МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ
«МИФИ»

ВВЕДЕНИЕ В ОПЕРАЦИОННЫЕ СИСТЕМЫ И ОСНОВЫ ПРОГРАММИРОВАНИЯ

Лабораторный практикум

Рекомендовано к изданию
УМО «Ядерные физика и технологии»

Москва 2015

УДК 004(075)
ББК 32.97я7
В24

ВВЕДЕНИЕ В ОПЕРАЦИОННЫЕ СИСТЕМЫ И ОСНОВЫ ПРОГРАММИРОВАНИЯ: *лабораторный практикум* / Г.П. Аверьянов, В.А. Будкин, В.В. Дмитриева, И.А. Кунов. – М.: НИЯУ МИФИ, 2015. – 260 с.

В учебно-методическом пособии представлены две темы, входящие в состав обзорного курса «Информатика» («Информатика и программирование»).

Первая тема посвящена изучению общих принципов и функциональных возможностей (с точки зрения пользователя) современных операционных систем (ОС) семейства Windows и UNIX-подобных ОС. Приведены два типа пользовательского интерфейса: «командная строка», на примере командных интерпретаторов (bash для ОС UNIX и cmd.exe для MS Windows), а также полноэкранный алфавитно-цифровой интерфейс, на примере наиболее популярных «файловых менеджеров» (Midnight Commander (mc) для ОС UNIX и FAR Manager для MS Windows).

Вторая тема связана с изучением основ программирования на современных императивных алгоритмических языках. Рассмотрено программирование стандартных алгоритмов (с использованием условных операторов и циклов) и структур данных (включая скалярные данные, массивы и другие структуры данных). В качестве языка программирования представлен традиционный для инженерных и научных расчетов алгоритмический язык Фортран. Использовался компилятор Gfortran (free Fortran 95/2003/2008 compiler for GCC, the GNU Compiler Collection), реализующий стандарт [ISO/IEC 1539-1:1997(E)], с поддержкой объектно-ориентированного программирования (ООП).

Даны примеры и задания в виде традиционных для курсов информатики алгоритмов, дополненных элементами ООП и вычислительной математики, необходимыми для эффективного программирования научных и инженерных задач.

Пособие предназначено для студентов очной и очно-заочной (вечерней) формы обучения факультета «Автоматика и электроника» НИЯУ МИФИ, а также может быть полезно студентам физических и инженерно-технических специальностей вузов.

Подготовлено в рамках Программы создания и развития НИЯУ МИФИ.

Рецензент Е.А.Сухарева, канд. физ.-мат. наук,
доц. Московского машиностроительного университета (МАМИ)

ISBN 978-5-7262-1994-3

Редактор М.В. Макарова
Подписано в печать 20.11.2014. Формат 60x84 1/16
Уч.-изд.л. 16,75. Печ.л. 16,25. Тираж 250 экз.
Изд. № 1/10. Заказ № 27.
Национальный исследовательский ядерный университет «МИФИ».
115409, Москва, Каширское ш., 31.
ООО «Баркас».
115230, Москва, Каширское ш., 4..

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Данное пособие написано авторами на основе более чем 20-летнего опыта проведения практических занятий со студентами первых курсов МИФИ и является дополнением к теоретическому курсу «Современная информатика» (НИЯУ МИФИ, 2011).

Информационные технологии развиваются на глазах и используются практически во всех сферах человеческой деятельности. С одной стороны, они все более усложняются и для их применения требуются все более глубокие знания. С другой стороны, упрощаются интерфейсы взаимодействия пользователя с компьютерами.

Компьютеры и информационные системы становятся все более дружественными к пользователям, не обладающим широкими познаниями в области информатики и вычислительной техники.

Разд. 1 и 2 посвящены знакомству с базовыми управляющими командами операционных систем (ОС), которые обеспечивают интерфейс пользователя с «железом» и прикладными программами, но при этом скрыты (в современных ОС) за пиктограммами, системами меню и другими элементами графических оболочек.

Включив компьютер (рабочую станцию, планшет или смартфон) и дождавшись загрузки операционной системы, в подавляющем большинстве случаев пользователь сталкивается с WIMP-итерфейсом («Window, Icon, Menu, Pointing device»). Что по-русски означает: «Окно, значок, меню, указательное устройство». На практике все еще значительно проще – использование «тыкательного инстинкта» пользователя и действие по одной и той же стереотипной схеме, с использованием в качестве указательного устройства «мыши» (или пальца, для сенсорного экрана), как практически единственного инструмента управления.

С одной стороны, использование означенных выше «указательных устройств» облегчает жизнь рядового пользователя, но при этом значительно снижает возможность эффективного использования компьютерной техники. Это становится критично, если от пользователя требуется немного больше, чем набрать текст в Word или поисковый запрос в Яндекс или Google.

Не рассматривая подробно все типы пользовательских интерфейсов, стоит отметить, что помимо графического интерфейса пользователя (GUI – Graphic User Interface), частным случаем кото-

рого является WIMP, профессионалами IT достаточно широко применяются инструменты из категории *текстовых пользовательских интерфейсов* (или «полноэкранных алфавитно-цифровых интерфейсов»). Такие интерфейсы (англ. *Text user interface, TUI* или *Character User Interface, CUI*) используют при вводе-выводе и представлении информации только набор буквенно-цифровых символов и символов псевдографики. TUI (CUI) характеризуется малой требовательностью к вычислительным ресурсам и высокой скоростью отображения информации.

Особая разновидность CUI – *командная строка* (англ. *Command Line Interface, CLI*) имеет отдельные преимущества перед графическим интерфейсом (при наличии определенной квалификации в его использовании). «Продвинутые» пользователи не могут обойтись без командной строки. Зная, какие средства командной строки выбрать и как их использовать, можно избежать многочисленных проблем и добиться быстрого и качественного выполнения не только административных, но и вполне «пользовательских» задач. Работа с ОС в режиме командной строки позволяет выявить общие и отличительные черты (с точки зрения пользователя) различных семейств ОС. С этой целью рассмотрены два наиболее распространенных на сегодняшний день семейства ОС для серверов, рабочих станций и домашнего применения. Это UNIX-подобные ОС и семейство проприетарных ОС Windows корпорации Microsoft.

UNIX-системы – многопользовательские многозадачные операционные системы. История создания UNIX начинается с операционных систем BESYS (1957 г.) и Multics (1964 г.), созданных в корпорации Bell Labs, под руководством Виктора Высотского (русского по происхождению). Следующая система – UNICS (позже название сократилось до UNIX) была разработана в конце 1960-х годов сотрудниками Bell Labs, воплотившими многие идеи Multics на компьютерах нового поколения. Версия UNIX для наиболее успешного семейства миникомпьютеров 1970-х, PDP-11, (в СССР его аналоги были известны как СМ ЭВМ и «Электроника», затем ДВК) получила название Edition 1 и была первой официальной версией. Системное время всей реализации UNIX отсчитывают с 1 января 1970 г. Первые версии UNIX были написаны на ассемблере и не имели встроенного компилятора с языком высокого уровня. В

1975 г. вышла пятая редакция UNIX, полностью написанная на языке высокого уровня Си.

В конце 1970-х и начале 1980-х начали появляться UNIX-подобные ОС, большинство из которых угадываются по названию (их еще называют *nix-системами): AIX, HP-UX, IRIX, Ultrix, Xenix, Mac OS X и т.д., хотя встречаются и исключения из этого правила, например, BSD/OS, Solaris и OSF/1. UNIX-подобные ОС обычно называют просто UNIX-системами и по классификации Эрика Рэймонда подразделяются на 3 типа:

1) генетический UNIX – ОС, в которых прослеживается происхождение от разработок AT&T;

2) «настоящий» UNIX ОС, имеющие юридическое право называться UNIX; к ним относятся системы, сертифицированные консорциумом The Open Group (правообладателем бренда UNIX) на соответствие Единой спецификации UNIX;

3) UNIX по функциональности.

В 1991 г. финским программистом Линусом Торвальдсом было опубликовано ядро Linux –*nix системы для ПК, содержащей в названии имя автора. Так было положено начало ОС, известной как GNU/Linux. Дистрибутивы этой системы, включающие ядро, утилиты GNU и дополнительное программное обеспечение, стали весьма популярными среди энтузиастов Free Ware. Один из создателей UNIX Деннис Ритчи отмечает, что ОС, подобные Linux, несомненно являются UNIX-системами.

Продвигаемая Google операционная система для смартфонов и планшетных компьютеров *Android*, как и проект *MeeGo*, поддерживаемый такими известными компаниями, как Intel (на сегодня основной участник проекта), Nokia, AMD, Novell, ASUS, Acer, MSI, созданы на основе ядра Linux.

В принципе, пытливому студенту ничто не мешает установить на своем мобильном устройстве Android Terminal Emulator (или другое приложение из Play Market, поддерживающее командную строку) и поупражняться в выполнении заданий этого практикума.

Хорошо продуманная командная строка была и остается сильной стороной UNIX-систем. За черным экраном и мерцающим на нем курсором скрывается чрезвычайно мощный и нетребовательный к ресурсам инструмент – командный интерпретатор (приложение, реализующее интерфейс командной строки – чаще всего это

bash), вот уже более полувека являющийся незаменимым посредником между человеком и компьютером.

Наиболее характерными признаки UNIX-систем являются:

- взаимодействие с пользователем посредством виртуального устройства – терминала;
- широкое применение командных интерпретаторов (shell) утилит, запускаемых из командной строки;
- применение файлов для представления физических и виртуальных устройств, а также некоторых средств взаимодействия между процессами;
- использование конвейеров из нескольких программ, каждая из которых выполняет одну задачу;
- использование простых текстовых файлов для настройки и управления системой.

По сравнению с командными интерпретаторами UNIX-систем командная строка Windows вплоть до Windows NT являлась гораздо менее мощным инструментом, однако, начиная с Windows 2000, командный интерпретатор был существенно переработан, и появилось множество новых консольных утилит.

С каждой новой версией Windows командная строка совершенствовалась, а ее возможности расширялись. Командная строка претерпела значительные изменения, связанные не только с повышением производительности, но и с увеличением гибкости. Теперь с помощью командной строки Windows можно решать задачи, которые нельзя было решить в предыдущих версиях Windows.

В эпоху доминирования графических интерфейсов консольные приложения и утилиты Windows продолжают существовать. Для опытных администраторов Windows и квалифицированных специалистов по технической поддержке командный интерпретатор всегда был и остается незаменимым и высокоэффективным инструментом, позволяющим решать сложнейшие задачи.

Для студентов инженерных специальностей, не связанных непосредственно с разработкой программного обеспечения, работа с командной строкой способствует формированию культуры инженерного мышления и пониманию базовой логики сложных технических систем, в том числе на примере операционной системы Windows.

Для работы с ОС Windows в режиме командной строки необходимо запустить командный интерпретатор (выбрав пункт «Командная строка» в меню кнопки «Пуск»). В зависимости от версии Windows запустится интерпретатор Cmd.exe (для MS Windows 2000, XP, Vista) или PowerShell (для Windows 7, Windows 8 или Windows Server 2008 R2). Работа с командной строкой поддерживается всеми Desktop-версиями Windows и Windows RT, на базе ARM-процессоров, включая планшетные компьютеры (в ОС Windows Phone не поддерживается). Команды, используемые в практикуме универсальны практически для всех версий Windows с поддержкой командного интерпретатора.

Разд. 3 пособия представляет основы программирования на «традиционных» (императивных) языках высокого уровня, которые (так же как и ОС, и любые другие информационные системы одинакового назначения) имеют общие базовые принципы. Изучение программирования, в классическом варианте, связано с рассмотрением структуры программных единиц (главной программы и подпрограмм), основных и производных типов данных и реализации стандартных алгоритмов, использующих условные операторы, циклы, массивы и другие структуры данных.

С учетом специфики подготовки научно-технических кадров, для программирования задач практикума выбран стандартный для научно-инженерных и расчетов алгоритмический язык Фортран. Название языка является сокращением от FORmula TRANslator (транслятор или переводчик формул), это самый первый язык программирования высокого уровня, для которого был создан транслятор (с 1954 по 1957 г. под руководством Джона Бэкуса в компании IBM был создан первый коммерческий компилятор Фортрана). До этого программы записывались либо в машинных кодах, либо на символических ассемблерах.

Фортран широко используется и остается доминирующим языком программирования для научных и инженерных вычислений уже более 50-ти лет. За это время создано огромное количество библиотек, содержащих готовые решения для многих сложных научных и инженерных проблем. Не существует практически ни одной операционной системы или процессора, для которых не был бы создан компилятор Фортрана, поскольку реализовать транслятор Фортрана для новой программно-аппаратной платформы, а затем

использовать готовые библиотеки – гораздо проще, чем преобразовывать миллионы строк программного кода на другие языки или создавать все заново. Так, компиляторы Фортрана (обычно это g77 и Gfortran) от фонда свободного программного обеспечения GNU входят в состав открытого программного обеспечения для суперкомпьютеров, на базе кластеров Beowulf и openMosix (для платформ, совместимых с GCC).

Общей отличительной чертой всех без исключения компиляторов Фортрана (собственно, с этой целью они и разрабатывались) является оптимизация машинного кода для максимального быстродействия вычислений с плавающей точкой (включая адресацию памяти). Сравнительное тестирование компиляторов различных языков программирования всякий раз показывает, что Фортрану нет равных по скорости вычислений.

Сначала Фортран существовал в рамках корпоративных стандартов компании IBM: FORTRAN II и FORTRAN IV, а затем в международных стандартах: рамках Американского национального института стандартов (ANSI) и Международной организации по стандартизации (ISO). В ANSI создан технический комитет X3J3, ответственный за стандартизацию Фортрана. Международная группа наблюдателей ISO/IEC JTC1/SC22/WG5 (сокращенно WG5) курирует работу X3J3 с целью стандартизации Фортрана в рамках ISO. Совместимость стандартов обеспечивается процедурой работы комитета X3J3, согласно которой исключение устаревших свойств языка возможно только по истечении заявленного предупредительного периода (как правило, не менее 10 лет).

На базе корпоративного стандарта фирмы IBM был создан первый международный стандарт FORTRAN 66 (1966 г.), затем последовали: FORTRAN 77 (1978 г.), обеспечивающий поддержку процедурного и структурного программирования, и Fortran 90 (1991 г.) с поддержкой объектно-ориентированного программирования. Принятый в 1997 г. стандарт Fortran 95 по существу является исправленной и улучшенной версией Fortran 90. В последующих стандартах Fortran 2003 (2004 г.) и Fortran 2008 сделан акцент на дальнейшее развитие средств поддержки объектно-ориентированного программирования и взаимодействия языка с операционной системой.

В тестах на производительность вычислений лидирующие позиции занимает коммерческий компилятор корпорации Intel (Intel Fortran Compiler) для платформ Intel IA-32 и IA-64, а для персональных компьютеров (IBM PC) долгое время лучшим компилятором Фортрана считался компилятор Watcom Fortran.

Среди бесплатных компиляторов Фортрана кроме Watcom Fortran, выделенного в отдельный проект Open Watcom, стоит отметить компилятор от компании Sun Microsystems, входящий в интегрированную среду разработки Oracle Solaris Studio и уже упомянутый ранее GNU Gfortran. Все примеры и задания данного практикума составлены и отлажены с помощью компилятора Gfortran.

В компиляторе Gfortran (free Fortran 95/2003/2008 compiler for GCC, the GNU Compiler Collection, реализующем стандарт [ISO/IEC 1539-1:1997(E)], с поддержкой элементов объектно-ориентированного программирования) реализованы практически все конструкции стандарта Фортран 95 и многие конструкции стандартов Фортран 2003 и Фортран 2008. Обеспечена полная совместимость со стандартом Фортран 77.

В разд. 3 представлено краткое (ориентированное на программирование базовых алгоритмов) описание языка Фортран, в рамках стандарта [ISO/IEC 1539-1:1997(E)], содержащее около 100 коротких отлаженных примеров, дающих наглядное представление об основных языковых конструкциях Фортрана.

Решение зачетных заданий разд. 3 предполагает использование математического аппарата. Для удобства зачетные задания раздела сгруппированы по соответствующим темам.

1. КОМАНДНЫЕ ОБОЛОЧКИ UNIX

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К РАЗД. 1

Первый раздел лабораторного практикума посвящен знакомству с командными оболочками UNIX-систем на примере GNU/Linux. Поскольку в практикуме изучаются и используются минимальные базовые возможности командных интерпретаторов на примере наиболее распространенного из них интерпретатора bash, то подготовка практикума преподавателем и системным администратором, а затем его выполнение студентами возможно практически на любых версиях Linux или других UNIX-подобных систем.

В стандартной для ОС Linux конфигурации каталогов файловой системы (подробное рассмотрение которой не входит в задачу данного лабораторного практикума) предусматривается расположение студенческих групп в директории /home/groups (в дальнейших описаниях и примерах предполагается, что лабораторные работы выполняются гипотетическим студентом группы v100, с домашним каталогом (логином) v100-1).

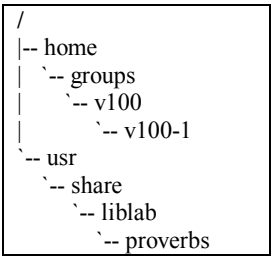


Рис. 1.1. Часть дерева файловой системы, используемая в практикуме

Работая в своем домашнем каталоге и системных каталогах, предназначенных для использования в практикуме (рис. 1.1), студент должен научиться (см. примеры и зачетные задания к разд. 1):

1) использовать абсолютные и относительные пути (от корневого, домашнего и текущего каталога) для навигации по файловой системе (примеры 1.1 ÷ 1.7);

2) осуществлять просмотр информации в доступных каталогах и файлах (примеры 1.8 ÷ 1.15, а так же задание 1.1);

3) создавать и редактировать дерево каталогов и файлов, используя различные варианты записи путей к каталогам и файлам (примеры 1.16 ÷ 1.21, а также задание 1.1);

4) выполнять копирование и перемещение объектов файловой системы с использованием различных способов записи путей к ним (примеры 1.23 ÷ 1.26, задание 1.2);

5) выяснять, какие пользователи и процессы работают в системе и устранять ненужные процессы (примеры 1.27 ÷ 1.31);

6) управлять правами доступа к личным файлам и директориям (примеры 1.32 ÷ 1.39, задание 1.3);

7) понимать на пользовательском уровне основные вопросы, связанные со стандартными потоками ввода/вывода и вводом/выводом команд (примеры 1.33 ÷ 1.44);

8) осуществлять поиск заданных фрагментов текста в файлах (примеры 1.45 ÷ 1.47, задание 1.3);

9) использовать стандартный для UNIX-систем файловый менеджер Midnight Commander (mc) для выполнения вышеуказанных заданий (задания 1.4 ÷ 1.5).

Для успешного выполнения задания 1.3 (поиск заданных фрагментов текста в файлах) преподавателю и системному администратору необходимо подготовить соответствующий каталог с текстовыми файлами в стандартной кодировке UTF-8, для корректного отображения кириллицы. Вместе с тем при подготовке такого каталога желательно комбинировать тексты на кириллице и латинице на тот случай, если по какой-то причине кириллица не отображается. Тогда можно выполнять поиск фрагментов текста на латинице. Чтобы не тратить усилия на придумывание текстов и фрагментов для поиска, целесообразно воспользоваться распространенными в интернете короткими английскими текстами с готовым переводом: например, известными английскими поговорками или популярными законами Мерфи (Murphy).

В дальнейшем описании и примерах данного лабораторного практикума предполагается, что файлы для поиска фрагментов текста размещены в директории /usr/share/liblab/proverbs – это тексты известных английских поговорок с переводом.

Каждый UNIX (Linux)-сервер имеет свое собственное название в сети (Host Name). Все примеры данного лабораторного практикума (включая вход в систему) были протестированы на сервере

Debian GNU/Linux, который называется beta.mephi.ru (точнее, «назывался beta.mephi.ru», поскольку к моменту публикации этого пособия, такой сервер уже не используется). Соответственно, для успешного выполнения лабораторного практикума, необходимо делать поправку: заменяя beta.mephi.ru на имя рабочего сервера.

Все примеры данного лабораторного практикума подразумевают, что их выполняет гипотетический студент группы, имеющий домашним каталог (и логин) v100-1, а приглашение командной строки, после входа в систему выглядит как:

v100-1@beta:~>

Это необходимо учитывать, так как в текстах примеров присутствует именно такое приглашение командной строки, а при выполнении практикума на разных серверах оно может выглядеть подругому.

При редактировании командной строки выполняются следующие действия.

1. Для подтверждения ввода команды необходимо нажать клавишу ввода – обычно это «Enter».
2. Стрелочки «вверх» и «вниз»: перелистывание списка ранее введенных команд.
3. Стрелочки «влево» и «вправо»: перемещение курсора по командной строке.
4. Клавиша «Delete»: удаление символа в позиции курсора.
5. Клавиша «Backspace»: удаление символа слева от курсора.

Далее по тексту в разд. 1 используются следующие стандартные обозначения.

1. Корневой каталог обозначается символом: « / ».
2. Элементы пути также разделяются символом: « / ».
3. Домашний каталог обозначается одним символом: « ~ ».
4. Путь относительно домашнего каталога начинается с комбинации символов: « ~/ ».
5. Текущий каталог обозначается символом: « . ».
6. Путь относительно текущего каталога начинается с комбинации символов: « ./ ».

В табл. 1.1 приведены рассматриваемые в практикуме команды UNIX-подобных операционных систем.

Таблица 1.1. Используемые команды UNIX

Команда	Операция
pwd	Показать имя (путь) текущей директории
ls	Показать список файлов и подкаталогов текущей директории
ls .	Также показать список файлов и подкаталогов текущей директории
ls –F	Отобразить содержание текущей директории с добавлением к именам символов, характеризующих тип файлов
ls *путь*	Отобразить содержание директории по указанному полному или относительному пути
tree	Показать дерево каталогов и файлов для текущей директории
tree .	Также показать дерево каталогов и файлов для текущей директории
tree ~	Показать дерево домашней директории;
tree *путь*	Показать дерево каталогов и файлов для директории по указанному полному или относительному пути
cd ~	Перейти в домашнюю директорию
cd	Также перейти в домашнюю директорию
cd –	Перейти в бывшую текущую директорию
cd ..	Перейти в директорию уровнем выше
cd ../..	Перейти в директорию на два уровня выше
cd *путь*	Сделать текущей директорию по указанному полному или относительному пути
mkdir *имя*	Создать каталог в текущей директории
mkdir *имя1 имя2*	Создать в текущей директории два каталога
mkdir *путь*	Создать директорию по указанному полному или относительному пути;
mkdir –p *путь*	Создать ветку дерева по указанному полному или относительному пути
ls	Просмотреть текущую директорию
ls .	Также просмотреть текущую директорию
ls ~	Просмотреть домашнюю директорию
ls ..	Просмотреть директорию уровнем выше
ls *путь*	Просмотреть директорию по указанному пути
ls –l	Просмотреть права доступа на файлы и подкаталоги текущей директории
more *имя*	Просмотреть многостраничного текстового файла, «Пробел» – листать страницы, «q» – выходить из режима просмотра
cat *имя*	Просмотреть текстовый файл

Окончание табл. 1.1

Команда	Операция
cat < *имя*	Просмотреть текстовый файл перенаправлением текста в stdin
cat > *имя*	Создать текстовый файл перенаправлением текста в stdout
cat >> *имя*	Добавить текст к файлу не деструктивным перенаправлением в stdout
rmdir *имя*	Удалить каталог в текущей директории
rmdir *имя1 имя2*	Удалить два каталога в текущей директории
rmdir *путь*	Удалить каталог по указанному полному или относительному пути
rm –rf *путь*	Удалить каталог по указанному полному относительному пути со всем содержимым
rm –rf *путь1 путь2*	Удалить каталоги по указанным полным или относительным путям со всем содержимым
cp *что куда*	Копировать файлы или директории, «что» и «куда», их имена или пути
mv *что куда*	Перемещение файлов или директорий, «что» и»куда» их имена или пути;
whoami	Печатать имени пользователя, работающего в системе;
who	Печаттать списка пользователей, работающих в системе
ps	Печатать списка процессов пользователя
jobs	Печатать списка внутренних процессов командного интерпретатора bash
bg	Переводить процесс в фоновый режим (background)
fg	Выводить процесс из фонового режима на передний план (foreground)
chmod u-r-w-x *имя*	Отменить для пользователя права чтения, записи и использования на файл или каталог
chmod u+r+w+x *имя*	Возвратить пользователю права чтения, записи и использования на файл или каталог
sort *имя*	Сортировка строк текстового файла в алфавитном порядке;
sort –r *имя*	Сортировка строк текстового файла в обратном алфавитном порядке
grep *образец путь*	Поиск файлов, содержащих образец текста, с выдачей имен файлов и строк, содержащих заданный образец
grep –l *образец путь*	Поиск файлов, содержащих образец текста, с выдачей только имен файлов

1.1. СЕАНС РАБОТЫ В UNIX-СИСТЕМЕ

1.1.1. Терминал и командная строка

Одним из наиболее характерных свойств UNIX-систем (как уже отмечалось в предисловии) является взаимодействие операционной системы (ОС) и пользователя посредством виртуального устройства, называемое *терминалом* (системной клавиатуры и экрана монитора, работающего в текстовом режиме). Терминал должен обеспечивать обмен текстовыми данными между пользователем и системой, передавать системе управляющие команды. При этом диалог пользователя с UNIX-системой выглядит как обмен текстами. Вводимый текст немедленно отображается на мониторе, за редким исключением, например при вводе пароля. Для удаления последнего неверно введённого символа используется клавиша Backspace, для подтверждения ввода – клавиша Enter.

Текстовый принцип работы позволяет отвлечься от конкретных частей компьютера, рассматривая терминал как единое оконечное устройство. В общем случае терминал – точка входа пользователя в систему. В роли терминала может выступать специальная программа, например Xterm или, как в данном практикуме, клиент сетевых протоколов PuTTY. Важнейшей задачей терминала в независимости от принципов его реализации является обеспечение взаимодействия пользователя с операционной системой. В данном практикуме пользователь взаимодействует с UNIX-системой через *интерфейс командной строки*, который требует от пользователя минимального представления об основных «внутренних» принципах работы ОС.

Работая с командной строкой, пользователь взаимодействует со специализированной программой, называемой *интерпретатором команд* (*командной оболочкой,* или просто *оболочкой* – англ. *shell*).

Командный интерпретатор обеспечивает базовые возможности управления ресурсами компьютера посредством ввода команд с клавиатуры, запуска на исполнение пакетных файлов, называемых в UNIX сценариями или скриптами. Командный интерпретатор также обеспечивает возможность вызова утилит и прикладных программ. По сути, командный интерпретатор реализует язык программирования, элементами которого помимо команд операционной системы являются, например, условные операторы, операторы

цикла и другие элементы, присущие языкам программирования высокого уровня. Вместе с тем это узкоспециализированный язык, ориентированный на разработку сценариев, управляющих работой ОС. В таких UNIX-системах, как Linux и FreeBSD, самым распространенным командным интерпретатором является *bash* (Bourne Again Shell), как правило, он используется по умолчанию, в том числе и в данном практикуме. Помимо *bash* есть большое семейство других командных интерпретаторов: *csh* (C shel), *ksh* (Korn Shell) и т.д. Как правило, у пользователя есть возможность менять командный интерпретатор, используемый по умолчанию.

1.1.2. Логины, пароли и доступ к серверу

Лабораторный практикум по UNIX может выполняться на домашнем компьютере или в компьютерном классе. В последнем случае нужно получить *логин* и *пароль* для входа на сервер локальной сети. Если компьютерный класс работает под управлением Windows, то логин и пароль выделяется один на всю студенческую группу, т.е. пользователем локальной сети компьютерного класса является группа, а не отдельный студент.

Практикум по UNIX выполняется на доступном через Интернет сервере beta.mephi.ru. Для входа на сервер каждый студент получает индивидуальный логин и пароль.

Таким образом, для успешного выполнения практикума необходимы два логина (и пароля) – один для входа в Windows (при работе в компьютерном классе на Windows), а другой непосредственно для входа на UNIX-сервер (с любого компьютера, подключенного к Internet). Все необходимые логины и пароли необходимо получить перед началом работы у преподавателя или системного администратора.

Для доступа к серверу через интернет используется свободно распространяемый клиент сетевых протоколов PuTTY (в свободном доступе по адресу http://www.putty.org/), работающий по защищенному протоколу SSH (Secure Shell – безопасная оболочка). SSH шифрует все передаваемые данные, в том числе пароли. Программа PuTTY активируется двойным щелчком, по пиктограмме на рабочем столе Windows (рис. 1.2).

Рис. 1.2. Пиктограмма клиента сетевых протоколов PuTTY

Далее указана последовательность действий при работе с интерфейсом PuTTY, сводящаяся к заданию необходимых параметров для категорий «Translation» (рис. 1.3) и «Session» (рис. 1.4).

1. Выбрать категорию «Translation».
2. Установить кодировку UTF-8.
3. Перейти к категории «Session».
4. В поле «Host Name» ввести имя сервера, к которому необходимо получить доступ (betta.mephi.ru).
5. Для типа соединения «Connection type» выбрать защищенный протокол SSH.
6. В поле сохраненных сессий «Saved Sessions» указать имя, под которым PuTTY запомнит только что созданную сессию (например, beta).
7. Кликнуть «Save» для сохранения сессии beta (она появится в списке «Saved Sessions»).
8. Выбрать сессию beta в списке «Saved Sessions».
9. Получить доступ к серверу, кликнув «Open» или двойным кликом по beta.

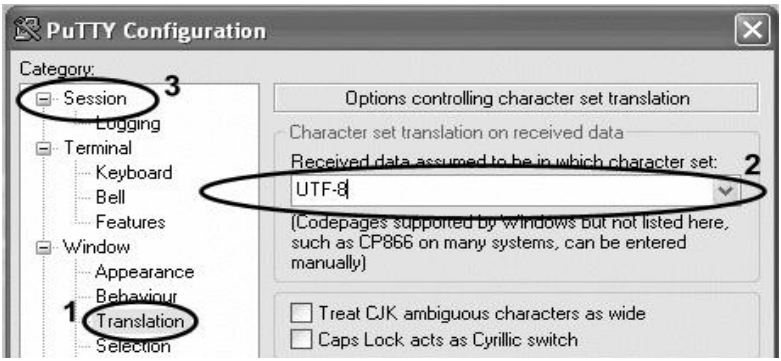


Рис. 1.3. Параметры категории «Translation»

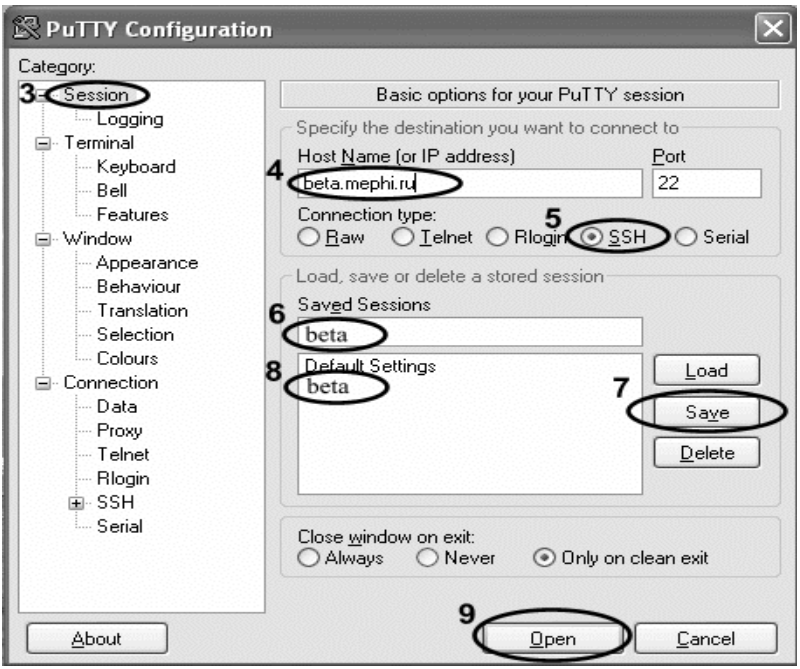


Рис. 1.4. Параметры категории «Session»

Сессия beta при следующих запусках PuTTY будет присутствовать в списке сохраненных сессий, поэтому для доступа к серверу достаточно выполнить только п. 8 и/или п. 9.

1.1.3. Вход в систему/завершение работы

При успешном запуске PuTTY появляется окно сессии операционной системы GNU/Linux – одной из наиболее распространенных UNIX-систем (рис. 1.5).

Всюду далее в примерах фигурирует гипотетический студент с логином и домашним каталогом v100-1. *Выполняя эти примеры, необходимо заменять* v100-1 *на свой личный домашний каталог.*

Работа с системой начинается с авторизации.

1. В поле «Login as» вводится личный логин.
2. В поле «Password» вводится личный пароль. Пароль при вводе не отображается ни какими символами.
3. При правильном вводе логина и пароля появится приглашение командной строки командного интерпретатора bash.

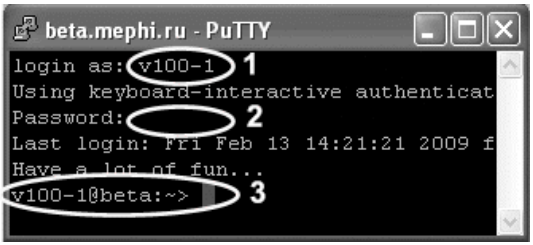


Рис. 1.5. Вход в систему на сервере beta.mephi.ru

В операционных системах клона UNIX, написанное в верхнем и нижнем регистре (маленькими и большими буквами), это не одно и то же. Например, v1 это не V1 и т.д.

Ввод логина, пароля, а также любой команды командного интерпретатора bash подтверждается клавишей «Enter».

Новичок, работающий с сессией UNIX в режиме удаленного терминала, когда у него «что-нибудь не получается», часто хочет кликнуть по «кнопке с крестиком» в правом верхнем углу окна. Поэтому первым делом при входе в UNIX важно усвоить две вещи.

1. На первом этапе изучения UNIX-систем желательно просто забыть о существовании «кнопки с крестиком» в правом верхнем углу окна UNIX-сессии (рис. 1.6).
2. Важно запомнить, что выход из UNIX-сессии осуществляется командой logout, или «Ctrl+D» (см. рис. 1.6).

Для того чтобы понять важность завершения работы командой logout (или «Ctrl+D»), а не «кнопкой с крестиком», кратко рассмотрим понятие *процесс*. Когда пользователь входит в систему и начинает взаимодействовать с командной оболочкой, то ОС порождает процесс сеанса работы пользователя.

Щелкая мышкой по «кнопке с крестиком», пользователь закрывает окно сессии UNIX, но все процессы этой сессии продолжают существовать, превращаясь в брошенные пользователями про-

граммы-призраки. Незавершенные процессы не выполняют никакой полезной работы, но продолжают потреблять ресурсы вычислительной системы, снижая ее производительность. Более подробно процессы рассматриваются в разд. 3.

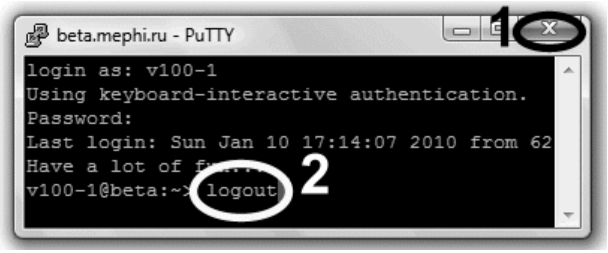


Рис. 1.6. Выход из сессии UNIX

Процесс – динамический программный объект. Ему присваиваются атрибуты, связанные с личным разделом пользователя: ресурсы аппаратуры (например, место на диске, квота оперативной памяти), привилегия (например, возможность доступа к файлам и т.п.). Когда пользователь вводит команды или запускает программы, операционная система порождает для них дополнительные процессы.

1.2. РАБОТА С ФАЙЛОВОЙ СИСТЕМОЙ UNIX

1.2.1. Файлы и каталоги. Абсолютный (полный) путь

Работа пользователя на компьютере сводится к взаимодействию с приложениями и манипулированию различными файлами. Базовый навык при изучении новой операционной системы – умение находить необходимые для работы файлы. Для успешной работы в UNIX необходимо научиться работать с каталогами и файлами, используя интерфейс командной строки.

Файл является основной единицей хранения данных и программ. Файл – именованная область памяти. Обычно файлы временно или постоянно хранятся во внешней памяти компьютера и загружаются в оперативную память при работе приложений. Кроме имени, файлы характеризуются целым рядом таких атрибутов, как размер, время создания и т.п. Операционная система и прикладные

программы (приложения) получают доступ к файлу по его имени. Максимальная длина имени файла в UNIX составляет 256 символов, включая расширение. Имя и расширение разделяются точкой. Расширение указывает на тип информации или на приложение, которым может быть открыт этот файл.

Файлы хранятся в системе вложенных каталогов (директорий), образующих *файловую систему*. Файловая система представляет собой древовидную структуру. Часть дерева файловой системы UNIX-сервера beta, доступная и необходимая при выполнении практикума, представлена на рис. 1.7 (все примеры практикума приведены для пользователя с логином v100-1).

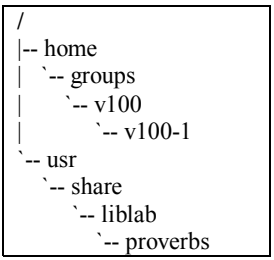


Рис. 1.7. Часть дерева файловой системы, используемая в практикуме

Местоположение файла или каталога (своего рода, адрес на дереве файловой системы) определяется понятием *пути*. Пример записи пути: /home/groups/v100/v100-1 – это так называемый *полный путь*, в данном случае он означает, что каталог v100-1 является подкаталогом каталога v100, который расположен в каталоге groups, находящемся в каталоге home, лежащем в *корневом каталоге*. Точкой отсчета полного пути всегда является корневой каталог. Даже, если в файловой системе нет ни одного файла, то она будет состоять из единственного каталога – корневого, он, как разделители пути, обозначается прямым слешем «/» (наклонной чертой слева направо – см. самый верхний символ на рис. 1.7).

Говоря: «каталог (директория) содержит файлы» или «файл находится в каталоге», следует понимать, что каталог не является физической областью памяти, вмещающей другие каталоги и файлы. По сути, каталог – особого рода файл, назначением которого является хранение списка отнесенных к нему файлов (в том числе и

«содержащихся» в нем подкаталогов) и их атрибутов. По этому, когда говорится о древовидной структуре и организации файловой системы, подразумевается, что речь идет о не физической, а о логической структуризации файлов. Только с учетом этого контекста далее будем использовать общепринятую терминологию: «директория содержит файлы, подкаталоги» и т.д.

1.2.2. Домашний и текущий каталог. Относительный путь

Каждому пользователю, зарегистрированному в системе, выделяется личный, или *домашний, каталог*. В этот каталог пользователь попадает сразу после входа в систему, здесь он может создавать и хранить собственные каталоги и файлы. Пользователь v100-1, войдя в систему, получает в свое распоряжение домашний каталог, полный путь которого: /home/groups/v100/v100-1. Сразу после входа в систему этот каталог является также и *текущим каталогом*. Для ответа на вопрос, «какой каталог является текущим?», в UNIX предусмотрена команда pwd (print working directory) – пример 1.1.

Команда pwd – просмотр пути текущего каталога

Здесь и далее в форматах команд, в квадратные скобки заключаются необязательные элементы в записи команды.

Пример 1.1. Просмотр пути текущего каталога

```
v100-1@beta:~> pwd
/home/groups/v100/v100-1
v100-1@beta:~>
```

Что такое текущий каталог? Для чего он нужен и чем удобен? Полный (абсолютный) путь описывает местоположение каталога или файла в дереве файловой системы относительно корня (корневого каталога). Теоретически этого вполне достаточно для манипулирования файлами и каталогами. При этом постоянно набирать на клавиатуре длинные, содержащие много символов, абсолютные пути долго и неудобно. Если пользователь работает с файлами, находящимися в каком-либо каталоге, то целесообразно и логично как-то зафиксировать путь к этому каталогу, т.е. сделать этот ката-

лог текущим. После этого можно работать с файлами выбранного таким образом каталога уже без указания длинного пути. Более того, появляется возможность записи более коротких путей относительно текущего каталога, обозначаемого как «.» – точка, к близлежащим (на дереве файловой системы) каталогам и файлам.

В UNIX-системах короткие (относительные) пути могут записываться относительно текущего или домашнего каталога, обозначаемого как «~» – символ волны или «тильда».

Для файлов и каталогов, находящихся непосредственно в текущем каталоге, путь можно не указывать совсем или в ряде случаев путь указывается как «*./имя*». По такому же принципу указывается путь к объектам, расположенным непосредственно в домашнем каталоге, как «*~/имя*». Если текущий или домашний каталог содержит вложенные каталоги, то, как и в случае абсолютного пути, записывается последовательность вложенных имен, разделяемая прямым слешем «/». Если же наоборот, необходимо указать путь к каталогам верхнего уровня, то каждый переход на уровень выше обозначается как «..» – две последовательные точки. Например, путь на два уровня вверх относительно текущего каталога будет записываться как «./../..», а такой же путь относительно домашнего каталога будет выглядеть как «~/../..».

Для задания и изменения текущего каталога в UNIX-системах предусмотрена команда cd (change directory).

Команда cd – задание или изменение текущего каталога

Пример 1.2 показывает, как по абсолютному пути перейти в каталог /usr/share/liblab/proverbs (сделать его текущим). Команда отделяется от пути одним или несколькими пробелами. Вернуться в домашний каталог можно аналогичным образом (пример 1.3).

Пример 1.2. Изменение текущего каталога

```
v100-1@beta:~> cd /usr/share/liblab/proverbs
v100-1@beta:/usr/share/liblab/proverbs> pwd
/usr/share/liblab/proverbs
v100-1@beta:/usr/share/liblab/proverbs>
```

Пример 1.3. Возвращение в домашний каталог по полному пути

```
v100-1@beta:/usr/share/liblab/proverbs>cd /home/groups/v100/v100-1
v100-1@beta:~>pwd
/home/groups/v100/v100-1
v100-1@beta:~>
```

Существует и более простой способ перехода в домашний каталог (пример 1.4). Более того, если использовать команду cd вообще без параметров – результат будет тот же.

Пример 1.4. Короткое возвращение в домашний каталог

```
v100-1@beta:/usr/share/liblab/proverbs>cd ~
v100-1@beta:~>pwd
/home/groups/v100/v100-1
v100-1@beta:~>
```

Как уже отмечалось, запись путей к каталогам и файлам возможна не только относительно корневого каталога, но и относительно текущего и домашнего каталога. Причем пути записываются не только «вниз» – по направлению от корня, но и «вверх» – по направлению к корню файловой системы. Путь вверх по дереву файловой системы может состоять как из одного, так и из многих шагов относительно текущей директории (пример 1.5).

Пример 1.5. Переходы по дереву «к корню»

```
v100-1@beta:~> cd ./..
v100-1@beta:/home/groups/v100> pwd
/home/groups/v105
v100-1@beta:/home/groups/v100> cd ./../..
v100-1@beta:/home> pwd
/home
v100-1@beta:/home>
```

Можно также выстраивать пути, состоящие из шагов к корню, а затем от корня, как путь ./../../usr (пример 1.6).

Пример 1.6. Переход по пути «к корню – от корня»

```
v100-1@beta:/home> cd ./../usr
v100-1@beta:/usr> pwd
/usr
v100-1@beta:/usr>
```

Если ставится задача перейти, например, из текущего каталога /usr в каталог v100, то оптимальным будет использование пути относительно домашнего каталога v100-1, поскольку это будет самая короткая запись (пример 1.7).

Пример 1.7. Переход по пути «от домашнего каталога» – самый короткий

```
v100-1@beta:/usr>~/..
v100-1@beta:/home/groups/v100>pwd
/home/groups/v100
v100-1@beta:/home/groups/v100>cd ./v100-1
v100-1@beta:~>pwd
/home/groups/v100/v100-1
v100-1@beta: ~>
```

На практике пользователь сам определяет, какие способы записи путей лучше использовать при навигации по файловой системе, руководствуясь удобством записи и личным опытом.

1.2.3. Просмотр каталогов и файлов

Важным элементом навигации по файловой системе является просмотр структуры каталогов и файлов, а также их содержания.

Команда tree – просмотр [части] дерева файловой системы

Пример 1.8 показывает использование команды tree для просмотра структуры каталога proverbs в виде дерева. Синим цветом на экране отображаются каталоги, белым цветом – файлы. Этого не видно на черно-белом листе бумаги, поэтому приведен пример команды tree –F. Ключ «F» обеспечивает отображение слеша «/» после имен каталогов при любой цветовой гамме, соответственно у имен файлов слеш отсутствует.

Пример 1.8. Древовидная структура каталога proverbs

```
v100-1@beta: ~>tree –F /usr/share/liblab/proverbs
/usr/share/liblab/proverbs
|-- body/
|   |-- birds/
|   |   |-- birds
|   |   `-- goose
|   |-- economy/
|   |   |-- money
|   |   `-- thrifty
|   `-- life/
|       |-- food
|       `-- work
`-- spirit/
```

```
   |-- happyend/
   |   |-- final
   |   `-- good
   |-- mind/
   |   |-- philosph
   |   `-- strange
   `-- opposite/
       |-- life
       `-- war
v100-1@beta: ~>
```

Команда ls – просмотр списка объектов в каталоге

Команда имеет много параметров. Наиболее часто используется:

ls без параметров – вывод файлов и подкаталогов текущего каталога;

ls путь/имя_ каталога – вывод файлов и подкаталогов произвольного каталога;

ls --color – вывод содержимого каталога в цвете:

зелёный – исполняемые файлы;

синий – каталоги;

чёрныё – обычные файлы;

ls -l путь/имя_ файлаа – вывод типа файла, владельца файла, прав доступа для разных категорий пользователей и т.д.;

ls -F – ключ «F» обеспечивает отображение слеша «/» после имен каталогов при любой цветовой гамме (пример 1.9).

Пример 1.9. Просмотр списка каталогов и файлов содержащихся в proverbs

```
v100-1@beta:~> ls -F /usr/share/liblab/proverbs
body/  spirit/
v100-1@beta:~>
```

Пример 1.9 демонстрирует применение команды ls –F для просмотра списка каталогов, являющихся подкаталогами proverbs. Смысл ключа «F» тот же, что и в предыдущем примере.

Применяя команды *tree* и *ls* без указания пути можно увидеть структуру и содержание текущего каталога.

Вопрос о просмотре содержания файла решается применением команды *cat* с указанием имени файла (пример 1.10), если он находится в текущей директории или пути к файлу (пример 1.11).

Команда cat – вывод содержания (текста) файла

Пример 1.10. Просмотр файла в текущей директории

```
v100-1@beta:~> cd /usr/share/liblab/proverbs/body/birds/
v100-1@beta:/usr/share/liblab/proverbs/body/birds> cat goose
  He can't say boo to a goose.
  Он не скажет "Бу-у" даже гусю,
  т.е. он и мухи не обидит.

  Everything is lovely and the goose hangs high.
  Все прекрасно, и гусь висит высоко.
  Аналог: дело на мази; все идет как по маслу.
v100-1@beta:/usr/share/liblab/proverbs/body/birds>
```

Пример 1.11. Просмотр файла из произвольной директории

```
v100-1@beta:/usr/share/liblab/proverbs/body/birds>cd ~
v100-1@beta:~> cat /usr/share/liblab/proverbs/body/birds/goose
  He can't say boo to a goose.
  Он не скажет "Бу-у" даже гусю,
  т.е. он и мухи не обидит.
  Everything is lovely and the goose hangs high.
  Все прекрасно, и гусь висит высоко.
  Аналог: дело на мази; все идет как по маслу.
v100-1@beta:~>
```

Команда **cat** позволяет также выполнять следующие операции:

1. Объединить несколько файлов в один с помощью оператора >.
2. Присоединить файл к существующему файлу с помощью оператора >>.
3. Создать копию файла с новым именем с помощью оператора >.
4. Создать новый текстовый файл без использования текстового редактора (пример 1.12).

Пример 1.12. Создание текстового файла через перенаправление ввода

```
v100-1@beta:~> cat > papavas.txt
Папа у Васи силен в математике<Enter>
Учится папа за Васю весь год. <Enter>
Где это видано, где это слыхано -<Enter>
Папа решает, а Вася сдает. <Enter><Ctrl + D>
v100-1@beta:~> ls
papavas.txt
v100-1@beta:~>
```

1.2.4. Создание и удаление каталогов и файлов

Создание и удаление файлов является повседневной необходимостью в работе пользователя. Для организации хранения личных файлов пользователю также необходимо создавать новые каталоги и удалять ненужные.

Создать новый каталог можно воспользовавшись командой mkdir (make directory). Пример 1.13 демонстрирует создание каталога с именем «firstcat» в домашней директории. Команда ls позволяет убедиться в успешном создании каталога.

.

Команда mkdir – создание нового каталога (директории)

Пример 1.13. Создание нового каталога (директории)

```
v100-1@beta:~> mkdir  firstcat
v100-1@beta:~> ls –F
firstcat
v100-1@beta:~>
```

Команда rmdir – удаление существующего каталога

Пример 1.14. Удаление существующего каталога

```
v100-1@beta:~> rmdir  firstcat
v100-1@beta:~> ls –F
v100-1@beta:~>
```

Для удаления каталогов применяется команда rmdir (remove directory), которой в качестве параметра указывается имя удаляемого каталога, допустим, это будет firstcat (пример 1.14). Команда ls позволяет удостовериться, что домашний каталог пуст.

Для создания текстового файла можно воспользоваться уже известной командой cat с перенаправлением стандартного вывода в файл (подробнее об этом далее, в п. 1.3.2). Пример 1.12 демонстрирует создание текстового файла «papavas.txt», Перенаправление вывода задается символом «>« (знак «больше»). После подтверждения команды клавишей «Enter», вводится текст «Папа у Васи …». Каждая напечатанная строчка подтверждается «Enter». Поскольку команда cat порождает процесс, то завершить его нужно «Ctrl + D». Просмотр созданного файла проиллюстрирован в примерах 1.10 и 1.11.

Для просмотра больших текстовых файлов, текст которых не помещается на одном экране, используется команда постраничного просмотра more. «Перелистывание» страниц осуществляется клавишей «Пробел», клавиша «Enter» позволяет просматривать файл построчно. Возврат в командную строку UNIX происходит при нажатии клавиши «q» (пример 1.15).

Команда more – постраничный просмотр файла

Пример 1.15. Постраничный просмотр файла

```
v100-1@beta:~> more papavas.txt<Enter>
Папа у Васи силен в математике<Enter>
Учится папа за Васю весь год. <Enter>
Где это видано, где это слыхано -<Enter>
Папа решает, а Вася сдает. <Enter><Ctrl + D>
v100-1@beta:~> ls
papavas.txt
v100-1@beta:~>
```

Для удаления файлов используется команда rm (пример 1.16). Для контроля удаления, как обычно, через ls.

Команда rm – удаление файла

Пример 1.16. Удаление файла

```
v100-1@beta:~> rm papavas.txt
v100-1@beta:~> ls
v100-1@beta:~>
```

1.2.5. Пример построения дерева каталогов

В качестве подготовки к выполнению зачетного задания предлагается построить в домашней директории (в описании лабораторного практикума это каталог /home/groups/v100/v100–1) дерево каталогов и файлов (рис. 1.8). Каталоги и подкаталоги дерева создаются с использованием различных вариантов команды mkdir и способов описания путей к каталогам и файлам (как абсолютных, так и относительных путей).

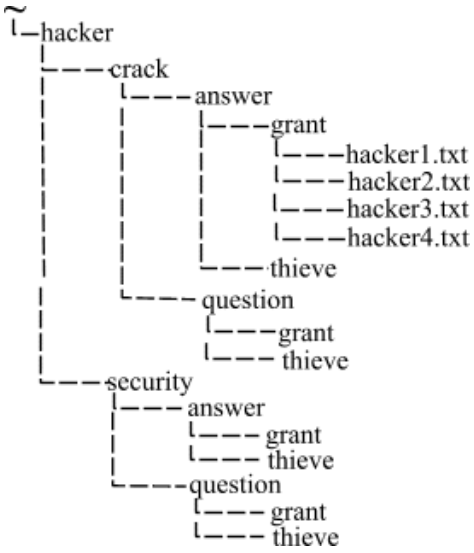


Рис. 1.8. Дерево каталогов и файлов для построения

Тексты файлов для дерева (см. рис. 1.8):

hacker1.txt
Если в сервер вашей фирмы
Вдруг забрался злобный хакер

hacker2.txt
То уверены вы будьте
Что залезет он еще

hacker3.txt
Абсолютно все пароли
Поместите в файл readme

hacker4.txt
Пусть порадуется хакер
Он ведь тоже человек

Команда mkdir –p создает цепочку вложенных каталогов

Таким образом, команда mkdir –p позволяет создавать каталоги не по одному, как простая команда mkdir, а сразу создает цепочку вложенных каталогов. При создании таких цепочек можно использовать полный путь, как /home/groups/v100/v100-1/hacker|crack/answer/grant (пример 1.17) или относительный (пример 1.18). Чтобы увидеть всю созданную ветвь, нужно воспользо-

ваться командой tree, поскольку команда ls отобразит только каталог hacker.

Пример 1.17. Создание ветви дерева (полный путь)

```
v100-1@beta:~>mkdir –p /home/groups/v100/v100–1/hacker/crack/answer/grant
v100-1@beta:~> ls –F
hacker/
v100-1@beta:~> tree –F
.
`-- hacker/
   `-- crack/
      `-- answer/
         `-- grant/
4 directories, 0 files
v100-1@beta:~>
```

Ветка hacker/crack/answer/thieve строится с использованием пути относительно текущей директории (пример 1.18), команда все та же с помощью mkdir –p. Контроль построения ветви дерева осуществляется командой tree.

Пример 1.18. Создание ветви дерева (путь от текущей директории – от корня)

```
v100-1@beta:~> mkdir –p ./hacker/crack/answer/thieve
v100-1@beta:~> tree
.
`-- hacker
   `-- crack
      `-- answer
         |-- grant
         `-- thieve
5 directories, 0 files
v100-1@beta:~>
```

Ветви hacker/crack/question/grant и hacker/crack/question/grant достраиваются относительно директории hacker/crack/answer/grant. Сначала ее надо сделать текущей, а затем использовать пути «сначала к корню, затем от корня» (пример 1.19).

Если сделать текущей директорией hacker/crack/answer/grant или hacker/crack/answer/thieve, то для любой из этих директорий относительные пути в каталог hacker/crack/question/grant или в каталог hacker/crack/question/thieve» будут выглядеть совершенно одинако-

во: ../../question/grant и ../../question/thieve. В приведенном примере используется директория «hacker/crack/answer/grant».

Пример 1.19. Создание ветви дерева (путь «к корню – от корня»)

```
v100-1@beta:~/lab1> cd hacker/crack/answer/grant
v100-1@beta:~/hacker/crack/answer/grant>mkdir -p ../../question/grant
v100-1@beta:~/hacker/crack/answer/grant>mkdir -p ../../question/thieve
v100-1@beta:~/hacker/crack/answer/grant> tree ~/hacker /home/groups/v100/v100-1/hacker

`-- crack
   |-- answer
   |   |-- grant
   |   `-- thieve
   `-- question
       |-- grant
       `-- thieve
7 directories, 0 files
v100-1@beta:~/hacker/crack/answer/grant>
```

Командой tree просматривается не текущая директория (в ней ничего нет), а директория hacker, путь к которой указан относительно домашней директории, всегда обозначаемой символом «~» (волна или тильда).

Оставшуюся часть дерева (~/hacker/security) каждый может достроить в соответствии со своим собственными предпочтениями.

В заключение остается создать текстовые файлы в директории ~/hacker/crack/answer/grant.

Самый простой способ – создавать файлы непосредственно в самой директории, т.е. зайти в директорию и создать файл. Однако в учебных целях необходимо научиться создавать файлы, используя полные и относительные пути к файлам, аналогично тому, как создавались ветви дерева ~/hacker.

Пример 1.20. Создание файла с использованием полного пути

```
v100-1@beta:~/lab1> cd
v100-1@beta:~> cat > /home/groups/v100/v100–1/hacker/security/answer/grant/hacker2.txt
 <Ctrl+D>
То уверены вы будьте
Что залезет он еще <Ctrl+D>
v100-1@beta:~>cat                                   /home/groups/v100/v100–
1/hacker/security/answer/grant/hacker2.txt <Enter>
То уверены вы будьте
Что залезет он еще
v100-1@beta:~>
```

Пример 1.20 показывает, как создать файл hacker2.txt с использованием полного пути к нему.

Файл hacker3.txt создается с использованием относительного пути из директории ~/hacker/crack/question/grant, или с таким же успехом из директории ~/hacker/crack/question/thieve. Соответственно, перед созданием файла одну из этих директорий необходимо сделать текущей (пример 1.21).

Пример 1.21. Создание файла с использованием относительного пути

```
v100-1@beta:~> cd hacker/crack/question/grant
v100-1@beta:~/hacker/crack/question/grant> cat > ../../answer/grant/hacker3.txt
Абсолютно все пароли
Поместите в файл readme<Ctrl+D>
v100-1@beta:~/hacker/crack/question/grant> cat ../../answer/grant/hacker3.txt
Абсолютно все пароли
Поместите в файл readme
v100-1@beta:~/hacker/crack/question/grant>
```

1.2.6. Копирование и перемещение файлов

Копирование и перемещение являются неотъемлемыми функциями управления конфигурацией файловой системы. Копирование дает возможность тиражировать файлы и целые каталоги в необходимом количестве экземпляров, а перемещение позволяет менять местоположение информации в связи с необходимостью решения новых задач. Для понимания и выполнения примеров в домашней директории необходимо наличие дерева ~/hacker, построение которого проиллюстрировано в п. 1.2.5.

