**1. ЛАБОРАТОРНАЯ РАБОТА №1.**

**ОПЕРАЦИОННАЯ СИСТЕМА MS-DOS.**

# 1.1 Работа с файлами

Файл – это поименованная область на диске или другом машинном носителе. В файлах могут храниться тексты программ, документы и т.д. Файл состоит из двух частей: имени и расширения. В имени файла может быть от 1 до 8 символов. Расширение начинается с точки, за которой следуют от 1 до 3 символов. Имя и расширение могут состоять из прописных и строчных латинских букв, цифр и символов - \_ $ # & @ ! % ( ) { } ‘ ~ ^. Символ \* обозначает любое число любых символов в имени файла или расширении файла. Символ ? обозначает один произвольный символ или отсутствие символа в имени файла или расширении файла.

Полное имя файла [дисковод:] [путь/] имя файла.

1. Создание текстовых файлов **copy con** имя\_файла

После ввода этой команды нужно поочередно ввести строки файла, а после ввода последней – нажать F6, либо CTRL+Z, и нажать ENTER. На диске появится файл с указанным именем.

1. Удаление файлов **del** [дисковод:] [путь/] имя\_файла

В имени файла можно употреблять символы \* и ?.

1. Переименование файлов **ren** [дисковод:] [путь/] имя\_файла имя\_файла

Первое имя файла в команде задает имя (имена) переименовываемых файлов, второе – новое имя (имена). Можно использовать \* и ? в имени файла.

1. Копирование файлов **copy** имя\_файла1 имя\_файла2 **copy** имя\_файла [имя\_каталога]

В имени файла можно употреблять символы \* и ?.

1. Просмотр содержимого файла **copy** имя\_файла **con**
2. Копирование файла на принтер **copy** имя\_файла **prn**
3. Переименование файлов **rename** имя\_файла1 имя\_файла2

В имени файла можно использовать символы \* и ?. Если режим опущен, то поиск осуществляется во всех каталогах на диске. Если задать режим **/С**, поиск будет вестись только в текущем каталоге. Если же каталог в имени задан, поиск производится только в этом каталоге. Для задания поиска файла в некотором каталоге и всех его подкаталогах используется режим /S. А можно не указывать каталог, но задать режимы /C и /S, тогда поиск файла будет идти в текущем каталоге и всех его подкаталогах.

1. Восстановление удаленных файлов **unerase** [имя\_файла]

В имени файла можно употреблять символы \* и ?.

# 1.2 Работа с каталогами

Каталог – это специальное место на диске, в котором хранятся имена файлов, сведения о размере файлов, времени их последнего обновления, атрибуты (свойства) файлов и т.д. Требования к именам каталогов те же, что и к именам файлов. Как правило, расширение имени для каталогов не используется. Корневой каталог – один главный каталог, который имеется на каждом магнитном диске. Текущий каталог – каталог, с которым в настоящий момент работает пользователь.

Команда смены текущего дисковода

Для смены текущего дисковода надо набрать имя дисковода, который должен стать текущим, и затем двоеточие. Например:

 **A:** - переход на дисковод А

 **B:** - переход на дисковод В

**С:** - переход на дисковод С Изменение текущего каталога **cd** [дисковод:] путь

Просмотр каталога

**dir** [дисковод:] [путь\] [имя\_файла] [/P] [/W]

В имени файла можно употреблять символы \* и ?. Параметр /P задает поэкранный вывод оглавления. Параметр /W задает вывод только информации об именах файлов в каталоге.

1. Создание каталога **md** [дисковод:] путь
2. Уничтожение каталога **rd** [дисковод:] путь
3. Установка спискоа каталогов для поиска выполняемых программ **path** имя\_каталога [; имя\_каталога] … - установка списка каталогов, в

которых производится поиск программ; **path** ; - устанавливает, что поиск программ должен вестись только в те-

кущем каталоге; **path** без параметров – выводит имена каталогов, в которых производит-

ся поиск программ.

1. Наглядный переход из каталога в каталог **ncd** [/r]

Команда позволяет вывести на экран изображение дерева каталогов на диске, указать на нем, в какой каталог надо перейти, перейти в другой каталог, указав только часть его имени. Режим /r приводит к считыванию информации о каталогах с диска.

1. Сортировка элементов каталогов **ds ne** [имя\_каталога] [/s]

Если указан параметр /S, то сортируются также и все подкаталоги.

*Примеры выполнения:*

Создать следующую структуру

1)

 **А1 В1**

 **В2**

 md A1 cd A1 md B1 md B2

2)

 **В1**

 1.bat

**А1**

c.txt

 **В2**

 md A1 cd A1 copy con c.txt md B1 md B2 cd B1

copy con 1.bat

# 1.3 Работа с экраном и принтером

Вывод текстового файла на экран **type** имя\_файла

Очистка экрана монитора

**cls**

Вывод текстового файла на печать **copy** имя\_файла **prn**

Перед выдачей этой команды необходимо, чтобы принтер был включен и находился в состоянии готовности. Печать файлов в фоновом режиме **print** имя\_файла print /t –отменяет фоновую печать

# 1.4 Работа с дисками

Установка и отмена режима проверки при записи на диски **verify on** – включить режим проверки при записи на диски **verify off** – выключить режим проверки при записи на диски

**verify** без параметров – вывести информацию о том, включен или вы-

ключен режим проверки при записи на диски. Форматирование дискет **format** дисковод: [/S]

Если в команде указать параметр /S, то будет подготовлена «системная» дискета, т.е. дискета, с которой можно загрузить операционную систему DOS.

Задание метки на диске **label** дисковод:

Метка диска может быть длиной до 11 символов. Форматирование нестандартных дискет **format** [дисковод:] /**T**: число\_дорожек /**N**: число\_секторов [режимы]

Подготовка компьютера к выключению питания **diskmon/park**

Проверка дисков

**NDD** дисковод: **/Q** – проверка логической структуры диска

**NDD** дисковод: **/C** – проверка логической структуры и наличия физических дефектов на диске.

Оптимизация размещения файлов на диске

Команда перемещает все файлы на диске к началу диска и устраняет фрагментацию файлов. **speedisk** дисковод: [режимы]

Способы оптимизации:

/FF – полная оптимизация с упорядочением размещения файлов;

/FD – полная оптимизация с перемещением каталогов в начало диска;

/U – устранение фрагментации файлов;

/V – выполнить проверку правильности записи на диск;

/B – перезагрузить компьютер после окончания работы программы SpeeDisk.

Проверка надежности жесткого диска

Программа позволяет проверять надежность чтения-записи на жесткий диск путем записи на диск различных специально разработанных данных (образцов) и тестирования правильности записи. Информация при этом не уничтожается.

**Calibrate** дисковод: **/Pattern:** число\_образцов\_тестирования **/Batch [/R:**

имя\_файла\_отчета] **/NoFormat**

# 1.5 Программы общесистемного назначения

Вывод информации о дате и установка даты в компьютере **date**

По этой команде на экран выводится информация о дне недели и дате, и вы можете ввести новое значение даты. Если вы не хотите менять дату, установленную в компьютере, то нажмите Enter.

Вывод информации о времени и установка времени в компьютере **time** [часы:минуты] где часы – число от 0 до 24, а минуты – число от 0 до 59.

Если эта команда выдана без параметров, то DOS выводит текущее время и просит установить новое значение времени в компьютере. Если вы не хотите менять время, нажмите сразу клавишу Enter.

Изменение вида приглашения DOS **promt** [текст]

Если команда выдана без параметров, то устанавливается подсказка, содержащая информацию о текущем дисководе и символ «>», т.е. команда promt без параметров эквивалентна команде $n$g.

В тексте, указываемом в команде promt, можно использовать следующие сочетания символов:

 $p – текущий дисковод и каталог; $h – удаление предыдущего симво-

 $n – текущий дисковод; ла;

 $d – текущая дата; $e – символ с кодом 27 (ESC);

 $t – текущее время; $g – символ «>»;

 $v – версия DOS; $l – символ «<»;

 $\_ – переход на новую строку; $b – символ «|»;

 $s – пробел; $$ - символ «$».

Получение информации о компьютере **sysinfo**

# 1.6 Задание

Создать следующие структуры каталогов и файлов:

Скопировать файлы из каталогов MIKE, LIZA и LINA в каталог TIM. Переместить файлы из каталога LINA в каталог KATY, поменяв первую букву в именах файлов на «as». Переименовать файл m.tey в файл kiss.lps. Уда-

лить созданную структуру файлов и каталогов.

