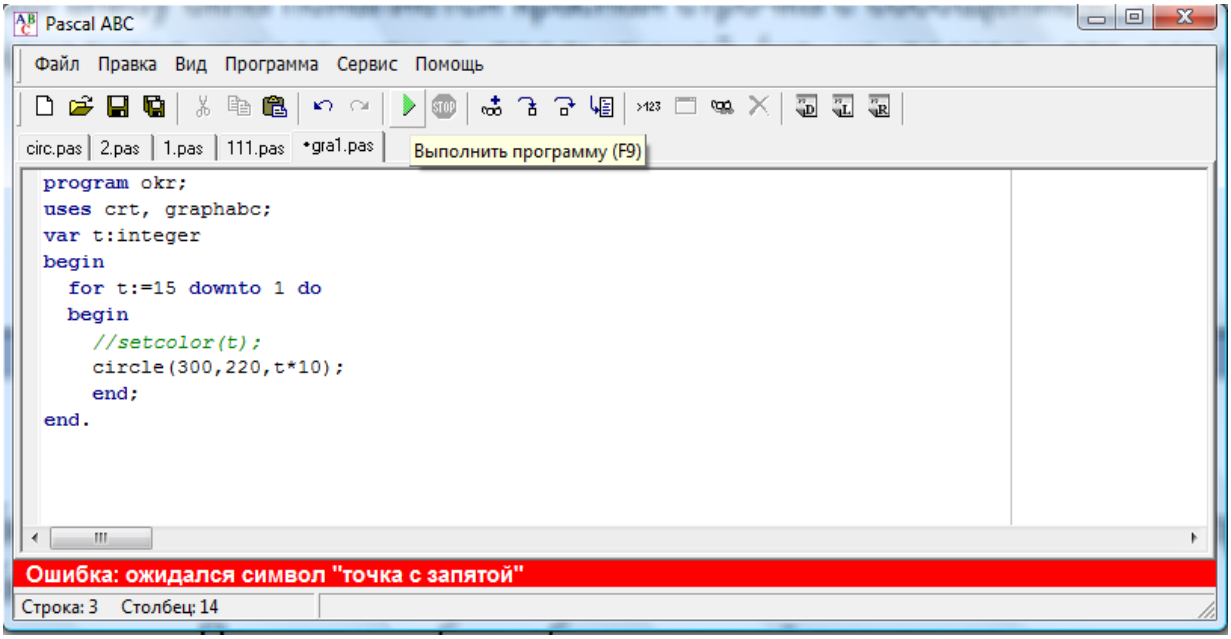


Рис. 4.1. Пример ошибки, обнаруженной Паскаль АВС

2. После исправления ошибки, вновь запустите программу.

3. После исправления всех ошибок и появления в новом окне начала работы программы, введите нужные данные (если в программе подразумевается ввод нескольких значений переменных, то это следует делать через **Enter** или **пробел**!), получите результат работы и проверьте его на правильность. Так как текст программы и ее работа показываются в разных окнах (если подключен модуль Crt), можно сопоставить программные строки и ее выполнение (рис. 4.2).

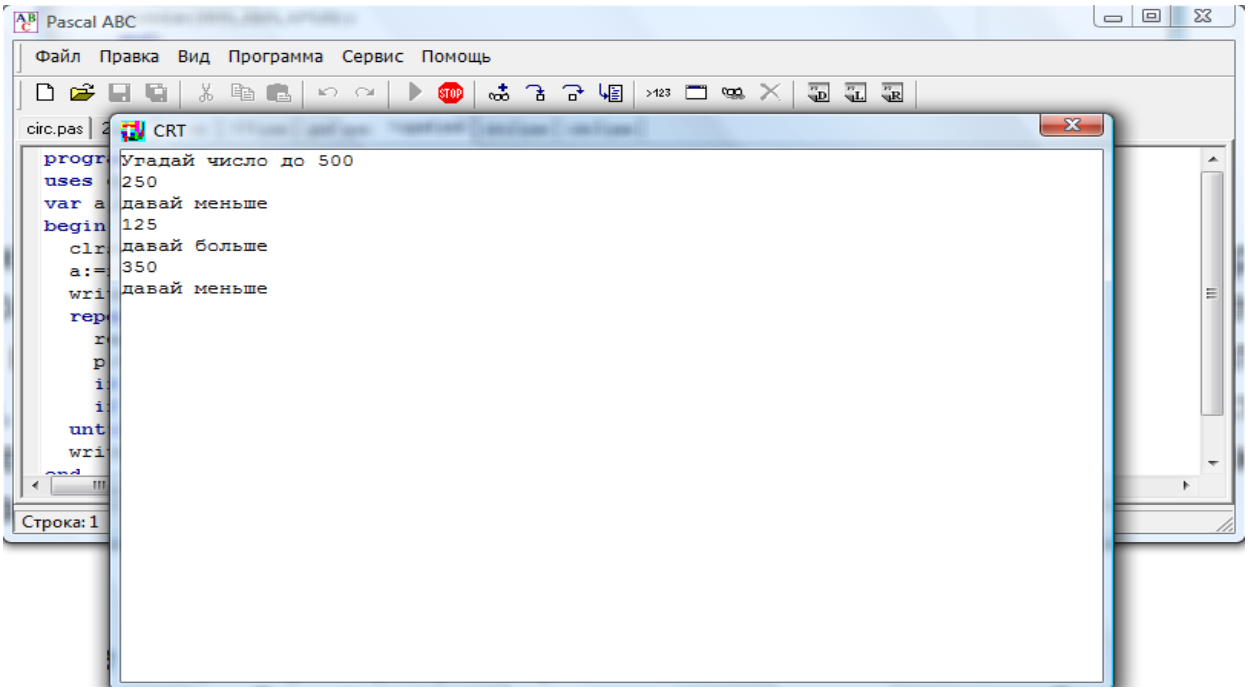


Рис. 4.2. Пример работы программы

4. Сохраните набранную программу в своей папке.