Копирование файлов осуществляется командой cp (сокращение от copy – копировать). Формат команды копирования: cp **что куда**, где «что» – это путь к копируемому файлу или директории, а «куда» – путь к копии (файлу или директории). При этом копия может иметь другое имя.

В качестве учебной задачи предлагается скопировать файлы из директории hacker/crack/answer/grant в hacker/crack/question/thieve (пример 1.22), используя различные способы записи путей.

Пример 1.22. Учебно-тренировочное дерево ~/hacker

```
v100-1@beta:~> tree ~/hacker
/home/groups/v100/v100-1/hacker
```

```
`-- hacker
   |-- crack
   |  |-- answer
   |  |  |-- grant
   |  |  |  |-- hacker1.txt
   |  |  |  |-- hacker2.txt
   |  |  |  |-- hacker3.txt
   |  |  |  `-- hacker4.txt
   |  |  `-- thieve
   |  `-- question
   |     |-- grant
   |     `-- thieve
   `-- security
      |-- answer
      |  |-- grant
      |  `-- thieve
      `-- question
         |-- grant
         `-- thieve

15 directories, 4 files
v100-1@beta:~>
```

Команда cp – копирование файлов

Пример 1.23. Копирование с длинными путями

```
v100-1@beta:~> cp hacker/crack/answer/grant/hacker1.txt  hacker/crack/question/thieve
v100-1@beta:~>ls hacker/crack/question/thieve
hacker1.txt
v100-1@beta:~>
```

Для удобства копирования нужно правильно выбирать текущую директорию, чтобы избегать написания длинных путей (см. пример 1.23). В зависимости от рабочей ситуации в качестве текущей директории лучше выбирать каталог-источник (пример 1.24) или каталог назначения, в который осуществляется копирование (пример 1.25). Имя файла-копии может отличаться от имени файла-оригинала (см. пример 1.24).

Пример 1.24. Копирование из текущей директории с изменением имени

```
v100-1@beta:~> cd hacker/crack/answer/grant
v100-1@beta:~/hacker/crack/answer/grant>cp hacker2.txt  ../../question/thieve/hck2.txt
v100-1@beta:~/hacker/crack/answer/grant>ls  ~/hacker/crack/question/thieve
hacker1.txt   hck2.txt
v100-1@beta:~/hacker/crack/answer/grant>
```

Конечная точка во второй строке примера 1.25 является обозначением текущего каталога.

Пример 1.25. Копирование в текущую директорию

```
v100-1@beta:~/hacker/crack/answer/grant> cd ../../question/thieve
v100-1@beta:~/hacker/crack/question/thieve> cp ../../answer/grant/hacker3.txt  .
v100-1@beta:~/hacker/crack/question/thieve> ls
hacker1.txt   hacker3.txt   hck2.txt
v100-1@beta:~/hacker/crack/question/thieve>
```

Операции с группами файлов. Использование масок

Возможно одновременное копирование группы файлов, как в примере 1.26. Для обозначения группы файлов может быть использована маска – «*» (символ «звездочка»). Маска обозначает группу символов. Например, «*» означает «группа любых символов имени файла», аналогичный результат копирования был бы получен при использовании масок «*.*» – файлы с любым именем и любым расширением и «hacker*» (файлы с началом «hacker» и любым продолжением) или «hacker*.*» (файлы с началом «hacker», любым завершением имени и любым расширением).

Пример 1.26. Копирование с применением маски

```
v100-1@beta:~/hacker/crack/answer/grant> cd ../../question/thieve
v100-1@beta:~/hacker/crack/question/thieve> cp ../../answer/grant/*
v100-1@beta:~/hacker/crack/question/thieve> ls
hacker1.txt   hacker2.txt   hacker3.txt   hacker4.txt   hck2.txt
v100-1@beta:~/hacker/crack/question/thieve>
```

Пример 1.23 демонстрирует, как копировать файл hacker1.txt, находясь в домашней директории. В предыдущих примерах начало пути из текущей директории начиналось с «./». Такой вариант – универсальный, однако если речь идет не об исполнимых файлах, а о файлах данных, то начало «./» не является обязательным. Очевидно, что текстовые файлы – файлы данных.

С практической точки зрения перемещение файлов отличается от копирования заменой команды cp на mv (move – перемещать). При этом в отличие от копирования перемещаемый файл (файл-оригинал) удаляется из директории-источника. В качестве упражнения предлагается переместить файлы из hacker/crack/answer/grant в директорию hacker/security/answer/grant, по аналогии с примерами 1.24 – 1.26.

1.3. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ UNIX

1.3.1. Пользователи и учетные записи

Для каждого зарегистрированного в UNIX-системе пользователя существует *учётная запись*, в которой система хранит информацию о пользователе. В рамках практикума представляют интерес следующие пункты учетной записи:

- системное имя (user name) – имя, под которым пользователь входит в систему;
- домашний каталог (home directory) – каталог (директория), в который пользователь попадает при входе в систему; пользователь v100-1 попадает в каталог /home/groups/v100/v100-1;
- начальная оболочка (login shell) – командный интерпретатор, который запускается для пользователя при входе в систему; для пользователя v100-1 и других студентов по умолчанию запускается командный интерпретатор bash.

В связи с тем, что UNIX – многопользовательская операционная система, в ней может одновременно работать множество пользователей. Подобно тому, как команда pwd сообщает имя текущего каталога, можно узнать имя «текущего» пользователя (пример 1.27), это делается посредством команды whoami (кто я?).

Команды whoi whoami – информация о пользователях системы

Пример 1.27. Запрос имени текущего пользователя

```
v100-1@beta:~ > whoami
v100-1
v100-1@beta:~ >
```

Выяснить имена пользователей, работающих в данный момент в системе можно при помощи команды who (кто?) (пример 1.28). В рамках практикума интерес представляет первый столбец выдачи, отображающий имена пользователей.

Пример 1.28. Запрос списка пользователей, работающих в системе

```
v100-1@beta:~> who
v100-2    pts/0        2010-03-01 10:33 (62.117.113.26)
v100-1    pts/1        2010-03-01 10:33 (62.117.113.26)
v100-4    pts/2        2010-03-01 10:33 (62.117.113.26)
```

```
v100-3    pts/3       2010-03-01 10:33 (62.117.113.26)
v100-1@beta:~>
```

1.3.2. Задачи и процессы в системе

Поскольку UNIX – многозадачная система, каждый пользователь может одновременно запустить несколько задач. Всякая запущенная пользователем или системой задача является процессом. Просмотр процессов, запущенных пользователем, выполняется командами ps и jobs. Команда ps – внешняя по отношению к командному интерпретатору, поэтому (пример 1.29) сообщает о двух процессах: о собственном процессе и об интерпретаторе bash. Команда jobs является внутренней командой интерпретатора bash, и если в нем не запущено никаких внутренних процессов, список выдачи jobs будет пустым.

Команды ps и jobs – информация о процессах в системе

Пример 1.29. Выдача сообщений о процессах, запущенных в системе

```
v100-1@beta:~> ps
PID TTY          TIME CMD (ID процесса, Терминал, Время, Команда)
30749 pts/1    00:00:00 bash
31074 pts/1    00:00:00 ps
v100-1@beta:~> jobs
v100-1@beta:~>
```

Для порождения процессов можно использовать уже известную команду cat без параметров (пример 1.30). В таком режиме cat порождает бесконечный процесс, создающий эхо-эффект – введенный с клавиатуры текст после нажатия «Enter» выводится на экран. Нажатие «Ctrl+Z» приостанавливает и подвешивает процесс, который при этом ни куда не исчезает, а переходит в фоновый режим (background). Статус процесса отображается командой jobs.

В фоновом режиме одновременно может быть запущено несколько процессов (к вопросу о «кнопке с крестиком», см. рис. 1.6).

Пример 1.30. Перевод процесса в фоновый режим

```
v100-1@beta:~> cat
Папа у Васи силен в математике <Enter>
Папа у Васи силен в математике<Ctrl+Z>
[1]+  Stopped                 cat
```

```
v100-1@beta:~> cat
Учится папа за Васю весь год<Enter>
Учится папа за Васю весь год<Ctrl+Z>
[2]+  Stopped                 cat
v100-1@beta:~> jobs
[1] – Stopped                 cat
[2]+  Stopped                 cat
v100-1@beta:~>
```

Из примера 1.30 видно, что один пользователь может (сознательно или случайно) запустить несколько процессов. В связи с этим естественно возникает вопрос об их завершении. Наиболее безопасный способ заключается в том, чтобы вывести процесс из background на «передний план» (англ. foreground) командой fg, а затем корректно завершить его «Ctrl+D» (пример 1.31).

Пример 1.31. Вывод процесса на передний план и его завершение

```
v100-1@beta:~> jobs
[1] – Stopped                 cat
[2]+  Stopped                 cat
v100-1@beta:~> fg
cat
<Ctrl+D>
v100-1@beta:~> fg
cat
<Ctrl+D>
v100-1@beta:~> jobs
v100-1@beta:~>
```

1.3.3. Права доступа к файлам и каталогам

Для UNIX, как многопользовательской системы, одной из актуальных проблема является регламентирование доступа к некоторому файлу со стороны пользователей. В отношении доступа к файлу все пользователи делятся на три категории:

u – user (пользователь, собственник файла);

g – group (группа);

o – other (все остальные).

Следует отметить, что термин и понятие group (группа) в UNIX-системах никак не связаны с организационно-административным делением людей (например, делением курса университета на студенческие группы). Как правило, в одну группу (group) объединяют

пользователей, которым для работы необходим доступ к одним и тем же информационным и вычислительным ресурсам системы. Соответственно, один и тот же пользователь может быть членом нескольких системных групп одновременно.

Под правами доступа понимаются права (или ограничения) в отношении действий и операций с файлом. В UNIX такие действия подразделяются на три группы:

r – read (право на чтение);

w – write (право на запись);

x – execute (право на использование, например на запуск программы или скрипта).

Право на чтение файла (атрибут r) означает, что пользователь может просматривать содержимое файла с помощью различных команд, например cat, more, или с помощью текстового редактора. Однако, редактируя содержимое файла в текстовом редакторе, пользователь не может сохранить изменения на диск при отсутствии права на запись файла (атрибут w). Право на выполнение (атрибут x) означает возможность запускать файл как исполняемую программу или скрипт, но не подразумевает возможности редактирования и даже просмотра.

По отношению к каталогам понятия «право на чтение», «право на запись» и «право на выполнение» имеют несколько другой смысл. Право на чтение каталога дает возможность просматривать его содержимое, т.е. список файлов и подкаталогов. Право на запись позволяет создавать и удалять файлы и подкаталоги. Право на выполнение дает возможность делать каталог текущим и обращаться к файлам и подкаталогам.

Команда ls – просмотр прав доступа к файлам и каталогам

Пример 1.32. Права доступа к файлам и подкаталогам текущей директории

```
v100-1@beta:~> ls –l
drwx--S--- 4 v100-1 v100  96 Окт 26 19:09 hacker
v100-1@beta:~>
```

Для просмотра прав доступа к файлам и подкаталогам текущей директории используется команда ls –l (пример 1.32 – в данном примере предполагается, что текущей директорией является домашний каталог).

Пример 1.33. Права доступа к файлам и подкаталогам по указанному пути

```
v100-1@beta:~> ls –l hacker/crack/question/thieve
-rw-r--r-- 1 v100-1 v100 0 Окт 30 11:05 hacker1.txt
-rw-r--r-- 1 v100-1 v100 0 Окт 30 11:05 hacker2.txt
-rw-r--r-- 1 v100-1 v100 0 Окт 30 11:05 hacker3.txt
-rw-r--r-- 1 v100-1 v100 0 Окт 30 11:05 hacker4.txt
-rw-r--r-- 1 v100-1 v100 0 Окт 30 10:35 hck2.txt
v100-1@beta:~>
```

Для просмотра прав доступа к файлам и подкаталогам произвольной директории команде ls –l в качестве параметра необходимо указать соответствующий путь (пример 1.33).

Выдача команды ls -l (см. примеры 1.32, 1.33) представляет собой последовательность записей с атрибутами файлов и подкаталогов указанной (или текущей директории), их имена указаны в конце каждой записи. Запись, относящаяся к подкаталогу, начинается с литеры «d», а к файлу – с символа «-» (тире). Далее следует описание прав доступа к файлу (подкаталогу) для всех категорий пользователей, в очередности: u (user), g(group) и o(other). Полный набор прав доступа: r (чтение), w (запись) и x (использование), обозначается тройкой литер «rwx». Отсутствие того или иного права доступа обозначается прочерком, например «rw-» – отсутствует право x (использование) или «r--» – есть только право чтения. В записи также указано, кто является пользователем (категория u) группой (категория g) для данного файла – в приведенных примерах это v100-1 и v100 соответственно.

Для изменения прав доступа предназначена команда chmod. В качестве параметров этой команде указывается категория пользователя (u, g или o) плюс (или минус) право (r, w или x). Плюс означает добавление или установка права, а минус – отмену или запрет права. Далее будут рассмотрены примеры изменения прав для категории u (пользователь), поскольку в рамках данного практикума для остальных категорий пользователей, в том числе g (группа), доступ в личный каталог закрыт.

Команда chmod – изменение прав доступа к файлам

Пример 1.34. Право на чтение каталога

```
v100-1@beta:~> ls -l
```

```
drwxr-xr-x 4 v100-1 v100 96 Окт 30 07:43 hacker
v100-1@beta:~> chmod u-r hacker
v100-1@beta:~> ls -l
d-wxr-xr-x 4 v100-1 v100 96 Окт 30 07:43 hacker
v100-1@beta:~> cd hacker
v100-1@beta: ~/hacker> ls
ls: невозможно открыть каталог .: Отказано в доступе
v100-1@beta: ~/hacker>
```

Влияние изменения прав доступа на работу с каталогами можно изучить на примере каталога hacker, на который у пользователя изначально установлены все права: rwx. Если закрыть r-право для каталога crack, то его просмотр (например, командой ls) невозможен, хотя вполне получится сделать его текущим из-за наличия x-права (пример 1.34).

Пример 1.35. Право на использование каталога

```
v100-1@beta: ~/hacker> cd ~
v100-1@beta:~> chmod u+r-x hacker
v100-1@beta:~> ls -l
drw-r-xr-x 4 v100-1 v100 96 Окт 30 07:43 hacker
v100-1@beta:~> cd hacker
-bash: cd: hacker: Отказано в доступе
v100-1@beta:~>
```

Запрет x-права делает невозможным выбор каталога в качестве текущего (пример 1.35), а запрет w-права не позволяет создавать внутренние подкаталоги и файлы (пример 1.36).

Пример 1.36. Право на запись в каталог

```
v100-1@beta:~> chmod u+x-w hacker
v100-1@beta:~> ls -l
dr-x-r-xr-x 4 v100-1 v100 96 Окт 30 07:43 hacker
v100-1@beta:~> cd hacker
v100-1@beta:~/hacker>mkdir new
mkdir: невозможно создать каталог `new': Отказано в доступе
v100-1@beta:~/hacker>cat > new.txt
-bash: new.txt: Отказано в доступе
v100-1@beta:~/hacker>
```

Пример 1.37. Право чтения файла

```
v100-1@beta:~/hacker> cd  crack/question/thieve
v100-1@beta:~ /hacker> cd  crack/question/thieve > ls –l
-rw-r--r-- 1 v100-1 v100 0 Окт 30 11:05 hacker1.txt
```

```
-rw-r--r—1 …….
-rw-r--r-- 1 v100-1 v100 0 Окт 30 10:35 hck2.txt
v100-1@beta:~ /hacker> cd  crack/question/thieve >chmod u-r *
--w-r--r-- 1 v100-1 v100 0 Окт 30 11:05 hacker1.txt
--w-r--r-- 1……..….
--w-r--r-- 1 v100-1 v100 0 Окт 30 10:35 hck2.txt
v100-1@beta:~ /hacker> cd  crack/question/thieve >cat hck2.txt
cat: hck2.txt: Отказано в доступе
v100-1@beta:~ /hacker> cd  crack/question/thieve >
```

Опыты по изменению прав доступа можно проделать также с файлами, например из каталога ~/hacker/crack/question/thieve. Право на выполнение (атрибут x) актуально только для исполняемых программ, поэтому для текстовых файлов смысла не имеет. Отмена права на чтение (атрибут r) закрывает возможность работы с файлом не только для текстовых редакторов, но и для команд просмотра: cat и more и других утилит и приложений (пример 1.37).

Право на запись (атрибут w) связано с внесением изменений в текст или иное содержимое файла. Внесение таких изменений без применения текстового редактора возможно с использованием не деструктивного перенаправления стандартного вывода для команды cat (подробнее об этом в п. 1.3.4) – это дает возможность добавлять текст в конец файла (пример 1.38).

Пример 1.38. Добавление текста в конец файла

```
v100-1@beta:~ /hacker/crack/question/thieve > chmod u+r *
v100-1@beta:~ /hacker/crack/question/thieve > ls -l
-rw-r--r-- 1 v100-1 v100 0 Окт 30 11:05 hacker1.txt
-rw-r--r—1 …….
-rw-r--r-- 1 v100-1 v100 0 Окт 30 10:35 hck2.txt
v100-1@beta:~ /hacker/crack/question/thieve > cat >> hacker1.txt
Новая строчка<Enter> <Ctrl + D>
v100-1@beta:~ /hacker/crack/question/thieve > cat  hacker1.txt
Если в сервер вашей фирмы
Вдруг забрался злобный хакер
Еще одна строчка
v100-1@beta:~ /hacker/crack/question/thieve >
```

Запрет права записи (w-права) закрывает как возможность добавления текста в файл командой cat (пример 1.39), так вообще возможность внесения в файл любых изменений для всех без исключения утилит и приложений.

Пример 1.39. Право записи файла

```
v100-1@beta:~ /hacker/crack/question/thieve > chmod u+r *
v100-1@beta:~ /hacker/crack/question/thieve > ls -l
-r--r--r-- 1 v100-1 v100 0 Окт 30 11:05 hacker1.txt
-r--r--r—1 …….
-r--r--r-- 1 v100-1 v100 0 Окт 30 10:35 hck2.txt
v100-1@beta:~ /hacker/crack/question/thieve > cat >> hacker1.txt
-bash: hacker1.txt: Отказано в доступе
v100-1@beta:~ /hacker/crack/question/thieve >
```

1.3.4. Ввод/вывод и конвейеры

Неотъемлемой частью любой компьютерной программы, а тем более операционной системы является система ввода/вывода команд и данных. В UNIX стандартные процесс (команды и утилиты) работают со стандартными каналами обмена информацией:

- *стандартный ввод* – stdin (standard input);
- *стандартный вывод* – stdout (standard output);
- *стандартный вывод ошибок* – stderr (standard error);

В рамках данного практикума будут рассмотрены только два из них: stdin и stdout. Первый связан с клавиатурой, а второй – с экраном монитора. Иллюстрацией может служить практически любая стандартная команда UNIX – команда вводится с клавиатуры, а результат ее работы выводится на экран.

Возникает вопрос: откуда, кроме клавиатуры, можно что-то вводить и куда, кроме экрана, что-то выводить? Ответ на оба эти вопроса один: эти «откуда» и «куда» есть не что иное, как файлы. Как правило, это текстовые файлы. Каналы stdin и stdout, по сути, являются текстовыми файлами для ввода/вывода по умолчанию. И это не удивительно, поскольку в UNIX по большому счету нет ничего, кроме учетных записей, процессов и файлов.

То, что вводится с клавиатуры, попадает в специализированный файл stdin, а на экран выводится содержимое файла stdout. Такая организация ввода/вывода позволяет легко заменять одни файлы другими. Для этого используются символы перенаправления:

- > (знак «больше» – перенаправление из stdout в файл);
- < (знак «меньше» – перенаправление в stdin из файла);
- >> (двойное «больше» – недеструктивное перенаправление stdout в файл).

Недеструктивное перенаправление вывода позволяет добавлять информацию в конец файла (см. пример 1.38), тогда как обычное (по сути, деструктивное) перенаправление уничтожает информацию и создает новый пустой файл. Таким образом, использование недеструктивного вывода позволяет, например, объединять текст нескольких файлов в одном без использования текстового редактора.

Раньше уже демонстрировалось перенаправление стандартного вывода stdout в файл при создании текстовых файлов командой cat. Для перенаправления потока данных из файла в stdin можно также использовать команду cat (продолжается работа с текущей директорией: ~ /hacker/crack/question/thieve). Поток stdin команды cat связан с клавиатурой, однако через перенаправление его можно связать с файлом (пример 1.40). Поток stdout команды cat связан с экраном монитора, поэтому текст из файла выводится на экран.

Пример 1.40. Ассоциация файла с потоком stdin команды cat

```
v100-1@beta: ~ /hacker/crack/question/thieve > cat < hacker1.txt
Если в сервер вашей фирмы
Вдруг забрался злобный хакер
v100-1@beta: ~ /hacker/crack/question/thieve >
```

Связь команд UNIX через потоки данных стандартного ввода/вывода позволяет осуществлять состыковку команд через файлы stdin и stdout.

В качестве учебной задачи предлагается выполнить сортировку файлов ~ /hacker/crack/question/thieve в обратном алфавитном порядке (от z до a). Прежде всего, необходимо проверить наличие файлов в каталоге, а затем получить список файлов, перенаправив stdout команды ls в новый текстовый файл, пусть это будет текстовый файл с именем file-list (пример 1.41). Просмотр текстового файла, как обычно, осуществляется командой cat.

Пример 1.41. Перенаправление stdout команды ls в файл

```
v100-1@beta: ~ /hacker/crack/question/thieve > ls
hacker1.txt  hacker2.txt  hacker3.txt  hacker4.txt  hck2.txt
v100-1@beta: ~ /hacker/crack/question/thieve > ls > file-list
v100-1@beta: ~ /hacker/crack/question/thieve > cat  file-list
file-list
hacker1.txt
hacker2.txt
hacker3.txt
```

```
hacker4.txt
hck2.txt
v100-1@beta: ~ /hacker/crack/question/thieve >
```

Для сортировки текстового файла в алфавитном порядке используется команда sort, а для сортировки в обратном алфавитном порядке команда sort –r (пример 1.42).

Пример 1.42. Создание файла непосредственно в директории

```
v100-1@beta: ~ /hacker/crack/question/thieve > sort –r  file-list
hck2.txt
hacker4.txt
hacker3.txt
hacker2.txt
hacker1.txt
file-list
v100-1@beta: ~ /hacker/crack/question/thieve >
```

Построив состыкованную команду (pipelines) из ls и sort, можно обойтись без промежуточного файла, каковым, по сути, является file-list. Для записи стыковки команд используется символ «|» (вертикальная черта). Состыкованная команда (или *конвейер*) имеет более короткую и удобную запись, по сравнению с последовательно выполняемыми командами (пример 1.43).

Пример 1.43. Конвейер команд ls и sort

```
v100-1@beta: ~ /hacker/crack/question/thieve >rm file-list
v100-1@beta: ~ /hacker/crack/question/thieve >ls | sort -r
hck2.txt
hacker4.txt
hacker3.txt
hacker2.txt
hacker1.txt
file-list
v100-1@beta: ~ /hacker/crack/question/thieve >
```

Пример 1.44. Запись результата работы конвейера в файл

```
v100-1@beta: ~ /hacker/crack/question/thieve >ls | sort –r > sort-list
v100-1@beta: ~ /hacker/crack/question/thieve >cat sort-list
hck2.txt
hacker4.txt
hacker3.txt
hacker2.txt
hacker1.txt
```

```
file-list
v100-1@beta: ~ /hacker/crack/question/thieve >
```

При необходимости можно комбинировать конвейеры с перенаправлением ввода/вывода. Пример 1.44 показывает, что можно записать отсортированный конвейером список в какой-нибудь текстовый файл.

1.3.5. Поиск и обработка текстовых данных

Работая в UNIX, иногда бывает необходимость найти группу файлов или папок, удовлетворяющих определенным критериям, и выполнить с ними некоторые действия. Для этих целей в UNIX есть несколько команд и утилит, в рамках выполняемого практикума предлагается рассмотреть только одну из них – команду, а точнее утилиту grep.

Утилита grep – поиск текстовых файлов по фрагментам текста

Утилита grep выполняет поиск в текстовых файлах образца текста, и в зависимости от способа применения выдает информацию о файле, содержащем этот образец. Например, можно узнать, есть ли в директории /usr/share/liblab/proverbs/spirit/mind файлы, содержащие текстовые фрагменты «show» и «shall». Выясняется, что есть файл /usr/share/liblab/proverbs/spirit/mind/philosph, содержащий текстовый образец «shall», а файлов, содержащих «show», в этой директории не существует (пример 1.45). Команда grep выдает на экран полное имя файла и строки из него, в которых найдет искомый образец текста.

Пример 1.45. Поиск файла, содержащего образец текста

```
v100-1@beta:~> grep shall /usr/share/liblab/proverbs/spirit/mind/*
/usr/share/liblab/proverbs/spirit/mind/philosph:   As you sow, you shall mow.
v100-1@beta:~> grep show /usr/share/liblab/proverbs/spirit/mind/*
v100-1@beta:~>
```

Для дальнейших манипуляций над найденным файлом потребуется форма команды grep, которая будет выдавать в качестве результата строку, содержащую только имя файла, в виде полного пути. Это команда grep –l (пример 1.46).

Пример 1.46. Печать имени файла, содержащего образец текста

```
v100-1@beta:~> grep –l shall /usr/share/liblab/proverbs/spirit/mind/*
/usr/share/liblab/proverbs/spirit/mind/philosph
v100-1@beta:~> grep show /usr/share/liblab/proverbs/spirit/mind/*
v100-1@beta:~>
```

Очевидным способом просмотра текста найденного файла является: cat /usr/share/liblab/proverbs/spirit/mind/philosph, однако чтобы не набирать по многу раз команды с длинными путями, можно использовать результат работы grep –l в качестве параметра cat (пример 1.47). При этом вся команда grep –l заключается в обратные апострофы (другой регистр клавиши с волной «~»).

Пример 1.47. Просмотр файла, содержащего образец текста

```
v100-1@beta:~> cat `grep -l shall /usr/share/liblab/proverbs/spirit/mind/*`
  No fool as old fool.
  Нет дурака хуже старого (дурака).

  Nothing must be done hastily but killing of fleas.
  Ничего не следует делать наспех, кроме избиения блох.
  ср. Спешка нужна только при ловле блох.

  Where there's a will, there's a way.
  Где есть желание, есть и средство.

  Every dark cloud must have its silver lining.
  У каждого темного облака должна быть своя серебристая подкладка.
  ср. Нет худа без добра.
  As you sow, you shall mow.
  Как посеешь, так и скосишь.
  По-русски, что посеешь, то и пожнешь.
v100-1@beta:~>
```

По сути, результатом grep –l является выдаваемая в stdout текстовая строка, которая может быть преобразована, например, потоковым текстовым редактором sed (англ. Stream EDitor). Например, можно выполнить замену текста, используя внутреннюю команду s/что_заменять/на_что_заменять/. Пример 1.48 показывает, как, используя конвейер, изменить текстовую строку из stdout, добавив в конец строки (обозначается $) группу символов « . » (пробел, точка, пробел).

Пример 1.48. Добавление символов к строке, полученной из stdout

```
v100-1@beta:~> grep -l shall /usr/share/liblab/proverbs/spirit/mind/* | sed s/$/" . "/
/usr/share/liblab/proverbs/spirit/mind/philosph .
v100-1@beta:~>
```

Полученная таким образом (см. пример 1.48) текстовая строка может быть использована в качестве аргумента (обрамленного обратными апострофами) для команды копирования cp (пример 1.49). Для копирования файла нужно предварительно создать в домашней директории каталог myproverbs и сделать его текущим.

Пример 1.49. Копирование файла, найденного по образцу текста

```
v100-1@beta:~>mkdir myproverbs
v100-1@beta:~>cd myproverbs
v100-1@beta:~/ myproverbs > cp `grep -l shall /usr/share/liblab/proverbs/spirit/mind/* |
sed s/$/" . "/`
v100-1@beta:~ / myproverbs >  ls
philosph
v100-1@beta:~ / myproverbs >  ls
```

Применение команды grep –l shall к текущей директории выдаст в качестве результата текстовую строку philosph. Эту строку можно передать в качестве параметра команде chmod, устанавливающей права доступа к файлам (пример 1.50). Изменение прав доступа (перед и после) контролируется командой ls –l.

Пример 1.50. Изменение прав доступа к файлу, найденному по образцу текста

```
v100-1@beta:~/ myproverbs > ls –l
-rw-r--r-- 1 v100-1 v100 792 Ноя  3 11:45 philosph
v100-1@beta:~/ myproverbs > chmod u-w  `grep -l shall *`
-r--r--r-- 1 v100-1 v100 792 Ноя  3 11:45 philosph
v100-1@beta:~/ myproverbs >
```

1.4. ФАЙЛОВЫЙ МЕНЕДЖЕР MIDNIGHT COMMANDER

1.4.1. Внешний вид, начало и завершение работы

Файловый менеджер (англ. file manager) – компьютерная программа, предоставляющая интерфейс пользователя для работы с файловой системой и файлами. Файловый менеджер позволяет вы-

полнять наиболее частые операции над файлами: создание, удаление, исполнение, просмотр, редактирование, копирование, перемещение (переименование), а также изменение атрибутов и свойств, поиск файлов и управление правами доступа. Помимо основных функций многие файловые менеджеры включают ряд таких дополнительных возможностей, как работа с сетью и внешними устройствами.

Одним из самых распространенных файловых менеджеров для UNIX-систем является Midnight Commander (рис. 1.9). Midnight Commander – двухпанельный файловый менеджер с текстовым интерфейсом. Каждая панель отображает содержимое одной директории (т.е. фактически панель – это каталог). Midnight Commander имеет удобный встроенный текстовый редактор, который позволяет создавать и редактировать текстовые файлы.

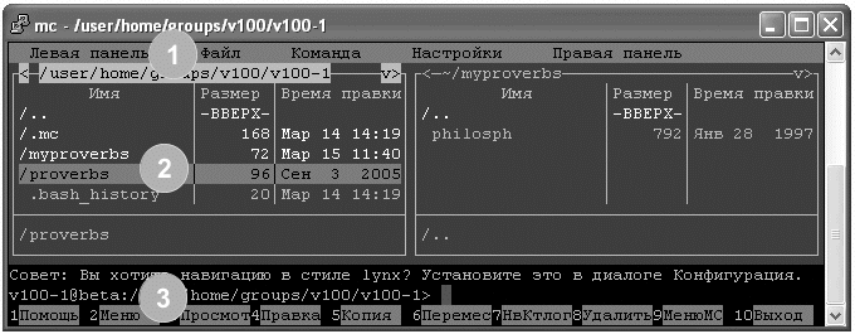


Рис. 1.9. Файловый менеджер Midnight Commander

Верхняя строка (1 рис. 1.9) представляет собой меню, активизируемое мышью или клавишей «F9». Далее вниз слева и справа расположены панели со списками файлов и подкаталогов директорий, имена которых указаны на верхних рамках панелей. Заголовок активной панели (директория /user/home/groups/v100/v100-1) подсвечивается. В активной панели также находится светлый прямоугольный курсор (2 рис. 1.9), который можно перемещать по списку стрелками вверх и вниз, а также мышью. Смена панелей (назначение панели активной) осуществляется клавишей «Tab».

Самая нижняя строка (3 рис. 1.9) представляет собой ряд экранных кнопок, каждая из которых ассоциирована с одной из экранных кнопок с «F1» по «F9». Можно пользоваться этой строкой как подсказкой по использованию функциональных клавиш, а можно непосредственно запускать команды Midnight Commander, щелкая мышью по экранным кнопкам.

Ниже двух панелей Midnight Commander, над экранными кнопками, находится командная строка, в которой могут выполняться стандартные команды UNIX. При помощи клавиш «Ctrl+O» (не нуль!) можно выключать (а затем также включать) обе панели, оставляя только командную строку и при этом не выходя из Midnight Commander.

Для запуска Midnight Commander нужно набрать mc в командной строке UNIX (рис. 1.10).

Рис. 1.10. Запуск файлового менеджера Midnight Commander

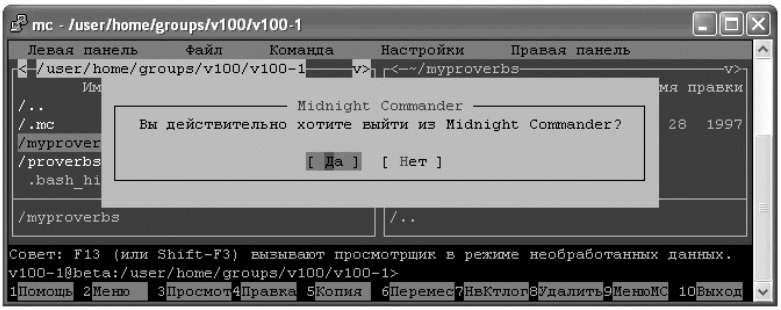


Рис. 1.11. Выход из Midnight Commander

Для выхода из Midnight Commander можно воспользоваться клавишей «F10», после чего в появившемся окне подтверждения нужно выбрать «Да» или «Нет», переместив подсветку на нужный вариант стрелками или «Tab» (рис. 1.11).

Выбор подтверждается нажатием «Enter» или мышью. Отказаться от выхода можно дважды нажав «Esc». После выхода из Midnight Commander, для завершения сеанса UNIX, необходимо воспользоваться командой logout.

1.4.2. Работа с каталогами и файлами

Файловый менеджер Midnight Commander позволяет создавать и удалять директории и файлы в любой из двух панелей, а также осуществлять копирование и перемещение файлов и каталогов из одной панели в другую.

Новый каталог создается в активной панели (в которой находится курсор-подсветка) Midnight Commander. Для создания каталога нужно нажать клавишу «F7», после чего ввести его имя в появившееся окно (рис. 1.12).

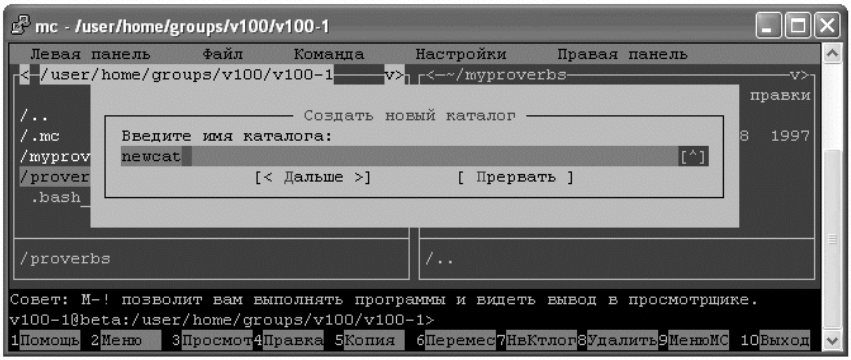


Рис. 1.12. Создание нового каталога

Ввод имени подтверждается нажатием клавиши «Enter» или щелчком мыши по экранной кнопке «Дальше». Отказаться от создания каталога можно дважды нажав «Esc» или через выбор экранной кнопки «Прервать». Перемещение подсветки между активными элементами окна создания каталога осуществляется стрелками или клавишей «Tab» (управляющие клавиши в Midnight Commander одинаковы для всех диалоговых окон).

Поместив курсор-подсветку на имя каталога в списке и нажав «Enter» (или двойной щелчок мыши), пользователь входит в ката-

лог. Выбрав элемент списка в виде двух последовательных точек в верхней части панели и подтвердив выбор, пользователь выходит из каталога.

Для удаления каталога его нужно выбрать в списке активной панели и нажать клавишу «F8» (рис. 1.13), а затем в появившемся красном диалоговом окне выбрать и подтвердить необходимое действие: «Да» – удалить, или «Нет» – отменить удаление. Удаление файлов осуществляется аналогичным способом.

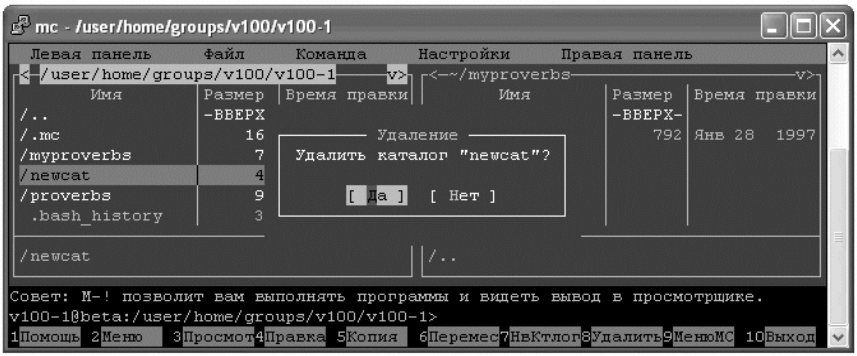


Рис. 1.13. Удаление каталога

Создание нового текстового файла производится в активной панели нажатием клавиш «Shift+F4». При этом запускается текстовый редактор mcedit. После набора текста, для его сохранения в файл, необходимо нажать «F10» или дважды «Esc», а затем в появившемся диалоговом окне (рис. 1.14) либо согласиться с изменением файла, либо отказаться от него.

Если файл – новый, то появляется диалог для ввода и подтверждения имени файла. На этом этапе также можно отказаться от создания файла (рис. 1.15).

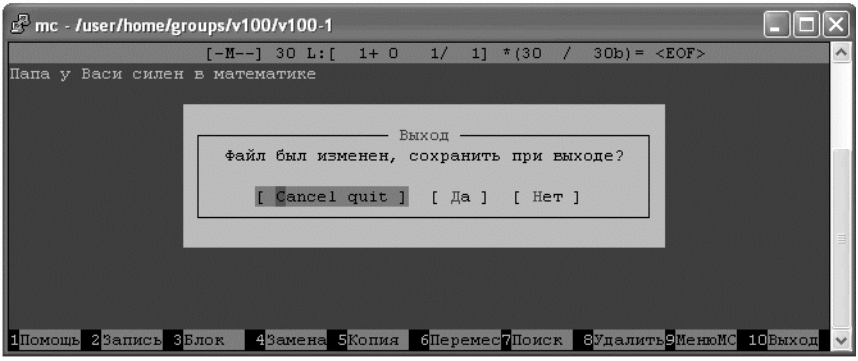


Рис. 1.14. Подтверждение сохранения изменений

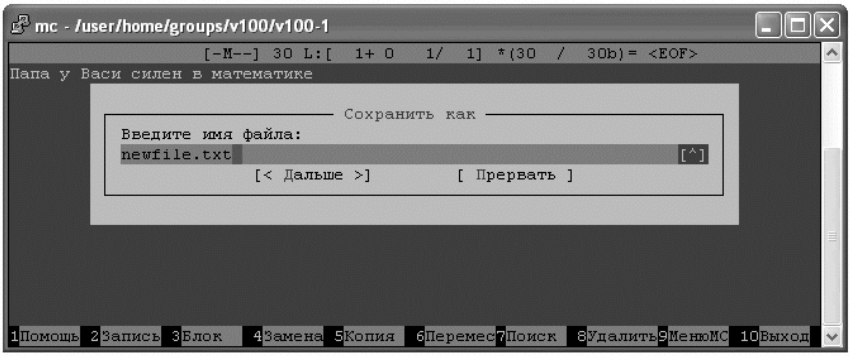


Рис. 1.15. Ввод и подтверждение имени файла

Для того, чтобы скопировать файл из одной директории в другую, нужно чтобы две директории – источник и приемник – отображались на панелях Midnight Commander. Причем нужно сделать так, чтобы директория-источник (из которой происходит копирование) находилась в активной панели. После этого курсором-подсветкой отмечается копируемый файл и нажимается клавиша «F5» – при этом последовательно открываются диалоговые окна подтверждения и параметров копирования (рис. 1.16).

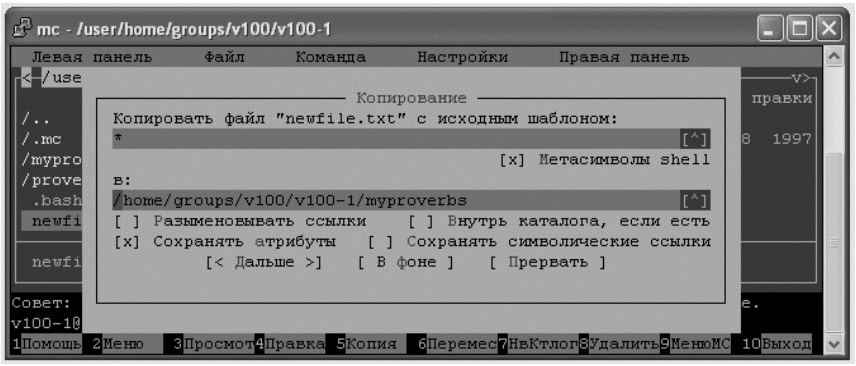


Рис. 1.16. Ввод и подтверждение имени файла

В поле «Копировать файл с исходным шаблоном» можно оставить «звездочку», если планируется копирование файла без изменения имени. Во втором поле диалогового окна (поле «в:») должна быть указана директория, в которую производится копирование – по умолчанию это директория второй (неактивной) панели.

Если файл копируется с изменением его имени, то оно должно быть дописано через слеш «/» к директории поля «в:» (а не в верхнем поле вместо звездочки, как это может показаться).

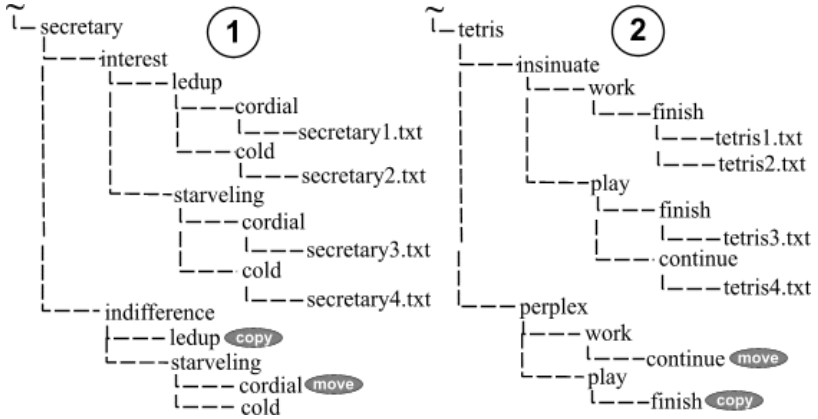
При необходимости перемещения файла из одной директории в другую оно производится совершенно аналогично копированию, только вместо клавиши «F5» используется «F6».

ЗАЧЕТНЫЕ ЗАДАНИЯ К РАЗД. 1

Задание 1.1

В своей рабочей директории построить дерево каталогов и файлов, используя различные варианты записи путей к каталогам и файлам (примеры 1.16 ÷ 1.21). На метки «copy» и «move» внимания не обращать – они нужны для второго задания.

Варианты задания



secretary1.txt
Если секретарша босса
Интерес к Вам потеряла,

secretary2.txt
Бутербродами не кормит
И на кофе не зовет –

tetris1.txt
Если ваше руководство
Вам прозрачно намекает,

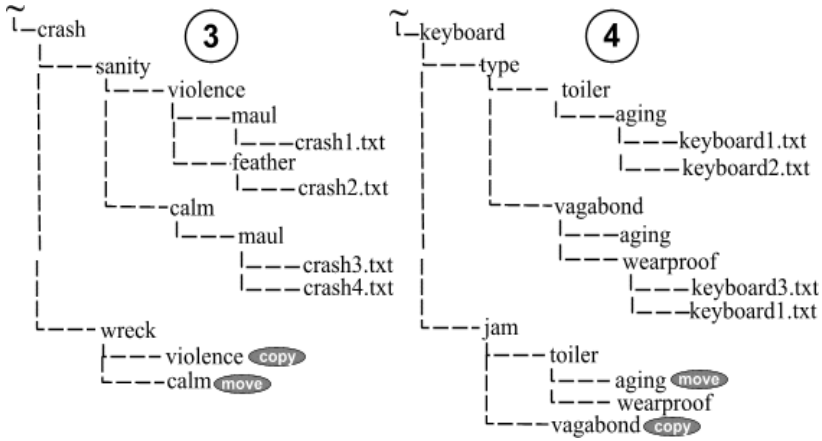
tetris 2.txt
Что заказчик будет завтра,
А заказ пока стоит –

secretary3.txt
На ее мышиный коврик
Вылейте немного клея.

secretary4.txt
И она Вам сразу скажет
Очень много теплых слов.

tetris3.txt
Оставайтесь на работе
И всю ночь играйте в "тетрис".

tetris4.txt
И, взглянув на Вас, заказчик
Крайний срок перенесет.



crash1.txt
Если вдруг твоя машина
Не работает, как надо,

crash 2.txt
Ты по материнской плате
Сильно стукни кулаком.

keyboard1.txt
Если на клавиатуре
Западает пара клавиш,

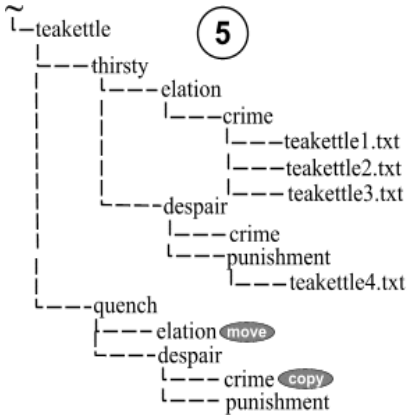
keyboard2.txt
Это значит, Вы – ударник
И вообще герой труда.

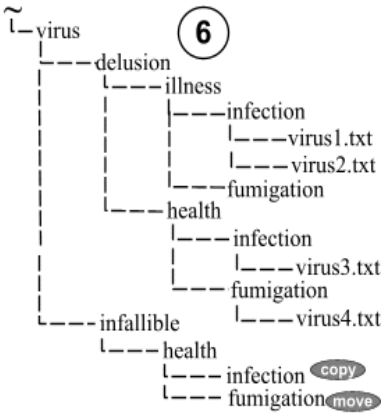
crash3.txt
Не поможет – бей кувалдой,
Дай ногой по монитору...

crash4.txt
Объяснишь потом начальству:
"Она первой начала!"

keyboard3.txt
Незаметно поменяйтесь
Ей с бездельником соседом:

keyboard4.txt
У таких клавиатуры
Не стареют никогда!





teakettle1.txt
Если Ваш любимый чайник
В тройнике лишен розетки,

teakettle2.txt
То за это преступленье
Надо строго наказать,

virus1.txt
Если ваш руководитель,
Заблуждаясь, утверждает,

virus2.txt
Что в его машине вирус
И он Вами занесен,

teakettle3.txt
И недрогнувшей рукою
Вынуть из розетки сервер,

teakettle4.txt
Чтобы все вокруг узнали,
Что пришла пора пить чай.

virus3.txt
То, когда он отлучится,
Выньте из нее винчестер,

virus4.txt
И в кастрюле кипятите,
Пока вирус не помрет.

Задание 1.2

Скопировать файлы дерева, созданного в задании 1.1, в директорию, отмеченную как «copy», затем переместить исходные файлы того же дерева в директорию отмеченную «move». Использовать различные способы записи путей (см. примеры 1.23 ÷ 1.26).

Варианты задания

Соответствуют вариантам задания 1.1.

Задание 1.3

Найти в каталоге /usr/share/liblab/proverbs русские аналоги английских поговорок (примеры 1.45 – 1.47) и скопировать файлы, найденные по образцам текста (пример 1.49) в каталог myproverbs, предварительно созданный в домашней директории. Для файлов с первым образцом текста закрыть право чтения для пользователя, а для файлов со вторым образцом текста закрыть право записи для пользователя. Для файлов с первым образцом текста закрыть право чтения для пользователя (пример 1.50).

Примечание 1. Для быстроты поиска и сокращения записи путей может быть использована маска – «*» (символ «звездочка»). Например, путь /usr/share/liblab/proverbs/spirit/mind/* может быть заменен на /usr/share/liblab/proverbs/*/*/*.

Примечание 2. Для поиска желательно выбрать уникальный образец текста, характерный именно для данной поговорки, не стоит использовать для поиска артикли и предлоги.

Варианты задания 1.3

Вариант 1

No pains, no gains.
A friend in need is a friend indeed.

Вариант 2

No pains, no gains.
Alive and kicking!

Вариант 3

No pains, no gains.
Kill or cure.

Вариант 4

No pains, no gains.
Crow will not pick out crow's eyes.

Вариант 5

No sweet without some sweat.
A friend in need is a friend indeed.

Вариант 6

No sweet without some sweat.
Alive and kicking!

Вариант 7

No sweet without some sweat.
Kill or cure.

Вариант 8

No sweet without some sweat.
Crow will not pick out crow's eyes.

Вариант 9

No living man all things can.
A friend in need is a friend indeed.

Вариант 10

No living man all things can.
Alive and kicking!

Вариант 11

No living man all things can.
Kill or cure

Вариант 12

No living man all things can.
Crow will not pick out crow's eyes.

Вариант 13

No pains, no gains.
A friend in need is a friend indeed.

Вариант 14

No pains, no gains.
Alive and kicking!

Вариант 15

No pains, no gains.
Kill or cure.

Вариант 16

No pains, no gains.
Crow will not pick out crow's eyes.

Вариант 17

No sweet without some sweat.
A friend in need is a friend indeed.

Вариант 18

No sweet without some sweat.
Alive and kicking!

Вариант 19

No sweet without some sweat.
Kill or cure.

Вариант 20

No sweet without some sweat.
Crow will not pick out crow's eyes.

Вариант 21

No living man all things can.
A friend in need is a friend indeed.

Вариант 22

No living man all things can.
Alive and kicking!

Вариант 23

No living man all things can.
Kill or cure.

Вариант 24

No living man all things can.
Crow will not pick out crow's eyes.

Вариант 25

No pains, no gains.
A friend in need is a friend indeed.

Вариант 26

No living man all things can.
Kill or cure

Вариант 27

No living man all things can.
Crow will not pick out crow's eyes.

Задание 1.4

С помощью файлового менеджера Midnight Commander в своей домашней директории построить дерево каталогов и файлов.

Варианты задания

Вариант 1 – см. дерево 6-го варианта задания 1.1.
Вариант 2 – см. дерево 5-го варианта задания 1.1.
Вариант 3 – см. дерево 4-го варианта задания 1.1.
Вариант 4 – см. дерево 3-го варианта задания 1.1.
Вариант 5 – см. дерево 2-го варианта задания 1.1.
Вариант 6 – см. дерево 1-го варианта задания 1.1.

Задание 1.5

С помощью файлового менеджера Midnight Commander скопировать файлы дерева, созданного в задании 1.4, в директорию, отмеченную как «copy», затем переместить исходные файлы того же дерева в директорию отмеченную «move».

Варианты задания

Соответствуют вариантам задания 1.4.

2. КОМАНДНЫЕ ОБОЛОЧКИ WINDOWS

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К РАЗД. 2

Настоящий лабораторный практикум посвящен знакомству со стандартной командной оболочкой *Cmd.exe*, реализующей режим командной строки для Windows-систем. В практикуме изучаются основные возможности командного интерпретатора Cmd.exe, связанные с файловой системой, стандартными потоками ввода/вывода и базовыми элементами командного языка интерпретатора Cmd.exe поддерживаемые практически во всех операционных системах линейки Windows XP/Vista/7.

Работая в личном каталоге Windows, студент должен научиться (см. примеры далее по тексту и зачетные задания к разд. 2):

1) самостоятельно получать справочную информацию о командах Cmd.exe и осуществлять простую навигацию по файловой системе Windows (примеры 2.1 – 2.4);

2) осуществлять просмотр, создание, редактирование, копирование и перемещение объектов файловой системы с использованием абсолютных и относительных путей (примеры 2.5 – 2.17, задание 2.1);

3) настраивать параметры стандартного Windows-приложения Cmd.exe для просмотра файлов в командной строке с учетом выбора правильной кодовой страницы и true type-шрифтов (пп. 2.2.1 – 2.2.2, задание 2.2);

4) использовать простейшие элементы командного языка интерпретатора Cmd.exe в сочетании с перенаправлением стандартного ввода/вывода и конвейерами команд для автоматизации работы с текстовыми файлами (поиск и сортировка, фильтрация – п. 2.2.3);

5) осуществлять стандартные операции с объектами файловой системы с помощью файлового менеджера FAR, а также автоматизировать с его помощью запуск приложений Windows (п. 2.3, задания 2.4 – 2.5).

2.1. ОСНОВЫ КОМАНДНОЙ СТРОКИ MS WINDOWS

2.1.1. Начало работы с командной строкой

Командная строка поддерживается во всех версиях Windows. Поддержка командной строки Microsoft Windows доступна через окно командного интерпретатора.

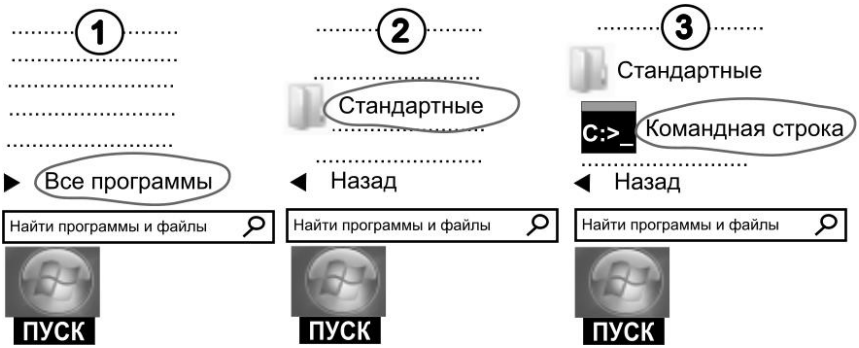


Рис. 2.1. Запуск командной строки через кнопку «Пуск»

Наиболее часто используют способ запуска командного интерпретатора Cmd.exe через меню кнопки «Пуск» графического интерфейса операционной системы MS Windows (рис. 2.1), последовательно выбирая пункты меню:

Пуск > Все программы > Стандартные > Командная строка

или в англоязычном интерфейсе:

Start > All Programs > Accessories > Command Prompt.

При этом в отдельном окне будет запущен командный интерпретатор Cmd.exe (рис. 2.2).

После запуска окна командной оболочки в поле курсора можно вводить команды с клавиатуры – при этом команды можно набирать как строчными (маленькими), так и прописными (большими) буквами – для командного интерпретатора Cmd.exe это безразлично. Завершение работы с командной строкой и закрытие окна командной оболочки осуществляется командой EXIT.

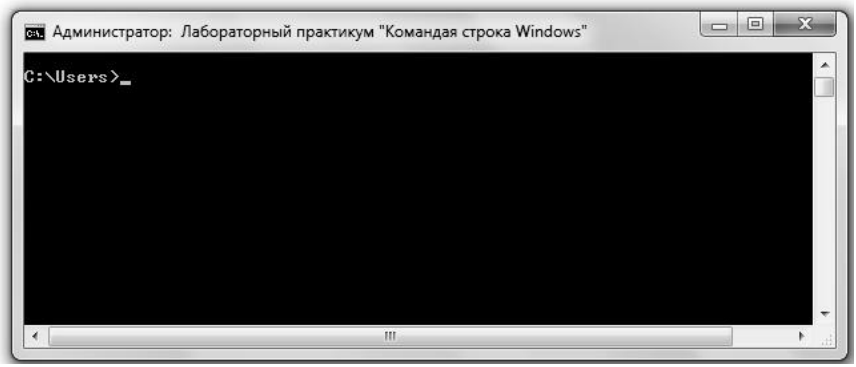


Рис. 2.2. Окно командного интерпретатора Cmd.exe

В процессе практического выполнения примеров разд. 1 необходимо ознакомиться с материалами разд. 2, прежде всего с пп. 2.2.1 и 2.2.2.

Перед входом в систему необходимо уточнить у преподавателя или системного администратора свой логин, пароль и рабочий каталог. Возможно, рабочий каталог придется самостоятельно создать в каталоге группы.

При запуске командной строки текущая директория может не совпадать с рабочей директорией. Для того чтобы сделать текущей свою рабочую директорию, необходимо действовать аналогично примеру (пример 2.1).

В примере предполагается, что при запуске командного интерпретатора текущей является директория C:\Users (в реальности, это может быть любая другая директория). Рабочей директорией (той, в которой должны выполняться задания лабораторного практикума), в примере является директория D:\WINLAB.

Пример 2.1. Переход в рабочую директорию C:\WINLAB

```
C:\Users> d:
D:\ > cd d:\winlab
d:\WINLAB >
```

Изменение текущего каталога (переход в нужный каталог) осуществляется командой CD (change directory), которой в качестве параметра указывается *путь* к нужному каталогу, в данном случае

это путь D:\WINLAB. Нужно учесть, что команда CD работает на текущем диске, поэтому первая попытка (пример 2.2) приведет к ошибке. Чтобы избежать такой ошибки, необходимо перейти с диска «C» на диск «D», указав в командной строке имя диска, на который осуществляется переход (см. пример 2.1), и двоеточие после имени диска.