1.

**V1**

**c.txt**

 **JOHKATY MIKE**

 **m.tey TIM lips.dat**

 **MARLIZA**

 **eyes.dat**

 **SANDRA LINA**  **style.txa vit.da**  **life.txp vix.dat**  **viz.dat**

**2.**

**V1**

**c.txt**

**JOHKATY MIKE**

  **em.txa TIM**

 **MAR** mit.txa **LIZA**

dep.txa Sih.txa

 **SANDRA** LINA vet.da  **vix.dat** viz.dat

Скопировать файлы с расширением .txa из всех в каталог JOHN. Затем скопировать все файлы из каталога JOHN, поменяв последнюю букву в расширениях файлов на «t». Переименовать файл Sih.txa в файл kiss.lps. Скопировать файлы vix.dat и viz.dat из каталога LINA в каталог KATY, используя шаблон. Удалить созданную структуру файлов и каталогов.

|  |  |
| --- | --- |
| as.txa ain.txa  | vt.da asd.da efgh.da  |

1. JOHNem.txa

KATY

TIM

MIKE

LIZA

LINA

mit.txa

dep.txa

**V1**

c.txt

MARY

SANDRA

 Скопировать все файлы с расширением .txa в каталог MIKE. Скопировать все файлы из каталога LINA в каталог KATY, используя шаблон, причем поменять в именах файлах вторую букву на «m». Удалить созданную структуру файлов и каталогов.

|  |  |
| --- | --- |
|    |  |
|  |

1. V1 JOHN KATY MIKE sc.ln m.tey TIM

 dark.lin MARY LIZA

 light.ln

beet.vnSANDRA LINA  vit.dat  vix.dat viz.dat

Скопировать все файлы из каталога V1 в каталог KATY, используя шаблон и заменяя последнюю букву в именах файлов на «xy». Скопировать все файлы из каталога LINA в каталог KATY, используя шаблон, причем поменять в именах файлах вторую букву на «j». Удалить созданную структуру файлов и каталогов.

1. **ЛАБОРАТОРНАЯ РАБОТА №2.**

**ПАКЕТНЫЕ, КОМАНДНЫЕ ФАЙЛЫ.**

***2.1 Пакетные, командные файлы.***

Довольно часто в процессе работы с компьютером обнаруживается, что необходимо повторять одни и те же команды DOS для того, чтобы осуществить некоторые периодически выполняемые действия. Операционная система DOS позволяет записать нужную для этого последовательность команд в специальный файл, называемый командным файлом. Командный файл должен иметь расширение .BAT. последовательность команд, записанную в файле, можно выполнить, набрав имя командного файла (расширение при этом можно не указывать).

**Выполнение одного командного файла из другого call** имя\_командного\_файла [параметры]

В версиях DOS до 3.3 эта команда работать не будет вместо нее можно использовать команду: **command** **/c** имя\_командного\_файла [параметры]

Командный префикс **@**

Перед любой командой командного файла можно использовать командный префикс «@». При этом действие команды не изменяется, но команда при исполнении не дублируется на экран. Командный префикс «@» полезно использовать в начале комментариев, которые нежелательно выводить на экран даже в режиме отладки.

***Комментарии* в командном файле rem** любые\_символы

**Сообщения при выполнении командного файла echo** сообщение

Перед этой командой желательно выполнить команду **@echo off**, чтобы сообщение не выводилось дважды на экран. Команды **echo on** и **echo off** управляют режимом вывода исполняемых команд на экран, а команда **echo** без параметров сообщает, что включен или выключен режим echo.

**Приостановка выполнения командного файла pause**

Выполнение командного файла приостанавливается. Если нажать любую алфавитно-цифровую клавишу, Пробел или Enter, выполнение командного файла будет продолжено. Если нажать CTRL + С или CTRL+BREAK, то при нажатии на клавишу Y выполнение командного файла будет окончено, а при нажатии на клавишу N – будет продолжено.

**Переходы в командном файле goto** метка

Если метка в команде **goto** не указана, то процесс пакетной обработки завершается.

**Проверка условий в командном файле IF** условие команда

**команда** – любая допустимая команда. Эта команда выполняется, если

условие истинно, в противном случае команда игнорируется; **условие** – одно из приведенных ниже выражений:

**ERRORLEVEL число** – условие истинно тогда, когда код завершения предыдущей выполненной программы больше заданного числа или равен

ему;

**строка1==строка2** – условие истинно тогда, когда строка1 и строка2

полностью совпадают.

**EXIST имя\_файла** – условие истинно тогда, когда указанный файл существует;

**NOT условие** – истинно тогда, когда указанное условие ложно.

**Создание диалоговых командных файлов BE ASK** «сообщение», список\_символов

**CHOICE /C**: список\_символов сообщение

Обе программы выводят указанное сообщение и ждут, пока пользователь не введет один из указанных в списке символов. Значение переменной ERRORLEVEL устанавливается равным номеру введенного символа в списке.

# 2.2 Конфигурирование системы

**Копирование файлов с жесткого диска**

Для того, чтобы скопировать основные файлы операционной системы, нужно ввести следующие команды: C:

SYS A:

COPY COMMAND.COM A:

**Написание файла CONFIG.SYS**

В корневом каталоге диска, с которого загружается операционная система, может находиться файл CONFIG.SYS, задающий параметры ОС DOS, а также указывающий, какие программы, расширяющие возможности ОС, необходимо загружать в оперативную память. Если файл CONFIG.SYS в корневом каталоге диска, с которого загружается ОС, отсутствует, то параметры ОС будут установлены по умолчанию. Файл CONFIG.SYS должен представлять собой текстовый файл. Каждая строка этого файла имеет вид:

имя\_команды = значение.

Команды файла CONFIG.SYS:

**Break = on** – установить режим проверки нажатия клавиш CTRL+BREAK при операциях ввода-вывода с диском. Это позволяет прерывать выполнение программ, которые иначе бы выполнялись до своего завершения.

**Buffers = число\_буферов** –установка числа буферов для операций ввода-вывода с диском.

**Country = код\_страны, кодовая\_страница, полное\_имя\_файла\_ COUNTRY.SYS** – установка удобного формата выдачи информации о дате и времени.

**Lastdrive = буква** – установка последней буквы, которая может использоваться в качестве имени дисковода в команде SUBST(н-р, LASTDRIVE=Z).

**Files = 20** –установка максимального числа одновременно открытых файлов. При работе с некоторыми базами данных необходимо большее значение параметра FILES: 50 или даже 80.

**Rem комментарий** – задание комментариев в файле CONFIG.SYS.

**Shell = COMMAND.COM /E: число\_байтов /P** – увеличение размера области памяти, в которой хранятся переменные окружения. Число байтов задает размер этой области.

**Device = имя\_файла\_драйвера [параметры]** – установка драйвера устройства. Наиболее полезные драйверы устройств:

ANSI.SYS – расширяет возможности по выводу на экран и позволяет переопределять значения клавиш на клавиатуре.

DRIVER.SYS – позволяет подсоединять к компьютеру дополнительные диски.

MOUSE.SYS – обеспечивает использование мыши в прикладных программах.

SMARTDRV.EXE – обеспечивает кэширование диска.

RAMDRIVE.SYS – позволяет создать «электронный диск» в расширенной или дополнительной памяти.

**Devicehigh = имя\_файла\_драйвера [параметры]** – то же, что и команда Device, но драйвер устанавливается не в обычную память, а в «верхнюю памать».

**Install = имя\_файла\_программы [параметры]** – установка резидентной программы. Этот способ экономит оперативную память, так как при его использовании для запускаемой программы не резервируется место для хранения переменных окружения.

**Numlock = off** –отключает при загрузке фиксацию цифровой клавиатуры.

*Пример файла CONFIG.SYS*

Break = on

Files = 20

Buffers = 16

Shell = command.com /e: 512 /p

Country = 007, 866, c:\ exe\msdos\country.sys

Device = c:\exe\sys\mouse.sys 2

Install = c:\exe\rk.com

**Написание файла AUTOEXEC.BAT**

При первоначальной загрузке операционная система DOS ищет в корневом каталоге того диска, с которого она загружается, файл AUTOEXEC.BAT.

Если этот файл найден, он выполняется. В командный файл AUTOEXEC.BAT удобно записать команды, которые должны выполняться каждый раз при начальной загрузке операционной системы. Эти команды могут осуществить необходимую настройку операционной системы и установить удобное для работы окружение. Кроме того, при наличии файла AUTOEXEC.BAT операционная система не задает в процессе начальной загрузки вопросов о текущей дате и времени.