5. Разберитесь с работой программы и измените ее так, чтобы она вычисляла не сумму, а разность чисел. Проверьте правильность работы измененной программы. Сохраните программу под новым именем в своей папке.

Контрольные вопросы

1. Приведите структуру программы на языке Паскаль.
2. Константы и переменные. Где и как объявляются. Их отличия.
3. Перечислите базовые типы данных в языке Паскаль. Формы представления вещественных чисел. Какие арифметические операции невозможны над данными вещественного типа?
4. Приведите таблицы стандартных функций возвращающих целый и вещественный результат.
5. Приведите форматы оператора ввода и оператора присваивания. Приведите примеры операторов ввода и присваивания.

6. Приведите формат оператора вывода. Приведите примеры оператора. Для чего нужно указание формата числа в операторе вывода. Приведите примеры. Отличия операторов Writeln и Write.
7. Приведите форматы полного и неполного условного оператора. Приведите примеры.
8. Приведите форматы операторов цикла While и Repeat. Приведите пример оператора. Их отличия.
9. Приведите форматы оператора цикла For. Приведите примеры оператора.
10. Что называется массивом?
11. Как объявить одномерный массив?
12. Приведите фрагмент ввода одномерного массива с клавиатуры.
13. Приведите фрагмент формирования одномерного массива случайными числами и укажите, какой диапазон чисел будет использован.
14. Приведите фрагмент вывода одномерного массива в строку.

Заключение

В одном пособии, ориентированном на знакомство с основами алгоритмизации, этапами эволюции технологий программирования, нельзя рассмотреть все вопросы, связанные с программированием на языке Паскаль, все компоненты и возможности среды разработки. Многие интересные темы остались за рамками книги. В пособии рассмотрены фундаментальные понятия программирования, базовые структуры данных и методы работы с ними, основные возможности среды разработки Pascal ABC и методы работы в ней – все то, что должен знать и уметь начинающий программист.

Работая над пособием, автор старался выстроить материал таким образом, чтобы в тот момент, когда читатель перевернет последнюю страницу, он был бы в состоянии написать вполне приличную программу, которая бы не «ломалась» от неверно введенных данных.

Научиться программировать, не владея общими вопросами теории и практики программирования, практически невозможно.

В заключение напомним, что научиться программировать можно только программируя, решая конкретные задачи. Поэтому ищите задачи и программируйте!

Библиографический список

1. Кадырова, Г. Р. Информатика: учебно-практическое пособие / Г. Р. Кадырова; Ульян. гос. техн. ун-т. – 2-е изд., доп. и перераб. – Ульяновск : УлГТУ, 2013. – 227 с.
2. Кадырова, Г. Р. Курс лекций по информатике : учебное пособие. В 2 частях. Часть 2 / Г. Р. Кадырова. – Ульяновск : УлГТУ, 2008. – 132 с.
3. Семакин, И. Г. Основы алгоритмизации и программирования : учебник для студ. учреждений сред. проф. образования / И. Г. Семакин, А. П. Шестаков. – 3-е изд., стер. – М. : Издательский центр «Академия», 2012. – 400 с.
4. Каймин, В. А. Информатика : учебник / В.А. Каймин. – М. : Проспект, 2011. – 272 с.

Интернет-ресурсы

1. http://venec.ulstu.ru/lib/disk/2012/Kadyrova.pdf – Кадырова, Г. Р. Информационное и компьютерное обеспечение. Обзор лекций по информатике. (В двух частях). Часть 1 : учебное пособие / Г. Р. Кадырова. – Ульяновск : УлГТУ, 2011. (дата обращения: 10.07.2014)
2. http://venec.ulstu.ru/lib/disk/2012/Kadyrova.pdf -Кадырова, Г. Р. Курс лекций по информатике : учебное пособие. В 2 частях. Часть. 2 / Г. Р. Кадырова. – Ульяновск : УлГТУ, 2008. (дата обращения: 10.07.2014)
3. http://www.intuit.ru/ – дистанционное обучение в Национальном Открытом Университете «ИНТУИТ». (дата обращения: 15.07.2014)
4. http://www.coders-library.ru/ – Библиотека программиста. (дата обращения 15.07.2014)
5. http://vbbook.ru/ – Сайт для программистов. (дата обращения 15.07.2014)

Учебное издание

КАДЫРОВА Гульнара Ривальевна

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Учебное пособие

ЭИ № 316.

Редактор Н. А. Евдокимова
ЛР №020640 от 22.10.97.
Подписано в печать 16.09.2014. Формат 60×84/16.
Усл. печ. л. 5,58. Тираж 230 экз. Заказ 1038.
Ульяновский государственный технический университет
432027, г. Ульяновск, ул. Северный Венец, д. 32.
ИПК «Венец» УлГТУ. 432027, г. Ульяновск, ул. Северный Венец, д. 32.