Пример 2.2. Переход в рабочую директорию D:\WINLAB

```
C:\Users> cd d:\winlab
"d:\winlab" не является внутренней или внешней
командой, исполняемой программой или пакетным файлом.
C:\Users> cd /d d:\winlab
d:\WINLAB >
```

Однако, используя команду CD с ключом /D, можно осуществлять переходы между директориями любых дисков без смены диска, поэтому вторая попытка (см. пример 2.2) будет успешной – подробнее возможности команды CD рассмотрены в п. 2.1.3.

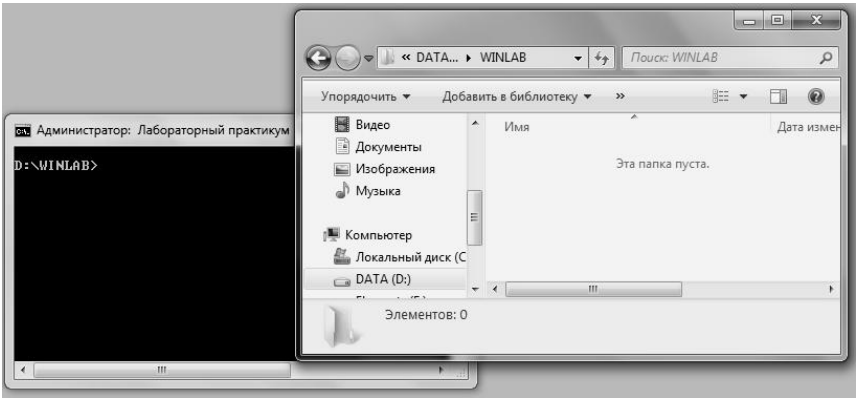


Рис. 2.3. Рабочая директория в окне командного интерпретатора и стандартном графическом интерфейсе Windows

При выполнении лабораторного практикума одновременно с переходом в свою рабочую директорию в окне командной оболочки целесообразно открыть эту же директорию средствами графического интерфейса, как папку Windows, – это позволит наблюдать результаты работы командного интерпретатора в привычном режиме графического интерфейса Windows (рис. 2.3).

2.1.2. Структура файловой системы Windows

Перечень наиболее распространенных операций с элементами файловой системы Windows – каталогами и файлами приведен в табл. 2.1 (квадратные скобки указывают на необязательность элемента), а полный перечень команд Cmd.exe можно получить командой HELP (пример 2.3).

Таблица 2.1. Базовые операции с каталогами и файлами в командной строке

Операция	Формат команды
Создание каталога	MKDIR [диск:]путь MD [диск:]путь
Удаление каталога	RMDIR [диск:]путь RD [диск:]путь
Просмотр каталога	DIR [диск:]путь
Просмотр дерева каталога	TREE [диск:]путь
Переименование каталога	MOVE [диск:][путь]путь1 новый_путь
Копирование каталога	XCOPY [диск:]путь1 путь2
Создание файла	COPY CON [диск:][путь]имя_файла
Удаление файла	DEL [диск:][путь]имя_файла ERASE [диск:][путь]имя_файла
Просмотр файла	TYPE [диск:][путь]имя_файла MORE [диск:][путь]имя_файла COPY [диск:][путь]имя_файла CON
Переименование файла	RENAME [диск:][путь]имя_файла1 имя_файла2 REN [диск:][путь]имя_файла1 имя_файла2
Перемещение файлов:	MOVE [диск:][путь]имя_файла1[,...] назначение
Копирование файла	COPY [диск:][путь]имя_файла1 имя_файла2.
Перемещение файла	RENAME [диск:][путь]имя_файла1 имя_файла2 REN [диск:][путь]имя_файла1 имя_файла2

Можно также получить подробную справку по отдельно взятой команде, снабдив ее ключом /? – например справку по команде MKDIR (пример 2.4).

Пример 2.3. Вывод перечня команд в командной строке

```
d:\WINLAB>help
Для получения сведений об определенной команде наберите HELP <имя команды>
ASSOC        Вывод либо изменение сопоставлений по расширениям имен файлов.
ATTRIB       Отображение и изменение атрибутов файлов.
……..……..
d:\WINLAB>
```

Пример 2.4. Получение справки по конкретной команде

```
d:\WINLAB>mkdir /?
Создание каталога.
MKDIR [диск:]путь
MD [диск:]путь
……..
d:\WINLAB>
```

Файловая система Windows представляет собой древовидную структуру (как в UNIX и большинстве других операционных систем). Соответственно, для такой файловой системы актуальны такие понятия, как *абсолютный путь* и *относительный путь* к каталогу или файлу (рис. 2.4).

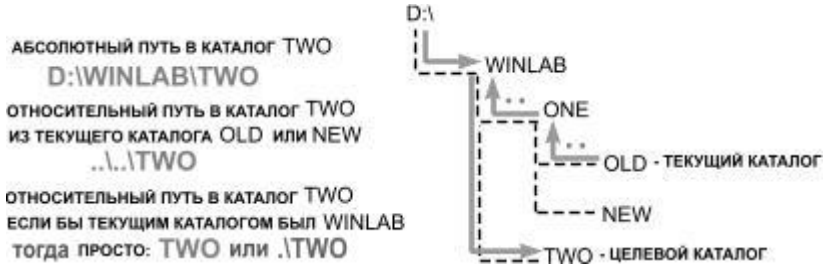


Рис. 2.4. Абсолютный и относительный путь в файловой системе Windows

Windows хранит данные на логических дисках: «C», «D», «E», «F» и т.д., физически расположенных на жестком диске («винчестере») или ассоциированных с такими внешними устройствами, как оптические CD и DVD приводы, USB Flash drive и др.

Абсолютный путь файловой системы Windows начинается с имени диска и двоеточия после него, а затем записывается последовательность имен вложенных каталогов. Для разделения элементов пути используется символ «\» (обратный слеш). В UNIX, если

вспомнить предыдущую лабораторную работу, для этой же цели используется «/» (прямой слеш).

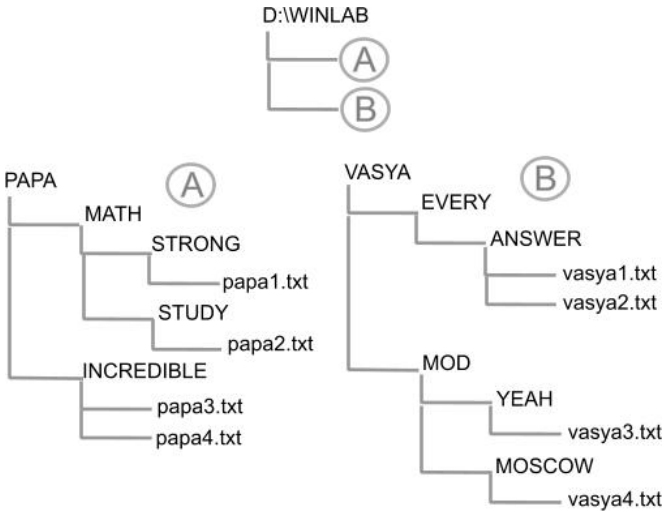


Рис. 2.5. Дерево, используемое в примерах лабораторного практикума

Таблица 2.2. Содержание файлов структуры (рис. 2.5)

Файл	Содержание
Структура A	
papa1.txt	Папа у Васи силен в математике,
papa2.txt	Учится папа за Васю весь год,
papa3.txt	Где это видано, где это слыхано:
papa4.txt	Папа решает, а Вася сдает?
Структура B	
vasya1.txt	На это каждый ответит, каждый ответит:
vasya2.txt	– Конечно, Вася, Вася, Вася,
vasya3.txt	Ну, кто его не знает? Yeah, yeah!
vasya4.txt	Вася, Вася, Вася – стиляга из Москвы.

Относительный путь строится относительно текущего каталога и также состоит из последовательности вложенных каталогов, разделенных обратным слешем. При записи относительного пути текущий каталог обозначается как «.» (точка), или совсем не указывается, а каталог вышестоящего уровня – двумя последовательными точками: «..», которые, как и все остальные элементы пути, разделяются обратными слешами.

В качестве упражнения по созданию структуры каталогов и файлов в файловой системе MS Windows с использованием команд интерпретатора Cmd.exe (см. табл. 2.1) предлагается построить дерево каталогов и файлов (см. рис. 2.5, табл. 2.2), а затем по аналогии выполнить задания лабораторного практикума (номер своего личного варианта задания необходимо получить у преподавателя). Для всех приведенных далее примеров использования команд интерпретатора Cmd.exe рабочим каталогом считается D:\WINLAB.

Для создания дерева каталогов (см. рис. 2.5) возможно использование двух команд: MD или MKDIR (make directory) в формате:

MKDIR [диск:]путь
MD [диск:]путь

При включении расширенной обработки команда MKDIR создает все промежуточные каталоги в пути (пример 2.5), тогда как при использовании команды MD пришлось бы создавать вложенные каталоги поочередно (пример 2.6).

Пример 2.5. Создание цепочки вложенных каталогов одной командой

```
d:\WINLAB>mkdir  papa\math\strong
d:\WINLAB>tree .
D:\WINLAB
└───papa
    └───math
        └───strong
d:\WINLAB>
```

Пример 2.6. Последовательное создание вложенных каталогов и попытка удаления непустого каталога

```
d:\WINLAB>md  papa
d:\WINLAB>md  papa\math
d:\WINLAB>md  papa\math\strong
d:\WINLAB>
```

Для просмотра существующей файловой структуры можно использовать команду TREE (см. пример 2.5) (tree – англ. – дерево), отображающую структуру каталогов в графическом виде. Если же нужно увидеть дерево каталогов вместе с содержащимися в них файлами, то необходимо использовать команду TREE с ключом /F (пример 2.7) – предположим, что дерево (см. рис. 2.5) построено. Напомним, что точкой в качестве параметра команды TREE обозначается текущий каталог (см. пример 2.5).

Пример 2.7. Просмотр дерева каталогов и файлов командой TREE

```
d:\WINLAB>tree papa /f
D:\WINLAB\PAPA
├───incredible
│       papa3.txt
│       papa4.txt
│
└───math
    ├───strong
    │       papa1.txt
    │
    └───study
            papa2.txt
d:\WINLAB>
```

В общем случае команда графического представления структуры папок или пути TREE имеет формат:

TREE [диск:][путь] [/F] [/A]

Ключ /F – Вывод имен файлов в каждой папке.

Ключ /A – Использовать символы ASCII вместо символов национальных алфавитов.

Весьма сильной стороной использования команды MKDIR является возможность одновременного создания нескольких цепочек вложенных каталогов (пример 2.8) – при этом такие цепочки должны отделяться друг от друга пробелами или запятыми.

Пример 2.8. Создание нескольких цепочек вложенных каталогов

```
d:\WINLAB>mkdir  papa\math\strong  papa\math\study
d:\WINLAB>tree .
D:\WINLAB
└───papa
    └───math
        ├───strong
        └───study
d:\WINLAB>
```

Довольно часто, для просмотра существующих каталогов и файлов используется команда DIR, отображающая список имен каталогов и файлов в указанной директории в различных форматах, определяемых ключами команды. Использование команды DIR без ключей приводит к появлению на экране избыточной информации. Чтобы получить только список имен каталогов и файлов, команду DIR необходимо запустить с ключом /B (см. пример 2.9).

Пример 2.9. Просмотр списков имен каталогов и файлов командой DIR

```
d:\WINLAB>dir /b
papa
d:\WINLAB>dir papa /b
incredible
math
d:\WINLAB>dir papa\math /b
strong
study
d:\WINLAB>dir papa\math\strong /b
papa1.txt
d:\WINLAB>
```

Будучи запущенной без параметров команда DIR отображает список каталогов и файлов текущей директории (см. пример 2.9). Одновременное использование ключа /S совместно с ключом /B инициирует вывод на экран всех вложенных каталогов и файлов с полными путями (пример 2.10).

Пример 2.10. Отображение всех вложенных файлов и подкаталогов

```
d:\WINLAB>dir papa /b /s
d:\WINLAB\papa\incredible
d:\WINLAB\papa\math
d:\WINLAB\papa\incredible\papa3.txt
d:\WINLAB\papa\incredible\papa4.txt
d:\WINLAB\papa\math\strong
d:\WINLAB\papa\math\study
d:\WINLAB\papa\math\strong\papa1.txt
d:\WINLAB\papa\math\study\papa2.txt
d:\WINLAB>
```

В наиболее общем случае формат команды DIR выглядит следующим образом:

DIR [диск:][путь][имя файла] [/A[[:]атрибуты]] [ключи]

/A – отображение файлов с указанными атрибутами: D – каталоги; R – файлы; H – скрытые файлы; S – системные файлы; A – файлы, готовые для архивирования; I – файлы с неиндексированным содержимым; L – точки повторной обработки; префикс «-» имеет значение НЕ.

/B – вывод только имен файлов.

/C – применение разделителя групп разрядов при выводе размеров файлов. Используется по умолчанию. Ключ /-C отключает применение разделителя групп разрядов.

/D – вывод списка в нескольких столбцах с сортировкой по столбцам.

/L – использование нижнего регистра для имен файлов.

/N – новый формат длинного списка, имена файлов выводятся в крайнем правом столбце.

/O – сортировка списка отображаемых файлов: N – по имени (по алфавиту); S – по размеру (с минимального); E – по расширению (по алфавиту); D – по дате и времени (начиная с самого старого); G – начало списка каталогов – префикс «-» обращает порядок; /P – пауза после заполнения каждого экрана.

/Q – вывод сведений о владельце файла.

/R – отображение альтернативных потоков данных файла.

/S – отображение файлов каталога и всех его подкаталогов.

/T – выбор поля времени для сортировки: C – время создания; A – время последнее использования; W – время последнего изменения.

/W – вывод списка в несколько столбцов.

/X – формат аналогичен выводу с ключом /N, но короткие имена выводятся слева от длинных имен файлов. Если короткого имени у файла нет, вместо него выводятся пробелы.

/4 – вывод номера года в четырехзначном формате.

Ненужные или неудачно созданные каталоги могут быть удалены из файловой системы путем использования полностью идентичные по своему действию команды RD или RMDIR (remove directory) в формате:

RMDIR [/S] [/Q] [диск:]путь
RD [/S] [/Q] [диск:]путь

Ключ: /S – удаление каталога со всеми вложенными каталогами.

Ключ: /Q – отключение запроса на подтверждение при удалении вложенных каталогов и при использовании ключа /S.

Применение RD или RMDIR без ключей для удаления каталога PAPA, содержащего вложенные каталоги, не приведет к успеху, тогда как использование ключей полностью решает задачу удаления любых каталогов – о чем свидетельствуют результаты просмотра как командой DIR, так и командой TREE (пример 2.11).

Пример 2.11. Удаление всех вложенных файлов и подкаталогов

```
d:\WINLAB>rd papa
Папка не пуста.
d:\WINLAB>rd papa /q /s
d:\WINLAB>dir /b
d:\WINLAB>tree .
D:\WINLAB
Подпапки отсутствуют
d:\WINLAB>
```

Ошибка при создании каталога может заключаться в его именовании. Например, если в дереве A (см. рис. 2.5) вместо каталога STUDY был создан STADY. При этом совсем необязательно удалять каталог с неправильным именем – достаточно его переименовать. Переименование каталогов может потребоваться и в ряде других случаев. Для этого применяются полностью идентичные команды: REN и RENAME в формате:

RENAME [диск:][путь]имя_файла1 имя_файла2.

REN [диск:][путь]имя_файла1 имя_файла2.

Весьма важно отметить, что команды REN и RENAME – это именно команды переименования каталогов и файлов, а не перемещения (как команда MOVE, см. п. 2.1.4), поэтому для конечного каталога (нового имени) не может быть указано другое местоположение – другой диск или другой каталог (пример 2.12).

Пример 2.12. Переименование каталога

```
d:\WINLAB>dir papa\math /b
stady
```

```
strong
d:\WINLAB>ren papa\math\stady papa\study
Ошибка в синтаксисе команды.
d:\WINLAB>ren papa\math\stady study
d:\WINLAB>dir papa\math /b
strong
study
d:\WINLAB>
```

Важнейшим вопросом для файловой системы является навигация по дереву существующих каталогов, т.е. возможность смены текущего каталога и, соответственно, использование коротких относительных путей для вложенных каталогов вместо абсолютных. Изменение текущего каталога в командном интерпретаторе Cmd.exe осуществляется идентичными командами CD или CHDIR (change directory) в формате:

CHDIR [/D] [диск:][путь]
CD [/D] [диск:][путь]

При отсутствии параметров на экран выводится имя текущего каталога (см. пример 2.8), а при наличии параметра происходит смена текущего каталога – текущим становится каталог, указанный в качестве параметра команды CD или CHDIR через абсолютный или относительный путь (пример 2.13).

Пример 2.13. Вывод имени текущего каталога

```
d:\WINLAB>cd
D:\WINLAB
d:\WINLAB>
```

Если в качестве параметра команде CD или CHDIR указано только имя диска, то результатом будет отображение текущего каталога на указанном диске – при смене диска командный интерпретатор запоминает текущий каталог на диске, и при следующем выборе диска текущий каталог не изменится (пример 2.14).

Пример 2.14. Вывод имени текущего каталога на диске

```
d:\WINLAB\papa\math\study>c:
C:\Users >cd d:
D:\WINLAB\papa\math\study
```

```
C:\Users >d:
d:\WINLAB\papa\math\study>
```

Использование ключа /D в командах CD и CHDIR позволяет осуществлять изменение текущего диска и текущего каталога одной командой (пример 2.15).

Пример 2.15. Одновременная смена текущего диска и текущего каталога

```
d:\WINLAB\papa\math\study>c:
C:\Users>
C:\Users> cd /D d:\winlab\papa
d:\WINLAB\papa>
```

Для выбора текущего каталога, удобного для работы в конкретной ситуации, используются как абсолютные пути к каталогам, так и различные способы записи относительных путей (пример 2.16). Это же касается всех остальных команд Cmd.exe, работающих с каталогами и файлами.

Пример 2.16. Смена текущего каталога с различной записью путей

```
d:\WINLAB>tree .
D:\WINLAB
└───papa
    ├───incredible
    └───math
        ├───strong
        └───study
d:\WINLAB>cd papa\math\strong
d:\WINLAB\papa\math\strong>cd ..\..\incredible
d:\WINLAB\papa\incredible>cd d:\WINLAB\papa\math\study
d:\WINLAB\papa\math\study>cd ..\strong
d:\WINLAB\papa\math\strong>cd \winlab
d:\WINLAB>
```

При включении расширенной обработки команд команда CHDIR перестает рассматривать пробелы как разделители, что позволяет перейти в подкаталог, имя которого содержит пробелы, не заключая все имя каталога в кавычки (пример 2.17), при этом имя каталога, указанного в команде, преобразуется к тому же регистру, что и для существующих на диске имен каталогов. При отключении расширенной обработки команд может быть использован только вариант с кавычками.

Пример 2.17. Смена текущего каталога в режиме расширенной обработки

```
d:\WINLAB>md "new name"
d:\WINLAB>chdir new name
d:\WINLAB\new name>
d:\WINLAB\new name>cd ..
d:\WINLAB>rd new name
Не удается найти указанный файл.
d:\WINLAB>rd "new name"
d:\WINLAB>
```

2.1.3. Работа с текстовыми файлами в Cmd.exe

После построения дерева каталогов (см. рис. 2.5) перейдем к рассмотрению основных способов создания и редактирования простых текстовых файлов (см. табл. 2.2).

Для пользователей Windows самым очевидным инструментом такой работы является приложение «Блокнот» (Notepad), запускаемое в русском варианте графического интерфейса Windows как:

Пуск > Все программы > Стандартные > Блокнот

или в англоязычном интерфейсе:

Start > All Programs > Accessories > Notepad

Для корректного отображения текста фалов, созданных в текстовом редакторе Notepad, необходимо установить для командной строки кодовую страницу 1251 (для каждого национального алфавита используется своя так называемая кодовая страница, необходимая для корректного отображения национальных шрифтов) и выбрать для отображения текста один из True Type шрифтов: Consolas или Lucida Console (выбор шрифтов подробно рассмотрен в п. 2.2.2). Для просмотра текущей кодовой страницы, в данный момент используемой командным интерпретатором, и для ее изменения служит команда CHCP (см. далее в п. 2.2.1, пример 2.28).

Приложение «Блокнот» может быть запущено и из командной строки (пример 2.18). В данном примере создается текстовый файл papa1.txt (см. рис. 2.5) с использованием абсолютного пути, включающего имя директории и имя файла.

Пример 2.18. Вывод текстового файла на экран

```
d:\WINLAB>tree papa
D:\WINLAB\PAPA
```

```
├───incredible
└───math
    ├───strong
    └───study
d:\WINLAB>notepad papa\math\strong\papa1.txt
d:\WINLAB>
```

После подтверждения запроса на создание файла и набора соответствующего текста (рис. 2.6) необходимо сохранить и закрыть файл (*Файл > Сохранить*), после чего убедиться в существовании файла командами DIR или TREE.

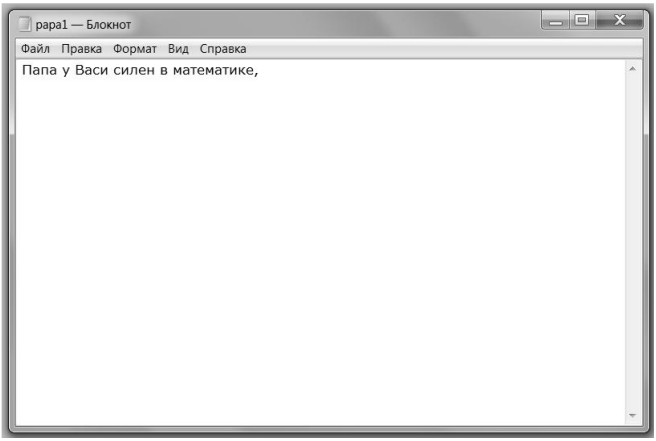


Рис. 2.6. Текстовый файл в редакторе Notepad

Для просмотра существующих текстовых файлов из командной строки применяются команды TYPE или MORE (пример 2.19).

Пример 2.19. Вывод текстового файла на экран

```
d:\WINLAB>type papa\math\strong\papa1.txt
Папа у Васи силен в математике,
d:\WINLAB>cd papa\math\strong
d:\WINLAB\papa\math\strong>more papa1.txt
Папа у Васи силен в математике,
d:\WINLAB\papa\math\strong>
```

При просмотре текстовых файлов можно использовать абсолютные и относительные пути к ним из текущей директории. Если

с помощью команды CD сделать текущей директорию, в которой находится просматриваемый файл, то команде TYPE или MORE достаточно указать в качестве параметра только имя файла.

Команда TYPE осуществляет вывод содержимого одного или нескольких текстовых файлов в формате:

TYPE [диск:][путь]имя_файла

Команда MORE выводит данные текстового файла по частям, размером в один экран. Наиболее простой форматы ее использования выглядит следующим образом:

MORE /E [/C] [/P] [/S] [/Tn] [+n] [файлы]

/E – разрешение использования дополнительных возможностей;
/C – очистка экрана перед выводом каждой страницы;
/P – учет символов перевода страницы;
/S – сжатие нескольких пустых строк в одну строку;
/Tn – замена табуляции n пробелами (по умолчанию n = 8);
+n – начало вывода первого файла со строки с номером n.

Для разделения имен в списке отображаемых файлов используются пробелы. Если включен режим использования дополнительных возможностей, то MORE будет работать как консольное приложение, в командной строке которого можно вводить следующие команды:

P n – вывод следующих n строк;
S n – пропуск следующих n строк;
F – вывод следующего файла;
Q – завершение работы;
= – вывод номера строки;
? – вывод строки подсказки;
<пробел> – вывод следующей страницы;
<ENTER> – вывод следующей строки.

Помимо текстового редактора Notepad, являющегося неотъемлемой частью стандартного набора приложений Windows, командная строка позволяет создавать и редактировать текстовые файлы при помощи утилиты Edit – текстового редактора, унаследованного еще от MS DOS. Меню (допускающее управление мышью) и команды (сочетания клавиш) этого редактора практически анало-

гичны Notepad – и в этом нет ничего удивительного, поскольку Notepad является прямым наследником Edit. В рамках данного лабораторного практикума использование этой весьма полезной утилиты осложняет проблемы с русификацией.

Для создания текстовых файлов непосредственно в командной строке можно использовать копирование текста (см. п. 2.1.4) со стандартной консоли (клавиатуры), а также перенаправление стандартного ввода/вывода команд в текстовый файл (см. п. 2.2.3).

Если в командной строке Windows набрать и подтвердить путь к текстовому файлу (с расширением «.txt»), то это приведет к открытию текстового файла в редакторе Notepad – поскольку txt-файлы, по умолчанию, ассоциированы с этим приложением Windows.

Удаление одного или нескольких файлов в командной строке Windows осуществляется равноценными по действию командами DEL и ERASE (пример 2.20). Для реализации примера необходимо создать в Notepad текстовый файл (скажем, ERROR.TXT) и разместить его в своем рабочем каталоге (в примерах практикума – это каталог D:\WINLAB).

Пример 2.20. Удаление файла (необязательно текстового)

```
d:\WINLAB>dir /b
error.txt
papa
d:\WINLAB>erase error.txt
d:\WINLAB>dir /b
papa
d:\WINLAB>
```

В общем случае команды DEL и ERASE имеют нижеследующий формат (далее по тексту формата: «имена» – список из одного или нескольких файлов или каталогов, если указан каталог, будут удалены все файлы в этом каталоге):

DEL [/P] [/F] [/S] [/Q] [/A[[:]атрибуты]] имена
ERASE [/P] [/F] [/S] [/Q] [/A[[:]атрибуты]] имена

/P – запрос подтверждения перед удалением каждого файла;
/F – принудительное удаление файлов, только для чтения;
/S – удаление указанных файлов из всех подкаталогов;
/Q – отключение запроса на подтверждение удаления файлов;

/A – отбор файлов для удаления по атрибутам: R – файлы, доступные только для чтения; S – системные файлы; H – Скрытые файлы; A – файлы, готовые для архивирования; I – файлы с неиндексированным содержимым; L – точки повторной обработки; префикс «-» имеет значение НЕ.

При включении расширенной обработки команд DEL и ERASE результаты вывода для ключа /S принимают обратный характер, т.е. выводятся только имена удаленных файлов, а не файлов, которые не удалось найти.

2.1.4. Копирование и перемещение файлов и директорий

Командный интерпретатор Cmd.exe предоставляет широкие возможности по редактированию файловой системы путем копирования, перемещения и переименования файлов и директорий с вложенными каталогами и файлами.

Команда COPY позволяет копировать один или несколько файлов из одной директории в другую, используя как абсолютные, так и относительные пути к файлам (в примере 2.21 считается, что дерево см. рис. 2.5 построено полностью).

Пример 2.21. Копирование файлов

```
d:\WINLAB>tree papa /f
D:\WINLAB\PAPA
├───incredible
│       papa3.txt
│       papa4.txt
└───math
    ├───strong
    │       papa1.txt
    └───study
            papa2.txt
d:\WINLAB>copy papa\math\study\papa2.txt papa\math\strong
Скопировано файлов:         1.
d:\WINLAB>cd papa\math\strong
d:\WINLAB\papa\math\strong>dir /b
papa1.txt
papa2.txt
d:\WINLAB\papa\math\strong>copy ..\..\incredible
..\..\incredible\papa3.txt
..\..\incredible\papa4.txt
Скопировано файлов:         2.
d:\WINLAB\papa\math\strong>dir /b
```

```
papa1.txt
papa2.txt
papa3.txt
papa4.txt
d:\WINLAB\papa\math\strong>
```

В наиболее общем случае команда COPY имеет следующий формат (далее по тексту описания формата: «источник» – имена одного или нескольких копируемых файлов, а «результат» – каталог и/или имя для конечных файлов):

COPY [/D] [/V] [/N] [/Y | /-Y] [/Z] [/L] [/A | /B] источник [/A | /B]
[+ источник [/A | /B] [+ ...]] [результат [/A | /B]]

/A – текстовый файл ASCII;
/B – двоичный файл;
/D – указание на возможность создания шифрованного файла;
/V – проверка правильности копирования файлов;
/N – использование, если возможно, коротких имен;
/Y – подавление запроса подтверждения на перезапись;
/-Y – запрос подтверждения на перезапись конечного файла;
/Z – копирование сетевых файлов с возобновлением;
/L – копирование символической ссылки, вместо файла.

Чтобы объединить несколько файлов в один, необходимо указать один конечный и несколько исходных файлов, используя формат: «файл1+файл2+файл3+...», использование масок, в частности «звездочки», обозначающей «любой набор символов», приводит ровно к тому же результату (пример 2.22).

Пример 2.22. Вывод текстового файла на экран

```
d:\WINLAB\papa\math\strong>copy papa1.txt+papa2.txt+papa3.txt+papa4.txt papa.txt
…….
d:\WINLAB\papa\math\strong>dir /b
papa.txt
…….
d:\WINLAB\papa\math\strong>type papa.txt
Папа у Васи силен в математике,
Учится папа за Васю весь год,
Где это видано, где это слыхано:
Папа решает, а Вася сдает!
d:\WINLAB\papa\math\strong>del papa.txt
d:\WINLAB\papa\math\strong>copy *1.txt+*2.txt+*3.txt+*4.txt papa.txt
```

```
d:\WINLAB\papa\math\strong>type papa.txt
d:\WINLAB\papa\math\strong>del papa.txt
d:\WINLAB\papa\math\strong>copy *.txt papa.txt
d:\WINLAB\papa\math\strong>type papa.txt
Папа у Васи силен в математике,
Учится папа за Васю весь год,
Где это видано, где это слыхано:
Папа решает, а Вася сдает!
d:\WINLAB\papa\math\strong>
```

Перемещение одного и более файлов и папок осуществляется командой MOVE (пример 2.23) в формате (где «назначение» – путь к новому местоположению файл/файлов):

MOVE [/Y | /-Y] [диск:][путь]файл1[,файла2...] назначение

Команда MOVE может также служить для переименования папок, будучи записанной в формате:

MOVE [/Y | /-Y] [диск:][путь]имя_папки новое_имя_папки

При одновременном перемещении и переименовании одного файла можно указать и его новое имя. Ключ /Y позволяет перезаписывать файлы без предупреждения, /-Y – предупреждает о перезаписи существующего файла.

Пример 2.23. Вывод текстового файла на экран

```
D:\WINLAB>cd papa\math\strong
d:\WINLAB\papa\math\strong>tree .. /f
D:\WINLAB\PAPA\MATH
├───strong
│       papa.txt
│       papa1.txt
│       papa2.txt
│       papa3.txt
│       papa4.txt
│
└───study
        papa2.txt
d:\WINLAB\papa\math\strong>move ..\study\papa2.txt
Заменить d:\WINLAB\papa\math\strong\papa2.txt [Yes (да)/No (нет)/All (все)]: y
Перемещено файлов:        1.
d:\WINLAB\papa\math\strong>tree .. /f
D:\WINLAB\PAPA\MATH
```

```
├───strong
│       papa.txt
│       papa1.txt
│       papa2.txt
│       papa3.txt
│       papa4.txt
└───study
d:\WINLAB\papa\math\strong>
```

Наиболее мощным инструментом копирования файлов, директорий и структур каталогов является утилита XCOPY. Самое простое применение XCOPY – копирование непустых каталогов (с файлами и подкаталогами), т.е. то, что невозможно выполнить при помощи команды COPY. Например, можно скопировать каталог STRONG в рабочий каталог под именем BACKUP (пример 2.24).

Пример 2.24. Копирование директорий

```
d:\WINLAB>tree . /f
D:\WINLAB
└───papa
    └───math
        └───strong
                papa.txt
                papa1.txt
                papa2.txt
                papa3.txt
                papa4.txt
d:\WINLAB>xcopy papa\math\strong  backup\
Скопировано файлов: 5.
d:\WINLAB>tree . /f
D:\WINLAB
├───backup
│       papa.txt
│       papa1.txt
│       papa2.txt
│       papa3.txt
│       papa4.txt
└───papa
d:\WINLAB>
```

Более серьезные задачи, которые позволяет решать утилита XCOPY, – это создание резервных копий и синхронизация каталогов, в том числе в разных местах, на разных носителях (Flash-дисках, внешних жестких дисках и т.д.). Для этого предусмотрены ключи, позволяющие исключить копирование одних и тех же фай-

лов и директорий, что значительно сокращает время при копировании больших объемов данных.

Формат вызова утилиты XCOPY позволяет реализовать гибкую и точную настройку копирования данных:

XCOPY источник [целевой_объект] [/A | /M] [/D[:дата]] [/P] [/S [/E]] [/V] [/W] [/C] [/I] [/Q] [/F] [/L] [/G] [/H] [/R] [/T] [/U] [/K] [/N] [/O] [/X] [/Y] [/-Y] [/Z] [/B] [/EXCLUDE:файл1[+файл2][+файл3]...]

/A – копирование только файлов с архивным атрибутом.

/M – копирование со снятием архивного атрибута.

/D:m-d-y – копирование файлов, измененных не ранее указанной даты. Если дата не указана, заменяются файлы на более старые, чем исходные.

/EXCLUDE:файл1[+файл2][+файл3] – список файлов, содержащих исключающие строки. Если какая-либо строка совпадает с любой частью абсолютного пути к копируемому файлу, то файл исключается из операции копирования.

/P – вывод запросов перед созданием каждого нового файла.

/S – копирование только непустых каталогов с подкаталогами.

/E – копирование каталогов с подкаталогами, включая пустые.

/V – проверка размера каждого нового файла.

/W – вывод запроса на нажатие клавиши перед копированием.

/C – копирование вне зависимости от наличия ошибок.

/I – если целевой объект не существует и копируется несколько файлов, считается, что целевой объект задает каталог.

/Q – запрет вывода имен копируемых файлов.

/F – вывод полных имен исходных и целевых файлов.

/L – вывод имен копируемых файлов.

/G – копирование зашифрованных файлов в целевой каталог, не поддерживающий шифрование.

/H – копирование, среди прочих, скрытых и системных файлов.

/R – перезапись файлов, предназначенных только для чтения.

/T – создание структуры каталогов без копирования файлов.

/U – копирование только уже имеющихся файлов.

/K – копирование атрибутов файлов.

/N – использование коротких имен при копировании.

/O – копирование сведений о владельце и данных ACL.

/X – копирование параметров аудита файлов.

/Y – подавление запроса подтверждения перезаписи файлов.

/-Y – запрос подтверждения на перезаписи файлов.

/Z – копирование сетевых файлов с возобновлением.

/B – символическая ссылка вместо целевого объекта.

/J – копирование с использованием небуферизованного ввода/вывода. Рекомендуется для очень больших файлов.

2.2. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ CMD.EXE

2.2.1. Полезные инструменты командной строки

Данный лабораторный практикум ориентирован на студентов, не специализирующихся в области системного администрирования. Более того, предполагается выполнение практикума в компьютерном классе. У компьютерного класса, обычно являющегося локальной сетью, как правило, есть системный администратор, который вряд ли сильно обрадуется, если особо любознательные студенты даже мало-мальски вмешаются в настройки системы. В связи с этим рассмотрен минимальный набор параметров командной строки, связанный с внешним видом и кодировкой, а также устранением «зависания» при выполнении команд (табл. 2.3).

Таблица 2.3. Управление параметрами Cmd.exe и выполнением команд

Операция	Формат команды
Приостановить / продолжить выполнение команды	Сочетание клавиш: Ctrl + S
Прервать выполнение команды	Сочетание клавиш: Ctrl + C
Задать цвета фона и текста	COLOR [цвета]
Изменить приглашения	PROMPT [описание приглашения]
Просмотреть и изменить кодовую страницу	CHCP [nnn]
Очистить экран	CLS
Работать с буфером хронологии команд	Клавиши: F7, F8, F9
Автоматически определить имена каталогов и файлов	Клавиша TAB

В частности, это может помочь в том случае, когда дисциплинированному студенту достался компьютер, на котором перед этим работал менее дисциплинированный, но более любознательный студент, а системный администратор сильно занят.

В окне запущенного командного интерпретатора Cmd.exe, по умолчанию, помещается 25 строк по 80 символов. По мере ввода команд и данных окно заполняется, а его содержимое прокручивается вверх (соответственно, в правой части окна, как и во всех приложениях Windows, появляется полоса прокрутки).

Иногда, например, при просмотре большого текстового файла выдача команды не помещается на одном экране и уходит вверх, до того как ее успеют прочитать. Чтобы приостановить вывод, необходимо нажать сочетание клавиш «Ctrl+S», что означает сначала нажатие «Ctrl» и, не отпуская «Ctrl», клавиши «S». Возобновление вывода осуществляется повторным нажатием «Ctrl+S». Нажатие сочетания клавиш «Ctrl+C» прекращает работу команды, в том числе «зависшей» по какой либо причине.

Команда COLOR. Окно командного интерпретатора по умолчанию имеет черный фон и белый передний план (текст). Команда COLOR (без параметров) устанавливает цвета по умолчанию для переднего плана и фона в текстовых окнах. В общем случае формат команды имеет следующий вид:

COLOR [цвета]

Атрибуты цветов задаются в виде двух шестнадцатеричных цифр: цвет фона и цвет переднего плана. Каждая цифра может иметь следующие значения:

0 = Черный	8 = Серый
1 = Синий	9 = Светло-синий
2 = Зеленый	A = Светло-зеленый
3 = Голубой	B = Светло-голубой
4 = Красный	C = Светло-красный
5 = Лиловый	D = Светло-лиловый
6 = Желтый	E = Светло-желтый
7 = Белый	F = Ярко-белый

Пример 2.25. Установка цветовой гаммы командной оболочки

```
d:\WINLAB>color fc
d:\WINLAB>
```

Пример 2.25 задает светло-красный передний план на ярко-белом фоне. Если аргумент не указан, команда COLOR восстанавливает исходный выбор цветов, каким он был на момент запуска командного интерпретатора (пример 2.26).

Пример 2.26. Установка цветовой гаммы по умолчанию

```
d:\WINLAB>color fc
d:\WINLAB>
```

Команда PROMPT. Если на одном компьютере с командной строкой работают несколько человек, то у каждого могут иметься свои предпочтения в отношении того, как должно выглядеть приглашение командной строки – по умолчанию это отображение абсолютного пути в текущий каталог, заканчивающееся правой угловой скобкой (или знаком «больше»), например: «d:\WINLAB>». Однако приглашение командной строки может быть изменено при помощи команды PROMPT, имеющей формат:

PROMPT [описание приглашения]

Описание приглашения командной строки может включать обычные символы и следующие коды:

$H BACKSPACE (удаление предыдущего символа).
$E ESC (символ ASCII с кодом 27).
$A – & (амперсанд). $B – | (вертикальная черта).
$C – ((левая круглая скобка). $D – текущая дата.
$F –) (правая круглая скобка). $G > (знак "больше")
$L – < (знак "меньше"). $N – текущий диск.
$P – текущие диск и каталог. $Q – = (знак равенства).
$S – пробел. $T – текущее время.
$V – номер версии Windows. $$ – $ (символ доллара).
$_ – возврат каретки и перевод строки

Пример 2.27. Изменение приглашения командной строки

```
D:\WINLAB>prompt $$
$
```

```
$prompt $p$g
D:\WINLAB>
D:\WINLAB>prompt $d$a
12.04.2013&
12.04.2013&prompt
D:\WINLAB>
```

Например, можно поменять приглашение командной строки на знак доллара ($$) или текущую дату с конечным амперсандом (DA), а затем вернуть в исходное состояние (PG). Применение команды PROMPT без параметров возвращает приглашение командной строки в состояние по умолчанию (пример 2.27).

Если включен режим расширенной обработки, командой PROMPT (по умолчанию) поддерживаются следующие дополнительные символы форматирования:

$+ – отображение нужного числа знаков плюс (+) в зависимости от текущей глубины стека каталогов PUSHD по одному знаку на каждый сохраненный путь;

$M – отображение полного имени удаленного диска, связанного с именем текущего диска, или пустой строки, если текущий диск не является сетевым.

Команда CHCP. В Windows-системах поддерживаются различные режимы кодирования текстов на основе национальных алфавитов. Для каждого национального алфавита используется своя так называемая кодовая страница. Для корректного отображения кириллического текста файлов, созданных в текстовом редакторе Notepad, необходимо установить для командной строки кодовую страницу Windows 1251 и выбрать для отображения текста один из True Type шрифтов: Consolas или Lucida Console (выбор шрифтов рассмотрен в п. 2.2.2). Для просмотра текущей кодовой страницы, используемой командным интерпретатором, и для ее изменения служит команда CHCP.

По умолчанию, в русифицированных версиях Windows командным интерпретатором используется кодовая страница 866, унаследованная от MS DOS, которую необходимо поменять на кодовую страницу Windows-1251 (пример 2.28) для корректного отображения простых текстовых файлов, созданных с использованием различных Windows-приложений.

Пример 2.28. Просмотр и изменение кодовой страницы

```
D:\WINLAB>chcp
Текущая кодовая страница: 866
D:\WINLAB>chcp 1251
Текущая кодовая страница: 1251
D:\WINLAB>
```

Команда CLS. При интенсивной работе с командной строкой на экране накапливается и прокручивается вверх информация, переставшая быть актуальной. Для того чтобы очистить от нее экран, применяется команда CLS (мнемоника команды происходит от сокращения английского clear screen – очистить экран). Команда CLS предельно проста и не содержит параметров.

Работа с буфером хронологии команд. Команды, вводимые в командной строке, накапливаются в буфере хронологии и к этим командам можно получить доступ. Самый простой способ заключается в использовании клавиш «стрелка вверх» и «стрелка вниз» – при этом происходит перемещение вверх и вниз по списку команд, хранящемуся в буфере хронологии. Пролистав таким образом список буфера хронологии команд, можно остановиться на нужной команде, отредактировать ее (используя клавиши «стрелка влево», «стрелка вправо», «Backspase» и «Delete») и затем выполнить, нажав клавишу «Enter».

Буфер хронологии команд может быть показан в виде списка во всплывающем окне, нажатием клавиши «F7», после чего с помощью клавиш-стрелок можно выбрать команду из списка (рис. 2.7а). Если же известен номер команды в списке (после неоднократного использования клавиши «F7»), то можно нажать «F9», и ввести на клавиатуре номер команды – ввод номера 1 (рис. 2.7б) приведет к выбору команды CLS (см. рис. 2.7а). Нажатие «Enter» приведет к выполнению команды, а клавиша «Esc» закроет всплывающее окно без выполнения команды.

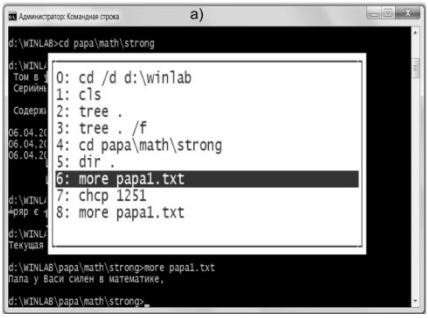


Рис. 2.7а. Список команд по «F7»

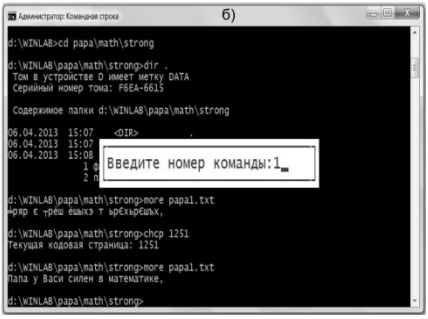


Рис. 2.7б. Ввод номера команды

Рассмотрим еще один способ работы с буфером хронологии команд. Ввод нескольких первых букв нужной команды и нажатие клавиши «F8» приведет к тому, что командная оболочка попытается найти последнюю по хронологии команду, начинающуюся с введенных символов, и воспроизведет ее в командной строке. Повторное нажатие клавиши «F8» запустит поиск в буфере предыдущей по хронологии команды, начинающейся с тех же символов. Так, ввод в командной строке символов «CD» и последовательное нажатие «F8» приведет к последовательному перебору команд, содержащихся в буфе: «CD PAPA\MATH\STRONG» и «CD /D D:\WINLAB», а при вводе только символа «C» (не перепутать с русским) на экране появятся также «CHCP 1251» и «CLS».

Автоматическое определение имен каталогов и файлов. В заключение этого пункта стоит рассмотреть еще один полезный инструмент – использование клавиши «TAB» для автоматизации ввода имен файлов. Действие похоже на «F8» только анализируется не буфер хронологии команд, а список имен каталогов и файлов текущей директории и дописывает имя первого файла, имя которого начинается с введенных символов. Дальнейшим нажатием клавиши «TAB» можно осуществить перебор имен каталогов и файлов, для которых начальные символы имени такие же (пример 2.29).

Пример 2.29. Автоматизация ввода имен клавишей «TAB»

```
d:\WINLAB>tree .
D:\WINLAB
└───papa
    ├───incredible
```

```
└─────math
      ├─────strong
      └─────study
d:\WINLAB>cd НАЖАТИЕ ТАВ ВЫБЕРЕТ ЕДИНСТВЕННЫЙ КАТАЛОГ PAPA
d:\WINLAB\papa>cd m НАЖАТИЕ ТАВ ДОСТРОИТ ИМЯ КАТАЛОГА MATH
```

2.2.2. Настройка параметров командной оболочки

Настройка параметров командной оболочки осуществляется в режиме привычного графического интерфейса Windows. Необходимо выбрать самый нижний пункт меню – «Свойства» (Properties) в таблице контекстного меню, которое открывается щелчком мыши по значку командной строки в верхнем левом углу окна командной оболочки или щелчком правой кнопки мыши по строке заголовка того же окна (рис. 2.8). При этом откроется диалоговое окно Command Prompt Properties (свойства: «Командная строка»), содержащее четыре вкладки:

- общие (Options) – настройка размера курсора, параметров отображения и редактирования, а также хронологии команд;
- шрифт (Font) – настройка размера и начертания шрифта в окне командной строки;
- расположение (Layout) – задание размера буфера экрана, размера и позиции окна;
- цвета (Colors) – настройка цветов текста и фона в окне командной строки.

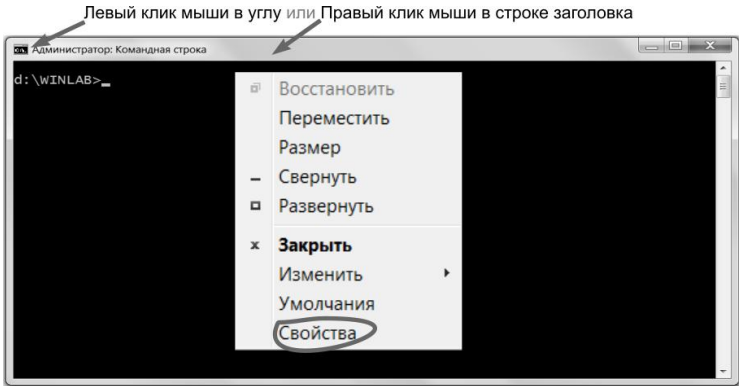


Рис. 2.8. Запуск диалогового окна настройки свойств командной строки

Во вкладке «Общие» (рис. 2.9) целесообразно установить флажок QuickEdit Mode (выделение мышью), чтобы с помощью мыши выделять и вставлять текст в окне командной строки, как в текстовом редакторе.

Флажок Insert Mode (быстрая вставка) является переключателем двух режимов редактирования – вставки (флажок выставлен) и замещения (флажок сброшен). Режим вставки позволяет редактировать командную строку как в обычном текстовом редакторе.

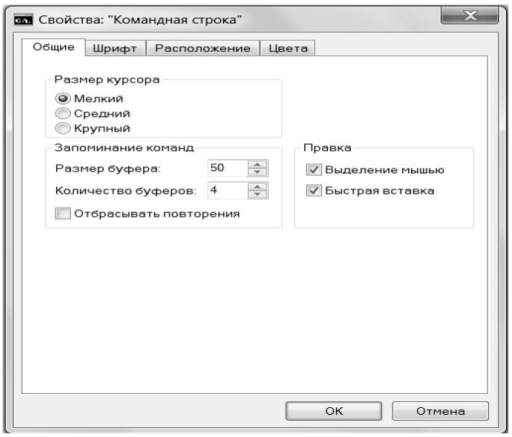


Рис. 2.9. Вкладка «Общие» окна свойств командной строки

Размер курсора (по умолчанию – «Мелкий») и параметры запоминания команд (по умолчанию: «Размер буфера» – 50; «Количество буферов» – 4) лучше не трогать, флажок «Отбрасывать повторения» не устанавливать.

Во вкладке «Шрифт» (рис. 2.10) можно просмотреть русский текст, набранный в Notepad в режиме командной строки. Для этого помимо выбора кодовой страницы 1251 (команда CHCP, описана ранее) необходимо, чтобы текст в командной строке отображался одним из True Type шрифтов: Consolas или Lucida Console – размер этих шрифтов задается в пунктах (pt), в отличие от точечных шрифтов, измеряемых в пикселях (px).

Во вкладке «Расположение» (рис. 2.11) при необходимости можно указать высоту буфера (количество строк), достаточную для того, чтобы просмотреть вывод предыдущих команд, установив

значение в диапазоне. Для того чтобы окно командной строки появлялось в заданном месте экрана, необходимо сбросить флажок Let System Position Window (автоматический выбор) и указать левый (Left) и верхний (Top) края координаты левого верхнего угла окна командной строки.

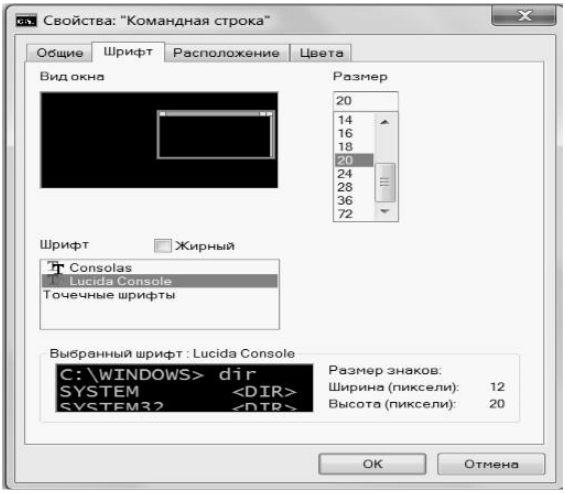


Рис. 2.10. Вкладка «Шрифт» окна свойств командной строки

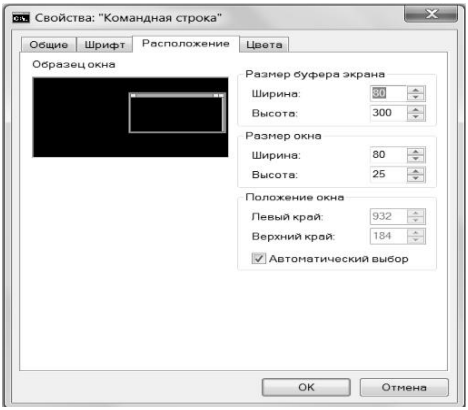


Рис. 2.11. Вкладка «Расположение» окна свойств командной строки

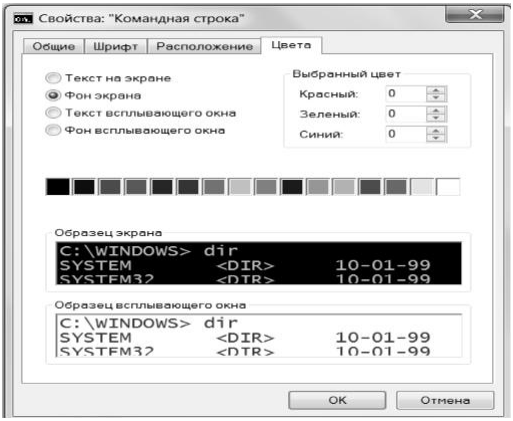


Рис. 2.12. Вкладка «Цвета» окна свойств командной строки

Во вкладке «Цвета» (рис. 2.12) можно задать текст и фон экрана в окне командной строки: «Текст на экране» (Screen Text) и «Фон экрана» (Screen Background), а также текст и фон экрана во всплывающих, например, окнах буфера хронологии команд и других диалоговых окнах, генерируемых при выполнении команд в окне командной строки: «Текст всплывающего окна» (Popup Text) и «Фон всплывающего окна» (Popup Background).

Завершение настройки свойств командной оболочки подтверждается нажатием «ОК» внизу диалогового окна свойств командной строки Windows и подтверждением запроса на применение указанных параметров.

2.2.3. Перенаправление ввода/вывода и конвейеры

Командная оболочка позволяет запускать команды разных типов: встроенные команды, утилиты и приложения, рассчитанные на командную строку. По умолчанию, все они получают ввод из параметров, указываемых при вызове команды, и выводят результат в стандартное окно консоли. Но иногда нужно либо получить ввод из другого источника, либо направить вывод в файл или на другое устройство вывода, например на принтер или в файл. Более того, в командном интерпретаторе Cmd.exe существует возможность перенаправить вывод отработавшей команды другим командам, в ка-

честве ввода – это так называемая *конвейеризация* (piping). Формы перенаправления ввода/вывода проиллюстрированы в табл. 2.4.

Таблица 2.4. Варианты перенаправления ввода/вывода и конвейеров

№	Вариант перенаправления	Комментарий
1	команда > [путь]файл	Вывод команды перенаправляется в файл – если такого файла нет, то он создается, а если файл существует, то он перезаписывается
2	команда >> [путь]файл	Вывод команды перенаправляется в файл – если такого файла нет, то он создается, а если файл существует, то данные добавляются в него
3	команда < [путь]файл	Команда получает ввод из файла
4	команда1 | команда2 | команда3 …	Конвейер: вывод первой команды является вводом для второй, вывод второй – вводом для третьей и т.д.

Любой из рассмотренных четырех вариантов может рассматриваться как отдельная команда и быть составной частью любого из тех же четырех вариантов.

При построении дерева B (см. рис. 2.5) для создания текстовых файлов без помощи Notepad можно использовать перенаправление стандартного вывода команды ECHO (пример 2.26), предназначенной для вывода текстовых сообщений в формате:

ECHO [сообщение]

Пример 2.26. Создание файла перенаправлением стандартного вывода в файл

```
d:\WINLAB>md vasya\every\answer, vasya\mod\yeah, vasya\mod\moscow
d:\WINLAB>tree vasya
D:\WINLAB\VASYA
├───every
│   └───answer
└───mod
    ├───moscow
    └───yeah
d:\WINLAB>chcp 1251
Текущая кодовая страница: 1251
d:\WINLAB>echo На это каждый ответит, каждый ответит: > vasya1.txt
d:\WINLAB>dir /b
```

```
backup
papa
vasya
vasya1.txt
d:\WINLAB>move vasya1.txt vasya\every\answer\
Перемещено файлов:        1.
d:\WINLAB>tree vasya /f
D:\WINLAB\VASYA
├───every
│   └───answer
│           vasya1.txt
└───mod
    ├───moscow
    └───yeah
d:\WINLAB>
d:\WINLAB>more < vasya\every\answer\vasya1.txt
На это каждый ответит, каждый ответит:
d:\WINLAB>
```

Аналогичным образом предлагается создать файл VASYA2.TXT (см. табл. 2.2, рис. 2.5).

Следует отметить, что команда ECHO используется также для переключения режима отображения команд на экране (ввод ECHO без параметра служит для определения текущего состояния):

ECHO [ON | OFF]

К перенаправлению ввода/вывода можно отнести копирование данных командой COPY с *консоли* в файл и из файла на консоль. По умолчанию, устройством ввода консоли является клавиатура, а устройством вывода – экран монитора, в общем случае входные и выходные устройства консоли могут быть переопределены. Системное имя консоли в Windows: CON.

Пример 2.27. Перенаправление ввода/вывода с консоли и на консоль

```
d:\WINLAB>tree vasya /f
D:\WINLAB\VASYA
├───every
│   └───answer
│           vasya1.txt
│           vasya2.txt
└───mod
    ├───moscow
    └───yeah
```

```
d:\WINLAB>copy con vasya\mod\yeah\vasya3.txt
Ну кто его не знает? Yeah, yeah!
^Z – Символ «конец файла», инициируемый нажатием «Ctrl+Z»
Скопировано файлов:         1.
d:\WINLAB>tree vasya /f
D:\WINLAB\VASYA
├───every
│   └───answer
│           vasya1.txt
│           vasya2.txt
└───mod
    ├───moscow
    └───yeah
            vasya3.txt
d:\WINLAB>more < vasya\mod\yeah\vasya3.txt
Ну кто его не знает? Yeah, yeah!
d:\WINLAB>copy vasya\mod\yeah\vasya3.txt con
Ну кто его не знает? Yeah, yeah!
Скопировано файлов:         1.
d:\WINLAB>
```

Когда речь идет о вводе данных, то CON – это клавиатура, а когда о выводе данных, то именем CON обозначается экран монитора. Используя копирование с консоли CON (пример 2.27), создадим файлы VASYA3.TXT и VASYA3.TXT как на ветке B (см. рис. 2.5, табл. 2.2).

Перенаправление вывода в файл с добавлением строк: «>>» будет рассмотрено немного позже, теперь же необходимо разобраться с *конвейерами* команд. Наиболее часто конвейеры используются в задачах поиска и сортировки данных – для этого в основном используются команды: FIND и MORE.