**Установка списка каталогов, в которых производится** поиск программ

Команду Path обычно включают в файл AUTOEXEC.BAT. В списке каталогов, задаваемом в команде **Path**, следует перечислить через точку с запятой каталоги, в которых находятся исполняемые программы общего назначения. Пример:

**Path** = c:\ exe;c: \ exe \ dos;c: \ exe \ nu;c: \ tc;..;.. \ …

Установка формата приглашения DOS **promt** текст\_приглашения

В тексте, указываемом в команде promt, можно употреблять специальные сочетания символов $p, $n, $d, $t, $h, $e, $g и другие.

**Установка переменных окружения**

 Операционная система отводит специальную область оперативной памяти, называемую «окружением», для хранения значений некоторых переменный, которые используются операционной системой и другими программами. Окружение состоит из строк вида «переменная=значение». Здесь переменная – любая строка, не содержащая символа «=». При этом в записи переменной большие и малые латинские буквы считаются одинаковыми. Значение – любая строка символов. ОС DOS использует три переменные окружения: path, promt и comspec (устанавливается командой Command с параметром /p). Пользователь может задавать переменные окружения с любыми другими именами. Для установки значения переменной окружения задается команда **set**:

**Set** переменная = значение

*Примеры:*

Set 87 = N

Set TEMP = E: \ WIN\ TEMP

*Пример выполнения:*

Командные файлы.

Создать командный файл: Если первый параметр = 1, то просмотреть файлы расширения, которые указаны во втором параметре. Если первый параметр = 2, то переименовываем файл, имя которого указать во втором, расширения в третьем на файл, указанный в четвертом.

C:>copy con start.bat @ echo on if %1==2 ren %2.%3 %4 if %1==1 for %%a in (\*.%2) do type %%a

^z

# 2.3 Задание

1. Создать командный файл. Если первый параметр равен 1, то меняем системное время. Если первый параметр равен 2, то меняем системную дату. Если первый параметр равен 3, то создаем каталог, имя которого указано во втором параметре.
2. Создать командный файл. Если первый параметр равен 1, то вызвать командный файл, который проверяет существование файла, указанного во втором параметре (Если существует, то копирует его в каталог, указанный в третьем параметре.).
3. Создать командный файл. Если первый параметр равен 1, то меняем системное время. Если первый параметр равен 2, то меняем системную дату. Если первый параметр равен 3, то создаем каталог, имя которого указано во втором параметре.
4. Создать командный файл. Если первый параметр равен 1, то вызвать командный файл, который проверяет существование файла, указанного во втором параметре (Если существует, то копирует его в каталог, указанный в третьем параметре.).

**3. ЛАБОРАТОРНАЯ РАБОТА №3.**

**ФАЙЛОВАЯ СИСТЕМА WINDOWS ХР**

# 3.1 Физическая организация файловой системы

Файл, имеющий образ цельного, непрерывающегося набора байт, очень часто разбросан «кусочками» по всему диску, причем это разбиение никак не связано с логической структурой файла, например, его отдельная логическая запись может быть расположена в несмежных секторах диска. Логически объединенные файлы из одного каталога совсем не обязаны соседствовать на диске. Принципы размещения файлов, каталогов и системной информации на реальном устройстве описываются *физической организацией файловой системы*.

# 3.2 Диски, разделы, секторы, кластеры

Основным типом устройства, которое используется в современных вычислительных системах для хранения файлов, являются дисковые накопители. Эти устройства предназначены для считывания записи и данных на жесткие и гибкие магнитные диски. Жесткий диск состоит из одной или нескольких стеклянных или металлических пластин, каждая из которых покрыта с одной или двух сторон магнитным материалом. Таким образом, диск в общем случае состоит из *пакета* пластин.

На каждой стороне каждой пластины размечены тонкие концентрические кольца – *дорожки* (*traks*), на которых хранятся данные. Количество дорожек зависит от типа диска. Нумерация дорожек начинается с 0 от внешнего края к центру диска. Когда диск вращается, элемент, называемый головкой, считывает двоичные данные с магнитной дорожки или записывает их на магнитную дорожку.

Головка может позиционироваться над заданной дорожкой. Головки перемещаются над поверхностью диска дискретными шагами, каждый шаг соответствует сдвигу на одну дорожку. Запись на диск осуществляется благодаря способности головки изменять магнитные свойства дорожки. В некоторых дисках вдоль каждой поверхности перемещается одна головка, а в других – имеется по головке на каждую дорожку. В первом случае для поиска информации головка должна перемещаться по радиусу диска. Обычно все головки закреплены на едином перемещающем механизме и двигаются синхронно.

Совокупность дорожек одного радиуса на всех поверхностях всех пластин пакета называется *цилиндром* (*cylinder*). Каждая дорожка разбивается на фрагменты, называемые *секторами* (*sectors*), или *блоками* (*blocks*), так что все дорожки имеют равное число секторов, в которые можно максимально записать одно и то же число байт. Сектор имеет фиксированный для конкретной системы размер, выражающийся степенью двойки. Чаще всего размер сектора составляет 512 байт. Учитывая, что дорожки разного радиуса имеют одинаковое число секторов, плотность записи становится тем выше, чем ближе дорожка к центру.

Сектор – наименьшая адресуемая единица обмена данными дискового устройства с оперативной памятью. Для того чтобы контроллер мог найти на диске нужный сектор, необходимо задать ему все составляющие адреса сектора: номер цилиндра, номер поверхности и номер сектора.

Операционная система при работе с диском использует, как правило, собственную единицу дискового пространства, называемую *кластером* (*cluster*). При создании файла место на диске ему выделяется кластерами. Например, если файл имеет размер 2560 байт, а размер кластера в ФС определен в 1024 байта, то файлу будет выделено на диске 3 кластера.

Дорожки и секторы создаются в результате выполнения процедуры *физического*, или *низкоуровневого*, *форматирования диска*, предшествующей использованию диска. Для определения границ блоков на диск записывается идентификационная информация.

Разметку диска под конкретный тип ФС выполняет процедуры *высокоуровневого*, или логического, *форматирования*. При высокоуровневом форматировании определяется размер кластера и на диск записывается информация, необходимая для работы ФС, в том числе информация о доступном и неиспользуемом пространстве, о границах областей, отведенных под файлы и каталоги, информация о поврежденных областях. Кроме того, на диск записывается загрузчик ОС – небольшая программа, которая начинает процесс инициализации ОС после включения питания или рестарта компьютера.

Прежде чем форматировать диск под определенную ФС, он может быть разбит на разделы. Раздел – это непрерывная часть физического диска, которую ОС представляет пользователю как *логическое устройство* (логический диск). Логическое устройство функционирует так, как если бы это был отдельный физический диск. Именно с логическими устройствами работает пользователь, обращаясь к ним по символьным именам. Операционные системы разного типа используют единое для всех них представление о разделах, но создают на его основе логические устройства, специфические для каждого типа ОС.

В частном случае, когда все дисковое пространство охватывается одним разделом, логическое устройство представляет физическое устройство в целом. Если диск разбит на несколько разделов, то для каждого из этих разделов может быть создано отдельное логическое устройство. Логическое устройство может быть создано и на базе нескольких разделов, причем эти разделы не обязательно должны принадлежать одному физическому устройству.

Все разделы одного диска имеют одинаковый размер блока, определенный для данного диска в результате низкоуровневого форматирования. ОС может поддерживать разные статусы разделов, особым образом отмечая разделы, которые могут быть использованы для загрузки модулей ОС, и разделы, в которых можно устанавливать только приложения и хранить файлы данных. Один из разделов диска помечается как загружаемый (или активный). Именно из этого раздела считывается загрузчик ОС.

# 3.3 Физическая организация и адресация файла

Важным компонентом физической организации ФС является *физическая* *организация файла*, то есть способ размещения файла на диске. Основными критериями эффективности физической организации файлов являются:

* Скорость доступа к данным;
* Объем адресной информации файла;
* Степень фрагментированности дискового пространства;  Максимально возможный размер файла.