Пример 2.28. Применение конвейера для получения списка текстовых файлов

```
d:\WINLAB>tree vasya /f

D:\WINLAB\VASYA
├───every
│   └───answer
│           vasya1.txt
│           vasya2.txt
└───mod
    ├───moscow
    │       vasya4.txt
```

```
        └────yeah
                vasya3.txt

d:\WINLAB>dir vasya /b /s

d:\WINLAB\vasya\every
d:\WINLAB\vasya\mod
d:\WINLAB\vasya\every\answer
d:\WINLAB\vasya\every\answer\vasya1.txt
d:\WINLAB\vasya\every\answer\vasya2.txt
d:\WINLAB\vasya\mod\moscow
d:\WINLAB\vasya\mod\yeah
d:\WINLAB\vasya\mod\moscow\vasya4.txt
d:\WINLAB\vasya\mod\yeah\vasya3.txt

d:\WINLAB>dir vasya /b /s | find ".txt"

d:\WINLAB\vasya\every\answer\vasya1.txt
d:\WINLAB\vasya\every\answer\vasya2.txt
d:\WINLAB\vasya\mod\moscow\vasya4.txt
d:\WINLAB\vasya\mod\yeah\vasya3.txt

d:\WINLAB>
```

Команда FIND ищет строки в файлах или в тексте ввода и выводит строки, соответствующие условию. Например, можно получить список всех файлов с расширением .txt в текущем каталоге и его подкаталогах, перенаправив вывод команды DIR на ввод FIND через конвейер (пример 2.28).

Пример 2.29. Применение конвейера и цикла для копирования файлов

```
d:\WINLAB>cd vasya
d:\WINLAB\vasya>dir /b
every
mod
vasya.list

d:\WINLAB\vasya>dir /b /s | find ".txt" > vasya.list

d:\WINLAB\vasya>dir /b
every
mod
vasya.list

d:\WINLAB\vasya>more vasya.list
```

```
d:\WINLAB\vasya\every\answer\vasya1.txt
d:\WINLAB\vasya\every\answer\vasya2.txt
d:\WINLAB\vasya\mod\moscow\vasya4.txt
d:\WINLAB\vasya\mod\yeah\vasya3.txt

d:\WINLAB\vasya>for /f %x in (vasya.list) do copy /y %x every\answer\

d:\WINLAB\vasya\every\answer\vasya1.txt
Невозможно скопировать файл поверх самого себя.

d:\WINLAB\vasya\every\answer\vasya2.txt
Невозможно скопировать файл поверх самого себя.
d:\WINLAB\vasya\mod\moscow\vasya4.txt
Скопировано файлов:         1.
d:\WINLAB\vasya\mod\yeah\vasya3.txt
Скопировано файлов:         1.

d:\WINLAB\vasya>dir /b every\answer
vasya1.txt
vasya2.txt
vasya3.txt
vasya4.txt

    d:\WINLAB\vasya>
```

Команда FIND как фильтр текстовых строк в одном или нескольких файлах может применяться в формате:

FIND [/V] [/C] [/N] [/I] [/OFF[LINE]] «строка» [путь]имя_файла

/V – вывод всех строк, не содержащих заданную строку.
/C – вывод общего числа строк, содержащих заданную строку.
/N – вывод номеров отображаемых строк.
/OFF[LINE] – не пропускать атрибут «Автономный».
/I – поиск без учета регистра символов.

Если путь не задан, то команда FIND выполняет поиск либо в тексте консоли, либо в тексте, переданном по конвейеру другой командой.

В предыдущем примере результаты фильтрации данных (список всех текстовых файлов директории VASYA) направлены в стандартный поток вывода – на экран, поэтому они носят чисто информативный характер. Для того чтобы использовать этот список в

практических целях, например для копирования всех файлов директории в один каталог, список должен находиться в файле. Имя такого файла произвольно – для определенности можно дать имя файлу VASYA.LIST (пример 2.29). Для анализа списка и выполнения операций с его элементами целесообразно использовать простые варианты специализированного цикла FOR в формате (где «переменная», обозначающая строку анализируемого одноэлементного списка, может иметь любое имя, например «X»):

FOR /F %переменная IN (файл) DO команда [параметры]
FOR /F %переменная IN (‘команда’) DO команда [параметры]

Используя цикл FOR в формате обработки строк файлов, можно скопировать все текстовые файлы всех подкаталогов директории VASYA в один каталог (см. пример 2.29), а затем объединить текстовые файлы (пример 2.30), применяя перенаправление вывода в файл с добавлением данных: «>>».

Команда MORE активно используется для перенаправления стандартного ввода и построения конвейеров команд в нижеследующих форматах (подробное описание ключей см. п. 2.1.3):

MORE [/E [/C] [/P] [/S] [/Tn] [+n]] < [диск:][путь]имя_файла

имя_команды | MORE [/E [/C] [/P] [/S] [/Tn] [+n]]

Пример 2.30. Применение и цикла и перенаправления для создания файлов

```
d:\WINLAB\vasya\every\answer>for /f %x in ('dir /b') do more %x >> vasya.txt
ВЫДАЧА
d:\WINLAB\vasya\every\answer>more vasya1.txt  1>>vasya.txt
d:\WINLAB\vasya\every\answer>more vasya2.txt  1>>vasya.txt
d:\WINLAB\vasya\every\answer>more vasya3.txt  1>>vasya.txt
d:\WINLAB\vasya\every\answer>more vasya4.txt  1>>vasya.txt

d:\WINLAB\vasya\every\answer>more < vasya.txt

На это каждый ответит, каждый ответит:
-Конечно, Вася, Вася, Вася,
Ну кто его не знает? Yeah, yeah!
Вася, Вася, Вася - стиляга из Москвы.

d:\WINLAB\vasya\every\answer>
```

2.3. ФАЙЛОВЫЙ МЕНЕДЖЕР FAR

2.3.1. Начало работы и внешний вид FAR

Консольное приложение FAR Manager является одним из популярных файловых менеджеров (англ. file manager) для операционных систем семейства Windows (рис. 2.13). FAR предоставляет интерфейс пользователя для работы с каталогами и файлами, позволяя выполнять наиболее частые операции над файлами: создание, удаление, исполнение, просмотр, редактирование, копирование, перемещение (переименование), а также изменение атрибутов и поиск файлов. Имеется еще ряд дополнительных возможностей, не рассматриваемых в настоящем практикуме, включая работу с внешними устройствами, сетевыми дисками и FTP-северами.

Программа FAR Manager унаследовал внешний вид от известного файлового менеджера Norton Commander, самой популярной командной оболочки MS DOS. Это, прежде всего, две равноценные панели для отображения списка файлов и дерева каталогов, стандартная (синяя) расцветка, система клавиатурных команд и т. п. FAR Manager имеет удобный встроенный текстовый редактор, позволяющий создавать и редактировать текстовые файлы – система команд встроенного текстового редактора FAR в основном соответствует редактору Notepad операционных систем Windows.

Рис. 2.13. Внешний вид файлового менеджера FAR Manager

Как приложение MS Windows FAR Manager может быть запущен из меню кнопки «Пуск» или двойным щелком мыши по пиктограмме («иконке») на рабочем столе Windows, имеющей вид синей таблицы (рис. 2.14) и подпись «FAR» или «FAR Manager».

Рис. 2.14. Пиктограмма («иконка») FAR Manager (синяя)

Удобство работы с FAR Manager обусловлено простой функциональной структурой его интерфейса, обеспечивающего мгновенную доступность наиболее часто используемых функций, и доступом к большинству остальных функций в два клика. При отсутствии мыши можно обойтись клавиатурой без особой потери скорости работы (рис. 2.15).

Рис. 2.15. Функциональная структура FAR Manager

Окно запуска FAR – стандартное окно MS Windows для консольных приложений – в самом вверху справа присутствуют стандартные кнопки всех окон – «Свернуть», «Развернуть», «Закрыть», а слева значок приложения и строка заголовка – обращение с которыми такое же, как с окном командной строки (см. п. 2.2.2).

Главное (конфигурационное) меню FAR Manager активируется горячей клавишей «F9» или кликом мыши по заголовкам как левой, так и правой панели менеджера.

Две панели FAR демонстрируют открытые каталоги со списком содержащихся в них подкаталогов и файлов. Панель, в которой в данный момент доступен выбор подкаталогов и файлов, является активной – в ней располагается курсор выбора подкаталогов и файлов. Смену (выбор) активной панели можно осуществить либо мышью, либо клавишей «Tab».

Курсор активной панели управляется мышью или клавишами со «стрелочками» на клавиатуре. Выбор каталога или файла осуществляется путем помещением на него курсора. Подтверждение выбора осуществляется клавишей «Enter» или двойным кликом мыши.

Диалог выбора диска (рис. 2.16), каталоги которого отображаются в левой панели, вызывается сочетанием клавиш «Alt+F1», аналогичным образом, для правой панели – «Alt+F2».

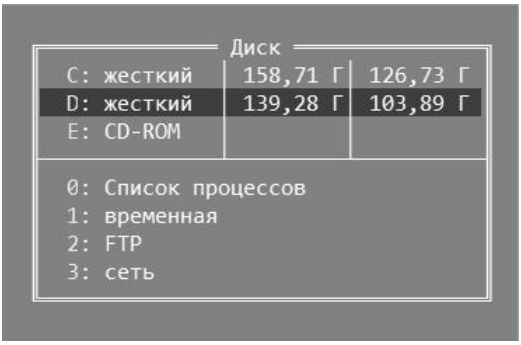


Рис. 2.16. Диалог выбора диска FAR Manager

Подтверждение выбора каталога сопровождается открытием выбранного каталога и появлением на панели FAR его подкаталогов и файлов. Для выхода из каталога в верхней части панели имеется строка, содержащая две последовательные точки – как и в ко-

мандной строке, ими обозначается родительский каталог. Родительского каталога не имеет только корневой каталог диска.

Под панелями каталогов находится полноценная командная строка интерпретатора Cmd.exe, а ниже список горячих клавиш наиболее частых операций (п. 2.3.2). Со списком горячих клавиш FAR Manager можно работать при помощи мыши.

Подтверждение выбора файла приводит к открытию файла в приложении, ассоциированном с расширением файла. Так, файл с расширением .txt будет открыт в редакторе Notepad.

2.3.2. Основные операции FAR Manager

Наиболее актуальные операции FAR Manager (табл. 2.5) перечислены в нижней части окна интерфейса (рис. 2.15) вместе со списком соответствующих «горячих клавиш».

Таблица 2.5. Основные операции и «горячие клавиши» FAR Manager

Клавиша	Функция	Клавиша	Функция
F1	Контекстная помощь	**F7**	Создание каталогов
F2	Меню пользователя	**F8**	Удаление файлов
F3	Просмотр файла	**F9**	Конфигурационное меню
F4	Редактирование файла	**F10**	Выход из FAR Manager
F5	Копирование файлов	**F11**	Команды внешних модулей
F6	Перемещение файлов	**F12**	Список экранов

Горячие клавиши (см. табл. 2.5) инициируют открытие соответствующих диалоговых окон, закрыть которые, отказавшись от выполнения операции, можно нажатием клавиши «Esc» (Escape).

Контекстная помощь «F1». Выполняя любую, еще незнакомую операцию в FAR, нажатием «F1» можно получить справку именно об этой операции.

Меню пользователя «F2». Стандартный список «горячих клавиш» FAR Manager можно дополнить списком собственных операций, поместив их в меню пользователя.

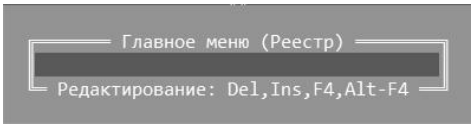


Рис. 2.17. Меню пользователя в FAR Manager

Меню пользователя вызывается по нажатию клавиши «F2». Вначале это меню может быть пустым – вставка нового пункта меню осуществляется клавишей «Insert», «Ins» (рис. 2.17).

Подтверждение вставки (рис. 2.18) инициирует переход в диалоговое окно «Редактирование пользовательского меню» (рис. 2.19).

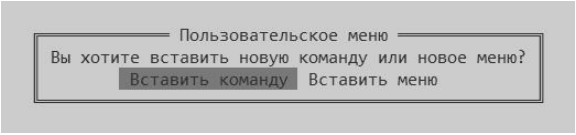


Рис. 2.18. Подтверждение вставки команды в меню пользователя FAR

«Команды» (см. рис. 2.19) – последовательность команд интерпретатора Cmd.exe, т.е. то, что необходимо было бы набрать в командной строке Windows, «Метка» – текст, описывающий команду пользователя, которая может быть запущена выбором в меню или по «горячей клавише».

Рис. 2.19. Редактирование пользовательского меню FAR

В качестве примера создания новой команды в меню пользователя рассмотрен запуск текстового редактора Notepad с «горячей клавишей», «N» (рис. 2.20).

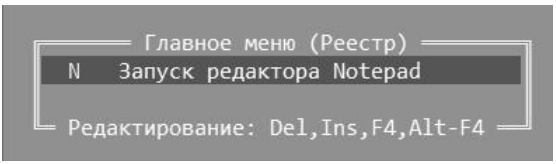


Рис. 2.20. Пользовательское меню FAR Manager

Переход в редактирование пункта меню осуществляется по «F4» (или по «Alt + F4» – в режиме текстового файла), для удаления пунктов меню используется клавиша «Delete» (см. рис. 2.20).

Просмотр файла «F3». Данная функция позволяет просматривать содержание файлов, работает только в отношении файлов и неприменима к каталогам.

Редактирование файла «F4». Выбор файла наведением курсора и нажатие F4 запускает встроенный редактор FAR Manager, в котором файл открывается и может быть отредактирован. Функциональные возможности этого редактора близки к Notepad.

Копирование файлов и каталогов «F5». Это похоже на команду XCOPY с графическим интерфейсом. Копируемые файлы или каталоги должны находиться в активной панели FAR Manager (безразлично, левой или правой), на другой панели должен быть открыт целевой каталог.

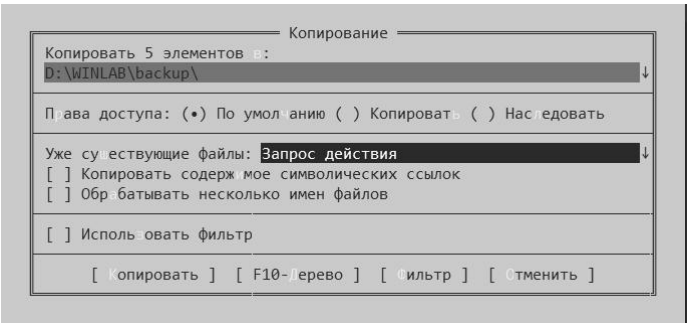


Рис. 2.21. Диалог копирования FAR Manager

На копируемый каталог или файл должен быть наведен курсор активной панели, если копируемых объектов несколько, то каждый

из них можно отметить клавишей «Insert» (при этом объект отмечается желтым цветом) – в итоге образуется группа отмеченных желтым цветом объектов, над которыми может быть выполнена единая операция – в данном случае копирование. Отмена выделения объектов также осуществляется клавишей «Insert».

Выбрав целевую папку для копирования, а затем копируемые объекты, клавишей F5 инициируется диалоговое окно копирования (рис. 2.21). Для простого копирования необходимо выбрать пункт «Копировать», в левом нижнем углу диалога. В результате копируемые объекты отобразятся в целевой папке копирования.

Перемещение файлов и каталогов «F6». Соответствующее диалоговое окно FAR Manager называется «Переименование/перенос». Данная операция осуществляется полностью аналогично копированию, при этом переименованным может быть только один объект – файл или каталог.

Создание каталогов «F7». Для создания нового каталога клавишей «F7» инициируется открытие соответствующего диалогового окна: «Создание папки» (рис. 2.22). В поле «Создать папку» вводится имя нового каталога (папки), который будет создан в активной панели. Выбрав (отметив крестиком) опцию «Обрабатывать несколько имен папок», можно одновременно создать несколько каталогов, перечислив их через запятую или пробел – при этом можно создавать цепочки вложенных каталогов, по аналогии с командами MD и MKDIR командного интерпретатора Cmd.exe.

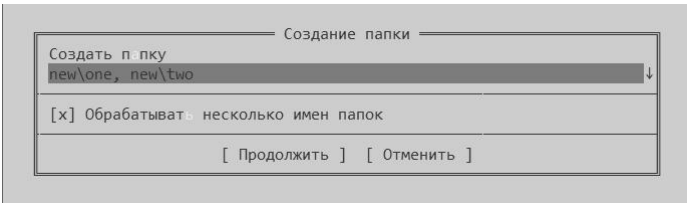


Рис. 2.22. Диалог создания папок FAR Manager

Удаление каталогов и файлов «F8». Один каталог или один файл может быть удален из каталога, открытого в активной панели. Удаляемый объект выбирается курсором, после чего нажимается «горячая клавиша» «F8» и подтверждается (или отменяется) удалением. Можно удалить несколько объектов, предварительно отметив

их клавишей «Insert» – выбранные объекты будут отмечены желтым цветом – и после выбора нажать «F8». С объектов, отмеченных для удаления клавишей «Insert», пометка убирается этой же клавишей.

Конфигурационное меню «F9». Меню конфигурации FAR Manager содержит все операции файлового менеджера (рис. 2.15), включая рассмотренные ранее основные операции, а также дополнительные функциональные возможности. Его подробное рассмотрение, так же как работа с командами внешних модулей («горячая клавиша» «F11») и списком экранов («горячая клавиша» «F12»), не входит в задачу данного лабораторного практикума, но может быть изучено самостоятельно.

Выход из FAR Manager «F10». Для завершения работы FAR Manager необходимо нажать клавишу «F10», а затем подтвердить или отменить завершение работы.

2.3.3. Дополнительные возможности FAR

Файловый менеджер FAR демонстрирует оптимальное сочетание простоты и сложности. Для начинающего пользователя все минимально необходимые функции находится перед глазами (см. п. 2.3.2). Но это лишь «верхушка айсберга». FAR Manager обладает функциональными возможностями, способными удовлетворить самого требовательного пользователя. В реальной практике каждый пользователь останавливается на своем индивидуальном уровне понимания и использования FAR, в дальнейшем открывая для себя новые возможности, о которых даже трудно было догадаться в начале его использования.

В качестве примера можно рассмотреть ассоциации файлов с приложениями. В MS Windows каждое приложение ассоциировано (связано) с определенными типами файлов. Так, текстовый редактор Notepad ассоциирован с текстовыми файлами, имеющими расширение .txt. Это означает, что при выборе и запуске значка файла (двойным кликом мыши или клавишей «Enter») в графическом интерфейсе Windows автоматически запустится текстовый редактор Notepad и откроет указанный файл.

Редактор Notepad также запустится автоматически, если набрать и подтвердить ввод имени файла в командной строке, не указав перед ним никакой команды, утилиты или приложения.

В FAR Manager умолчания Windows сохраняются, но при этом пользователь может создавать свои ассоциации для файлов того или иного типа, а также альтернативные ассоциации для уже существующих ассоциаций Windows.

В качестве эксперимента можно выбрать в FAR любой ранее текстовый файл и нажать «Enter» – этот файл откроется в Notepad. Если теперь вместо «Enter» нажать «Ctrl+PgDown», то не должно произойти ничего, если только раньше за этим компьютером не выполнялась та же лабораторная работа. Комбинация «Ctrl+PgDown» отвечает в FAR Manager за одну из альтернатив запуска приложений, т. е. к этому сочетанию клавиш можно привязать запуск приложения, дополнительного к тем, которые запускаются для файла по нажатию клавиши «Enter» или «F4».

Именно эту задачу и предстоит решить: чтобы при нажатии комбинация «Ctrl+PgDown» в отношении выбранного файла запускалось какое-либо приложение. А поскольку заранее неизвестно, какие приложения могут быть установлены на компьютере во время выполнения лабораторного практикума, но в любой системе Windows всегда есть редактор Notepad, то его и предстоит ассоциировать с этими клавишами.

Для создания или редактирования ассоциаций файлов необходимо активировать конфигурационное меню FAR Manager и выбрать соответствующий пункт:

Команды > Ассоциации файлов

Для создания новой ассоциации в окне ассоциаций файлов (рис. 2.23), для добавления ассоциации используется клавишу «Insert».

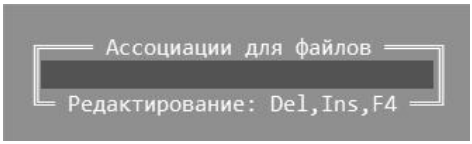


Рис. 2.23. Меню ассоциаций файлов FAR Manager

В открывшемся диалоге «Редактирование ассоциаций файлов» необходимо заполнить соответствующие поля (рис. 2.24).

Маска текстовых файлов как в командной строке Windows, так и в FAR Manager выглядит одинаково: *.txt, для описания ассоциации можно использовать любой осмысленный текст, например: «Альтернативный запуск Notepad».

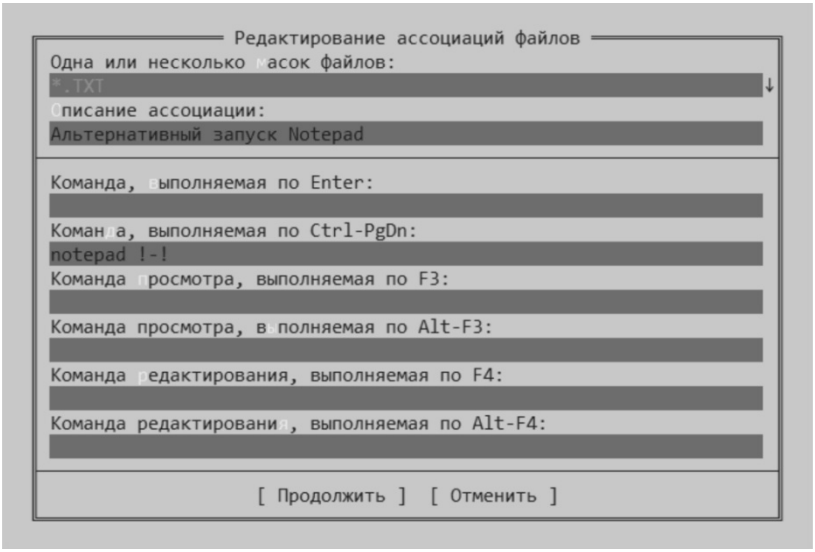


Рис. 2.24. Редактирование ассоциаций файлов FAR Manager

Последним пунктом необходимо указать команду редактирования, выполняемую по «Ctrl+PgDown» – здесь есть определенные сложности, поскольку выражение типа «Notepad *.txt» работать не будет. Необходимо использовать текстовые выражения на основе специальных метасимволов (табл. 2.6) – в данном случае вполне подойдет «!-!» – выражение, обозначающее короткое имя файла с расширением.

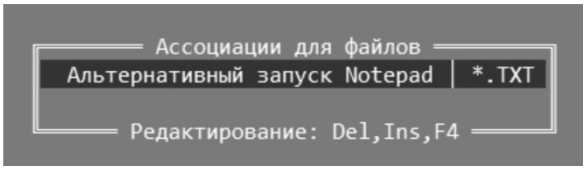


Рис. 2.25. Список ассоциаций файлов FAR Manager

После выбора «Продолжить» в окне редактирования ассоциаций файлов появится окно с обновленным списком ассоциаций файлов (рис. 2.25). Для удаления элемента (строки) ассоциации, связанного с Notepad или другим приложением, предусмотрена клавиша «Delete». Закрыть окно ассоциаций файлов можно клавишей «Escape».

Таблица 2.6. Метасимволы маскирования имен файлов в FAR Manager

Метасимволы	Значение комбинации метасимволов
!!	Символ «!»
!	Длинное имя файла без расширения
!~	Короткое имя файла без расширения
!`	Длинное расширение файла без имени (ext)
!`~	Короткое расширение файла без имени (ext)
!.!	Длинное имя файла с расширением
!-!	Короткое имя файла с расширением
!&	Список помеченных файлов
!&	Список помеченных файлов с короткими именами
!:	Текущий диск в формате C:, D: и т.д.
!\	Текущий путь
!/	Короткое имя текущего пути

ЗАЧЕТНЫЕ ЗАДАНИЯ К РАЗД. 2

Задание 2.1

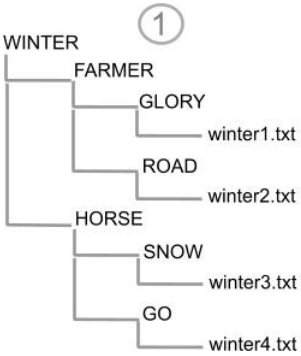
В своей домашней директории построить два дерева каталогов A и B (по аналогии с рис. 2.5 – без файлов!). При построении дерева A использовать п. 2.1.2 (см. примеры 2.5 ÷ 2.17).

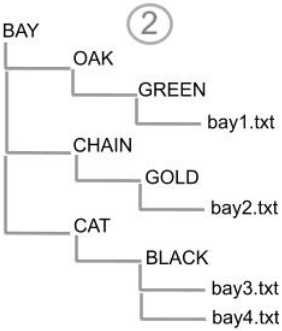
При построении дерева B освоить работу с буфером хронологии команд и другими полезными инструментами командной строки (из п. 2.2.1).

Варианты задания

Вариант 1:	1,2.	**Вариант 2:**	3,4.	**Вариант 3:**	2,3.
Вариант 4:	1,3.	**Вариант 5:**	3,5.	**Вариант 6:**	2,4.
Вариант 7:	1,4.	**Вариант 8:**	3,6.	**Вариант 9:**	2,5.
Вариант 10:	1,5.	**Вариант 11:**	4,5.	**Вариант 12:**	2,6.
Вариант 13:	1,6.	**Вариант 14:**	4,6.	**Вариант 15:**	5,6.

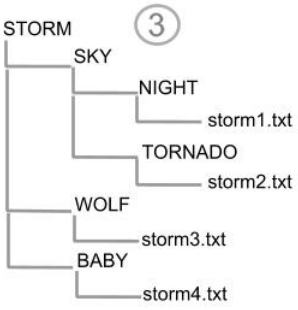
Деревья каталогов в вариантах заданий

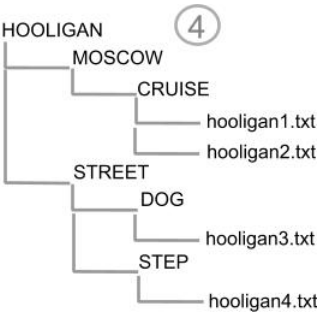




winter1.txt: Зима!.. Крестьянин, торжествуя,
winter2.txt: На дровнях обновляет путь;
winter3.txt: Его лошадка, снег почуя,
winter4.txt: Плетется рысью как-нибудь.

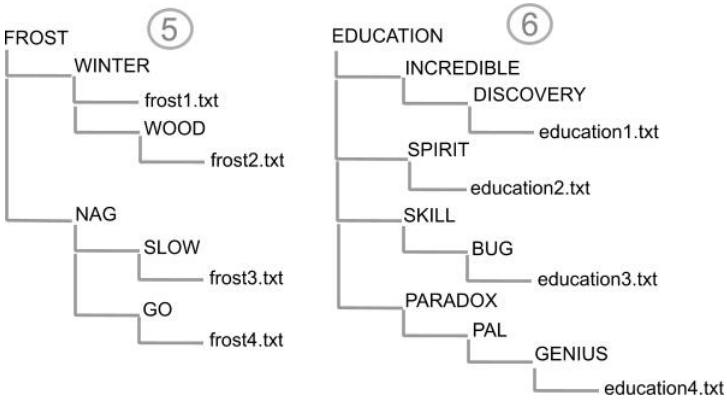
bay1.txt: У лукоморья дуб зеленый;
bay2.txt: Златая цепь на дубе том:
bay3.txt: И днем и ночью кот ученый
bay4.txt: Всё ходит по цепи кругом.





storm1.txt: Буря мглою небо кроет,
storm2.txt: Вихри снежные крутя;
storm3.txt: То, как зверь, она завоет,
storm4.txt: То заплачет, как дитя.

hooligan1.txt: Я московский озорной гуляка.
hooligan2.txt: По всему тверскому околотку
hooligan3.txt: В переулках каждая собака
hooligan4.txt: Знает мою легкую походку.



frost1.txt: Однажды, в студеную зимнюю пору,
frost2.txt: Я из лесу вышел; был сильный мороз.
frost3.txt: Гляжу, поднимается медленно в гору
frost4.txt: Лошадка, везущая хворосту воз.

education1.txt: О, сколько нам открытий чудных
education2.txt: Готовит просвещенья дух
education3.txt: И опыт, сын ошибок трудных,
education4.txt: И гений, парадоксов друг.

Задание 2.2

В структуре каталогов A создать текстовые файлы, используя редактор Notepad. В структуре каталогов B создать текстовые файлы, используя инструменты перенаправления стандартного ввода/вывода (см. примеры 2.29 ÷ 2.30). Добиться качественного отображения файлов при использовании инструментов просмотра файлов в командной строке (см. примеры 2.18 ÷ 2.20), с учетом выбора правильной кодовой страницы (см. пример 2.28) и true type-шрифтов отображения текста в командной строке Windows (вкладка «Шрифт» окна свойств командной строки из п. 2.2.2).

Варианты задания

Соответствуют вариантам задания 1.

Задание 2.3

Скопировать все текстовые файлы структуры каталогов A в папку, в которой находится файл с порядковым номером №1, используя обычные способы копирования (см. примеры 1.21 ÷ 1.23).

Скопировать все текстовые файлы структуры каталогов B в папку, в которой находится файл с порядковым номером №1, используя обычные способы копирования (см. примеры 2.31 ÷ 2.32).

Объединить занумерованные одноименные файлы структуры каталогов A в файл с тем же именем, но без номера (см. пример 2.33).

Аналогично поступить с файлами структуры каталогов B.

Варианты задания

Соответствуют вариантам задания 1.

Задание 2.4

С помощью файлового менеджера FAR Manager построить структуру каталогов и файлов (см. номер варианта).

Варианты задания

Вариант 1:	3.	**Вариант 2:**	5.	**Вариант 3:**	1.
Вариант 4:	2.	**Вариант 5:**	4.	**Вариант 6:**	3.
Вариант 7:	5.	**Вариант 8:**	2.	**Вариант 9:**	4.
Вариант 10:	6.	**Вариант 11:**	6.	**Вариант 12:**	1.
Вариант 13:	4.	**Вариант 14:**	1.	**Вариант 15:**	2.

Задание 2.5

Создать пользовательское меню FAR Manager по аналогии с примером из п. 2.3.2 или по отдельному заданию преподавателя. Создать ассоциацию текстовых файлов по аналогии п. 2.3.3 или по отдельному заданию преподавателя.

3. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ FORTRAN

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К РАЗД. 3

В настоящее время существует большое количество программного обеспечения для научных и инженерных расчетов (математические пакеты, САПР и т.д.), содержащих готовые алгоритмы решения стандартных задач численных методов.

Для эффективного использования популярных математических пакетов прикладных программ и программных библиотек, реализующих численные методы и алгоритмы на различных языках программирования, необходимо общее представление и навыки программирования простейших вычислительных задач.

Традиционно изучение основ программирования на императивных языках связано с рассмотрением структуры программных единиц (главной программы и подпрограмм), основных и производных типов данных и реализации стандартных алгоритмов, использующих условные операторы, циклы, массивы и другие структуры данных. В современном Фортране присутствуют достаточно простые языковых конструкции – модули, позволяющие связать воедино структуры данных, производные от основных типов с функциями, позволяющими реализовать полезные преобразования с такими структурами. В связи с этим имеет смысл рассмотреть элементы объектно-ориентированного программирования.

Очевидно, что теоретическое изучение программирования не эффективно без практических упражнений, поэтому в данном практикуме содержится около 100 коротких отлаженных примеров, работа с которыми позволит получить практические навыки и наглядное представление об основных языковых конструкциях. Все примеры, представленные в учебном пособии, строго соответствуют стандарту Fortran-95 [ISO/IEC 1539-1:1997(E)].

При начальном обучении программированию весьма целесообразным представляется использование наиболее простого транслятора, запускаемого из командной строки и не отягощенного разного рода графическими средами и сервисами. Именно таким транслятором является компилятор Gfortran. Изначально этот компилятор разработан для GNU/Linux (http://gcc.gnu.org/fortran/), но его можно свободно скачать и установить под Windows, в составе па-

кета MinGW (Minimalist GNU for Windows) – с официального сайта MinGW: http://www.mingw.org. Эта информация может оказаться актуальной, для внеаудиторного выполнения заданий.

Принципиальным моментом при изучении основ программирования является понимание важности следования стандарту языка при написании текста программы и его переносимости. Конкретный программный продукт, например Fortran Power Station (FPS), поддерживает стандарт [ISO/IEC 1539-1:1997(E)], так же как и Gfortran. Однако тексты программ, написанные и отлаженные на FPS, далеко не всегда будут правильно компилироваться другими компиляторами Фортрана, ориентированными на тот же стандарт. Это весьма возможно, если в тексте программы будет использован тип данных или языковая конструкция, не стандартная в рамках [ISO/IEC 1539-1:1997(E)], но присутствующая в FPS как опция, дополняющая возможности стандарта. Это вовсе не означает, что нестандартными возможностями какого-либо транслятора или среды разработки не следует пользоваться. Нужно всего лишь уметь четко оценивать текст написанной программы на предмет соответствия стандарту языка программирования и делать правильные выводы относительно переносимости данной программы.

При этом представляется весьма важным формирование правильного представления о взаимосвязи стандарта языка программирования и программном продукте, реализующем этот стандарт через транслятор и дополняющий его графический интерфейс пользователя или среду разработки. Зачастую, изучая программирование в различных средах, студенты либо вообще остаются в неведении относительно того, что такое транслятор, либо имеют о трансляторе довольно смутные представления, поскольку взаимодействуют с ним опосредованно, через систему многоуровневого меню.

В целом ряде случаев – при описании свободного и фиксированного формата записи программы (пп. 3.1.3, 3.1.4), а также представления буквальных текстовых констант, весьма важно обозначить символы «пробел» в текстах примеров и пояснений. Для их обозначения в данном учебном пособии используется символ «^» («крышечка»), как, например, в буквальной текстовой константе «HELLO^WORLD». Символ – «^» – в данном практикуме не используется больше не для чего, чтобы не вызывать разночтений.

3.1. ПРАВИЛА ЗАПИСИ ПРОГРАММЫ

3.1.1. Набор символов Фортрана

При написании программ на Фортране используется строго определенный набор символов или алфавит. Он состоит из 26 букв латинского алфавита (от «A» до «Z»), 10 арабских цифр (от 0 до 9) и набора специальных символов, представленных в табл. 3.1.

Буквы латинского алфавита могут быть как прописными, так и строчными (т.е. «большими» и «маленькими»). По существу, это означает две группы символов: от «A» до «Z» или от «a» до «z».

В Фортране нет различия между строчными и прописными буквами, за исключением работы с текстовыми данными. Так два текстовых сообщения “HELLO, WORLD” и “hello, world” различаются с точки зрения их символьного представления.

Таблица 3.1. Специальные символы Фортрана

Символ	Название	Символ	Название
=	Знак равенства	:	Двоеточие
+	Знак плюс		Пробел
–	Знак минус	!	Восклицательный знак
*	Звездочка	″	Кавычки
/	Слеш	%	Процент
(	Левая скобка	&	Амперсанд
)	Правая скобка	;	Точка с запятой
,	Запятая	<	Меньше
.	Десятичная точка	>	Больше
$	Денежный знак	?	Вопросительный знак
′	Апостроф	_	Символ подчеркивания

Символ «пробел» в табл. 3.1 не обозначен ни каким видимым символом. Его необходимо каким-то образом визуализировать, поскольку в ряде случаев бывает важно показать количество пробелов. Во многих руководствах и учебниках по Фортрану для этого используется символ «^» («крышечка»). Этот символ не входит в алфавит Фортрана, поэтому не вызывает путаницы. Например, можно наглядно показать различие количества пробелов в двух текстах: «HELLO^WORLD» и «HELLO^^WORLD».

3.1.2. Форматы записи программы

Способ записи программ на Фортране изначально был ориентирован на ввод данных с перфокарт (рис. 3.1), содержащих в каждой строке 80 позиций для перфорирования. При этом 72 позиции использовалось собственно для записи программы и комментариев, а позиции с 73 по 80 как служебные, например для управления обработкой перфокарт.

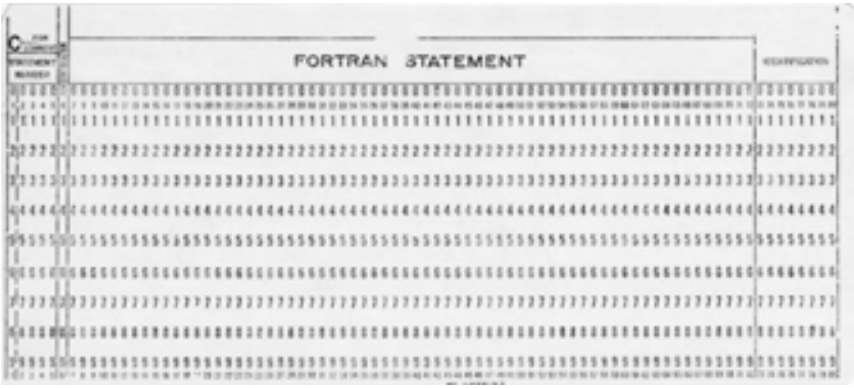


Рис. 3.1. Перфокарта с разметкой колонок для Фортрана

Формат записи текстов программ на Фортране сохранился и при переходе на текстовые и графические терминалы (пример 3.1). Современные терминалы и текстовые редакторы продолжают поддерживать текстовый режим отображения 80-ти символьных строк, обеспечивая программно-аппаратную совместимость вычислительной техники различных стандартов.

Пример 3.1. Простейшая программа в фиксированном формате

```
^^^^^^program HELLO
^^^^^^print*, "Hello, World"
^^^^^^end
```

Запись программ в виде строк ограниченной длины связана именно с режимами отображения символьной информации текстовыми терминалами. Такой подход гарантирует, что программа на Фортране может быть написана с использованием самого прими-

тивного текстового редактора или без него – важно лишь наличие двух символов: «конец строки» (или «перевод каретки»), а также «конец файла», которые генерируются клавиатурами любых модификаций. Так, символ «конец строки» генерируется нажатием клавиши ввода, например «Enter». В текстовых редакторах этот символ, как правило, не виден, но его ввод переводит текстовый курсор на следующую строчку. Символ конца файла, как правило, генерируется сочетанием клавиш Ctrl+Z (например, в UNIX).

Формат записи программ, ориентированный на 72-х символьное отображение текста, был единственным вплоть до стандарта Фортран 90 и поэтому не имел никакого названия. В стандарте Фортран 90 и последующих он получил название «*фиксированный формат*», поскольку появился новый «*свободный формат*», ориентированный на 132-х символьные текстовые строки (см. пример 3.2), что связано с появлением соответствующих стандартов отображения символьной информации текстовыми терминалами (режим отображения текстовых терминалов «132 символа в строке»).

Пример 3.2. Простейшая программа в свободном формате

```
program HELLO
print*, "Hello, World"
end
```

Фиксированный формат записи программ поддерживается всеми современными компиляторами для обеспечения совместимости со стандартом FORTRAN 77 (Стандарты Fortran 90/95 декларируют FORTRAN 77 как свое неотъемлемое подмножество).

Используемый в данном практикуме компилятор Gfortran по умолчанию обрабатывает исходные файлы, имеющие расширение .f95, как тексты программ в свободном формате, а файлы с расширением .for как программы в фиксированном формате.

При написании программы в фиксированном формате нужно обращать внимание, каким образом используемый текстовый редактор нумерует символьные позиции строки. Например, редактор mcedit (утилита файлового менеджера UNIX MidNight Cjmmander) нумерует символьные позиции строки начиная с нуля, вследствие чего шестая символьная позиция строки (по нумерации редактора) на самом деле оказывается седьмой.

3.1.3. Фиксированный формат

Фиксированный формат записи программы на Фортране определяется следующими правилами.

1. Каждая строка программы рассматривается как последовательность символьных позиций с 1-й по 72-ю. Символы после 72 позиции строки имеют служебные функции по отношению к операционной системе и не обрабатываются компилятором. При этом операторы программы записываются в позициях с 7-й по 72-ю (пример 3.3) по одному оператору в строке.

Пример 3.3. Нумерация текстовых полей в фиксированном формате

```
1^^^^^7^^^^^^^^^^^^^^НУМЕРАЦИЯ ПОЗИЦИЙ СТРОКИ^^72^^^^^^^^^^^^^^^^80

^^^^^^PROGRAM HELLO
^^^^^^PRINT*, 'HELLO, WORLD'
^^^^^^END
```

Для новичков важно отметить, что фраза 'HELLO, WORLD' заключена в одиночные апострофы. На стандартной клавиатуре одиночный апостроф расположен на одной клавише с кавычками (на стандартной русифицированной клавиатуре там же обычно находится буква «Э»). Не стоит искать одиночный апостроф на одной клавише с «волной» (символ «~») и буквой «Ё» в левом верхнем углу клавиатуры.

2. Пробелы не значимы, за исключением работы с текстовыми данными (например, текст “HELLO^WORLD” – пример 3.4).

Пример 3.4. Запись программы без пробелов

```
1^^^^^7^^^^^^^^^^^^^^НУМЕРАЦИЯ ПОЗИЦИЙ СТРОКИ^^72^^^^^^^^^^^^^^^^80

^^^^^^PROGRAMHELLO
^^^^^^PRINT*,'HELLO,^WORLD'
^^^^^^END
```

Пробелы служат только для улучшения читаемости текста человеком и полностью игнорируются при компиляции программы. С точки зрения логики компилятора Фортрана программы нижеследующих примеров (пример 3.5, 3.6) полностью эквива-

лентны как между собой, так и с программами из всех ранее приведенных примеров.

Пример 3.5. Запись программы с произвольной расстановкой пробелов

```
1^^^^^7^^^^^^^^^^^^^^НУМЕРАЦИЯ ПОЗИЦИЙ СТРОКИ^^72^^^^^^^^^^^^^^^80

^^^^^^PROG^^RAMHEL^LO
^^^^^^P^R^I^N^T*,'HELLO,^WORLD'
^^^^^^EN^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^D
```

3. Программа на Фортране может содержать комментарии – произвольный пояснительный текст. Латинская буква «С» (строчная или прописная), а также «*» (звездочка) в первой позиции строки маркируют всю строку как строку комментария (см. пример 3.6). Соответственно, компилятор, найдя такую строку, просто не обрабатывает ее. Комментарий может быть записан уже со второго символа и располагается в любом месте текста программы между операторами.

Типичной ошибкой, связанной с написанием комментариев, является использование русской «Эс» вместо латинской «Си». Ситуация объясняется (и усугубляется) тем, что совпадает не только изображение символов, но также их расположение на клавиатуре. Оба символа «Эс» и «Си» располагаются на одной клавише – на это следует обратить особое внимание.

Пример 3.6. Текст программы с комментариями

```
1^^^^^7^^^^^^^^^^^^^^НУМЕРАЦИЯ ПОЗИЦИЙ СТРОКИ^^72^^^^^^^^^^^^^^^80

C  ЭТО ТЕКСТ ПРОГРАММЫ
c   HELLO
*  С КОММЕНТАРИЕМ
^^^^^^PROGRAM HELLO
^^^^^^PRINT*, 'HELLO, WORLD'
^^^^^^END
```

4. Оператор Фортрана может быть достаточно длинным и не уместиться с 7 по 72 позицию, поэтому необходим механизм для продолжения оператора на следующей строке. Для этого в 6-й позиции следующей строки печатается любой символ из алфавита Фортрана, отличный от нуля или пробела, например единица, двойка и т.д. (пример 3.7).

Пример 3.7. Запись программы с продолжениями строки

```
1^^^^67^^^^^^^^^^^^^^^НУМЕРАЦИЯ ПОЗИЦИЙ СТРОКИ^^72^^^^^^^^^^^^^^^^80

^^^^^^PROGRAM
^^^^^1^^^^^^^^^^HEL
^^^^^2^^^^^^^^^^LO
^^^^^^PRINT*, 'HELLO, WORLD'
^^^^^^END
```

Всего, в соответствии со стандартом, допускается до 19-ти строк продолжения одного оператора.

5. Позиции строки с 1-й по 5-ю используются для записи меток, представляющих собой целые числа от 0 или 00000 до 99999. Начальные нули меток не являются значащими (пример 3.8).

Пример 3.8. Запись программы с использованием меток

```
1^^^5^7^^^^^^^^^^^^^^^НУМЕРАЦИЯ ПОЗИЦИЙ СТРОКИ^^72^^^^^^^^^^^^^^^^80

^^^^^^PROGRAM HELLO
^^^^^^PRINT 10
^^010^FORMAT ('HELLO, WORLD')
^^^^^^END
```

3.1.4. Свободный формат

Свободный формат записи программы на Фортране рекомендуется использовать в стандарте Fortran 90 и последующих стандартах. Это формат записи программ определяется следующими правилами.

1. Программа записывается построчно, строка может содержать до 132 символов, запись оператора может начинаться с любой позиции строки (с 1-й по 132-ю). В отличие от фиксированного формата это избавляет от необходимости отслеживать шестисимвольный отступ от начала строки (пример 3.9).

Пример 3.9. Запись программы в свободном формате

```
program HELLO
print*, 'HELLO,^WORLD'
end
```

2. При большой длине оператора, он может быть разделен на несколько строк. Строка оператора считается продолжаемой в следующей строке, если она заканчивается символом «&» (амперсанд) (пример 3.10). Стандарт Фортран 90/95 допускает не более 39 строк продолжения одного оператора.

Пример 3.10. Запись программы с построчным разделением операторов

```
program &
HELLO
print&
*&
, 'HELLO,^WORLD'
end
```

3. В одной строке может быть записано несколько операторов, разделенных «;» (точкой с запятой). Например, программа HELLO может быть записана в одну строчку (пример 3.11):

Пример 3.11. Запись программы в одну строчку

```
program^HELLO; print*,'HELLO, WORLD'; end
```

4. Если в строке программы присутствует символ «!» (восклицательный знак), то весь текст после него, вплоть до конца строки, считается комментарием и не обрабатывается компилятором, Исключение составляют восклицательные знаки, являющиеся составной частью текстовых данных (пример 3.12).

Пример 3.12. Запись комментариев в свободном формате

```
! Программа HELLO с комментариями
program HELLO
print*, 'HELLO!^WORLD!!!'     ! Восклицательные знаки в приветствии
end                           ! к комментарию не относятся!
```

5. Метки операторов, в отличие от фиксированного формата, не привязаны к позициям строки с 1-ю по 5-ю и, вообще, ни к каким позициям (пример 3.13).

Пример 3.13. Использование меток в свободном формате

```
program HELLO
010 format ('HELLO,^WORLD')
print 10
end
```

Если непосредственно перед оператором встречается целое число от 0 до 99999, отделенное от оператора одним или несколькими пробелами, то это число считается меткой оператора. Начальные нули метки не являются значащими.

Поскольку в свободном формате метка не связана с символьными позициями строки, то программа с метками может быть записана в одной строке (пример 3.14).

Пример 3.14. Запись программы с метками в одну строчку

```
program HELLO; print 10; 10 format ('HELLO,^WORLD'); end
```

6. В связи с тем, что в программах, записанных в фиксированном формате, пробелы могут использоваться практически произвольно, возникает вопрос о значимости пробелов при записи программы в свободном формате.

При записи программы в свободном формате пробелы не должны нарушать целостность имен или ключевых слов. Например, недопустимо написание HE^LLO (для имени программы) или P^^RI^NT (в фиксированном формате записи программы это допускалось, см. пример 3.5). В остальном пробелы, как и в фиксированном формате, используются для улучшения читаемости текста.

3.2. ТРАНСЛЯЦИЯ ПРОГРАММЫ

3.2.1. Программа в одном исходном файле

Наиболее простая программа на Фортране – единственная (основная или главная) программа, записанная в одном файле, с расширением .f95.

При работе в операционной системе GNU/Linux для набора текста программы можно использовать такие текстовые редакторы, как mcedit, pico или Vi. При работе в Microsoft Windows вполне подойдет редактор Notepad или встроенный редактор FAR Manager.

Для того чтобы преобразовать файл, содержащий текст програмы, в исполняемый файл, необходим транслятор. При выполнении данного компьютерного практикума по Фортрану в качестве транслятора рекомендуется использовать компилятор Gfortran.

Работая в GNU/Linux необходимо удостовериться в его наличии в составе GNU Compiler Collection (GCC). Если же используется Windows, то необходимо скачать и установить MinGW (Minimalist GNU for Windows) – программный порт GCC (с официального сайта: http://www.mingw.org).

При создании файла желательно называть файл именем главной программы с обязательным расширением .f95 (для программы, написанной в свободном формате Фортран 90, 95).

Для примера рассмотрим создание исполнимого файла из исходного файла hello.f95 (пример 3.15).

Пример 3.15. Исходник hello.f95 для трансляции

```
program HELLO
print*, "HELLO,^WORLD"
end
```

При этом важны следующие моменты.

1. Программа начинается с первой позиции первой строки.
2. Строку «Пример 3.15. Исходник hello.f95 для трансляции» в файл писать не нужно.
3. Программа начинается с оператора PROGRAM с указанием имени программы (в данном случае HELLO).
4. Крышечкой «^» обозначаются пробелы.
5. Любая программа или подпрограмма на Фортране в обязательном порядке должна завершаться оператором END.
6. В файле не должно быть ничего кроме текста программы.
7. Проверьте имя файла: hello.f95, и никак иначе.

3.2.2. Трансляция исходного файла

Файл hello.f95 с текстом программы (см. пример 3.15) необходимо транслировать в исполнимый файл, который можно будет запустить на исполнение из командной строки Linux или Windows и увидеть на экране: текстовое сообщение: HELLO, WORLD.

Создание исполняемого файла с именем по умолчанию. Наиболее простой (хотя и не самый лучший) вариант – запустить в командной строке команду gfortran, указав через пробел имя компилируемого файла:

gfortran hello.f95

В этом случае, если исходный файл hello.f95 не содержит ошибок, в текущей директории будет создан исполнимый файл с именем: a.out – если используется какой-либо коммандер, то это сразу будет видно в таблице; иначе придется воспользоваться командой операционной системы:

ls (для Linux)

dir (для Windows)

Для запуска файла из командной строки Linux необходимо явное указание на текущую директорию: – «./». При запуске из командной строки Windows в этом нет необходимости:

./a.out (для Linux)

a (для Windows)

После запуска, на экране появится текст:

HELLO, WORLD

Если в процессе компиляции обнаружится наличие ошибок в тексте программы, то необходимо их исправить, а затем повторить процесс компиляции.

Создание исполняемого файла с заданным именем. Наиболее часто возникает ситуация, когда исполняемому файлу программы требуется присвоить такое же имя, как и у исходного файла. Для этого у компилятора Gfortran имеется специальная опция «–o *имя*», важно не перепутать латинскую букву «o» с нулем! Эту опцию можно указать в любом месте списка параметров компилятора Gfortran, например:

gfortran -o hello hello.f95 или gfortran hello.f95 -o hello

Правильное выполнение этой команды приведет к созданию исполнимого файла с именем hello (или hello.exe), который затем можно запустить обычным образом:

./hello (для Linux)

hello или hello.exe (для Windows)

Типичной ошибкой, на которую стоит обратить внимание, является попытка запустить исходный файл вместо исполняемого файла, т.е. в командной строке набирают ./hello.f95 вместо ./hello и т.д.

Пример 3.16. Программные компоненты в одном файле

```
program MESSAGE            ! В файле hello.f95
  call HELLO
end
subroutine HELLO!          ! В файле hello.f95
  print*, 'HELLO,^WORLD'
end
```

Исходный файл может содержать несколько программных компонент – одну основную программу подпрограмму или несколько. Основная программа MESSAGE (пример 3.16) вызывает подпрограмму HELLO, печатающую уже знакомое текстовое сообщение «HELLO, WORLD». Для вызова подпрограммы используется оператор CALL (см. п. 3.7).

Поскольку все программные компоненты находятся в одном файле, то в написании и трансляции программы ничего не изменяется – создается исходный файл, например: message.f95, в котором размещается главная программа MESSAGE и подпрограмма HELLO, а затем исходный файл компилируется точно так же, как исходный файл hello.f95 в предыдущих примерах:

gfortran -o message message.f95

или же

gfortran message.f95 -o message

3.2.3. Трансляция нескольких исходных файлов

Программные компоненты одной программы могут размещаться отдельно в нескольких исходных файлах. Основная программа

MESSAGE может размещаться в файле message.f95 (пример 3.17), а подпрограмма HELLO в файле hello.f95 (пример 3.18).

Пример 3.17. Файл message.f95

```
program MESSAGE          ! В файле hello.f95
  call HELLO
end
```

Пример 3.18. Файл hello.f95

```
subroutine HELLO         ! В файле hello.f95
  print*, 'HELLO,^WORLD'
end
```

Компиляция с созданием исполняемого файла. При трансляции имена исходных файлов перечисляются в произвольном порядке, а опция «–o massage» определяет имя исполняемого файла:

```
gfortran -o massage hello.f95 message.f95
```

или

```
gfortran  message.f95 hello.f95  -o massage
```

Напомним, что в принципе исполняемый файл может называться как угодно, но во избежание путаницы его, как правило, именуют так же, как и главную программу.

Компиляция с созданием объектных файлов. В ряде случаев интерес может представлять компиляция исходных файлов с их преобразованием в машинный код, без сборки исполняемого файла. Для этого компилятор gfortran необходимо запустить с опцией «–c», т.е. только компиляция. Файлы могут компилироваться по одному или по нескольку:

```
gfortran -c hello.f95
```

```
gfortran -c message.f95
```

или

```
gfortran -c hello.f95 message.f95
```

В любом случае результатом будет появление в текущей директории объектных файлов: hello.o и message.o, из которых затем можно осуществить сборку исполняемого файла так же, как из исходников:

```
gfortran -o massage hello.o message.o
```

Исходные файлы всегда сначала компилируются в объектные файлы, а уже затем из объектных файлов специальной программой, называемой редактором связей или линкером (линковщиком), осуществляется сборка исполняемого файла. Транслятор gfortran просто скрывает этот этап (если текста программ в исходных файлах достаточно для полного цикла генерации исполняемого файла).

При работе над большими проектами компиляция занимает значительное время, из-за чего постоянно перекомпилировать исходные файлы с уже отлаженным кодом нет никакого смысла, поэтому для сборки проекта удобнее пользоваться уже скомпилированными объектными файлами.

Компилятор gfortran поддерживает смешанную обработку исходных и объектных файлов, которые могут быть перечислены через пробел в любом порядке:

```
gfortran -o message hello.o message.f95
```

или

```
gfortran -o message hello.f95 message.o
```

Для получения более полных сведений о возможностях транслятора Gfortran следует воспользоваться опцией «--help»:

```
gfortran --help
```

3.2.4. Трансляция модулей

Исходный файл, содержащий модуль (пример 3.19), необходимо компилировать с опцией «-c» (только компиляция):

```
gfortran -c vector_arithmetic.f95
```

Успешное выполнение этой команды приведет к появлению в текущей директории специализированного файла структуры модуля (vector_arithmetic.mod) и объектного файла (vector_arithmetic.o), который затем необходимо транслировать вместе с использующей его программой (см. пример 3.19) по уже известным правилам трансляции программы, расположенной в нескольких файлах:

```
gfortran -o vector_test vector_test.f95 vector_arithmetic.o
```

или

```
gfortran -o vector_test vector_arithmetic.o vector_test.f95
```

Пример 3.19. Модуль – контейнер векторной арифметики

```
module VECTOR_ARITHMETIC
type VECTOR
real X, Y
end type VECTOR
interface operator(+)
module procedure  ADD_VECTORS
end interface
contains
function ADD_VECTORS(A,B)
type(VECTOR) ADD_VECTORS
type(VECTOR), intent(in)  :: A, B
ADD_VECTORS%X = A%X +B%X
ADD_VECTORS%Y = A%Y +B%Y
end function ADD_VECTORS
end module VECTOR_ARITHMETIC
```

Возможна трансляция с использованием только исходных файлов, но при этом важна очередность их следования – первым должен быть указан файл, содержащий модуль, и только затем указывается исходный файл программы, использующей это модуль:

gfortran -o vector_test vector_arithmetic.f95 vector_test.f95

Дело в том, что компилятор обрабатывает файлы последовательно и для успешной компиляции файла vector_test.f95, использующего модуль, требуется наличие файла vector_arithmetic.mod, а он появляется только после обработки файла vector_arithmetic.f95. Поэтому если переставить файлы:

gfortran -o vector_test vector_test.f95 vector_arithmetic.f95

то возникает ошибка компилятора, связанная с невозможностью открыть файл модуля: vector_arithmetic.mod.

3.3. КОНЦЕПЦИЯ ДАННЫХ ЯЗЫКА ФОРТРАН

3.3.1. Имена (идентификаторы)

Императивные языки программирования, к числу которых относится Фортран (а также Паскаль, Си, Бейсик и др.), в различной степени являются абстракцией компьютерной архитектуры фон Неймана. Две основные компоненты этой архитектуры − *память* и *процессор*. Ячейки памяти служат для хранения данных и команд, а процессор – для изменения состояния памяти.

Простейшими абстракциями ячеек памяти компьютера в императивных языках программирования являются переменные и константы. По существу, переменные и константы – ячейки машинной памяти (машинными словами измеряемыми, как правило, в байтах), адресация которых в программе осуществляется не по физическому адресу, а через *имя* (*идентификатор*). При этом для константы существует ограничение, связанное с запретом изменения ее значения. Адресация таких более сложных конструкций, как массивы и структуры данных (по сути состоящих из переменных и констант), также осуществляется через их имена. Имеют свои названия типы данных и программные компоненты (главная программа подпрограммы и модули).

Имя объекта – это выстроенная по определенным правилам последовательность символов алфавита языка. В Фортране 90/95 действуют следующие правила записи имен.

1. Строчные и прописные (большие и маленькие) буквы в Фортране не различаются.

2. Символьная длина имени не должна превышать 31 символ.

3. Имя может состоять из алфавитно-цифровых символов (букв и цифр) и символа подчеркивания – «_».

4. Первым символом имени обязательно должна быть буква.

Допустимые имена:

X ALFA Beta2S
This_Is_Very_Long_Name (допустимо в Фортране 90/95)

Недопустимые имена:

2SBETA (первый символ не буква);

AL&FA (содержит недопустимый символ: «&»);

IT IS Impossible (содержит пробелы).

В языке Фортран 77, который является подмножеством языка Фортран 90/95, не допускаются имена длиннее шести алфавитно-цифровых (A-Z, a-z, 0-9) символов (никакие подчеркивания и иные специальные символы из алфавита Фортрана не допускаются). При этом первым символом обязана быть буква.

Для имен (идентификаторов) и связанных с ними объектами данных в Фортране существует правило неявного определения типов (п. 3.3.2). Если в программе используется именованный объект данных (скаляр или массив – пп. 3.3.5, 3.3.6), тип которого не определен явно, то объект данных считается:

1) целым (INTEGER), стандартной разновидности целого типа (п. 3.3.4), если первым символом его имени будет одна из букв: 'I', 'J', 'K','L','M' или 'N', например: J, IND, LSTREAM и т.д.;

2) вещественным (REAL), стандартной разновидности вещественного типа (п. 3.3.4), если первым символом его имени будет любая другая буква, например: X, DIN, EPSILON.

Умолчаний по неявному определению других типов в Фортране не существует.

Часто считают, что возможность неявного определения типов переменных и констант в Фортране является избыточным, бесполезным и даже вредным наследием ранних стандартов. Действительно, отсутствие строгого контроля типов может приводить к досадным ошибкам, например опечатка в имени переменной расценивается компилятором как появление в программе переменной с другим именем. Однако не следует забывать, что Фортран создавался как язык для научных расчетов – транслятор формул, позволяющий быстро и оперативно выполнять небольшие оценочные расчеты, без необходимости предварять их громоздкими описательными блоками. Такая возможность может существовать только при отсутствии строгого контроля типов.

Строгий контроль типов необходим при разработке конечного программного продукта, но излишне загромождает программу, и тормозит работу при выполнении небольших оценочных и предварительных расчетов. Неявное определение типов легко отключить при помощи инструкции IMPLICIT NONE (пример 3.20).

Пример 3.20. Инициализация именованных констант с запретом неявного определения типа

```
program CONSTINIT5
implicit none                    ! Запрет неявного определения типа
real (kind=8) PI, E
integer MIN, MAX
parameter  (PI= 3.1415926535897931_8, E= 2.7182818284590451_8)
parameter  (MIN = 0, MAX = 100)
print*, PI, E, MIN, MAX
end
```

3.3.2. Понятие типа

Фортран, как императивный язык программирования, имеет дело с числовыми символьными и логическими данными, т.е. с данными различных *типов*.

Тип данных в императивных языках определяется:

1) правилами записи (символьного представления) значений;
2) диапазоном допустимых значений;
3) набором действий и операций над данными этого типа.

Например, значения целого типа (INTEGER) имеют привычное символьное представление в виде целых чисел, где целое число –

это последовательность цифр со знаком или без знака: «–123», «+98» или «4567», а также целый нуль, который может быть записан со знаком или без знака: «–0», «+0» или просто «0».

Диапазон допустимых значений целых чисел обычно лежит в пределах ±2147483647, в зависимости от *параметра разновидности типа* (п. 3.3.3). Для объектов данных целого типа определены стандартные арифметические действия: сложение, вычитание, умножение и деление.

В Фортране 90/95 определено пять *встроенных типов данных*, на основании которых могут быть созданы *производные* типы данных. Встроенные типы данных отражают характерные особенности организации памяти большинства компьютеров (архитектура фон Неймана) и наиболее распространенные способы хранения данных. Встроенные типы можно использовать непосредственно, без предварительных описаний и подключения каких-либо библиотек.

Такими типами являются:

- INTEGER (целые числа);
- REAL (вещественные числа);
- COMPLEX (комплексные числа);
- CHARACTER (текстовые символы и строки);
- LOGICAL (ИСТИНА или ЛОЖЬ).

В Фортране 90/95 существуют средства создания *производных типов данных* на основе встроенных типов.

При написании программ на языке Фортран 77 можно также использовать специфический вещественный тип (не рекомендованный к использованию в Фортране 90/95): DOUBLE PRECSION (вещественные числа двойной точности).

Для объектов числовых типов определены стандартные арифметические действия и операции, для логического типа – логические операции, а для текстового типа – только операция конкатенации, т.е. слияния текстовых строк. Проблема точности (количества значащих цифр) решается через механизм разновидности типа (п. 3.3.3).

Символьное представление значений, которые принимают переменные (или другие объекты) перечисленных типов, в Фортране называется *буквальными константами* (то, что в других языках и в ранних версиях Фортрана называется *литералами*).

3.3.3. Буквальные константы

Одно и то же число может иметь множество вариантов представления, например число «четыре» может быть записано как:

IV − римская символика;

4 − арабская символика, десятичное представление;

100 − арабская символика, двоичное представление и т.д.

Применительно к Фортрану (как и другим языкам программирования) необходимо понимать, что *буквальная константа* (или литералы в Си или Паскале) – это, прежде всего, *способ его записи,* т.е. способ *видимого представления* этого числа.

Буквальные константы целого типа (INTEGER) представленные в десятичной системе – обычные целые числа (в том числе нуль) со знаком или без знака:

0 +0 –0 12 +345 –675_8 423_K4

Для двух последних буквальных констант явно указаны значения параметра разновидности типа, который может быть выражен через буквальную или именованную целую константу (п. 3.3.4).

Иногда для различных задач удобно представлять положительные целые числа в двоичной, восьмеричной или шестнадцатеричной форме. Для числовых констант без знака в Фортране 90/95 предусмотрены следующие формы записи (например, для десятичного числа «1234»):

B‘10011010010’ – двоичное представление (основание 2);

O‘2322’ – восьмеричное представление (основание 8);

Z‘4D2’ – шестнадцатеричное представление (основание 16).

Буквальные константы вещественного типа (REAL) имеют стандартную форму записи с плавающей точкой, которая состоит из целой части со знаком или без знака, десятичной точки, дробной части и степенной части. Степенная часть состоит из буквы «E» (без относительно регистра) и целой части со знаком или без знака.

Если дробная (или целая) часть равна нулю, то ее можно не записывать, но десятичная точка обязательна:

0.0 –.12 34. 5.64 –98.7E+10 +2.E–100_16
3.21E+99_LONG

Для двух последних буквальных констант явно указаны значения параметра разновидности типа (п. 3.3.4).

Буквальные константы комплексного типа (COMPLEX) обеспечивают возможность непосредственно оперировать с комплексными числами, которые представляют собой заключенную в круглые скобки пару буквальных констант вещественного или целого типа, разделенных запятой. Первая буквальная константа определяет вещественную часть комплексного числа, а вторая – его мнимую часть:

(1.23, 4.56) (7.8, 90) (–98.7E+21, 1.23) (+0, 1.25E–34)

Наличие встроенного комплексного типа является одним из преимуществ Фортрана, как языка предназначенного для научных и инженерных расчетов.

Буквальные константы логического типа (LOGICAL) в Фортране 90/95 присутствуют всего в двух вариантах:

.TRUE. .FALSE.

Обрамляющие точки в записи констант .TRUE. (ИСТИНА) и .FALSE. (ЛОЖЬ) являются обязательными элементами. Некоторые компиляторы Фортрана 77 и 90/95 поддерживают также константы .T. и .F. (опять-таки с обязательными обрамляющими точками).

Буквальные константы текстового или символьного типа (CHARACTER) – последовательности символов, заключенные либо в одиночные апострофы, либо в кавычки:

'HELLO, WORLD' или ″HELLO, WORLD″

Если текстовая константа содержит элемент, заключенный в кавычки, то внутренние кавычки необходимо сдвоить, например: ″The book ″″War and Peace″″ of Leo Tolstoy″. То же самое можно отнести и к апострофам. Чтобы избежать такого сдваивания, нужно заключать константу в апострофы, если внутренний текст содержит кавычки: 'The book "War and Peace" of Leo Tolstoy', и заключать константу в кавычки, если текст содержит апострофы.

Для типа данных CHARACTER символьная длина текстовой константы характеризуется спецификатором LEN. Для буквальной

константы ″HELLO, WORLD″ значение спецификатора LEN=12 (с учетом одного пробела после запятой в тексте константы).

3.3.4. Разновидности типов и диапазоны значений

Очевидно, что для объекта данных любого типа выделяется некоторая память (в машинных словах или байтах). Иногда эта память может оказаться избыточной (если например, для целых чисел в диапазоне ±127 отводится по два или четыре байта, тогда как вполне достаточно одного байта). Это может иметь серьезное значение при обработке больших массивов данных, характерных для математических моделей сложных физических процессов и технических систем.

Идея *разновидности типа* заключается в том, чтобы обеспечить требуемый диапазон значений данных с максимальной экономией памяти, независимо от применяемого компьютера и операционной системы. Таким образом, нужно определить минимальное количество байтов, необходимых для хранения данных, эта величина в Фортране 90/95 имеет название *параметра разновидности типа* и обозначение KIND.

Значение *параметра разновидности* KIND можно вычислить заранее (до программирования) или использовать специальные функции Фортрана. Предположим, требуется обеспечить работу с целыми числами в диапазоне от – 9999 до 9999 (т.е. работу с целыми величинами в пределах четырех значащих цифр). В этом случае значение параметра разновидности можно получить с помощью специальной встроенной функции SELECTED_INT_KIND, аргументом функции будет количество значащих цифр (пример 3.21).

Пример 3.21. Определение параметра KIND через буквальную константу

```
program INT4KIND
print*, SELECTED_INT_KIND(4)
end
```

В примере 3.22 значение функции SELECTED_INT_KIND(4) присваивается константе с именем K4 (может быть любое другое имя), затем объявляются целые переменные X, Y и Z с параметром разновидности KIND=K4. Переменные инициализируются целыми

буквальными константами соответствующей разновидности целого типа (см. также п. 3.3.5).

Пример 3.22. Определение параметра KIND через SELECTED_INT_KIND

```
program INT4KIND
integer(kind=1), parameter :: K4=SELECTED_INT_KIND(4)
integer(kind=K4) :: X=1235_K4, Y=4678_K4, Z=90_K4
print*, K4, X, Y, Z
end
```

Для определения параметра разновидности вещественных значений используется функция SELECTED_REAL_KIND (пример 3.23). Если требуется точность не менее девяти значащих цифр (первый аргумент) и степенной диапазон в пределах от 1.E–99 до 1.E+99 (второй аргумент), то значение KIND можно вычислить, указав соответствующие аргументы: SELECTED_REAL_KIND(9, 99).

Пример 3.23. Определение параметра KIND через SELECTED_REAL_KIND

```
program REALONG
integer(kind=1), parameter :: LONG=SELECTED_REAL_KIND(9,99)
real(kind=LONG) :: X=1.234567895E+98_LONG
print*, LONG, X
end
```

С помощью встроенных функций Фортрана можно решить обратную задачу – определить диапазон значений для заданной разновидности типа KIND.

Для определения модуля максимального значения данной разновидности целого типа, например KIND=8, можно воспользоваться встроенной функцией HUGE (пример 3.24).

Пример 3.24. Максимальное значение для 8-байтовых INTEGER

```
program MAXINT8
integer (kind=8) X
print *, huge(X)
end
```

Для определения модуля максимального значения данной разновидности вещественного типа, например, KIND=16, также мож-

но воспользоваться встроенной функцией HUGE, а для определения границ машинного нуля – функцией TINY (пример 3.25).

Пример 3.25. Диапазон значений и машинный ноль 16-байтовых REAL

```
program MAXMINREAL16
real (kind=16) X
print *, huge(X), tiny(X)
end
```

Компилятор Gfortran предусматривает для данных встроенных типов следующие значения параметра разновидности типа.

Для целых чисел (INTEGER):

KIND=1 (однобайтные целые) в пределах: **±127**;
KIND=2 (двухбайтные целые) в пределах: **±32767**;
KIND=4 (двухбайтные целые) в пределах: **±2147483647**;
KIND=8 (восьмибайтные целые): **±9223372036854775807_8**.
По умолчанию (если параметр разновидности не указан), для целых чисел разновидность KIND=4 считается стандартной.

Для данных вещественного типа REAL:

KIND=4 (четырехбайтные REAL) в пределах: **± 3.40282347E+38**, где границы машинного нуля: **±1.17549435E–38**;
KIND=8 в пределах: **±1.79769313486231571E+308_8**, где границы машинного нуля: **±2.22507385850720138E–308_8**;
KIND=16 имеют значения в пределах:
±1.189731495357231765085759326628007 02E+4932_16,
при этом границы машинного нуля:
± 3.362103143112093506262677817321752 60E-4932_16.
Стандартной разновидностью вещественного типа, по умолчанию, считается KIND=4.
Константа, для которой KIND=4 или параметр разновидности не задан, имеет точность до 7 значащих цифр.

Для комплексных буквальных констант (COMPLEX)

Разновидность типа определяется в соответствии со следующими правилами.

Если вещественная и мнимая части комплексного числа заданы целыми числами, то разновидность комплексной константы совпадает со стандартной разновидностью вещественных чисел. Для компилятора Gfortran это означает KIND=4.

Если имеет место комбинация целого и вещественного числа, то разновидность комплексной константы совпадает с разновидностью вещественного числа.

При наличии двух вещественных чисел с несовпадающими разновидностями максимальная из них будет являться разновидностью комплексной константы. Если разновидность у двух вещественных чисел одна и та же, то такую же разновидность будет иметь комплексная константа.

Компилятор Gfortran работает со следующими разновидностями **логических буквальных констант (LOGICAL):**

KIND=1 – байт, содержащий либо 0 (.FALSE.), либо 1 (.TRUE.);

KIND=2 – двухбайтовый объект логического типа (старший байт соответствует KIND=1, второй содержит значение *NULL*);

KIND=4 – четырехбайтовый объект логического типа (старший (первый) байт в точности аналогичен KIND=1, остальные содержат значение *NULL*).

Для символьных (текстовых) данных (CHARACTER) параметр разновидности KIND типа применяется для указания кодовой таблицы символов, например: ASCII, UTF, Windows-1251и т.д. Оптимальная кодовая таблица всегда задана операционной системой, поэтому параметр KIND лучше не использовать без крайней необходимости. Более детально работа с текстовыми строками рассмотрена в п. 3.4.4.

Правильное (или неправильное) представление буквальных констант может серьезно и даже критично отражаться на точности вычислений. Если, например, для каких-то вычислений потребовалось число π с точностью до 15-ти значащих цифр, то недостаточно объявить соответствующую вещественную переменную или константу как real (KIND=8). Не менее важно правильно инициализи-

ровать такую переменную или константу (т.е. присвоить ей первоначальное значение).

В примере 3.26 две такие константы (PI_1 и PI_2) инициализируются двумя почти одинаковыми вещественными константами. Однако, запустив и выполнив эту программу, можно увидеть ощутимую разницу: для константы PI_1 обеспечена точность только до 7 значащих цифр, а для переменной PI_2 – 15 значащих цифр, как и требовалось. Дело в том, что буквальная константа, не содержащая явного описания разновидности типа (как PI_1), считается буквальной константой стандартной разновидности (т.е. KIND=4) и может обеспечивать соответствующую точность (и не более того) в независимости от того, сколько значащих цифр указано в ее записи. Соответственно, константа 3.1415926535897931 может обеспечить точность только 7 значащих цифр, тогда как константа с явным указанием параметра разновидности 3.1415926535897931_8 (постфикс «_8» явно определяет параметр разновидности) обеспечивает требуемую точность (см. пример 3.26).

Пример 3.26. Точность представления 8-байтовых целых констант

```
program PRECINT8
real (kind=8), parameter ::  PI_1=3.1415926535897931, & ! Перенос строки!
                             PI_2=3.1415926535897931_8
print*, PI_1
print*, PI_2
end
```

3.3.5. Скалярные переменные и константы

Все данные, используемые при программировании на Фортране, подразделяются на *константы* и *переменные*. Значение переменной может изменяться во время работы программы, а значение константы не может быть изменено.

Переменная всегда имеет имя (или идентификатор – п. 3.3.1), а константы подразделяются на *буквальные* (п. 3.3.3) и *именованные* (т.е. константы, имеющие идентификатор). Язык программирования Фортран 90/95 оперирует со *скалярными* переменными и константами, а также с *массивами*.

Скаляр – единичный (не являющийся *массивом*) объект данных (переменная или константа) одного типа.

Массив представляет собой индексированный (определенным образом пронумерованный) набор скаляров одного типа. Обращение к элементу массива осуществляется через имя массива и индекс элемента. Элементы массива могут быть либо только переменными, либо только константами.

Переменные и именованные константы описываются в начале программы до первого исполнимого оператора.

Объявление именованных констант в Фортране 90/95 имеет следующий вид:

тип ([KIND=K]), PARAMETER :: Ик1=БК, Ик2=ИК, Ик3=ВЫР…,

где *тип* – название (идентификатор типа); *K* – параметр разновидности типа либо в виде буквальной или именованной константы, либо как целое арифметическое выражение.

Атрибут *PARAMETER* указывает, на то, что объекты данных, перечисленные через запятую после разделителя «::» (двойное двоеточие), являются *именованными константами* (с именами *Ик1*, *Ик2*, *Ик3* и т.д.), которые могут быть инициализированы (*инициализация* – присвоение первоначального значения) тремя способами:

1) посредством буквальных констант – *БК*;
2) через ранее определенные именованные константы – *ИК*;
3) путем вычисления значения выражения соответствующего типа – *ВЫР*.

Во всех трех случаях необходимо, чтобы именованные константы были инициализированы значениями соответствующей разновидности типа: *KIND=K*. При этом значение параметра разновидности типа *K* может быть задано в виде целой константы (буквальной или ранее определенной именованной) или целого арифметического выражения.

Приведенные примеры 3.20, 3.27 – 3.31 иллюстрируют три способа инициализации именованных констант.

Пример 3.27. Инициализация именованных констант стандартной разновидности (параметр KIND не обязателен) через буквальные константы

```
program CONSTINIT1
real, parameter:: PI= 3.1415927, E= 2.7182817
print*, PI, E
end
```

Пример 3.28. Инициализация именованных констант нестандартной разновидности. Параметр разновидности типа обязателен

```
program CONSTINIT2
real (kind=8), parameter:: PI= 3.1415926535897931_8
real (8), parameter:: E= 2.7182818284590451_8
print*, PI, E
end
```

Пример 3.29. Инициализация именованных констант через выражения, содержащие ранее определенные именованные константы

```
program CONSTINIT3
real, parameter:: PI= 3.1415927, R=1., L= 2.*PI*R
real, parameter:: S= PI*R**2.
print*, PI, R, L, S
end
```

Пример 3.30. Инициализация именованных констант с использованием математических функций

```
program CONSTINIT4
real(8), parameter:: PI = acos(–1._8), E = exp(1._8)
print *, PI, E
end
```

Именованные константы могут быть инициализированы в стиле Фортран 77, в том числе в соответствии с правилами неявного определения типов (см. примеры 3.20 и 3.31).

Пример 3.31. Инициализация именованных констант с неявным определением типа (только для стандартной разновидности типа)

```
program CONSTINIT6
parameter (PI= 3.1415927, MIN = 0)
parameter (E= 2.7182817, MAX =100)
print*, PI, E, MIN, MAX
end
```

Объявление и инициализация скалярных переменных в Фортране 90/95 аналогичны объявлению именованных констант, за исключением атрибута PARAMETER, используемого только при объявлении констант:

тип ([KIND=K]) :: Пер1=БК, Пер2=ИК, Пер3=ВЫР…

Объекты данных, перечисленные через запятую после разделителя «::» (двойное двоеточие), являются *переменными* (с именами

Пер1, *Пер2*, *Пер3* и т.д.), которые, как и именованные константы, могут быть инициализированы тремя способами:

1) посредством буквальных констант – *БК*;
2) через ранее определенные именованные константы – *ИК*;
3) путем вычисления значения выражения *ВЫР*.

Во всех трех случаях переменные следует инициализировать значениями соответствующей разновидности типа: *KIND=K*.

Далее приводятся примеры 3.32 – 3.37, иллюстрирующие объявление переменных в Фортране 90/95.

Пример 3.32. Инициализация переменных стандартной разновидности (параметр KIND не обязателен) через буквальные константы

```
program VARINIT1
real :: PI= 3.1415927, E= 2.7182817
print*, PI, E
end
```

Пример 3.33. Инициализация переменных нестандартной разновидности. Параметр разновидности типа обязателен

```
program VARINIT2
real (kind=8):: PI= 3.1415926535897931_8
real (8):: E= 2.7182818284590451_8
print*, PI, E
end
```

Пример 3.34. Инициализация переменных через выражения, содержащие ранее определенные именованные константы

```
program VARINIT3
real, parameter:: PI= 3.1415927, R=1.
real:: L= 2.*PI*R, S= PI*R**2.
print*, PI, R, L, S
end
```

Пример 3.35. Инициализация переменных с использованием математических функций

```
program VARINIT4
real(8):: PI = acos(–1._8), E = exp(1._8)
print *, PI, E
end
```

Пример 3.36. Инициализация переменных с запретом неявного определения типа

```
program VARINIT5
```

```
implicit none                    ! Запрет неявного определения типа
real (kind=8) PI, E
integer MIN, MAX
PI= 3.1415926535897931_8; E= 2.7182818284590451_8; MIN = 0; MAX = 100
print*, PI, E, MIN, MAX
end
```

Переменные могут быть объявлены (в смысле типа) и инициализированы в стиле Фортран 77, в том числе в соответствии с правилами неявного определения типов. При этом можно использовать свободный формат записи программы.

Пример 3.37. Инициализация переменных с неявным определением типа (только для стандартной разновидности типа)

```
program VARINIT6
PI= 3.1415927; E= 2.7182817; MIN = 0; MAX =100
print*, PI, E, MIN, MAX
end
```

3.3.6. Массивы

Массивы представляют собой индексированные (нумерованные) наборы скалярных объектов (переменных или констант) одного типа. Доступ к элементам массивов осуществляется по индексам (номерам) элементов в массивах.

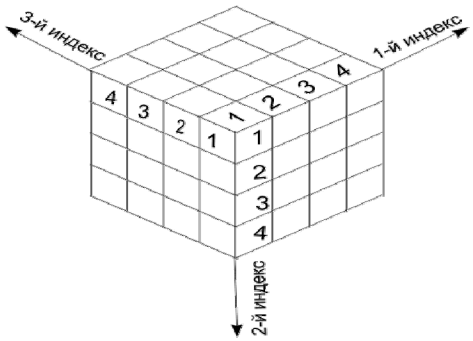


Рис. 3.2. Массив ранга три (трехмерный)

Легко представить себе куб (рис. 3.2), состоящий из мелких кубиков, каждый из которых пронумерован тремя индексами – это модель распределения памяти для трехмерного массива (назовем его CUBE – имя не имеет значения). Элементы массива будут нумероваться от CUBE(1, 1, 1) до CUBE(4, 4, 4).

Ранг массива равен трем (хотя чаще вместо термина «ранг» употребляются старые термины, как *размерность* или *число измерений*). Максимальный ранг массивов в Фортране не может быть больше семи. *Размер массива*, т.е. общее число элементов в массиве CUBE составляет шестьдесят четыре элемента, а *экстенты*, или протяженности, по каждому измерению одинаковы и равны четырем. Совокупность экстентов массива определяют его *форму*. Для CUBE ранг массива (равный трем) и форма: (3, 3, 3) полностью определяют его как массив в модели данных Фортрана 90/95.

Массивы (переменные и константы) объявляются аналогично скалярным переменным и константам, но с дополнительным атрибутом DIMENSION (который, собственно и указывает на то, что объявляемый объект является массивом).

Массивы подразделяются на *статические* (их размер – количество элементов – определяется до начала выполнения программы) и *динамические* (размер которых выясняется в процессе выполнения программы).

Для начала рассмотрим **статические массивы**. Из них простейшими являются **одномерные массивы** (с *размерностью* или *рангом*, равным единице). Одномерный массив является аналогом такого математического объекта как последовательность. Например, последовательность цифр от 0 до 9 может представлять собой одномерный массив целых констант с именем DIGITS (как в примере 3.38). То, что описываемый объект является массивом, состоящим из десяти элементов, показывает атрибут DIMENSION с указанием в круглых скобках размерности массива. Наличие атрибута PARAMETER говорит о том, что элементы массива являются константами.

Пример 3.38. Объявление и инициализация массива констант с указанием верхней границы

```
program ARRINIT1
integer (kind=1), parameter, dimension(10):: DIGITS = (/ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9/)
```

```
print*, DIGITS
end
```

В приведенном примере одномерный массив DIGITS объявлен как DIMENSION(10). Величина 10 – это размер (количество элементов) в массиве. В то же время такая запись показывает, что 10 – это *верхняя граница массива* (максимальный номер элемента), а *нижняя граница массива* (минимальный номер элемента) будет равна единице, поскольку явно не определена. Это означает, что элементы массива DIGITS будут нумероваться следующим образом (и иметь соответствующие значения):

DIGITS (1) = 0, DIGITS (2) = 1, … , DIGITS (10) = 9

В наиболее общем случае задаются обе границы массива – они могут иметь положительное, отрицательное или нулевое (но обязательно целое) значение. При этом важно, чтобы значение верхней границы было больше значения нижней границы.

Можно изменить нумерацию элементов массива DIGITS, потребовав, например, чтобы они нумеровались с нуля:

DIGITS (0) = 0, DIGITS (1) = 1, … , DIGITS (9) = 9

Для этого нужно явно задать нижнюю границу массива с учетом того, что по правилам Фортрана верхняя и нижняя границы массива в атрибуте DIMENSION разделяются двоеточием (пример 3.39).

Пример 3.39. Объявление и инициализация массива констант с указанием нижней и верхней границ

```
program ARRINIT2
integer (kind=1), parameter, dimension(0:9):: DIGITS = (/ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9/)
print*, DIGITS
end
```

Для инициализации (присвоения значений элементам) одномерных массивов в Фортране 90/95 предусмотрен специализированный конструктор (он использован в примерах 3.38 и 3.39), который в простейшем случае представляет собой список (через запятую) буквальных констант и ограниченный лексемами «(/» и «/)».

Пример 3.40. Объявление и инициализация массива констант при помощи конструктора

```
program ARRINIT3
integer (kind=1), parameter, dimension(0:9):: DIGITS = (/ ( I, I = 0, 9) /)
print*, DIGITS
end
```

Конструктор инициализации массивов (пример 3.40) позволяет представлять список инициализации в виде значений, зависящих от некоторой переменной, например, список: 0, 1, 2, …, 9 может быть представлен в виде *неявного цикла*: (I, I = 0, 9). Здесь переменная цикла I (имя не принципиально, но это должна быть целая величина) пробегает все целые значения от 0 до 9.

Конструкторы инициализации массивов позволяют формировать списки значений, подчиненные более сложным арифметическим закономерностям. Например, можно составить список только нечетных элементов от 1 до 9 или в десять раз меньших элементов.

В первом случае значение переменной неявного цикла I начинается с единицы, заканчивается девяткой, при этом для отбора только нечетных значений шаг изменения I равен двум: (I, I = 1, 9, 2).

Во втором случае шаг изменения I стандартный (равен единице и потому не указывается), но добавляется масштабирующий множитель: 0.1, уменьшающий значения, пробегаемые переменной цикла I в десять раз: (I*0.1, I = 1, 9, 2).

Следует отметить, что неявные циклы имеют большое значение и часто применяются при выводе массивов на печать, в файлы и т.д. (например, в операторе PRINT, пример 3.41).

Пример 3.41. Использование неявных циклов при инициализации массива и его выводе на печать

```
program ARRINIT4
integer (kind=1), parameter, dimension(0:9):: DIGITS = (/ ( I, I = 0, 9) /)
print*, (DIGITS(I), I = 0, 9))
end
```

Для приведенных примеров важно еще раз отметить, что список значений элементов (/ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9/) и диапазон индексов (0:9) – это разные вещи, не смотря на то, что выглядят одинаково.

Широкое распространение в компьютерных вычислениях имеют **двумерные массивы** (с *размерностью* или *рангом*, равным двум).

Двумерный массив является аналогом такого математического объекта, как двумерная таблица или матрица, например:

1 2 3
4 5 6
7 8 9

Элементы двумерного массива (например, матрицы; обозначим ее переменной или константой MATRIX) адресуются двумя индексами: номером строки, часто обозначаемым целой переменной I, и номером столбца, за него обычно отвечает целая переменная J.

Таким образом, любой элемент матрицы может быть обозначен как MATRIX(I, J), где I и J могут принимать целые значения в диапазоне от единицы до трех. Так, элемент «5» приведенной выше матрицы будет иметь индексацию MATRIX(2, 2).

В машинной памяти все двумерные массивы размерности (M, N) хранятся как одномерные массивы, имеющие сквозную нумерацию от 1 до (M*N) и сформированные из последовательности столбцов соответствующей матрицы.

Рассмотрим вопрос об объявлении и инициализации подобного массива MATRIX (пример 3.42). При этом будем считать, что элементы матрицы – переменные целого типа INTEGER стандартной разновидности.

Для инициализации массива воспользуемся оператором DATA, который позволяет присвоить массиву-переменной список значений: разделителями элементов списка значений являются запятые, а сам список целиком заключен в прямые слеши «/»:

Пример 3.42. Объявление и инициализация двумерного массива переменных при помощи оператора DATA и его построчный вывод

```
program ARRINIT5
integer, dimension(3, 3) :: MATRIX
data MATRIX /1, 2, 3, 4, 5, 6, 7, 8, 9/
I=1; print*, (MATRIX(I, J), J = 1, 3)
I=2; print*, (MATRIX(I, J), J = 1, 3)
I=3; print*, (MATRIX(I, J), J = 1, 3)
end
```

Если инициализирующий список матрицы MATRIX состоит из чисел от единицы до девяти в порядке возрастания, то при выводе

на экран получается не совсем тот результат, который ожидался. Ожидалась матрица:

```
1   2   3
4   5   6
7   8   9
```

При этом на экране видим, что первая строка стала первым столбцом, вторая – вторым столбцом, а третья – третьим:

```
1   4   7
2   5   8
3   6   9
```

Это произошло потому, что при поточном вводе данных в массив (например, оператор DATA) они записываются по столбцам, а не по строкам – это одна из особенностей Фортрана (например, в языке Си при подобном вводе данные запишутся по строкам). Если внести соответствующие изменения входных данных, то все станет на свои места (пример 3.43).

Вернемся к вопросу об индексации массивов. При объявлении двумерного массива MATRIX в атрибуте DIMENSION(3, 3) указаны только верхние границы массива по каждому измерению, это означает, что нижние границы по каждому измерению равны единице. Как было в случае с одномерным массивом или как в принципе допустимо с семимерным массивом, можно указывать диапазоны границ по каждому измерению (см. пример 3.43).

Пример 3.43. Объявление и инициализация двумерного массива переменных при помощи оператора DATA и его построчный вывод

```
program ARRINIT6
integer, dimension(3, 3) :: MATRIX
data MATRIX /1, 4, 7, 2, 5, 8, 3, 6, 9/
I=1; print*, (MATRIX(I, J), J = 1, 3)
I=2; print*, (MATRIX(I, J), J = 1, 3)
I=3; print*, (MATRIX(I, J), J = 1, 3)
end
```

Диапазон индексов представляет собой нижнюю и верхнюю границы индекса, разделенные двоеточием (пример 3.44).

Пример 3.44. Объявление и инициализация двумерного массива с явным указанием диапазона индексов по каждому измерению

```
program ARRINIT7
integer, dimension(1:3, 1:3) :: MATRIX
data MATRIX /1, 4, 7, 2, 5, 8, 3, 6, 9/
I=1; print*, (MATRIX(I, J), J = 1, 3)
I=2; print*, (MATRIX(I, J), J = 1, 3)
I=3; print*, (MATRIX(I, J), J = 1, 3)
end
```

При необходимости можно было бы определить матрицу MATRIX как DIMENSION(–1:1, 0:2) с диапазоном индекса строк от минус единицы до плюс единицы (с учетом нуля размерность будет равна трем) и диапазоном столбцов от нуля до двух.

В примерах 3.41 – 3.44 показана инициализация массивов при помощи конструктора инициализации, пригодного только для одномерных массивов. Для того чтобы воспользоваться таким конструктором для инициализации двумерных (и более) массивов, используется функция RESHAPE (пример 3.45).

Пример 3.45. Объявление и инициализация двумерного массива с применением конструктора инициализации и функции преобразования формы

```
program ARRINIT8
integer, dimension(3, 3) :: MATRIX=RESHAPE( (/1, 4, 7, 2, 5, 8, 3, 6, 9/), (/3,3/) )
I=1; print*, (MATRIX(I, J), J = 1, 3)
I=2; print*, (MATRIX(I, J), J = 1, 3)
I=3; print*, (MATRIX(I, J), J = 1, 3)
end
```

В простейшем случае эта функция содержит два аргумента (в круглых скобках, через запятую): сам конструктор одномерного массива констант и константу, определяющую форму конечного массива – верхние границы индексов, которые указываются для атрибута DIMENSION, но в упаковке лексем «(/» и «/)».

Возможно объявление и инициализация массивов в стиле Фортран 77 (с сохранением свободного формата записи) с операторами PARAMETER и DIMENSION (пример 3.46). Это можно проиллюстрировать на примере рассмотренных выше массивов: массива-константы DIGITS и массива-переменной MATRIX.

Пример 3.46. Объявление и инициализация массивов в стиле Фортран 77 при помощи операторов PARAMETER и DIMENSION

```
program ARRINIT9
integer DIGITS      ! MATRIX – объект целого типа по умолчанию
dimension DIGITS(10), MATRIX(3,3)
parameter  (DIGITS=(/0, 1, 2, 3, 4, 5, 6, 7, 8, 9/) )
data MATRIX /1, 4, 7, 2, 5, 8, 3, 6, 9/
print*, DIGITS
I=1; print*, (MATRIX(I, J), J = 1, 3)
I=2; print*, (MATRIX(I, J), J = 1, 3)
I=3; print*, (MATRIX(I, J), J = 1, 3)
end
```

В заключение рассмотрим работу с динамическими или, как они называются в Фортране, выделяемыми массивами (пример 3.47). Для объявления выделяемого массива помимо уже известного атрибута DIMENSION требуется еще атрибут – ALLOCATABLE. Ранг массива описывается в атрибуте DIMENSION как шаблон, т.е. на месте диапазонов границ по каждому измерению просто ставится двоеточие (границы при этом не указываются).

Пример 3.47. Объявление и инициализация динамического массива

```
program ARRINIT10
real, dimension(:,:), allocatable :: MATRIX
N=2                                ! В качестве теста берется матрица 2x2
ALLOCATE( MATRIX(1:N, 1:N) ) ! Можно оставить только верхние границы
read *, MATRIX    !Чтение массива. Ввести четыре числа через пробел
I=1; print*, (MATRIX(I, J), J = 1, 2)
I=2; print*, (MATRIX(I, J), J = 1, 2)
DEALLOCATE( MATRIX)
end
```

Память для такого массива выделяется после успешного выполнения оператора ALLOCATE (до этого момента границы массива по каждому измерению остаются неопределенными). В ALLOCATE границы (в данном случае массива MATRIX) описываются в круглых скобках после имен массивов, так же как для атрибута DIMENSION при работе со статическими массивами.

Если выделяемый массив MATRIX больше не нужен, то можно освободить от него память при помощи оператора DEALLOCATE.

3.3.7. Производные типы данных

Для обеспечения возможности работы со сложными структурами данных в Фортране 90/95 предусмотрена возможность создания новых (производных) типов данных, на основе встроенных типов. В качестве иллюстрации (пример 3.48) рассмотрим создание производного типа POINT, моделирующего точку на координатной плоскости с вещественными (стандартной разновидности) координатами (X, Y). Скалярный объект производного типа называется *структурой*, содержащей несколько элементов, которые являются данными встроенных типов и объявляются точно так же, как переменные встроенных типов, но при этом заключены в операторные скобки (TYPE … END TYPE). Для обращения к элементу структуры сначала указывается имя объекта, а затем через разделитель «%» (символ процента) имя элемента структуры: C%X, C%Y и т.д.

Пример 3.48. Объявление производного типа «Точка на плоскости»

```
program POINTYPE
type POINT
real  X, Y
end type POINT
type(POINT) , parameter :: A = POINT(1., 2.), B = POINT(3., 4.)
type(POINT) C, D
C%X =5.; C%Y =6.; D%X =7.; D%Y =8.
print *, A; print *, B; print *, C; print *, D
end
```

Скалярные объекты производного типа, так же как объекты встроенных типов, подразделяются на переменные (как C и D в примере 3.48) и константы A и B (там же).

Как и для встроенных типов, существует понятие буквальных констант, например: POINT(1., 2.) и B = POINT(3., 4.). Буквальная константа встроенного типа определяется как список (через запятую) констант, являющихся значениями элементов структуры объекта – этот список записывается в круглых скобках после имени производного типа.

Пример 3.49. Объявление производного типа «Вектор»

```
program VECTYPE
! Объявление производных типов
```

```
type POINT
real  X, Y
end type POINT
type VECTOR
type (POINT) P1, P2
end type VECTOR
! Объявление и инициализация переменных производных типов
type (POINT), parameter :: A = POINT(1., 2.), B = POINT(3., 4.)
type (VECTOR) ::  AB =VECTOR(A, B), &
CD = VECTOR( POINT(3., 4.),  POINT(2., 1.))
type (VECTOR) EF
EF%P1%X  =5.; EF%P1%Y  =6.; EF%P2%X  =7.; EF%P2%Y  =8.;
print *, AB; print *, CD
end
```

Производный тип может быть определен как через встроенные типы, так и через ранее определенные производные типы. В качестве примера можно, используя точки на координатной плоскости (тип POINT), определить тип VECTOR как пару точек P1 и P2 на координатной плоскости (пример 3.49).

В приведенном примере заданы три объекта типа VECTOR – векторы AB, CD и EF. Первый из них (вектор AB) инициализирован двумя предварительно определенными именованными константами A и B типа POINT, второй (вектор CD) – двумя буквальными константами того же типа. Третий (вектор EF) инициализирован по компонентам: поскольку вектор EF, как и любой объект типа VECTOR, имеет элементы структуры P1 и P2, каждый из которых – компоненты X и Y, то необходимо инициализировать компоненты EF%P1%X, EF%P1%Y и т.д.

Стоит отметить, что, используя атрибут PARAMETER (см. п. 3.3.6), можно объявлять массивы производных типов. Массивы также могут являться элементами структуры объектов производного типа, при этом единичный объект (производного типа), в состав которого входит массив, считается скалярным объектом.

3.4. ВЫРАЖЕНИЯ И ПРЕОБРАЗОВАНИЕ ТИПОВ

3.4.1. Скалярное присваивание

Если с точки зрения элементарной математики запись X=X+1, констатирующая равенство левой и правой части, не имеет смысла ввиду отсутствия значения X, удовлетворяющего такому уравнению, то для программирования такая запись имеет важный практический смысл. Символ «=» (знак равенства в математике) используется в таких языках, как Фортран, Си и ряде других, для того, чтобы записать значение выражения X+1 (вычисленного для текущего значения переменной) в ячейку памяти X. Тем самым текущее значение переменной X заменяется новым, равным величине текущего значения X, увеличенного на единицу или, как говорят, переменной X, посредством *оператора присваивания* «=», присваивается новое значение, равное X+1.

Действие оператора присваивания наглядно иллюстрируется примером 3.50. После инициализации вещественной переменной X нулем ее значение три раза подряд инкрементируется (увеличивается на единицу), т.е. берется текущее значение X, к нему добавляется единица и получившееся значение записывается в переменную X посредством оператора присваивания. С помощью этого оператора можно присваивать значения переменным и именованным константам всех встроенных типов Фортрана.

Пример 3.50. Иллюстрация работы оператора присваивания

```
program ASSIGN
real :: X=0; print*, X
X=X+1; print*, X
X=X+1; print*, X
X=X+1; print*, X
end
```

Рассмотрим общую форму скалярного числового присваивания, она имеет вид: ПЕР = ВЫР, и подразумевает, что переменной ПЕР присваивается значение, полученное в результате вычисления выражения ВЫР (под выражением может подразумевается не только какая-либо вычислительная формула но и переменная, а также именованная или буквальная константа).

Идеальный и наиболее быстрый, с точки зрения работы программы, вариант, когда переменная слева от оператора присваивания (ПЕР) и объект справа от него (ВЫР) являются данными одного типа, с одним значением параметра разновидности типа KIND. В противном случае автоматически будет произведена процедура преобразования типа в соответствии с табл. 3.2. Таким образом, результат вычисления выражения преобразуется к типу переменной, с соответствующим параметром разновидности KIND, с использованием стандартных функций Фортрана: INT, REAL, CMPLX и KIND, приведенных в разд. 4.

Таблица 3.2. Преобразование типов при выполнении присваивания

Тип переменной	Преобразование выражения
INTEGER	INT(ВЫР, KIND=KIND(ПЕР))
REAL	REAL(ВЫР, KIND=KIND(ПЕР))
COMPLEX	CMPLX(ВЫР, KIND=KIND(ПЕР))

Вывод в отношении вышесказанного и дальнейших рассуждений довольно простой – необходимо следить за соответствием типов при написании выражений и присваиваний.

3.4.2. Арифметика Фортрана

Для числовых типов Фортрана (INTEGER, REAL и COMPLEX) всех разновидностей (см. п. 3.3.4) определены *арифметические операции* (табл. 3.3) и *скобочные конструкции*, соответствующие стандартным (математическим) правилам арифметики.

Таблица 3.3. Арифметические операторы Фортрана

Оператор	Операция	Приоритет
( )	Выражение в скобках	**1 – макс.**
**	Возведение в степень	**2**
*	Умножение	**3**
/	Деление	**3**
+	Сложение или унарный плюс	**4 – мин.**
–	Вычитание или унарный минус	**4 – мин.**

С помощью арифметических операций со скобочными конструкциями и числовых объектов данных (переменных, а также буквальных и именованных констант) можно программировать *арифметические выражения* любой сложности.

Операторы «+» и «–», представленные ранее, могут применяться для унарных операций, например для выражений именованными объектами как: +Z или X**(–Y), или буквальными константами числовых типов: – (–1). Отметим, что в последнем случае унарный минус применяется к буквальной константе «–1», для которой ее знак является неотъемлемой частью.

Вернемся к бинарным операциям (сложению, вычитанию, умножению и делению). Наивысшим приоритетом (в 1-ю очередь) при вычислении арифметических выражений обладают скобочные конструкции. Если имеет место вложенность скобочных конструкций, то сначала будет произведен расчет выражений во внутренних скобках. Между собой скобочные конструкции имеют равный приоритет, т.е. вычисляются по мере их появления в выражении.

В последнюю очередь (по порядку с равным приоритетом) производится сложение и вычитание – перед сложением и вычитанием с равным приоритетом (в порядке следования в выражении) выполняется умножение или деление. Если в выражении присутствует возведение в степень, то его вычисление должно предшествовать умножению или делению, но при этом осуществляться после вычислений в скобках (при наличии таковых).

Приоритет арифметических вычислений рассмотрен на примере вычисления арифметического выражения:

A + B / (C / D) – E**F*(G – H*(I+J))

Сначала на печать выводится расчет по единой формуле, а затем разбирается ее пошаговое вычисление (пример 3.51). Критерием правильности пошаговых вычислений является их совпадение с результатом, полученным по единой формуле.

Пример 3.51. Приоритет возведения в степень (вариант 1)

```
program STEPCALC
real :: A=9.,  B=8., C = 7., D = 6., E = 5., F = 4., G = 3., H = 2., I = 1. , J = 2.
print*, A + B / (C / D) – E**F*(G – H*(I+J)) ! ОДНОЙ ФОРМУЛОЙ
! A + B / (C / D) – E**F*(G – H*(I+J)) ПО ШАГАМ:
```

```
! В выражении три равноценных слагаемых: A, B / (C / D) и – E**F*(G – H*(I+J))
! Вычисление слагаемого B / (C / D) начинается со скобок (C / D), затем деление
STEP1 = C / D; STEP2 = B / STEP1
! Вычисление E**F*(G – H*(I+J)) начинается со скобок (G – H*(I+J))
! Расчет (G – H*(I+J)) начинается со скобок (I+J), далее умножение и вычитание
STEP3 = I+J; STEP4 = H*STEP3; STEP5 = G – STEP4
! Степень E**F вычисляется после скобок, затем производится умножение
STEP6 = E**F; STEP7 = STEP6* STEP5
! Осталось сложить результаты
RESULT = A + STEP2 – STEP7
print*, RESULT
end
```

Необходимо отметить важную особенность операции возведения в степень. Например, вычисление выражения A**B**C (пример 3.52), так же как вычисление выражения с любым количеством последовательных возведений в степень, всегда выполняется справа налево. Таким образом, сначала будет вычислено B**C, а полученный результат послужит степенью, в которую будет возведено A.

Пример 3.52. Приоритет возведения в степень (вариант 2)

```
program POWCALC
real :: A=4.,  B=3., C = 2.
print*, A**B**C !Возведение в степень одним выражением
! Пошаговое вычисление последовательных возведений в степень
STEP1 = B**C; STEP2 = A**STEP1
print*, STEP2
end
```

Важнейшим вопросом компьютерных вычислений является преобразование типов данных, одновременно присутствующих в арифметических выражениях. Арифметическое выражение может содержать операнды нескольких числовых типов Фортрана. При этом операнды всегда попарно связаны арифметическими операциями. Перед выполнением каждой арифметической операции операнды проверяются на соответствие типов, и если типы операндов (включая KIND) совпадают, то результат будет того же типа.

Если же типы (или KIND при совпадении типов) различны, то перед выполнением операции производится сравнение типов операндов. Операнд с более мощным типом никак не преобразуется, а операнд с менее мощным типом преобразуется к более мощному

типу, таким же будет и тип результата операции. Мощность числового типа возрастает по цепочке INTEGER–REAL–COMLEX, а в пределах каждого типа растет с увеличением значения параметра разновидности KIND.

Зависимость типа результата арифметической операции, если это сложение, вычитание, умножение или деление, в зависимости от типов операндов представлена в табл. 3.4, а операции возведения в степень соответствует табл. 3.5.

Таблица 3.4. Таблица преобразования типов операндов для сложения, вычитания, умножения и деления

Тип A	Тип B	Преобразование типа для A	Преобразование типа для B	Тип результата
I	**I**	A	B	**I**
I	**R**	REAL (A, KIND(B))	B	**R**
I	**C**	CMPLX (A, 0, KIND(B))	B	**C**
R	**I**	A	REAL (B, KIND(A))	**R**
R	**R**	A	B	**R**
R	**C**	CMPLX (A, 0, KIND(B))	B	**C**
C	**I**	A	CMPLX (B, 0, KIND(A))	**C**
C	**R**	A	CMPLX (B, 0, KIND(A))	**C**
C	**C**	A	B	**C**

Таблица 3.5. Таблица преобразования типов операндов для операции возведения в степень

Тип A	Тип B	Преобразование типа для A	Преобразование типа для B	Тип результата
I	**I**	A	B	**I**
I	**R**	REAL (A, KIND(B))	B	**R**
I	**C**	CMPLX (A, 0, KIND(B))	B	**C**
R	**I**	A	B	**R**
R	**R**	A	B	**R**
R	**C**	CMPLX (A, 0, KIND(B))	B	**C**
C	**I**	A	B	**C**
C	**R**	A	CMPLX (B, 0, KIND(A))	**C**
C	**C**	A	B	**C**

В табл. 3.4 и 3.5 в первых двух столбцах жирным шрифтом обозначены исходные типы операндов, а в последнем столбце тип результата операции (I – INTEGER, R – REAL, C – COMPLEX). Опе-

ранды арифметической операции обозначены как A – первый операнд и B – второй операнд.

Из табл. 3.5 видно, что единственный случай, когда не происходит преобразования типов операндов арифметической операции к одному типу – это возведение вещественного или комплексного операнда в целую степень.

Пример 3.53 дает наглядное представление о преобразовании типов операндов в простейших арифметических выражениях.

Пример 3.53. Преобразование типов и тип результата

```
program CONVERTYPE
integer :: a=1; real :: b=1.; complex :: c=(1. , 1.)
print*, a*a !Результат  INTEGER
print*, a*b !Переменная A преобразуется в REAL. Результат  REAL
print*, a*c !Переменная A преобразуется в COMPLEX. Результат COMPLEX
print*, b*b ! Результат  REAL
print*, b*c !Переменная B преобразуется в COMPLEX. Результат COMPLEX
print*, c*c ! Результат COMPLEX
end
```

Интересным разделом компьютерных вычислений является целочисленная арифметика (она соответствует первым строчкам табл. 3.4 и 3.5). Особенно «парадоксальным» образом это проявляется при операциях целочисленного деления. Казалось бы, простой вопрос: сколько будет один делить на два? Вроде бы очевидно, что нуль целых пять десятых. Но запрограммируйте это действие с целыми числами, и такого результата не будет, а будет нуль.

При делении целых операндов результат округляется до целого значения, путем отбрасывания дробной части. Особенность деления целых операндов показана в примере 3.54.

Пример 3.54. Особенности деления целых чисел

```
program INTDIV
print*,1/2!Результат=0 (целочисленное деление)
print*,1./2          !Результат=0.5, поскольку 1. – вещественное число
print*,3/2!Результат=1(целочисленное деление)
print*,3./2          !Результат=1.5, поскольку 3. – вещественное число
print*,3/2*2         !Результат=2 (целочисленное деление)
print*,3./2*2        !Результат=3.
print*,1/2*2         !Результат=0
print*,1./2*2        !Результат=1.
end
```

3.4.3. Логические выражения

Логическими данными чаще всего являются *логические отношения* числовых или текстовых данных (п. 3.5.1), хотя могут иметь вид самостоятельных логических выражений. Каждая операция логического отношения имеет два равноценных обозначения, поддерживаемых стандартом Фортрана 90/95 (табл. 3.6). Для обозначений в первом столбце таблицы, унаследованных от ранних версий Фортрана (Фортран 77), обязательны обрамляющие точки, являющиеся частью мнемоники операции, т.е. запись с обрамляющими точками: «.EQ.» – правильно, а без них: «EQ» – неправильно. При записи таких операций отношения, как «= =», «/ =» и ниже, в том же столбце табл. 3.6 обрамляющие символы (точки или что-то еще) не требуются, но нужно следить, чтобы между символами не было пробелов. Если в тексте данного руководства такие пробелы присутствуют, то только для улучшения читаемости, поскольку не всякий читатель найдет на клавиатуре кнопку с длинным знаком равенства (по аналогии с клавишей Any Key).

Таблица 3.6. Логические отношения в Фортране

Обозначения			Отношение
.EQ.	или	= =	РАВНО
.NE.	или	/ =	НЕ РАВНО
.GT.	или	>	БОЛЬШЕ
.GE.	или	> =	БОЛЬШЕ ИЛИ РАВНО
.LT.	или	<	МЕНЬШЕ
.LE.	или	< =	МЕНЬШЕ ИЛИ РАВНО

Правила объявления и инициализация логических переменных и констант практически ничем не отличаются от объявления и инициализация числовых данных (см. п. 3.4.1) – в точности также используется атрибут PARAMETER для объявления именованных констант и оператор присваивания «=» для присваивания логических значений переменным и именованным константам (пример 3.55). Для объявления логических массивов в примере используется атрибут DIMENSION.

Для обеспечения возможности построения из операций отношения более сложных логических выражений в Фортране предусмот-

рены логические операции, представленные в табл. 3.7 в порядке убывания приоритета.

Пример 3.55. Особенности деления целых чисел

```
program LOGFACT1
logical, parameter :: TRUFACT=.TRUE., FALFACT=.FALSE.
logical, parameter , dimension(0:1):: FACTS=(/.FALSE., .TRUE. /)
logical FACT1, FACT2
FACT1 = 0 .EQ. 0; FACT2 = – 1 /= 1  !Присваивание логических значений
print*, TRUFACT, FALFACT            ! Печать логических именованных констант
print*, FACT1, FACT2
print*, FACTS
end
```

Таблица 3.7. Логические операторы Фортрана

Обозначение	Название оператора
.NOT.	ЛОГИЧЕСКОЕ ОТРИЦАНИЕ (НЕ)
.AND.	ЛОГИЧЕСКОЕ ПЕРЕСЕЧЕНИЕ (И)
.OR.	ЛОГИЧЕСКОЕ ОБЪЕДИНЕНИЕ (ИЛИ)
.EQV.	ЛОГИЧЕСКАЯ ЭКВИВАЛЕНТНОСТЬ
.NEQV.	ЛОГИЧЕСКАЯ НЕЭКВИВАЛЕНТНОСТЬ

Если операндами логических отношений (см. табл. 3.6) могут быть только текстовые и числовые данные, то операндами логических операций могут быть только логические данные.

Логические отношения (см. табл. 3.6) имеют более высокий приоритет выполнения по сравнению с логическими операциями (см. табл. 3.7). Каждое логическое отношение (утверждение) является элементарным логическим выражением и может принимать значения: .TRUE. или .FALSE., все операции отношения имеют одинаковый приоритет выполнения по отношению друг к другу.

Из логических данных и элементарных логических выражений можно выстраивать достаточно сложные логические выражения. При записи логических выражений можно использовать круглые скобки. Заключенные в круглые скобки части логического выражения вычисляются в первую очередь.

Оба логических выражения (для переменной FACT1 и FACT2 в примере 3.56) содержат смешение стилей: для X > 0 используется знак «больше», а для Y.LT.2 используется мнемоника «меньше», что допустимо, но чего лучше избегать в силу возникающей пута-

ницы. В первом выражении проверяется истинность хотя бы одного из условий: принадлежности X интервалу от нуля до двух (включая два) или принадлежности Y тому же интервалу (включая нуль), а во втором выражении проверяется одновременная истинность тех же условий.

Поскольку логическое объединение «ИЛИ» (.OR.) имеет меньший приоритет, по сравнению с логическим пересечением «И» (.AND.), то в первом выражении скобки не требуются, а во втором выражении они необходимы для регулирования приоритета – иначе логические пересечения «И» (.AND.) будут выполняться последовательно, как имеющие равный приоритет.

Пример 3.56. Сложные логические выражения

```
program LOGFACT2
logical  FACT1, FACT2
real :: X= 1., Y =2.
FACT1=X > 0. .and. X <= 2. .or. Y .ge. 0. .and. Y .lt. 2.
FACT2=(X > 0. .and. X <= 2.) .and. (Y .ge. 0. .and. Y .lt. 2.)
print*, FACT1, FACT2
end
```

Практическое использование логических выражений рассматривается в связи с их использованием в логических операторах и конструкциях IF (п. 3.6.1).

3.4.4. Работа с текстовыми строками

Как уже отмечалось (см. п. 3.3.4), тестовые данные (CHARACTER) имеют особенность описания, связанную с параметром разновидности типа: KIND, который применяется для указания кодовой таблицы символов: ASCII, UTF, Windows-1251 и т.д. При этом оптимальная кодовая таблица символов уже выбрана операционной системой, при инсталляции компилятора Фортрана и поэтому с кодовыми страницами символов нужно работать аккуратно. Если есть необходимость явного указания кодовой страницы, то KIND указывается через запятую, с длиной тестовой строки в символах LEN, например: CHARACTER (KIND= KIND(UTF), LEN=13).

LEN является параметром, отвечающим за длину текстовой строки, измеряемую количеством символов. С его помощью мож-

но, например, объявить именованную 13-ти символьную текстовую константу TEXT (произвольное имя) со значением «Hello, World!».

С константой TEXT (пример 3.57) можно работать не только как с целостным объектом, но и выделять в нем подстроки, задавая их границы через номера начального и конечного символа. Например, нетрудно видеть, что подстрока «Hello» – это символы строки TEXT с первого по пятый, а подстрока «World!» – символы с восьмого по тринадцатый. Самое простое, что можно сделать со строкой и ее подстроками – это напечатать их (см. пример 3.57).

Пример 3.57. Текстовая константа и ее подстроки

```
program TEXTLINE1
character(LEN=13), parameter :: TEXT=”Hello, World!”
print*, TEXT;  print*, TEXT(1:5);  print*,TEXT(8:12)
end
```

Если в программе используются текстовые константы, то их значения (как и любых других констант) не изменяются, соответственно, еще на стадии компиляции может быть автоматически посчитано количество символов в константе и произведена ее инициализация. Необязательно считать символы в константе – это сделает компилятор, если в качестве значения длины строки LEN указать: «*» (звездочку). При этом возможно сокращение записи – «звездочка» без LEN (пример 3.58).