Непрерывное размещение – простейший вариант физической организации, при котором файлу предоставляется последовательность кластеров диска, образующих непрерывный участок дисковой памяти. Основным достоинством этого метода является высокая скорость доступа, так как затраты на поиск и считывание кластеров файла минимальны. Также минимален объем адресной информации – достаточно хранить только номер первого кластера и объем файла. Данная физическая организация максимально возможный размер файла не ограничивает. Однако этот вариант имеет существенные недостатки, которые затрудняют его применимость на практике, не смотря на всю его логическую простоту. Поэтому на практике используются методы, в которых файл размещается в нескольких, в общем случае несмежных областях диска.

Следующий способ физической организации – размещение файла в виде связанного списка кластеров дисковой памяти. При таком способе в начале каждого кластера содержится указатель на следующий кластер. В этом случае адресная информация минимальна: расположение файла может быть задано одним числом – номером первого кластера. В отличие от предыдущего способа каждый кластер может быть присоединен к цепочке кластеров какого-либо файла, следовательно, фрагментация на уровне кластеров отсутствует. Файл может изменять свой размер во время своего существования, наращивая число кластеров. Недостатком является сложность реализации доступа к произвольно заданному месту файла – чтобы прочитать пятый по порядку кластер файла, необходимо последовательно прочитать четыре первых кластера, прослеживая цепочку номеров кластеров.

Популярным способом, применяемым, например, в файловой системе FAT, является использование связанного списка индексов. Этот способ является некоторой модификацией предыдущего. Файлу также выделяется память в виде связанного списка кластеров. Номер первого кластера запоминается в записи каталога, где хранятся характеристики этого файла. Остальная адресная информация отделена от кластеров файла. С каждым кластером диска связывается некоторый элемент – индекс. Индексы располагаются в отдельной области диска – в MS-DOS это таблица FAT (*File Allocation Table*), занимающая один кластер. Когда память свободна, все индексы имеют нулевое значение. Если некоторый кластер N назначен некоторому файлу, то индекс этого кластера становится равным либо номеру M следующего кластера данного файла, либо принимает специальное значение являющееся признаком того, что этот кластер является для файла последним. Индекс же предыдущего кластера файла принимает значение N, указывая на вновь назначенный кластер.

При такой физической организации сохраняются все достоинства предыдущего способа: минимальность адресной информации, отсутствие фрагментации, отсутствие проблем при изменении размера. Для доступа к произвольному кластеру файла не требуется последовательно считывать его кластеры, достаточно прочитать только секторы диска, содержащие таблицу индексов, отсчитать нужное количество кластеров файла по цепочке и определить номер нужного кластера. Данные файла заполняют кластер целиком, а значит, имеют объем, равный степени двойки.

Для того чтобы корректно принимать решение о выделении файлу набора кластеров, ФС должна отслеживать информацию о состоянии всех кластеров диска: свободен или занят. Эта информация может хранится как отдельно от адресной информации файлов, так и вместе с ней.

# 3.4 Физическая организация FAT

Логический раздел, отформатированный под файловую систему FAT, состоит из следующих областей.

* *Загрузочный сектор* содержит программу начальной загрузки ОС. Вид этой программы зависит от типа ОС, которая будет загружаться из этого раздела.
* *Основная копия* FAT содержит информацию о размещении файлов и каталогов на диске.
* *Резервная копия* FAT.
* *Корневой каталог* занимает фиксированную область размером в 32 сектора (16 Кбайт), что позволяет хранить 512 записей о файлах и каталогах, так как каждая запись каталога состоит из 32 байт.
* *Область данных* предназначена для размещения всех файлов и всех каталогов, кроме корневого каталога.

Файловая система FAT поддерживает два типа файлов: обычный файл и каталог. ФС распределяет память только из области данных, причем использует в качестве минимальной единицы дискового пространства кластер.

Таблица FAT (как основная копия, так и резервная) состоит из массива индексных указателей, количество которых равно количеству кластеров области данных. Между кластерами и индексными указателями имеется взаимно однозначное соответствие – нулевой указатель соответствует нулевому кластеру и т.д.

Индексный указатель может принимать следующие значения, характеризующие состояние связанного с ними кластера:

Кластер свободен (не используется); Кластер используется файлом и не является последним кластером файла; в этом случае индексный указатель содержит номер следующего кластера файла; Последний кластер файла; Дефектный кластер; Резервный кластер.

Таблица FAT является общей для всех файлов раздела. В исходном состоянии (после форматирования) все кластеры раздела свободны и все индексные указатели принимают значение «кластер свободен». При размещении файла ОС просматривает FAT, начиная с начала, и ищет первый свободный индексный указатель. После его обнаружения в поле записи каталога «номер первого кластера» фиксируется номер этого указателя. В кластер с этим номером записываются данные файла, он становится первым кластером файла. Если файл умещается в одном кластере, то в указатель, соответствующий данному кластеру, заносится специальное значение «последний кластер файла». Если же размер файла больше одного кластера, то ОС продолжает просмотр FAT и ищет следующий указатель на свободный кластер. После его обнаружения в предыдущий указатель заносится номер этого кластера, который теперь становится следующим кластером файла. Процесс повторяется до тех пор, пока не будут размещены все данные файла. Таким образом, создается связанный список всех кластеров файла.

Размер таблицы FAT и разрядность используемых в ней индексных указателей определяется количеством кластеров в области данных. При форматировании диска под ФС FAT обычно выбирается компромиссное решение и размеры кластеров выбираются из диапазона от 1 до 128 секторов, или от 512 байт до 64 Кбайт.

Существует несколько разновидностей FAT, отличающихся разрядностью индексных указателей, которая используется в качестве условного обозначения: FAT12, FAT16, и FAT32. Максимальный размер раздела FAT32 составляет 2³² кластеров по 32 Кбайт.

Таблица FAT при фиксированной разрядности индексных указателей имеет переменный размер, зависящий от объема области данных диска.

При удалении файла из ФС FAT в первый байт соответствующей записи каталога заносится специальный признак, свидетельствующий о том, что эта запись свободна, а во все индексные указатели файла заносится признак «кластер свободен». Остальные данные в записи каталога, в том числе номер первого кластера файла, остаются нетронутыми, что оставляет шансы для восстановления ошибочно удаленного файла.

Резервная копия FAT всегда синхронизируется с основной копией при любых операциях с файлами, поэтому резервную копию нельзя использовать для отмены ошибочных действий пользователя, выглядевших с точки зрения системы вполне корректными. Резервная копия может быть полезна только в том случае, когда секторы основной памяти оказываются физически поврежденными и не читаются.

Используемый в FAT метод хранения адресной информации о файлах не отличается большой надежностью – при разрыве списка индексных указателей в одном месте, например из-за сбоя в работе программного кода ОС по причине внешних электромагнитных помех, теряется информация обо всех последующих кластерах файла.

Файловые системы FAT12 и FAT16 получили большое распространение благодаря их применению в ОС MS-DOS и Windows 3.x – самых массовых ОС первого десятилетия эры персональных компьютеров. Но из-за постоянно растущих объемов жестких дисков и возрастающих требований к надежности, эти ФС быстро вытесняются как системой FAT32, впервые появившейся в Windows 95 OSR2, так и ФС других типов.

# 3.5 Физическая организация s5 и ufs

Файловые системы s5 (получившие название от System V, родового имени нескольких версий ОС UNIX, разработанных в Bell Labs компании AT&T) и ufs (UNIX File System) используют очень близкую физическую модель. Это не удивительно, так как система ufs является развитием системы s5. ФС ufs расширяет возможности s5 по поддержке больших дисков и файлов, а также повышает ее надежность.

Расположение ФС s5 на диске иллюстрирует рис. 7.15. Раздел диска, где размещается ФС, делится на четыре области:

* *Загруженный блок*;
* *Суперблок* (*superblock*) содержит самую общую информацию о файловой системе: размер ФС, размер области индексных дескрипторов, число индексных дескрипторов, список свободных блоков и список свободных индексных дескрипторов, а также другую административную информацию;
* *Область индексных дескрипторов* (*inode list*), порядок расположения индексных дескрипторов в которой соответствует их номерам;
* *Область данных*, в которой расположены как обычные файлы, так и файлы-каталоги, в том числе и корневой каталог; специальные файлы представлены в ФС только записями в соответствующих каталогах и индексными дескрипторами специального формата, но места в области данных не занимают.

Основной особенностью физической организации файловой системы s5 является отделение имени файла от его характеристик, хранящихся в отдельной структуре, называемой индексным дескриптором (inode). Индексный дескриптор в s5 имеет размер 64 байта и содержит данные о типе файла, адресную информацию, привилегии доступа к файлу и некоторую другую информацию.