Пример 3.58. Автоматическое определение длины текстовой константы

```
program TEXTLINE2
character(LEN = *), parameter :: TEXT1=”Hello, World!”
character(*), parameter :: TEXT2=”Global World! “
print*, TEXT1;  print*, TEXT1(1:5);  print*,TEXT1(8:12)
print*, TEXT2;  print*, TEXT1(1:6);  print*,TEXT1(8:12)
end
```

С текстовой переменной, как и с константой, можно работать как с целостным объектом (это происходит при инициализации), а можно работать с подстроками, задавая их границы (пример 3.59).

Пример 3.59. Текстовая переменная и ее подстрока (вариант 1)

```
program TEXTLINE3
character(LEN=13) :: TEXT= “Hello, World!”
TEXT(1:6) = “Global”
```

```
print*, TEXT
end
```

Указание длины строки LEN для текстовых переменных обязательно.

Для тестовых данных определена единственная операция – конкатенация (пример 3.60), обозначаемая сдвоенными прямыми слешами: «//». Конкатенация позволяет последовательно объединить несколько текстовых строк в одну строку.

Пример 3.60. Текстовая переменная и ее подстрока (вариант 2)

```
program TEXTLINE4
character(LEN=*), parameter :: TXT1=”Hello”, TXT2 = “Global”, TXT3=”World!”
character(LEN=LEN(TXT1)+LEN(TXT2)+LEN(TXT3)+3) :: TEXT
TEXT= TXT1 // “,^” // TXT2 // “^” // TXT3 // “!”; print *, TEXT
end
```

Помимо конкатенации в примере 3.60 использована строковая функция LEN, возвращающая в качестве результата длину строки, являющейся аргументом этой функции: LEN(TXT1) и т.д. Необходимо также вспомнить соглашения и рекомендации, по поводу обозначения пробелов (см. п. 3.1.1).

3.4.5. Операции с массивами

Любую операцию, определенную для скалярных встроенных объектов, можно применить к одному массиву, если речь идет об унарной операции, или двум массивам, когда дело касается бинарной операции. В последнем случае оба массива должны быть одинаковой формы (см. п. 3.3.6), т.е. одного ранга и с одинаковыми экстентами.

Пример 3.61. Объявление и инициализация массива констант с указанием верхней границы (вариант 1)

```
program OPERARR1
integer, parameter, dimension(5):: A = (/ 0, 1, 2, 3,4/), B = (/5, 6, 7, 8, 9/)
integer, dimension(5):: C, D, E, F, G
C = A + B; D = A – B; E = A * B; F = A / B; G = A**B
print*, A; print*, B; print*, C; print*, D; print*, E;  print*, F;  print*, G
end
```

Бинарная операция, примененная к двум массивам, имеет такой же результат, как если бы эта операция была попарно применена к элементам первого и второго массива с одинаковыми номерами (индексами). Это легко можно увидеть на примере арифметических операций (см. табл. 3.3) между одномерными (пример 3.61) и двумерными (пример 3.62) массивами вещественных чисел, т.е. операции с массивами не имеют ничего общего с операциями над матрицами или векторами.

Допускаются бинарные операции между массивами и скалярами. Действие бинарной операции между массивом и скаляром равноценно совокупности операций между скаляром и всеми элементами массива, например операция умножения суммарного массива A+B (см. пример 3.62) на скалярную переменную X = 2 равноценна удвоению каждого элемента этого массива. Действие унарных операций (например, унарный минус) имеет аналогичный характер.

Пример 3.62. Объявление и инициализация массива констант с указанием верхней границы (вариант 2)

```
program OPERARR2
integer, parameter, dimension(2, 2):: A = RESHAPE ((/ 0, 1, 2, 3 /), (/2, 2/) )
integer, parameter, dimension(2, 2):: B = RESHAPE ((/ 4, 5, 6, 7 /), (/2, 2/) )
integer, dimension(2, 2):: C; integer :: X = 2
C = X*(A+B)
I=1; print*, (A(I, J), J = 1, 2); I=2; print*, (A(I, J), J = 1, 2)
I=1; print*, (B(I, J), J = 1, 2); I=2; print*, (B(I, J), J = 1, 2)
I=1; print*, (C(I, J), J = 1, 2); I=2; print*, (C(I, J), J = 1, 2)
end
```

Для пояснения текста программы (см. пример 3.62) полезно напомнить, что для инициализации массивов A и B используется функция RESHAPE, позволяющая переформатировать одномерные массивы к требуемой форме. В данном случае эта функция содержит два аргумента (в круглых скобках, через запятую): первым аргументом является конструктор одномерного массива констант, а вторым аргументом − константа, определяющая форму конечного массива (см. п. 3.3.6).

3.5. УПРАВЛЯЮЩИЕ ОПЕРАТОРЫ

3.5.1. Условный оператор и конструкция IF

Условный оператор IF проверяет значение скалярного логического выражения. Если его значение истинно, т.е. имеет значение «.TRUE.» (см. п. 3.3.3), то выполняется действие в виде одиночного оператора (например, вывод на печать, присваивание и т. д.). Обрамляющие круглые скобки для условия обязательны:

IF (условие) одиночный оператор

Оператору IF не могут в качестве действия указываться другие условные операторы и конструкции, а также циклы. Условием может быть логическая константа, переменная или выражение – чаще всего это логические отношения, связанные логическими операторами (см. п. 3.4.3).

В приведенном примере (пример 3.63) определяется знак введенного с клавиатуры целого числа (положительное или отрицательное), или сообщается, что это нуль. В качестве условий используются простейшие логические отношения (см. табл. 3.6).

Пример 3.63. Пример использования логического оператора IF

```
program SIGNDETECT
integer NUMBER
read*, NUMBER
if(NUMBER > 0) print*, “Положительное”
if(NUMBER < 0) print*, “Отрицательное”
if(NUMBER == 0) print*, “Нуль”
end
```

Оператор IF позволяет выполнить одиночный оператор (или исключить его выполнение при невыполнении условия) – это в ряде случаев позволяет обеспечить компактность программы, но явно недостаточно, если в качестве альтернатив используются последовательности более чем из одного оператора. В этом случае целесообразно использовать конструкцию IF.

Первый блок конструкции IF всегда начинается со строки, содержащей ключевое слово IF, за которым в круглых скобках следу-

ет проверяемое условие (логическая величина) и ключевое слово THEN. После THEN в первой строке конструкции IF не может быть ничего, кроме комментария. Если строка – длинная, она может быть продолжена в соответствии с правилами продолжения строк свободного или фиксированного формата. После первой строки конструкции IF следует последовательность операторов, выполняемых при истинности проверяемого условия:

```
IF (условие) THEN
        блок операторов
ELSE IF (альтернативное условие 1) THEN
        альтернативный блок операторов 1
ELSE IF (альтернативное условие 2) THEN
        альтернативный блок операторов 2
…
ELSE IF (альтернативное условие N) THEN
        альтернативный блок операторов N
ELSE
        пост-альтернативный блок
END IF
```

Далее в конструкции IF могут присутствовать число альтернативных блоков ELSE IF со своими условиями – правила их записи точно такие же, как для первого IF-блока. Количество блоков ELSE IF не ограничено – их может быть и десять и сто, а может не быть ни одного.

Затем после блоков ELSE IF может следовать (или отсутствовать) блок ELSE, содержащий программный код, выполняемый в отсутствие истинности всех условий IF и ELSE IF. После чего для завершения конструкции IF используется оператор END IF.

Таким образом, простейшая форма конструкции IF выгляди следующим образом:

```
IF (условие) THEN
        блок операторов
END IF
```

В качестве примера с одним альтернативным блоком в конструкции IF рассмотрим определение четности целого числа, введенного с клавиатуры.

Для определения четности числа используется правило целочисленной арифметики, при делении целых чисел дробная часть отбрасывается. Если четное целое число сначала разделить на два, а затем результат деления опять умножить на два, получится то же самое четное число, что и до деления. Если ту же операцию проделать с нечетным числом, то при делении на два дробная часть будет отброшена и при умножении результата деления на два исходное нечетное число не восстановится (пример 3.64).

Пример 3.64. Пример логической конструкции IF с альтернативой

```
program EVENDETECT
integer NUMBER
logical EVEN
read*, NUMBER
EVEN = NUMBER / 2 * 2 == NUMBER
if(EVEN) then
    print*, “Четное”
else
    print*, “Нечетное”
end if
end
```

Развернутую конструкцию IF можно показать на примере, где одновременно определяется знак и четность целого числа, введенного с клавиатуры (пример 3.65).

Пример 3.65. Пример логической конструкции IF с множеством альтернатив

```
program EVENSIGN
integer NUMBER
logical EVEN
read*, NUMBER
EVEN = NUMBER / 2 * 2 == NUMBER
if(EVEN .and. NUMBER > 0) then
    print*, “Четное положительное”
else if(EVEN .and. NUMBER < 0) then
    print*, “Четное отрицательное”
else if((.not. EVEN) .and. NUMBER > 0) then
    print*, “Нечетное положительное”
else if((.not. EVEN) .and. NUMBER < 0) then
```

```
    print*, “Нечетное отрицательное”
else
    print*, “Нуль”
end if
end
```

Для конструкций IF допускается произвольная вложенность при условии, что вложенные элементы не пересекаются.

3.5.2. Оператор варианта – конструкция CASE

Подобно конструкции IF конструкция CASE позволяет выполнять тот или иной блок операторов, в зависимости от определенного условия. Условие выбора задается значением скалярного выражения целого, логического или текстового типа. Выражение, определяющее условие выбора, должно быть записано в круглых скобках в операторе SELECT CASE:

SELECT CASE (выражение)
CASE селектор 1
блок операторов 1
CASE селектор 1
блок операторов 1
…
CASE селектор N
блок операторов N

CASE DEFAULT
блок операторов
END SELECT

Для выбора блока операторов необходимо, чтобы *селектор* оператора CASE этого блока содержал элемент, совпадающий по значению с выражением в SELECT CASE. Общая форма селектора – заключенный в скобки список неперекрывающихся значений и интервалов того же типа, что и выражение, определяющее условие выбора блока исполняемых операторов.

В качестве примера (пример 3.66) можно запрограммировать алгоритм определения четности целых чисел (тот же алгоритм, что и в примере 3.64).

Селекторы могут задаваться списком, диапазоном или комбинировано (диапазон, как элемент списка). При этом диапазоны значений селекторов не должны пересекаться в пределах одной конструкции SELECT CASE (пример 3.67).

Пример 3.66. Пример логической конструкции IF с альтернативой

```
program EVENCASE
integer NUMBER
logical EVEN
read*, NUMBER
EVEN = NUMBER / 2 * 2 == NUMBER
select  case (EVEN)
    case (.TRUE.)
        print*, “Четное”
    case (.FALSE.)
        print*, “Нечетное”
end select
end
```

Пример 3.67. Задание селекторов в виде списка и закрытого диапазона

```
program DETECTSYM
character SYMBOL
read*, SYMBOL
select case (SYMBOL)
    case(“–” , “+”, “/”, “*”)  ! Селектор в виде списка
        print*,'Арифметический оператор'
    case(“0”: “9”)  ! Селектор в виде закрытого диапазона значений
        print*,'Цифра'
    case default
        print*,'Неизвестный символ'
end select
end
```

Селекторы могут задаваться списком и закрытым диапазоном, а в примере 3.68 значения селектора задаются как закрытым, так и открытым диапазоном.

Закрытый диапазон значений задается своей левой и правой границами, разделенными двоеточием. Соответственно, в диапазоне, открытом слева, отсутствует левая граница, а в диапазоне, открытом справа, – правая граница. По своему смыслу открытые

диапазоны аналогичны логическим отношениям «меньше либо равно» и «больше либо равно».

Конструкция CASE может содержать не обязательный блок, начинающийся строкой CASE DEFAULT. В этот блок входят операторы, выполняемые, если все прочие CASE-блоки не сработали. Такой блок не обязательно должен быть последним.

Пример 3.68. Задание селекторов в виде закрытого и открытого диапазона

```
program SUBSTEMP
!Состояние субстанции, как функция температуры.
integer TEMPERATURE, ABSNUL, FLAME
parameter(ABSNUL = –273, ICE = 0, FLAME = 1000000)
read*, TEMPERATURE
select case (TEMPERATURE)
    case(:–273)
        print*, “Давно замерзло все, что можно”
    case(ABSNUL:ICE)
        print*, “Лед”
    case(FLAME:)
        print*, “Пламень”
    case default
        print*, “Туман и сырость”
end select
end
```

3.5.3. Циклы – разновидности конструкции DO

При решении математических задач, связанных с вычислением последовательностей, полиномов и рядов, возникает необходимость вычисления одинаковых по структуре выражений, зависящих от номера элемента, а также суммирования этих выражений, вычисления произведений и т.д. Для программной реализации таких алгоритмов практически во всех императивных языках программирования предусмотрены циклические конструкции. В Фортране 90/95 для этих целей используется конструкция DO.

Наиболее часто используются циклы в виде конструкции DO с фиксированным числом повторений, имеющие общий вид:

```
DO  переменная цикла = нач_знач, кон_знач, шаг_изм
        блок операторов
END DO
```

В операторе DO указывается *переменная цикла*, которой присваивается *начальное значение* и через запятую *конечное значение* и *шаг* изменения переменной цикла. В строках, следующих за оператором DO, записываются операторы, выполняемые в цикле. Завершается конструкция цикла оператором END DO.

Переменная цикла – скалярная целая переменная, а ее начальное, конечное значения и шаг изменения должны быть скалярными целыми выражениями. Если шаг изменения переменной цикла равен единице, то его можно не записывать, т.е. первая строка конструкции DO будет выглядеть как:

DO переменная цикла = нач_знач, кон_знач

Число итераций цикла может быть вычислено по формуле:

(кон_знач – нач_знач + шаг_изм)/ шаг_изм)

Если результат меньше нуля, цикл игнорируется. На практике, чтобы цикл был рабочим (не вырожденным) необходимо чтобы при положительном шаге конечное значение переменной было больше начального, а при отрицательном шаге – наоборот: начальное больше конечного.

Следует особо отметить, что в первой строке цикла оператор DO должен быть единственным оператором.

В качестве простейшего примера использования конструкции DO можно рассмотреть вычисление суммы целых чисел от единицы до N (пример 3.69). Здесь переменная цикла I изменяется от единицы до N (введенного с клавиатуры) с шагом «единица» (можно не писать). На каждой из N итераций цикла к изначально нулевой сумме (переменная SUM) прибавляется очередное число.

Пример 3.69. Вычисление суммы целых чисел от единицы до N

```
program SUMN1
integer :: SUM = 0 !Инициализация суммы
read*, N
do I = 1, N
    SUM = SUM + I
end do
print *, “SUM=”, SUM
end
```

Этот пример можно модифицировать несколькими способами.

1. Просуммировать числа от N до единицы, заменив «DO I = 1, N» на «DO I = N, 1, –1».
2. Просуммировать только нечетные числа от единицы до N, заменив «DO I = 1, N» на «DO I = 1, N, 2».
3. Просуммировать только четные числа от единицы до N, заменив «DO I = 1, N» на «DO I = 2, N, 2».
4. Вычислить произведение чисел от единицы до N (факториал числа N, обозначаемый N!), инициализировав переменную SUM не нулем «SUM = 0», а единицей «SUM = 1» и заменив накопление суммы чисел «SUM = SUM + I» накоплением произведения «SUM = SUM * I», можно поменять и имя переменной в соответствии со смыслом новой задачи.

В нотации Фортран 77 (поддерживаемой Фортраном 90/95) цикл начинается оператором DO, в котором после ключевого слова DO сначала указывается метка (например, «100») последнего оператора тела цикла, а затем, как и в конструкции DO, переменная цикла, диапазон ее изменения и шаг. Далее следует тело цикла, завершающееся оператором, помеченным меткой, указанной в операторе DO. Как правило, меткой помечается «пустой» (не вызывающий какого-либо действия) оператор CONTINUE (пример 3.70). Отличительной особенностью цикла DO в Фортране 77 является то, что переменная цикла, ее границы и шаг могут быть не только целыми, но и вещественными, однако от такой экзотической особенности в последующих стандартах было решено отказаться.

Пример 3.70. Вычисление суммы целых чисел в Фортран 77

```
program SUMN2
integer :: SUM = 0 !Инициализация суммы
read*, N
do 100 I = 1, N
    SUM = SUM + I
100 continue
print *, “SUM=”, SUM
end
```

В стандарты Фортран 90/95 была внесена и тут же отнесена к избыточным и не рекомендованным свойствам языка циклическая конструкция с предусловием DO WHILE, имеющая общий вид:

DO WHILE (лог_выр)
блок операторов
END DO

Если при входе в цикл логическое выражение в условии DO WHILE имеет значение ИСТИНА (.TRUE.), то выполняется первая итерация цикла. Проверка осуществляется перед каждой следующей итерацией, и как только выражение будет иметь значение ЛОЖЬ (FALSE), итерации прекращаются. В качестве иллюстрации с помощью конструкции DO WHILE проведено вычисление суммы чисел (пример 3.71), аналогичное примерам 3.69 и 3.70.

Пример 3.71. Вычисление суммы целых чисел с помощью DO WHILE

```
program SUMN3
integer :: SUM = 0, I=0        !Инициализация суммы и переменной цикла
read*, N
do while(I < N)
    I=I+1
    SUM = SUM + I
end do
print *, “SUM=”, SUM
end
```

Причина отказа от конструкции DO WHILE связана с возможными проигрышами в оптимизации программного кода, проблема подробно описана в гл. 10 работы: Optimizing Supercompilers for Supercomputers, M. Wolfe (Pitman, 1989).

Для управления работой цикла в зависимости от условий рекомендуется конструкция (по сути реализующая бесконечный цикл):

DO
блок операторов
END DO

с включением в тело цикла операторов IF – для контроля условий, а также операторов EXIT – для выхода из текущего цикла и операторов CYCLE – передающих управление на END DO текущего цикла и, таким образом, инициирующих начало следующей итерации цикла. Операторы EXIT и CYCLE, как правило, используются в составе условного оператора IF.

Вычисление суммы целых чисел от 1 до N в вариантах с предусловием и постусловием представлено в примерах 3.72 и 3.73. Прекращение работы цикла осуществляется оператором EXIT.

Пример 3.72. Вычисление суммы целых чисел в цикле с предусловием

```
program SUMN4
integer :: SUM = 0, I=0        !Инициализация суммы и переменной цикла
read*, N
do
    if (I == N) exit
    I=I+1
    SUM = SUM + I
end do
print *, “SUM=”, SUM
end
```

Пример 3.73. Вычисление суммы целых чисел в цикле с постусловием

```
program SUMN5
integer :: SUM = 0, I=0        !Инициализация суммы и переменной цикла
read*, N
do
    I=I+1
    SUM = SUM + I
    if (I == N) exit
end do
print *, “SUM=”, SUM
end
```

Действие оператора CYCLE можно наблюдать (пример 3.74) при селекции (отборе) и выводе на экран только четных чисел из диапазона от единицы до N. Если на очередной итерации переменная цикла I будет иметь нечетное значение, то оператор CYCLE сразу передает управление на END DO, минуя оператор PRINT и тем самым игнорируя печать нечетных чисел.

Пример 3.74. Печать четных чисел в диапазоне от 1 до N

```
Program EVENPRINT
read*, N
do I = 1, N
    if (I / 2 * 2 /= I) cycle
    print *, I
end do
end
```

Обзор циклических конструкций Фортрана и их возможностей будет не полным без упоминания о работе с массивами – это одно из наиболее частых применений циклов. В качестве иллюстраций рассмотрим поиск максимального элемента в одномерном массиве (пример 3.75) и транспонирование двумерной квадратной матрицы размером с экстентами равными N (пример 3.76).

Пример 3.75. Поиск максимального элемента в одномерном массиве

```
program ARRMAX
integer, parameter :: N = 5
integer, parameter, dimension(N):: ARR = (/ 1, 4, 5, 2, 3/)
MX = ARR(N – 1) ! Разгонное значение для максимального элемента
do I = 1, N
    if( ARR(I) > ARR(MX)) MX = ARR(I)
end do
print*, NUMAX
end
```

На роль максимального элемента в одномерном массиве (см. пример 3.75) сначала назначается любой элемент массива, например предпоследний (с номером N – 1). Затем назначенное максимальное значение MX поочередно сравнивается со всеми остальными элементами массива в цикле, перебирающем номера элементов. Если на очередной итерации цикла найдется элемент массива больший MX, то в переменную MX запишется новый максимум. Таким образом, по завершении цикла в переменной MX зафиксируется значение максимального элемента массива.

Транспонирование квадратной матрицы (N строк и N столбцов) заключается в том, чтобы поменять местами строки и столбцы матрицы – первая строка должна стать первым столбцом, вторая строка – вторым столбцом – и так вплоть до последней строки. Для хранения такой матрицы потребуется массив ранга два (двумерный массив) с экстентами (протяженностями по каждому измерению), равными N. Исходная и транспонированная матрицы хранятся в разных массивах: M – исходная матрица, T – транспонированная матрица (см. пример 3.76). Матрица M инициализируется специализированным конструктором одномерных массивов (см. пример 3.23), переформатированным функцией RESHAPE (см. пример 3.28). Для работы с двумерными матрицами, в частности для их транспонирования, обычно используется конструкция из двух вло-

женных циклов: внешний цикл осуществляет последовательный перебор строк, а внутренний – перебор столбцов. Сама процедура транспонирования заключается в перекрестном изменении номеров строк и номеров столбцов, т.е. индекс I, являющийся номером строки в матрице M, становится номером столбца в матрице T, и наоборот. Последующие циклы используются для вывода на экран исходной и транспонированной матрицы.

В примере 3.76 применена конструкция из двух вложенных циклов. В общем случае в Фортране допускается произвольная вложенность циклов, при этом пересечение циклов недопустимо.

Пример 3.76. Транспонирование квадратной матрицы

```
program TSPMATR
integer , parameter ::N=2
integer, dimension(N, N) :: M=RESHAPE( (/( I, I = 1, N*N)/), (/N, N/) ), T
do I = 1, N
do J = 1, N
    T (J, I) = M (I, J)
end do
end do
print*, “Матрица M”
do I = 1, N
    print*, (M(I, J), J = 1, N)
end do
print*, “Матрица T”
do I = 1, N
    print*, (T(I, J), J = 1, N)
end do
end
```

3.5.4. Оператор GO TO

Большинство критиков Фортрана внушают своим слушателям, что программы на этом языке состоят сплошь из операторов с метками и безусловных переходов GO TO между ними. Читатель, ознакомленный с предыдущим материалом, может по достоинству оценить подобных «специалистов» с их представлениями о Фортране, соответствующими 60-м годам прошлого столетия. При программировании на современных языках высокого уровня (с высоким уровнем абстракции данных) скачкообразные переходы между

блоками программы используются крайне редко, можно сказать только в экстренных случаях, но все-таки они необходимы.

В Фортране 90/95 оператор GO TO или (GOTO) имеет вид:

GO TO метка

где *метка* – метка оператора, соответствующая правилам записи программы в свободном формате (см. п. 3.1.4, примеры 3.13 и 3.14).

Один из немногих случаев, в которых необходим GO TO, – это выход из внутреннего цикла в конструкции из нескольких вложенных циклов. Например, при обнаружении искомого элемента в многомерном массиве CUBE следует прекратить поиск, что означает выход из структуры вложенных циклов (пример 3.77).

Пример 3.77. Поиск элемента в трехмерном массиве

```
program FINDELM
integer , parameter ::N=3
integer, dimension(N, N, N):: CUBE=RESHAPE((/( I, I = 1, N*N*N)/), (/N, N, N/))
read*, ELM                    !Введите целое число от 1 до 27
do I = 1, N
    do J = 1, N
        do K = 1, N
            if (CUBE(I, J, K) .eq. ELM) goto 10
        enddo
    enddo
enddo
10 continue
print*, I, J, K
end
```

Оператор EXIT позволяет выйти только из внутреннего цикла в предшествующий цикл, поэтому в данном случае логично воспользоваться оператором GO TO.

3.6. ВВОД/ВЫВОД ДАННЫХ

3.6.1. Простейшие операции ввода/вывода

Любая работающая программа преобразует исходные данные в конечный результат. Исходные данные считываются из файлов,

ассоциированных с внешними устройствами, а затем обрабатываются и записываются в другие внешние файлы. В Фортране каждое устройство и файл, с которым взаимодействует программа, в смысле ввода/вывода имеет идентификационный номер. Например, номер стандартного устройства ввода – клавиатуры, обычно «5», а номер стандартного устройства вывода – экрана монитора, обычно «6». Для устройства ввода/вывода по умолчанию в качестве номера используется «*» – символ «звездочка». Обычно устройствами ввода/вывода по умолчанию являются экран и клавиатура, хотя в зависимости от настроек вычислительной системы такими устройствами может оказаться все, что угодно – от пылесоса до крылатой ракеты, если они управляются данным компьютером.

Для ввода данных в Фортране предусмотрен оператор READ, а для вывода – оператор WRITE. Для вывода данных на стандартное устройство чаще используется оператор PRINT, поскольку формат его записи предполагает, что устройство вывода предопределено.

Входные и выходные данные всегда имеют определенный *вид* или *формат* представления. Для правильного восприятия информации, как человеком, так и компьютером, имеет значение, представлены ли целые переменные со значениями «один», «два» и «три» на экран в виде «123» или «1^^2^^3» (через пробелы). Поскольку при написании программы внимание разработчика чаще всего сосредоточено на правильной реализации вычислительного алгоритма, то для ввода/вывода данных предусмотрен формат по умолчанию, обозначаемый «*» (звездочкой).

Таким образом, умолчание в отношении ввода/вывода данных, подразумевает соглашение относительно *стандартного устройства* и *формата* ввода и вывода данных – именно этот вариант ввода/вывода используется в большинстве примеров программ данного учебного пособия. Соответствующие операторы Фортрана приведены в табл. 3.8.

Таблица 3.8. Варианты операторов ввода/вывода под управлением списка

Оператор	Действие
READ (*,*) *список ввода*	Ввод с клавиатуры
READ *, *список ввода*	Ввод с клавиатуры
WRITE (*,*) *список вывода*	Вывод на экран
PRINT *, *список вывода*	Вывод на экран

В операторах READ(*,*) и WRITE(*,*) в круглых скобках через запятую записаны две «звездочки», первая означает выбор устройства по умолчанию. Если не оговорено иное, то для ввода по умолчанию используется клавиатура, а для вывода – экран монитора. Вторая «звездочка» имеет тот же смысл, что и единственная «звездочка» в операторах READ* и WRITE* – она означает выбор правил «по умолчанию» в отношении формата ввода или вывода данных. Это сигнал компилятору, что программисту лень или некогда думать о формате представления результатов, и ему (компилятору Фортрана) придется взять это на себя. Формат ввода/вывода по умолчанию будет зависеть от типа вводимых или выводимых элементов, записанных в виде *списка*, проще говоря перечисленных через запятую, поэтому такой вариант ввода/вывода еще называют *вводом/ выводом под управлением списка*.

Список ввода/вывода начинается сразу после закрытия круглых скобок для READ(*,*) и WRITE(*,*) или через запятую после единственной «звездочки» для READ* и WRITE*. В списке могут присутствовать скаляры и массивы всех встроенных и производных типов, включая переменные, буквальные и именованные константы и выражения.

Наиболее распространен вывод данных под управлением списка, когда числовые данные сопровождаются текстом в виде текстовых буквальных констант (пример 3.78).

Пример 3.78. Простейший вывод данных под управлением списка

```
program DATAOUT
integer, parameter :: N=3
real:: X=1., Y=2., Z=3.
print*, “Вывод значений  N=”, N, “переменных:”
print*, “X=”, X, “Y=”, Y, “Z=”, Z
end
```

Вывод данных под управлением списка визуально понятен, но не слишком презентабелен:

```
Вывод значений  N=          3 переменных:
X=  1.0000000       Y=      2.0000000       Z=      3.0000000
```

Для управления отступами, пробелами и количеством знаков необходимо использовать форматный вывод данных (см. далее).

В отношении рассмотренного примера необходимо сделать несколько замечаний. Начинающие программисты, приступающие к программированию на Фортране, иногда считают, что при выводе переменной X перед ней нужно обязательно писать «X=», что здесь есть какая-то связь. Смысловая связь действительно существует, но она в голове программиста, а не в списке оператора PRINT, которому безразлично, будет ли это текст «X=» или «X равно» или же это будет «Hello, World!» или не будет вообще никакого текста. Оператор PRINT просто выводит на экран все элементы, указанные ему в списке вывода, и текстовая буквальная константа «X=» – такой же равноценный и независимый элемент этого списка, как имя вещественной переменной X.

При вводе данных под управлением списка (пример 3.79) после запуска программы у начинающих программистов порой складывается впечатление, что компьютер «завис» – на темном экране виден только мигающий курсор, но это и есть приглашение к вводу данных. Данные вводятся подряд, в одну строку, через пробел или через запятую. При этом количество вводимых данных в списке ввода должно соответствовать количеству данных, введенных с клавиатуры.

Пример 3.79. Простейший ввод данных под управлением списка

```
program DATAIN
integer:: X
integer, dimension (2, 2) :: A
read*, X, A
print*, “X=”, X, “A=”, A
end
```

В представленном примере один оператор READ осуществляет ввод с клавиатуры скалярной переменной X и массива A из четырех элементов. Соответственно, после запуска программы необходимо ввести с клавиатуры пять целых чисел через пробел или запятую. Также можно подтверждать ввод каждого числа клавишей «Enter», при этом для завершения ввода необходимо пять чисел, только тогда оператор READ завершит свою работу.

Объекты производных типов данных вводятся и выводятся как последовательность значений элементов, составляющих структуру объекта. О вводе/выводе массивов речь пойдет далее (п. 3.6.3).

3.6.2. Форматный ввод/вывод данных

Форматный ввод/вывод данных позволяет управлять шириной полей ввода/вывода, количеством значащих цифр в числовых данных, а также отступами, пробелами, табуляцией, переводами строк и другими параметрами представления, т.е. форматом данных при их вводе и выводе.

Таблица 3.9. Варианты операторов форматного ввода/вывода

Оператор	Действие
READ (*, ‘(*fmt)’*), *список ввода*	Ввод с клавиатуры
READ (*, *метка*), *список ввода*	Ввод с клавиатуры
READ ‘(*fmt)’*, *список ввода*	Ввод с клавиатуры
READ, *метка*, *список ввода*	Ввод с клавиатуры
WRITE (*,‘(*fmt)’*), *список вывода*	Вывод на экран
WRITE (*, *метка*), *список вывода*	Вывод на экран
PRINT ‘(*fmt)’*, *список вывода*	Вывод на экран
PRINT, *метка*, *список вывода*	Вывод на экран
Метка FORMAT(*fmt*)	Контейнер спецификаций

Операторы форматного ввода/вывода (табл. 3.9) точно такие же, как и в табл. 3.8, за исключением спецификации формата *fmt*, или ссылки на нее через метку оператора FORTMAT – контейнера форматных спецификаций.

Спецификация формата *fmt* (см. примеры далее) представляет собой список дескрипторов формата данных (табл. 3.10).

Дескрипторы, используемые для описания формата данных (см. табл. 3.10) позволяют задавать символьную длину поля *w* для размещения объекта данных при вводе выводе и учитывать особенности числовых и нечисловых типов данных Фортрана.

При использовании непосредственно в операторах ввода/вывода ссылка на формат представляется как текстовая константа: ‘*(fmt)’* или «*(fmt)*», содержащая спецификацию формата *fmt*, заключенную в круглые скобки. Такой подход рекомендуется, если список дескрипторов в спецификации формата достаточно короткий.

Таблица 3.10. Дескрипторы формата данных

Дескриптор	Тип данных	Внешнее представление
Iw[.m]	Целый тип	Десятичное представление целого числа в поле шириной **w** символов. Величина **m** определяет минимальное число цифр при выводе, недостающие позиции заполняются нулями слева
Bw[.m] **Ow[.m]** **Zw[.m]**	Целый тип	Двоичное, восьмеричное и шестнадцатеричное представление целого числа, выводится аналогично **Iw[.m]**
Fw.d	Вещественный тип	Представление вещественных чисел без десятичной экспоненты (степени числа десять). Задается ширина поля **w** и количество цифр **d** после десятичной точки
Ew.d[Ee]	Вещественный тип	Представление вещественного числа с десятичной экспонентой. Обязательно задается ширина поля **w** и количество цифр **d** после десятичной точки. Дополнительно можно задать **e** – ширину поля порядка десятичной экспоненты
EN.d	Вещественный тип	Аналогичен **E**. Десятичный порядок кратен трем. Целая часть числа в диапазоне от 1 до 1000
ES.d	Вещественный тип	Аналогичен **E**. Целая часть числа в диапазоне от 1 до 10. Масштабный множитель не действует
–	Комплексный тип	Задается парой дескрипторов **F**, **E**, **EN** или **ES**
Lw	Логический тип	Представление логических данных. Задается ширина поля **w** для вывода в минимальном варианте T и F, а в максимальном варианте .TRUE. и .FALSE
A[w]	Текстовый тип	Можно задавать символьную ширину поля **w**. Если ширина поля не указана, то определяется фактическим количеством символов в тексте
Gw.d[Ee]	Все встроенные типы	Универсальный дескриптор. Применяется для вещественных чисел, порядок величины которых заранее не известен: работает как **F** или **E**. Для целого, логического и текстового типа работает как **Iw**, **Lw** и **Aw**
–	Производные типы	Задается дескрипторами в соответствии с встроенными типами элементов структуры производного типа

Если же список дескрипторов в спецификации формата имеет большую длину, сложную структуру, а также содержит текстовые константы с внутренними апострофами или кавычками (буквальные константы текстового или символьного типа – п. 3.3.3), то целесообразно использовать меточную ссылку на оператор FORMAT.

При использовании любого из вышеперечисленных дескрипторов значение переменной считывается из поля шириной *w*, а при вводе записывается в поле такой же ширины, со смещением к правому краю. При этом знак числа для числовых данных (плюс или минус) учитывается как одна позиция поля.

Для дополнения целого числа начальными нулями при выводе используется спецификатор *m* , определяющий минимальное количество цифр в записи целой величины.

Пример 3.80. Неправильный формат при вводе

```
program FMT1
read '(i1, i2, i3)', K, L, M ! Предлагается ввести через пробел 1^23^456
print '(i1, i2, i3)', K, L, M
end
```

При вводе с клавиатуры форматный ввод используется редко, поскольку требует крайней аккуратности (пример 3.80), т.е. сравнение ввода и вывода необходимо пояснить:

```
Ввод:    1^23^456
Вывод:   1^2^34
```

По спецификации «I1» считывается одна символьная позиция – первая единица, как целое число, соответственно переменная «K» получает значение «1», далее по спецификации «I2» считывается два символа «^2» в предположении, что это целое число и переменной «L» присвоится значение «2». И наконец, по спецификации «I3» будут считаны еще три позиции: «3^4» – компилятор Gfortran исключает все пробелы, и переменная «L» инициализируется как «34», но возможна ситуация, когда будут считаны символы только до пробела и тогда в «L» запишется целое значение, равное трем.

При выводе также нужно считать символьные позиции, чтобы получить приемлемый результат (пример 3.81). Первый PRINT выведет «1234» без пробелов.

Пример 3.81. Расстановка пробелов при выводе

```
program FMT2
K=1; L=2; M=3; N=4
print '(4i1)', K, L, M, N
print '(4i2)', K, L, M, N
print 10, K, L, M, N
10  format(4(1x,i1))
end
```

Второй и третий PRINT дадут одинаковый результат «1^2^3^4». Во втором PRINT это происходит за счет избыточной ширины поля и смещения выводимого результата к правой границе. Третий PRINT по метке «10» ссылается на оператор FORMAT. В спецификации, описанной оператором FORMAT, между целыми числами введен отступ в один пробел: «1X». В обоих случаях использован коэффициент повторения: «4I2» и «4(1X,I1)» вместо «I2, I2, I2, I2», и еще более длинной записи: «1X, I1, 1X, I1, 1X, I1, 1X, I1,».

Принципы формирования спецификаций формата с использованием коэффициентов повторения, разделения пробелами и оператора FORMAT универсальны для всех дескрипторов (см. табл. 3.10). Далее приводятся некоторые особенности спецификаций формата для различных дескрипторов.

Пример 3.82. Вывод одного и того же числа в разных форматах

```
program FMT3
X=12345
print '(f6.0)', X                ! 12345.
print '(g10.5)', X    ! 12345.
print '(f7.1)', X                ! 12345.0
print '(e10.5)', X    ! .12345E+05
print '(e11.5)', X    ! 0.12345E+05
print '(e11.5e3)', X ! .12345E+005
print '(en10.3)', X   ! 12.345E+03
print '(es10.4)', X   ! 1.2345E+04
end
```

При использовании дескриптора F одна позиция в поле вывода должна быть зарезервирована под десятичную точку, а применение дескриптора E помимо этого требует четыре позиции под экспоненциальную часть, если число позиций под степень десятичной экспоненты не задано явно (пример 3.82).

Целая часть числа при выводе под управлением дескриптора E равна нулю, что не всегда удобно. Сдвинуть десятичную точку на k порядков вправо, уменьшив настолько же показатель десятичной экспоненты, позволяет масштабный множитель kP (пример 3.83).

Пример 3.83. Использование масштабирующего множителя

```
program FMT4
X=0.76543E+12; Y=0.34567E+21
print '(e11.5)', X                  ! 0.76543E+12
print '(2p, e11.5)', X                        ! 76.543E+10
print '(2p, 2(2x,e11.5)', X, Y ! 76.543E+10^^34.567E+19
end
```

Масштабный множитель является элементом списка спецификации формата и отделяется от последующих дескрипторов запятой. При этом он действует на все последующие дескрипторы E, F и G до появления в списке следующего масштабного множителя. Действие масштабного множителя не распространяется на дескрипторы EN и ES.

При выводе данных непосредственно в спецификацию формата может включаться текст в виде буквальных констант (пример 3.84) что в ряде случаев позволяет разгрузить оператор вывода данных, используя оператор FORMAT.

Пример 3.84. Дескрипторы текстовых строк в спецификации формата

```
program FMT5
integer :: X=1, Y=2, Z=3
print '(1x, "X=", i1, 1x, "Y=", i1, 1x, "Z=", i1)', X, Y, Z
print 10, X, Y, Z
print 20, X, Y, Z
10  format(1x, "X=", i1, 1x, "Y=", i1, 1x, "Z=", i1)
20  format(1x, "X=", i1/ 1x, "Y=", i1/ 1x, "Z=", i1)
end
```

Появление в спецификации формата прямого слеша «/» (оператор FORMAT с меткой «20» в предыдущем примере) приводит к переводу строки, несколько слешей подряд инициируют соответствующее количество переводов строки.

Необходимо заметить, что после выполнения операторов вывода данных WRITE и PRINT сразу же происходит перевод строки. Для предотвращения этого в конце спецификации формата необхо-

димо поставить символ $ (знак денежной единицы), не отделяя его запятой от предыдущего списка спецификации формата. Это может потребоваться, если после вывода на экран требования ввода данных курсор оставался на месте (пример 3.85).

Пример 3.85. Исключение перевода строки

```
program FMT6
print'("X="$)'
read *, X
print'("X=", g12.7)', X
end
```

Форматный ввод/вывод логических и текстовых данных полностью аналогичен вышеизложенным правилам и примерам, поэтому не должен вызывать затруднений.

3.6.3. Неявные циклы и ввод/вывод массивов

Неявные циклы аналогичны конструкции DO с фиксированным числом повторений (см. п. 3.5.3), но не являются самостоятельными конструкциями, поскольку могут быть использованы только в операторах ввода/вывода, как правило, для работы с массивами.

Для вывода двумерного массива на экран в виде матрицы можно использовать конструкцию DO с фиксированным числом повторений и ухищрение в спецификации формата вывода, когда все элементы строки с номером выводятся без перевода строки (запрет переводить строку контролирует символ «$» в конце спецификации формата). Перевод строки осуществляется за счет пустого оператора PRINT за пределами внутреннего цикла (пример 3.86).

При использовании для той же задачи вложенных неявных циклов программа становится более компактной. Неявный цикл представляет собой заключенную в круглые скобки конструкцию, в которой сначала записано выражение, зависящее от переменной цикла (в данном случае элемент массива), и через запятую записывается переменная цикла, диапазон пробегаемых ею значений и шаг (в точности, как в конструкции DO с фиксированным числом повторений).

Пример 3.86. Печать таблицы с помощью конструкции DO

```
program ARRDO1
integer, parameter:: N = 3
integer, dimension(N, N) :: DIGITS = RESHAPE( (/1, 4, 7, 2, 5, 8, 3, 6, 9/), (/3,3/) )
do I = 1, N
do J = 1, N
print '(1x,i1 $)',DIGITS(I,J)
end do
print*,''
end do
end
```

Записанный неявный цикл может служить выражением, для последующего внешнего цикла, т. е. через запятую записывается переменная внешнего цикла со своими границами и шагом и все это заключается в круглые скобки (пример 3.87).

Пример 3.87. Печать таблицы с помощью неявного цикла для ввода/вывода массивов

```
program ARRDO2
integer, parameter:: N = 3
integer, dimension(N, N) :: DIGITS = RESHAPE( (/1, 4, 7, 2, 5, 8, 3, 6, 9/), (/3,3/) )
print'(3(1x,i1))', ((DIGITS (I, J), J = 1, N ), I = 1, N) ! Неявный цикл
end
```

Чтобы показать, что неявные циклы можно использовать не только для ввода/вывода массивов, но выражений (например, арифметических), можно напечатать все туже квадратную (3x3) таблицу чисел от единицы до девяти (пример 3.88).

Пример 3.88. Печать таблицы с помощью неявного цикла для ввода/вывода массивов и выражений

```
program TBLPRN
integer, parameter:: N = 3
print'(3(1x,i1))', ((3*(I – 1)+J,  J = 1, N ), I = 1, N) ! Неявный цикл
end
```

Использование неявных циклов в конструкторах массивов показано в п. 3.3.6, но в этом случае в неявный цикл не могут быть включены даже простейшие выражения – только переменные.

3.6.4. Файловый ввод/вывод

Ранее рассматривалось взаимодействие со стандартными устройствами: клавиатурой и монитором. Также упоминалось, что доступные компьютеру устройства пронумерованы, и обращение к ним осуществляется через их номера. Так, номер стандартного устройства ввода – клавиатуры, обычно «5», а номер стандартного устройства вывода – экрана монитора, обычно «6». Но поскольку эти устройства используются по умолчанию, для них предусмотрено единое обозначение – символ «звездочка».

Если быть точнее, то внешние устройства и их номера в Фортране всегда относятся к файлам, т.е. программа считывает данные и записывает их в файлы, связанные с устройствами. Например, в UNIX стандартное устройство «5» ассоциируется с файлом потока стандартного ввода *stdin*, а стандартное устройство за номером «6» – с файлом потока стандартного ввода *stdout*. Не зарезервированные номера устройств (а это практически все множество буквальных целых констант) по умолчанию связаны с жестким диском компьютера, т.е. эти номера можно ассоциировать с файлами жесткого диска, обычно для этого хватает первых четырех номеров от «1» до «4».

Пример 3.89. Открытие, запись, чтение и закрытие файлов

```
program FINOUT1
real ::  X=1., Y=2., Z=3.
!Открытие пустого файла, запись данных и закрытие файла
open(1, file= "info.txt")
write(1,*)  X, Y, Z
close(1)
!Открытие существующего файла, чтение данных и закрытие файла
open(1, file='info.txt')
read(1,*)  X, Y, Z
close(1)
!Вывод на экран данных, считанных из файла
print*, X, Y, Z
end
```

Ассоциирование файла с номером устройства – важный технологический момент, предшествующий чтению или записи данных. Сначала внешний файл связывается с номером устройства и от-

крывается оператором OPEN, затем производится чтение или запись данных операторами READ или WRITE. После необходимых операций чтения и записи файл закрывается оператором CLOSE с освобождением номера устройства, и освобожденный номер можно использовать для дальнейших файловых операций (пример 3.89).

В результате работы этой программы в текущей директории появится файл «info.txt» – обычный текстовый файл. Однако использованные умолчания требуют детальных разъяснений.

Оператор OPEN в развернутом виде содержит весьма значительный перечень спецификаторов. Наиболее часто применяемыми из них являются:

- номер устройства: спецификатор UNIT;
- имя файла – спецификатор FILE;
- статус файла – спецификатор STATUS;
- способ доступа к данным – спецификатор ACCESS;
- длина записи – спецификатор RECL;
- представление данных – спецификатор FORM;
- статус ввода/вывода – спецификатор IOSTAT.

Спецификатор UNIT является обязательным, например UNIT=1 в рассмотренном примере. При этом достаточно указать только номер, в данном случае это «1». Можно использовать любые номера, кроме «5» и «6», зарезервированных под стандартные устройства или иных зарезервированных, если таковые имеются.

Спецификатор FILE – текстовое выражение, переменная или константа, содержащая имя файла в формате, предусмотренном операционной системой, в рассмотренном примере – это файл в текущей директории: FILE = «INFO.TXT».

Спецификатор STATUS – статус файла с именем, определенным в спецификации FILE – определяется как скалярная текстовая величина с одним из пяти перечисленных значений:

- OLD – открывается уже существующий файл;
- NEW – создается новый, еще несуществующий файл;
- REPLACE – если файл не существует, то он будет создан, а существующий файл, будет удален и создан новый файл с тем же именем;

- SCRATCH – создается временный файл, который удаляется после закрытия. Для SCRATCH-файла спецификатор FILE не имеет смысла и не используется;
- UNKNOWN – для существующего файла устанавливает статус OLD, а для несуществующего – статус NEW. UNKNOWN назначается по умолчанию, если спецификатор STATUS явно не прописан в операторе OPEN (см. пример 3.89).

Когда статус файла явно не задан, то при компиляции программы иногда выдается соответствующее предупреждение (warning), которое начинающие программисты часто принимают за ошибку.

Спецификатор ACCESS – способ доступа к данным, определяется скалярным текстовым выражением, принимающим одно из двух значений: SEQUENTIAL (файл последовательного доступа) или DIRECT (файл прямого доступа). Файлы прямого доступа состоят из записей одинаковой длины, и к произвольной записи файла можно обратиться по ее номеру. Файлы последовательного доступа могут состоять из записей произвольной длины, а от текущей записи можно перейти только к следующей или предыдущей. Если способ доступа к данным явно не указан (см. пример 3.89), то считается что это файл – последовательного доступа.

Спецификатор RECL – длина одной записи файла (в байтах или машинных словах), актуальна только для файлов прямого доступа.

Спецификатор FORM задается скалярным текстовым выражением и определяет форму представления данных: FORMATTED, соответствующее форматному представлению данных или UNFORMATTED, соответствующее бесформатному представлению данных. Если форма представления не указана, то файл последовательного доступа считается форматным (см. пример 3.89), а файл прямого доступа – бесформатным. Форматный (по сути, текстовый) файл содержит данные в удобочитаемом для человека виде. Эти данные могут считываться и записываться с использованием спецификаций формата. Бесформатный файл не предполагает визуального прочтения и хранит данные во внутреннем машинном представлении.

Спецификатор IOSTAT характеризует состояние ввода/вывода и задается как любая целая скалярная переменная, например IOSTAT= IOS, при этом IOS = 0 означает успешную операцию ввода/вывода, а любое другое значение констатирует ошибку.

Для предъявления оператора OPEN в развернутом виде целесообразно разделить пример 3.89 на две части (примеры 3.90 и 3.91).

Пример 3.90. Оператор OPEN в развернутом виде

```
program FINOUT2
real ::  X=1., Y=2., Z=3.
open(unit=1, file= "info.txt", status="new", access= "sequential",  &
form="formatted", iostat =IOS)
if (IOS /=0) then
print*, "Ошибочка вышла! "
stop                              !Оператор останова программы
end if
write(1,*)  X, Y, Z
close(1)
end
```

Если при запуске программы (см. пример 3.90) в текущей директории присутствует файл INFO.txt с явно указанным для него статусом NEW в операторе OPEN, то программа завершится по ошибке: «IOS > 0». Файл INFO.txt является результатом работы программы и должен появиться после ее запуска.

Пример 3.91. Открытие, запись, чтение и закрытие файлов

```
program FINOUT3
open(unit=4, file= "info.txt", status="old", access= "sequential",  &
form="formatted", iostat =IOS)
if (IOS /=0) then
print*, "Ошибочка вышла! "
stop                              !Оператор останова программы
end if
read(4,*)  X, Y, Z
close(4)
print *, X, Y, Z
end
```

Для корректной работы программы (см. пример 3.91) требуется наличие в текущей директории файла INFO.txt – результата работы предыдущего примера. Если такого файла не окажется, то программа завершится с ошибкой: «IOS > 0», поскольку для файла данных INFO.txt явно указан статус OLD в операторе OPEN.

В операторах READ и WRITE при работе с текстовыми файлами могут использоваться все дескрипторы формата ввода/вывода дан-

ных (см. п. 3.6.3). Однако помимо номера устройства и спецификации формата, операторы READ и WRITE могут содержать и другие спецификаторы, например спецификатор IOSTAT в точности такой же, как для оператора OPEN.

Наличие спецификатора IOSTAT в операторах READ и WRITE позволяет программисту самостоятельно обрабатывать ошибки чтения и записи в штатном режиме, без прерывания работы программы. Это может быть весьма актуально при обработке текстовых файлов с заранее неизвестным количеством текстовых строк, например при копировании одного файла в другой (пример 3.92).

Для работы этого примера в текстовом редакторе потребуется подготовить небольшой текстовый файл с именем FILE1.txt, в котором не менее трех строк, каждая не длиннее 80-ти символов. На выходе программа создаст копию файла FILE1.txt в файле с именем FILE2.txt.

Пример 3.92. Копирование файлов

```
program FINOUT4
character (LEN =80) STRING
open(unit=1, file= "file1.txt")
open(unit=2, file= "file2.txt")
do
read (1, '(A80)', iostat = IOS) STRING
write (2, '(A80)') STRING
if (IOS >0) exit      ! Цикл завершен по достижении конца файла
end do
close(1); close(2)
end
```

При открытии существующего текстового файла оператором OPEN текущая позиция чтения находится перед первой записью файла. Чтение первой записи перемещает позицию к следующей записи и т.д. Записи при открытии нового файла последовательно добавляются в конец файла, где постоянно находится позиция для записи. В связи с этим для файлов последовательного доступа представляет интерес позиционирование внутри файла.

При помощи оператора BACKSPACE можно из любой позиции файла перейти к предыдущей (или только что прочитанной) записи файла, а при помощи оператора REWIND – перейти к началу файла. В качестве параметра оба оператора используют номер ассо-

циированного с файлом устройства, например BACKSPACE 1 или REWIND 4. Если файл уже позиционирован в начало, то применение этих операторов не повлечет за собой ошибки или изменения в позиционировании. Проиллюстрировать возможности позиционирования можно на своеобразном «плеере» файла с элементами «перемотки» записей (пример 3.93). Для работы программы потребуется текстовый файл с именем FILE1.txt, с несколькими строками, не длиннее 80-ти символов.

Пример 3.93. «Перемотка записей» текстового файла

```
program FINOUT5
integer(kind=1):: ACTION, IOS=0
character (LEN =80) STRING
open(unit=1, file= "file1.txt")
do
    read (1, '(A80)' , iostat = IOS) STRING
    if (IOS /=0) exit              ! Цикл завершен по достижении конца файла
    print '(A80)', STRING     ! Печать строки, считанной из файла
    print '("1 - читать дальше, 2 - шаг назад, 3 - в начало, ", i1 $)'
    read *, ACTION
    if (ACTION ==1) cycle
    if (ACTION ==2) then
        backspace 1; backspace 1
        ! backspace 2 раза, иначе backspace + read из файла = стояние на месте
    endif
    if (ACTION ==3)rewind 1
end do
print *, "Достигнут конец файла"
close(1)
end
```

В заключение стоит отметить, что в любой позиции файла, используя оператор ENDFILE, можно внести запись «Конец файла», при этом все последующие строки файла будут утеряны. В качестве параметра оператору ENDFILE указывается номер устройства, так же как для BACKSPACE и REWIND.

Описание программирования ввода/вывода для файлов прямого доступа (ACCESS = «DIRECT») и бесформатной формой представления данных (FORM = «UNFORMATTED») выходит за рамки данного учебного пособия.

3.7. ПРОГРАММНЫЕ КОМПОНЕНТЫ И ЭЛЕМЕНТЫ ООП

3.7.1. Структура программных компонентов

В простейшем случае программа на Фортране может состоять из одной главной программы, находящейся в файле с соответствующим расширением (для компилятора Gfortran это .f95). При этом она может состоять даже из одного единственного оператора END. Операторы PROGRAMM … END являются операторными скобками, ограничивающими последовательность операторов главной программы**,** при этом оператор PROGRAMM – не обязательный (хотя настоятельно рекомендуемый, особенно для программ, предъявляемым сторонним заказчикам, к которым стоит отнести и преподавателя на зачете).

С учетом синтаксиса Фортрана 90/95, структуру главной программы упрощенно можно представить в виде:

PROGRAM имя программы
операторы описания
исполняемые операторы
CONTAINS
внутренние подпрограммы
END имя программы

К операторам описания относятся объявления скалярных переменных и констант, а также объявление массивов и описание производных типов (п. 3.3). В исполняемые входят операторы ввода/вывода данных, управляющие операторы, присваивания и выражения всех встроенных и производных типов, а также использования программных компонентов.

Оператор CONTAINS используется только при наличии внутренних подпрограмм (п. 3.7.3), а имя главной программы после оператора END так же необязательно, как и оператор PROGRAM.

Программные компоненты подразделяются на *подпрограммы-процедуры* (SUBROUTINE), *подпрограммы-функции* (FUNCTION) и *модули* (MODULE). Каждый модуль (MODULE) должен размещаться в отдельном файле, а расположение и группировка по фай-

лам внешних подпрограмм, как процедур (SUBROUTINE), так и функций (FUNCTION), не являющихся компонентами модулей, не регламентирована, т.е. полностью на усмотрении программиста: в одном файле с главной программой или в отдельных файлах При этом исходные файлы, содержащие внешние подпрограммы и модули, должны иметь расширение .f95 (для Gfortran).

Все программные компоненты Фортрана, т.е. подпрограммы-процедуры, подпрограммы-функции и модули, имеют одинаковую структуру (табл. 3.11).

Таблица 3.11. Структура программных компонентов

<table>
<tr><td colspan="3">Оператор PROGRAM, SUBROUTINE, FUNCTION или MODULE</td></tr>
<tr><td colspan="3">Операторы USE (4. 3.7.4)</td></tr>
<tr><td rowspan="3">Операторы FORMAT</td><td colspan="2">Оператор IMPLICIT NONE</td></tr>
<tr><td rowspan="2">Операторы PARAMETER и DATA</td><td>Операторы IMPLICIT</td></tr>
<tr><td>Операторы описания: объявления переменных и констант, массивов, описание производных типов, интерфейсных блоков модулей</td></tr>
<tr><td colspan="2">Исполняемые операторы</td></tr>
<tr><td colspan="3">Оператор CONTAINS</td></tr>
<tr><td colspan="3">Внутренние или модульные (для MODULE) подпрограммы</td></tr>
<tr><td colspan="3">Оператор END</td></tr>
</table>

3.7.2. Внешние подпрограммы

Внешние подпрограммы служат для выполнения определенной задачи в рамках основной программы, а также используются во избежание повторения одинаковых блоков операторов. В Фортране существует два вида внешних подпрограмм – *подпрограммы-процедуры* и *подпрограммы-функции*. Оба вида подпрограмм имеют практически одинаковую структуру:

SUBROUTINE имя процедуры (список формальных параметров)
описание формальных параметров
описание внутренних параметров
исполняемые операторы

CONTAINS
внутренние подпрограммы
END имя процедуры

или

тип функции FUNCTION имя функции (список формальных параметров)
описание формальных параметров
описание внутренних параметров
исполняемые операторы
CONTAINS
внутренние функции
END подпрограммы-процедуры

Подпрограмма-процедура ограничена операторными скобками (SUBROUTINE … END), а подпрограмма-функция операторными скобками (FUNCTION … END).

После имени подпрограммы в круглых скобках указывается список формальных параметров, которые передаются в подпрограмму при ее вызове. Формальные параметры, переданные в подпрограмму-процедуру, изменяются алгоритмом процедуры и возвращаются в программную компоненту, из которой была вызвана подпрограмма.

Подпрограмма-процедура может вернуть в вызывающую программу только список формальных параметров, для ее вызова используется оператор CALL (пример 3.94).

Пример 3.94. Подпрограмма-процедура, складывающая два числа

```
program SUBPRG1
real :: X=1., Y=2., Z
call SUMAB(X, Y, Z)
print *, Z
end
subroutine SUMAB(A,B,C)
real :: A, B, C
C = A + B
return
end
```

Подпрограмма-функция может использоваться в вызывающей программе аналогично переменной соответствующего типа (пример 3.95), в том числе в выражениях и присваиваниях, для этого функция должна быть объявлена внешней с помощью оператора EXTERNAL. Точно также используется внутри подпрограммы-функции ее имя – переменной соответствующего типа, но перед завершением работы подпрограммы-функции эта переменная должна получить значение, которое вернется в вызывающую программу. При этом подпрограмма-функция, как и подпрограмма-процедура, может изменять значения элементов списка формальных параметров (но этого делать не рекомендуется).

Пример 3.95. Подпрограмма-функция, складывающая два числа

```
program SUBPRG2
external SUM
real :: X=1., Y=2., Z
Z = SUM(X, Y)
print *, Z
end
real function SUM(A,B)
real :: A, B
SUM = A + B
return
end
```

Возврат управления в вызывающую процедуру осуществляется оператором RETURN, а если в силу каких-то причин необходимо прекратить работу программы в целом, то можно воспользоваться оператором STOP.

Если в подпрограмме (процедуре или функции) предполагается обработка массива, то ранги (размерности) массивов в вызывающей и вызываемой программной компоненте должны совпадать, а экстенты (количество элементов по каждому измерению) должны передаваться через список формальных параметров (пример 3.96).

Пример 3.96. Передача массива в подпрограмму

```
program SUBPRG3
integer, parameter :: M=1, N=2
integer , dimension(M,N) :: A, B, C
data A /1, 2/ B /3, 4/
call SUMARR(A, B, C, M, N)
```

```
print *, C
end
subroutine SUMARR(X,Y, Z, K, L)
integer , dimension(K, L) :: X,Y, Z
Z = X+Y
return
end
```

Формальные параметры подпрограммы должны быть объявлены в разделе описания, или к ним будут применены правила по умолчанию для имен переменных. При вызове подпрограммы в список формальных параметров предаются соответствующие фактические параметры. Списки формальных и фактических должны иметь строгое поэлементное соответствие в отношении данных. Скаляру должен соответствовать скаляр того же типа, массиву – массив, а функции – функция.

Одноименные переменные в списках формальных и фактических параметров программ и подпрограмм ни как не связаны друг с другом, поскольку являются внутренними переменными своих программ и подпрограмм (пример 3.97).

Пример 3.97. Независимость имен (A и B) в программных компонентах

```
program SUBPRG4
external SUM
real :: A=1., B=2.
print *,  SUM (A, B)
end
real function SUM(A, B)
real :: A, B
SUM = A + B
return
end
```

3.7.3. Внутренние подпрограммы

Внутренние подпрограммы (процедуры и функции) по структуре и способу применения ничем не отличаются от внешних подпрограмм (см. п. 3.7.2). Однако это не отдельная программная компонента, а составная часть главной программы или внешней подпрограммы. Хотя внутренние подпрограммы не могут иметь собственных внутренних подпрограмм, они имеют доступ ко всем объ-

ектам своего носителя, в том числе могут вызывать его другие внутренние подпрограммы (пример 3.98).

Внутренние подпрограммы располагаются внутри своих носителей (в главной программе или во внешних подпрограммах) последовательно одна за другой.

Последовательность внутренних подпрограмм находится в блоке операторов сразу после оператора CONTAINS и заканчивается оператором END программной компоненты-носителя.

Пример 3.98. Пример внутренней подпрограммы

```
program SUBPRG5
real :: A=1., B=2., C
print *, SUM(A, B)
contains
real function SUM(A, B)
real :: A, B
SUM = A + B
return
end
end
```

3.7.4. Модули как библиотеки производных типов

В Фортране 90/95 *модули* используются для хранения глобальных данных и как контейнер производных типов. В данном учебном пособии рассматривается именно последнее назначение модулей. В контексте использования модуля как контейнера производных типов его структура имеет общий вид:

MODULE имя модуля
описание производных типов
описание интерфейсов внутренние подпрограммы модуля
CONTAINS
внутренние подпрограммы модуля
END имя модуля

В качестве примера рассмотрим создание производного типа VECTOR, обеспечивающего работу с векторами, лежащими на плоскости (x, y), и начало каждого вектора совпадает с началом координат (0, 0). Каждый такой вектор может быть описан конечной

точкой с координатами (*X*, *Y*). В соответствии с правилами векторной алгебры и аналитической геометрии могут быть описаны такие операции над векторами, как умножение вектора на число, сумма (разность) и т.д. В качестве примера реализуем операцию сложения данных типа VECTOR так, чтобы для трех векторов этого типа A, B и C с точки зрения Фортрана была правомерна операция сложения «C=A+B». Это означает, что нужно создать *задаваемую скалярную операцию* сложения с обозначением «+» для *производного типа данных* VECTOR.

Дело в том, что для производных типов данных не существует заведомо определенных операций – доступны операции только с элементами структуры, поскольку они являются данными встроенных типов, и для них определены соответствующие покомпонентные операции, что-то вроде:

C%X = A%X + B%X
C%Y = A%Y + B%Y

Модуль с реализацией типа VECTOR и векторной арифметики (см. пример 3.19) так и называется: VECTOR_ARITHMETIC.

Описание производных типов следует в модулях сразу после заголовка. Тип VECTOR объявлен в соответствии с правилами объявления производных типов (см. п. 3.3.7) и имеет в своей структуре два вещественных компонента – X и Y, или в геометрической интерпретации координаты конца вектора.

Для создания операции сложения для элементов типа VECTOR нужно сделать две вещи – написать функцию, которая будет заниматься покомпонентным сложением, и связать ее с интерфейсом виде знака «плюс» (если не нравится «плюс», можно использовать обозначение любой встроенной операции Фортрана или последовательность букв, не длиннее 31, ограниченную точками, предположительно .PLUS.).

Тип функции сложения (ADD_VECTORS – название произвольное) и возвращаемое ею значение должно быть таким же, как у слагаемых A и B, т.е. VECTOR.

Поскольку операция сложения – бинарная, то входными параметрами функции с атрибутом INTENT(IN) объявлены два входных параметра: A и B.

При описании унарной операции потребуется только один входной параметр.

Функция ADD_VECTORS реализует операцию сложения на компонентном уровне: ADD_VECTORS%X = A%X +B%X и т.д., поскольку компоненты типа VECTOR являются вещественными и для них определена операция сложения вещественных чисел, а интерфейсный блок (INTERFACE ... END INTERFACE) связывает функцию ADD_VECTORS, являющуюся внутренней модульной процедурой, с оператором «плюс».

Любой модуль должен располагаться в отдельном файле, название которого должно совпадать с названием модуля, а расширение зависит от компилятора – для компилятора Gfortran созданный модуль VECTOR_ARITHMETIC должен быть помещен в отдельный файл с именем vector_arithmetic и расширением .f95. Ограничений на имена программ и программных компонентов, использующих модули, не существует.

Пример 3.99. Использование модуля в программе

```
program VECTOR_TEST
use  VECTOR_ARITHMETIC
type (VECTOR) :: A = VECTOR(1., 2.), B = VECTOR(3., 4.), C
C = A + B
print*, A; print*, B; print*, C
end
```

Чтобы использовать модуль в других программе и программных компонентах, необходимо объявить об его использовании при помощи оператора USE (пример 3.99). Подключив таким образом модуль VECTOR_ARITHMETIC, можно объявлять переменные типа VECTOR, не описывая структуру типа, и применять к переменным операции, определенные в модуле.

Два файла – с модулем и тестирующей программой, необходимо правильно скомпилировать для получения запускаемой программы, для этого необходимо внимательно изучить специально посвященный этому п. 3.2.4.

В качестве упражнения и тренировки работы с модулями полезно дополнить созданный модуль VECTOR_ARITHMETIC другими такими стандартными векторными операциями, как скалярное и векторное произведение и т.д.

3.7.5. Встроенные функции Фортрана

НЕКОТОРЫЕ СПРАВОЧНЫЕ ФУНКЦИИ

Функция	Тип функции	Тип аргумента	Действие
selected_int_kind(R)	Стандартный integer	Количество знаков integer числа	Возвращает минимально возможное значение KIND для R – значного целого
selected_real_kind(R, P)	Стандартный integer	Количество знаков и степень real числа	Возвращает минимально возможное значение KIND для R – значного real в степени P
kind(X)	Стандартный integer	Любой встроенный тип	значение KIND аргумента X
huge(X)	Совпадает с типом аргумента	Числовой тип любой разновидности	Модуль максимального значения для чисел данной разновидности типа
tiny(X)	Совпадает с типом аргумента	Числовой тип любой разновидности	Модуль минимального значения для чисел данной разновидности типа

НЕКОТОРЫЕ ФУНКЦИИ ПРЕОБРАЗОВАНИЯ

Функция	Тип функции	Тип аргумента	Действие
abs(X)	integer для целого аргумента и real для остальных	Числовой тип любой разновидности	Возвращает модуль аргумента
aimag(Z)	real с параметром разновидности типа как у аргумента	comlex	Возвращает мнимую часть комплексной величины
aint(X [, KIND])	Совпадает с типом аргумента	real любой поддерживаемой разновидности	Отбрасывает дробную часть числа, не изменяя тип

МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

Функция	Тип функции	Тип аргумента	Действие
anint(X [, KIND])	Совпадает с типом аргумента	real любой поддерживаемой разновидности	Возвращает ближайшее целое, как real
ceiling(X)	integer	real любой поддерживаемой разновидности	Возвращает ближайшее целое справа от X
cmplx(X[, Y])	complex	real любой поддерживаемой разновидности	Преобразует пару чисел real (или одно) в comlex
floor(X)	integer	real любой поддерживаемой разновидности	Возвращает ближайшее целое слева от X
int(X[, KIND]]	integer	real любой поддерживаемой разновидности	Преобразует к целому типу разновидности KIND
nint(X)	integer	real любой поддерживаемой разновидности	Возвращает ближайшее целое
real(X[, KIND]]	real	real и integer любой разновидности	Преобразует к вещественному типу заданной разновидности KIND
acos(X)	Совпадает с типом аргумента	real любой поддерживаемой разновидности	Возвращает значение функции арккосинуса: $\|X\| \leq 1$
asin(X)	Совпадает с типом аргумента	real любой поддерживаемой разновидности	Возвращает значение функции арксинуса: $\|X\| \leq 1$
atan(X)	Совпадает с типом аргумента	real любой поддерживаемой разновидности	Возвращает значение функции арктангенса: $\|X\| \leq 1$
atan2(X,Y)	Совпадает с типом аргумента	real любой поддерживаемой разновидности	Главное значение аргумента комплексного числа
cos(X)	Совпадает с типом аргумента	real и complex любой разновидности	Значение функции косинуса

Окончание таблицы

Функция	Тип функции	Тип аргумента	Действие
cosh(X)	Совпадает с типом аргумента	real любой поддерживаемой разновидности	Значение функции гиперболического косинуса
exp(X)	Совпадает с типом аргумента	real и complex любой разновидности	Значение экспоненциальной функции
log(X)	Совпадает с типом аргумента	real и complex любой разновидности	Значение функции натурального логарифма
log10(X)	Совпадает с типом аргумента	real любой поддерживаемой разновидности	Значение функции десятичного логарифма: X > 0
sin(X)	Совпадает с типом аргумента	real и complex любой разновидности	Значение функции синуса
sinh(X)	Совпадает с типом аргумента	real любой поддерживаемой разновидности	Значение функции гиперболического синуса
sqrt(X)	Совпадает с типом аргумента	real и complex любой разновидности	Значение функции квадратного корня
tan(X)	Совпадает с типом аргумента	real любой поддерживаемой разновидности	Значение функции тангенса
tanh(X)	Совпадает с типом аргумента	real любой поддерживаемой разновидности	Значение функции гиперболического тангенса

4. ТЕМЫ И ЗАДАЧИ ДЛЯ ПРОГРАММИРОВАНИЯ

4.1. ПРОСТЫЕ ИНТЕРПОЛЯЦИОННЫЕ ПОЛИНОМЫ

4.1.1. Интерполяционные полиномы первой степени

Известно, что уравнение прямой на координатной плоскости представляет собой линейную функцию $y = kx + b$, где коэффициенты k и b известны (рис. 4.1).

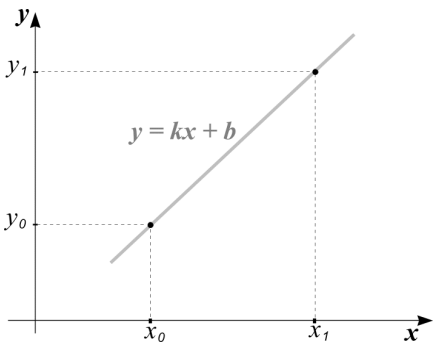


Рис. 4.1. Линейная функция

Теория интерполяции функций рассматривает уравнение прямой как одну из форм записи *интерполяционного полинома* первой степени в канонической форме:

$$C_1(x) = a_0 + a_1 x. \qquad (4.1)$$

Это, в принципе, то же самое школьное уравнение прямой, с заменой $a_0 = b$; $a_1 = k$, но значения a_0 и a_1 неизвестны, и их требуется найти по двум точкам, координаты которых известны.

Пусть на координатной плоскости (X, Y) заданы две точки с известными координатами (x_0, y_0) и (x_1, y_1), где $x_2 > x_1$ (см. рис. 4.1), тогда для нахождения неизвестных коэффициентов a_0 и a_1 потребуется решить систему из двух алгебраических уравнений:

$$\begin{cases} y_1 = a_0 + a_1 x_1; \\ y_2 = a_0 + a_1 x_2. \end{cases} \quad (4.2)$$

Уравнения (4.2) получаются подстановкой известных координат точек в выражение (4.1). Система (4.2) показывает, что для построения или функционального определения прямой необходимо и достаточно иметь на плоскости две точки с известными координатами. Решение системы (4.2) дает значения коэффициентов a_0 и a_1:

$$a_1 = \frac{(y_1 - y_0)}{(x_1 - x_0)}; \qquad a_0 = \frac{(y_0 x_1 - y_1 x_0)}{(x_1 - x_0)}. \quad (4.3)$$

Прямую, проходящую через две известные точки, можно представить полиномом в канонической форме (4.1) а также *интерполяционными полиномами* первых степеней в форме Лагранжа:

$$L_1(x) = y_0 \frac{(x - x_1)}{(x_0 - x_1)} + y_1 \frac{(x - x_0)}{(x_1 - x_0)} \quad (4.4)$$

или в форме Ньютона:

$$P_1(x) = y_0 + \frac{(y_1 - y_0)}{(x_1 - x_0)}(x - x_0). \quad (4.5)$$

Нетрудно убедиться, что $L_1(x) \equiv C_1(x)$:

$$\begin{aligned} \boldsymbol{L_1(x)} &\equiv y_0 \frac{(x - x_1)}{(x_0 - x_1)} + y_1 \frac{(x - x_0)}{(x_1 - x_0)} = \\ &= -\frac{(y_0 x - y_0 x_1)}{(x_1 - x_0)} + \frac{(y_1 x - y_1 x_0)}{(x_1 - x_0)} = \\ &= \frac{(-y_0 x + y_0 x_1 + y_1 x - y_1 x_0)}{(x_1 - x_0)} = \\ &= \frac{x(y_1 - y_0) + y_0 x_1 - y_1 x_0}{(x_1 - x_0)} = \\ &= \frac{y_0 x_1 - y_1 x_0}{(x_1 - x_0)} + \frac{(y_1 - y_0)}{(x_1 - x_0)} x \equiv \boldsymbol{C_1(x)}. \end{aligned} \quad (4.6)$$

По аналогии с (4.6) доказывается, что $P_1(x) \equiv C_1(x)$ и, в конечном счете:

$$C_1(x) \equiv L_1(x) \equiv P_1(x), \quad (4.7)$$

т.е. полином в канонической форме, полином Лагранжа и полином Ньютона – это разные, но абсолютно эквивалентные формы записи одного и того же выражения. В общем случае это справедливо как для полиномов первой степени, так и для полиномов всех целых положительных степеней.

4.1.2. Интерполяционные полиномы второй степени

Перейдем к рассмотрению интерполяционных полиномов второй степени. Из школьной программы известно уравнение параболы, заданной на координатной плоскости, которое представляет собой квадратичную функцию $y = ax^2 + bx + c$, где a, b и c – известные коэффициенты (рис. 4.2).

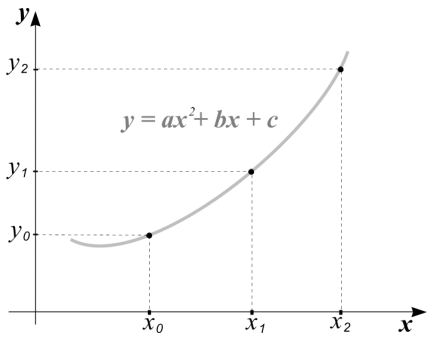


Рис. 4.2. Квадратичная функция

Интерполяционный полином второй степени в канонической форме

$$C_2(x) = a_0 + a_1 x + a_2 x^2 \qquad (4.8)$$

также представляет собой квадратичную функцию, с тем отличием, что коэффициенты при степенях x неизвестны.

Для нахождения неизвестных коэффициентов a_0, a_1 и a_2 необходимо решить систему трех алгебраических уравнений:

$$\begin{cases} y_0 = a_0 + a_1 x_0 + a_2 x_0^2; \\ y_1 = a_0 + a_1 x_1 + a_2 x_1^2; \\ y_2 = a_0 + a_1 x_2 + a_2 x_2^2. \end{cases} \quad (4.9)$$

Система (4.9) может быть получена при наличии трех точек с известными координатами: $(x_0,\ y_0)$, $(x_1,\ y_1)$ и $(x_2,\ y_2)$ путем подстановки значений этих координат в (4.8). Решая систему (4.9) получаем значения коэффициентов:

$$\begin{aligned} a_0 &= \frac{1}{x_1(x_2 - x_0)}\left(x_0 \frac{y_2 x_1^2 - y_1 x_2^2}{x_2 - x_1} - x_2 \frac{y_1 x_0^2 - y_0 x_1^2}{x_1 - x_0}\right); \\ a_1 &= -\frac{1}{x_2 - x_0}\left(\frac{(y_2 - y_1)(x_1 + x_0)}{(x_2 - x_1)} - \frac{(y_1 - y_0)(x_2 + x_1)}{(x_1 - x_0)}\right); \\ a_2 &= \frac{1}{x_2 - x_0}\left(\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0}\right). \end{aligned} \quad (4.10)$$

Если для построения или функционального определения линейной функции (полинома степени $N = 1$) требуется две точки, то для квадратичной функции (полинома степени $N = 2$) будет необходимо и достаточно трех точек. В общем можно показать, что для построения или функционального определения полинома степени n будет необходимо и достаточно $N + 1$ точек.

Помимо полинома в каноническом виде (4.8), квадратичная функция при наличии трех известных точек может быть представлена также *интерполяционными полиномами* второй степени в форме Лагранжа

$$\begin{aligned} L_2(x) = y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + \\ + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \end{aligned} \quad (4.11)$$

или в форме Ньютона:

$$\begin{aligned} P_2(x) = y_0 + \frac{(y_1 - y_0)}{(x_1 - x_0)}(x - x_0) + \\ + \left(\frac{(y_2 - y_1)}{(x_2 - x_1)} - \frac{(y_1 - y_0)}{(x_1 - x_0)}\right)\frac{(x - x_0)(x - x_1)}{(x_2 - x_0)}. \end{aligned} \quad (4.12)$$

Эквивалентность выражений интерполяционных полиномов второй степени

$$C_2(x) \equiv L_2(x) \equiv P_2(x) \tag{4.13}$$

можно показать по аналогии с (4.7).

Задачи по теме «Простые итерационные полиномы»

Задача 4.1. Точки на одной прямой. На координатной плоскости заданы две точки с координатами (x_0, y_0) и (x_1, y_1). Третья точка, для которой известна только координата x_2, находится на той же прямой (рис. 4.3). Требуется определить координату y_2 третьей точки.

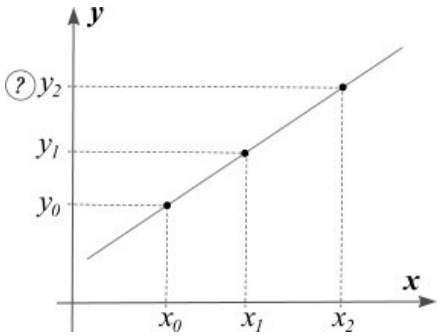


Рис. 4.3. Точки на одной прямой

Координату y_2 вычислить при помощи двух разных интерполяционных полиномов первой степени, указанных в варианте задания. Погрешность вычислений принять равной $\varepsilon = 0{,}01$.

Варианты данных к задаче 4.1

1. $(x_0 = 0{,}38;\ y_0 = 8{,}63)$; $(x_1 = 4{,}83;\ y_1 = 2{,}85)$; $x_2 = 9{,}64$; $y_2 = C_1(x_2)$ и $y_2 = L_1(x_2)$.

2. $(x_0 = 3{,}12;\ y_0 = 4{,}56)$; $(x_1 = 5{,}63;\ y_1 = 0{,}95)$; $x_2 = 7{,}41$; $y_2 = C_1(x_2)$ и $y_2 = P_1(x_2)$.

3. $(x_0 = 0{,}53;\ y_0 = 1{,}92)$; $(x_1 = 2{,}73;\ y_1 = 8{,}96)$; $x_2 = 5{,}63$; $y_2 = L_1(x_2)$ и $y_2 = P_1(x_2)$.

4. $(x_0 = 1{,}96;\ y_0 = 9{,}01)$; $(x_1 = 1{,}85;\ y_1 = 3{,}89)$; $x_2 = 5{,}34$; $y_2 = C_1(x_2)$ и $y_2 = L_1(x_2)$.

5. $(x_0 = 1{,}23;\ y_0 = 3{,}21)$; $(x_1 = 2{,}34;\ y_1 = 4{,}32)$; $x_2 = 6{,}43$; $y_2 = C_1(x_2)$ и $y_2 = P_1(x_2)$.

6. $(x_0 = 1{,}98;\ y_0 = 2{,}96)$; $(x_1 = 6{,}81;\ y_1 = 8{,}51)$; $x_2 = 8{,}64$; $y_2 = L_1(x_2)$ и $y_2 = P_1(x_2)$.

7. $(x_0 = 1{,}03;\ y_0 = 8{,}21)$; $(x_1 = 0{,}74;\ y_1 = 7{,}39)$; $x_2 = 4{,}36$; $y_2 = L_1(x_2)$ и $y_2 = P_1(x_2)$.

8. $(x_0 = 0{,}23;\ y_0 = 4{,}21)$; $(x_1 = 2{,}58;\ y_1 = 9{,}32)$; $x_2 = 6{,}34$; $y_2 = C_1(x_2)$ и $y_2 = L_1(x_2)$.

9. $(x_0 = 1{,}74;\ y_0 = 7{,}21)$; $(x_1 = 3{,}16;\ y_1 = 3{,}26)$; $x_2 = 9{,}34$; $y_2 = L_1(x_2)$ и $y_2 = P_1(x_2)$.

Задача 4.2. Точки на одной параболе. На координатной плоскости заданы три точки с координатами $(x_0,\ y_0)$, и $(x_1,\ y_1)$ и $(x_2,\ y_2)$. Четвертая точка, для которой известна только координата x_3, находится на той же параболе (рис. 4.4). Требуется определить координату y_3 четвертой точки.

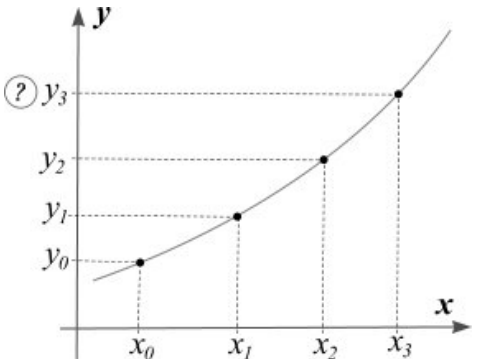


Рис. 4.4. Точки на одной параболе

Координату y_3 вычислить при помощи двух разных интерполяционных полиномов второй степени, указанных в варианте задания. Погрешность вычислений принять равной: $\varepsilon = 0{,}01$.

Варианты данных к задаче 4.2

1. $(x_0 = 3{,}26;\ y_0 = 4{,}56); (x_1 = 4{,}63;\ y_1 = 0{,}95); (x_2 = 3{,}12;\ y_2 = 5{,}81);\ x_3 = 7{,}41;\ y_3 = C_2(x_3)$ и $y_3 = P_2(x_3)$.

2. $(x_0 = 0{,}62;\ y_0 = 3{,}28); (x_1 = 4{,}67;\ y_1 = 2{,}13); (x_2 = 6{,}32;\ y_2 = 9{,}60);\ x_3 = 9{,}79;\ y_3 = C_2(x_3)$ и $y_3 = L_2(x_3)$.

3. $(x_0 = 0{,}87;\ y_0 = 7{,}56); (x_1 = 6{,}26;\ y_1 = 3{,}69); (x_2 = 7{,}87;\ y_2 = 4{,}47);\ x_3 = 8{,}41;\ y_3 = L_2(x_3)$ и $y_3 = P_2(x_3)$.

4. $(x_0 = 2{,}25;\ y_0 = 5{,}57); (x_1 = 2{,}90;\ y_1 = 0{,}65); (x_2 = 4{,}21;\ y_2 = 6{,}13);\ x_3 = 8{,}62;\ y_3 = C_2(x_3)$ и $y_3 = P_2(x_3)$.

5. $(x_0 = 0{,}13;\ y_0 = 2{,}33); (x_1 = 3{,}55;\ y_1 = 7{,}16); (x_2 = 4{,}98;\ y_2 = 1{,}41);\ x_3 = 5{,}93;\ y_3 = C_2(x_3)$ и $y_3 = L_2(x_3)$.

6. $(x_0 = 0{,}19;\ y_0 = 0{,}00); (x_1 = 3{,}34;\ y_1 = 7{,}64); (x_2 = 8{,}84;\ y_2 = 5{,}31);\ x_3 = 9{,}47;\ y_3 = L_2(x_3)$ и $y_3 = P_2(x_3)$.

7. $(x_0 = 2{,}73;\ y_0 = 9{,}78); (x_1 = 4{,}07;\ y_1 = 1{,}46); (x_2 = 6{,}21;\ y_2 = 7{,}57);\ x_3 = 7{,}02;\ y_3 = C_2(x_3)$ и $y_3 = P_2(x_3)$.

8. $(x_0 = 4{,}81;\ y_0 = 2{,}94); (x_1 = 5{,}91;\ y_1 = 7{,}28); (x_2 = 6{,}14;\ y_2 = 4{,}51);\ x_3 = 8{,}26;\ y_3 = C_2(x_3)$ и $y_3 = L_2(x_3)$.

9. $(x_0 = 0{,}34;\ y_0 = 2{,}94); (x_1 = 4{,}86;\ y_1 = 7{,}28); (x_2 = 5{,}39;\ y_2 = 4{,}51);\ x_3 = 6{,}78;\ y_3 = P_2(x_3)$ и $y_3 = L_2(x_3)$.

Задача 4.3. Принадлежность точек одной прямой. На координатной плоскости двумя точками $(x_0,\ y_0)$ и $(x_1,\ y_1)$ определена линейная функция. Требуется проверить, принадлежат ли точки $(x_2,\ y_2)$ и $(x_3,\ y_3)$ этой линейной функции (рис. 4.5).

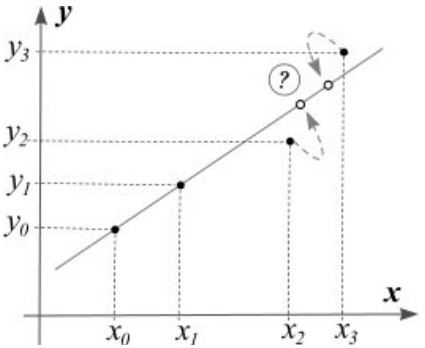


Рис. 4.5. Принадлежность точек одной прямой

Проверку выполнить двумя способами, при помощи двух разных интерполяционных полиномов первой степени, указанных в варианте задания. Погрешность вычислений: $\varepsilon = 0{,}01$.

Варианты данных к задаче 4.3

1. $(x_0 = 0{,}24;\ y_0 = 7{,}81); (x_1 = 2{,}89;\ y_1 = 8{,}13); (x_2 = 4{,}09;\ y_2 = 8{,}27);\ (x_3 = 6{,}69;\ y_3 = 9{,}05)$. Проверочные полиномы $C_1(x)$ и $L_1(x)$.

2. $(x_0 = 0{,}73;\ y_0 = 4{,}20); (x_1 = 3{,}09;\ y_1 = 6{,}91); (x_2 = 5{,}68;\ y_2 = 8{,}59);\ (x_3 = 7{,}58;\ y_3 = 12{,}07)$. Проверочные полиномы $C_1(x)$ и $L_1(x)$.

3. $(x_0 = 1{,}24;\ y_0 = 7{,}51); (x_1 = 2{,}09;\ y_1 = 6{,}34); (x_2 = 3{,}89;\ y_2 = 7{,}95);\ (x_3 = 5{,}68;\ y_3 = 1{,}40)$. Проверочные полиномы $C_1(x)$ и $L_1(x)$.

4. $(x_0 = 2{,}65;\ y_0 = 4{,}68); (x_1 = 3{,}29;\ y_1 = 3{,}18); (x_2 = 4{,}34;\ y_2 = 0{,}72);\ (x_3 = 5{,}72;\ y_3 = -2{,}15)$. Проверочные полиномы $C_1(x)$ и $L_1(x)$.

5. $(x_0 = 0{,}61;\ y_0 = 7{,}81); (x_1 = 1{,}83;\ y_1 = 8{,}13); (x_2 = 5{,}35;\ y_2 = 7{,}92);\ (x_3 = 6{,}43;\ y_3 = 5{,}12)$. Проверочные полиномы $C_1(x)$ и $L_1(x)$.

6. $(x_0 = 2{,}04;\ y_0 = 3{,}92); (x_1 = 3{,}18;\ y_1 = 6{,}15); (x_2 = 5{,}09;\ y_2 = 10{,}01);\ (x_3 = 8{,}89;\ y_3 = 17{,}32)$. Проверочные полиномы $C_1(x)$ и $L_1(x)$.

7. $(x_0 = 4{,}32;\ y_0 = 8{,}74); (x_1 = 5{,}49;\ y_1 = 2{,}13); (x_2 = 7{,}79;\ y_2 = -17{,}95);\ (x_3 = 9{,}78;\ y_3 = -22{,}11)$. Проверочные полиномы $C_1(x)$ и $L_1(x)$.

Задача 4.4. Принадлежность точек одной параболе. На координатной плоскости тремя точками $(x_0,\ y_0)$, $(x_1,\ y_1)$ и $(x_2,\ y_2)$ определена квадратичная функция. Требуется проверить, принадлежат ли точки $(x_3,\ y_3)$ и $(x_4,\ y_4)$ квадратичной функции (рис. 4.6).

Проверку выполнить двумя способами, при помощи двух разных интерполяционных полиномов второй степени, указанных в варианте задания. Погрешность вычислений: $\varepsilon = 0{,}01$.

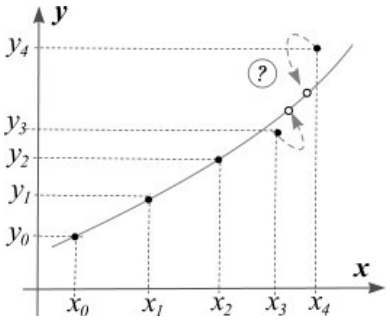


Рис. 4.6. Принадлежность точек одной параболе

Варианты данных к задаче 4.4

1. $(x_0 = 0{,}73;\ y_0 = 4{,}20);(x_1 = 3{,}09;\ y_1 = 6{,}91);(x_2 = 4{,}68;\ y_2 = 8{,}59);\quad (x_3 = 6{,}58;\ y_3 = 10{,}44);\quad (x_4 = 8{,}57;\quad y_4 = 12{,}07)$. Проверочные полиномы $C_2(x)$ и $L_2(x)$.

2. $(x_0 = 0{,}24;\ y_0 = 7{,}81);(x_1 = 2{,}89;\ y_1 = 8{,}13);(x_2 = 4{,}09;\ y_2 = 8{,}27);\ (x_3 = 6{,}69;\ y_3 = 9{,}05);\ (x_4 = 7{,}21;\ y_4 = 8{,}62)$. Проверочные полиномы $C_2(x)$ и $P_2(x)$.

3. $(x_0 = 2{,}65;\ y_0 = 4{,}68);(x_1 = 3{,}29;\ y_1 = 3{,}18);(x_2 = 4{,}34;\ y_2 = 0{,}72);\quad (x_3 = 5{,}72;\ y_3 = -2{,}15);\quad (x_4 = 6{,}34;\quad y_4 = -3{,}96)$. Проверочные полиномы $P_2(x)$ и $L_2(x)$.

4. $(x_0 = 0{,}61;\ y_0 = 7{,}81);(x_1 = 1{,}83;\ y_1 = 8{,}13);(x_2 = 5{,}35;\ y_2 = 7{,}92);\ (x_3 = 6{,}43;\ y_3 = 7{,}52);\ (x_4 = 7{,}55;\ y_4 = 9{,}07)$. Проверочные полиномы $C_2(x)$ и $L_2(x)$.

5. $(x_0 = 1{,}24;\ y_0 = 7{,}51);(x_1 = 2{,}09;\ y_1 = 6{,}34);(x_2 = 3{,}89;\ y_2 = 7{,}95);\ (x_3 = 5{,}68;\ y_3 = 1{,}40);\ (x_4 = 6{,}83;\ y_4 = 22{,}52)$. Проверочные полиномы $C_2(x)$ и $P_2(x)$.

6. $(x_0 = 4{,}32;\ y_0 = 8{,}74);(x_1 = 5{,}49;\ y_1 = 2{,}13);(x_2 = 7{,}79;\ y_2 = 9{,}75);\quad (x_3 = 9{,}78;\ y_3 = 38{,}39);\quad (x_4 = 11{,}33;\quad y_4 = 12{,}07)$. Проверочные полиномы $P_2(x)$ и $L_2(x)$.

7. $(x_0 = 2{,}04;\ y_0 = 3{,}92);(x_1 = 3{,}18;\ y_1 = 6{,}15);(x_2 = 5{,}09;\ y_2 = 10{,}01);\quad (x_3 = 6{,}89;\ y_3 = 17{,}32);\quad (x_4 = 9{,}83;\quad y_4 = 20{,}26)$. Проверочные полиномы $C_2(x)$ и $L_2(x)$.

Задача 4.5. Принадлежность точек параболе или прямой. На координатной плоскости тремя точками (x_0, y_0), (x_1, y_1) и (x_2, y_2) определена квадратичная функция. Точками (x_0, y_0) и (x_2, y_2) задана также линейная функция. Требуется выяснить, какая из точек (x_3, y_3) и (x_4, y_4) принадлежит параболе, а какая прямой (рис. 4.7). Принадлежность кривым проверять двумя полиномами соответствующих степеней. Погрешность вычислений: $\varepsilon = 0{,}01$.

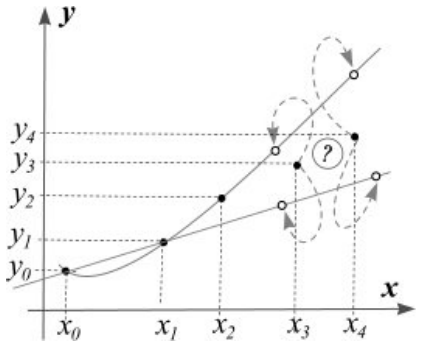


Рис. 4.7. Принадлежность точек одной параболе

Варианты данных к задаче 4.5

1. $(x_0 = 1{,}24;\ y_0 = 7{,}51); (x_1 = 2{,}09;\ y_1 = 6{,}34); (x_2 = 3{,}89;\ y_2 = 7{,}95);\ (x_3 = 5{,}68;\ y_3 = 15{,}06);\ (x_4 = 6{,}83;\ y_4 = -0{,}18)$. Проверочные полиномы $C_N(x)$ и $P_N(x)$.

2. $(x_0 = 0{,}61;\ y_0 = 7{,}81); (x_1 = 1{,}83;\ y_1 = 8{,}13); (x_2 = 5{,}35;\ y_2 = 7{,}92);\ (x_3 = 6{,}43;\ y_3 = 9{,}34);\ (x_4 = 7{,}55;\ y_4 = 6{,}93)$. Проверочные полиномы $C_N(x)$ и $L_N(x)$.

3. $(x_0 = 4{,}32;\ y_0 = 8{,}74); (x_1 = 5{,}49;\ y_1 = 2{,}13); (x_2 = 7{,}79;\ y_2 = 9{,}75);\ (x_3 = 9{,}78;\ y_3 = -22{,}11);\ (x_4 = 11{,}33;\ y_4 = 74{,}88)$. Проверочные полиномы $P_N(x)$ и $L_N(x)$.

4. $(x_0 = 2{,}04;\ y_0 = 3{,}92); (x_1 = 3{,}18;\ y_1 = 6{,}15); (x_2 = 5{,}09;\ y_2 = 10{,}01);\ (x_3 = 6{,}89;\ y_3 = 13{,}41);\ (x_4 = 9{,}63;\ y_4 = 19{,}81)$. Проверочные полиномы $C_N(x)$ и $L_N(x)$.

5. $(x_0 = 0{,}24;\ y_0 = 7{,}81);\ (x_1 = 2{,}89;\ y_1 = 8{,}13);\ (x_2 = 4{,}09;\ y_2 = 8{,}27);\ (x_3 = 6{,}69;\ y_3 = 8{,}56);\ (x_4 = 7{,}21;\ y_4 = 8{,}65)$. Проверочные полиномы $C_N(x)$ и $P_N(x)$.

6. $(x_0 = 0{,}73;\ y_0 = 4{,}20);\ (x_1 = 3{,}09;\ y_1 = 6{,}91);\ (x_2 = 4{,}68;\ y_2 = 8{,}59);\ (x_3 = 6{,}58;\ y_3 = 10{,}92);\ (x_4 = 8{,}57;\ y_4 = 12{,}21)$. Проверочные полиномы $C_N(x)$ и $L_N(x)$.

7. $(x_0 = 2{,}65;\ y_0 = 4{,}68);\ (x_1 = 3{,}29;\ y_1 = 3{,}18);\ (x_2 = 4{,}34;\ y_2 = 0{,}72);\ (x_3 = 5{,}72;\ y_3 = -2{,}51);\ (x_4 = 6{,}34;\ y_4 = -3{,}97)$. Проверочные полиномы $P_N(x)$ и $L_N(x)$.

4.2. ФИГУРЫ НА КООРДИНАТНОЙ ПЛОСКОСТИ

4.2.1. Треугольник на координатной плоскости

Простейшей фигурой на координатной плоскости является треугольник (рис. 4.8). Если координаты x_0, x_1, и x_2 точек A, B и C различны и расположены по возрастанию, то существует всего два вида треугольников: с точкой B выше или ниже прямой AC (как бы ни проводилась сторона AC и какие бы формы или размеры не принимал такой треугольник).

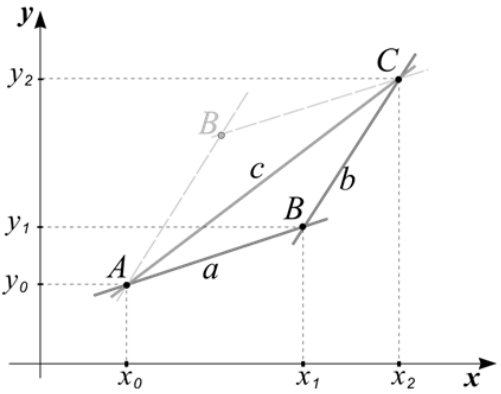


Рис. 4.8. Треугольник на координатной плоскости

Первый закономерный вопрос в отношении треугольника: если отсутствует картинка, но известны длины сторон a, b и c или координаты точек A, B и C – это вопрос о его существовании?

Если треугольник определен тремя сторонами a, b и c, то диагональ (максимальная из сторон) может оказаться слишком длинной и две другие стороны просто «не дотянутся» друг до друга, т.е. при трех заданных сторонах длина максимальной из них должна быть строго меньше суммы двух других. Если же длина диагонали равна сумме двух других сторон – это означает что все точки A, B и C лежат на одной прямой?

Точки A, B и C могут оказаться лежащими на одной прямой и в том случае, если треугольник задан координатами этих точек. Это легко проверить, составив уравнение прямой, проходящей через две любые точки, а затем проверить принадлежность оставшейся точки этой прямой.

Убедившись в существовании треугольника можно классифицировать его как тупоугольный, прямоугольный – при наличии в нем таких тупого или прямого угла, а также как остроугольный, если все углы треугольника острые. Классификацию можно построить на теореме косинусов:

$$a^2 = b^2 + c^2 - 2bc\cos\alpha, \tag{4.14}$$

где α – угол противолежащий стороне a (аналогичное соотношение можно записать для двух других сторон).

Однако для классификации треугольника можно обойтись следствием теоремы косинусов – теоремой Пифагора. Если сумма квадратов катетов равна квадрату гипотенузы – треугольник прямоугольный, если меньше – треугольник тупоугольный, если больше, тогда остроугольный.

Также можно классифицировать треугольник как равнобедренный при наличии двух равных сторон, а если между собой равны все три стороны – как равносторонний. В противном случае треугольник является разносторонним.

Если треугольник определен точками A, B и C на координатной плоскости (рис. 4.8), то длины его сторон легко находятся по теореме Пифагора:

$$\begin{aligned} a^2 &= (x_1 - x_0)^2 + (y_1 - y_0)^2; \\ b^2 &= (x_2 - x_1)^2 + (y_2 - y_1)^2; \\ c^2 &= (x_2 - x_0)^2 + (y_2 - y_0)^2. \end{aligned} \tag{4.15}$$

Рассмотрим задачу нахождения площади треугольника, заданного тремя сторонами или тремя точками. Площадь треугольника при трех известных сторонах a, b и c может быть вычислена по формуле Герона:

$$S_{\Delta abc} = \frac{1}{4}\sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}. \quad (4.16)$$

Площадь треугольника, заданного точками A, B и C на координатной плоскости (рис. 4.8), может быть также вычислена по формуле Герона (4.16) (после вычисления сторон a, b и c).

В дополнение к вышесказанному рассмотрим геометрический метод вычисления площади треугольника. На рис. 4.8 можно видеть три трапеции: x_0ABx_1, x_1BCx_2 и x_0ACx_2, соответственно площадь треугольника будет вычисляться как

$$S_{\Delta ABC} = \left|S_{x_0ACx_2} - S_{x_0ABx_1} - S_{x_1BCx_2}\right|. \quad (4.17)$$

Здесь взятие модуля разности гарантирует выполнение этого соотношения для случаев, когда точка B расположена как выше, так и ниже стороны AC.

Подстановка формул площади трапеции (половина суммы оснований, умноженная на высоту) в (4.17) приводит к формуле

$$S_{\Delta ABC} = \frac{1}{2}|(y_2+y_0)(x_2-x_0) - (y_1+y_0)(x_1-x_0) - (y_2+y_1)(x_2-x_1)|. \quad (4.18)$$

4.2.2. Криволинейная трапеция на координатной плоскости

Рассмотрим простейшие методы вычисления площади криволинейной трапеции x_0ABx_2 (рис. 4.9).

Криволинейная трапеция представляет собой область под графиком параболы $y(x) = ax^2 + bx + c$, ограниченную вертикальными прямыми, проходящими через точки x_0 и x_2. Значения коэффициентов a, b и c считаются известными.

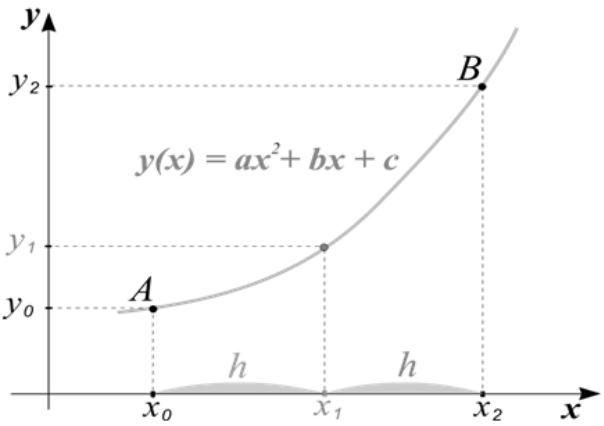


Рис. 4.9. Криволинейная трапеция

Поскольку вычисление первообразной $Y(x)$ от квадратичной функции $y(x)$ не представляет никаких трудностей, то площадь трапеции легко вычислить по формуле Ньютона–Лейбница:

$$S_{x_0ABx_2} = \int_{x_0}^{x_2} y(x)dx = Y(x_2) - Y(x_0) = \\ = \frac{1}{3}ax^3 + \frac{1}{2}bx^2 + cx\Big|_{x_0}^{x_2}. \quad (4.19)$$

Для контроля правильности полученного результата при программировании задачи вычислим $S_{x_0ABx_2}$ с использованием интерполяционного полинома Лагранжа второй степени (4.11). Для этого потребуется еще одна точка: (x_1, y_1). Выберем точку x_1 в центре отрезка $[x_0, x_2]$, соответственно: $y_1 = y(x_1)$ и имеет место:

$$x_1 = x_0 + h,\ x_2 = x_0 + 2h, \quad (4.20)$$

где $h = \frac{x_2 - x_0}{2}$.

Удобно ввести новую безразмерную переменную

$$q = \frac{(x - x_0)}{h}, \quad (4.21)$$

принимающую целые значения 0, 1 и 2 в точках x_0, x_1 и x_2. Тогда с учетом (4.20) и (4.21) выражение для интерполяционного полинома Лагранжа второй степени (4.11) запишется как

$$L_2(q) = \frac{1}{2}(q-1)(q-2)y_0 - q(q-2)y_1 + \frac{1}{2}q(q-1)y_2. \qquad (4.22)$$

Интегрируя полином Лагранжа в соответствующих пределах, получим выражение

$$S_{x_0ABx_2} = \int_{q(x_0)}^{q(x_2)} L_2(q)\frac{dx}{dq}dq = h\int_0^2 L_2(q)dq = \\ = \frac{1}{3}h(y_0 + 4y_1 + y_2), \qquad (4.23)$$

называемое формулой Симпсона.

Задачи по теме «Фигуры на координатной плоскости»

Задача 4.6. Существование и классификация треугольника, определённого через длины сторон. Задано несколько троек чисел *a*, *b* и *c* – возможные стороны треугольника (см. варианты заданий). Необходимо выполнить следующие действия.

1. Запрограммировать последовательный ввод с клавиатуры значений *a*, *b* и *c*.
2. Переставить значения длин сторон *a*, *b* и *c* таким образом, чтобы сторона *c* оказалась гипотенузой.
3. Проверить возможность существования треугольника с гипотенузой в соответствии с теоремой косинусов (используя формулу (4.14)) или теоремой Пифагора.
4. Если треугольник существует, то на основании теоремы косинусов или теоремы Пифагора классифицировать его как:
   1) тупоугольный;
   2) прямоугольный;
   3) остроугольный.

5. Классифицировать существующий треугольник как:
1) разносторонний;
2) равносторонний;
3) равнобедренный.

Варианты данных для задачи 4.6

1. $a = 3{,}22;\ b = 2{,}02;\ c = 33{,}19;$ $a = 19{,}91;\ b = 26{,}55;\ c = 2{,}09;\ a = 6{,}24;\ b = 6{,}24;\ c = 6{,}24.$

2. $a = 13{,}45;\ b = 17{,}93;\ c = 32{,}42;$ $a = 4{,}48;\ b = 4{,}48;\ c = 6{,}34;\ a = 6{,}62;\ b = 6{,}88;\ c = 13{,}50.$

3. $a = 6{,}93;\ b = 9{,}23;\ c = 11{,}54;$ $a = 6{,}62;\ b = 6{,}88;\ c = 15{,}12;\ a = 7{,}51;\ b = 7{,}51;\ c = 5{,}49.$

4. $a = 10{,}08;\ b = 13{,}44;\ c = 16{,}80;$ $a = 7{,}93;\ b = 7{,}93;\ c = 11{,}22;\ a = 2{,}76;\ b = 2{,}20;\ c = 5{,}02.$

5. $a = 7{,}01;\ b = 4{,}88;\ c = 11{,}89;$ $a = 7{,}51;\ b = 7{,}51;\ c = 7{,}51;\ a = 3{,}07;\ b = 4{,}09;\ c = 5{,}12.$

6. $a = 1{,}24;\ b = 7{,}51;\ c = 2{,}09;$ $a = 3{,}75;\ b = 3{,}75;\ c = 5{,}30;\ a = 11{,}24;\ b = 14{,}99;\ c = 19{,}73.$

7. $a = 4{,}28;\ b = 8{,}96;\ c = 13{,}27;$ $a = 3{,}94;\ b = 5{,}25;\ c = 6{,}57;\ a = 1{,}31;\ b = 1{,}31;\ c = 1{,}94.$

Задача 4.7. Существование и классификация треугольника, определённого через координаты вершин на плоскости. На координатной плоскости задано несколько наборов по три точки *A*, *B* и *C* с координатами $(x_0\ ,\ y_0)$, $(x_1\ ,\ y_1)$ и $(x_2\ ,\ y_2)$ соответственно (см. рис. 4.8).

1. Необходимо проверить существование треугольника (не лежат ли точки на одной прямой).

2. Если треугольник существует, то классифицировать его как тупоугольный, прямоугольный или остроугольный, а также как разносторонний, равнобедренный или равносторонний.

3. Погрешность вычислений: $\varepsilon = 0{,}01$.

Варианты данных для задачи 4.7

1. $(x_0 = 1{,}29;\ y_0 = 2{,}37); (x_1 = 2{,}26;\ y_1 = 4{,}08); (x_2 = 3{,}57;\ y_2 = 2{,}20);$ $(x_0 = 1{,}40;\ y_0 = -8{,}95); (x_1 = 3{,}42;\ y_1 = -26{,}88); (x_2 = 5{,}59;\ y_2 = -46{,}08);$

$(x_0 = 1{,}16;\ y_0 = 4{,}16); (x_1 = 1{,}31;\ y_1 = 3{,}83); (x_2 = 9{,}65;\ y_2 = 8{,}21)$.

2. $(x_0 = 2{,}33;\ y_0 = 4{,}72); (x_1 = 6{,}83;\ y_1 = 6{,}87); (x_2 = 15{,}18;\ y_2 = 10{,}85);$
$(x_0 = -3{,}31;\ y_0 = -0{,}06); (x_1 = 0{,}75;\ y_1 = 7{,}10); (x_2 = 2{,}82;\ y_2 = 5{,}92);$
$(x_0 = -16{,}69;\ y_0 = -0{,}82); (x_1 = -2{,}03;\ y_1 = 8{,}58); (x_2 = 13{,}77;\ y_2 = 1{,}26)$.

3. $(x_0 = -7{,}42;\ y_0 = 2{,}94); (x_1 = -2{,}26;\ y_1 = 2{,}66); (x_2 = 1{,}80;\ y_2 = 5{,}86);$
$(x_0 = -14{,}15;\ y_0 = 20{,}48); (x_1 = -8{,}43;\ y_1 = 2{,}14); (x_2 = -2{,}85;\ y_2 = 6{,}73);$
$(x_0 = 0{,}37;\ y_0 = 9{,}37); (x_1 = 5{,}39;\ y_1 = 13{,}50); (x_2 = 14{,}87;\ y_2 = 21{,}30)$.

4. $(x_0 = -0{,}21;\ y_0 = 0{,}99); (x_1 = 0{,}04;\ y_1 = 9{,}64); (x_2 = 1{,}95;\ y_2 = 9{,}37);$
$(x_0 = 3{,}86;\ y_0 = 15{,}52); (x_1 = 13{,}30;\ y_1 = 33{,}39); (x_2 = 17{,}24;\ y_2 = 40{,}85);$
$(x_0 = -2{,}01;\ y_0 = 4{,}40); (x_1 = 3{,}12;\ y_1 = 1{,}70); (x_2 = 11{,}98;\ y_2 = 30{,}90)$.

5. $(x_0 = 1{,}25;\ y_0 = 2{,}34); (x_1 = 7{,}74;\ y_1 = -6{,}15); (x_2 = 8{,}14;\ y_2 = -6{,}66);$
$(x_0 = -4{,}51;\ y_0 = 10{,}95); (x_1 = -0{,}55;\ y_1 = 4{,}15); (x_2 = 7{,}79;\ y_2 = 9{,}75);$
$(x_0 = 4{,}32;\ y_0 = 8{,}74); (x_1 = 5{,}49;\ y_1 = 2{,}13); (x_2 = 1{,}15;\ y_2 = 6{,}16)$.

6. $(x_0 = -1{,}34;\ y_0 = 2{,}53); (x_1 = 4{,}42;\ y_1 = 1{,}05); (x_2 = 13{,}20;\ y_2 = 59{,}05);$
$(x_0 = -6{,}45;\ y_0 = 2{,}70); (x_1 = -1{,}31;\ y_1 = 1{,}21); (x_2 = 3{,}33;\ y_2 = 3{,}88);$
$(x_0 = 3{,}27;\ y_0 = 8{,}62); (x_1 = 11{,}17;\ y_1 = 6{,}86); (x_2 = 17{,}27;\ y_2 = 5{,}50)$.

7. $(x_0 = 7{,}18;\ y_0 = 35{,}15); (x_1 = 15{,}17;\ y_1 = 70{,}95); (x_2 = 29{,}08;\ y_2 =$
$= 110{,}91); (x_0 = -3{,}31;\ y_0 = 6{,}04); (x_1 = 5{,}79;\ y_1 = -10{,}23); (x_2 = 8{,}43;\ y_2 =$
$= 12{,}60); (x_0 = -0{,}93;\ y_0 = 4{,}10); (x_1 = 17{,}26;\ y_1 = 4{,}24); (x_2 = 34{,}99;\ y_2 =$
$= 0{,}13)$.

8. $(x_0 = -10{,}67;\ y_0 = -0{,}03); (x_1 = -6{,}96;\ y_1 = 2{,}74); (x_2 = -2{,}38;\ y_2 =$
$= 2{,}06); (x_0 = 7{,}31;\ y_0 = -0{,}06); (x_1 = 17{,}15;\ y_1 = -0{,}27); (x_2 = 18{,}40;\ y_2 =$
$= -0{,}29); (x_0 = 0{,}33;\ y_0 = 8{,}27); (x_1 = 8{,}11;\ y_1 = -16{,}96); (x_2 = 17{,}62;\ y_2 =$
$= 13{,}60)$.

Задача 4.8. Вычисление площади треугольника, определённого через координаты вершин на плоскости. Дополнить задачу 3.7 (используя те же варианты данных) вычислением площади треугольников по формуле Герона, а также через суперпозицию площадей трапеций. Сравнить результаты. Погрешность вычислений: $\varepsilon = 0{,}01$.

Задача 3.9. Вычисление площади криволинейной трапеции. Вычислить площадь криволинейной трапеции x_0ABx_2 (см. рис. 4.9) по формуле Ньютона–Лейбница (4.19) и проверить результат по формуле Симпсона (4.23). Трапеция ограничена параболой $y(x) = ax^2 + bx + c$ и вертикальными прямыми x_0 и x_2. Погрешность вычислений не превышает $\varepsilon = 0{,}01$.

Варианты данных к задаче 3.9

1. $a = 7{,}54;\ b = 6{,}34;\ c = 4{,}52; x_0 = 1{,}19;\ x_2 = 3{,}02$.

2. $a = 1{,}71;\ b = 8{,}64;\ c = 6{,}79;\ x_0 = 0{,}70;\ x_2 = 9{,}11.$
3. $a = 4{,}97;\ b = 6{,}90;\ c = 8{,}32;\ x_0 = 1{,}16;\ x_2 = 3{,}77.$
4. $a = 8{,}29;\ b = 5{,}07;\ c = 9{,}91;\ x_0 = 1{,}03;\ x_2 = 2{,}38.$
5. $a = 2{,}25;\ b = 7{,}74;\ c = 9{,}68;\ x_0 = 3{,}96;\ x_2 = 10{,}09.$
6. $a = 0{,}49;\ b = 6{,}54;\ c = 7{,}48;\ x_0 = 2{,}05;\ x_2 = 7{,}18.$
7. $a = 7{,}32;\ b = 6{,}82;\ c = 9{,}41;\ x_0 = 1{,}46;\ x_2 = 9{,}59.$
8. $a = 6{,}21;\ b = 3{,}66;\ c = 9{,}35;\ x_0 = 8{,}37;\ x_2 = 14{,}84.$
9. $a = 9{,}15;\ b = 3{,}85;\ c = 7{,}03;\ x_0 = 4{,}16;\ x_2 = 13{,}21.$
10. $a = 3{,}95;\ b = 1{,}31;\ c = 8{,}97;\ x_0 = 0{,}04;\ x_2 = 9{,}49.$

4.3. РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

4.3.1. Подход к решению нелинейных уравнений

Распространенной задачей в самых разных областях вычислений является численное решение уравнения $f(x) = 0$ для нелинейной функции $f(x)$ на отрезке $[a, b]$. При этом, как правило, идёт речь о поиске заведомо единственного корня ξ функции $f(x)$ непрерывной на отрезке $[a, b]$ (рис. 4.10, 4.11).

Эта задача решается путем нахождения приближенного значение корня уравнения ξ_0, отстоящего от него не более чем на величину малой положительной погрешности ε (расположенного справа или слева от истинного значения ξ корня уравнения).