Каждый индексный дескриптор имеет номер, который одновременно является уникальным именем файла. Индексные дескрипторы расположены в особой области диска в строгом соответствии со своими номерами. Запись о файле в каталоге состоит всего из двух полей: символьного имени файла и номера индексного дескриптора. Файловая система не накладывает особых ограничений на размер корневого каталога.

Доступ к файлу осуществляется путем последовательного просмотра всей цепочки каталогов, входящих в полное имя файла, и соответствующих им индексных дескрипторов. Поиск завершается после получения всех характеристик из индексного дескриптора заданного файла. В ufs имена файлов могут иметь длину до 255 символов (кодировка ASCII, по одному байту на символ), в то время как в s5 длина имени не может превышать 14 символов.

# 3.6 Физическая организация NTFS

ФС NTFS была разработана в качестве основной ФС ОС Windows NT в начале 90-х годов с учетом опыта разработки файловых систем FAT и HPFS (основная файловая система для OS/2), а также других существовавших в то время файловых систем. Основными отличительными свойствами NTFS являются:

* Поддержка больших файлов и больших дисков объемом до 2³²\*2³² байт;
* Восстанавливаемость после сбоев и отказов программ и аппаратуры управления дисками;
* Высокая скорость операций, в том числе и для больших дисков;
* Низкий уровень фрагментации, в том числе и для больших дисков;
* Гибкая структура, допускающая развитие за счет добавления новых типов записей и атрибутов файлов с сохранением совместимости с предыдущими версиями ФС;
* Устойчивость к отказам дисковых накопителей;
* Поддержка длинных символьных имен;
* Контроль доступа к каталогам и отдельным файлам.

# 3.7 Структура тома NTFS

В отличие от разделов FAT и s5/ufs все пространство тома (логический раздел) NTFS представляет собой либо файл, либо часть файла. Основой структуры тома NTFS является *главная таблица файлов* (*Master File Table,* *MFT*), которая содержит по крайней мере одну запись для каждого файла тома, включая одну запись для самой себя. Каждая запись MFT имеет фиксированную длину, зависящую от объема диска, - 1,2 или 4 Кбайт. Для большинства дисков, используемых сегодня, размер записи MFT равен 2 Кбайт, который мы далее будем считать размером записи по умолчанию.

Все файлы на томе NTFS идентифицируются номером файла в MFT. Этот способ идентификации файла близок к способу, используемому в ФС s5 и ufs, где файл однозначно идентифицируется номером его записи в области индексных дескрипторов.

Весь том NTFS состоит из последовательности кластеров, что отличает эту файловую систему от рассмотренных ранее, где на кластеры делилась только область данных. Порядковый номер кластера в томе NTFS называется *логическим номером кластера* (*Logical Cluster Number, LCN*). Файл NTFS также состоит из последовательности кластеров, при этом порядковый номер кластера внутри файла называется *виртуальным номером кластера* (*Virtual Cluster Number, VCN*).

# 3.8 Задание

1. Разбейте диск на два раздела.
2. Отформатируйте один раздел в NTFS, другой в FAT.
3. Создайте на разделе в NTFS каталог и пропишите доступ к нему только для текущего пользователя.

1. **ЛАБОРАТОРНАЯ РАБОТА №4.**

**ФОРМАЛЬНЫЕ ГРАММАТИКИ.**

***4.1 СТРУКТУРА КОМПИЛЯТОРА. ТИПЫ ТРАНСЛИРУЮЩИХ***

# ПРОГРАММ

Каждый компьютер способен непосредственно выполнять ограниченный набор относительно простых команд. Любые более сложные действия представляются последовательностью таких команд. Для выполнения программы, написанной на языке высокого уровня, ее обычно переводят в последовательность команд машинного кода.

**Исходная программа** (написанная на каком-либо языке программирования) представляет собой последовательность символов, которая вводится в компьютер и преобразуется в форму, пригодную для непосредственного выполнения.

**Компилятор** является программой, которая способна воспринимать строку символов определенного вида (т.е. текст программы на исходном языке) и выдавать другую строку символов (программу на машинном языке). Компиляторам присущ ряд общих черт, что упрощает процесс создания компилирующих программ. В состав любого компилятора входят три основных компонента:

* лексический анализатор (блок сканирования);
* синтаксический анализатор;
* генератор кода машинных команд.

Принцип действия анализаторов можно описать с помощью формальных моделей, в то время как для генератора кода пока не существует общепринятых четких формальных представлений. На фазе **лексического анализа** исходный текст программы в виде цепочки несвязанных друг с другом символов разбивается на единицы, называемые **лексемами**. Такими текстовыми единицами являются ключевые слова, используемые в языке (например, IF,DO и др.), имена переменных, константы и знаки операций (например,\* или +). Далее эти слова рассматриваются как неделимые образования, а не как группы отдельных символов. После разбиения программы на лексемы следует фаза **синтаксического анализа**, называемая **грамматическим разбором**, на которой проверяется правильность следования операторов. Например, для предложения с оператором IF, имеющего вид IF выражение THEN предложение; грамматический разбор состоит в том, чтобы убедиться, что вслед за лексемой IF следует правильное выражение, за этим выражением следует лексема THEN, за которой в свою очередь следует правильное предложение, оканчивающееся знаком ";". Последним выполняется процесс **генерации кода**, который использует результаты синтаксического анализа и создает программу на машинном языке, пригодную к выполнению. Хотя в состав любого компилятора входят все три описанных выше компонента, их взаимодействие может осуществляться разнообразными способами. Рассмотрим наиболее распространенные варианты взаимосвязи между этими компонентами.

Блок сканирования считывает исходную программу и представляет ее в форме файла лексем. Синтаксический анализатор читает этот файл и выдает новое представление программы, например, в постфиксной форме. Наконец, этот файл считывается генератором кода, который создает объектный код программы. Компилятор такого вида называется трехпроходным (рис.1), так как программа считывается трижды (исходный текст программы, файл лексем и файл в постфиксной форме).

**Недостаток:**

Невысокая скорость выполнения, так как в большинстве вычислительных систем операции, связанные с обращением к файлам, осуществляются сравнительно медленно.

**Преимущества:**

Относительная независимость каждой фазы компилирования. Так как связь между обрабатывающими блоками осуществляется только через файлы данных, любой проход может быть реализован независимо от остальных. Это обеспечивает:

1. Возможность автономной разработки различных блоков компилятора разными разработчиками, необходимо только согласовать форматы промежуточных файлов.
2. Гибкость компилятора. Например, для реализации одного и того же языка для различных типов компьютеров, возможно использовать одни и те же блоки сканирования и синтаксического анализа, но написать специальные генераторы кода для каждого типа компьютера. При реализации семейства компиляторов с различных языков для одного типа компьютеров, очевидно, потребуются различные блоки сканирования и синтаксического анализа, но возможно использование общего генератора кода.

СИНТАКСИЧЕСК

И

Й АНАЛИЗАТОР

2)

(

ПРОХОД

БЛОК

СКАНИРОВАНИЯ

(

ПРОХОД

1)

ГЕНЕРАТОР

КОДА

(

ПРОХОД

3)

ФАЙЛ ПОСТФИКСНОЙ ЗАПИСИ

ИСХОДНАЯ ПРО

ГРАММА

ОБЪЕКТНЫЙ КОД

ФАЙЛ ЛЕКСЕМ

Рис.1. Трехпроходный транслятор.

1. Минимальные требования к объему используемой оперативной памяти (модули различных фаз компиляции можно загружать по очереди, выгружая при этом предыдущий).

Для достижения высокой скорости компиляции применяется компилятор с однопроходной структурой (рис.2). На рисунке связи по управлению показаны сплошными линиями, передача данных – пунктиром.

В этом случае синтаксический анализатор выступает в роли основной управляющей программы, вызывая блок сканирования и генератор кода, организованные в виде подпрограмм. Синтаксический анализатор постоянно обращается к блоку сканирования, получая от него лексему за лексемой из просматриваемой программы, до тех пор, пока не построит новый элемент постфиксной записи, после чего он обращается к генератору кода, который создает объектный код для этого фрагмента программы.

Лексический анализатор

Исходная

программа

Синтаксический анализатор

Лексема

Генератор кода

Фрагмент постфиксной записи

Фрагмент

объектного

кода

Рис.2. Однопроходный транслятор.

**Преимущество:**

Максимальная эффективность и скорость выполнения, так как программа просматривается лишь однажды, количество операций обращения к файлам минимально (только чтение из исходного и запись в объектный файлы).