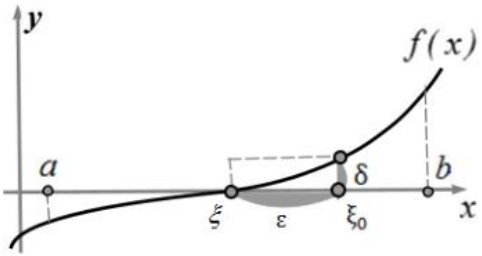


Рис. 4.10. Приближенное решение нелинейного уравнения. Пологая функция

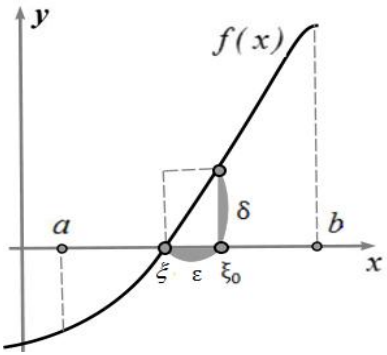


Рис. 4.11. Приближенное решение нелинейного уравнения. Крутая функция

При этом значения функции в точках ξ и ξ_0 должны отличаться не более чем на величину невязки δ – малой положительной величины. Величины погрешности и невязки являются необходимыми начальными данными для решения задачи нахождения корня нелинейного уравнения:

$$f(x)|_{x=\xi} \equiv 0, \text{ где } \xi \in [a, b];$$
$$|\xi - \xi_0| \leq \varepsilon, \quad |f(\xi_0)| \leq \delta. \tag{4.24}$$

Значимость величин отклонения – погрешности ε и невязки δ, иллюстрируется на рисунках выше. Если в окрестности корня ξ функция достаточно пологая (см. рис. 4.10), то величина погрешности ε достаточна для регулирования точности решения. Если же функция круто возрастает или убывает (см. рис. 4.11), то для точности решения становится существенным значение невязки δ.

4.3.2. Деление отрезка пополам (дихотомия)

Основная идея решения уравнения (4.24) методом дихотомии заключается в том, что корень уравнения – ξ, являющийся внутренней точкой отрезка $[a, b]$, будет также внутренней точкой левой половины отрезка $[a, x_{\text{ср}}]$ или правой половины $[x_{\text{ср}}, b]$. При этом середина отрезка определяется как $x_{\text{ср}} = a + (b - a)/2$ (см. рис. 4.6).

Последовательность итераций метода деления отрезка пополам проиллюстрирована на рис. 4.6. Номера точек a, b и $x_{\text{ср}}$, соответствующих очередной итерации, проставлены в скобках, в левом верхнем углу, например: ${}^{(1)}a$, ${}^{(1)}b$ и ${}^{(1)}x_{\text{ср}}$ или ${}^{(2)}a$, ${}^{(2)}b$ и ${}^{(2)}x_{\text{ср}}$, исходные границы $[a,\ b]$ соответствуют нулевой итерации.

Отрезок $[a, b]$ делится пополам (вычисляется значение $x_{\text{ср}}$). Из двух отрезков: $[a, x_{\text{ср}}]$ и $[x_{\text{ср}}, b]$ выбирается тот, который содержит корень и рассматривается как новый отрезок $[a, b]$ на следующей итерации. Процесс деления отрезка пополам завершается (задача нахождения корня решена) при выполнении условий:

$$|b - a| \leq \varepsilon, \quad \frac{|f(b)-f(a)|}{2} \leq \delta. \tag{4.25}$$

В качестве решения, для определенности можно принять $\xi_0 = x_{\text{ср}}$. Значения погрешности ε и невязки δ определяются из соображений необходимой точности решения задачи и входят в блок исходных данных наряду с выражением для функции $f(x)$ и границами поиска корня a и b.

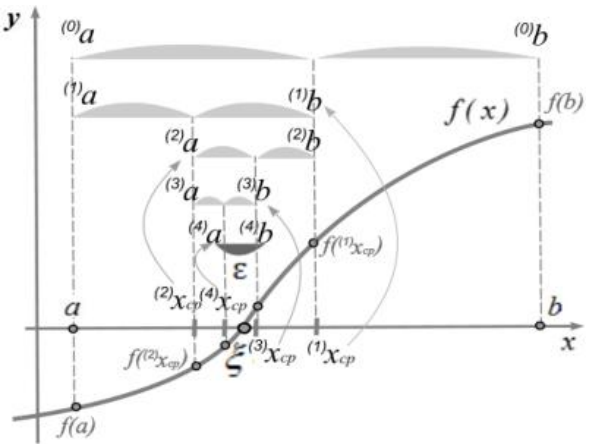


Рис. 4.12. Решение уравнения делением отрезка пополам

Выбрать из двух половин отрезка $[a, b]$ (на каждой итерации) ту половину, которая содержит корень уравнения, можно, сравнивая

знаки значений функции в точках a, $x_{\text{ср}}$ и b. В момент пересечения оси x в точке ξ, функция меняет знак. Знак функции меняется с отрицательного на положительный, если функция возрастает и с положительного на отрицательный, если функция убывает.

Соответственно, на концах половины отрезка, которая содержит корень уравнения, функция будет иметь разные знаки, а на концах половины отрезка, не содержащей корня, – одинаковые знаки.

Произведение чисел с разными знаками отрицательно, тогда как произведение чисел с одинаковыми знаками положительно.

На первой (1) итерации (рис. 4.12) функция меняет знак на отрезке $[a, x_{\text{ср}}]$, поскольку: $f(a)f(x_{\text{ср}}) < 0$ – и значит знаки функции на концах отрезка разные. Соответственно, отрезок $[x_{\text{ср}}, b]$ выбывает из дальнейшего рассмотрения, и точка $x_{\text{ср}}$ становится новой точкой b для следующей итерации.

На второй итерации (2) функция меняет знак уже на отрезке $[x_{\text{ср}}, b]$ (выполняется критерий: $f(b)f(x_{\text{ср}}) < 0$), и точка $x_{\text{ср}}$ становится новой точкой a. Все последующие итерации выполняются по той же схеме.

Условием применимости метода деления отрезка пополам является непрерывность функции $f(x)$ на заданном отрезке $[a, b]$ и наличие единственного корня функции на этом отрезке.

4.3.3. Метод хорд

Если кривая функции $f(x)$ на отрезке $[a, b]$ не содержит точек перегиба, т.е. функция на отрезке выпукла вниз (вторая производная $f''(x) > 0$ всюду на отрезке) или выпукла вверх (вторая производная $f''(x) < 0$ всюду на отрезке), то для решения уравнения (4.24) можно воспользоваться методом хорд (рис. 4.13).

Сначала строится хорда (прямолинейный отрезок) соединяющий точки графика функции $f(x)$ на границах отрезка $[a, b]$ (точки A и B), и ищется точка пересечения хорды с осью x. Таким образом, определяется приближенное значение корня уравнения – точка x_0 и соответствующая ей на графике точка A_1.

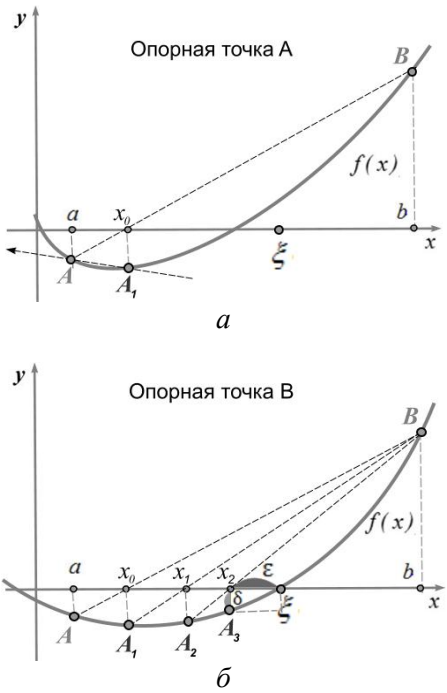


Рис. 4.13. Метод хорд: *а* – опорная точка *A*; *б* – опорная точка *B*

Далее строится хорда, соединяющая точку A_1 с опорной точкой (A или B) и позволяющая найти следующие приближение корня уравнения – точку x_1, и т.д. В результате последовательного построения хорд (см. рис. 4.13, *б*) формируется последовательность их точек пересечения с осью x: $\{x_0,\ x_1,\ x_2 \ldots\ x_N\}$, сходящаяся к корню ξ уравнения (4.24).

Критерием выбора опорной точки является совпадение знака функции в опорной точке со знаком второй производной функции $f''(x)$ (постоянство знака второй производной на отрезке $[a, b]$ является условием применимости метода хорд).

В примере, представленном на рис. 4.13, указанному критерию соответствует точка B (см. рис. 4.13, б).

Если же в качестве опорной точки выбрать точку A (см. рис. 4.13, *а*), то последовательность точек пересечения с осью x выйдет за пределы отрезка $[a, b]$.

Уравнение прямой, проходящей через две точки A и B, можно записать как

$$\frac{x-a}{b-a} = \frac{f(x)-f(a)}{f(b)-f(a)}. \tag{4.26}$$

Соответственно, координаты пересечения точки пересечения хорды $[AB]$ и точек пересечения последующих хорд осью x будет определяться выражениями:

$$\begin{aligned} x_0 &= a - \frac{f(a)}{f(b)-f(a)}(b-a); \\ x_1 &= a - \frac{f(a)}{f(b)-f(x_0)}(b-x_0); \\ x_2 &= a - \frac{f(a)}{f(b)-f(x_1)}(b-x_1); \\ &\ldots; \\ x_N &= a - \frac{f(a)}{f(b)-f(x_{N-1})}(b-x_{N-1}). \end{aligned} \tag{4.27}$$

Процесс решения завершается на итерации с номером N при достижении заданной погрешности и невязки:

$$|x_N - x_{N-1}| \leq \varepsilon, \ |f(x_N)| \leq \delta. \tag{4.28}$$

4.3.4. Метод касательных (метод Ньютона)

Метод касательных используется для решения уравнения (4.24), если кривая функции $f(x)$ на конечном (или бесконечном) отрезке монотонно возрастает или убывает без точек перегиба, т.е. на отрезке рассмотрения функции $a < x < b$ сохраняют знак и непрерывны $f'(x)$ и $f''(x)$ (первая и вторая производная функции).

Критерий выбора опорной точки – совпадение знака функции в опорной точке со знаком второй производной функции $f''(x)$ (постоянство знака второй производной на отрезке $[a, b]$ является условием применимости метода хорд).

По своей сути этот метод похож на метод хорд и отличается только способом построения линейных функций, с помощью кото-

рых определяется очередное приближённое значение корня уравнения $f(x)$ – вместо хорд используются касательные (рис. 4.14).

Рис. 4.14. Метод касательных. Опорная точка *B*

В точке B строится касательная к графику функции:

$$y - f(b) = f'(b)(x - b), \tag{4.29}$$

которая при пересечении с осью x дает начальное приближение корня уравнения:

$$x_0 = b - \frac{f(b)}{f'(b)}. \tag{4.30}$$

Аналогичным образом получаются все последующие приближения:

$$x_N = x_{N-1} - \frac{f(x_{N-1})}{f'(x_{N-1})}. \tag{4.31}$$

Метод касательных является условно сходящимся методом, для его сходимости в области поиска корня должно быть выполнено условие:

$$|ff''| < (f')^2, \tag{4.32}$$

в противном случае сходимость будет лишь в некоторой окрестности корня ξ. Процесс решения завершается на итерации с номером

N при достижении заданной погрешности и невязки, как и для метода хорд.

4.3.5. Метод секущих

Модификацией метода касательных является метод секущих, если значения производной $f'(x)$ на всём отрезке $[a, b]$ незначительно отличаются друг от друга по величине.

В этом случае достаточно вычислить значение производной в опорной точке для нахождения точки x_0 по формуле (4.28) и использовать это значение для построения последовательности секущих линий и определения последовательности приближенных значений корня уравнения $f(x) = 0$ (рис. 4.15).

Рис. 4.15. Метод секущих. Опорная точка *B*

В отличие от метода касательных для вычисления производной функции $f(x)$ в методе секущих используются конечно-разностные приближения для производной функции $f'(x)$:

$$\begin{aligned} f'(x_0) &= \frac{f(x_0) - f(b)}{x_0 - b}; \\ f'(x_1) &= \frac{f(x_1) - f(x_0)}{x_1 - x_0}; \\ &\ldots; \\ f'(x_N) &= \frac{f(x_N) - f(x_{N-1})}{x_N - x_{N-1}}. \end{aligned} \tag{4.33}$$

Из соотношения (4.28) и с учетом (4.33) и получается формула метода секущих:

$$x_{N+1} = x_N - \frac{x_N - x_{N-1}}{f(x_N) - f(x_{N-1})} f(x_N). \qquad (4.34)$$

Обычно в методе секущих требуется больше итераций, чем в методе касательных, но зато каждая итерация выполняется значительно быстрее, так как не требуется вычислять производные, и поэтому часто при таком же объеме вычислений можно сделать больше итераций и получить более высокую точность.

Задачи по теме «Решение нелинейных уравнений»

Вычислить корень уравнения (4.24) двумя различными методами, указанными в задачах и сравнить полученные результаты.

Задача 4.10. Метод дихотомии и метод хорд.
Задача 4.11. Метод дихотомии и метод Ньютона.
Задача 4.12. Метод дихотомии и метод секущих.
Задача 4.13. Метод хорд и метод Ньютона.
Задача 4.14. Метод хорд и метод секущих.
Задача 4.15. Метод Ньютона и методом секущих.

Варианты данных к задачам 4.10 – 4.15

Для всех вариантов $[ab] = [0,1]$.
Коэффициенты: $i, j, k, l, m = 1 \div 4$.
Максимальное число итераций $N_{\max} = 20 \div 100$.
Правильность решения проверяется невязкой: $|f(\xi_0)| < \delta$, где ξ_0 – вычисленный корень уравнения (4.24), погрешность решения ε и невязка δ выбираются из диапазона $10^{-7} \div 10^{-5}$.

1. $f(x) = -\cos^k(\pi x^m/2) + 2x^i$.
2. $f(x) = -\sin^k(\pi x^m/2) - (1-x)^i$.
3. $f(x) = \operatorname{sh} x^j - \cos^k(\pi x^m)$.
4. $f(x) = 1 - x^j - \operatorname{tg}^k(\pi x^m/4)$.
5. $f(x) = (1-x)^j - \operatorname{tg}^k(\pi x^m/4)$.
6. $f(x) = (1-x)^{1/j} - \operatorname{tg}^k(\pi x^m/4)$.

7. $f(x) = e^{-x^j} - \sin(\pi x^m/2)$.
8. $f(x) = e^{-x^j} - x^k \sin(\pi x^m/2)$.
9. $f(x) = (1-x)^m - 2x^j$.
10. $f(x) = (1-x)^{1/m} - 2x^{1/j}$.
11. $f(x) = (1-x)^j \mathrm{ch}^k x^m - x^j$.
12. $f(x) = (1-x)^j \cos^k(\pi x^m/2) - x^j$.
13. $f(x) = (1-x)^j - \mathrm{sh}^k x^m$.
14. $f(x) = (1-x)^{1/j} - \mathrm{sh}^k x^m$.
15. $f(x) = (1-x)^j - \mathrm{sh}^{1/k} x^m$.
16. $f(x) = (1-x)^j \mathrm{ch}^k x^m - x^l$.
17. $f(x) = \arcsin x^m - (1-x^j)^k$.
18. $f(x) = k\cos(\pi \mathrm{x}^\mathrm{m}) + x^j(1-x)$.
19. $f(x) = \ln^m\left(1+x^j\right) - (1-x)^k$.
20. $f(x) = \ln^m\left(1+x^j\right) - (1-x)^k \mathrm{ch}\, x^l$.

4.4. ИНТЕРПОЛЯЦИЯ ФУНКЦИЙ ПОЛИНОМАМИ

4.4.1. Интерполяция функции полиномами степени *N*

Ранее было показано, что по двум заданным точкам координатной плоскости можно построить интерполяционный полином первой степени $C_1(x)$, а по трем точкам – интерполяционный полином первой степени $C_2(x)$. В общем случае при наличии $N+1$ точек можно сконструировать интерполяционный полином степени N в виде:

$$C_N(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_i x^i + \cdots a_N x^N \qquad (4.35)$$

или в компактном виде

$$C_N(x) = \sum_{i=0}^{N} a_i x^i. \qquad (4.36)$$

График такого полинома представлен на рис. 4.16.

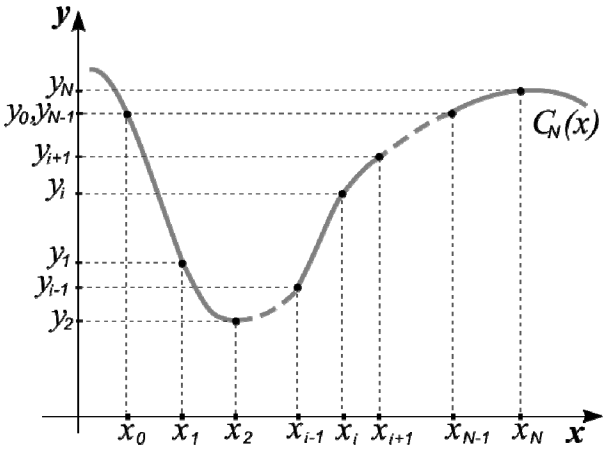


Рис. 4.16. Интерполяционный полином степени N

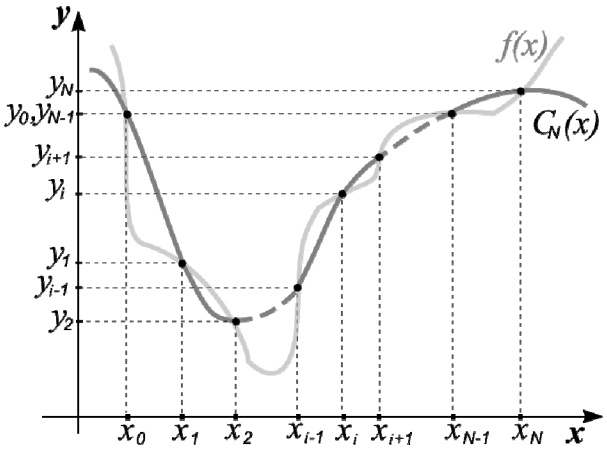


Рис. 4.17. Интерполяция функции полиномом

Форма представления интерполяционного полинома (4.35) или (4.36) называется *канонической*. Чтобы записать полином в канонической форме, нужно вычислить коэффициенты a_i, где i – номер точки, изменяющийся от нуля до N. Для этого решается система

$N+1$ алгебраических уравнений, которая получается путем подстановки в (4.35) значений координат точек (x_i, y_i):

$$\begin{cases} y_0 = a_0x_0^0 + a_1x_0^1 + a_2x_0^2 + \cdots + a_ix_0^i + \cdots + a_Nx_0^N; \\ y_1 = a_0x_1^0 + a_1x_1^1 + a_2x_1^2 + \cdots + a_ix_1^i + \cdots + a_Nx_1^N; \\ y_2 = a_0x_2^0 + a_1x_2^1 + a_2x_2^2 + \cdots + a_ix_2^i + \cdots + a_Nx_2^N; \\ \quad\quad\quad\quad\quad\quad\quad\quad \ldots; \\ y_i = a_0x_i^0 + a_1x_i^1 + a_2x_i^2 + \cdots + a_ix_i^i + \cdots + a_Nx_i^N; \\ \quad\quad\quad\quad\quad\quad\quad\quad \ldots; \\ y_N = a_0x_N^0 + a_1x_N^1 + a_2x_N^2 + \cdots + a_ix_N^i + \cdots + a_Nx_N^N. \end{cases} \quad (4.37)$$

Рассмотренные полиномы называются интерполяционными, поскольку служат для интерполяции функций. Если функция *f*(*x*) не имеет разрывов и скачков на множестве $(x_0, x_1, \ldots, y_N)$, ее можно заменить другой функцией, близкой к ней на множестве точек x_i. Такую замену называют *аппроксимацией*. Если требуется, чтобы аппроксимирующая функция точно совпадала с *f*(*x*) во всех x_i, то такую апроксимацию называют *интерполяцией*, а точки x_i называются *узлами интерполяции* (рис. 4.17).

Значения координат (x_i, y_i) вместе образуют таблицу чисел, поэтому функция *f*(*x*) называется *таблично заданной* функцией. Последовательность всех x_i от x_0 до x_N называется *сеткой узлов*, соответственно, каждый x_i – это *узел сетки*. В связи с этим таблично заданные функции называют так же *сеточными*. Узлы интерполяции обычно совпадают с узлами сетки.

Сетка узлов называется *равномерной*, если построена таким образом, что:

$$x_1 - x_0 = x_2 - x_1 = \cdots = x_i - x_{i-1} = \cdots = x_N - x_{N-1} = \boldsymbol{h}. \quad (4.38)$$

Чем ниже степень полинома, интерполирующего функцию, тем проще вычислительные модели, построенные на его основе. Можно интерполировать таблично заданную функцию *f*(*x*) не одним полиномом степени N на всем отрезке $[x_0, x_N]$, а несколькими полиномами более низких степеней на коротких участках этого отрезка. Такая интерполяция называется *кусочной*.

4.4.2. Кусочная интерполяция полиномами малых степеней

Для лучшего понимания этой идеи рассмотрим простейшие способы построения графиков функций по точкам. Самый простой способ построить график функции на отрезке $[a, b]$ (где $x_0 = a$ и $x_N = b$) по известным точкам – это взять линейку и соединить точки, нанесенные на координатную плоскость отрезками прямых линий, как на рис. 4.18.

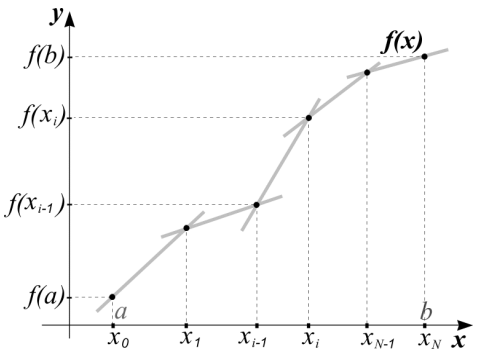


Рис. 4.18. Построение графика функции из отрезков прямых

Можно так же взять лекало и соединить точки плавными кривыми. Плавность кривых, соединяющих точки графика, зависит от вида функций, описывающих эти кривые. Такими кривыми могут быть параболы типа x^2, как это показано на рис. 4.19.

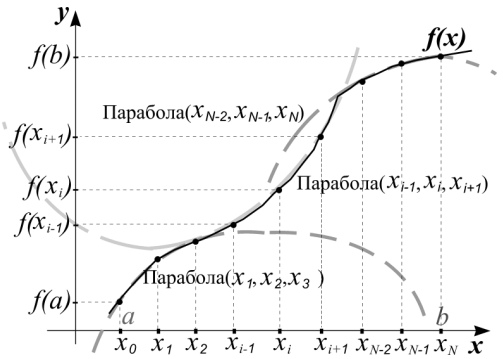


Рис. 4.19. Построение графика функции из фрагментов парабол типа x^2

Соединение точек отрезками прямых означает, что на каждом отрезке $[x_{i-1}\ ,\ x_i]$ (где $i = 1, N$) оси x функция $f(x)$ интерполируется полиномом первой степени $C_{1i}(x)$:

$$C_{1i}(x) = a_{0i} + a_{1i}x, \tag{4.39}$$

где

$$a_{1i} = \frac{(y_i - y_{i-1})}{(x_i - x_{i-1})}; \qquad a_{0i} = \frac{(y_{i-1}x_i - y_i x_{i-1})}{(x_i - x_{i-1})}. \tag{4.40}$$

Здесь первый индекс 1 – это степень полинома, а второй индекс i – это номер полинома.

Эквивалентный полином Лагранжа представляется в виде:

$$L_{1i}(x) = y_{i-1}\frac{(x - x_i)}{(x_{i-1} - x_i)} + y_i\frac{(x - x_{i-1})}{(x_i - x_{i-1})}, \tag{4.41}$$

а полином Ньютона можно записать в виде:

$$P_{1i}(x) = y_{i-1} + \frac{(y_i - y_{i-1})}{(x_i - x_{i-1})}(x - x_{i-1}). \tag{4.42}$$

При построении кусочно-квадратичного приближения узлы разбиваются не на пары, а на тройки $(x_{i-1},\ x_i,\ x_{i+1})$. Так же как в случае линейной интерполяции: $i = 1, N$. Но при этом N должно быть четным целым положительным числом. При нечетном значении N ситуация значительно осложняется. Если множество узлов удалось разбить на тройки, то значения функции в узлах позволят получить уравнение полиномов второй степени $P_{2i}(x)$ на каждом отрезке $[x_{i-1}, x_{i+1}]$. Интерполирующий полином на отрезке квадратичной интерполяции имеет три эквивалентные формы записи.

Это полином в канонической форме:

$$C_{2i}(x) = a_{0i} + a_{1i}x + a_{2i}x^2, \tag{4.43}$$

где

$$\begin{aligned} a_{0i} &= \frac{1}{x_i(x_{i+1} - x_{i-1})}\left(x_{i-1}\frac{y_{i+1}x_i^2 - y_i x_{i+1}^2}{x_{i+1} - x_i} - x_{i+1}\frac{y_i x_{i-1}^2 - y_{i-1}x_i^2}{x_i - x_{i-1}}\right); \\ a_{1i} &= -\frac{1}{x_{i+1} - x_{i-1}}\left(\frac{(y_{i+1} - y_i)(x_i + x_{i-1})}{(x_{i+1} - x_i)} - \frac{(y_i - y_{i-1})(x_{i+1} + x_i)}{(x_i - x_{i-1})}\right); \\ a_{2i} &= \frac{1}{x_{i+1} - x_{i-1}}\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}}\right); \end{aligned} \tag{4.44}$$

полином Лагранжа:

$$L_{2i}(x) = y_{i-1}\frac{(x - x_i)(x - x_{i+1})}{(x_{i-1} - x_i)(x_{i-1} - x_{i+1})} + \\ + y_i\frac{(x - x_{i-1})(x - x_{i+1})}{(x_i - x_{i-1})(x_i - x_{i+1})} + \\ + y_{i+1}\frac{(x - x_{i-1})(x - x_i)}{(x_{i+1} - x_{i-1})(x_{i+1} - x_i)} \tag{4.45}$$

и полином Ньютона:

$$P_{2i}(x) = y_{i-1} + \frac{(y_i - y_{i-1})}{(x_i - x_{i-1})}(x - x_{i-1}) + \\ + \left(\frac{(y_{i+1} - y_i)}{(x_{i+1} - x_i)} - \frac{(y_i - y_{i-1})}{(x_i - x_{i-1})}\right)\frac{(x - x_{i-1})(x - x_i)}{(x_{i+1} - x_{i-1})}. \tag{4.46}$$

4.4.3. Кусочная интерполяция полиномом степени *N*

Для кусочного приближения функции *f*(*x*) интерполяционными полиномами более высоких степеней можно использовать представление полиномов в общем виде.

Полином Лагранжа:

$$L_N(x) = \sum_{i=0}^{N} y_i l_i, \tag{4.47}$$

где l_i – так называемый *базисный полином*:

$$l_i = \prod_{\substack{j=0 \\ j\neq i}}^{N}\frac{(x - x_j)}{(x_i - x_j)} = \frac{(x - x_0)(x - x_1)(x - x_2)\ldots}{(x_i - x_0)(x_i - x_1)(x_i - x_2)\ldots}\frac{(x - x_{N-1})}{(x_i - x_{N-1})}\frac{(x - x_N)}{(x_i - x_N)} \tag{4.48}$$

и полином Ньютона:

$$P_N(x) = y_0 + \sum_{i=1}^{N} p_i(x), \tag{4.49}$$

где

$$p_i(x) = f(x_0; x_1; \ldots; x_i)(x - x_0)(x - x_1)\ldots(x - x_{i-1}), \tag{4.50}$$

которая называется *разделенной разностью* i-го порядка $f(x_0; x_1; \dots; x_i)$ и вычисляется по схеме:

$$
\begin{array}{lll}
f(x_0) = y_0, & & \\
 & f(x_0;x_1) = \frac{y_1-y_0}{x_1-x_0}, & \\
f(x_1) = y_1, & f(x_0;x_1;x_2) = \frac{f(x_1;x_2)-f(x_0;x_1)}{x_2-x_0}, & \\
 & f(x_1;x_2) = \frac{y_2-y_1}{x_2-x_1}, & f(x_0;x_1;x_2;x_3) = \frac{f(x_1;x_2;x_3)-f(x_0;x_1;x_2)}{x_3-x_0}, \\
f(x_2) = y_2, & f(x_1;x_2;x_3) = \frac{f(x_2;x_3)-f(x_1;x_2)}{x_3-x_1}, & f(x_0;x_1;x_2;x_3;x_4) = \cdots, \\
 & f(x_2;x_3) = \frac{y_3-y_2}{x_3-x_2}, & f(x_1;x_2;x_3;x_4) = \frac{f(x_2;x_3;x_4)-f(x_1;x_2;x_3)}{x_4-x_1}, \\
f(x_3) = y_3, & f(x_2;x_3;x_4) = \frac{f(x_3;x_4)-f(x_2;x_3)}{x_4-x_2}, & \\
 & f(x_3;x_4) = \frac{y_4-y_3}{x_4-x_3}, & \\
f(x_4) = y_4. & &
\end{array}
\qquad (4.51)
$$

С учетом (4.51) интерполяционный полином Ньютона второй степени можно записать в виде:

$$
\begin{array}{c}
P_{2i}(x) = y_{i-1} + f(x_{i-1}; x_i)(x - x_{i-1}) + \\
+f(x_{i-1}; x_i; x_{i+1})(x - x_{i-1})(x - x_i).
\end{array}
\qquad (4.52)
$$

Кусочная интерполяция функции на равномерной сетке узлов (4.38) полиномом степени N дает погрешность этой степени относительно величины h:

$$f(x) = P_N(x) + O(h^N). \qquad (4.53)$$

В заданиях данного лабораторного практикума рассматриваются функции аналитического вида, т.е. явные формулы для которых известны. По такой формуле можно посчитать «истинное» значение функции. Поэтому погрешность приближения функции полиномом можно оценить величиной невязки, которая представляет собой разность между «истинным» значением функции и значением, полученным в результате приближенных вычислений, или, еще точнее – это модуль такой разности:

$$\xi = |f(x) - P_N(x)|. \qquad (4.54)$$

Задачи по теме «Интерполяция функций полиномами»

Задача 4.16. Исследование свойств таблично заданной функции.

1. Вычислить таблицу значений функции $f(x)$ в равноотстоящих узлах на отрезке $[ab]$:

$$y_j = f(x_j),\ x_j = a + jh;\ j = 0, N;\ h = (b-a)/N.$$

2. Найти максимальное и минимальное значения функции:

$$y_{\min} = \min_j(y_j),\ y_{\max} = \max_j(y_j),$$

и соответствующие им номера узлов j.

3. Вычислить среднее значение $\bar{y}$, средний квадрат $\overline{y^2}$ и среднеквадратичное значение функции y_m:

$$\bar{y} = \frac{1}{N+1}\sum_{j=0}^{N} y_j\ ;\ \overline{y^2} = \frac{1}{N+1}\sum_{j=0}^{N} y_j^2\ ;\ y_m = \sqrt{\overline{y^2}}.$$

4. Найти относительное число положительных p_+ и отрицательных p_- значений функции:

$$p_\pm = N_\pm/(N+1),$$

где N_+ и N_- – число положительных и отрицательных значений таблицы y_j соответственно.

5. Вычислить среднеквадратичное отклонение от среднего значения:

$$\sigma = \sqrt{\frac{1}{N}\sum_{j=0}^{N}(y_j - \bar{y})^2}.$$

Варианты функций для задач 4.16 – 4.19

Для всех вариантов $[ab] = [0,1]$.

Коэффициенты: $r, s = 1 \div 5;\quad k, m = 1 \div 4;\quad N = 20 \div 100.$

1. $f(x) = \sin^k(\pi x^m)$.
2. $f(x) = \cos^k(\pi x^m)$.
3. $f(x) = \sin^k(\pi x^{1/m})$.
4. $f(x) = \cos^k(\pi x^{1/m})$.
5. $f(x) = \mathrm{tg}^k(\pi x^m/4)$.
6. $f(x) = \mathrm{tg}^k(\pi x^{1/m}/4)$.
7. $f(x) = \pm rx^4 \pm sx^3$.
8. $f(x) = (r + sx^m)^{-k}$.
9. $f(x) = (r + sx^m)^{-1/k}$.
10. $f(x) = (r + sx^{1/m})^{-k}$.
11. $f(x) = (r + sx^{1/m})^{-1/k}$.
12. $f(x) = x^k/(r + sx^m)$.
13. $f(x) = x^k/(r + sx)^m$.
14. $f(x) = x^{1/k}/(r + sx)^m$.
15. $f(x) = x^k/(r + sx)^{1/m}$.
16. $f(x) = x^{1/k}/(r + sx)^{1/m}$.
17. $f(x) = (r + x^2)^k(r + x^4)^m$.
18. $f(x) = (r + x^2)^{1/k}(r + x^4)^m$.
19. $f(x) = (r + x^2)^k(r + x^4)^m$.
20. $f(x) = (r + x^2)^{1/k}(r + x^4)^{1/m}$.
21. $f(x) = x^k/(r + sx^m)$.
22. $f(x) = x^{1/k}/(r + sx^m)$.

Задача 4.17. Вычисление биноминальных коэффициентов. Вычислить таблицу биноминальных коэффициентов C_N^k для заданного (вводимого с клавиатуры) натурального числа $N \leq 7$ двумя способами:

1) по формуле:

$$C_N^k = \frac{N!}{(N-k)!k!};$$

2) при помощи *треугольника Паскаля*:

N \ *k*	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0								1							
1							1		1						
2						1		2		1					
3					1		3		3		1				
4				1		4		6		4		1			
5			1		5		10		10		5		1		
6		1		6		15		20		15		6		1	
7	1		7		21		35		35		21		7		1

Левое и правое значение в строке треугольника Паскаля всегда равно единице. Значение любого другого элемента вычисляется как сумма ближайших левого и правого элементов предыдущей строки.

Задача 4.18. Вычисление таблицы разделенных разностей. Для табличной функции на отрезке $[ab]$:

$$y_j = f(x_j), \qquad x_j = a + jh; \;\; j = 0, N;$$

где

$$h = (b - a)/N \qquad (N \le 7).$$

1. Составить таблицу разделенных разностей $f(x_0; \dots; x_N)$ по вычислительной схеме (4.51).

2. Для проверки использовать вычисление разделенных разностей по формуле:

$$f(x_0; \dots; x_N) = \frac{1}{N! h^N} \sum_{k=0}^{N} (-1)^k C_N^k y_k.$$

Вычисление биноминальных коэффициентов рассмотрено в задаче 4.17.

Задача 4.19. Исследование интерполяционных полиномов. Для табличной функции на отрезке $[ab]$:

$$y_j = f(x_j), \qquad x_j = a + jh, \qquad h = \frac{b - a}{N}; \;\; j = 0, N.$$

1. Построить интерполяционный полином Ньютона второй степени $P_{2i}(x)$ в соответствии с (4.46).

2. Проверить правильность вычислений по формуле (4.45) для табличной функции $f(x)$ в узлах интерполяции x_j и в промежуточных точках между узлами сетки: $x_j + h/2, j = 0, N$.

3. Вычислить погрешности интерполирования:

$$\varepsilon(x_j + h/2) = \left| f(x_j + h/2) - P_{2j}(x_j + h/2) \right| ;$$

$$\varepsilon_{\max} = \max_{\mathrm{j}} \left(\varepsilon(x_j + h/2) \right).$$

4. Вычислить средний квадрат погрешности $\overline{\varepsilon^2}$ и среднеквадратичную погрешность ε_{m}:

$$\overline{\varepsilon^2} = \frac{1}{N+1} \sum_{j=0}^{N} \left[\varepsilon(x_j + h/2) \right]^2 ; \; \varepsilon_{\mathrm{m}} = \sqrt{\overline{\varepsilon^2}}.$$

5. Исследовать, как меняются $\varepsilon_{\max}$ и ε_{m} с изменением N.

4.5. ЧИСЛЕННОЕ ДИФФЕРЕНЦИРОВАНИЕ

4.5.1. Формулы дифференцирования, вытекающие из кусочной интерполяции функций

Численное дифференцирование – задача вычисления производных функции $f(x)$ в заданных точках отрезка $[ab]$. Например, в равноотстоящих узлах сетки с шагом h: $(x_0 = a;\ x_1 = x_0 + h;\ x_2 = = x_0 + 2h;\ \ldots;\ x_j = x_{i-1} + jh;\ \ldots;\ x_N = b)$. Величина h либо известна, либо определяется соотношением:

$$h = \frac{(b - a)}{N}. \tag{4.55}$$

Если дифференцируемая функция задана таблично в узлах сетки или имеет сложный аналитический вид (формулу), то ее можно аппроксимировать функцией с простым аналитическим представлением, например интерполяционным полиномом Ньютона $P_N(x)$, который будет интерполировать функцию с погрешностью $R_N(x)$:

$$f(x) = P_N(x) + R_N(x). \tag{4.56}$$

Вполне достаточную для практических приложений точность дифференцирования можно получить путем кусочной интерполяции полиномами малых степеней ($k = 1, 2$ или 3) и их последующего дифференцирования.

Для интерполяции функции на некотором отрезке $[x_j, x_{j+k}]$ удобно ввести безразмерную переменную q, принимающую целые значения в узлах интерполяции

$$q = \frac{(x - x_j)}{h}, \tag{4.57}$$

а вместо разделенных разностей $f(x_0; x_1; \ldots; x_i)$ использовать конечные разности $\Delta^i y_0$ $(i = 1, k)$, вычисляемые по схеме:

$$\begin{array}{ll}
y_{j-1}, & \\
 & \Delta y_{j-1} = y_j - y_{j-1}, \\
y_j, & \Delta^2 y_{j-1} = \Delta y_j - \Delta y_{j-1} = y_{j+1} - 2y_j + y_{j-1}, \\
 & \Delta y_j = y_{j+1} - y_j \quad \Delta^3 y_{j-1} = \Delta^2 y_{j+1} - \Delta^2 y_j = \boldsymbol{y_{j+2} - 3y_{j+1} +} \\
 & \qquad\qquad \boldsymbol{+3y_j - y_{j-1}}, \\
y_{j+1}, & \Delta^2 y_j = \Delta y_{j+1} - \Delta y_j = y_{j+2} - 2y_{j+1} + y_j, \\
 & \Delta y_{j+1} = y_{j+2} - y_{j+1}, \\
y_{j+2}. &
\end{array} \tag{4.58}$$

Тогда полином Ньютона (4.49) можно записать в виде:

$$P_k(x) = y_j + q\Delta y_j + \frac{q(q-1)}{2}\Delta^2 y_j + \frac{q(q-1)(q-2)}{6}\Delta^3 y_j + \cdots + \\ + \frac{q(q-1)(q-2)\ldots(q-N+1)}{N!}\Delta^k y_j. \tag{4.59}$$

При этом кусочная интерполяция ($k = 3$) имеет вид:

$$f(x)_{[x_{j-1}, x_{j+2}]} = y_{j-1} + q\Delta y_{j-1} + \frac{q(q-1)}{2}\Delta^2 y_{j-1} + \\ + \frac{q(q-1)(q-2)}{6}\Delta^3 y_{j-1} + R_3(x), \tag{4.60}$$

где $R_3(x)$ – погрешность интерполяции.

После чего, продифференцировав полином как сложную функцию, получаем выражения для первой и второй производной:

$$P_k^{'}(x) = P_k^{'}(q)q^{'}(x) = \frac{1}{h}P_k^{'}(q) \tag{4.61}$$

и

$$P_k^{''}(x) = \left(P_k^{'}(q)\right)^{'} q^{'}(x) = \frac{1}{h^2}P_k^{''}(q), \tag{4.62}$$

тогда при кусочной интерполяции полиномами третьей степени:

$$f^{'}(x)_{[x_{j-1},x_{j+2}]} = \frac{1}{h}\left[\Delta y_{j-1} + \frac{1}{2}(2q-1)\,\Delta^2 y_{j-1} + \right.$$
$$\left. + \frac{1}{6}(3q^2 - 6q + 2)\Delta^3 y_{j-1}\right] + R_3^{'}(x) \tag{4.63}$$

и

$$f^{''}(x)_{[x_j,x_{j+3}]} = \frac{1}{h^2}\left[\Delta^2 y_{j-1} + \ (q-1)\Delta^3 y_{j-1}\right] + R_3^{''}(x). \tag{4.64}$$

4.5.2. Конечно-разностные формулы для производных

На основе (4.63) и (4.64) получаются выражения для вычисления первой и второй производной со вторым порядком точности относительно величины h:

$$f'(x)_{[x_{j-1},x_{j+1}]} = \frac{1}{h}\left[\Delta y_{j-1} + \frac{2q-1}{2}\Delta^2 y_{j-1}\right] + O(h^2); \tag{4.65}$$

$$f''(x)_{[x_{j-1},x_{j+2}]} = \frac{1}{h^2}\left[\Delta^2 y_{j-1} + (q-1)\Delta^3 y_{j-1}\right] + O(h^2). \tag{4.66}$$

Подставляя значения: $q = 0$ $(x = x_{j-1})$, $q = 1$ $(x = x_j)$, $q = 2$ $(x = x_{j+1})$ и $q = 3$ $(x = x_{j+2})$, получаем окончательные выражения для вычисления первых и вторых производных:

$$\begin{aligned} f'(x_{j-1}) &= \frac{1}{h}\left[\Delta y_{j-1} - \frac{1}{2}\Delta^2 y_{j-1}\right] + O(h^2); \\ f^{'}(x_j) &= \frac{1}{h}\left[\Delta y_{j-1} + \Delta^2 y_{j-1}\right] + O(h^2); \\ f'(x_{j+1}) &= \frac{1}{h}\left[\Delta y_{j-1} + \frac{3}{2}\Delta^2 y_{j-1}\right] + O(h^2); \end{aligned} \tag{4.67}$$

$$
\begin{aligned}
f''(x_{j-1}) &= \frac{1}{h^2}\left[\Delta^2 y_{j-1} - \Delta^3 y_{j-1}\right] + O(h^2);\\
f''(x_j) &= \frac{1}{h^2}\Delta^2 y_{j-1} + O(h^2);\\
f''(x_{j+1}) &= \frac{1}{h^2}\left[\Delta^2 y_{j-1} + \Delta^3 y_{j-1}\right] + O(h^2);\\
f''(x_{j+2}) &= \frac{1}{h^2}\left[\Delta^2 y_{j-1} + 2\Delta^3 y_{j-1}\right] + O(h^2).
\end{aligned}
\tag{4.68}
$$

Таким образом, в узлах $(x_j,\ j = 1, N-1)$ первая и вторая производная функции $f(x)$ будут определяться выражениями:

$$
\begin{aligned}
f'(x_j) &= \frac{1}{2h}\left(y_{j+1} - y_{j-1}\right) + O(h^2);\\
f''(x_j) &= \frac{1}{h^2}(y_{j+1} - 2y_j + y_{j-1}) + O(h^2).
\end{aligned}
\tag{4.69}
$$

На левой границе первая и вторая производная будут иметь вид:

$$
\begin{aligned}
f'(x_0) &= \frac{1}{2h}(-3y_0 + 4y_1 - y_2) + O(h^2);\\
f''(x_0) &= \frac{1}{h^2}(2y_0 - 5y_1 + 4y_2 - y_3) + O(h^2).
\end{aligned}
\tag{4.70}
$$

Значения первой и второй производной на правой границе сетки узлов будут вычисляться по формулам:

$$
\begin{aligned}
f'(x_N) &= \frac{1}{2h}(y_{N-2} - 4y_{N-1} + 3y_N) + O(h^2);\\
f''(x_N) &= \frac{1}{h^2}(-y_{N-3} + 4y_{N-2} - 5y_{N-1} + 2y_N) + O(h^2).
\end{aligned}
\tag{4.71}
$$

Задачи по теме «Численное дифференцирование»

Задача 4.20. Вычислить таблицу точных значений первых производных функции $f'(x)$ на основе аналитических формул дифференцирования и приближенных значений производных по формулам (4.69) – (4.71).

Сравнить результаты.

1. Оценить погрешности (модули разности между точным и приближенным значением).

2. Выяснить влияние на погрешность величины шага h и количества точек N.

Варианты функций для задачи 4.20 и 4.21

1. $f(x) = \sin(\pi x^2)$.
2. $f(x) = x^2 e^x$.
3. $f(x) = x\ \mathrm{sh}\ x$.
4. $f(x) = \cos\ (\pi x^2)$.
5. $f(x) = x^2 e^{-x}$.
6. $f(x) = x\ \mathrm{ch}\ x$.
7. $f(x) = \mathrm{sh}\ x^2$.
8. $f(x) = e^{x^2/2}$.
9. $f(x) = x^2\ \mathrm{sh}\ x$.
10. $f(x) = \mathrm{ch}\ x^2$.
11. $f(x) = e^{-x^2/2}$.
12. $f(x) = x^2\ \mathrm{ch}\ x$.
13. $f(x) = x \sin x$.
14. $f(x) = \sin^2 x$.
15. $f(x) = x\ \mathrm{sh}\ x^2$.
16. $f(x) = x \cos x$.
17. $f(x) = \cos^2 x$.
18. $f(x) = \mathrm{tg}(\pi x^2/4)$.
19. $f(x) = x^2 \sin x$.
20. $f(x) = x \sin^2 x$.
21. $f(x) = (x+1)\ln(x+1)$.
22. $f(x) = x^2 \cos x$.
23. $f(x) = x \cos^2 x$.
24. $f(x) = x \ln(x^2+1)$.
25. $f(x) = e^x \sin x$.
26. $f(x) = x e^{x^2/2}$.
27. $f(x) = \arcsin\ (x^2/2)$.
28. $f(x) = \mathrm{arctg}\, x^2$.
29. $f(x) = x e^{-x^2/2}$.
30. $f(x) = (x+1)^2 \ln(x+1)$.

Задача 4.21. Вычислить таблицу точных значений вторых производных функции $f''(x)$ на основе аналитических формул диффе-

ренцирования и приближенных значений производных по формулам (4.69) – (4.71).

Сравнить результаты.

1. Оценить погрешности (модули разности между точным и приближенным значением).

2. Выяснить влияние на погрешность величины шага *h* и количества точек *N*.

4.6. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

4.6.1. Формулы интегрирования, вытекающие из кусочной интерполяции функций

Численное интегрирование – задача вычисления определенного интеграла функции *f*(*x*), непрерывной на отрезке [*ab*]. Если известна первообразная *F*(*x*), то определенный интеграл вычисляется по формуле Ньютона–Лейбница:

$$\int_a^b f(x)dx = F(b) - F(a). \quad (4.72)$$

При трудностях нахождения первообразной функция *f*(*x*) может быть задана таблицей $y_i = f(x_i)$, где $i = 0, \dots, N$, с соблюдением граничных условий $x_0 = a$ и $x_N = b$. После этого можно выполнить кусочно-линейную интерполяцию функции *f*(*x*), заменив ее на отрезках $[x_{i-1}, x_i]$ интерполяционным полиномом Лагранжа первой степени:

$$\int_a^b f(x)dx = \sum_{i=1}^{N} \left[\int_{x_{i-1}}^{x_i} f(x)dx \right] = \sum_{i=1}^{N} \left[\int_{x_{i-1}}^{x_i} L_{1i}(x)dx \right] + R_1(x). \quad (4.73)$$

Для увеличения точности интегрирования используется кусочно-квадратичная интерполяция функции на отрезке $[x_{i-1}, x_{i+1}]$ полиномом Лагранжа второй степени:

$$\int_a^b f(x)dx = \sum_{i=1}^{N-1} \left[\int_{x_{i-1}}^{x_{i+1}} L_{2i}(x)dx \right] + R_2(x), \quad (4.74)$$

$R_1(x)$ и $R_2(x)$ – остаточные члены интегрирования соответствующего прядка точности. В обоих случаях определенный интеграл функции на отрезке $[ab]$ будет равен сумме интегралов на составляющих его отрезках.

Для удобства интегрирования вводится переменная:

$$q = \frac{(x - x_{i-1})}{(x_i - x_{i-1})}. \tag{4.75}$$

В результате полиномы Лагранжа первой и второй степени запишутся в виде:

$$L_{1i}(x) = (q-1)y_{i-1} + qy_i \tag{4.76}$$

и

$$L_{2i}(q) = \frac{1}{2}(q-1)(q-2)y_{i-1} + q(q-2)y_i + \\ + \frac{1}{2}q(q-1)y_{i+1}. \tag{4.77}$$

Интегрирование полинома Лагранжа первой степени на отрезке $[x_{i-1},\ x_i]$ сводится к его интегрированию на отрезке $[0, 1]$ по переменной q. Новые пределы интегрирования получаются подстановкой старых пределов в выражение (4.75):

$$\int_{x_{i-1}}^{x_i} L_{1i}(x)dx = \int_0^1 L_{1i}(x)\frac{dx}{dq}dq = \\ = (x_i - x_{i-1})\int_0^1 L_{1i}(q)dq = \frac{1}{2}(x_i - x_{i-1})(y_{i-1} + y_i). \tag{4.78}$$

Определенный интеграл на отрезке $[ab]$ вычисляется как сумма интегралов (4.78) на всех отрезках $[x_{i-1}, x_i]$ (4.79):

$$\int_a^b f(x)dx = \frac{1}{2}\sum_{i=1}^{N}\big((x_i - x_{i-1})(y_{i-1} + y_i)\big) + R_1(x). \tag{4.79}$$

Интерполяция функции полиномами Лагранжа второй степени позволяет получить квадратурную формулу Симпсона:

$$\int_{x_{i-1}}^{x_{i+1}} L_{2i}(x)dx = \int_0^2 L_{1i}(x)\frac{dx}{dq}dq = h\int_0^2 L_{2i}(q)dq = \\ = \frac{1}{3}h(y_{i-1} + 4y_i + y_{i+1}). \tag{4.80}$$

На отрезке $[ab]$ локальные интегралы суммируются, при этом количество узлов N должно быть четным:

$$\int_a^b f(x)dx = \frac{1}{3}h\sum_{i=1}^{N-1}(y_{i-1} + 4y_i + y_{i+1}) + R_2(x). \tag{4.81}$$

Численное интегрирование по формулам (4.79) и (4.81) имеет наглядную геометрическую интерпретацию: определенный интеграл – это площадь под кривой функции $f(x)$ на отрезке $[ab]$.

Кусочно-линейная интерполяция позволяет рассматривать интеграл как сумму площадей локальных трапеций (криволинейных, в случае кусочно-квадратичной интерполяции).

4.6.2. Метод прямоугольников

Существуют еще более простые геометрические интерпретации и соответствующие им методы вычисления определенных интегралов. Это так называемые методы прямоугольников (рис. 4.20 и 4.21). Интегралы вычисляются по формулам (4.82) или (4.83) для ступенчатых функций, которыми интерполируется функция $f(x)$ на отрезке $[ab]$.

В результате интеграл представляет собой сумму площадей прямоугольников:

$$\int_a^b f(x)dx = \sum_{i=0}^{N-1} y_i(x_{i+1} - x_i) + R_0(x) \tag{4.82}$$

или

$$\int_a^b f(x)dx = \sum_{i=1}^{N} y_i(x_i - x_{i-1}) + R_0(x). \tag{4.83}$$

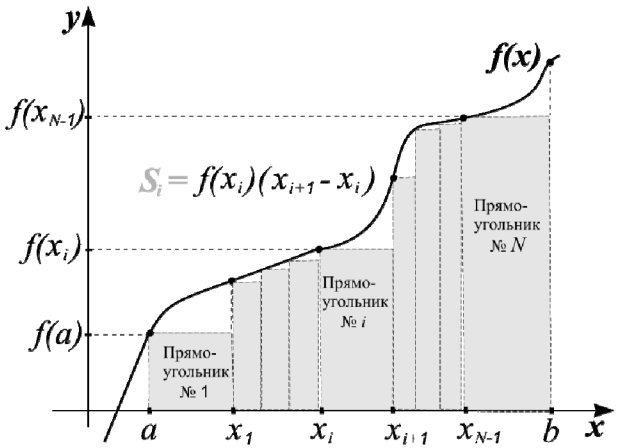


Рис. 4.20. Метод прямоугольников (по нижнему пределу)

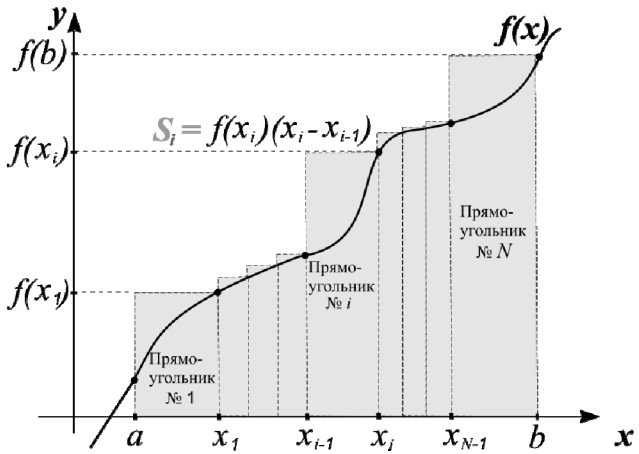


Рис. 4.21. Метод прямоугольников (по верхнему пределу)

В первом случае вычисляемая площадь заведомо меньше фактического интеграла, а во втором – заведомо больше, поэтому эти вычислительные схемы называют методами прямоугольников по нижнему и верхнему пределу.

Для уточнения метода прямоугольников на отрезках $[x_{i-1}, x_i]$ или $[x_i, x_{i+1}]$ берут высоту как среднее арифметическое от высот в граничных точках локальных отрезков. В этом случае получается вычислительная формула, совпадающая с методом трапеций (4.79).

Задачи по теме «Численное интегрирование»

На равномерной сетке узлов с шагом $h = (b - a)/N$ вычислить интеграл от функции $f(x)$ на отрезке $[ab]$. Оценить влияние величины шага h (количества точек N) на расхождение результатов.

Для вычисления интеграла используются два метода, указанных в задаче.

Задача 4.22. Формула прямоугольников и формула трапеций.

Задача 4.23. Формула прямоугольников и формула Симпсона.

Задача 4.24. Формула трапеций и формула Симпсона.

Варианты данных для задач 4.22, 4.23 и 4.24

Границы $[ab]$= [0,1].

Параметры: $r, s = 0{,}1 \div 5; \quad i, m = 1 \div 4; \quad N = 20 \div 500.$

1. $f(x) = \pm r x^i \pm s.$

2. $f(x) = \pm x^i/(r x^m + s).$

3. $f(x) = \pm x^i/(r(1-x)^m + s).$

4. $f(x) = \pm r(1-x)^m e^{\pm s x^{i/4}}.$

5. $f(x) = \pm r(1-x)^m \pm s;$

6. $f(x) = \pm x^i/(r(1-x^m) \pm s).$

7. $f(x) = \pm r e^{\pm s x^i/4}.$

8. $f(x) = \pm r(1-x^m) e^{\pm s x^i/4}.$

9. $f(x) = \pm x^i (r x^m + s)^{-i}.$

10. $f(x) = \pm r \operatorname{tg}\left(\frac{\pi x^m}{4}\right) \pm s.$

11. $f(x) = \pm r \operatorname{sh}^i x^m \pm s.$

12. $f(x) = \pm r \operatorname{ch}^i x^m \pm s.$

13. $f(x) = \pm r \ln^i(x^m + s).$

14. $f(x) = \pm r \arccos^i x^m \pm s.$

15. $f(x) = \pm r \arcsin^i x^m \pm s.$

16. $f(x) = \pm r \operatorname{arctg}^i x^m \pm s.$

СПИСОК ЛИТЕРАТУРЫ

1. Таненбаум Э. Современные операционные системы. 3-е изд. СПб.: Питер, 2010.

2. Столяров А.В. Введение в операционные системы: конспект лекций. М.: Издательский отдел факультета ВМиК МГУ им. В.М. Ломоносова, 2006.

3. Иртегов Д.В. Введение в операционные системы. 2-е изд., перераб. и доп. СПб.: БХВ. Петербург, 2008.

4. Браун П. Введение в операционную систему UNIX: пер. с англ. М.: Мир, 1987.

5. Колисниченко Д.Н., Аллен Питер В. LINUX: полное руководство. СПб: Наука и техника, 2006.

6. Ктитров С.В., Овсянникова Н.В. Командный язык ОС UNIX: лабораторный практикум. М.: МИФИ, 2007.

7. Уильям Р. Станек Командная строка Microsoft Windows. Справочник администратора: пер. с англ. М.: Издательско-торговый дом «Русская Редакция», 2004.

8. Попов А.В. Командные файлы и сценарии Windows Script Host. СПб.: БХВ-Петербург, 2002.

9. Баррон Д. Введение в языки программирования: пер. с англ. М.: Мир, 1980.

10. Роберт У. Себеста. Основные концепции языков программирования: пер. с англ. 5-е изд. М.: Вильямс, 2001.

11. Меткалф М., Рид. Дж. Описание языка программирования Фортран 90: пер. с англ. М.: Мир, 1995.

12. Бартенев О.В. Современный FORTRAN. М.: Диалог-МИФИ, 1999.

13. Программирование на Фортране 77: пер. с англ. / Дж. Ашкрофт, Р. Элдридж, Р. Полсон, Г. Уилсон. М.: Радио и связь, 1990.

14. Ращиков В.И. Численные методы. Компьютерный практикум: учебно-методическое пособие. М.: НИЯУ МИФИ, 2009.

15. Рошаль А.С. Лабораторный практикум по курсу «Численные методы и программирование». М.: МИФИ, 1992.

16. Самарский А.А, Михайлов А.П. Математическое моделирование: Идеи. Методы. Примеры. 2-е изд., испр. М.: Физматлит, 2002.