**Недостатки:**

1. Проблемы при организации переходов вперед. Например, во время обработки предложения GOTO метка; могут встретиться трудности, так как "метка" еще не встречалась в тексте программы.
2. Неоптимальность создаваемой объектной программы. Например, если встречается текст:

А = (В + С);

Р = (В + С) + (Е + М); компилятор мог бы построить более эффективный объектный код,

трансформировав программу следующим образом:

А = (В + С);

Р = А + (Е + М);

Однако однопроходный компилятор может утратить часть нужной информации к тому времени, когда в тексте встретится формула (Е + М).

1. Поскольку однопроходный компилятор должен полностью размещаться в памяти, его реализация сопровождается повышенными требованиями к ресурсу памяти, которые не всегда можно удовлетворить, имея систему с ограниченным объемом памяти.

Для повышения эффективности выполнения объектной программы в процесс компилирования может включаться фаза оптимизации. Блок оптимизации легко встраивается в трехпроходный компилятор, где размещается, обычно, между синтаксическим анализатором и генератором кода. На этой фазе постфиксный файл используется в качестве входных данных и создается новый файл, содержащий постфиксную запись эквивалентной программы с улучшенными характеристиками. Поскольку блок оптимизации записывает свои выходные данные в формате постфиксного файла, генератор кода не нуждается в изменении. На практике возможность оптимизации предусматривается по желанию пользователя: если необходимо, чтобы время компилирования было небольшим, блок оптимизации игнорируется; если же требуется получить программу с высокой скоростью выполнения, то после работы синтаксического анализатора вызывается блок оптимизации. Возможны и другие способы структурной организации компилятора. На рис.3 показана структура двухпроходного компилятора, занимающая промежуточное положение между двумя описанными выше вариантами организации. В этом случае синтаксический анализатор, вызывая блок сканирования, получает лексему за лексемой и строит файл постфиксной записи программы. Генератор кода считывает этот файл и создает объектный код программы. Подобной структуре свойственно относительно небольшое время выполнения, так как программа считывается лишь дважды (исходный текст и постфиксная запись). В этом случае легко разрешается проблема с оператором перехода вперед на метку, так как эта метка считывается на фазе первого прохода, перед вызовом генератора кода. В такой компилятор при необходимости легко включить блок оптимизации.

Лексический анализатор

Исходная

программа

Синтаксический анализатор

Лексема

Генератор кода

Файл постфиксной записи

Файл

объектного

кода

Рис.3. Двухпроходный транслятор.

Возможны различные модификации рассмотренных схем. Ясно, что рассмотренные типы компиляторов проявляют свои достоинства в определенных условиях работы и оказываются неэффективными в других случаях.

**Интерпретаторы** реализуют принцип, альтернативный компилированию. Компиляторы и интерпретаторы имеют много общего. Интерпретатор, тоже вначале просматривает исходную программу и выделяет в ней лексемы. Для этого используются блоки сканирования и анализаторы, аналогичные тем, которые входят в состав компилирующих программ. Однако интерпретатор вместо построения объектного кода, способного выполнять те или иные операции, сам производит соответствующие действия.

Лексический анализатор

Исходная

программа

Синтаксиче

ский анализатор

Лексема

Блок исполнения

Файл постфиксной записи

Рис. 4. Структура интерпретатора.

Достоинства интерпретатора: - относительная простота реализации; - удобство отладки программ.

Достоинства компилятора:

* скорость выполнения ;
* независимость выполняемого кода от системы программирования;
* возможность передавать программы заказчикам без исходных текстов.

Большинство современных интерпретаторов выполняют не исходный код, а преобразуют его в промежуточный, который затем интерпретируется. Это позволяет несколько повысить скорость исполнения и избежать передачи заказчикам исходных текстов. Для трансляции классических языков программирования (C, C++, Паскаль, Delphi и др.) обычно используются компиляторы. Как интерпретаторы выполнены большинство реализаций Бейсика и языков управления СУБД. Язык сетевого программирования Java реализован как интерпретируемый специальной виртуальной Java-машиной. Это обеспечивает возможность исполнения промежуточного кода Java на любом типе компьютера, где имеется такая виртуальная Java-машина. Поскольку компилятор преобразует исходную программу в совокупность битов, полученную строку битов можно использовать и на другой машине. Так как подготовка программы для одного типа ЭВМ осуществляется с помощью ЭВМ другого типа, соответствующие компиляторы получили название "**кроссовых**" (т.е. перекрестных).

**Конвертор** – это транслятор с одного языка на другой язык того же уровня. Примером конвертора может быть программа, преобразующая код на языке Паскаль в код на С, или данные об объекте проектирования во внутреннем формате одной САПР в формат другой САПР.

Под **транслятором** понимают любую программу, которая преобразует строку символов (т.е. исходную программу) в другую строку символов (объектную программу). Результатом этого процесса может быть как программа на машинном языке для той или иной машины, так и исходный текст программы на каком-либо другом языке. Термин "**компилятор**" будем использовать, понимая под ним программу, которая осуществляет классическое преобразование исходной программы в программу на машинном языке. Если же будут подразумеваться все разновидности процесса трансляции, будем использовать термин "**транслятор**".

**4.1.1 Контрольные вопросы**

1. Каковы преимущества и недостатки одно-, двух- и трехпроходных компиляторов?
2. Чем отличаются интерпретатор и компилятор?
3. В каких случаях применяются конверторы?
4. Для чего нужны кросс-компиляторы?

# 4.2 Определение языка. Синтаксис и семантика

Прежде чем приступить к созданию компилятора, необходимо иметь четкое и однозначное определение исходного языка. Можно предста-

вить язык состоящим из ряда строк (последовательностей символов). В описании языка определяется, какие строки принадлежат этому языку (синтаксис языка), и значение этих строк (семантика языка). **Синтаксис** – множество правил, которые задают множество формально правильных предложений. Синтаксис для конечного языка (состоящего из конечного числа строк) можно специфицировать, задав список строк.

Строки, принадлежащие языку, обычно называется предложениями языка. В реальных языках – бесконечное число предложений, так что их синтаксис нельзя определит путем перечисления этих предложений.

Синтаксис очень простого языка можно описать на естественном языке, например – "все строки, состоящие только из 1 и 0" тогда 1111 и 1000110 – принадлежат языку, а 1020 – нет. Однако естественным языкам свойственны неоднозначности, поэтому для спецификации синтаксиса более сложных языков применяется более формальный метод определения синтаксиса. Рассмотрим, например, предложение "Кошки спят". Слово "кошки" – подлежащее, а "спят" – сказуемое. Это предложение принадлежит языку, который можно описать, например, при помощи следующих синтаксических правил:

<предложение> ::= <подлежащее><сказуемое>

<подлежащее> ::= кошки | собаки

<сказуемое> ::= едят | спят

Смысл этих трех строк таков: предложение состоит из подлежащего, за которым следует сказуемое. Подлежащее состоит либо из одного слова "кошки", либо из одного слова "собаки". Сказуемое состоит либо из слова "спят", либо из слова "едят". Идея заключается в том, что любое предложение можно получить из начального символа <предложение> последовательным применением правил подстановки.

Формализм, или нотация, использованный при написании этих правил, называется **Бэкуса–Наура формой (БНФ)**. синтаксические единицы <предложение>, <подлежащее> и <сказуемое> называются нетерминальными символами, слова "кошки", "собаки", "спят", "едят" –

терминальными символами, а правила – порождающими правилами. Символы “::=”, “|”, а также скобки нетерминальных символов “<>” – это метасимволы языка БНФ (метасимволы не принадлежат описываемому языку, а относятся к языку описания. **Семантика** задает значение всем предложениям языка. Изменим пример про животных:

<предложение>::=<подлежащее><сказуемое>

<подлежащее> ::= люди | собаки

<сказуемое> ::= говорят | спят

Методы формального описания семантики разработаны недостаточно хорошо и в этих целях часто используется естественный язык. Для рассмотрения задачи спецификации синтаксиса дадим несколько определений.

**Алфавит** – множество символов. Например: Русские буквы, Латинские буквы, цифры.

Если *A* – алфавит, *A\** (замыкание *A*) обозначает множество всех строк (включая пустую строку), составленных из символов, входящих в *A*. *A+* обозначает множество всех строк (исключая пустую строку), состоящих из символов, входящих в *A*. Пустая строка обычно обозначается с помощью *ε* (эпсилон).

Синтаксис языка можно определить, пользуясь системой изображения множеств, например *L*0*n*1*n* |*n* 0. Данный язык включает строки, состоящие из одного или более нулей и того же числа последующих единиц, а также пустую строку.

Более сложный синтаксис языка лучше определять с помощью грамматики. В грамматику входит набор правил для получения предложений языка.

Возьмем синтаксис *L* и воспользуемся следующими правилами: **1.** *J*0*J*1;

 **2.** *J**.*

Чтобы вывести предложение этого языка, поступим следующим образом. Начнем с символа *J* и заменим его на 0*J*1или *ε*. Если *J* опять появился в полученной строке, его опять можно заменить с помощью одного из этих правил, и т.д. Полученная таким образом любая строка, не содержащая *J*, является предложением этого языка. Например, *J*0*J*100*J*11000*J*111000111*.*

Последовательность таких шагов называется выводом строки 000111, а символ  служит для разделения шагов вывода. Все предложения данного языка можно вывести, руководствуясь двумя правилами, любая строка, которую нельзя вывести с их помощью, не является предложением этого языка.

Грамматику часто называют системой перезаписи.

**Грамматика** определяется как четверка *VT*,*VN*,*P*,*J* , где *VT* – алфавит, символы которого называются **терминальными** (терминалами), из них строятся цепочки порождаемые грамматикой. *VN* – алфавит, символы которого называется **нетерминальными** (нетерминалами), используются при построении цепочек; они могут входить в промежуточные цепочки, но не должны входить в результат порождения. *VN* и *VT* не имеют общих символов, т.е. *VT* *VN* , полный алфавит (словарь) грамматики *V* определяется как

*V* *VT* *VN* . *P* – набор **порождающих правил(продукций)**, каждый элемент

которых состоит из пары (*α*, *β*), где *α* находится в *V+*, а *β* в *V\**. *α* – левая часть правила, а *β* – правая, т.е. цепочки, построенные из символов алфавита *V*. Правило записывается *α* → *β*. *J* принадлежит *VN* и называется **начальным символом** (аксиомой). Этот символ – отправная точка в получении любого предложения языка.

Грамматикой, генерирующей язык *L*0*n*1*n* |*n* 0 является

*G*0 0;1,*J*,*P*,*J* , где *P**J*0*J*1,*J*.

Грамматикой, генерирующей язык *L**anbm* |*n*,*m* 0 является

*G*1 *a*;*b*,*J*,*A*,*B*,*P*,*J* , где *P**J**AB*,*A**aA*,*A*,*B**bB*,*B*.

 Начав с символа *J* и применяя последовательно по одному из правил замены нетерминала в выводимой строке можно, например, генерировать строку *aaabb*:

*J*  *AB*  *aAB*  *aaAB*  *aaaAB*  *aaaB*  *aaabB*  *aaabbB*  *aaabb*

Каждая строка, которую можно вывести из начального символа, называется сентенциальной формой. Предложение – сентенциальная форма, содержащая только терминалы. Терминалы обозначаются строчными (маленькими) буквами, а нетерминалы – прописными (большими). Как правило, предложения языка можно генерировать с помощью более чем одной грамматики. Две грамматики, генерирующие один и тот же язык, называют **эквивалентными**. С точки зрения разработчика транслятора одна грамматика может быть гораздо удобней другой.

**4.2.1 Контрольные вопросы**

1. Что определяет синтаксис языка?
2. В чем отличие синтаксиса языка от семантики языка?
3. Что такое грамматика и как она задается?
4. Чем различаются терминальные и нетерминальные символы языка?
5. Чем определяется возможность появления того или иного символа в предложении языка?
6. Что такое «начальный символ» и чем он отличается от других символов языка?

**4.2.2 Задание**

**1.** Задана грамматика с порождающими правилами: *J*(*J*);

*J**JJ* ;

*J*;

где *J* - начальный символ, *ε* - пустая строка. Определить, принадлежат ли языку, генерируемому данной грамматикой следующие строки (аргументировать ответ, в случае принадлежности, доказать с помощью вывода): a) строка ( ( 1 ) ( ) ) ;

1. строка (()()());
2. строка ((*JJ*).

# 4.3 Классификация грамматик

Ограничение типов правил, которые могут появляться в грамматике, позволяет определить ряд специальных классов грамматик. Одна из стандартных классификаций известна как иерархия Хомского. Ее описывают следующим образом:

1. Любая грамматика определенного ранее вида – **грамматика типа 0**.
2. Если для всех правил вида *α* → *β*, |*α*|≤ |*β*|, где |*α*| и |*β*| – длина, т.е.

число символов соответственно *α* и *β*, то грамматика называется **граммати-**

**кой типа 1, или контекстно-зависимой (КЗ).**

1. Если все левые части правил грамматики состоят из одного нетерминального символа, то это **грамматика типа 2, или контекстно - свободная (КС).**
2. Грамматика называется **грамматикой типа 3, или регулярной**, если каждое правило грамматики имеют одну из следующих форм:

1. В случае грамматики, выровненной вправо, в правой части грамматики имеется не более одного нетерминала, который может быть только самым правым символом:

*A* → *a*

*A* → *aB*

2. B случае грамматики, выровненной влево, в правой части грамматики имеется не более одного нетерминала, который может быть только самым левым символом:

*A* → *a*

*A* → *Ba*

Иерархия – включающая, т.е. все грамматики типа 3 являются грамматиками типа 2 и т.д. Иерархия грамматик соответствует иерархии языков. Например, если язык можно генерировать с помощью грамматики типа 2, то его называют языком типа 2. Необходимо помнить, что если для генерации языка можно использовать несколько грамматик, то тип языка соответствует грамматике с наибольшим типом. Иерархия Хомского важна с точки зрения построения трансляторов с различных языков. Чем меньше ограничений в грамматике, тем сложнее ограничения, которые можно наложить на генерируемый язык. Чем более универсален класс используемой грамматики, тем больше свойств языка мы можем описать. Однако чем более универсальна грамматика, тем сложнее должна быть программа, распознающая строки соответствующего языка. Грамматики типа 3 можно использовать для описания некоторых свойств языков программирования или высокоуровневых языков описания аппаратуры. Например, для генерирования идентификаторов по определению многих языков программирования можно воспользоваться следующими правилами:

*I* → *l*

*I* → *l R*

*R* → *l*

*R* → *d*

*R* → *l R*

*R* → *d R*

где буква (*l*) и цифра (*d*) обозначают терминалы (для краткости будем

считать так, потому что перечисление всех возможных букв и цифр потребовало бы написания слишком большого числа правил). Иногда удобно объединять правые части правил, имеющих одинаковые левые части. Вышеприведенную грамматику можно также записать в виде:

*I* → *l* | *l R*

*R* → *l* | *d* | *l R* | *d R*

Вертикальную черту здесь надо понимать как "или". Многие "локальные" средства языков программирования, например константы, ключевые слова языка и строки, представляются с помощью грамматик типа 3. Некоторые очень простые языки описания аппаратуры также можно описать с помощью регулярной грамматики. Однако грамматики типа 3 генерируют только строго ограниченные типы языков – регулярные выражения.

В алфавите *А* к регулярным выражениям относятся следующие:

1. Элемент *А* (или пустая строка).

Если *P* и *Q* – регулярные выражения, то регулярными будут также и выражения

1. *PQ* (*Q* следует за *P*)
2. *P* | *Q* (*P* или *Q*)
3. *P*\* (нуль или более экземпляров *P*) В алфавите {*a, b, c*}

*ab\** | *ca\** – регулярное выражение, которое описывает язык, включаю-

щий следующие строки (помимо прочих):

*abb c caaa ab ca*

**Пример регулярного выражения.** Регулярное выражение, описывающее идентификатор, имеет вид:

*L ( L* | *D )\**, где *L* обозначает букву, *D* – цифру.

У регулярных выражений есть существенные ограничения. Например, регулярное выражение не может задавать шаблоны скобок произвольной длины, и, следовательно, их нельзя генерировать с помощью грамматики типа 3.

**Пример нерегулярного выражения.** Рассмотрим язык, состоящий из строк открывающих и закрывающих скобок (плюс пустая строка), обладающих следующими свойствами:

1.) При чтении слева направо число встреченных закрывающих скобок никогда не превышает число встреченных открывающих скобок.

2.) В каждой строке содержится одинаковое число открывающих и закрывающих скобок.

Например, следующие строки принадлежат языку:

( ) ( ( ) ( ) ( ( ) ) )

( ( ) ( ) ( ( ) ( ( ) ) ) ) ( ( ) )

а приводимые ниже – нет:

( ( ( ) ( ) ) – не соответствует правилу 2.

( ) ) ) ( ( ( ) – не соответствует правилу 1.

Не существует способа представления этого языка с помощью регулярного выражения или его генерирования с помощью грамматики типа 3. Однако этот язык генерируется следующей контекстно-свободной грамматикой:

*J* → *(J)*

*J* → *JJ*

*J* → *ε*

В большинстве языков программирования и языков описания аппаратуры имеются пары скобок, которые необходимо согласовывать, например:

( ) , [ ] , begin end

каждой открывающей скобке должна соответствовать закрывающая.

Так begin () end – правильно begin (end) – неправильно

Контекстно-свободная грамматика позволяет специфицировать подобные ограничения. Как правило, большая часть синтаксиса языков программирования и специализированных языков САПР описывается с помощью КС-грамматики. Однако у большинства языков есть некоторые свойства, которые нельзя выразить с помощью КС-грамматики. Например, присваивание X:=Y может быть допустимым, если объявлено, что X и Y имеют соответствующие типы, или недопустимым при несоответствии типов. Условие такого вида не может быть специфицировано КС - грамматикой, и компиляторы обычно выполняют проверку типа не на фазе формального синтаксического анализа. Однако идею КС-грамматики можно расширить, включив некоторые контекстно-зависимые свойства языков. **Двухуровневые грамматики** (W-грамматики, названной так в честь их изобретателя – А. ван Вейнгаардена). Идея применения двухуровневой грамматики состоит в том, что если правила обычной грамматики обеспечивают конечный способ описания языка, состоящего из бесконечного числа строк, то здесь вторая грамматика применяется для генерирования бесконечного числа правил, которые в свою очередь генерируют предложения языка. Применение второй грамматики позволяет избежать рутинной работы, связанной с написанием бесконечного числа порождающих правил. С помощью двухуровневой грамматики можно генерировать любой язык типа 0. Данная концепция является даже слишком мощной (КЗ свойства большинства машинных языков относительно просты). **Атрибутивные грамматики**. Атрибуты применяются для описания КЗ (точнее К-несвободных) аспектов языка. Рассмотрим пример. Пусть в некотором языке идентификаторы могут иметь тип int или char и являются терминалами грамматики. Описание с помощью КС порождающих правил.

*D* → int *I*

*D* → char *I*

Описав идентификатор, мы хотим запомнить его тип. Этот тип будет свойством описания, видоизменим грамматику, чтобы это указать:

*D* (.ID, MODE) → int (.MODE1) *I*(.ID1)

ID = ID1

MODE = MODE1

аналогично для char. MODE, MODE1, ID, ID1 пишутся в скобках после терминала или нетерминала грамматики и представляют собой его признаки (атрибуты). Преимущество атрибутивных грамматик в том, что они выглядят как КС, но могут специфицировать КЗ конструкции языка. Фактически любой язык типа 0 можно описать с помощью атрибутивной грамматики. Так как языки программирования представляется как КС, к которым добавляются контекстные ограничения, атрибутивные грамматики хорошо подходят для их описания.

**4.3.1 Контрольные вопросы**

1. Сколько типов грамматик насчитывает иерархия Хомского? Назовите эти типы.
2. Каковы преимущества и недостатки регулярных грамматик?
3. Кто впервые описал двухуровневые грамматики и для чего они применяются?
4. Каково назначение атрибутивных грамматик?

**4.3.2 Задание**

* 1. Построить регулярную грамматику для идентификаторов. Идентификатор состоит из букв, цифр и символов "\_" и начинается обязательно с буквы.
	2. Найти регулярную грамматику, генерирующую тот же язык, что и грамматика со следующими порождающими правилами (*J* – начальный символ):

*J* → *A B*

*A* → *X* | *Y*

* + - 1. → *x* | *x X*
			2. → *y* | *y Y*

*B* → *b* | *b B*

* 1. Построить регулярную грамматику, генерирующую выражение. (101)\*

(010)\*

1. **ЛАБОРАТОРНАЯ РАБОТА №5.**

**СОСТАВЛЕНИЕ ТРАНСЛЯТОРА С ЯЗЫКА PASCAL НА ЯЗЫК C.**

* + 1. **Теоретический материал.**

Принципы построения трансляторов описаны в п. 4.1, 4.2.

* + 1. **Задание.**
1. Написать программу на языке Pascal (или C), осуществляющую трансляцию текста программы на языке Pascal в текст программы на языке С. Программа должна осуществлять следующие функции:
	* синтаксический и семантический контроль исходного текста программы (в случае ошибки должно выдаваться сообщение о характере ошибки и транслятор должен завершить работу);
	* преобразование структуры программы (блок описания переменных, начала и конца программы и т.д);
	* преобразование операторов ввода, вывода, присваивания (с возможностью арифметических расчетов), безусловного перехода, развилок и циклов;
	* предусмотреть обработку вложения развилки в цикл.
2. Составить отчет, который должен содержать:
	* Титульный лист, задание;
	* листинг программы – транслятора;
	* контрольный пример с содержанием текста исходной программы, описанием ее работы, содержанием программы - результата работы транслятора, результаты выполнения исходной программы и обработанной транслятором при одних и тех же исходных данных.
	* Контрольный пример обязательно должен содержать: оператор ввода, оператор вывода, простые переменные, арифметические расчеты, оператор безусловного перехода, один цикл с вложенной в него развилкой.
3. Предоставить отчет в бумажном виде, контрольный пример и транслятор в электронном виде для проверки.

**СОДЕРЖАНИЕ**

1. Лабораторная работа №1. Операционная система MS-DOS. ...................... 1
	1. Работа с файлами ....................................................................................... 1
	2. Работа с каталогами .................................................................................. 2
	3. Работа с экраном и принтером ................................................................. 4
	4. Работа с дисками ....................................................................................... 5
	5. Программы общесистемного назначения ............................................... 6
	6. Задание ........................................................................................................ 7
2. Лабораторная работа №2. Пакетные, командные файлы. .......................... 10
	1. Пакетные, командные файлы. ................................................................ 10
	2. Конфигурирование системы ................................................................... 12
	3. Задание ...................................................................................................... 16
3. Лабораторная работа №3. Файловая система Windows ХР ....................... 17
	1. Физическая организация файловой системы ........................................ 17
	2. Диски, разделы, секторы, кластеры ....................................................... 17
	3. Физическая организация и адресация файла ........................................ 20
	4. Физическая организация FAT ................................................................ 22
	5. Физическая организация s5 и ufs ........................................................... 24
	6. Физическая организация NTFS .............................................................. 26
	7. Структура тома NTFS ............................................................................. 26
	8. Задание ...................................................................................................... 27
4. Лабораторная работа №4. Формальные грамматики. ................................. 27
	1. СТРУКТУРА КОМПИЛЯТОРА. ТИПЫ ТРАНСЛИРУЮЩИХ

ПРОГРАММ ....................................................................................................... 27

* + 1. Контрольные вопросы ...................................................................... 35
	1. ОПРЕДЕЛЕНИЕ ЯЗЫКА. СИНТАКСИС И СЕМАНТИКА .............. 35
		1. Контрольные вопросы ...................................................................... 38
		2. Задание ............................................................................................... 39
	2. КЛАССИФИКАЦИЯ ГРАММАТИК .................................................... 39
		1. Контрольные вопросы ...................................................................... 44
		2. Задание ............................................................................................... 44
1. Лабораторная работа №5. Составление транслятора с языка Pascal на

язык C. .................................................................................................................... 45

5.1.1 Теоретический материал. ................................................................. 45 5.1.2 Задание. .............................................................................................. 45

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Учебно-методическое пособие

для студентов специальностей: 230101 «Вычислительные машины, комплексы, системы и сети», 200900 «Сети связи и системы коммутации»

С.Н. Насонов

Редактор Остроухова Г.Н.

Подписано в печать \_\_\_\_\_\_\_\_\_\_\_\_ Формат 60х84 1/16. Бумага офсетная.

Ризография. Печ.л. \_\_\_\_ Тираж \_\_\_\_ экз.

Липецкий филиал Международного института компьютерных технологий.

398600, Липецк, проезд Кувшинова, 5Б.

Типография ЛФ МИКТ, 398600, Липецк, проезд Кувшинова, 5Б.